Guarantees for Approximate Incremental SVMs

Nicolas Usunier
LIP6, Université Paris 6,
Paris, France
nicolas.usunier@lip6.fr

Antoine Bordes
LIP6, Université Paris 6,
Paris, France
antoine.bordes@lip6.fr

Léon Bottou
NEC Laboratories America,
Princeton, USA
leon@bottou.org

Abstract

Assume a teacher provides examples $(\mathbf{x}_t, y_t)$ one by one. An approximate incremental SVM computes a sequence of classifiers that are close to the true SVM solutions computed on the successive incremental training sets. We show that simple algorithms can satisfy an averaged accuracy criterion with a computational cost that scales as well as the best SVM algorithms with the number of examples. Finally, we exhibit some experiments highlighting the benefits of joining fast incremental optimization and curriculum and active learning [Schohn and Cohn, 2000, Bordes et al., 2005, Bengio et al., 2009].

1 Introduction

It has been observed that some form of *curriculum learning* [Bengio et al., 2009] like *selective sampling* can reduce the computational costs required to achieve a predefined generalization error [Schohn and Cohn, 2000, Bordes et al., 2005, Dasgupta and Hsu, 2008]. Such procedures involve a (*teacher*, *learner*) pair, where the teacher provides examples $(\mathbf{x}_t, y_t)$ one by one, and the learner computes a sequence of classifiers representing at each instant the set $S_t = \{(\mathbf{x}_i, y_i), i \leq t\}$ of the examples observed so far. When the learning objective is convex, it is desirable that the sequence of classifiers is close to the sequence of optimal solutions. It becomes a requirement when the quality of the teacher is affected by the quality of the learner, like in selective sampling. Then, in order to be successful from the computational point of view, these procedures need a learner that is computationally efficient and exhibits strong approximation guarantees.

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 6 of JMLR: W&CP 6. Copyright 2010 by the authors.

In this paper, we consider approximate incremental SVM algorithms that strive to track the sequence of optima with a predefined tolerance. Our analysis shows that adequately designed algorithms can track the successive optima by performing a *constant number of iterations for each additional example*. This translates into a total number of iteration growing linearly with the number of examples, comparable to the best *batch* algorithms computing approximate SVM solutions [Joachims, 2006, Shalev-Shwartz et al., 2007] which require the train set to be known beforehand.

The bound we obtain holds regardless of the quality of the teacher, making the algorithms suitable for any online learning setup, even with adversarial teachers, or when learning with a curriculum like selective sampling. In a second part of the paper, we discuss, on the basis of experimental results, why an efficient incremental procedure together with selective sampling can dramatically improve the learning curve.

These results corroborate the extensive study on fast SVM optimization carried out by the family of *online SVMs* proposed by Bordes et al. [2005, 2007, 2008]. In fact, the algorithms we study are variations on these algorithms, simplified for the sake of the theoretical analysis. Thus, as a by-product, our work leads to a new interpretation of these online SVMs as incremental approximate optimizers, and explains some of their empirical behavior.

Despite its importance in the context of large scale learning, approximate incremental algorithms that efficiently *track* the sequence of optima have, to the best of our knowledge, never been studied in the literature. Cauwenberghs and Poggio [2001] describe an incremental algorithm for SVM that performs rank one updates of the SVM coefficients until reaching the new optimal solution. In contrast to their approach, the algorithm we propose do not try to converge to the optimum at each time index, but guarantee that a predefined accuracy is maintained *on average* on the sequence of batch optima. Although we accept a slightly weaker approximation guarantee, it comes

together with a considerable benefit in terms of computational costs. The algorithms we propose are also similar to some online algorithms studied in [Shalev-Shwartz and Singer, 2007]. Yet our analysis is completely different, since the quality of these online algorithms was measured in terms of *regret* rather than in terms of approximate optimization guarantee.

In the rest of the paper, we first describe our SVM setup, and give a new result on the duality gap. Then we present and analyze two approximate incremental SVM algorithms, and exhibit some experiments that illustrate the theoretical results. Finally, we discuss the impact of joining selective sampling and fast incremental optimization on the computational cost.

2 Setup

We consider a stream of examples $(\mathbf{x}_i, y_i)_{i\geq 1}$ with patterns $\mathbf{x}_i$ verifying $\|\mathbf{x}_i\| \leq 1$ and with labels $y_i = \pm 1$. We consider discriminant functions of the form $\hat{y}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ where the notation $\langle \cdot, \cdot \rangle$ designates a suitable dot product in pattern space.

Let $P_t(\mathbf{w})$ be the primal cost function restricted to the set S_t containing the first t examples,

$$P_t(\mathbf{w}) \stackrel{\triangle}{=} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{t}[1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle]_+ \qquad (1)$$

with notation $[z]_+ \stackrel{\triangle}{=} \max\{z, 0\}$ and let $D_t(\boldsymbol{\alpha})$ be the associated dual objective function

$$D_t(\boldsymbol{\alpha}) \stackrel{\triangle}{=} \sum_{i=1}^{t}\alpha_i - \frac{1}{2}\sum_{i,j\leq t} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \qquad (2)$$

with $\boldsymbol{\alpha} = \{\alpha_1 \dots \alpha_t\} \in [0, C]^t$.

If $\boldsymbol{\alpha}^*$ maximizes D_t, it is well known that $\mathbf{w}(\boldsymbol{\alpha}^*) \stackrel{\triangle}{=} \sum_{i=1}^{t}\alpha_i^* y_i \mathbf{x}_i$ minimizes P_t, and

$$D_t^* \stackrel{\triangle}{=} D_t(\boldsymbol{\alpha}^*) \stackrel{\triangle}{=} \max_{\boldsymbol{\alpha}\in[0,C]^t} D_t(\boldsymbol{\alpha}) = \min_{\mathbf{w}} P_t(\mathbf{w}) = P_t(\mathbf{w}(\boldsymbol{\alpha}^*)) \stackrel{\triangle}{=} P_t^*$$

Dual coordinate ascent is a simple procedure to maximize D_t. Let $(\mathbf{e}_1 \dots \mathbf{e}_t)$ be the canonical basis of $\mathbb{R}^t$. Starting from a dual parameter vector $\boldsymbol{\alpha}^k \in [0, C]^t$, each dual coordinate ascent iteration picks a search direction $\mathbf{e}_{\sigma(k)}$ and outputs a new dual parameter vector $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + a^* \mathbf{e}_{\sigma(k)}$ with a^* chosen to maximize $D(\boldsymbol{\alpha}^{k+1})$ subject to $\boldsymbol{\alpha}^{k+1} \in [0, C]^t$. A simple derivation shows that, with $g_i(\boldsymbol{\alpha}) \stackrel{\triangle}{=} 1 - y_i \langle \mathbf{w}(\boldsymbol{\alpha}), \mathbf{x}_i \rangle$,

$$a^* = \max\left(-\alpha_{\sigma(k)}^k, \min\left(C - \alpha_{\sigma(k)}^k, \frac{g_{\sigma(k)}(\boldsymbol{\alpha}^k)}{\|\mathbf{x}_{\sigma(k)}\|^2}\right)\right). \qquad (3)$$

An approximate minimizer of the primal cost D can therefore be obtained by choosing a suitable starting value $\boldsymbol{\alpha}^0$, performing an adequate number K of successive dual coordinate ascent iterations and outputting $\mathbf{w}(\boldsymbol{\alpha}^K)$. The convergence and the efficiency of this procedure depends on the *scheduling policy* used to chose the successive search direction $\mathbf{e}_{\sigma(k)}$ at each step. A simple policy, which we will discuss in the paper, is to chose the direction randomly.

3 Duality Lemma

The following lemma is interesting because it connects the two quantities of interest: the gap, which measures the accuracy of the solution, and the expected effect of the next coordinate ascent iteration.

Lemma 1 *Let $t \geq 1$, $\max_{i=1..t}\|\mathbf{x}_i\| \leq 1$, and $\boldsymbol{\alpha} \in [0, C]^t$. Then:*

$$\frac{P_t(\mathbf{w}(\boldsymbol{\alpha})) - D_t(\boldsymbol{\alpha})}{Ct} \leq \mu\left(\mathop{\mathbb{E}}_{i\sim U(t)} \Delta_{t,i}(\boldsymbol{\alpha})\right)$$

where $\mu(x) = \sqrt{2x} + x/C$, $U(t)$ denotes the uniform distribution over $\{1...t\}$, and

$$\Delta_{t,i}(\boldsymbol{\alpha}) \stackrel{\triangle}{=} \max_{a\in[-\alpha_i, C-\alpha_i]} [D_t(\boldsymbol{\alpha} + a\mathbf{e}_i) - D_t(\boldsymbol{\alpha})]$$

A bound on the gap is of course a bound on both the primal $P_t(\mathbf{w}(\boldsymbol{\alpha})) - P_t^*$ and dual $D_t(\boldsymbol{\alpha}) - D^*$ suboptimalities. The left hand side denominator Ct makes sense because it normalizes the loss in the expression of the primal (1).

Proof: The result follows from elementary arguments regarding $\Delta_{t,i}(\boldsymbol{\alpha})$ and the duality gap

$$\begin{aligned} G(\boldsymbol{\alpha}) &\stackrel{\triangle}{=} P_t(\mathbf{w}(\boldsymbol{\alpha})) - D_t(\boldsymbol{\alpha}) \\ &= \|\mathbf{w}(\boldsymbol{\alpha})\|^2 + C\textstyle\sum_{i=1}^{t}[g_i(\boldsymbol{\alpha})]_+ - \sum_{i=1}^{t}\alpha_i \end{aligned}$$

Recalling $\|\mathbf{w}(\boldsymbol{\alpha})\|^2 = \sum_{i=1}^{t} y_i \alpha_i \langle \mathbf{w}(\boldsymbol{\alpha}), \mathbf{x}_i \rangle = \sum_{i=1}^{t}\alpha_i\big(1 - g_i(\boldsymbol{\alpha})\big)$, we obtain the identity

$$G(\boldsymbol{\alpha}) = \sum_{i=1}^{t}\max\left[(C - \alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right]. \qquad (4)$$

We now turn our attention to quantity $\Delta_{t,i}(\boldsymbol{\alpha})$. Equation (3) shows that a^* has always the same sign as $g_i(\boldsymbol{\alpha})$ and $|a^*| \leq |g_i|/\|\mathbf{x}_i\|^2$. Since $D_t(\boldsymbol{\alpha} + a\mathbf{e}_i) - D_t(\boldsymbol{\alpha}) = a\left(g_i(\boldsymbol{\alpha}) - a/2\,\|\mathbf{x}_i\|^2\right)$,

$$\frac{1}{2}|a^*||g_i(\boldsymbol{\alpha})| \leq \Delta_{t,i}(\boldsymbol{\alpha}) = |a^*||g_i(\boldsymbol{\alpha})| - \frac{1}{2}\|\mathbf{x}_i\|^2 |a^*|^2. \qquad (5)$$

To use this result in equation (4), we fix some index i and consider two cases:

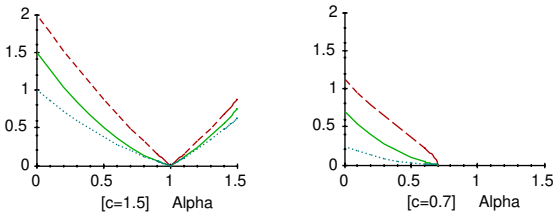

Figure 1: Duality lemma with a single example $\mathbf{x}_1 = 1$, $y_1 = 1$. The figures compare the gap $P_t - D_t$ (continuous blue curve), the bound (dashed red curve), and the primal suboptimality $P_t - P_t^*$ (dotted cyan curve) as a function of α_1. The left plot shows a free support vector ($C = 1.5$, $\alpha_1^* = 1$). The right plot shows a support vector at bound ($C = 0.7$, $\alpha_1^* = C$).

1. If $|a^*| \geq |g_i(\boldsymbol{\alpha})|$, then, using equation (5), we have $|g_i(\boldsymbol{\alpha})| \leq \sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})}$, and thus

$$C|g_i(\boldsymbol{\alpha})| \geq \max\left[(C-\alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right],$$

$$\text{and} \quad C|g_i(\boldsymbol{\alpha})| \leq C\sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})} \leq C\mu(\Delta_{t,i}(\boldsymbol{\alpha})).$$

2. If $|a^*| < |g_i(\boldsymbol{\alpha})|$, then, given (3) and the assumption $\|\mathbf{x}_i\|^2 \leq 1$, $\alpha_i + a^*$ has necessarily reached a bound. Since a^* and $g_i(\boldsymbol{\alpha})$ have the same sign, it means that if $g_i(\boldsymbol{\alpha}) \leq 0$, then $a^* = -\alpha_i$, and $a^* = C - \alpha_i$ otherwise. This implies

$$\begin{aligned}\max\left[(C-\alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right] &= |a^*||g_i(\boldsymbol{\alpha})| \\ &= \Delta_{t,i}(\boldsymbol{\alpha}) + \tfrac{1}{2}\|\mathbf{x}_i\|^2 |a^*|^2 \quad .\end{aligned}$$

In order to obtain a bound involving only $\Delta_{t,i}(\boldsymbol{\alpha})$, we need to bound the last term of the last equation. Since we are in the case $|a^*| < |g_i(\boldsymbol{\alpha})|$, the left-hand side of equation (5) gives us $|a^*| \leq \sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})}$. Moreover, since $\|\mathbf{x}_i\|^2 \leq 1$, we have $\frac{1}{2}\|\mathbf{x}_i\|^2 |a^*| \leq \frac{1}{2}C$ and

$$\begin{aligned}&\max\left[(C-\alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right] \\ &\quad\leq \Delta_{t,i}(\boldsymbol{\alpha}) + \tfrac{1}{2}C\sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})} \\ &\quad\leq C\mu(\Delta_{t,i}(\boldsymbol{\alpha})).\end{aligned}$$

Putting points 1 and 2 in equation (4), and using the concavity of μ, we obtain the desired result:

$$\begin{aligned}\frac{P_t(\mathbf{w}(\boldsymbol{\alpha})) - D_t(\boldsymbol{\alpha})}{Ct} &\leq \frac{1}{t}\sum_{i=1}^t \mu(\Delta_{t,i}(\boldsymbol{\alpha})) \\ &\leq \mu\left(\mathbb{E}_{i\sim U(t)}\,\Delta_{t,i}(\boldsymbol{\alpha})\right). \qquad \square\end{aligned}$$

To ascertain the quality of this bound, consider how a SVM with a single scalar example $\mathbf{x}_1 = 1$, $y_1 = 1$ illustrates the two cases of the proof. The left plot in figure 1 shows a situation where the optimum uses the example as a free support vector, that is, case 1 in the proof. The lack of a vertical tangent near the optimum $\alpha_1 = 1$ confirms the square root behavior of $\mu(x)$ when x approaches zero. The right plot shows a situation of a bounded support vector, that is, case 2 in the proof. The bound is much looser when α_i approaches C. However this is less important because coordinate ascent iterations usually set such coefficient to C at once. The bound is then exact.

4 Algorithms and Analysis

4.1 The Analysis Technique

Let us illustrate the analysis technique on the dual coordinate ascent algorithm outlined in section 2 running on a fixed training set with t examples. Assume the successive search directions are picked randomly. We can easily copy the collapsing sum method of Shalev-Shwartz and Singer [2007].

Let $\mathcal{F}_k$ represent all the successive search directions $\mathbf{e}_{\sigma(i)}$ for $i < k$. For all k, we can rewrite lemma 1 as,

$$\frac{P_t(\mathbf{w}(\boldsymbol{\alpha}^k)) - D_t(\boldsymbol{\alpha}^k)}{Ct} \leq \mu\left(\mathbb{E}\left[D_t(\boldsymbol{\alpha}^{k+1}) - D_t(\boldsymbol{\alpha}^k)\middle|\,\mathcal{F}_k\right]\right).$$

Taking the expectation, averaging over all k, and using twice Jensen's inequality,

$$\begin{aligned}&\frac{1}{K}\sum_{k=1}^K \mathbb{E}\left[\frac{P_t(\mathbf{w}(\boldsymbol{\alpha}^k)) - D_t(\boldsymbol{\alpha}^k)}{Ct}\right] \\ &\leq \frac{1}{K}\sum_{k=1}^K \mathbb{E}\left[\mu\left(\mathbb{E}\left[D_t(\boldsymbol{\alpha}^{k+1}) - D_t(\boldsymbol{\alpha}^k)\middle|\,\mathcal{F}_k\right]\right)\right] \\ &\leq \mu\left(\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^K D_t(\boldsymbol{\alpha}^{k+1}) - D_t(\boldsymbol{\alpha}^k)\right]\right) \\ &\leq \mu\left(\frac{\mathbb{E}\left[D_t(\boldsymbol{\alpha}^{K+1}) - D_t(\boldsymbol{\alpha}^1)\right]}{K}\right) \quad \leq \quad \mu\left(\frac{\mathbb{E}\,D_t^*}{K}\right).\end{aligned}$$

Since the gap bounds both the primal and dual suboptimality, we obtain a dual convergence bound

$$\begin{aligned}\mathbb{E}\left[\frac{D_t^* - D_t(\boldsymbol{\alpha}^K)}{Ct}\right] &\leq \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^K \frac{D_t^* - D_t(\boldsymbol{\alpha}^k)}{Ct}\right] \\ &\leq \mu\left(\frac{\mathbb{E}\,D_t^*}{K}\right),\end{aligned}$$

and a somehow less attractive primal convergence bound

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^K \frac{P_t(\mathbf{w}(\boldsymbol{\alpha}^k)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\,D_t^*}{K}\right).$$

These bounds are different because each iteration increases the value of the dual objective, but does not necessarily reduce the value of the primal cost. However it is easy to obtain a nicer primal convergence bound by considering an averaged algorithm. Let $\bar{\boldsymbol{\alpha}}^K \triangleq \frac{1}{K}\sum_{k=1}^K \boldsymbol{\alpha}_k$. Thanks to the convexity of the primal cost, we can write

$$\mathbb{E}\left[\frac{P_t(\mathbf{w}(\bar{\boldsymbol{\alpha}}^K)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\,D_t^*}{K}\right).$$

In practice, this averaging operation is dubious because it ruins the sparsity of the dual parameter vector $\boldsymbol{\alpha}$. However it yields bounds that are easier to interpret, albeit not fundamentally different.

4.2 Tracking Inequality for a Simple Algorithm

We now return to our incremental setup. A teacher provides a new example $(\mathbf{x}_t, y_t)$ at each time step. We seek to compute a sequence of classifiers $\mathbf{w}_t$ that tracks $\min_{\mathbf{w}} P_t(\mathbf{w})$ with a predefined accuracy.

Algorithm 1 Simple Averaged Tracking Algorithm

1: **input**: stream of examples $(\mathbf{x}_i, y_i)_{i\geq 1}$, number of iterations $K \geq 1$ at each time index.
2: **initialization**: $\forall i, \alpha_i \leftarrow 0$, $t \leftarrow 1$
3: **while** exists$(\mathbf{x}_t, y_t)$ **do**
4: **let** $\alpha_t \leftarrow 0$
5: **for** $k = 1$ **to** K **do**
6: **pick** i randomly in $\{1, ..., t\}$
7: **let** $\alpha_i \leftarrow \alpha_i + \max\left(-\alpha_i, \min\left(C - \alpha_i, \frac{g_i(\boldsymbol{\alpha})}{\|\mathbf{x}_i\|^2}\right)\right)$
8: **let** $\bar{\boldsymbol{\alpha}}^t \leftarrow \frac{k-1}{k}\bar{\boldsymbol{\alpha}}^t + \frac{1}{k}\boldsymbol{\alpha}$
9: **end for**
10: **output** classifier $\mathbf{w}^t = \mathbf{w}(\bar{\boldsymbol{\alpha}}^t)$
11: $t \leftarrow t + 1$
12: **end while**

After receiving each new example $(\mathbf{x}_t, y_t)$, Algorithm 1 performs a predefined number K of dual coordinate ascent iterations on randomly picked coefficients associated with the currently known examples (line 7). The parameter vector given as output $\mathbf{w}(\bar{\boldsymbol{\alpha}}^t)$ is the average of the parameter vectors on these K iterations (lines 8 and 10). Notice that this averaging only affects the output vector, and not the parameters used by the algorithm to perform the updates. The algorithm is thus as fast as if it did not contain the averaging step. Moreover, the averaging only concerns the last K iterations, so that it does not has much effect on the sparsity of the solution as soon as $K << t$.

Theorem 2 *Let $\mathbf{w}(\bar{\boldsymbol{\alpha}}^t)$ be the sequence of classifiers output by algorithm 1. Assume furthermore that $\max_t \|\mathbf{x}_t\| \leq 1$. Then, for any $T \geq 1$, we have*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\frac{P_t(\mathbf{w}(\bar{\boldsymbol{\alpha}}^t)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\, D_T^*}{KT}\right)$$

where $\mu(x) = \sqrt{2x} + \frac{x}{C}$. Moreover, the number of dual coordinate ascent performed by the algorithm after seeing T examples is exactly KT.

Theorem 2 does not bound the primal suboptimality at each time index. However, since all these primal suboptimalities are positive, the theorem guarantees that an upper bound of the excess misclassification error, $(P_t - P_t^*)/Ct$, will be bounded *on average*. This weaker guarantee comes with a possibly huge computational benefit as it saves the computation of a stopping criterion. It is true that computing the duality gap can be cheap if all the gradients $g_i(\boldsymbol{\alpha})$ are cached. However, such a strategy is only relevant with non-linear kernels, and even then, careful and efficient implementations (e.g. [Bordes et al., 2007]) might choose to maintain a cache for a fraction of the dual coefficients, making the computation of a criterion likely to be expensive.

The proof of the theorem follows the schema of section 4.1: setup the collapsing sum of dual objective values; apply the lemma; use Jensen's inequality to distribute the function μ and the expectations on each term and regroup the terms on the left hand side.

Expectations in the theorem can have two interpretations. In the simplest setup, the teacher fixes the sequence of examples before the execution of the algorithm. Expectations are then taken solely with respect to the successive random choices of coordinate ascent directions. In a more general setup, the teacher follows an unspecified causal policy. At each time index t, he can use past values of the algorithm variables to choose the next example $(\mathbf{x}_t, y_t)$. The sequence of examples becomes a random variable. Expectations are then taken with respect to both the random search directions and the sequence of examples. The proof is identical in both cases.

4.3 Tracking Inequality for a Process/Reprocess Algorithm

Algorithm 2 is inspired by the PROCESS/REPROCESS idea of Bordes et al. [2005]. Before performing K dual coordinate ascent iterations on coefficients associated with examples randomly picked among the currently known examples (the REPROCESS operation), this algorithm performs an additional iteration on the coefficient associated with the new example (the PROCESS operation, compare lines 4 in both algorithms).

Theorem 3 *Let $\mathbf{w}(\bar{\boldsymbol{\alpha}}^t)$ be the sequence of classifiers output by algorithm 2. Let $\boldsymbol{\alpha}^t$ denote the successive value taken by variable $\boldsymbol{\alpha}$ before each execution of line 4 of algorithm 2. Assume furthermore that $\max_t \|\mathbf{x}_t\| \leq 1$. Then, for any $T \geq 1$, we have*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\frac{P_t(\mathbf{w}(\bar{\boldsymbol{\alpha}}^t)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\left[D_T^* - \delta_T\right]}{KT}\right)$$

where $\mu(x) = \sqrt{2x} + \frac{x}{C}$ and $\delta_T = \sum_{t=1}^{T}\Delta_{t,t}(\boldsymbol{\alpha}^t)$ is the cumulative dual increase during the PROCESS operations. Moreover, the number of elementary optimization steps performed by the algorithm after seeing t examples is exactly $(K+1)t$.

Algorithm 2 Averaged Tracking Algorithm with PROCESS/REPROCESS

1: **input**: stream of examples $(\mathbf{x}_i, y_i)_{i\geq 1}$, number of iterations $K \geq 1$ at each time index.
2: **initialization**: $\forall i, \alpha_i \leftarrow 0$, $t \leftarrow 1$
3: **while** exists$(\mathbf{x}_t, y_t)$ **do**
4: **let** $\alpha_t \leftarrow \max\left(0, \min\left(C, \frac{g_t(\boldsymbol{\alpha})}{\|\mathbf{x}_t\|^2}\right)\right)$
5: **for** $k = 1$ **to** K **do**
6: **pick** i randomly in $\{1, ..., t\}$
7: **let** $\alpha_i \leftarrow \alpha_i + \max\left(-\alpha_i, \min\left(C - \alpha_i, \frac{g_i(\boldsymbol{\alpha})}{\|\mathbf{x}_i\|^2}\right)\right)$
8: **let** $\bar{\boldsymbol{\alpha}}^t \leftarrow \frac{k-1}{k}\bar{\boldsymbol{\alpha}}^t + \frac{1}{k}\boldsymbol{\alpha}$
9: **end for**
10: **output** classifier $\mathbf{w}^t = \mathbf{w}(\bar{\boldsymbol{\alpha}}^t)$
11: $t \leftarrow t + 1$
12: **end while**

The proof of the theorem is similar to the proof of theorem 2 except that terms of the collapsing sum corresponding to the PROCESS operations (line 4 in the algorithm) are collected in quantity δ_T.

Adding this PROCESS operation gives the bound $\mu\left(\mathbb{E}\left[D_T^* - \bar{\delta}_T\right]/KT\right)$ instead of $\mu\left(\mathbb{E}\left[D_T^*\right]/KT\right)$. Since the quantity δ_T is related to the online loss incurred by the online algorithm [Shalev-Shwartz and Singer, 2007], $\delta_T = \Omega(T)$ unless the all training examples received after a given time index are separable. Under this condition, the PROCESS operation saves a multiplicative factor on the number K of REPROCESS operations necessary to reach a predefined accuracy. Although we cannot give a precise value for δ_T, we can claim that Algorithm 2 should perform significantly better than Algorithm 1 in practice.

4.4 Rough Comparisons

Since $D_T^* - \delta_T \leq D_T^* \leq P_T(\mathbf{0}) = CT$ the following corollary can be derived from the theorems.

Corollary 4 *Under the assumptions of theorem 2 and 3, let $4C \geq \epsilon \geq 0$. When $K = \left\lceil \frac{8C}{\epsilon^2} \right\rceil$, both algorithms 1 and 2 satisfy $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \frac{P_t(\mathbf{w}(\bar{\boldsymbol{\alpha}}^t)) - P_t^*}{Ct}\right] \leq \epsilon$.*

The total number n of iterations therefore scales like CT/ϵ^2 where T is the number of examples. Since the cost of each iteration depends on details of the algorithm (see section 5), let us assume, as a first approximation, that the cost of each iteration is proportional to either the number of support vectors, or, in the case of linear kernels, on the effective dimension of the patterns. The results reported in this contribution are then comparable to the bounds reported in [Joachims, 2006, Franc and Sonnenburg, 2008]. Improved bounds for generic bundle methods [Smola et al., 2008] are difficult to compare because their successive iterations solve increasingly complex optimization problems. Bounds for stochastic gradient algorithms [Shalev-Shwartz et al., 2007] also scale linearly with the number T of examples[1] but offer better scaling in $1/\epsilon$.

5 Reducing the Computational Cost

Corollary 4 gives a bound on the number of dual coordinate ascent steps the algorithm should perform to maintain, on average, a predefined accuracy. But, it also provides an estimate of the learning time: computing a gradient $g_i(\boldsymbol{\alpha})$ costs, at most, as many kernel computations as the number of support vectors, which itself grows linearly with t when the data is not separable. This translates in a time complexity of $O(Ct^2/\epsilon^2)$ to train on t examples. We now discuss how caching strategies can reduce this cost.

5.1 Detecting Useless Ascent Directions

Algorithms 1 and 2 select dual coordinate ascent directions randomly. As a consequence, most of their iterations have no effect because the selected coefficient α_i cannot be improved. This happens when $\alpha_i = 0\,,\ g_i(\boldsymbol{\alpha}) \leq 0$ or when $\alpha_i = C\,,\ g_i(\boldsymbol{\alpha}) \geq 0$.

In practice, a vast majority of ascent directions may actually do nothing. Given a training set of t training examples, let $n_0(\boldsymbol{\alpha})$ and $n_C(\boldsymbol{\alpha})$ be the number of examples falling into these two cases. These numbers approach $n_0(\boldsymbol{\alpha}_t^*)$ and $n_C(\boldsymbol{\alpha}_t^*)$ when $\boldsymbol{\alpha}$ approaches the SVM solution $\boldsymbol{\alpha}_t^*$. Provided that C decreases with an appropriate rate to ensure consistency, Steinwart [2004] has famously proved that the total number of support vectors $t - n_0(\boldsymbol{\alpha}_t^*)$ scales linearly with the total number t of examples. But his work also shows that the number of support vectors is dominated by the number $n_C(\boldsymbol{\alpha}_t^*)$ of margins violators. Therefore the fraction $\left(n_0(\boldsymbol{\alpha}) + n_C(\boldsymbol{\alpha})\right)/t$ of useless coordinate ascent iterations tends to 1 when t increases and the algorithm converges.

To avoid spending time in computing gradient values of useless ascent directions, the traditional solution in SVM solvers (see [Bottou and Lin, 2007] for a review) consists in allocating new variables g_i that always contain quantity $g_i(\boldsymbol{\alpha})$. It is then immediate to check whether a direction can improve the objective function or not. Whenever a coefficient of $\boldsymbol{\alpha}$ changes, updating all the g_i variables requires a time proportional to the total number t of examples. Since this only happens when an actual update takes place, the amortized cost of a coordinate ascent iteration is then proportional to $t - n_0(\boldsymbol{\alpha}) - n_C(\boldsymbol{\alpha})$ which grows slower than t. This is

[1] Shalev-Shwartz et al. report a bound in $\tilde{O}\left(\frac{1}{\lambda\epsilon}\right)$. Their λ is $1/CT$ in our setup.

Dataset	Train. Ex.	Test. Ex.	Features	C	$k(x, \bar{x})$	Cache
ADULT	32,561	16,281	123	1	$e^{-0.005\|x-\bar{x}\|^2}$	512MB
MNIST (8vsAll)	60,000	10,000	780	1000	$e^{-0.005\|x-\bar{x}\|^2}$	512MB

Table 1: Dataset and parameters used for the experiments.

usually significantly smaller than the worst-case cost of $t - n_0(\boldsymbol{\alpha})$ per iteration.

5.2 Tracking Inequalities for LASVM Algorithms

The previous discussion suggests that maintaining the gradient cache actually dominates the amortized cost of an iteration. A large part of this cost is due to the $n_0(\boldsymbol{\alpha})$ examples that are correctly classified. It is reasonable to assume that most of them will remain correctly classified on future iterations. Updating their corresponding cached gradients is thus useless.

Consider the variant of Algorithm 2 with the following modifications:

i) we maintain variables g_i representing $g_i(\boldsymbol{\alpha})$ for only those i such that $\alpha_i > 0$,
ii) we shortcut line 7 in Algorithm 2 whenever $\alpha_i = 0$ or $\alpha_i = C\ ,\ g_i \geq 0$.

This modified algorithm can be viewed as a randomized variant of the LASVM family of algorithms discussed by Bordes et al. [2005, 2007, 2008]. Updating the g_i is now proportional to the number of support vectors instead of the total number of examples. This brings very positive effects on the memory requirements of the kernel cache [Bordes et al., 2005].

The results in section 4.3 do not directly apply to these algorithms, because the ascent directions are not selected randomly anymore. However, we can still use lemma 1 at any time step, considering that the algorithm chooses its ascent directions randomly, on its current set of support vectors. The complete analysis would be rather involved technically, but one could then interpret algorithms of the LASVM family as approximately optimizing the SVM objective on multiple example sets that change in a solution-dependent manner. In practice, it appears that the shortcut (*ii*) does not discard too many examples that should be in the final support vector expansion. This explains why these algorithms tend to give solutions close to the optimum after only a single pass on the dataset.

6 Experiments

In this section, we intend to empirically display the tracking behaviour of our fast approximate incremental optimizers. Besides, we exhibit some results demonstrating the interest of using such methods, in particular when dealing with selective sampling.

We compare several setups of Algorithm 1 and 2. We first vary the parameter K fixing the number of dual coordinate ascent iterations. We also confront two ways for the teacher to provide new examples to the learner, randomly or using selective sampling. Following Bordes et al. [2005], our selective sampling strategy employs an *active selection* criterion which chooses the example that comes closest to the decision boundary of the current classifier. In practice, at each round t, our implementation first randomly picks a pool of 50 unseen training examples and then, among those, selects the one for which $|\langle \mathbf{w}(\bar{\boldsymbol{\alpha}}^t), \mathbf{x} \rangle|$ is minimal. We finally compare Algorithm 1 with its batch counterpart that consists in: (i) loading the whole dataset in memory and (ii) repeatedly performing updates using equation (3) on randomly picked examples.

Our experiments were conducted on two benchmarks for classification: ADULT[2] and MNIST[3]. MNIST is a multiclass dataset but we restricted our work to the sub-task of classifying the "8" digit against the others. Implementations of all algorithms employ a cache of 512MB for kernel values. All other parameters come from the literature and are displayed in Table 1. All reported results were obtained after an averaging over ten runs on the randomly shuffled training sets.

6.1 Tracking the Sequence of Optima

Figure 2 displays the duality gap normalized by both C and the number of training examples for algorithms 1 and 2 and for various values of K. These curves clearly shows that the normalized gap keeps a constant value on average during the whole pass over the training set: both algorithms successfully track the successive optima. The influence of the increase K is twofold. This decreases the mean value of the gap and reduces its fluctuations. Indeed, the gap can increase drastically if the algorithm picks a nasty example, but, as such examples are rare, the gap decreases to its mean value in subsequent iterations. This is the reason why theorems 2 and 3 do not bound the gap but its mean value at each iteration.

[2] Available from the UCI repository.
[3] Available at `http://yann.lecun.com/exdb/mnist/`.

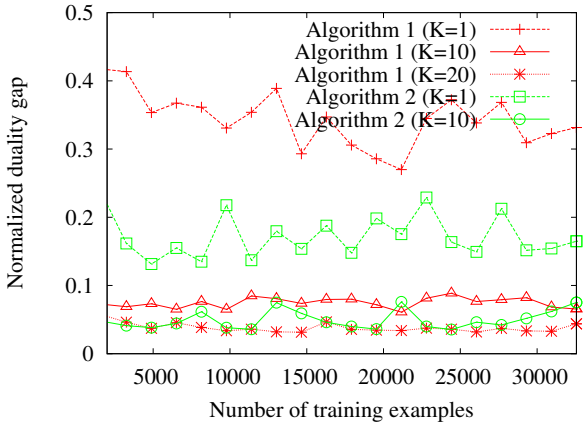

Figure 2: Normalized duality gap for various values of K on the course of training on ADULT.

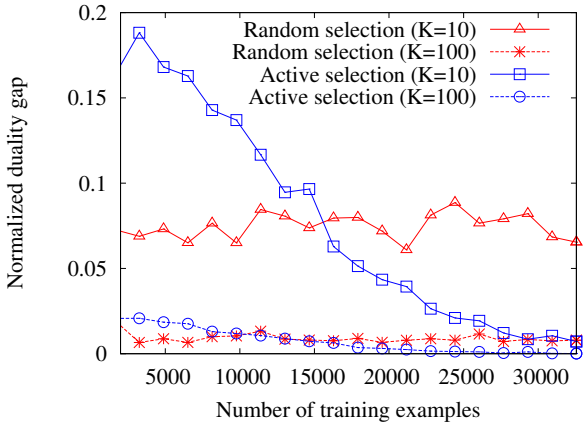

Figure 3: Normalized duality gap for two example selection strategies for Algorithm 1 on ADULT.

A comparison between Algorithm 1 (red) and 2 (green) on figure 2 also supports the observation of section 4.3 stating that an algorithm using process/reprocess should perform better in practice. Hence, we observe that Algorithm 2 with $K = 10$ performs nearly as well as Algorithm 1 with $K = 20$.

Figure 3 confirms the very little assumptions about the teacher's policy of our theorems by showing that, when combined with active example selection, Algorithm 1 is still able to track the optimum on the course of training. Indeed, its normalized gap does not grow. On the contrary, the curriculum induced by the active selection even causes it to decrease regularly. This happens because the very beginning of the curriculum first proposes very informative bits that rapidly increase the dual and primal costs. Bounds of both theorems manage this phenomenon via the term $\mathbb{E}\, D_T^*/T$.

Figure 4 depicts the mean value taken by the normalized gap of Algorithm 1 according to eight values of the K parameter. The fitting curve tends to express a dependency of K in $\frac{1}{\epsilon}$ and not in $\frac{1}{\epsilon^2}$ as stated in corollary 4. This suggests that, in practice, optimization might be much better than our theoretical results and tracking algorithms as fast as batch optimizers.

6.2 Benefits of a Good Curriculum

Previous work on curriculum learning [Bengio et al., 2009] and active selection [Schohn and Cohn, 2000, Bordes et al., 2005, Dasgupta and Hsu, 2008] indicate that using a good teacher's policy allows to reach good generalization accuracies with less training examples and less kernel computations. We now confirm that for the special case of Algorithm 1 with the figures 5 and 6 which display the test error according to the number of training examples and kernel computations.

Both figures demonstrate the interest of using accurate online optimization algorithms. Combined with an active selection strategy, Algorithm 1 provides a fair solution early during training. As a consequence, its intermediate solutions soon perform almost as well as the optimal solution and thus require much less kernels computations, memory usage and training duration to be refined. It is worth noting that the more accurate the incremental algorithm is the higher the benefits are: using $K = 100$ leads to better performances in terms of strong generalization abilities and low computational requirements than using $K = 10$.

6.3 Batch Optimization

Figure 6 also demonstrates that the test error decreases much faster for Algorithm 1 than for its batch counterpart, independently of the example selection. This could be surprising because both algorithms use the same update. However, at the beginning of train-

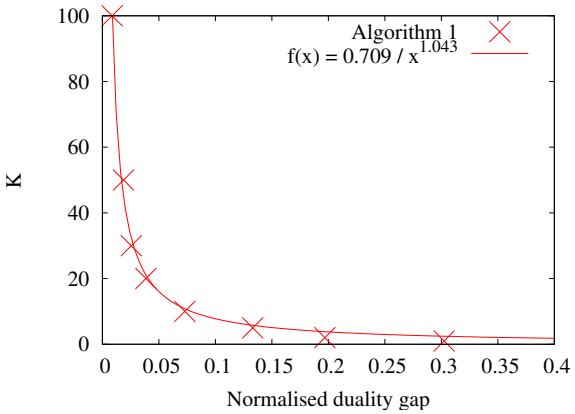

Figure 4: K parameter against the resulting mean normalized gap value for Algorithm 1 on ADULT.

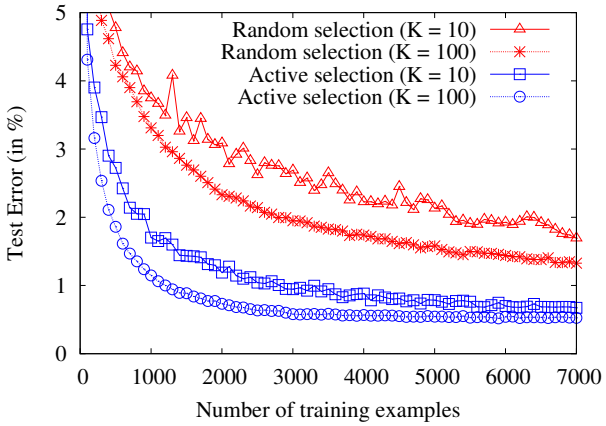

Figure 5: Test error for two example selection strategies for the first iterations of Algorithm 1 on MNIST.

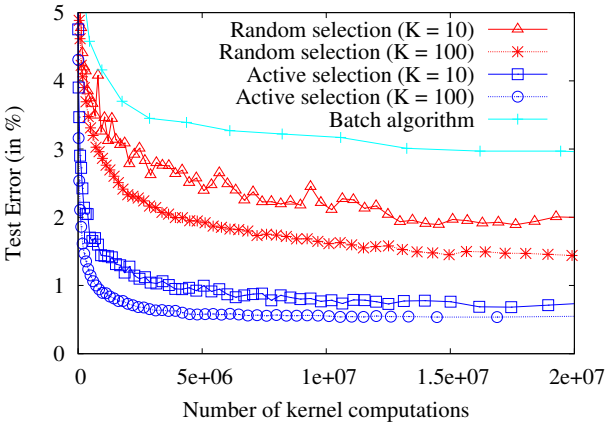

Figure 6: Test error against number of kernel computations for several setups of Algorithm 1 on MNIST.

ing, the incremental algorithm optimizes on successive working sets of relatively small sizes where the batch algorithm has immediately access to the whole dataset. Hence, Algorithm 1 has a much higher probability to pick the same example several times than its batch counterpart sketched in Section 2 and thus usually gathers less support vectors. This translates in far less computations overall to reach similar accuracies. This effect is greatly magnified by active selection.

7 Conclusion

Leveraging a novel duality lemma, we have presented tracking guarantees for approximate incremental SVMs that compare with results about batch SVMs. We have also introduced two algorithms and displayed some experiments allowing to illustrate the theoretical results and highlight some of the benefits of fast incremental optimization, in particular when joined with curriculum and active learning.

Acknowledgments

Part of this work was funded by NSF grant ???-????? and by the Network of Excellence PASCAL2. Antoine Bordes was also supported by the French DGA.

References

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proc. of the 26th International Conference on Machine Learning*. Omnipress, 2009.

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, 2005.

A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with larank. In *Proc. of the 24th International Machine Learning Conference*. OmniPress, 2007.

A. Bordes, N. Usunier, and L. Bottou. Sequence labelling SVMs trained in one pass. In *Proc. of ECML PKDD 2008*, pages 146–161. Springer, 2008.

L. Bottou and C.-J. Lin. Support vector machine solvers. In *Large Scale Kernel Machines*, pages 301–320. MIT Press, 2007.

G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Processing Systems*. MIT Press, 2001.

S. Dasgupta and D. Hsu. Hierarchical sampling for active learning. In *Proc. of the 25th International Conference on Machine Learning*. Omnipress, 2008.

V. Franc and S. Sonnenburg. OCAS optimized cutting plane algorithm for support vector machines. In *Proc of the 25th International Conference on Machine Learning*. Omnipress, 2008.

T. Joachims. Training linear SVMs in linear time. In *Proc. of the 12th ACM SIGKDD International Conference*, 2006.

G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proc. of the 17th International Conference on Machine Learning*, pages 839–846. Morgan Kaufmann, 2000.

S. Shalev-Shwartz and Y. Singer. A primal-dual perspective of online learning algorithms. *Machine Learning*, 69 (2-3):115–142, 2007.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proc. of the 24th International Conference on Machine Learning*, pages 807–814. ACM, 2007.

A. Smola, S.V.N. Vishwanathan, and Q. Le. Bundle methods for machine learning. In *Advances in Neural Information Processing Systems*, pages 1377–1384. MIT Press, 2008.

I. Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.