The Kernelized Stochastic Batch Perceptron

Andrew Cotter COTTER@TTIC.EDU
Toyota Technological Institute at Chicago 6045 S. Kenwood Ave., Chicago, IL 60637 USA

Shai Shalev-Shwartz SHAIS@CS.HUJI.AC.IL
John S. Cohen SL in CS, The Hebrew University of Jerusalem, Israel

Nathan Srebro NATI@TTIC.EDU
Toyota Technological Institute at Chicago 6045 S. Kenwood Ave., Chicago, IL 60637 USA

Abstract

We present a novel approach for training kernel Support Vector Machines, establish learning runtime guarantees for our method that are better then those of any other known kernelized SVM optimization approach, and show that our method works well in practice compared to existing alternatives.

1. Introduction

We present a novel algorithm for training kernel Support Vector Machines (SVMs). One may view a SVM as the bi-criterion optimization problem of seeking a predictor with large margin (low norm) on the one hand, and small training error on the other. Our approach is a stochastic gradient method on a non-standard scalarization of this bi-criterion problem. In particular, we use the "slack constrained" scalarized optimization problem introduced by Hazan et al. (2011) where we seek to maximize the classification margin, subject to a constraint on the total amount of "slack", i.e. sum of the violations of this margin. Our approach is based on an efficient method for computing unbiased gradient estimates on the objective. Our algorithm can be seen as a generalization of the "Batch Perceptron" to the non-separable case (i.e. when errors are allowed), made possible by introducing stochasticity, and we therefore refer to it as the "Stochastic Batch Perceptron" (SBP).

The SBP is fundamentally different from Pegasos (Shalev-Shwartz et al., 2011) and other stochastic gradient approaches to the problem of training SVMs, in that calculating each stochastic gradient estimate still requires considering the *entire* data set. In this regard, despite its stochasticity, the SBP is very much a "batch" rather than "online" algorithm. For a linear SVM, each iteration would require runtime linear in the training set size, resulting in an unacceptable overall runtime. However, in the kernel setting, essentially all known approaches already require linear runtime per iteration. A more careful analysis reveals the benefits of the SBP over previous kernel SVM optimization algorithms.

In order to compare the SBP runtime to the runtime of other SVM optimization algorithms, which typically work on different scalarizations of the bi-criterion problem, we follow Bottou & Bousquet (2008); Shalev-Shwartz & Srebro (2008) and compare the runtimes required to ensure a generalization error of $\mathcal{L}^* + \epsilon$, assuming the existence of some unknown predictor u with norm $\|u\|$ and expected hinge loss $\mathcal{L}^*$. The main advantage of the SBP is in the regime in which $\epsilon = \Omega(\mathcal{L}^*)$, i.e. we seek a constant factor approximation to the best achievable error (e.g. we would like an error of $1.01\mathcal{L}^*$). In this regime, the overall SBP runtime is $\|u\|^4/\epsilon$, compared with $\|u\|^4/\epsilon^3$ for Pegasos and $\|u\|^4/\epsilon^2$ for the best known dual decomposition approach.

2. Setup and Formulations

Training a SVM amounts to finding a vector w defining a classifier $x \mapsto \text{sign}(\langle w, \Phi(x)\rangle)$, that on the one hand has small norm (corresponding to a large classification margin), and on the other has a small training error, as measured through the average hinge loss on the training sample: $\hat{\mathcal{L}}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \langle w, \Phi(x_i)\rangle)$, where each (x_i, y_i) is a labeled example, and $\ell(a) = \max(0, 1-a)$ is the hinge loss. This is captured by

Appearing in *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

the following bi-criterion optimization problem:

$$\min_{w \in \mathbb{R}^d} \quad \|w\| \quad , \quad \hat{\mathcal{L}}(w). \tag{2.1}$$

We focus on *kernelized* SVMs, where the feature map $\Phi(x)$ is specified implicitly via a kernel $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$, and assume that $K(x, x') \leq 1$. We consider only "black box" access to the kernel (i.e. our methods work for any kernel, as long as we can compute $K(x, x')$ efficiently), and in our runtime analysis treat kernel evaluations as requiring $O(1)$ runtime. Since kernel evaluations dominate the runtime of all methods studied (ours as well as previous methods), one can also interpret the runtimes as indicating the number of required kernel evaluations. To simplify our derivation, we often discuss the explicit SVM, using $\Phi(x)$, and refer to the kernel only when needed.

A typical approach to the bi-criterion Problem 2.1 is to scalarize it using a parameter λ controlling the tradeoff between the norm (inverse margin) and the empirical error:

$$\min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i \langle w, \Phi(x_i) \rangle\right) \tag{2.2}$$

Different values of λ correspond to different Pareto optimal solutions of Problem 2.1, and the entire Pareto front can be explored by varying λ.

We instead consider the "slack constrained" scalarization (Hazan et al., 2011), where we maximize the "margin" subject to a constraint of ν on the total allowed "slack", corresponding to the average error. That is, we aim at maximizing the margin by which all points are correctly classified (i.e. the minimal distance between a point and the separating hyperplane), after allowing predictions to be corrected by a total amount specified by the slack constraint:

$$\max_{w \in \mathbb{R}^d} \max_{\xi \in \mathbb{R}^n} \min_{i \in \{1,\ldots,n\}} \left(y_i \langle w, \Phi(x_i) \rangle + \xi_i\right) \tag{2.3}$$

$$\text{subject to: } \|w\| \leq 1, \ \xi \succeq 0, \ \mathbf{1}^T \xi \leq n\nu$$

In this scalarization, varying ν explores different Pareto optimal solutions of Problem 2.1. This is captured by the following Lemma, which also quantifies how suboptimal solutions of the slack-constrained objective correspond to Pareto suboptimal points:

Lemma 2.1. (Hazan et al., 2011, Lemma 2.1) *For any $u \neq 0$, consider Problem 2.3 with $\nu = \hat{\mathcal{L}}(u) / \|u\|$. Let $\bar{w}$ be an $\bar{\epsilon}$-suboptimal solution to this problem with objective value γ, and consider the rescaled solution $w = \bar{w}/\gamma$. Then:*

$$\|w\| \leq \frac{1}{1 - \bar{\epsilon}\|u\|} \|u\| \quad , \quad \hat{\mathcal{L}}(w) \leq \frac{1}{1 - \bar{\epsilon}\|u\|} \hat{\mathcal{L}}(u)$$

3. The Stochastic Batch Perceptron

In this section, we will develop the Stochastic Batch Perceptron. We consider Problem 2.3 as optimization of the variable w with a single constraint $\|w\| \leq 1$, with the objective being to maximize:

$$f(w) = \max_{\xi \succeq 0, \mathbf{1}^T \xi \leq n\nu} \min_{p \in \Delta^n} \sum_{i=1}^{n} p_i \left(y_i \langle w, \Phi(x_i) \rangle + \xi_i\right) \tag{3.1}$$

Notice that we replaced the minimization over training indices i in Problem 2.3 with an equivalent minimization over the probability simplex, $\Delta^n = \{p \succeq 0 : \mathbf{1}^T p = 1\}$, and that we consider p and ξ to be a part of the objective, rather than optimization variables. The objective $f(w)$ is a concave function of w, and we are maximizing it over a convex constraint $\|w\| \leq 1$, and so this is a convex optimization problem in w.

Our approach will be to perform a stochastic gradient update on w at each iteration: take a step in the direction specified by an unbiased estimator of a (super)gradient of $f(w)$, and project back to $\|w\| \leq 1$. To this end, we will need to identify the (super)gradients of $f(w)$ and understand how to efficiently calculate unbiased estimates of them.

3.1. Warmup: The Separable Case

As a warmup, we first consider the separable case, where $\nu = 0$ and no errors are allowed. The objective is then:

$$f(w) = \min_i y_i \langle w, \Phi(x_i) \rangle, \tag{3.2}$$

This is simply the "margin" by which all points are correctly classified, i.e. γ s.t. $\forall_i \ y_i \langle w, \Phi(x_i) \rangle \geq \gamma$. We seek a linear predictor w with the largest possible margin. It is easy to see that (super)gradients with respect to w are given by $y_i \Phi(x_i)$ for any index i attaining the minimum in Equation 3.2, i.e. by the "most poorly classified" point(s). A gradient ascent approach would then be to iteratively find such a point, update $w \leftarrow w + \eta y_i \Phi(x_i)$, and project back to $\|w\| \leq 1$. This is akin to a "batch Perceptron" update, which at each iteration searches for a violating point and adds it to the predictor.

In the separable case, we could actually use *exact* supergradients of the objective. As we shall see, it is computationally beneficial in the non-separable case to base our steps on unbiased gradient estimates. We therefore refer to our method as the "Stochastic Batch Perceptron" (SBP), and view it as a generalization of the batch Perceptron which uses stochasticity and is applicable in the non-separable setting. In the same

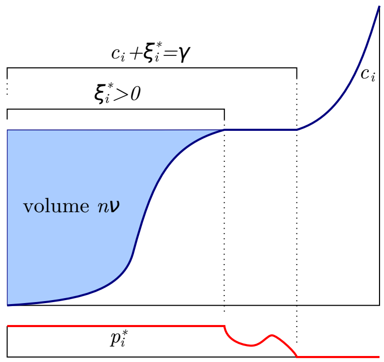

Figure 1. Illustration of how one finds ξ^* and p^*. The upper curve represents the values of the responses c_i, listed in order of increasing magnitude. The lower curve illustrates a minimax optimal probability distribution p^*.

way that the "batch Perceptron" can be used to maximize the margin in the separable case, the SBP can be used to obtain any SVM solution along the Pareto front of the bi-criterion Problem 2.1.

3.2. Supergradients of $f(w)$

For a fixed w, we define $c \in \mathbb{R}^n$ be the vector of "responses":

$$c_i = y_i \langle w, \Phi(x_i) \rangle \tag{3.3}$$

Supergradients of $f(w)$ at w can be characterized explicitly in terms of minimax-optimal pairs p^* and ξ^* such that $p^* = \arg\min_{p \in \Delta^n} p^t(c + \xi^*)$ and $\xi^* = \arg\max_{\xi \succeq 0, \mathbf{1}^T \xi \leq n\nu} (p^*)^T (c + \xi)$.

Lemma 3.1 (Proof in Appendix C)**.** *For any w, let p^*, ξ^* be minimax optimal for Equation 3.1. Then $\sum_{i=1}^n p_i^* y_i \Phi(x_i)$ is a supergradient of $f(w)$ at w.*

This suggests a simple method for obtaining unbiased estimates of supergradients of $f(w)$: sample a training index i with probability p_i^*, and take the stochastic supergradient to be $y_i \Phi(x_i)$. The only remaining question is how one finds a minimax optimal p^*.

It is possible to find a minimax optimal p^* in $O(n)$ time. For any ξ, a solution of $\min_{p \in \Delta^n} p^T (x + \xi)$ must put all of the probability mass on those indices i for which $c_i + \xi_i$ is minimized. Hence, an optimal ξ^* will maximize the minimal value of $c_i + \xi_i^*$. This is illustrated in Figure 1. The intuition is that the total mass $n\nu$ available to ξ is distributed among the indices as if this volume of water were poured into a basin with height c_i. The result is that the indices i with the lowest responses have columns of water above them such that the common surface level of the water is γ.

Once the "water level" γ has been determined, the optimal p^* must be uniform on those indices i for which $\xi_i^* > 0$, i.e. for which $c_i < \gamma$, must be zero on all i s.t. $c_i > \gamma$, and could take any intermediate value when $c_i = \gamma$ (that is, for some $q > 0$, we must have $c_i < \gamma \rightarrow p_i^* = q$, $c_i = \gamma \rightarrow 0 \leq p_i^* \leq q$, and $c_i > \gamma \rightarrow p_i^* = 0$—see Figure 1). In particular, the uniform distribution over all indices such that $c_i \leq \gamma$ is minimax optimal. Notice that in the separable case, where no slack is allowed, $\gamma = \min_i c_i$ and any distribution supported on the minimizing point(s) is minimax optimal, and $y_i \Phi(x_i)$ is an *exact* supergradient for such an i, as discussed in Section 3.1.

It is straightforward to find the water level γ in linear time once the responses c_i are sorted (as in Figure 1), i.e. with a total runtime of $O(n \log n)$ due to sorting. It is also possible to find the water level γ in linear time, without sorting the responses, using a divide-and-conquer algorithm, further of which may be found in Appendix B[1].

3.3. Kernelized Implementation

In a kernelized SVM, w is an element of an implicit space, and cannot be represented explicitly. We therefore represent w as $w = \sum_{i=1}^n \alpha_i y_i \Phi(x_i)$, and maintain not w itself, but instead the coefficients α_i. Our stochastic gradient estimates are always of the form $y_i \Phi(x_i)$ for an index i. Taking a step in this direction amounts to simply increasing the corresponding α_i.

We could calculate all the responses c_i at each iteration as $c_i = \sum_{j=1}^n \alpha_j y_i y_j K(x_i, x_j)$. However, this would require a quadratic number of kernel evaluations *per iteration*. Instead, as is typically done in kernelized SVM implementations, we keep the responses c_i on hand, and after each stochastic gradient step of the form $w \leftarrow w + \eta y_j \Phi(x_j)$, we update the responses as:

$$c_i \leftarrow c_i + \eta y_i y_j K(x_i, x_j) \tag{3.4}$$

This involves only n kernel evaluations per iteration.

In order to project w onto the unit ball, we must either track $\|w\|$ or calculate it from the responses as $\|w\| = \sum_{i=1}^n \alpha_i c_i$. Rescaling w so as to project it back into $\|w\| \leq 1$ is performed by rescaling all coefficients α_i and responses c_i, again taking time $O(n)$ and no additional kernel evaluations.

3.4. Putting it Together

We are now ready to summarize the SBP algorithm. Starting from $w^{(0)} = 0$ (so both $\alpha^{(0)}$ and all responses

[1] Appendices may be found in the long version of this paper, arXiv:1204.0566.

are zero), each iteration proceeds as follows:

1. Find p^* by finding the "water level" γ from the responses (Section 3.2), and taking p^* to be uniform on those indices for which $c_i \leq \gamma$.
2. Sample $j \sim p^*$.
3. Update $w^{(t+1)} \leftarrow \mathcal{P}\left(w^{(t)} + \eta_t y_j \Phi\left(x_j\right)\right)$, where $\mathcal{P}$ projects onto the unit ball and $\eta_t = \frac{1}{\sqrt{t}}$. This is done by first increasing $\alpha \leftarrow \alpha + \eta_t$ and updating the responses as in Equation 3.4, then calculating $\|w\|$ (Section 3.3) and scaling α and c by $\min(1, 1/\|w\|)$.

Updating the responses as in Equation 3.4 requires $O(n)$ kernel evaluations (the most computationally expensive part) and all other operations require $O(n)$ scalar arithmetic operations.

Since at each iteration we are just updating using an unbiased estimator of a supergradient, we can rely on the standard analysis of stochastic gradient descent to bound the suboptimality after T iterations:

Lemma 3.2 (Proof in Appendix C)**.** *For any $T, \delta > 0$, after T iterations of the Stochastic Batch Perceptron, with probability at least $1-\delta$, the average iterate $\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$ (corresponding to $\bar{\alpha} = \frac{1}{T}\sum_{t=1}^{T} \alpha^{(t)}$), satisfies: $f(\bar{w}) \geq \sup_{\|w\|\leq 1} f(w) - O\left(\sqrt{\frac{1}{T}\log\frac{1}{\delta}}\right)$.*

Since each iteration is dominated by n kernel evaluations, and thus takes linear time (we take a kernel evaluation to require $O(1)$ time), the overall runtime to achieve ϵ suboptimality for Problem 2.3 is $O(n/\epsilon^2)$.

3.5. Learning Runtime

The previous section has given us the runtime for obtaining a certain suboptimality of Problem 2.3. However, since the suboptimality in this objective is not directly comparable to the suboptimality of other scalarizations, e.g. Problem 2.2, we follow Bottou & Bousquet (2008); Shalev-Shwartz & Srebro (2008), and analyze the runtime required to achieve a desired generalization performance, instead of that to achieve a certain optimization accuracy on the empirical optimization problem.

Recall that our true learning objective is to find a predictor with low generalization error $\mathcal{L}_{0/1}(w) = \Pr_{(x,y)}\{y\langle w, \Phi(x)\rangle \leq 0\}$ with respect to some unknown distribution over x, y based on a training set drawn i.i.d. from this distribution. We assume that there exists some (unknown) predictor u that has norm $\|u\|$ and low expected hinge loss $\mathcal{L}^* = \mathcal{L}(u) = \mathbb{E}\left[\ell(y\langle u, \Phi(x)\rangle)\right]$ (otherwise, there is no point in training a SVM), and analyze the runtime to find a predictor w with generalization error $\mathcal{L}_{0/1}(w) \leq \mathcal{L}^* + \epsilon$.

In order to understand the SBP runtime, we must determine both the required sample size and optimization accuracy. Following Hazan et al. (2011), and based on the generalization guarantees of Srebro et al. (2010), using a sample of size:

$$n = \tilde{O}\left(\left(\frac{\mathcal{L}^* + \epsilon}{\epsilon}\right)\frac{\|u\|^2}{\epsilon}\right) \tag{3.5}$$

and optimizing the empirical SVM bi-criterion Problem 2.1 such that:

$$\|w\| \leq 2\|u\| \quad ; \quad \hat{\mathcal{L}}(w) - \hat{\mathcal{L}}(u) \leq \epsilon/2 \tag{3.6}$$

suffices to ensure $\mathcal{L}_{0/1}(w) \leq \mathcal{L}^* + \epsilon$ with high probability. Referring to Lemma 2.1, Equation 3.6 will be satisfied for $\bar{w}/\gamma$ as long as $\bar{w}$ optimizes the objective of Problem 2.3 to within:

$$\bar{\epsilon} = \frac{\epsilon/2}{\|u\|(\hat{\mathcal{L}}(u) + \epsilon/2)} \geq \Omega\left(\frac{\epsilon}{\|u\|(\hat{\mathcal{L}}(u) + \epsilon)}\right) \tag{3.7}$$

where the inequality holds with high probability for the sample size of Equation 3.5. Plugging this sample size and the optimization accuracy of Equation 3.7 into the SBP runtime of $O(n/\bar{\epsilon}^2)$ yields the overall runtime:

$$\tilde{O}\left(\left(\frac{\mathcal{L}^* + \epsilon}{\epsilon}\right)^3 \frac{\|u\|^4}{\epsilon}\right) \tag{3.8}$$

for the SBP to find $\bar{w}$ such that its rescaling satisfies $\mathcal{L}_{0/1}(w) \leq \mathcal{L}(u) + \epsilon$ with high probability.

In the realizable case, where $\mathcal{L}^* = 0$, or more generally when we would like to reach $\mathcal{L}^*$ to within a small constant multiplicative factor, we have $\epsilon = \Omega(\mathcal{L}^*)$, the first factor in Equation 3.8 is a constant, and the runtime simplifies to $\tilde{O}(\|u\|^4/\epsilon)$. As we will see in Section 4, this is a better guarantee than that enjoyed by any other SVM optimization approach.

3.6. Including an Unregularized Bias

It is possible to use the SBP to train SVMs with a bias term, i.e. where one seeks a predictor of the form $x \mapsto (\langle w, \Phi(x)\rangle + b)$. We then take stochastic gradient steps on:

$$f(w) = \tag{3.9}$$
$$\max_{\substack{b\in\mathbb{R},\xi\succeq 0 \\ \mathbf{1}^T\xi\leq n\nu}} \min_{p\in\Delta^n} \sum_{i=1}^{n} p_i\left(y_i\langle w, \Phi(x_i)\rangle + y_i b + \xi_i\right)$$

Lemma 3.1 still holds, but we must now find minimax optimal p^*,ξ^* and b^*. This can be accomplished

Table 1. Upper bounds, up to log factors, on the runtime (number of kernel evaluations) required to achieve $\mathcal{L}_{0/1}(w) \leq \mathcal{L}(u) + \epsilon$.

	Overall	$\epsilon = \Omega\left(\mathcal{L}\left(u\right)\right)$
SBP	$\left(\frac{\mathcal{L}(u)+\epsilon}{\epsilon}\right)^3 \frac{\|u\|^4}{\epsilon}$	$\frac{\|u\|^4}{\epsilon}$
Dual Decomp.	$\left(\frac{\mathcal{L}(u)+\epsilon}{\epsilon}\right)^2 \frac{\|u\|^4}{\epsilon^2}$	$\frac{\|u\|^4}{\epsilon^2}$
SGD on $\hat{\mathcal{L}}$	$\left(\frac{\mathcal{L}(u)+\epsilon}{\epsilon}\right) \frac{\|u\|^4}{\epsilon^3}$	$\frac{\|u\|^4}{\epsilon^3}$

using a modified "water filling" involving two basins, one containing the positively-classified examples, and the other the negatively-classified ones. As in the case without an unregularized bias, this can be accomplished in $O(n)$ time—see Appendix B for details.

4. Relationship to Other Methods

We discuss the relationship between the SBP and several other SVM optimization approaches, highlighting similarities and key differences, and comparing their performance guarantees.

4.1. SIMBA

Recently, Hazan et al. (2011) presented SIMBA, a method for training *linear* SVMs based on the same "slack constrained" scalarization (Problem 2.3) we use here. SIMBA also fully optimizes over the slack variables ξ at each iteration, but differs in that, instead of fully optimizing over the distribution p (as the SBP does), SIMBA updates p using a stochastic mirror descent step. The predictor w is then updated, as in the SBP, using a random example drawn according to p. A SBP iteration is thus in a sense more "thorough" then a SIMBA iteration. The SBP theoretical guarantee (Lemma 3.2) is correspondingly better by a logarithmic factor (compare to Hazan et al. (2011, Theorem 4.3)). All else being equal, we would prefer performing a SBP iteration over a SIMBA iteration.

For linear SVMs, a SIMBA iteration can be performed in time $O(n+d)$. However, fully optimizing p as described in Section 3.2 requires the responses c_i, and calculating or updating all n responses would require time $O(nd)$. In this setting, therefore, a SIMBA iteration is much more efficient than a SBP iteration.

In the kernel setting, calculating even a single response requires $O(n)$ kernel evaluation, which is the same cost as updating *all* responses after a change to a single coordinate α_i (Section 3.3). This makes the responses essentially "free", and gives an advantage to methods such as the SBP (and the dual decomposition methods discussed below) which make use of the responses.

Although SIMBA is preferable for linear SVMs, the SBP is preferable for kernelized SVMs. It should also be noted that SIMBA relies heavily on having direct access to features, and that it is therefore not obvious how to apply it directly in the kernel setting.

4.2. Pegasos and SGD on $\hat{\mathcal{L}}(w)$

Pegasos (Shalev-Shwartz et al., 2011) is a SGD method optimizing the regularized scalarization of Problem 2.2. Alternatively, one can perform SGD on $\hat{\mathcal{L}}(w)$ subject to the constraint that $\|w\| \leq B$, yielding similar learning guarantees (e.g. (Zhang, 2004)). At each iteration, these algorithms pick an example uniformly at random from the training set. If the margin constraint is violated on the example, w is updated by adding to it a scaled version of $y_i\Phi(x_i)$. Then, w is scaled and possibly projected back to $\|w\| \leq B$. The actual update performed at each iteration is thus very similar to that of the SBP. The main difference is that in Pegasos and related SGD approaches, examples are picked uniformly at random, unlike the SBP which samples from the set of violating examples.

In a linear SVM, where $\Phi(x_i) \in \mathbb{R}^d$ are given explicitly, each Pegasos (or SGD on $\hat{\mathcal{L}}(w)$) iteration is extremely simple and requires runtime which is linear in the dimensionality of $\Phi(x_i)$. A SBP update would require calculating and referring to all $O(n)$ responses. However, with access only to kernel evaluations, even a Pegasos-type update requires either considering all support vectors, or alternatively updating all responses, and might also take $O(n)$ time, just like the much "smarter" SBP step.

To understand the learning runtime of such methods in the kernel setting, recall that SGD converges to an ϵ-accurate solution of the optimization problem after at most $\|u\|^2/\epsilon^2$ iterations. Therefore, the overall runtime is $n\|u\|^2/\epsilon^2$. Combining this with Equation 3.5 yields that the runtime requires by SGD to achieve a learning accuracy of ϵ is $\tilde{O}\left(((\mathcal{L}^* + \epsilon)/\epsilon)\|u\|^4/\epsilon^3\right)$. When $\epsilon = \Omega(\mathcal{L}^*)$, this scales as $1/\epsilon^3$ compared with the $1/\epsilon$ scaling for the SBP (see also Table 1).

4.3. Dual Decomposition Methods

Many of the most popular packages for optimizing kernel SVMs, including LIBSVM (Chang & Lin, 2001) and SVM-Light (Joachims, 1998), use dual-decomposition approaches. This family of algorithms works on the dual of the scalarization 2.2, given by:

$$\max_{\alpha \in \left[0, \frac{1}{\lambda n}\right]^n} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad (4.1)$$

and proceed by iteratively choosing a small working set of dual variables α_i, and then optimizing over these variables while holding all other dual variables fixed. At an extreme, SMO (Platt, 1998) uses a working set of the smallest possible size (two in problems with an unregularized bias, one in problems without). Most dual decomposition approaches rely on having access to all the responses c_i (as in the SBP), and employ some heuristic to select variables α_i that are likely to enable a significant increase in the dual objective.

On an objective without an unregularized bias the structure of SMO is similar to the SBP: the responses c_i are used to choose a single point j in the training set, then α_j is updated, and finally the responses are updated accordingly. There are two important differences, though: how the training example to update is chosen, and how the change in α_j is performed.

SMO updates α_j so as to exactly optimize the *dual* Problem 4.1, while the SBP takes a step along α_j so as to improve the *primal* Problem 2.3. Dual feasibility is *not* maintained, so the SBP has more freedom to use large coefficients on a few support vectors, potentially resulting in sparser solutions.

The use of heuristics to choose the training example to update makes SMO very difficult to analyze. Although it is known to converge linearly after some number of iterations (Chen et al., 2006), the number of iterations required to reach this phase can be very large (see a detailed discussion in Appendix E). To the best of our knowledge, the most satisfying analysis for a dual decomposition method is the one given in Hush et al. (2006). In terms of learning runtime, this analysis yields a runtime of $\tilde{O}\left(\left(\left(\mathcal{L}(u)+\epsilon\right)/\epsilon\right)^2 \|u\|^4 / \epsilon^2\right)$ to guarantee $\mathcal{L}_{0/1}(w) \leq \mathcal{L}(u) + \epsilon$. When $\epsilon = \Omega(L^*)$, this runtime scales as $1/\epsilon^2$, compared with the $1/\epsilon$ guarantee for the SBP.

4.4. Stochastic Dual Coordinate Ascent

Another variant of the dual decomposition approach is to choose a single α_i randomly at each iteration and update it so as to optimize Equation 4.1 (Hsieh et al., 2008). The advantage here is that we do not need to use all of the responses at each iteration, so that if it is easy to calculate responses on-demand, as in the case of linear SVMs, each SDCA iteration can be calculated in time $O(d)$ (Hsieh et al., 2008). In a sense, SDCA relates to SMO in a similar fashion that Pegasos relates to the SBP: SDCA and Pegasos are preferable on linear SVMs since they choose working points at random; SMO and the SBP choose working points based on more information (namely, the responses), which are unnecessarily expensive to compute in the linear case, but, as discussed earlier, are essentially "free" in kernelized implementations. Pegasos and the SBP both work on the primal (though on different scalarizations), while SMO and SDCA work on the dual and maintain dual feasibility.

The current best analysis of the runtime of SDCA is not satisfying, and yields the bound $n/\lambda\epsilon$ on the number of iterations, which is a factor of n larger than the bound for Pegasos. Since the cost of each iteration is the same, this yields a significantly worse guarantee. We do not know if a better guarantee can be derived for SDCA. See a detailed discussion in Appendix E.

4.5. The Online Perceptron

We have so far considered only the problem of optimizing the bi-criterion SVM objective of Problem 2.1. However, because the online Perceptron achieves the same form of learning guarantee (despite not optimizing the bi-criterion objective), it is reasonable to consider it, as well.

The online Perceptron makes a *single* pass over the training set. At each iteration, if w errs on the point under consideration (i.e. $y_i \langle w, \Phi(x_i) \rangle \leq 0$), then $y_i \Phi(x_i)$ is added into w. Let M be the number of mistakes made by the Perceptron on the sequence of examples. Support vectors are added only when a mistake is made, and so each iteration of the Perceptron involves at most M kernel evaluations. The total runtime is therefore Mn.

While the Perceptron is an online learning algorithm, it can also be used for obtaining guarantees on the generalization error using an online-to-batch conversion (e.g. (Cesa-Bianchi et al., 2001)).

From a bound on the number of mistakes M (e.g. Shalev-Shwartz (2007, Corollary 5)), it is possible to show that the expected number of mistakes the Perceptron makes is upper bounded by $n\mathcal{L}(u) + \|u\| \sqrt{n\mathcal{L}(u)} + \|u\|^2$. This implies that the total runtime required by the Perceptron to achieve $\mathcal{L}_{0/1}(w) \leq \mathcal{L}(u) + \epsilon$ is $O\left(\left(\left(\mathcal{L}(u)+\epsilon\right)/\epsilon\right)^3 \|u\|^4 / \epsilon\right)$. This is of the same order as the bound we have derived for SBP. However, the Perceptron does *not* converge to a Pareto optimal solution to the bi-criterion Problem 2.1, and therefore cannot be considered a SVM optimization procedure. Furthermore, the online Perceptron generalization analysis relies on an "online-to-batch" conversion technique (e.g. (Cesa-Bianchi et al., 2001)), and is therefore valid only for a *single* pass over the data. If we attempt to run the Perceptron for multiple passes, then it might begin to overfit uncontrollably. Although the worst-case theoretical guarantee obtained after a single pass is indeed similar to that for an optimum of the SVM objective, in practice an optimum of the empirical SVM optimization problem does seem to have significantly better generalization performance.

5. Experiments

We compared the SBP to other SVM optimization approaches on the datasets in Table 2. We compared to Pegasos (Shalev-Shwartz et al., 2011), SDCA (Hsieh et al., 2008), and SMO (Platt, 1998) with a second order heuristic for working point selection (Fan et al., 2005). These approaches work on the regularized formulation of Problem 2.2 or its dual (Problem 4.1). To enable comparison, the parameter ν for the SBP was derived from λ as $\|\hat{w}^*\| \nu = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i \left\langle w^*, \Phi\left(x_i\right)\right\rangle\right)$, where $\hat{w}^*$ is the known (to us) optimum.

We first compared the methods on a SVM formulation *without* an unregularized bias, since Pegasos and SDCA do not naturally handle one. So that this comparison would be implementation-independent, we measure performance in terms of the number of kernel evaluations. As can be seen in Figure 2, the SBP outperforms Pegasos and SDCA, as predicted by the upper bounds. The SMO algorithm has a dramatically different performance profile, in line with the known analysis: it makes relatively little progress, in terms of generalization error, until it reaches a certain critical point, after which it converges rapidly. Unlike the other methods, terminating SMO early in order to obtain a cruder solution does not appear to be advisable.

We also compared to the online Perceptron algorithm. Although use of the Perceptron is justified for non-separable data only if run for a single pass over the training set, we did continue running for multiple passes. The Perceptron's generalization performance is similar to that of the SBP for the first epoch, but the SBP continues improving over additional passes. As discussed in Section 4.5, the Perceptron is unsafe and might overfit after the first epoch, an effect which is clearly visible on the Adult dataset.

To give a sense of actual runtime, we compared our implementation of the SBP[2] to the SVM package LIBSVM, running on an Intel E7500 processor. We allowed an unregularized bias (since that is what LIBSVM uses), and used the parameters in Table 2. For these experiments, we replaced the Reuters dataset with the version of the Forest dataset used by Nguyen et al. (2010), using their parameters. LIBSVM converged to a solution with 14.9% error in 195s on Adult, 0.44% in 1980s on MNIST, and 1.8% in 35 hours on Forest. In *one-quarter* of each of these runtimes, SBP obtained 15.0% error on Adult, 0.46% on MNIST, and 1.6% on Forest. These results of course depend heavily on the specific stopping criterion used.

[2]Source code is available from `http://ttic.uchicago.edu/~cotter/projects/SBP`

6. Summary and Discussion

The Stochastic Batch Perceptron is a novel approach for training kernelized SVMs. The SBP fares well empirically, and, as summarized in Table 1, our runtime guarantee for the SBP is the best of any existing guarantee for kernelized SVM training. An interesting open question is whether this runtime is optimal, i.e. whether any algorithm relying only on black-box kernel accesses must perform $\Omega\left(\left(\left(\mathcal{L}^* + \epsilon\right)/\epsilon\right)^3 \|u\|^4 / \epsilon\right)$ kernel evaluations.

As with other stochastic gradient methods, deciding when to terminate SBP optimization is an open issue. The most practical approach seems to be to terminate when a holdout error stabilizes. We should note that even for methods where the duality gap can be used (e.g. SMO), this criterion is often too strict, and the use of cruder criteria may improve training time.

Acknowledgements: S. Shalev-Shwartz is supported by the Israeli Science Foundation grant number 590-10.

References

Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L., and Tarjan, R. E. Time bounds for selection. *JCSS*, 7(4): 448–461, August 1973.

Bottou, L. and Bousquet, O. The tradeoffs of large scale learning. In *NIPS'08*, pp. 161–168, 2008.

Cesa-Bianchi, N., Conconi, A., and Gentile, C. On the generalization ability of on-line learning algorithms. *IEEE Trans. on Inf. Theory*, 50:2050–2057, 2001.

Chang, C-C. and Lin, C-J. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

Collins, M., Globerson, A., Koo, T., Carreras, X., and Bartlett, P. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *JMLR*, 9:1775–1822, 2008.

Table 2. Datasets, downloaded from `http://leon.bottou.org/projects/lasvm`, and parameters used in the experiments. In our experiments, we used the Gaussian kernel with bandwidth σ.

			Without unreg. bias			With unreg. bias		
Data set	Training size n	Testing size	σ^2	λ	ν	σ^2	λ	ν
Reuters money_fx	7770	3229	0.5	$1/n$	6.34×10^{-4}			
Adult	31562	16282	10	$1/n$	1.10×10^{-2}	100	$1/100n$	5.79×10^{-4}
MNIST "8" vs. rest	60000	10000	25	$1/n$	2.21×10^{-4}	25	$1/1000n$	6.42×10^{-11}
Forest	522910	58102				5000	$1/10000n$	7.62×10^{-10}

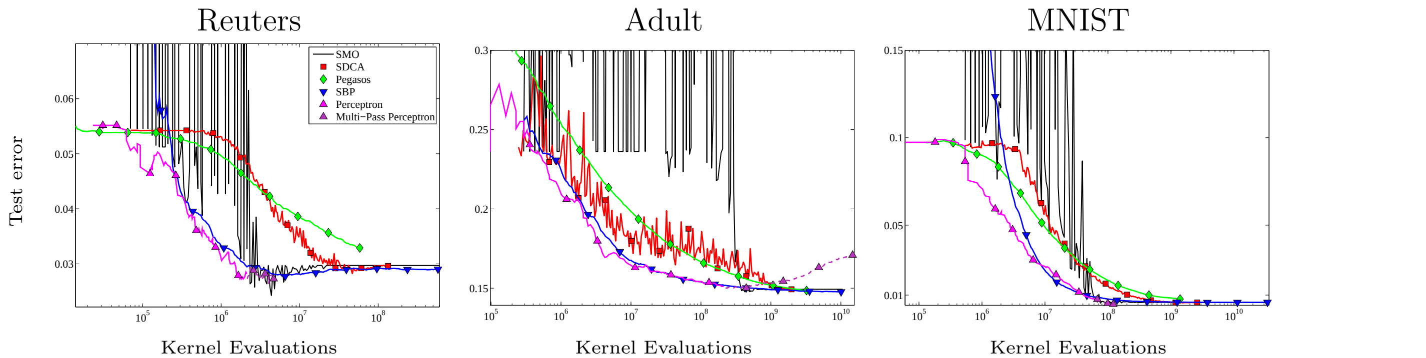

Figure 2. Classification error on the held-out testing set (linear scale) vs. the number of kernel evaluations performed during optimization (log scale), averaged over ten runs. The Perceptron was run for multiple passes over the data—its curve becomes dashed after the first epoch (n iterations). All algorithms were run for ten epochs, *except* for Perceptron on Adult, which we ran for 100 epochs to better illustrate its overfitting.

Fan, R-E., Chen, P-S., and Lin, C-J. Working set selection using second order information for training support vector machines. *JMLR*, 6:1889–1918, 2005.

Hazan, E., Koren, T., and Srebro, N. Beating SGD: Learning SVMs in sublinear time. In *NIPS'11*, 2011.

Hsieh, C-J., Chang, K-W., Lin, C-J., Keerthi, S. S., and Sundararajan, S. A dual coordinate descent method for large-scale linear SVM. In *ICML'08*, pp. 408–415, 2008.

Hush, D., Kelly, P., Scovel, C., and Steinwart, I. QP algorithms with guaranteed accuracy and run time for support vector machines. *JMLR*, 7:733–769, 2006.

Joachims, T. Making large-scale support vector machine learning practical. In Schölkopf, B., Burges, C., and Smola, A. J. (eds.), *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

Kakade, S. M. and Tewari, A. On the generalization ability of online strongly convex programming algorithms. In *NIPS'09*, 2009.

Nguyen, D D, Matsumoto, K., Takishima, Y., and Hashimoto, K. Condensed vector machines: learning fast machine for large data. *Trans. Neur. Netw.*, 21(12): 1903–1914, Dec 2010.

Platt, J. C. Fast training of support vector machines using Sequential Minimal Optimization. In Schölkopf, B., Burges, C., and Smola, A. J. (eds.), *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *NIPS'07*, 2007.

Scovel, C., Hush, D., and Steinwart, I. Approximate duality. *JOTA*, 2008.

Shalev-Shwartz, S. *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, The Hebrew University of Jerusalem, July 2007.

Shalev-Shwartz, S. and Srebro, N. SVM optimization: Inverse dependence on training set size. In *ICML'08*, pp. 928–935, 2008.

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

Srebro, N., Sridharan, K., and Tewari, A. Smoothness, low-noise and fast rates. In *NIPS'10*, 2010.

Zhang, T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML'04*, 2004.

Zinkevich, M. Online convex programming and generalized infinitesimal gradient ascent. In *ICML'03*, 2003.