ASSET: Approximate Stochastic Subgradient Estimation Training for Support Vector Machines

Sangkyun Lee and Stephen J. Wright

Abstract—Subgradient methods for SVMs have been successful in solving the primal formulation with linear kernels. The approach is extended here to nonlinear kernels, and the assumption of strong convexity of the objective is dropped, allowing an intercept term to be used in the classifier.

1 Introduction

Support vector machines (SVMs) have become a highly successful methodology in machine learning and data mining. Derivation, implementation, and analysis of efficient solution methods for SVMs have been the subject of a great deal of research during the past 12 years. We broadly categorize the algorithms that have been proposed as follows.

(i) *Decomposition* techniques based on the dual SVM formulation, including SMO [17], LIBSVM [5], SVM-Light [9], GPDT [21], and an online variant LASVM [1]. The dual formulation allows nonlinear kernels to be introduced neatly into the formulation via kernel trick [20].

(ii) *cutting-plane* methods based on the primal formulation: SVM-Perf [10] and OCAS [7] handle linear kernels, and their extensions to nonlinear kernels in a new version of SVM-Perf [11] (which we refer to as CPNY) and CPSP [12].

(iii) *subgradient* methods for linear kernels in the primal formulation, for which we have Pegasos [22] and SGD [2].

When datasets are extremely large, the computation required by some of the existing algorithms becomes excessive. Subgradient methods are of our particular interest. They take simple steps, each typically based on a single training point, so can be implemented in a data-streaming context. They are simple to implement. While requiring a great many iterates to find accurate solutions, they can sometimes calculate solutions that are "accurate enough" for the purposes at hand using much less computation than frameworks that more explicitly target an exact solution. The subgradient methods mentioned above are closely aligned to stochastic approximation (SA) methods [15], [16] and incremental subgradient methods [14] from the optimization literature, and to incremental methods for online learning [23].

This paper outlines an improved algorithm based on subgradient methods for solving primal SVM formulations. It extends current subgradient methods by allowing nonlinear kernels to be used, and not requiring strict convexity of the function to be minimized. This allows the classic SVM formulation with a non-penalized intercept term to be used, thus reclaiming the formulation on which many theoretic results have been built.

Our approach uses low-dimensional approximations to nonlinear kernels, obtained either by approximating the Gram matrix, or by constructing the subspace with random bases. The approximation yields a *linear* formulation with transformed feature vectors, which can be solved with the use of well known subgradient approaches. The approach has the added benefits that the approximate solution to the SVM yields an approximate classification function that can be evaluated cheaply, typically in time proportional to the dimension of approximation.

We mention two more approaches that are related to the methods of this paper. Fine & Scheinberg [6] use a low-rank approximation to the Gram matrix to obtain a reduced dual formulation whose structure can be exploited by an interior-point solver. Rahimi & Recht [19] exploit random projections to have simple least-squares formulation; we re-use their projection technique as part of our approach. Our method differs from these in that (i) we find an approximate solution of the primal SVM formulation in much less time than an interior-point method would require; and (ii) we use the SVM formulation rather than the least-squares formulation of [19], and (iii) out approach works both in batch and online settings.

2 Nonlinear SVM in the Primal

In this section we develop a general theory for nonlinear SVM in the primal form focusing on classification, then show how it reformulates to a linear SVM problem by

• *S. Lee and S. J. Wright are with the Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, 53706. E-mail: {sklee, swright}@cs.wisc.edu*

Manuscript received January 00, 2000; revised January 00, 2000.

means of the low-dimensional approximation of a kernel. We discuss techniques for approximating the kernel and finally how to classify data points efficiently.

2.1 Basic Derivation

Here we use the tools of convex analysis to justify the primal SVM formulation with kernels, which was first introduced by Chapelle [3] but no rigorous justification has been done to our knowledge. Consider the training point and label pairs $\{(\mathbf{t}_i, y_i)\}_{i=1}^m$, $\mathbf{t}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$ and feature mapping $\phi : \mathbb{R}^n \to \mathbb{R}^d$. Given a convex loss function $\ell : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$, the primal SVM problem can be formulated as follows:

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \ \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{m}\sum_{i=1}^m \ell(y_i(\mathbf{w}^T\phi(\mathbf{t}_i) + b)) \tag{1}$$

with $\lambda > 0$ and a convex loss function ℓ. The necessary and sufficient optimality conditions are

$$\lambda\mathbf{w} + \frac{1}{m}\sum_{i=1}^m \chi_i y_i \phi(\mathbf{t}_i) = 0, \tag{2a}$$

$$\frac{1}{m}\sum_{i=1}^m \chi_i y_i = 0, \tag{2b}$$

$$\text{for some } \chi_i \in \partial\ell(y_i(\mathbf{w}^T\phi(\mathbf{t}_i) + b)),\ i = 1, 2, \ldots, m. \tag{2c}$$

where $\partial\ell$ is the subdifferential of ℓ.

We now consider the following substitution:

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{t}_i) \tag{3}$$

(which mimics the form of (2a)) and, motivated by this expression, we formulate the following problem

$$\min_{\alpha\in\mathbb{R}^m, b\in\mathbb{R}} \ \frac{\lambda}{2}\alpha^T\Psi\alpha + \frac{1}{m}\sum_{i=1}^m \ell(\Psi_{i\cdot}\alpha + y_i b), \tag{4}$$

where $\Psi \in \mathbb{R}^{m\times m}$ is defined by

$$\Psi_{ij} := y_i y_j \phi(\mathbf{t}_i)^T\phi(\mathbf{t}_j), \quad i, j = 1, 2, \ldots, m, \tag{5}$$

and $\Psi_{i\cdot}$ denotes the i-th row of Ψ. Optimality conditions for (4) are as follows:

$$\lambda\Psi\alpha + \frac{1}{m}\sum_{i=1}^m \beta_i \Psi_{i\cdot}^T = 0, \tag{6a}$$

$$\frac{1}{m}\sum_{i=1}^m \beta_i y_i = 0, \tag{6b}$$

$$\text{for some } \beta_i \in \partial\ell(\Psi_{i\cdot}\alpha + y_i b),\ i = 1, 2, \ldots, m. \tag{6c}$$

The following result shows that the solution of (4) can be used to derive a solution of (1). This observation is potentially interesting because (4) is formulated in terms of the kernel Ψ and does not require explicit knowledge of the feature mapping ϕ.

Proposition 1: Let $(\alpha, b) \in \mathbb{R}^m \times \mathbb{R}$ be a solution of (4). Then if we define $\mathbf{w}$ by (3), $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ is a solution of (1).

Proof: Since (α, b) solves (4), the conditions (6) hold, for some β_i, $i = 1, 2, \ldots, m$. To prove the claim, it suffices to show that $(\mathbf{w}, b)$ and χ satisfy (2), where $\mathbf{w}$ is defined by (3) and $\chi_i = \beta_i$ for all $i = 1, 2, \ldots, m$.

By substituting (5) into (6), we have

$$\lambda\sum_{i=1}^m y_i y_j \phi(\mathbf{t}_j)^T\phi(\mathbf{t}_i)\alpha_i + \frac{1}{m}\sum_{i=1}^m \beta_i y_i y_j \phi(\mathbf{t}_j)^T\phi(\mathbf{t}_i) = 0, \quad j = 1, 2, \ldots, m,$$

$$\frac{1}{m}\sum_{i=1}^m \beta_i y_i = 0,$$

$$\beta_i \in \partial\ell(\sum_{j=1}^m y_i y_j \phi(\mathbf{t}_j)^T\phi(\mathbf{t}_i)\alpha_j + y_i b), \quad i = 1, 2, \ldots, m.$$

From the first equality above, we have that

$$-\sum_{i=1}^m \left(\alpha_i + \frac{1}{\lambda m}\beta_i\right) y_i \phi(\mathbf{t}_i) + \xi = 0,$$

for some $\xi \in \text{Null}\left(\left[y_j\phi(\mathbf{t}_j)^T\right]_{j=1}^m\right)$. Since the two components in this sum are orthogonal, we have

$$0 = \left\|\sum_{i=1}^m \left(\alpha_i + \frac{1}{\lambda m}\beta_i\right) y_i \phi(\mathbf{t}_i)\right\|_2^2 + \xi^T\xi,$$

which implies that $\xi = 0$. We can therefore rewrite the optimality conditions for (4) as follows:

$$\sum_{i=1}^m \left(\lambda\alpha_i + \frac{1}{m}\beta_i\right) y_i \phi(\mathbf{t}_i) = 0, \tag{7a}$$

$$\frac{1}{m}\sum_{i=1}^m \beta_i y_i = 0, \tag{7b}$$

$$\beta_i \in \partial\ell(y_i\phi(\mathbf{t}_i)^T\sum_{j=1}^m \alpha_j \phi(\mathbf{t}_j) + y_i b), \ i = 1, 2, \ldots, m. \tag{7c}$$

By defining $\mathbf{w}$ as in (3) and setting $\chi_i = \beta_i$ for all i, we see that (7) is identical to (2), as claimed. □

While Ψ is clearly symmetric positive semidefinite, the proof makes no assumption about nonsingularity of this matrix, or uniqueness of the solution α of (4). However, (6a) suggests that without loss of generality, we can constrain α to have the form $\alpha_i = -\beta_i/(\lambda m)$ where β_i is restricted to $\partial\ell$. In particular, if we use hinge loss function, that is,

$$\ell(\delta) := \max\{0, 1 - \delta\}, \tag{8}$$

the subdifferential is

$$\partial\ell(\delta) = \begin{cases} \{-1\} & \text{if } \delta < 1, \\ [-1, 0] & \text{if } \delta = 1, \\ \{0\} & \text{if } \delta > 1. \end{cases}$$

Thus $\beta_i \in [-1, 0]$ for all $i = 1, 2, \ldots, m$.

2.2 Reformulation to a Linear SVM Problem

We consider the feature mapping $\phi^\circ : \mathbb{R}^n \to \mathcal{H}$ to a Hilbert space $\mathcal{H}$ induced by a kernel function $k^\circ : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, where k° satisfies the conditions of Mercer's Theorem [20] to guarantee the existence of ϕ° satisfying $k^\circ(\mathbf{s},\mathbf{t}) := \langle \phi^\circ(\mathbf{s}), \phi^\circ(\mathbf{t})\rangle$. Suppose that we have a low-dimensional approximation $\phi : \mathbb{R}^n \to \mathbb{R}^d$ of ϕ° for which

$$k^\circ(\mathbf{s},\mathbf{t}) \approx \phi(\mathbf{s})^T \phi(\mathbf{t}), \tag{9}$$

for all feature $\mathbf{s}$ and $\mathbf{t}$ of interest. We construct a matrix $V \in \mathbb{R}^{m\times d}$ for training examples $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_m$ by defining the ith row as $V_{i\cdot} = y_i \phi(\mathbf{t}_i)^T,\ i = 1, 2, \ldots, m$. Then V satisfies

$$\Psi := VV^T \approx \Psi^\circ := [y_i y_j k^\circ(\mathbf{t}_i, \mathbf{t}_j)]_{i,j=1,2,\ldots,m}. \tag{10}$$

Note that Ψ is a rank-d, positive semidefinite approximation to Ψ°. By substituting this Ψ in (4), we obtain

$$\min_{\alpha\in\mathbb{R}^m, b\in\mathbb{R}} \frac{\lambda}{2}\alpha^T VV^T\alpha + \frac{1}{m}\sum_{i=1}^m \ell(\mathbf{v}_i^T V^T\alpha + y_i b), \tag{11}$$

where $\mathbf{v}_i := V_{i\cdot}^T$ is the transpose of the i-th row of V. By introducing the change of variables

$$\gamma = V^T\alpha, \tag{12}$$

we obtain the equivalent formulation

$$\min_{\gamma\in\mathbb{R}^d, b\in\mathbb{R}} \frac{\lambda}{2}\gamma^T\gamma + \frac{1}{m}\sum_{i=1}^m \ell(\mathbf{v}_i^T\gamma + y_i b). \tag{13}$$

This problem is a *linear* SVM with feature vectors $y_i\mathbf{v}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, m$.

We can solve (13) by applying linear SVM techniques to find (γ, b). Any α that solves the overdetermined system (12) will yield a solution of (11). (Note that α satisfying (12) need have at most d nonzeros.) In Section 2.4, we provide an efficient way to classify data points without recovering α.

2.3 Approximating the Kernel

We discuss two techniques for finding V that satisfies (10). The first uses randomized linear algebra to calculate a low-rank approximation to the scaled Gram matrix Ψ°, as in (10). The second approach approximates the feature mapping $\phi^\circ(\cdot)$ explicitly by approximate feature mapping $\phi(\cdot)$ constructed using random projections.

2.3.1 Kernel Matrix Approximation

Our first approach is to use the Nyström method [4], to find a good approximation of specified rank d to the $m \times m$ matrix Ψ° in (10). In this approach, we specify some integer s with $d \le s < m$, and choose s elements at random from the index set $\{1, 2, \ldots, m\}$ to form a subset $\mathcal{S}$. We then find the best rank-d approximation to $(\Psi^\circ)_{\mathcal{SS}}$ and denote it by $W_{s,d}$, with pseudo-inverse $W_{s,d}^+$. We then choose V so that

$$VV^T = (\Psi^\circ)_{\cdot\mathcal{S}} W_{s,d}^+ (\Psi^\circ)_{\cdot\mathcal{S}}^T, \tag{14}$$

where $(\Psi^\circ)_{\cdot\mathcal{S}}$ denotes the column submatrix of Ψ° defined by the indices in $\mathcal{S}$. The results in [4] indicate that in expectation and with high probability, the rank-d approximation obtained by this process has an error that can be made as close as we wish to the *best* rank-d approximation by choosing s sufficiently large.

We calculate $W_{s,d}$ by forming the eigen-decomposition $(\Psi^\circ)_{\mathcal{SS}} = QDQ^T$, where Q is $s \times s$ orthogonal and D is a diagonal matrix with decreasing nonnegative diagonal entries. Taking $\bar{d}$ to be the smaller of d and the number of positive diagonals in D, we then have that

$$W_{s,d} = Q_{\cdot,1..\bar{d}} D_{1..\bar{d},1..\bar{d}} Q_{\cdot,1..\bar{d}}^T,$$

(where $Q_{\cdot,1..\bar{d}}$ denotes the first $\bar{d}$ columns of Q, and so on). The pseudo-inverse is thus

$$W_{s,d}^+ = Q_{\cdot,1..\bar{d}} D_{1..\bar{d},1..\bar{d}}^{-1} Q_{\cdot,1..\bar{d}}^T. \tag{15}$$

The matrix V satisfying (14) is therefore

$$V = (\Psi^\circ)_{\cdot\mathcal{S}} Q_{\cdot,1..\bar{d}} D_{1..\bar{d},1..\bar{d}}^{-1/2}. \tag{16}$$

For practical implementation, rather than defining d a priori, we can choose a positive threshold ϵ_d with $0 < \epsilon_d \ll 1$, then choose d to be the largest integer in $1, 2, \ldots, s$ such that $D_{dd} \ge \epsilon_d$. (In this case, we have $\bar{d} = d$.)

The time complexity of the kernel approximation discussed above is $O(s^3 + sm(n + d))$, comprising (i) a cost of $O(s^3)$ for the QDQ^T factorization of $(\Psi^\circ)_{\mathcal{SS}}$, (ii) $O(smn)$ for computation of $(\Psi^\circ)_{\cdot,\mathcal{S}}$, and (iii) $O(smd)$ for the matrix multiplication of (16). Note that the cost of (ii) and (iii) dominates the cost of (i) since $d \le s \ll m$.

2.3.2 Feature Mapping Approximation

The second approach, following [19], finds a mapping $\phi : \mathbb{R}^n \to \mathbb{R}^d$ that satisfies

$$\langle \phi^\circ(\mathbf{s}), \phi^\circ(\mathbf{t})\rangle = E\left[\langle \phi(\mathbf{s}), \phi(\mathbf{t})\rangle\right],$$

where the expectation on the right hand side is over the random variables that determine ϕ. The mapping ϕ can be constructed explicitly by random projections. Following [19], we write

$$\phi(\mathbf{t}) = \sqrt{1/d}\left[\cos(\nu_1^T\mathbf{t} + \beta_1), \cdots, \cos(\nu_d^T\mathbf{t} + \beta_d)\right]^T \tag{17}$$

where $\nu_1, \ldots, \nu_d \in \mathbb{R}^n$ are i.i.d. samples from a distribution with density $p(\nu)$ and $\beta_1, \ldots, \beta_d \in \mathbb{R}$ are from the uniform distribution on $[0, 2\pi]$. The density function $p(\nu)$ is determined by the types of the kernels we want to use. For the Gaussian kernel

$$k^\circ(\mathbf{s},\mathbf{t}) = \exp(-\|\mathbf{s} - \mathbf{t}\|_2^2/(2\sigma^2)), \tag{18}$$

it can be shown that

$$p(\nu) = \frac{1}{(2\pi)^{r/2}\sigma^2}\exp\left(-\frac{||\nu||_2^2}{2\sigma^2}\right),$$

from the Fourier transformation of k°. We can define the matrix V satisfying (10) by setting

$$\mathbf{v}_i := V_{i\cdot}^T = y_i\phi(\mathbf{t}_i),\ \ i = 1, 2, \ldots, m, \tag{19}$$

thus setting up the formulation (13).

This method is suitable for online settings, since it takes only $O(dn)$ to prepare $\nu_1, \ldots, \nu_d$ (assuming that sampling each component of these vectors takes constant time) and $O(d)$ to process each data point. As we observe in Section 4, however, this approach tends to give lower classification accuracy than the first approach.

2.4 Efficient Classification

Given the solution (γ, b) of (4), we now describe how a new data point $\mathbf{t} \in \mathbb{R}^n$ can be classified efficiently. The imposed low dimensionality of the approximate kernel in our approach can lead to significantly lower cost of classification, as low as a fraction of d/m of the cost of a full-space approach.

For the feature mapping approximation of Section 2.3.2, where ϕ is defined explicitly by (17), we use the classifier f suggested immediately by (1), that is, $f(\mathbf{t}) = \mathbf{w}^T\phi(\mathbf{t}) + b$. By substituting from (3) and using the definition (19), we obtain

$$f(\mathbf{t}) = \phi(\mathbf{t})^T \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{t}_i) + b = \phi(\mathbf{t})^T V^T \alpha + b = \phi(\mathbf{t})^T \gamma + b,$$

where we used (12) for the final equality. Note in particular that the classifier can be evaluated directly from γ; there is no need to recover α explicitly.

For the kernel approximation approach of Section 2.3.1, the classifier $\mathbf{w}^T\phi(\mathbf{t}) + b$ cannot be used directly, as we have no way to evaluate $\phi(\mathbf{t})$ for an arbitrary point $\mathbf{t}$. We can however use the approximation (9) to note that

$$\begin{aligned} \phi(\mathbf{t})^T\mathbf{w} + b &= \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{t})^T \phi(\mathbf{t}_i) + b \\ &\approx \sum_{i=1}^{m} \alpha_i y_i k^\circ(\mathbf{t}_i, \mathbf{t}) + b, \end{aligned} \tag{20}$$

so we can define the function (20) to be the classifier. To evaluate this function, we need only evaluate those kernels $k^\circ(\mathbf{t}_i, \mathbf{t})$ for which $\alpha_i \neq 0$. As noted in Section 2.2, we can satisfy (12) by using just d nonzero components of α, so (20) requires only d kernel evaluations.

If we set $\alpha_i = 0$ for all components $i \notin \mathcal{S}$, where $\mathcal{S}$ is the sample set from Section 2.3, we can compute α that approximately satisfies (12) without performing further matrix factorizations. Denoting the nonzero subvector of α by $\alpha_\mathcal{S}$, we have $V^T\alpha = V_{\mathcal{S}\cdot}^T \alpha_\mathcal{S} = \gamma$, so from (16) and the fact that $(\Psi^\circ)_{\mathcal{S}\mathcal{S}} = QDQ^T$, we have

$$\gamma = D_{1..\bar{d},1..\bar{d}}^{-1/2} Q_{\cdot,1..\bar{d}}^T (\Psi^\circ)_{\mathcal{S}\mathcal{S}} \alpha_\mathcal{S} = D_{1..\bar{d},1..\bar{d}}^{1/2} Q_{\cdot,1..\bar{d}}^T \alpha_\mathcal{S}.$$

An approximate solution of this equation (which is exact when $\bar{d} = d = s$) is

$$\alpha_\mathcal{S} = Q_{\cdot,1..\bar{d}} D_{1..\bar{d},1..\bar{d}}^{-1/2} \gamma.$$

3 Stochastic Approximation Algorithm

We describe here a stochastic approximation algorithm for solving the linear SVM reformulation (13). Consider the general convex optimization problem

$$\min_{x \in X} f(x), \tag{21}$$

where f is a convex function and X is a bounded closed convex set with the radius D_X defined by

$$D_X := \max_{x \in X} ||x||_2. \tag{22}$$

The subdifferential of f at x is denoted by $\partial f(x)$, and we use $g(x)$ to denote a particular subgradient. By convexity of f, we have

$$f(x') - f(x) \geq g(x)^T(x' - x), \quad \forall x, x' \in X, \ \forall g(x) \in \partial f(x).$$

f is *strongly* convex when there exists $\mu > 0$ such that

$$(x' - x)^T [g(x') - g(x)] \geq \mu ||x' - x||^2, \tag{23}$$

for all $x, x' \in X$, all $g(x) \in \partial f(x)$, and all $g(x') \in \partial f(x')$. Note that the objective in (13) is strongly convex in γ, but only weakly convex in b. Pegasos [22] requires f to be strongly convex in all variables and modifies the SVM formulation to have this property. The approach we describe below is suitable for the original SVM formulation.

3.1 The Algorithm

The algorithm assumes that at any $x \in X$, we have available $G(x; \xi)$, a stochastic subgradient estimate depending on random variable ξ that satisfies $E[G(x; \xi)] = g(x)$ for some $g(x) \in \partial f(x)$. The norm deviation of the stochastic subgradients is measured by D_G defined as follows:

$$E[\|G(x; \xi)\|_2^2] \leq D_G^2 \quad \forall x \in X. \tag{24}$$

At iteration j, the general algorithm takes the following step:

$$x^{j+1} = \Pi_X(x^j - \eta_j G(x^j; \xi^j)), \quad j = 1, 2, \ldots, \tag{25}$$

where ξ^j is a random variable (i.i.d. with the random variables used at previous iterations), Π_X is the projection onto X, and $\eta_j > 0$ is a step length. For our function (13), we have $x^j = (\gamma^j, b^j)$, and ξ^j is selected to be one of the indices $\{1, 2, \ldots, m\}$ with equal probability, and the subgradient estimate is constructed from the subgradient for the ξ^jth term in the summation. Specifically, when the hinge loss ℓ from (8) is used, we have

$$G\left(\begin{bmatrix}\gamma^j \\ b^j\end{bmatrix}; \xi^j\right) = \begin{bmatrix}\lambda\gamma^j + d_j \mathbf{v}_{\xi^j}^T \\ d_j y_{\xi^j}\end{bmatrix}, \tag{26}$$

where $d_j = -1$ if the kernelized training point ξ^j is currently misclassified and $d_j = 0$ otherwise. We define the feasible set X to be the Cartesian product of a ball in the γ component (due to strong duality, similarly to [22]) with an interval $[-B, B]$ for the b component:

$$X = \left\{ \begin{bmatrix}\gamma \\ b\end{bmatrix} \in \mathbb{R}^d \times \mathbb{R} : \ ||\gamma||_2 \leq 1/\sqrt{\lambda}, \ |b| \leq B \right\}$$

Algorithm 1 ASSET Algorithm

Input: $T = \{(\mathbf{t}_1, y_1), \ldots, (\mathbf{t}_m, y_m)\}$, Ψ°, λ, positive integers $\bar{N}$ and N with $0 < \bar{N} < N$, and D_X and D_G satisfying (22) and (24);
Set $(\gamma^0, b^0) \leftarrow (\mathbf{0}, 0)$, $j \leftarrow 1$;
Set $(\tilde{\gamma}, \tilde{b}) \leftarrow (\mathbf{0}, 0)$, $\tilde{\eta} = 0$;
for $j = 1, 2, \ldots, N$ **do**
  $\eta_j \leftarrow \frac{D_X}{D_G\sqrt{j}}$
  Choose $\xi^j \in \{1, \ldots, m\}$ independently at random.
  $\mathbf{v}_{\xi^j} = \begin{cases} V^T_{\xi^j \cdot} & \text{for } V \text{ as in (16)} \\ y_{\xi^j}\phi(\mathbf{t}_{\xi^j}) & \text{for } \phi(\cdot) \text{ as in (17)} \end{cases}$
  $d_j \leftarrow \begin{cases} -1 & \text{if } \mathbf{v}_{\xi^j}\gamma^j + y_{\xi^j} b < 1 \\ 0 & \text{otherwise} \end{cases}$
  $\begin{bmatrix} \gamma^j \\ b^j \end{bmatrix} \leftarrow \Pi_X \left(\begin{bmatrix} (1 - \eta_j\lambda)\gamma^{j-1} - \eta_j d_j \mathbf{v}_{\xi^j} \\ b^{j-1} - \eta_j d_j y_{\xi^j} \end{bmatrix} \right)$
  if $j \geq \bar{N}$ **then**
    {update averaged iterate}

$$\begin{bmatrix} \tilde{\gamma} \\ \tilde{b} \end{bmatrix} \leftarrow \frac{\tilde{\eta}}{\tilde{\eta} + \eta_j} \begin{bmatrix} \tilde{\gamma} \\ \tilde{b} \end{bmatrix} + \frac{\eta_j}{\tilde{\eta} + \eta_j} \begin{bmatrix} \gamma^j \\ b^j \end{bmatrix}.$$

$$\tilde{\eta} \leftarrow \tilde{\eta} + \eta_j.$$

  end if
end for
Define $\tilde{\gamma}^{\bar{N},N} := \tilde{\gamma}$ and $\tilde{b}^{\bar{N},N} := \tilde{b}$.

for sufficiently large $B > 0$, resulting $D_X = \sqrt{1/\lambda + B^2}$.

The solution of (21) is estimated not by the iterates x^j but rather by a weighted sum of the final few iterates. Specifically, if we define N to be the total number of iterates to be used and $\bar{N} < N$ to be the point at which we start averaging, the final reported solution estimate would be

$$\tilde{x}^{\bar{N},N} := \frac{\sum_{t=\bar{N}}^{N} \eta_t x^t}{\sum_{t=\bar{N}}^{N} \eta_t}. \tag{27}$$

These is no need to store all the iterates x^t, $t = \bar{N}, \bar{N} + 1, \ldots, N$ in order to evaluate (27). Instead, a running average can be maintained over the last $N - \bar{N}$ iterations, requiring the storage of only a single x.

The steplengths η_j require knowledge of the subgradient estimate variance D_G (24). We use a small sample of random variables $\xi^{(l)}$, $l = 1, 2, \ldots, M$, at the first iterate (γ^0, b^0), and estimate D_G^2 as

$$E\left[\left|\left| G\left(\begin{bmatrix} \gamma^0 \\ b^0 \end{bmatrix}; \xi\right)\right|\right|_2^2\right] \approx \frac{1}{M^2}\sum_{l=1}^{M} d_l^2(||\mathbf{v}_{\xi^{(l)}}||_2^2 + y_{\xi^{(l)}}^2).$$

We summarize this framework in Algorithm 1 and refer it as ASSET. The integer $\bar{N} > 0$ specifies the iterate at which the algorithm starts averaging the iterates, which can be set to 1 to average all iterates, to a predetermined maximum iteration number to output the last iterate without averaging, or to a number in between.

3.2 Convergence

The analysis of robust stochastic approximation in [15], [16] provides theoretical support for the algorithm above. Considering Algorithm 1 applied to the general formulation (21), and denoting the algorithm's output $\tilde{x}^{\bar{N},N}$, we have the following result.

Theorem 1: Given the output $\tilde{x}_{\bar{N},N}$ and optimal function value $f(x^*)$, we have

$$E[f(\tilde{x}^{\bar{N},N}) - f(x^*)] \leq C(\rho)\frac{D_X D_G}{\sqrt{\bar{N}}} \tag{28}$$

where $C(\rho)$ solely depends on the fraction $\rho \in (0, 1)$ for which $\bar{N} = \lceil \rho N \rceil$.

3.3 Strongly Convex Case

Suppose that we omit the intercept b from the linear formulation (13). Then its objective function $f(x)$ becomes strongly convex for all of its variables. In this special case we can apply different steplength $\eta_j = 1/(\lambda j)$ to achieve faster convergence in theory. The algorithm remains the same as Algorithm 1 except that averaging is no longer needed and a faster convergence rate can be proved: essentially a rate of $1/j$ rather than $1/\sqrt{j}$ (see [15] for a general proof). The feasible set X is simplified as follows

$$X = \{\gamma \in \mathbb{R}^d : ||\gamma||_2 \leq 1/\sqrt{\lambda}\},$$

and the update steps are changed accordingly to omit the component b. The resulting algorithm, we refer it as ASSET*, is the same as Pegasos [22] and SGD [2], except for our extension to nonlinear kernels.

Note that averaging like (27) may still be useful, as it can be shown to improve the convergence rate by some constant [18].

4 Computational Results

We based our implemented our algorithms on the open-source Pegasos code[1]. We refer our algorithms with kernel matrix approximation as ASSET and ASSET* (for the versions that do and do not allow an intercept term, resp.) and with feature mapping approximation as ASSET$_{on}$ and ASSET$^*_{on}$. In the interests of making direct comparisons with other codes, we do not include intercept terms in our experiments, since some of the other codes do not allow such terms to be used without penalization.

We run all experiments on load-free 64-bit Linux systems with 2.66 GHz processors and 8 GB memory. Kernel cache size is set to 1 GB when applicable. All experiments with randomness are repeated 50 times unless otherwise specified.

Table 1 summarizes the six binary classification tasks we use for the experiments[2]. The `ADULT` dataset is randomly split into training/validation/test sets. In the `MNIST` data set, we obtain a binary problem by classifying the digits 0-4 versus 5-9. In the `CCAT` dataset from

1. Our code is available at http://pages.cs.wisc.edu/~sklee/asset/. Pegasos is from http://mloss.org/software/view/35/.

2. `ADULT`, `MNIST`, `CCAT` and `COVTYPE` datasets are downloaded from the UCI Repository, http://archive.ics.uci.edu/ml/.

TABLE 1
Datasets and Training Parameters.

Name	m (train)	valid/test	n (density)	$\frac{1}{\lambda m}$, $\frac{1}{2\sigma^2}$
ADULT	32561	8140/8141	123 (11.2%)	1000, 0.001
MNIST	58100	5950/5950	784 (19.1%)	100, 0.01
CCAT	78127	11575/11574	47237 (1.6%)	10, 1.0
IJCNN	113352	14170/14169	22 (56.5%)	100, 1.0
COVTYPE	464809	58102/58101	54 (21.7%)	3.0, 1.0
MNIST-E	1000000	20000/20000	784 (25.6%)	100, 0.01

the RCV1 collection [13], we use the original test set as the training set, and divide the original training set into validation and test sets. IJCNN is constructed by a random splitting of the IJCNN 2001 Challenge dataset[3]. In COVTYPE, the binary problem is to classify type 1 against the other forest cover types. Finally, MNIST-E is an extended set of MNIST, generated with elastic deformation of the original digits[4]. Table 1 also indicates the values of the regularization parameter λ and Gaussian kernel parameter σ (18) selected by the SVM-Light solver [8] to maximize the classification accuracy on each validation set. (For MNIST-E we use the same parameters as in MNIST.)

For the first five batch-mode tasks, we compare our algorithms against four publicly available codes. Two of these are the cutting-plane methods CPNY [11] and CPSP [12] that are implemented in the version 3.0 of of SVM-Perf. Both search for a solution as a linear combination of approximate basis functions, where the approximation is based on Nyström sampling (CPNY) or on constructing optimal bases (CPSP). The other two comparison codes are SVM-Light [8], which solves the dual SVM formulation via a succession of small subproblems, and LASVM [1], which makes a single pass over the data, selecting pairs of examples to optimize with the SMO algorithm. The original SVM-Perf [10] and OCAS [7] are not included in the comparison because they cannot handle nonlinear kernels. For the final test — an online test with the large data set MNIST-E — we compare our online algorithms ASSET$_{on}$ and ASSET$^*_{on}$ to LASVM.

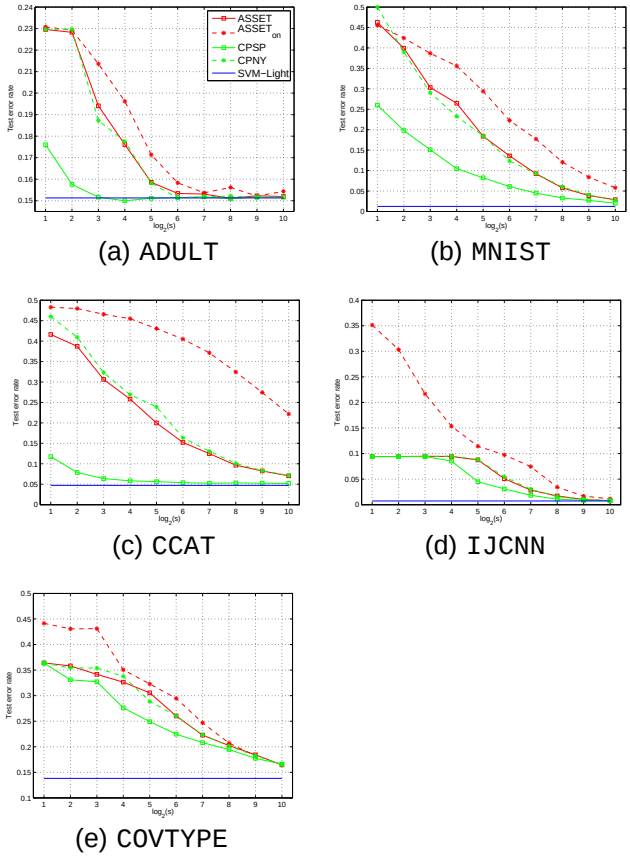
For our codes, the averaging parameter is set to $\bar{N} = m - 100$ for all experiments, and the error values are computed using the efficient approximate classification schemes of Section 2.4.

4.1 Accuracy vs. approximation dimension

The first experiment investigates the effect of the dimension of the approximate kernel on classification accuracy on the test set. We set the dimension parameter s in Section 2.3 to values in the range $[2, 1024]$, with the eigenvalue threshold $\epsilon_d = 10^{-16}$. Note that s is an upper bound on the actual dimension d of approximation for ASSET$^{(*)}$, but is equal to d in the case of ASSET$^{(*)}_{on}$.

3. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
4. http://leon.bottou.org/papers/loosli-canu-bottou-2006/

Fig. 1. The effect of the approximation dimension d to the test error. The x-axis shows the values of s in log scale (base 2). For ASSET$_{on}$, $d = s$ and for the others $d \leq s$.



(a) ADULT (b) MNIST (c) CCAT (d) IJCNN (e) COVTYPE

For the batch tasks, we ran our algorithms for 1000 epochs ($1000m$ iterations) so that they converged to a near-optimal value with small variation among different randomization.

The CPSP and CPNY have a parameter similar to s (as an upper bound of d); we compared by setting that parameter to the same values as for s. We obtained the baselines of batch tasks by running SVM-Light. SVM-Light do not have dimension parameters but can be expected to give the best achievable performance by the kernel-approximate algorithms as s approaches m.

Figure 1 shows the results. Since ASSET and ASSET* yield very similar results in all experiments, we do not plot ASSET*. (For the same reason we show only ASSET$_{on}$ for online settings.) We would expect the codes to perform well when the underlying kernel is well approximated by a low-dimensional surrogate. When σ in (18) is very large, as in Figure 2(a) of ADULT dataset, all codes achieve good classification performance for small values of s. In other data sets, the chosen values of σ are smaller and the intrinsic dimension of the kernel is higher, so classification performance continues to improve as s increases. In particular, it is known that linear kernels work as well as nonlinear kernels on the CCAT. If linear kernels are optimal for CCAT,

TABLE 2
Training CPU time (s:seconds, h:hours) and test error in parentheses. Kernel approximation dimension is varied by setting $s = 512$ and $s = 1024$ for ASSET, ASSET*, CPSP and CPNY.

	Subgradient Methods		Cutting-plane		Decomposition	
$s = 512$	ASSET	ASSET*	CPSP	CPNY	LASVM	SVM-Light
ADULT	**22.5s** (15.06±0.06%)	23.9s (15.07±0.06%)	3020.0s (15.17%)	8.2h (15.13%)	1011.4s (18.02%)	856.8s (15.13%)
MNIST	**96.8s** (4.03±0.05%)	100.9s (4.03±0.04%)	549.6s (2.72%)	348.0s (4.07%)	587.5s (1.40%)	1322.6s (1.24%)
CCAT	95.0s (8.23±0.08%)	99.2s (8.26±0.06%)	799.9s (5.24%)	**62.0s** (8.31%)	2616.0s (4.71%)	3422.6s (4.72%)
IJCNN	**86.7s** (1.08±0.02%)	89.1s (1.08±0.02%)	726.8s (0.84%)	319.5s (1.1%)	288.1s (0.76%)	1331.3s (0.73%)
COVTYPE	697.2s (18.19±0.06%)	**585.7s** (18.18±0.07%)	1.8h (17.73%)	1841.5s (18.24%)	38.3h (13.46%)	52.7h (13.82%)
$s = 1024$	ASSET	ASSET*	CPSP	CPNY	LASVM	SVM-Light
ADULT	**77.6s** (15.10±0.05%)	83.2s (15.12±0.04%)	3398.5s (15.16%)	7.5h (15.17%)	1011.4s (18.02%)	856.8s (15.13%)
MNIST	**274.9s** (2.66±0.03%)	275.4s (2.67±0.02%)	1273.2s (2.03%)	515.4s (2.69%)	587.5s (1.40%)	1322.6s (1.24%)
CCAT	264.6s (7.09±0.05%)	278.4s (7.11±0.04%)	2949.9s (5.19%)	**122.9s** (7.15%)	2616.0s (4.71%)	3422.6s (4.72%)
IJCNN	307.1s (0.79±0.02%)	**297.0s** (0.79±0.01%)	1649.4s (0.78%)	598.0s (0.80%)	288.1s (0.76%)	1331.3s (0.73%)
COVTYPE	2259.4s (16.47±0.04%)	**2063.9s** (16.47±0.06%)	4.1h (16.61%)	3597.7s (16.52%)	38.3h (13.46%)	52.7h (13.82%)

the optimal Gaussian kernel may choose a very small value of σ producing near-identity thus high-rank Gram matrix. ASSET$_{on}$ seems to suffer from approximating the kernel function rather than the kernel matrix; the former is generally a more difficult problem. For a given dimension, the overall performance of ASSET$_{on}$ is worse than other methods, especially in the CCAT experiment.

CPSP generally requires lower dimension than the other methods to achieve the same classification performance. The power seems to come from the fact that CPSP spends extra time to construct good basis functions whereas the other methods depend on random sampling. However, all approximate-kernel methods including CPSP suffer considerably from the restriction in dimension for the COVTYPE task.

4.2 Speed of achieving similar test error

In performing timing comparisons, we ran all codes other than ours with their default stopping criteria. For ASSET and ASSET*, we checked the classification error on the test sets ten times per epoch, terminating when the error matched the performance of CPNY. (Since this code uses a similar Nyström approximation of the kernel, it is the one most directly comparable with ours in terms of classification accuracy.) The test error was measured using the iterate averaged over the 100 iterations immediately preceding each reporting point.

Results for the first five data sets are shown in Table 2 for the values $s = 512$ and $s = 1024$. (Note that LASVM and SVM-Light do not depend on s and so their results are the same in both tables.) The shortest time values to achieve similar test accuracy are marked as bold, showing that our methods are among the fastest in most cases. The best classification errors among the approximate codes are obtained by CPSP but the runtimes are considerably longer than for our methods. In fact, if we compare the performance of ASSET with $s = 1024$ and CPSP with $s = 512$, ASSET achieves similar test accuracy to CPSP (except for CCAT) but is faster by a factor between two and forty. CPNY requires an abnormally long run time on the ADULT dataset; we surmise that the code may be affected by numerical difficulties associated with the highly ill conditioned kernel for this problem.

Interestingly, ASSET shows similar performance to ASSET* despite the less impressive theoretical error bound of the former. When the value of regularization parameter λ is near zero, the objective function loses strong convexity and thereby breaks the condition required for ASSET* to work. We observe similar slowdown of Pegasos and SGD when λ approaches zero for linear kernel SVMs.

4.3 Online performance on very large data set

We take the final data set MNIST-E to be an online learning problem and compare the performance of ASSET$_{on}$ and ASSET$^*_{on}$ to the online SVM code LASVM. (Other algorithms such as CPSP, CPNY, and SVM-Light are less suitable for comparison because they operate in batch mode.) For a fair comparison, we fed the training samples to the algorithms in the same order.

Figure 2 shows the progress on a single run of our algorithms, with various approximation dimensions d in the range $[1024, 16384]$. Vertical bars in the graphs indicate the completion of training. ASSET$_{on}$ tends to converge faster and shows smaller test error values than ASSET$^*_{on}$, despite the theoretical slower convergence rate of the former. With $d = 16384$, ASSET$_{on}$ and ASSET$^*_{on}$ required 7.2 hours to finish with a solution of 2.7% and 3.5% test error, respectively. LASVM produced a better solution with only 0.2% test error, but it required 4.3 days of computation.

In Table 3, we represent the variability of solutions in the single runs of ASSET$_{on}$ and ASSET$^*_{on}$. We sampled ten test errors during the last 10000 iterations, evaluating the error values after each batch of 1000 iterations using the averaged solution over the 100 iterations immediately preceding each evaluation point. We also show the average time for the evaluations, and the total training time averaged over the single runs of ASSET$_{on}$ and ASSET$^*_{on}$ (these have negligible variations). The average error and the deviation of ASSET$_{on}$ up to $d = 8192$ tend to decrease as the dimension increases; one contributing

Fig. 2. Online progress of ASSET_{on} and ASSET^*_{on} to their completion (`MNIST-E`).

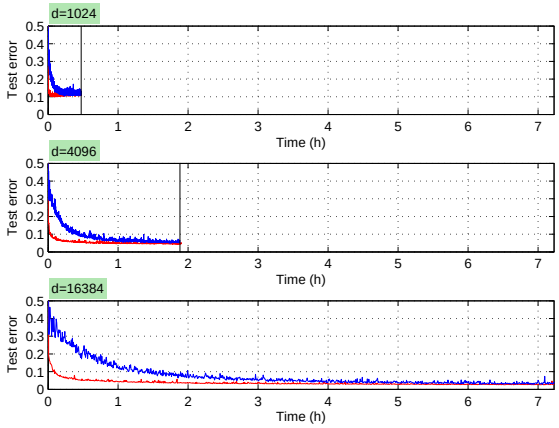


TABLE 3
Test error statistics (mean and standard deviation) for the last 10k iterations of online training (`MNIST-E`).

Dim. d	ASSET_{on}	ASSET^*_{on}	Avg. Time (s)	
			Train	Test
1024	11.5±0.56%	11.8±0.57%	1705	34
2048	7.2±0.47%	8.1±0.45%	3418	69
4096	4.8±0.33%	5.8±0.54%	6824	139
8192	3.6±0.30%	4.3±1.06%	13375	270
16384	3.0±0.63%	3.5±0.29%	26053	546

factor would be that the approximate feature mapping approaches to the true function in exponentially increasing probability with the dimension growth [19]. (This phenomenon was not as evident for ASSET^*_{on}.) We believe that the variability of stochastic subgradients eventually increases with dimension, leading to an increased variability in performance, as happens with $d = 16384$ for ASSET_{on}. Finally ASSET_{on} seems to produce more accurate classifier than ASSET^*_{on} with the same level of approximation.

The testing time depends on the controllable parameter d for our codes but it depends on the number of support vectors in the computed solution for LASVM (LASVM required 1504 seconds).

5 Conclusion

We have proposed a general framework for training support vector machines based on stochastic subgradients. Our algorithms can operate in batch and in online mode, and allows for the use of nonlinear kernels via kernel approximation and reformulation of the primal form. They do not require strong convexity. Our methods find solutions of reasonable quality for large problems, often in much shorter time than existing algorithms. Since the approaches require only (weak) convexity of the objective function, they can be extended easily to regression, ranking, and other learning problems.

6 Acknowledgements

The authors acknowledge the support of NSF Grants DMS-0914524 and DMS-0906818.

References

[1] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.
[2] L. Bottou. SGD: Stochastic gradient descent, 2005. http://leon.bottou.org/projects/sgd.
[3] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178, 2007.
[4] P. Drineas and M. W. Mahoney. On the nystrom method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.
[5] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training svm. *Journal of Machine Learning Research*, 6:1889–1918, 2005.
[6] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–164, 2001.
[7] V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 320–327, New York, NY, USA, 2008. ACM.
[8] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.
[9] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11. MIT Press, 1999.
[10] T. Joachims. Training linear SVMs in linear time. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 217–226, 2006.
[11] T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.
[12] T. Joachims and C.-N. J. Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2-3):179–193, 2009. Special Issue for European Conference on Machine Learning (ECML).
[13] D. D. Lewis, Y. Yang, T. G. Rose, G. Dietterich, F. Li, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.
[14] A. Nedic and D. P. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12:109–138, 2001.
[15] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.
[16] A. Nemirovski and D. B. Yudin. *Problem complexity and method efficiency in optimization*. John Wiley, 1983.
[17] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 12, pages 185–208. MIT Press, Cambridge, MA, 1999.
[18] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.
[19] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. MIT Press, Cambridge, MA, 2008.
[20] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.
[21] T. Serafini, G. Zanghirati, and L. Zanni. Gradient projection methods for large quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20:353–378, 2005.
[22] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML '07*, pages 807–814, New York, NY, USA, 2007. ACM.
[23] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML '03*, pages 928–936, 2003.