Fast Stochastic Frank-Wolfe Algorithms for Nonlinear SVMs

Hua Ouyang Alexander Gray
College of Computing
Georgia Institute of Technology
{houyang,agray}@cc.gatech.edu

Abstract

The high computational cost of nonlinear support vector machines has limited their usability for large-scale problems. We propose two novel stochastic algorithms to tackle this problem. These algorithms are based on a simple and classic optimization method: the Frank-Wolfe method, which is known to be fast for problems with a large number of linear constraints. Formulating the nonlinear SVM problem to take advantage of this method, we achieve a provable time complexity of $O(dQ^2/\epsilon^2)$. The proposed algorithms achieve comparable or even better accuracies than the state-of-the-art methods, and are significantly faster.

1 Introduction

Support vector machines (SVM) [1] are one of the most popular nonlinear discriminative learning methods that can achieve good generalization. It can be used for a variety of learning problems, such as classification [1], ranking [2], regression [3], quantile estimation [4]. The focus of this work will be on the computational aspect of SVMs, especially the scalability of **nonlinear** SVM classifiers to large-scale problems. The proposed algorithms can also be readily used for ranking, regression and other regularized risk minimization problems.

Training SVM classifiers can be formulated as a minimization

$$\min_{\mathbf{w}\in\mathbb{H}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{w};(\mathbf{x}_i,y_i)\right) \tag{1.1}$$

where L is the loss function and $\mathbb{H}$ is a vector space. The parameter λ is used as a trade-off between the squared 2-norm regularizer and the empirical risk. The training set is denoted as $\{(\mathbf{x}_i,y_i)\}_{i=1}^N$ with samples $\mathbf{x}_i\in\mathbb{R}^d$ and corresponding labels $y_i\in\{-1,1\}$. A popular loss function often chosen is the hinge loss: $L\left(\mathbf{w},b;(\mathbf{x}_i,y_i)\right) = \max\left\{0,1-y_i\left(\langle\mathbf{w},\phi(\mathbf{x}_i)\rangle+b\right)\right\}$, which is a convex surrogate of the $0-1$ loss. Note that there are many alternative convex loss functions available for classification tasks, such as the squared hinge loss, log-likelihood loss, Huber loss and its variants. The *squared hinge loss* is adopted in this work.

The prediction function can be expressed as: $\mathrm{sign}\left(h(\mathbf{x})\right) = \mathrm{sign}\left(\langle\mathbf{w},\phi(\mathbf{x})\rangle+b\right)$. Here $\phi(\cdot)$ is a mapping from $\mathbb{R}^d$ to a feature space $\mathbb{H}$ which is often chosen as a kernel-induced Hilbert space equipped with an inner product $\langle\cdot,\cdot\rangle$. The kernel function is $k(\mathbf{x}_i,\mathbf{x}_j) = \langle\phi\left(\mathbf{x}_i\right),\phi\left(\mathbf{x}_j\right)\rangle$. The $N\times N$ kernel matrix is denoted as $K=\{K_{ij}\}$, where $K_{ij}=k(\mathbf{x}_i,\mathbf{x}_j)$. If $\phi\left(\mathbf{x}_i\right)=\mathbf{x}_i$, $h(\mathbf{x})$ is a linear hyperplane in $\mathbb{R}^d$ and it is called a linear SVM, otherwise it is a nonlinear SVM.

Slack variables $\boldsymbol{\xi}$ can be introduced to handle cases where the two classes are not separable. The soft-margin SVM with p^{th} polynomial hinge loss [5] can then be expressed as

(sP)
$$\begin{aligned}\min_{\mathbf{w}\in\mathbb{H},b\in\mathbb{R},\boldsymbol{\xi}\in\mathbb{R}^N}\quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i^p \\ \text{s.t.}\quad & y_i\left(\langle\mathbf{w},\phi(\mathbf{x}_i)\rangle+b\right)\geq 1-\xi_i,\ \xi_i\geq 0,\ \forall i.\end{aligned}$$

(sP) is called the *primal problem* of SVM. When $p=1$, it is often called L1-SVM. When $p=2$ it is called L2-SVM [1] which corresponds to squared hinge loss.

By introducing Lagrangian multipliers $\boldsymbol{\alpha}$ and using Karush-Kuhn-Tucker (KKT) optimality conditions, the *dual problem* of SVM with (squared) hinge loss can be expressed as:

(sD)
$$\begin{aligned}\max_{\boldsymbol{\alpha}\in\mathbb{R}^N}\quad & -\frac{1}{2}\boldsymbol{\alpha}^T\left(K\odot\mathbf{y}\mathbf{y}^T\right)\boldsymbol{\alpha}+\boldsymbol{\alpha}^T\mathbb{1} \\ \text{s.t.}\quad & 0\leq\alpha_i\leq U,\ i=1,\ldots,N,\ \mathbf{y}^T\boldsymbol{\alpha}=0,\end{aligned}$$

where the Hadamard product $K\odot\mathbf{y}\mathbf{y}^T=\{y_iy_jK_{ij}\}$, and $\mathbb{1}$ is a vector of all ones. For L1-SVM, $U=C$, and $U=\infty$ for L2-SVM.

Historically, a majority of previous SVM solvers deal with (sD), due to the fact that the nonlinear mapping $\phi(\cdot)$ in (sP) is hard to handle and its constraints are complex, while (sD) is a standard quadratic programming (QP) problem with box and equality constraints.

[1] Please notice the difference between L2-SVM (squared hinge loss) and SVM with 2-norm regularization term (any loss).

Nonetheless solving (sD) using general QP solvers has not been widely adopted for large-scale problems, since no matter how easy or difficult the problem is, the kernel matrix K is always dense. For instance, the interior point method needs $O(dN^2)$ memory to store K and $O(N^3)$ time for matrix inversions, and both are prohibitive even for small problems like $N = 10^3$. The low-rank approximation method [6] has been proposed to tackle this problem.

Another vein of solving SVMs are chunking and decomposition methods [1][7][8]. The main idea is to optimize over a small working set B where $|B| \ll N$ and update this set after each iteration, while the α_is of the rest of the samples remain unchanged. How to choose B is crucial for this class of methods. Sequential minimal optimization (SMO) [8] takes $|B| = 2$. The two samples are chosen as the pair that violates the complementarity conditions the most. Although there is no rate of convergence analysis for SMO, yet empirically its worst-case time complexity is at least $O(N^{2.3})$ [8]. Shrinking and caching techniques have been proposed for further speed up [9], and second order information has been used for working set selection [10].

Linear SVM solvers are adopted for large-scale problems, especially text classifications [11][12][13][14]. Coordinate descent, cutting-plane and bundle methods are utilized to solve the unconstrained problem 1.1 directly. Since there are only d variables instead of N in nonlinear SVMs, state-of-the-art linear SVM solvers can achieve an $O(dN)$ time complexity or even better. However, generally speaking, the prediction error of a linear classifier could be larger than a nonlinear one, as shown in our experiments.

Stochastic programming techniques have shown to be very efficient for large-scale learning problems, both theoretically and empirically [16][17][18][19]. For large-scale problems, it is often desired to make a trade-off between computational complexity and the precision of underlying optimization algorithms [15]. Stochastic methods, despite of its slow rate of convergence compared with batch methods, can make each iteration very cheap. Hence if only approximate solutions are desired, as in large-scale learning problems, stochastic methods can be much faster than batch methods. This is an important motivation of our work. Among many previous work, *stochastic approximation* [20] is often used for online/stochastic convex optimizations. The celebrated stochastic gradient descent (SGD) $\mathbf{w}_{t+1} = \Pi(\mathbf{w}_t - \eta \nabla f(\mathbf{w}, (\mathbf{x}_t, y_t))|_{\mathbf{w}=\mathbf{w}_t})$ is a most popular example [21], where η is a learning rate, $\nabla f(\mathbf{w}, (\mathbf{x}_t, y_t))$ is a noisy approximation of the true gradient. In many cases it only involves a single training sample. $\Pi()$ is a projection on the feasible set of $\mathbf{w}$. The $O(d)$ memory requirement of SGD makes it perfect for large-scale online learning scenarios. Second order information is also used in more recent work [22][23]. Note that most of these work solve linear SVMs via the primal problem (1.1) since it is an unconstrained problem. Thus there is no need to do the extra projection $\Pi()$ which could be even more expensive.

Inspired by the core vector machine [24] and its relations with sparse greedy methods [25], in this paper we propose a family of stochastic nonlinear SVM solvers which solve L2-SVMs in the dual problems. These new algorithms are based on a simple deterministic constrained optimization method: the *Frank-Wolfe method (FW)* [26], hence they are named *stochastic Frank-Wolfe algorithms (SFW)*. Unlike stochastic approximation algorithms such as SGD and its variants, SFW has the flavor of *sample average approximation* [28] which is another vein of stochastic programming.

With a slight modification of the primal problem (sP), an alternative dual problem can be formulated as a simplex constrained QP problem which can be solved efficiently by SFW with time complexity $O\left(d\frac{Q^2}{\epsilon^2}\right)$, where $\epsilon = f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha})$, and Q is a constant explained in Theorem 4.1. A modified algorithm using SFW with "away steps" [27] is used to improve the performance of SFW.

2 The Frank-Wolfe Method

The Frank-Wolfe method (FW), also known as the conditional gradient method [29], is a simple and classic first order feasible direction method. It was among the earliest methods that solve convex problems with a continuously differentiable objective function and linear constraints:

$$\min f(\mathbf{x}) \text{ s.t. } \mathbf{x} \in \mathcal{X}. \tag{2.2}$$

Although in its original form, FW has only a sublinear convergence [26], a modified version of it can achieve linear convergence [27]. Besides, the computations in each iteration can be made cheap provided that the constraints are linear.

FW generates a sequence of feasible vectors $\{\mathbf{x}^{(k)}\}$ using line search: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda^k \mathbf{d}^{(k)}$, where stepsize $\lambda^{(k)} \in [0, 1]$, $\mathbf{d}^{(k)} = \bar{\mathbf{x}} - \mathbf{x}^{(k)}$ is a feasible descent direction satisfying $\bar{\mathbf{x}} \in \mathcal{X}$ and $(\mathbf{d}^{(k)})^T \nabla f(\mathbf{x}^{(k)}) < 0$:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda^{(k)} \left(\bar{\mathbf{x}} - \mathbf{x}^{(k)}\right). \tag{2.3}$$

To search for a best feasible direction, i.e., the best $\bar{\mathbf{x}}$, FW uses the first order Taylor expansion of $f(\mathbf{x})$ and solves the optimization problem:

$$\bar{\mathbf{x}} = \arg\min_{\mathbf{x} \in \mathcal{X}} \left(\mathbf{x} - \mathbf{x}^{(k)}\right)^T \nabla f\left(\mathbf{x}^{(k)}\right). \tag{2.4}$$

The direction search problem (2.4) needs to be at least no harder than (2.2) such that FW can be of practical use. This is the case for continuously differentiable $f(\mathbf{x})$ and linear constraints, since (2.4) is then a linear program that can be solved via the simplex method.

A special case is when the feasible set is a unit simplex: $\triangle := \left\{\mathbf{x} \mid \sum_{i=1}^N x_i = 1, x_i \geq 0, \forall i\right\}$, and (2.4) is simplified as:

$$\min_{\mathbf{x}\in\triangle} \sum_{i=1}^N \frac{\partial f\left(\mathbf{x}^{(k)}\right)}{\partial x_i}\left(x_i - x_i^{(k)}\right). \tag{2.5}$$

In this case the solution $\bar{\mathbf{x}}$ of (2.5) has all coordinates equal to 0 except for a single coordinate indexed by p^*, corresponding to the smallest partial derivative, for which $x_{p^*} = 1$. We denote this solution by $\mathbf{e}(p^*)$.

The sublinear convergence of FW can be stated by the following theorem.

THEOREM 2.1. ([26]) *Let $\mathbf{x}^*$ be optimal for (2.2) and $\{\mathbf{x}^{(k)}\}$ be a sequence generated by FW, then there exists an index K, constants B and ζ such that*

$$f(\mathbf{x}^{(k)}) - f(\mathbf{x}^*) \leq \frac{B}{k+\zeta},\ \forall k \geq K. \tag{2.6}$$

3 Stochastic Frank-Wolfe for SVM

3.1 Alternative Formulation of SVM It might be possible to solve the dual (sD) using FW with the simplex method. However, an alternative formulation of SVM's primal problem can make the dual problem more suitable for FW. The motivation is that, if we can obtain a dual problem that has only simplex constraints, then instead of solving (2.4), we can look for the explicit solution of the simpler problem (2.5).

The alternative formulation, which was proposed and successfully utilized by core vector machines [24], is a variant of L2-SVM. It can be expressed as:

(aP)
$$\begin{aligned}\min_{\mathbf{w}\in\mathbb{H},b\in\mathbb{R},\rho\in\mathbb{R},\boldsymbol{\xi}\in\mathbb{R}^N}\ & \frac{1}{2}\left(\|\mathbf{w}\|^2 + b^2\right) + C\sum_{i=1}^N \xi_i^2 - \nu\rho \\ \text{s.t.}\ & y_i\left(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b\right) \geq \rho - \xi_i,\ \forall i.\end{aligned}$$

Comparing with (sP), (aP) adds a regularization term b^2 and a margin factor $\nu\rho$, where $\nu > 0$ can be removed when we see the dual problem later. By introducing b^2 we can remove the equality constraint $\mathbf{y}^T\boldsymbol{\alpha} = 0$, while by $\nu\rho$ and the usage of the squared hinge loss, we can obtain the simplex constraint.

The dual problem of (aP) can be expressed as:

(aD)
$$\begin{aligned}\max_{\boldsymbol{\alpha}\in\mathbb{R}^N}\ & -\boldsymbol{\alpha}^T\left(K \odot \mathbf{y}\mathbf{y}^T + \mathbf{y}\mathbf{y}^T + I/2C\right)\boldsymbol{\alpha} \\ \text{s.t.}\ & \boldsymbol{\alpha}^T\mathbb{1} = \nu,\ \alpha_i \geq 0,\ \forall i = 1,\ldots,N,\end{aligned}$$

where I is an identity matrix. We denote

$$A := K \odot \mathbf{y}\mathbf{y}^T + \mathbf{y}\mathbf{y}^T + I/2C, \tag{3.7}$$

with component $A_{ij} = y_i y_j\left(K_{ij} + 1\right) + \delta_{ij}/2C$ where $\delta_{pm} = 1$ if $p = m$ and $\delta_{pm} = 0$ otherwise. The objective function of (aD) is homogeneous in $\boldsymbol{\alpha}$, hence we can simply let $\nu = 1$. In this setting, when taking $\xi = 0$, the constraints of (aP) states that the two classes are separated by the margin $2\rho/\|\mathbf{w}\|$. This margin can also be calculated from the KKT complementary slackness conditions.

Since A is positive definite, (aD) is a concave QP problem with unit simplex constraints. This make it in a good situation that FW method can be readily utilized.

Note that in (aP) we do not impose the nonnegativity of ρ explicitly due to the following fact:

PROPOSITION 3.1. *Imposing $\rho \geq 0$ in (aP) leads to a dual problem that has the same optimal solution as (aD).*

Proof. With $\rho \geq 0$, the new dual problem is $\max_{\boldsymbol{\alpha}\in\mathbb{R}^N} -\boldsymbol{\alpha}^T A\boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T\mathbb{1} \geq \nu$, $\alpha_i \geq 0$, $i = 1,\ldots,N$. Suppose that it has an optimal solution $\boldsymbol{\alpha}^*$ such that $\boldsymbol{\alpha}^{*T}\mathbb{1} > \nu$, then $0 < \nu/(\boldsymbol{\alpha}^{*T}\mathbb{1}) < 1$. Define $\tilde{\boldsymbol{\alpha}} = \nu\boldsymbol{\alpha}^*/(\boldsymbol{\alpha}^{*T}\mathbb{1})$. It follows that $-\tilde{\boldsymbol{\alpha}}^T A\tilde{\boldsymbol{\alpha}} \leq -\boldsymbol{\alpha}^{*T}A\boldsymbol{\alpha}^*$ and $-\tilde{\boldsymbol{\alpha}}^T A\tilde{\boldsymbol{\alpha}} = -\left(\nu/(\boldsymbol{\alpha}^{*T}\mathbb{1})\right)^2\boldsymbol{\alpha}^{*T}A\boldsymbol{\alpha}^* \geq -\boldsymbol{\alpha}^{*T}A\boldsymbol{\alpha}^*$. Hence $\tilde{\boldsymbol{\alpha}} = \boldsymbol{\alpha}^*$ and $\boldsymbol{\alpha}^{*T}\mathbb{1} = \nu$.

It should be mentioned that, compared with conventional formulation of L2-SVM, the regularized bias term in the alternative formulation does not affect its performance, as will be observed in our experiments. This has also been shown empirically by many other previous work.

3.2 SFW Algorithm The stochastic Frank-Wolfe algorithm (SFW) solves the dual problem (aD) stochastically. Within the k^{th} iteration, SFW solves the following approximation problem

(aD-k)
$$\begin{aligned}\max_{\boldsymbol{\alpha}\in\mathbb{R}^k} & f(\boldsymbol{\alpha}) = -\boldsymbol{\alpha}^T A\boldsymbol{\alpha} \\ \text{s.t.}\ & \boldsymbol{\alpha}^T\mathbb{1} = 1,\ \alpha_i \geq 0,\ \forall i = 1,\ldots,k,\end{aligned}$$

for a few FW steps. Then the sample size k is increased and a number of additional steps are performed for the updated approximation problem. k is then increased again, a few FW steps are performed. For each iteration, one does not need to solve the approximation problem with a high precision.

In online learning settings, k can be increased as long as new samples are available. In stochastic batch learning settings, when all samples are in a working set and no new sample is available, SFW will proceed along

a direction guided by an existed sample in the working set.

Algorithm 1 shows the proposed Stochastic Frank-Wolfe Algorithm (SFW). The time-complexities of the most time-consuming steps are given.

Algorithm 1 SFW

1: $p^*_{(0)} = \text{rand}()$, $q^{(1)} = A_{p^*_{(0)}p^*_{(0)}}$, $\lambda^*_{(0)} = 1$
2: $\boldsymbol{\alpha}^{(1)} = [0, \ldots, 0, 1, 0, \ldots, 0]^T = \mathbf{e}(p^*_{(0)})$, update $g^{(1)}$
3: **for** $k = 1, \ldots$ **do**
4: Sample $\{\mathbf{x}_k, y_k\}$ provided. Calculate $g^{(k)}_k$. // $O(dk)$
5: **if** $g^{(k)}_k \geq \max_{p \in S^{(k-1)}} g^{(k)}_p$ **then**
6: $S^{(k)} \leftarrow S^{(k-1)} \cup \{k\}$
7: **else**
8: $S^{(k)} \leftarrow S^{(k-1)}$
9: **end if**
10: TOWARD // $O(dk)$ or $O(k)$
11: **end for**
12: $b = \mathbf{y}^T \boldsymbol{\alpha}$

Algorithm 2 shows the FW step of Algorithm 1 (line 10). It is named TOWARD since in the next subsection we will introduce an AWAY step.

Algorithm 2 TOWARD

1: $p^*_{(k)} \leftarrow \text{argmax}_{p \in S^{(k)}} g^{(k)}_p$
2: $\lambda^*_{(k)} \leftarrow \text{Clamp}\left\{1 - \frac{g_{p^*_{(k)}}/2 + A_{p^*_{(k)}p^*_{(k)}}}{q^{(k)} + g_{p^*_{(k)}} + A_{p^*_{(k)}p^*_{(k)}}}, [0, 1]\right\}$
3: $q^{(k+1)} \leftarrow \left(1 - \lambda^*_{(k)}\right)^2 q^{(k)} - \lambda^*_{(k)}\left(1 - \lambda^*_{(k)}\right) g_{p^*_{(k)}} + \left(\lambda^*_{(k)}\right)^2 A_{p^*_{(k)}p^*_{(k)}}$
4: **for** $p = 1, \ldots, |S^{(k)}|$ **do**
5: $g^{(k+1)}_p \leftarrow \left(1 - \lambda^*_{(k)}\right) g^{(k)}_p - 2\lambda^*_{(k)} A_{pp^*_{(k)}}$
6: **end for**
7: $\boldsymbol{\alpha}^{(k+1)} \leftarrow \left(1 - \lambda^*_{(k)}\right) \boldsymbol{\alpha}^{(k)} + \lambda^*_{(k)} \mathbf{e}\left(p^*_{(k)}\right)$

We denote the objective function of (aD-k) as $f(\boldsymbol{\alpha}) := -\boldsymbol{\alpha}^T A \boldsymbol{\alpha}$, and the corresponding gradient as $\mathbf{g}(\boldsymbol{\alpha}) = -2A\boldsymbol{\alpha}$. For $p = 1, \ldots, k$, the p^{th} component of $\mathbf{g}(\boldsymbol{\alpha})$ is calculated as:

$$\begin{aligned} g_p &= -2y_p \sum_{m=1}^{k} y_m \alpha_m \left(K_{pm} + 1 + y_p y_m \delta_{pm}/2C\right) \\ &= -2y_p \sum_{m=1}^{k} y_m \alpha_m \left(K_{pm} + 1\right) - \alpha_p/C. \end{aligned} \tag{3.8}$$

The algorithm maintains a working set $S^{(k)}$. For initialization, a random sample indexed by $p^*_{(0)}$ is chosen and we set $S^{(0)} = \{p^*_{(0)}\}$, $\boldsymbol{\alpha}^{(1)} = \mathbf{e}(p^*_{(0)})$. The gradient is then initialized using (3.8).

In the following k^{th} iteration, if the new training sample $\{\mathbf{x}_k, y_k\}$ can provide a better feasible direction than **any** old samples in $S^{(k)}$, this new sample is selected as $p^*_{(k)}$ and is included in the new working set $S^{(k+1)}$. Otherwise, the best old sample within $S^{(k)}$ will be selected as $p^*_{(k)}$ for updating $\boldsymbol{\alpha}$ and $\mathbf{g}(\boldsymbol{\alpha})$, and $S^{(k+1)}$ will remains the same as $S^{(k)}$. Therefore the number of indices in the working set $|S^{(k)}| \leq k$, and $\boldsymbol{\alpha}^{(k)}$ has at most k nonzero items.

The search direction $\mathbf{d}^{(k)} = \mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)}$ starts from the current solution $\boldsymbol{\alpha}^{(k)}$ and points to one of the vertices of the unit simplex. Once this optimal vertex is determined, we can use the limited minimization rule to determine the stepsize $\lambda^*_{(k)}$:

$$\begin{aligned} \lambda^*_{(k)} &= \arg\max_{\lambda} f\left(\boldsymbol{\alpha}^{(k)} + \lambda(\mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)})\right) \\ &= \arg\min_{\lambda} \lambda^2 \left(\mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)}\right)^T A \left(\mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)}\right) + \\ & 2\lambda \left(\mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)}\right)^T A\boldsymbol{\alpha}^{(k)} + \boldsymbol{\alpha}^{(k)T} A \boldsymbol{\alpha}^{(k)} \\ &= 1 + \frac{\mathbf{e}^T(p^*_{(k)}) A\boldsymbol{\alpha}^{(k)} - A_{p^*_{(k)}p^*_{(k)}}}{\boldsymbol{\alpha}^{(k)T} A \boldsymbol{\alpha}^{(k)} - 2\mathbf{e}^T(p^*_{(k)}) A\boldsymbol{\alpha}^{(k)} + A_{p^*_{(k)}p^*_{(k)}}} \\ &= 1 + \frac{-g_{p^*_{(k)}}/2 - A_{p^*_{(k)}p^*_{(k)}}}{q^{(k)} + g_{p^*_{(k)}} + A_{p^*_{(k)}p^*_{(k)}}}, \end{aligned} \tag{3.9}$$

where we denote $q^{(k)} := \boldsymbol{\alpha}^{(k)T} A \boldsymbol{\alpha}^{(k)}$. Note that $q^{(k+1)}$ can be calculated by updating from $q^{(k)}$ rather than starting from scratch:

$$\begin{aligned} q^{(k+1)} &= \left[\boldsymbol{\alpha}^{(k)} + \lambda^*_{(k)}(\mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)})\right]^T A \\ &\quad \left[\boldsymbol{\alpha}^{(k)} + \lambda^*_{(k)}(\mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)})\right] \\ &= (1 - \lambda^*_{(k)})^2 q^{(k)} + 2\lambda^*_{(k)}(1 - \lambda^*_{(k)}) \mathbf{e}^T(p^*_{(k)}) A\boldsymbol{\alpha}^{(k)} + \\ &\quad (\lambda^*_{(k)})^2 A_{p^*_{(k)}p^*_{(k)}}. \end{aligned} \tag{3.10}$$

In order to make sure that the new feasible solution remains in the feasible set, we need to clamp $\lambda^*_{(k)}$ in the interval $[0, 1]$: if $\lambda^*_{(k)} < 0 \rightarrow \lambda^*_{(k)} = 0$, if $\lambda^*_{(k)} > 1 \rightarrow \lambda^*_{(k)} = 1$.

Calculations of $g^{(k+1)}_p$ can also be done by updating from $g^{(k)}_p$ in the same manner as (3.10), as shown in line 5 of Algorithm 2. A practical method for further accelerating line 7 is that, instead of scaling vector $\boldsymbol{\alpha}$ for every iteration, which take $O(k)$ time, we can simply maintain a scaler $c = \prod_k (1 - \lambda^*_{(k)})$, and only update the $p^*_{(k)}$-th component of $\boldsymbol{\alpha}^{(k+1)}$ by $\lambda^*_{(k)} \mathbf{e}(p^*_{(k)})/(1 - \lambda^*_{(k)})$.

This can reduce its time complexity to $O(1)$. Scaler c is multiplied back when the true values of $\boldsymbol{\alpha}$ are needed.

Calculation of the bias term $b = \mathbf{y}^T\boldsymbol{\alpha}$ is directly obtained by taking partial derivative of the Lagrangian of (aP) wrt b to 0.

3.3 SFW Algorithm with Away Steps

In addition to "toward steps" as in Eq.(2.3), "away steps"

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda^{(k)}\left(\mathbf{x}^{(k)} - \bar{\mathbf{x}}\right) \tag{3.11}$$

can also be considered. We denote search directions for toward steps as $\mathbf{d}_t^{(k)} := \bar{\mathbf{x}} - \mathbf{x}^{(k)}$ and $\mathbf{d}_a^{(k)} := \mathbf{x}^{(k)} - \bar{\mathbf{x}}$ for away steps.

Introducing away steps is a method that can boost FW to linear convergence [27]. We can illustrate it using a simplex-constrained example shown in Fig.1, where $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2$ are vertices of the simplex and the initial solution is x_0. If we only use toward steps, the convergence is pretty slow: $\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3 \to \cdots$. However, with away steps, the solution converges in merely 3 steps: $\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}^*$, where the last one is an away step: $\mathbf{d}_a = \mathbf{x}_2 - \mathbf{e}_0$.

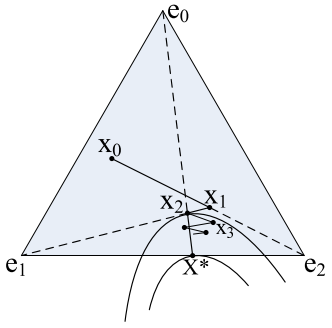

Figure 1: FW with Away Steps.

Applying this method in our dual SVM problem (aD-k), we can use the following to decide whether to make a toward or away step in the k^{th} iteration:

Algorithm 3 DECIDESTEP

1: **if** $\mathbf{d}_t^{(k)T}\nabla f(\boldsymbol{\alpha}^{(k)}) \geq \mathbf{d}_a^{(k)T}\nabla f(\boldsymbol{\alpha}^{(k)})$ **then**
2: TOWARD
3: **else**
4: AWAY
5: **end if**

The AWAY step is given in Algorithm 4. The stepsize is obtained in a similar way as (3.9). In order to keep the new solution feasible, we need $(1+\lambda)\alpha_{p^*_{(k)}} - \lambda \geq 0$, that is $0 \leq \lambda \leq \alpha_{p^*_{(k)}}/(1-\alpha_{p^*_{(k)}})$.

Algorithm 4 AWAY

1: $p^*_{(k)} \leftarrow \operatorname{argmin}_{p\in S^{(k)}} g_p^{(k)}$
2: $\lambda^*_{(k)} \leftarrow \text{Clamp}\left\{\frac{g_{p^*_{(k)}}/2 + A_{p^*_{(k)}p^*_{(k)}}}{q^{(k)} + g_{p^*_{(k)}} + A_{p^*_{(k)}p^*_{(k)}}} - 1, \left[0, \frac{\alpha_{p^*_{(k)}}}{1-\alpha_{p^*_{(k)}}}\right]\right\}$
3: $q^{(k+1)} \leftarrow \left(1+\lambda^*_{(k)}\right)^2 q^{(k)} + \lambda^*_{(k)}\left(1+\lambda^*_{(k)}\right) g_{p^*_{(k)}} + \left(\lambda^*_{(k)}\right)^2 A_{p^*_{(k)}p^*_{(k)}}$
4: **for** $p = 1, \ldots, |S^{(k)}|$ **do**
5: $g_p^{(k+1)} \leftarrow \left(1+\lambda^*_{(k)}\right) g_p^{(k)} + 2\lambda^*_{(k)} A_{pp^*_{(k)}}$
6: **end for**
7: $\boldsymbol{\alpha}^{(k+1)} \leftarrow \left(1+\lambda^*_{(k)}\right)\boldsymbol{\alpha}^{(k)} - \lambda^*_{(k)}\mathbf{e}\left(p^*_{(k)}\right)$

Replacing TOWARD in Algorithm 1 (line 10) with DECIDESTEP, we obtain a modified SFW algorithm with away steps, named MSFW.

4 Convergence Analysis

In this section, we give the rate of convergence of SFW and MSFW, where we use some techniques from [25] and [31]. We firstly introduce the *Wolfe dual* and its weak duality [30].

DEFINITION 4.1. *Let f and g_i be concave and continuously differentiable on $\mathbb{R}^N$. The primal problem is*

$$\max\ f(\mathbf{x}) \quad s.t.\ g_i(\mathbf{x}) \geq 0, i = 1, \ldots, m. \tag{4.12}$$

Its Wolfe dual is

$$\begin{aligned} &\min\ L(\mathbf{x}, u) = f(\mathbf{x}) + \sum_{i=1}^{m} u_i g_i(\mathbf{x}) \\ &s.t.\ \nabla f(\mathbf{x}) + \sum_i u_i \nabla g_i(\mathbf{x}) = 0, \quad u \geq 0 \end{aligned} \tag{4.13}$$

The following proposition establishes the weak duality of Wolfe dual.

PROPOSITION 4.1. *Let $q^* = \inf_{u\geq 0}\sup_{\mathbf{x}} L(\mathbf{x}, u)$, $f^* = \sup_{\mathbf{x}}(f(\mathbf{x}))$. Then $q^* \geq f^*$.*

Applying this weak duality to a simplex-constrained concave problem as used in (aD), we have the following result:

LEMMA 4.1. *Let $\boldsymbol{\alpha}^*$ be the optimal solution of (aD). Solving (aD) using SFW algorithm with toward directions $\mathbf{d}_t^{(k)} = \mathbf{e}(p^*_{(k)}) - \boldsymbol{\alpha}^{(k)}$, the following holds for every iteration k:*

$$\mathbf{d}_t^{(k)T}\nabla f(\boldsymbol{\alpha}) \geq f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)}). \tag{4.14}$$

Proof. Consider a general unit simplex-constrained concave problem

$$\max_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}),\ s.t.\ \boldsymbol{\alpha} \in \triangle. \tag{4.15}$$

Introducing Lagrange multipliers y and z_i we obtain its Wolfe dual:

$$\min_{\boldsymbol{\alpha},y,z_i} f(\boldsymbol{\alpha}) - \boldsymbol{\alpha}^T\nabla f(\boldsymbol{\alpha}) - y$$
$$s.t.\ y + z_i + \nabla f(\boldsymbol{\alpha})_i = 0,\ \forall i.$$

The constraint is equivalent to $-y \geq \max_i \nabla f(\boldsymbol{\alpha})_i$. Taking the smallest feasible $-y = \max_i \nabla f(\boldsymbol{\alpha})_i$, the above dual problem can be rewritten as:

$$\min_{\boldsymbol{\alpha}} \left(f(\boldsymbol{\alpha}) - \boldsymbol{\alpha}^T\nabla f(\boldsymbol{\alpha}) + \max_i \nabla f(\boldsymbol{\alpha})_i \right).$$

Using Proposition 4.1, it follows that

$$f(\boldsymbol{\alpha}) - \boldsymbol{\alpha}^T\nabla f(\boldsymbol{\alpha}) + \max_i \nabla f(\boldsymbol{\alpha})_i \geq f(\boldsymbol{\alpha}^*). \tag{4.16}$$

Now replacing the general concave f in (4.15) with the one in (aD-k), (4.16) can be rewritten as:

$$f(\boldsymbol{\alpha}^{(k)}) - \boldsymbol{\alpha}^{(k)T}\nabla f(\boldsymbol{\alpha}) + \max_{i\in S^{(k)}} \nabla f(\boldsymbol{\alpha})_i \geq f(\boldsymbol{\alpha}^*_{(k)}) \tag{4.17}$$

where $\boldsymbol{\alpha}^*_{(k)}$ is the optimal solution of (aD-k).

Since matrix A defined in (3.7) is symmetric positive definite (s.p.d.), and $\boldsymbol{\alpha}$ for non-working samples are all set to 0, we can decompose the objective function of (aD) as $-\boldsymbol{\alpha}^T A\boldsymbol{\alpha} = -\boldsymbol{\alpha}^T A_k\boldsymbol{\alpha} - \boldsymbol{\alpha}^T(A - A_k)\boldsymbol{\alpha}$, where $-\boldsymbol{\alpha}^T A_k\boldsymbol{\alpha}$ corresponds to the working samples $S^{(k)}$ used for solving (aD-k). Since A_k is s.p.d., using similar techniques as in [6], we can prove that $A - A_k$ is also s.p.d.. It follows that

$$\begin{aligned} f(\boldsymbol{\alpha}^*_{(k)}) &= -\boldsymbol{\alpha}^T A_k\boldsymbol{\alpha} = -\boldsymbol{\alpha}^T A\boldsymbol{\alpha} + \boldsymbol{\alpha}^T(A - A_k)\boldsymbol{\alpha} \\ &\geq -\boldsymbol{\alpha}^T A\boldsymbol{\alpha} = f(\boldsymbol{\alpha}^*). \end{aligned} \tag{4.18}$$

Combining (4.17) and (4.18), we have

$$\max_{i\in S^{(k)}} \nabla f(\boldsymbol{\alpha})_i - \boldsymbol{\alpha}^{(k)T}\nabla f(\boldsymbol{\alpha}) \geq f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)})$$

which completes the proof.

We are now ready to present the rate of convergence for SFW.

THEOREM 4.1. *Algorithm SFW solves (aD) such that there exist a constant $\zeta \geq -1/2$, and for all $k = 1, 2, \ldots$*

$$f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)}) \leq \frac{Q}{k+\zeta}, \tag{4.19}$$

where $Q = \sup_k Q_k$ and $Q_k = \mathbf{d}_t^{(k)T} A \mathbf{d}_t^{(k)}$.

Proof. Using the updating rule of SFW for $\boldsymbol{\alpha}$, we have:

$$f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k+1)}) = f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)}) - \frac{\left[\mathbf{d}_t^{(k)T} A\boldsymbol{\alpha}^{(k)}\right]^2}{Q_k}.$$

Using Lemma 4.1 we have

$$f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k+1)}) \leq f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)}) - \frac{\left[f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)})\right]^2}{Q_k}.$$

Denoting $\beta_k = f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)})$, it follows that

$$\beta_{k+1} \leq \beta_k - \frac{\beta_k^2}{Q_k} \leq \frac{\beta_k}{1 + \frac{\beta_k}{Q_k}} = \frac{1}{\frac{1}{Q_k} + \frac{1}{\beta_k}} \leq \frac{1}{\frac{1}{Q} + \frac{1}{\beta_k}}.$$

For $k = 1$, denote its stepsize $\lambda^*_{(1)} = \eta$. Then

$$\eta = \lambda^*_{(1)} = \frac{\mathbf{d}_t^{(1)T}\nabla f(\boldsymbol{\alpha})}{2Q_1} \leq 1.$$

Using Lemma 4.1, $\beta_1 \leq \mathbf{d}_t^{(1)T}\nabla f(\boldsymbol{\alpha}) = 2\eta Q_1 \leq 2\eta Q$. By induction we have:

$$\beta_2 \leq \frac{1}{\frac{1}{Q} + \frac{1}{2\eta Q}} = \frac{Q}{1 + 1/2\eta}, \ldots, \beta_k \leq \frac{Q}{k + (\frac{1}{2\eta} - 1)}.$$

Since $0 \leq \eta \leq 1$, $\zeta = \frac{1}{2\eta} - 1 \geq -1/2$ which completes the proof.

Theorem 4.1 means that, to achieve ϵ-accuracy on the dual problem, i.e. $\epsilon = f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha})$, Algorithm 1 needs $t = O(\frac{Q}{\epsilon})$ iterations. It follows that the worst-case time complexity of Algorithm 1 is $\sum_{k=1}^t O(dk) = O(dt^2) = O\left(d\frac{Q^2}{\epsilon^2}\right)$.

The convergence analysis for MSFW is more involved. We firstly prove that for k large enough, all FW steps will be away steps.

LEMMA 4.2. *In MSFW, there exist $M > 0$ such that for all $k \geq M$, $\mathbf{d}_t^{(k)T}\nabla f(\boldsymbol{\alpha}^{(k)}) \leq \mathbf{d}_a^{(k)T}\nabla f(\boldsymbol{\alpha}^{(k)})$.*

Proof. See the first part of Theorem 5's proof in [31].

The rate of convergence is give below.

THEOREM 4.2. *Algorithm MSFW solves (aD) such that there exists $0 < c < 1$, and for all $k \geq M$,*

$$\frac{f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k+1)})}{f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^{(k)})} \leq c, \tag{4.20}$$

where M is defined in Lemma 4.2.

Proof. (scratch) The objective function $f(\boldsymbol{\alpha})$ is Lipschitz continuous and strongly convex, i.e. there exists γ_1 and γ_2 such that

$$\begin{aligned} &\gamma_1\|\boldsymbol{\alpha}^{(k+1)} - \boldsymbol{\alpha}^{(k)}\|^2 \\ &\leq \left(\boldsymbol{\alpha}^{(k+1)} - \boldsymbol{\alpha}^{(k)}\right)^T \left(\nabla f(\boldsymbol{\alpha}^{(k+1)}) - \nabla f(\boldsymbol{\alpha}^{(k)})\right) \\ &\leq \gamma_2\|\boldsymbol{\alpha}^{(k+1)} - \boldsymbol{\alpha}^{(k)}\|^2. \end{aligned} \tag{4.21}$$

The following proof essentially follows the proof of Theorem 2 in [31].

5 Experimental Results

5.1 Datasets In this section, several real-world datasets from various application domains will be used to evaluate the efficiency of the proposed stochastic algorithms. The first four datasets are at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`. The last two are at `http://www.cse.ust.hk/~ivor/cvm.html`. The sizes of these datasets range from small scale (thousands samples) to mid-large scale (half a million samples). In some datasets, the number of training samples for each class are fairly unbalanced. Among these datasets, "w3a" is text data which is of relatively high dimension but very sparse. Table 1 gives the details of them.

5.2 SVM Solvers for Comparison The proposed algorithms are SFW: stochastic Frank-Wolfe and MSFW: modified stochastic Frank-Wolfe with away steps. We compare them with several existing methods described as below.

- SMO-L1: sequential minimal optimization for L1-SVM [8]. This is one of the most popular solver for nonlinear SVM classifications with hinge loss.
- SMO-L2: sequential minimal optimization for L2-SVM. The squared hinge loss is adopted. The implementation of its training algorithm is essentially the same as SMO-L1. The only differences are: the box constrains of SMO-L1 is replaced with an non-negativity constraint; an additional $1/2C$ term is added in each kernel calculation.
- SMO-shrink: SMO with shrinking technique [9]. The shrinking technique is based on the heuristics that some samples that have reached their bounds tend to remain unchanged in following iterations, thus can be temporarily removed from the working set. In our implementations, the working set is shrunk after every 1000 iterations.
- SMO-wss2: SMO with shrinking technique and second order information for working set selection [10]. It is the state-of-the-art nonlinear SVM solver used in LibSVM.
- SGD-linear: stochastic gradient descent for online/stochastic linear SVM with hinge loss [18]. The learning rate is chosen as $1/k$ for the k^{th} iteration.
- SGD-kernel: kernelized stochastic gradient descent for online/stochastic nonlinear SVM with hinge loss, also known as NORMA [17]. The learning rate is chosen as $1/k$ for the k^{th} iteration. The margin parameter ρ is set to 1, which corresponds to hinge loss.
- Pegasos: primal estimated sub-gradient solver for online/stochastic linear SVM with hinge loss [19].

5.3 Parameters and Implementation Issues For nonlinear methods, we choose Gaussian radial basis function kernels $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\sigma^2)\right)$. The kernel bandwidth σ and the parameter C in (sD) and (aD) are chosen via grid searches, and the results with best testing accuracies are reported. Note that the C parameters for L1-SVM and L2-SVM are generally different. Table 2 shows the different parameter combinations used for our datasets.

Table 2: Parameters used for L1-SVM and L2-SVM.

Dataset	σ	C for L1-SVM	C for L2-SVM
svmguide1	20	1	0.4
w3a	4	18	5
a9a	10	1.5	0.5
ijcnn1	0.61	7	5
usps01	10	10	50
covertype	100	10000	1000

For stopping criteria in batch learning tasks, all the above SMO-type algorithms will stop when the "gap" $\min_{i\in I_{\text{up}}} y_i g_i - \max_{i\in I_{\text{down}}} y_i g_i$ is less than 0.001, where $I_{\text{up}} = \{i|\alpha_i < C \text{ if } y_i = 1, \text{ or } \alpha_i > 0 \text{ if } y_i = -1\}$ and $I_{\text{down}} = \{i|\alpha_i > 0 \text{ if } y_i = 1, \text{ or } \alpha_i < C \text{ if } y_i = -1\}$. This tolerance is the same as the default stopping tolerance used in LibSVM. For all the stochastic algorithms, they will stop after running for 1 or 2 epochs of iterations, where an epoch stands for N iterations.

We do not include the caching technique in SMO or our SFW implementations for the purpose of fair comparisons, since its performance varies for different datasets.

All the methods are implemented in C++, and all the data are in double-precision. The experiments are carried out on a workstation with Xeon 3.00GHz CPU, 8 GB RAM and 64-bit Linux system. In the following results, all the times shown are training time measured in CPU time, excluding the time spent for data loading and prediction.

5.4 Comparisons on Convergence In this section, we will compare performances of SFW and MSFW to other nonlinear SVM solvers, namely SMO-L1, SMO-

Table 1: Real-world datasets from various application domains.

Dataset Name	# of Classes	# of Training Samples (N)	# of Testing Samples	# of Dim. (d)	Density	Comments
svmguide1	2	3,089	4,000	4	100%	real values; class 1: 64.7%
w3a	2	4,912	44,837	300	3.88%	integer values; class -1: 97.1%
a9a	2	32,561	16,281	123	11.28%	integer values; class -1: 75.9%
ijcnn1	2	35,000	91,701	22	59.09%	integer and real values; class -1: 90.2%
usps01	2	266,079	75,383	676	14.95%	real values; class 1: 54.3%
covertype	2	522,910	58,102	54	22.00%	integer values; class -1: 51.2%

L2, SMO-shrink, SMO-wss2 and SGD-kernel. The primal/dual values are expensive to calculate, so instead of evaluating the convergence on objective values, we show the convergence history in testing accuracy, which is indeed what we care about in practice.

For each dataset and solver, we run several trials with different number of iterations. After each trial we use the testing sets to calculate testing accuracies. Fig.2∼7 show the convergence histories of testing accuracies, against training time.

In our implementations of stochastic algorithms, in order to mimic online learning scenarios, each dataset is randomly permuted before feeding to the solvers. Hence these plots can also be used to evaluate the performance of SFW and MSFW in online learning tasks.

Note that SGD-kernel is not shown in Fig. 3 and 5, since our experiments show that the method's testing accuracy is always the portion of the major class for datasets "w3a" and "ijcnn1", in which the two classes are high unbalanced.

It can be observed that stochastic Frank-Wolfe methods can quickly reach acceptable accuracies in the very early iterations. Although this is a general benefit of stochastic algorithms, SFW and MSFW still converge much faster than the stochastic SGD-kernel.

Although MSFW is proved to have a faster convergence rate than SFW, this theoretical advantage is not very prominent in our experiments. This deserves further investigation.

5.5 Comparisons on Batch Learning Tasks In this section, we will compare the performances of nonlinear solvers for batch learning. In batch learning tasks, normally a pre-defined stopping criterion should be set to terminate the optimization process.

As we will shown in Table 3, running SFW or MSFW for 1 or 2 epochs is enough to obtain very good testing accuracies. As a comparison, for SMO-type algorithms, we use the stopping criterion as discussed in section 5.3. This criterion is widely adopted by many software packages, e.g. LibSVM and SVMLight.

The testing accuracies in the left half of Table 3 show that all the proposed stochastic Frank-Wolfe algorithms achieve comparable or even higher accuracies than the rest of the L1/L2-SVM solvers. The right

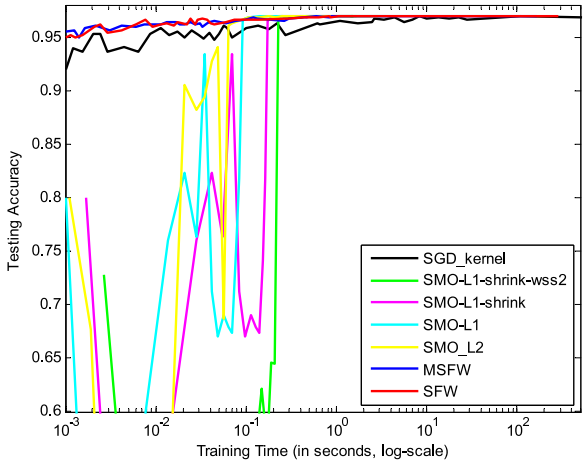

Figure 2: Dataset: svmguide1

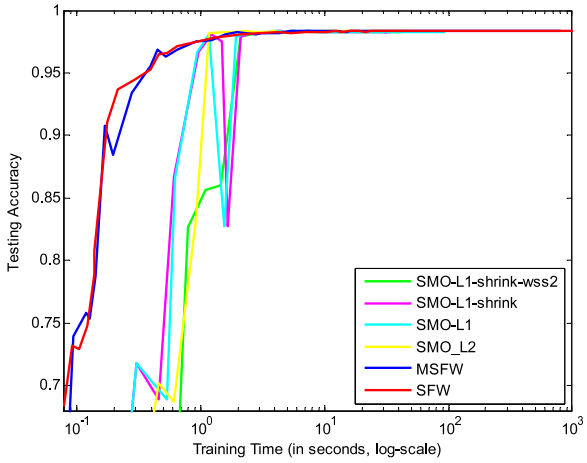

Figure 3: Dataset: w3a

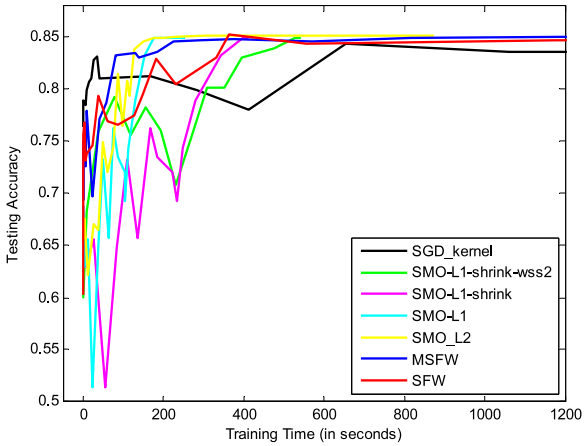

Figure 4: Dataset: a9a

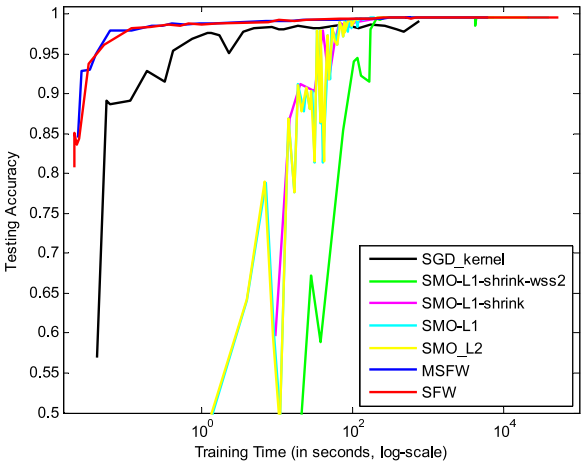

Figure 6: Dataset: usps01

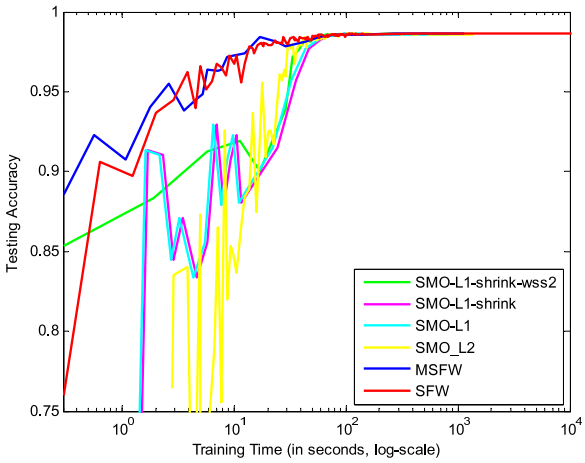

Figure 5: Dataset: ijcnn1

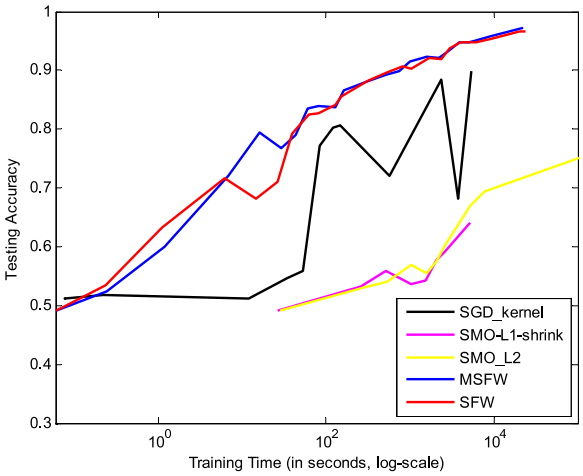

Figure 7: Dataset: covertype

half shows that the proposed algorithms are consistently faster than SMO-type methods. Table 4 shows the number of iterations and number of support vectors. We can see that the Frank-Wolfe methods have generally larger amount of support vectors than L1-SVM solvers. This is due to the difference between hinge loss and squared hinge loss.

5.6 Comparisons of Linear and Nonlinear SVMs In this section, we use Figure 8∼12 to demonstrate the performance of SFW algorithms against two linear SVM solvers, namely SGD-linear and Pegasos. The result is very interesting. In most datasets, the linear solvers have a much lower testing accuracy than nonlinear ones. While in some datasets, e.g. "w3a", linear SVM solvers can be much faster than nonlinear solvers, with almost comparable testing accuracy on convergence. The reason might be the sparseness of this webpage data. In such datasets, a linear hyperplane is already enough to separate the two classes.

6 Conclusions and Future Work

We propose two stochastic Frank-Wolfe algorithms for nonlinear SVMs. These algorithms are very simple and efficient for both batch and online tasks. They achieve comparable or even better accuracies than state-of-the-art batch and online algorithms, and are significantly faster. SFW has a provable time complexity $O\left(d\frac{Q^2}{\epsilon^2}\right)$.

Table 3: Experimental results for nonlinear SVM solvers. Best results are bolded and underscored.

Dataset	Best Testing Accuracy (%)								CPU Time for Training (in seconds)							
	SFW 1 epo.	SFW 2 epo.	MSFW 1 epo.	MSFW 2 epo.	SMO-L2	SMO-L1	SMO-shrink	SMO-wss2	SFW 1 epo.	SFW 2 epo.	MSFW 1 epo.	MSFW 2 epo.	SMO-L2	SMO-L1	SMO-shrink	SMO-wss2
svmguide1	96.85	96.98	**97.00**	97.00	97.00	97.00	97.00	97.00	**0.30**	0.86	0.30	0.89	2.12	2.16	1.28	1.53
w3a	98.13	98.29	98.06	**98.33**	98.32	98.30	98.30	98.30	2.94	9.74	**2.79**	10.19	32.61	72.47	31.93	34.38
a9a	**85.24**	84.69	84.87	85.13	85.21	84.92	84.92	84.92	362.35	1178.7	375.75	1257.0	872.45	**222.55**	404.96	533.72
ijcnn1	98.07	98.58	98.17	98.55	**98.61**	98.61	98.61	98.61	24.08	75.73	**23.67**	78.51	637.32	808.32	134.12	76.83
usps01	**99.54**	99.53	99.54	99.53	99.54	99.54	99.54	99.54	3262.8	7228.7	**2534.2**	6242.1	19208	23226	3470.7	4421.0
covertype	96.68	98.23	97.23	98.23	98.23	**98.24**	98.24	98.24	23549	1.33e5	**22167**	1.13e5	2.46e5	2.13e5	2.06e5	2.33e5

Table 4: Experimental results for nonlinear SVM solvers, continued.

Dataset	Number of Epochs/Iterations								Number of Support Vectors							
	SFW 1 epo.	SFW 2 epo.	MSFW 1 epo.	MSFW 2 epo.	SMO-L2	SMO-L1	SMO-shrink	SMO-wss2	SFW 1 epo.	SFW 2 epo.	MSFW 1 epo.	MSFW 2 epo.	SMO-L2	SMO-L1	SMO-shrink	SMO-wss2
svmguide1	1	2	1	2	2929	3010	3058	1250	693	971	672	942	1487	515	518	510
w3a	1	2	1	2	4178	9285	9360	4034	570	797	529	779	1273	562	572	564
a9a	1	2	1	2	34408	8829	8801	8110	12804	16743	13131	17267	20436	11980	11984	11981
ijcnn1	1	2	1	2	56139	69725	71003	10880	2772	3779	2706	3724	5286	2897	2899	2897
usps01	1	2	1	2	19394	23398	23398	11692	2509	2736	1382	1692	1959	1893	1893	1885
covertype	1	2	1	2	1.00e6	9.97e5	9.25e5	8.64e5	38004	1.10e5	29060	1.10e5	1.10e5	1.07e5	1.07e5	1.07e5

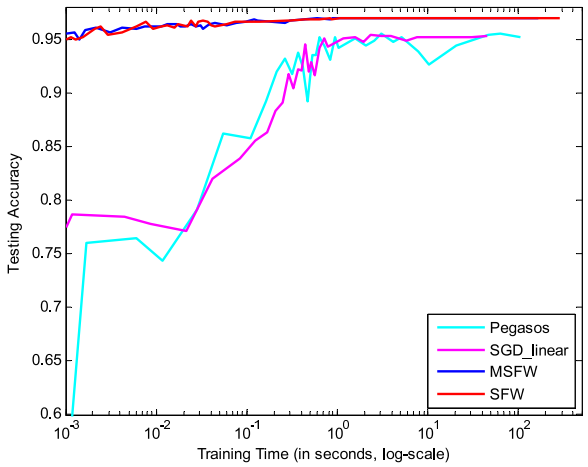

Figure 8: Dataset: svmguide1

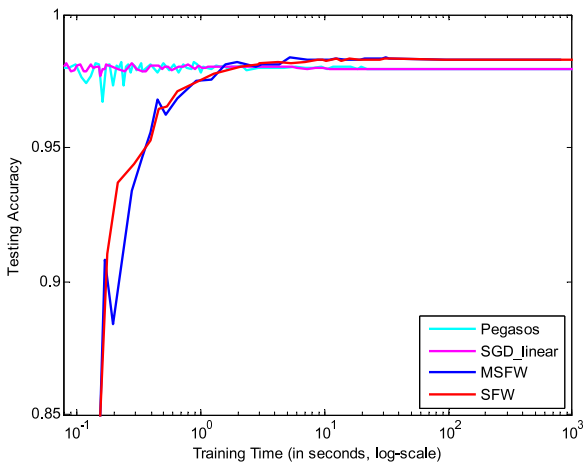

Figure 9: Dataset: w3a

On-going work includes adopting shrinking and caching techniques. We will extend SFW to regression, ranking and semi-supervised learning problems. Sparse matrix storage and computation will be implemented for further speed up. The gap between the theoretical linear convergence of MSFW and its practical performance will also be investigated.

References

[1] Vapnik, V. N. (1982) *Estimation of Dependences Based on Empirical Data.* Springer-Verlag.

[2] Joachims, T. (2002) Optimizing Search Engines Using Clickthrough Data. In, *Proc. ACM Conf. on Knowledge Discovery and Data Mining (KDD).*

[3] Smola, A. J. & Schölkopf, B. (1998) A Tutorial on Support Vector Regression. In, *NeuroCOLT2 Technical Report Series, NC2-TR-1998-030.*

[4] Schölkopf, B., Platt, J. C., Shawe-Taylor, J. & Smola, A. J., Williamson, R. C. (2001) Estimating the Support of a High-Dimensional Distribution. *Neural Computation* **13**:1443-1471.

[5] Cortes, C. & Vapnik, V. N. (1995) Support Vecotr Networks. *Machine Learning* **20**(3):273-297.

[6] Smola, A. J. & Schölkopf, B. (2000) Sparse Greedy

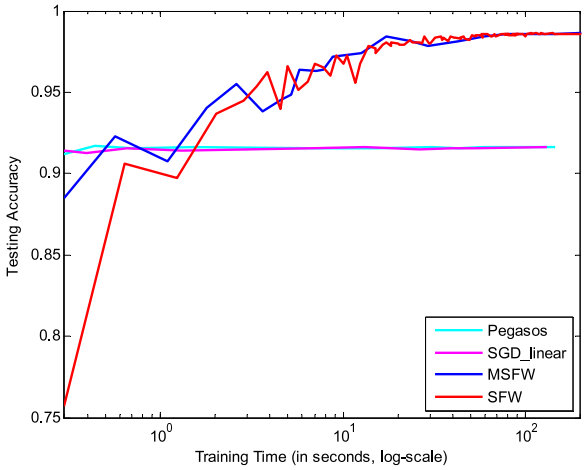

Figure 10: Dataset: ijcnn1

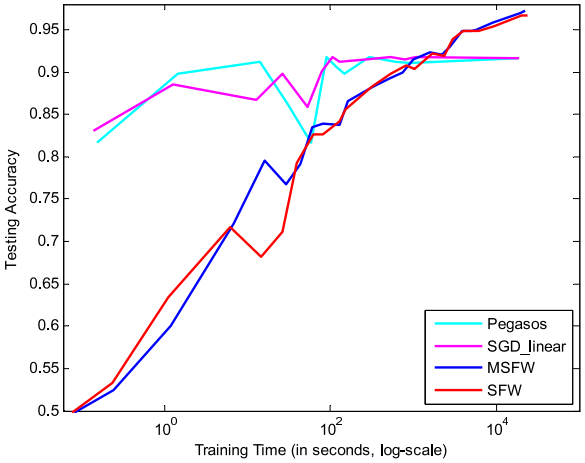

Figure 12: Dataset: covertype

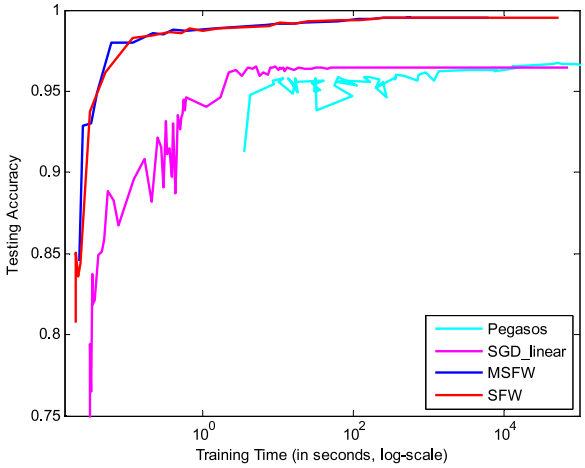

Figure 11: Dataset: usps01

Matrix Approximation for Machine Learning. In, *Proc. 17th Intl. Conf. on Machine Learning (ICML)*.

[7] Osuna, E., Freund, R. & Girosi, F. (1997). Support Vector Machines: Training and Applications. In, *Tech. Rep. AIM-1602 MIT Artificial Intelligence Laboratory*.

[8] Platt, J. C. (1999) Fast Training of Support Vector Machines using Sequential Minimal Optimization. In, *Advances in Kernel Methods: Support Vector Learning*. The MIT Press.

[9] Joachims, T. (1999) Making Large-Scale SVM Learning Practical. In, *Advances in Kernel Methods: Support Vector Learning*. The MIT Press.

[10] Fan, R. E., Chen, P. H. & Lin C. J. (2005) Working Set Selection Using Second Order Information for Training Support Vector Machines. *Journal of Machine Learning Research* **6**:1889-1918.

[11] Chang, K. W., Hsieh, C. J. & Lin, C. J. (2008) Coordinate Descent Method for Large-scale L2-loss Linear Support Vector Machines. *Journal of Machine Learning Research* **9**:1369-1398.

[12] Hsieh, C. J., Chang, K. W. & Lin, C. J. (2008) A Dual Coordinate Descent Method for Large-scale Linear SVM. In, *Proc. 25th Intl. Conf. on Machine Learning (ICML)*.

[13] Joachims, T. (2005) Training Linear SVMs in Linear Time. In, *Proc. ACM Conf. on Knowledge Discovery and Data Mining (KDD)*.

[14] Teo, C. H., Le, Q., Smola, A. & Vishwanathan, S.V.N. (2007) A Scalable Modular Convex Solver for Regularized Risk Minimization. In, *Proc. ACM Conf. on Knowledge Discovery and Data Mining (KDD)*.

[15] Leon Bottou & Olivier Bousquet (2007) The Tradeoffs of Large Scale Learning . In, *Advances in Neural Information Processing Systems (NIPS)*.

[16] David Saad (1998) *On-Line Learning in Neural Networks*. Cambridge University Press.

[17] Kivinen, J., Smola, A. J. & Williamson, R. C. (2004) Online Learning with Kernels. *IEEE Trans. on Signal Processing* **52**(8):2165-2176.

[18] Zhang, T. (2004) Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms. In, *Proc. 21st Intl. Conf. on Machine Learning (ICML)*.

[19] Shalev-Shwartz, S., Singer, Y. & Srebro, N. (2007) Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In, *Proc. 24th Intl. Conf. on Machine Learning (ICML)*.

[20] Kushner, Harold J. & Yin, G. George(2003) *Stochastic Approximation and Recursive Algorithms and Applications*. 2nd Edition, Springer-Verlag.

[21] Bottou, Leon & LeCun, Yann (2005) On-line Learning

for Very Large Datasets. *Applied Stochastic Models in Business and Industry*

[22] Nicol N. Schraudolph, Jin Yu & Simon Günter (2007) A Stochastic Quasi-Newton Method for Online Convex Optimization. In *AISTATS*.

[23] Bordes, A., Bottou, L. & Gallinari, P. (2009) SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent. *Journal of Machine Learning Research* **10**:1737-1754.

[24] Tsang, I. W., Kwok, J. T. & Cheung, P. M. (2005) Core Vector Machines: Fast SVM Training on Very Large Data Sets. *Journal of Machine Learning Research* **6**:363-392.

[25] Clarkson, K. L. (2008) Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. In, *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[26] Frank, M. & Wolfe, P. (1956) An Algorithm for Quadratic Programming. *Naval Research Logistics Quarterly* **3**(1-2):95-110.

[27] Wolfe, P. (1970) Convergence Theory in Nonlinear Programming. In, *Integer and Nonlinear Programming.* North-Holland Publishing Company.

[28] Alexander Shapiro, Darinka Dentcheva & Andrzej Ruszczynski (2009) *Lectures on Stochastic Programming: Modeling and Theory.* SIAM.

[29] Bertsekas, D. P. (1999) *Nonlinear Programming, 2nd Edition.* Athena Scientific.

[30] Wolfe, Philip (1961) A Duality Theorem for Non-Linear Programming. *Quarterly of Applied Mathematics* **19**:239-244.

[31] Guelat, Jacques & Marcotte, Patrice. (1986) Some Comments on Wolfe's 'Away Step'. *Mathematical Programming* **35**:110-119.