Mach Learn
DOI 10.1007/s10994-007-5003-0

Tracking the best hyperplane with a simple budget Perceptron

Giovanni Cavallanti · Nicolò Cesa-Bianchi · Claudio Gentile

Received: 7 September 2006 / Revised: 15 December 2006 / Accepted: 21 December 2006

Abstract Shifting bounds for on-line classification algorithms ensure good performance on any sequence of examples that is well predicted by a sequence of changing classifiers. When proving shifting bounds for kernel-based classifiers, one also faces the problem of storing a number of support vectors that can grow unboundedly, unless an eviction policy is used to keep this number under control. In this paper, we show that shifting and on-line learning on a budget can be combined surprisingly well. First, we introduce and analyze a shifting Perceptron algorithm achieving the best known shifting bounds while using an unlimited budget. Second, we show that by applying to the Perceptron algorithm the simplest possible eviction policy, which discards a random support vector each time a new one comes in, we achieve a shifting bound close to the one we obtained with no budget restrictions. More importantly, we show that our randomized algorithm strikes the optimal trade-off $U = \Theta(\sqrt{B})$ between budget B and norm U of the largest classifier in the comparison sequence. Experiments are presented comparing several linear-threshold algorithms on chronologically-ordered textual datasets. These experiments support our theoretical findings in that they show

Editors: Hans Ulrich Simon, Gabor Lugosi and Avrim Blum

An extended abstract appeared in the *Proceedings of the 19th Annual Conference on Learning Theory*, Springer, 2006. The work of all authors was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778.

G. Cavallanti (✉) · N. Cesa-Bianchi
DSI, Università di Milano, via Comelico 39, 20135 Milano, Italy
e-mail: cavallanti@dsi.unimi.it

N. Cesa-Bianchi
e-mail: cesa-bianchi@dsi.unimi.it

C. Gentile
DICOM, Università dell'Insubria, Varese, Italy
e-mail: claudio.gentile@uninsubria.it

to what extent randomized budget algorithms are more robust than deterministic ones when learning shifting target data streams.

Keywords Pattern classification · Mistake bounds · Perceptron algorithm · Budget algorithms

1 Introduction

On-line or incremental learning is a powerful technique for building kernel-based classifiers. On-line algorithms, like the kernel Perceptron algorithm and its many variants, are typically easy to implement, efficient to run, and have strong performance guarantees. In this paper, we study two important aspects related to incremental learning: tracking ability and memory boundedness. The need to track a target arises from the fact that on-line algorithms are often designed to perform well with respect to the best fixed classifier in hindsight within a given comparison class. However, this is a weak guarantee: in many real-world tasks, such as categorization of text generated by a newsfeed, it is not plausible to assume that a fixed classifier could perform consistently well on a long sequence of newsitems generated by the feed. For this reason, a "shifting" performance model has been introduced (see, e.g., Littlestone & Warmuth, 1994; Herbster & Warmuth, 1998; Auer & Warmuth, 1998; Herbster & Warmuth, 2001; Kivinen, Smola, & Williamson, 2004, and references therein) where the on-line algorithm is evaluated against an arbitrary sequence of *comparison classifiers*. In this shifting model, which is strictly harder than the traditional nonshifting performance model, the tracking ability refers to the fact that the performance of the algorithm is good to the extent that the data sequence is well predicted by a sequence of classifiers whose coefficients may change with time under certain constraints. If the algorithm is kernel-based we face the additional issue of the time and space needed to compute the classifier. In fact, kernel-based learners typically use a subset of previously observed data instances to encode a classifier (borrowing terminology from the Support Vector Machine literature (e.g., Vapnik, 1998; Schölkopf & Smola, 2002, we call these instances "support vectors"). The problem is that nearly all on-line algorithms need to store a new support vector after each prediction mistake. Thus, the number of supports grows unboundedly unless the data sequence is linearly separable in the Reproducing Kernel Hilbert Space (RKHS) induced by the kernel under consideration. To address this specific issue, variants of the Perceptron algorithm using a fixed *budget* of support vectors have been proposed by Crammer, Kandola, and Singer (2004) and Weston, Bordes, and Bottou (2005), and analyzed by Dekel, Shalev-Shwartz, and Singer (2006). These algorithms use a rule that, once the number of stored supports has reached the budget, evicts a support from the storage each time a new vector comes in. The eviction rule at the basis of our Randomized Budget Perceptron algorithm is surprisingly simple: On a mistaken trial, the algorithm adds in the new support vector after an old one has been chosen *at random* from the storage and discarded.

Since the tracking ability is naturally connected to a weakened dependence on the past, memory boundedness could be viewed as a way to obtain a good shifting performance. In fact, we will show that our Randomized Budget Perceptron algorithm

has a strong performance guarantee in the shifting model. In addition, and more importantly, this algorithm strikes the optimal trade-off $U = \Theta(\sqrt{B})$ between the largest norm U of a classifier in the comparison sequence and the given budget B. This improves on $U = O(\sqrt{B/(\ln B)})$ obtained by Dekel, Shalev-Shwartz, and Singer (2006) via a more complicated algorithm.

We ran experiments comparing our random budget policy with other rules (such as the "least recent" eviction rule adopted by Dekel, Shalev-Shwartz, and Singer, 2006) on shifting datasets derived from the Reuters Corpus Volume 1 (Reuters, 2000). These experiments show the effectiveness of budget algorithms when learning time-changing datasets and, in particular, the robustness of our random budget policy as compared to deterministic ones.

The paper is organized as follows. In the rest of this section we introduce our main notation, along with preliminary definitions. Section 2 introduces the Shifting Perceptron algorithm, a simple variant of the Perceptron algorithm achieving the best known shifting bound without budget restriction. This result will be used as a yardstick for the results of Section 3, where the Randomized Budget Perceptron algorithm is introduced and analyzed. Section 4 reports the experimental results. Finally, Section 5 is devoted to conclusions and open problems.

All of our algorithms are kernel-based. For notational simplicity, however, our analyses will be carried out using the linear kernel.

1.1 Basic definitions, preliminaries and notation

An *example* is a pair $(\boldsymbol{x}, y)$, where $\boldsymbol{x} \in \mathbb{R}^d$ is an *instance* vector and $y \in \{-1, +1\}$ is the associated binary label. We consider the standard on-line learning model of Angluin (1988) and Littlestone (1988), in which learning proceeds in a sequence of *trials*. In the generic trial t the algorithm observes instance $\boldsymbol{x}_t$ and outputs a prediction $\widehat{y}_t \in \{-1, +1\}$ for the label y_t associated with $\boldsymbol{x}_t$. We say that the algorithm has made a *prediction mistake* if $\widehat{y}_t \neq y_t$.

In this paper, we consider variants of the standard Perceptron algorithm of Block (1962) and Novikoff (1962). At each trial $t = 1, 2, \ldots$ this algorithm predicts y_t through the linear-threshold function $\widehat{y}_t = \text{SGN}(\boldsymbol{w}^\top \boldsymbol{x}_t)$, where $\boldsymbol{w} \in \mathbb{R}^d$ is a weight vector that is initially set to the zero vector $\mathbf{0}$. If a mistake is made at trial t, the algorithm updates $\boldsymbol{w}$ by performing the assignment $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_t \boldsymbol{x}_t$.

When the Perceptron algorithm is run in a RKHS the current hypothesis is represented as a linear combination of (kernel) dot-products with all past mistaken ("support") vectors $\boldsymbol{x}_t$. Since in any given trial the running time required to make a prediction scales linearly with the number of mistakes made so far, the overall running time needed by the kernel Perceptron algorithm is *quadratic* in the total number m of mistakes made. A memory bounded Perceptron algorithm tries to overcome this drawback by maintaining only a prearranged number of past support vectors, thereby turning the quadratic dependence on m into a linear one.

We measure the performance of our linear-threshold algorithms by the total number of mistakes they make on an arbitrary sequence of examples. In the standard performance model, the goal is to bound this total number of mistakes in terms of the performance of the best *fixed* linear classifier $\boldsymbol{u} \in \mathbb{R}^d$ in hindsight (note that we identify an arbitrary linear-threshold classifier with its coefficient vector $\boldsymbol{u}$). Since

finding $\boldsymbol{u} \in \mathbb{R}^d$ that minimizes the number of mistakes on a known sequence is a computationally hard problem, the performance of the best predictor in hindsight is often measured using the cumulative *hinge loss* (see, e.g., Freund & Schapire, 1999; Gentile & Warmuth, 1999). The hinge loss of a linear classifier $\boldsymbol{u}$ on example $(\boldsymbol{x}, y)$ is defined by $d(\boldsymbol{u}; (\boldsymbol{x}, y)) = \max\{0, 1 - y\boldsymbol{u}^\top \boldsymbol{x}\}$. Note that d is a convex function of the margin $y\boldsymbol{u}^\top \boldsymbol{x}$, and is also an upper bound on the indicator function of $\text{SGN}(\boldsymbol{u}^\top \boldsymbol{x}) \neq y$.

In the *shifting* or *tracking* performance model the learning algorithm faces the harder goal of bounding its total number of mistakes in terms of the cumulative hinge loss achieved by an arbitrary sequence $\boldsymbol{u}_0, \boldsymbol{u}_1 \ldots, \boldsymbol{u}_{n-1} \in \mathbb{R}^d$ of linear classifiers (also called *comparison vectors*). To make this goal feasible, the bound is allowed to scale also with the maximum norm $U = \max_t \|\boldsymbol{u}_t\|$ of the classifiers in the sequence and with the *total shift*

$$S_{\text{tot}} = \sum_{t=1}^{n-1} \|\boldsymbol{u}_{t-1} - \boldsymbol{u}_t\| \tag{1}$$

of the classifier sequence.

We assume for simplicity that all instances $\boldsymbol{x}_t$ are normalized, that is, $\|\boldsymbol{x}_t\| = 1$ for all $t \geq 1$. Finally, throughout this paper, we will use $\{\phi\}$ to denote the indicator function of the event defined by a predicate ϕ.

2 The shifting Perceptron algorithm

Our learning algorithm for shifting hyperplanes (Shifting Perceptron Algorithm, SPA) is described in Fig. 1. SPA has a positive input parameter λ which determines the rate of weight decay. The algorithm maintains a weight vector $\boldsymbol{w}$ (initially set to zero) and two more variables: a mistake counter k (initialized to zero) and a time-changing decaying factor λ_k (initialized to 1). When a mistake is made on some example $(\boldsymbol{x}_t, y_t)$ the signed instance vector $y_t \boldsymbol{x}_t$ is added to the old weight vector, just like in the Perceptron update rule. However, unlike the Perceptron rule, before adding $y_t \boldsymbol{x}_t$ SPA scales down the old weight, so as to diminish the importance of early update stages. The important thing to observe here is that the scaling factor $(1 - \lambda_k)$ changes with time, since $\lambda_k \to 0$ as more mistakes are made. Note that subscript t runs over all trials, while subscript k runs over mistaken trials only, thus k serves as an index for quantities ($\boldsymbol{w}_k$ and λ_k) which get updated only in those trials. In particular, at the *end* of each trial, k is equal to the number of mistakes made so far.

It is worth observing what the algorithm really does by unwrapping the recurrence $\boldsymbol{w}_{k+1} = (1 - \lambda_k)\boldsymbol{w}_k + y_t \boldsymbol{x}_t$. Assume at the end of trial t the algorithm has made $k + 1$ mistakes, and denote the mistaken trials by $t_0, t_1, \ldots, t_k$. We have

$$\boldsymbol{w}_{k+1} = \alpha_0\, y_{t_0} \boldsymbol{x}_{t_0} + \alpha_1\, y_{t_1} \boldsymbol{x}_{t_1} + \cdots + \alpha_k\, y_{t_k} \boldsymbol{x}_{t_k},$$

Algorithm: Shifting Perceptron.
Parameters: $\lambda > 0$;
Initialization: $\boldsymbol{w}_0 = \mathbf{0}$, $\lambda_0 = 1$, $k = 0$.
For $t = 1, 2, \ldots$

1. Get instance vector $\boldsymbol{x}_t \in \mathbb{R}^d$, $\|\boldsymbol{x}_t\| = 1$;
2. Predict with $\widehat{y}_t = \textsc{sgn}(\boldsymbol{w}_k^\top \boldsymbol{x}_t) \in \{-1, +1\}$;
3. Get label $y_t \in \{-1, +1\}$;
4. **If** $\widehat{y}_t \neq y_t$ **then**

$$\boldsymbol{w}_{k+1} = (1 - \lambda_k)\boldsymbol{w}_k + y_t \boldsymbol{x}_t \ , \qquad k \leftarrow k + 1 \ , \qquad \lambda_k = \frac{\lambda}{\lambda + k} \ .$$

Fig. 1 The shifting Perceptron algorithm

with[1]

$$\alpha_i = \prod_{j=i+1}^{k} (1 - \lambda_j) = \exp\left(\sum_{j=i+1}^{k} \log(1 - \lambda_j) \right) \approx \exp\left(- \sum_{j=i+1}^{k} \lambda_j \right)$$
$$= \exp\left(- \sum_{j=i+1}^{k} \frac{\lambda}{\lambda + j} \right) \approx \left(\frac{\lambda + i + 1}{\lambda + k + 1} \right)^{\lambda} \approx c_k \, (i + 1)^{\lambda},$$

c_k being a positive constant independent of i. Thus SPA is basically following a (degree-λ) polynomial vector decaying scheme, where the most recent "support vector" $\boldsymbol{x}_{t_k}$ is roughly worth $(k + 1)^{\lambda}$ times the least recent one (i.e., $\boldsymbol{x}_{t_0}$). Clearly enough, if $\lambda = 0$ all support vectors are equally important and we recover the classical Perceptron algorithm. Note that the simpler update $\boldsymbol{w}_{k+1} = (1 - \lambda)\boldsymbol{w}_k + y_t \boldsymbol{x}_t$ (i.e., with a constant scaling factor $\lambda \in [0, 1)$) would yield the exponential decaying scheme

$$\alpha_i = (1 - \lambda)^{k-i} = c_k \left(\frac{1}{1 - \lambda} \right)^{i}$$

investigated by, e.g., Kivinen, Smola, and Williamson (2004) and Dekel, Shalev-Shwartz, and Singer (2006).

Now, since we are facing a shifting target problem, it is reasonable to expect that the optimal degree λ in Fig. 1 depends on how fast the underlying target is drifting with time. As we will see in a moment, the above polynomial weighting scheme gives SPA a desirable robustness to parameter tuning, beyond making the analysis fairly simple.

2.1 Analysis

The analysis is a standard potential-based analysis for mistake-driven on-line algorithms (see Block, 1962; Littlestone, 1988; Novikoff, 1962).

[1] See the appendix for more precise approximations.

The following simple lemma is central to our analysis. The lemma bounds the growth rate of the norm of the algorithm's weight vector. The key point to remark is that, unlike previous algorithms and analyses (Dekel, Shalev-Shwartz, & Singer, 2006; Herbster & Warmuth, 2001; Kivinen, Smola, & Williamson, 2004), we do not force the weight vector $\boldsymbol{w}_k$ to live in a ball of bounded radius. Instead, we allow the weight vector to grow unboundedly, at a pace controlled in a rather precise way by the input parameter λ. The proof is given in the appendix.

Lemma 1. *With the notation introduced in Fig.* 1*, we have*

$$\|\boldsymbol{w}_k\| \leq e\sqrt{\frac{\lambda + k + 1}{2\lambda + 1}}$$

for any $k = 0, 1, 2 \ldots$*, where* e *is the base of natural logarithms.*

The following theorem contains our mistake bounds for SPA. The theorem delivers shifting bounds for any constant value of parameter λ. For instance, $\lambda = 0$ gives a shifting bound for the classical (non-shifting) Perceptron algorithm.[2] For any sequence $(\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots)$ of comparison vectors, the bound is expressed in terms of the cumulative hinge loss D, the shift S, and the maximum norm U of the sequence. These quantities are defined as follows:

$$D = \sum_{k=0}^{m-1} d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k})) \tag{2}$$

$$S = \sum_{k=1}^{m-1} \|\boldsymbol{u}_k - \boldsymbol{u}_{k-1}\| \tag{3}$$

$$U = \max_{t=0,\ldots,n-1} \|\boldsymbol{u}_t\| \,. \tag{4}$$

We recall that t_k is the trial at the end of which $\boldsymbol{w}_k$ gets updated and $\boldsymbol{u}_k$ is the comparison vector in trial t_k. Note that D and S are only summed over mistaken trials. Larger (but more interpretable) bounds can be obtained if these sums are replaced by sums running over all trials t. In particular, S may be replaced by S_{tot} defined in (1).

As expected, the optimal tuning of λ grows with S and, in turn, yields a mistake bound which scales linearly with S. We emphasize that, unlike previous investigations (such as those by Kivinen, Smola, and Williamson (2004)), our shifting algorithm is independent of scaling parameters (like the margin of the comparison classifiers $\langle \boldsymbol{u}_t \rangle$). In fact, our "optimal" tuning of λ turns out to be scale-free.

Theorem 2. *For any* $n \in \mathbb{N}$*, any sequence of examples* $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^d \times \{-1, +1\}$ *such that* $\|\boldsymbol{x}_t\| = 1$ *for each* t*, and any sequence of comparison vectors* $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1} \in \mathbb{R}^d$*, the algorithm in Fig.* 1 *makes a number* m *of mistakes bounded*

[2] Thus, even in a shifting framework the Perceptron algorithm, with no modifications, achieves a (suboptimal) shifting bound.

by

$$m \leq D + K^2 + K\sqrt{D + \lambda + 1}, \tag{5}$$

where $K = \frac{e}{\sqrt{2\lambda+1}}\,(S + (4\lambda + 1)\,U)$. *Moreover, if we set* $\lambda = \frac{S}{4U}$, *then we have* $K \leq e\sqrt{8SU + U^2}$ *and*

$$m \leq D + e\sqrt{\left(8SU + U^2\right) D} + e^2\,(8SU + U^2) + e\,(2S + 3U). \tag{6}$$

Proof: Consider how the potential $\boldsymbol{u}_k^\top \boldsymbol{w}_{k+1}$ evolves over mistaken trials. We can write

$$\begin{aligned}
\boldsymbol{u}_k^\top \boldsymbol{w}_{k+1} &= \boldsymbol{u}_k^\top((1-\lambda_k)\boldsymbol{w}_k + y_{t_k}\boldsymbol{x}_{t_k}) \\
&= (1-\lambda_k)\left(\boldsymbol{u}_k^\top \boldsymbol{w}_k - \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k + \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k\right) + y_{t_k}\boldsymbol{u}_k^\top \boldsymbol{x}_{t_k} \\
&= (1-\lambda_k)(\boldsymbol{u}_k - \boldsymbol{u}_{k-1})^\top \boldsymbol{w}_k + (1-\lambda_k)\boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k + y_{t_k}\boldsymbol{u}_k^\top \boldsymbol{x}_{t_k} \\
&\geq -(1-\lambda_k)\,\|\boldsymbol{u}_k - \boldsymbol{u}_{k-1}\|\,\|\boldsymbol{w}_k\| - \lambda_k\,\|\boldsymbol{u}_{k-1}\|\,\|\boldsymbol{w}_k\| + \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k + y_{t_k}\boldsymbol{u}_k^\top \boldsymbol{x}_{t_k} \\
&\geq -(1-\lambda_k)\,\|\boldsymbol{u}_k - \boldsymbol{u}_{k-1}\|\,\|\boldsymbol{w}_k\| - \lambda_k\,\|\boldsymbol{u}_{k-1}\|\,\|\boldsymbol{w}_k\| + \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k \\
&\quad + 1 - d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k}))
\end{aligned}$$

the last inequality following from the very definition of $d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k}))$. Rearranging yields

$$\begin{aligned}
\boldsymbol{u}_k^\top \boldsymbol{w}_{k+1} - \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k \geq &-(1-\lambda_k)\,\|\boldsymbol{u}_k - \boldsymbol{u}_{k-1}\|\,\|\boldsymbol{w}_k\| - \lambda_k\,\|\boldsymbol{u}_{k-1}\|\,\|\boldsymbol{w}_k\| \\
&+ 1 - d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k})).
\end{aligned}$$

Recalling that $\boldsymbol{w}_0 = \boldsymbol{0}$, we sum the above inequality over[3] $k = 0, 1, \ldots, m-1$, then we rearrange and overapproximate. This results in

$$m \leq D + \underbrace{\sum_{k=1}^{m-1}(1-\lambda_k)||\boldsymbol{u}_k - \boldsymbol{u}_{k-1}||\,||\boldsymbol{w}_k||}_{\text{(I)}} + \underbrace{\sum_{k=1}^{m-1}\lambda_k||\boldsymbol{u}_{k-1}||\,||\boldsymbol{w}_k||}_{\text{(II)}} + \underbrace{||\boldsymbol{u}_{m-1}||\,||\boldsymbol{w}_m||}_{\text{(III)}}.$$

We now use Lemma 1 to bound from above the three terms (I), (II), and (III):

$$\begin{aligned}
\text{(I)} &\leq S \max_{k=1,\ldots,m-1}((1-\lambda_k)\,\|\boldsymbol{w}_k\|) \\
&\leq S\,\frac{e\,(m-1)}{\lambda + m - 1}\sqrt{\frac{\lambda + m}{2\lambda + 1}}
\end{aligned}$$

[3] For definiteness, we set $\boldsymbol{u}_{-1} = \boldsymbol{0}$, though $\boldsymbol{w}_0 = \boldsymbol{0}$ makes this setting immaterial.

(from Lemma 1 and the definition of λ_k)

$$\leq e\, S \sqrt{\frac{\lambda + m}{2\lambda + 1}}. \tag{7}$$

Moreover, from Lemma 1 and the inequality $\frac{\sqrt{x+1}}{x} \leq 4(\sqrt{x+1} - \sqrt{x})$, $\forall x \geq 1$, applied with $x = \lambda + k$, we have

$$\begin{aligned}
\text{(II)} &\leq U \sum_{k=1}^{m-1} \lambda_k \, \|\boldsymbol{w}_k\| \\
&\leq U \sum_{k=1}^{m-1} \frac{e\lambda}{\lambda + k} \sqrt{\frac{\lambda + k + 1}{2\lambda + 1}} \\
&\leq U \frac{4e\lambda}{\sqrt{2\lambda + 1}} \sum_{k=1}^{m-1} \left(\sqrt{\lambda + k + 1} - \sqrt{\lambda + k}\right) \\
&= U \frac{4e\lambda}{\sqrt{2\lambda + 1}} \left(\sqrt{\lambda + m} - \sqrt{\lambda + 1}\right).
\end{aligned} \tag{8}$$

Finally, again from Lemma 1, we derive

$$\text{(III)} \leq e \, \|\boldsymbol{u}_{m-1}\| \sqrt{\frac{\lambda + m + 1}{2\lambda + 1}}. \tag{9}$$

At this point, in order to ease the subsequent calculations, we compute upper bounds on (7), (8) and (9) so as to obtain expressions having a similar dependence[4] on the relevant quantities around. We can write

$$\begin{aligned}
(7) &\leq e\, S \sqrt{\frac{\lambda + m + 1}{2\lambda + 1}}, \\
(8) &\leq 4\, e\, \lambda\, U \sqrt{\frac{\lambda + m + 1}{2\lambda + 1}}, \\
(9) &\leq e\, U \sqrt{\frac{\lambda + m + 1}{2\lambda + 1}}.
\end{aligned}$$

Putting together gives

$$m \leq D + e \, (S + (4\,\lambda + 1)\, U) \sqrt{\frac{\lambda + m + 1}{2\lambda + 1}}.$$

Solving for m and overapproximating once again gets

$$m \leq D + K^2 + K \sqrt{D + \lambda + 1},$$

[4] This seems to be a reasonable trade-off between simplicity and tightness.

where $K = K(\lambda) = \frac{e}{\sqrt{2\lambda+1}}\,(S + (4\,\lambda + 1)\,U)$. This is the claimed bound (5).

We now turn to the choice of λ. Choosing λ minimizing the above bound would require, among other things, prior knowledge of D. In order to strike a good balance between optimality and simplicity (and to rely on as little information as possible) we come to minimizing (an upper bound on) $K(\lambda)$. Set $\lambda = cS/U$, where c is some positive constant to be determined. This yields

$$K(\lambda) = e\,U\,\frac{(4c+1)\,S/U + 1}{\sqrt{2cS/U + 1}} \le e\sqrt{\frac{(4c+1)^2}{2c}\,SU + U^2}, \tag{10}$$

where we used

$$\frac{\alpha\,r + 1}{\sqrt{\beta\,r + 1}} \le \sqrt{\frac{\alpha^2}{\beta}r + 1} \qquad \alpha, r \ge 0,\ \beta > 0,$$

with $\alpha = 4c + 1$, $\beta = 2c$, and $r = S/U$. We minimize (10) w.r.t. c by selecting $c = 1/4$. Plugging back into (5) and overapproximating once more gives (6). □

3 A randomized Perceptron with budget

Consider the update $\boldsymbol{w}_{k+1} = (1 - \lambda_k)\boldsymbol{w}_k + y_t\boldsymbol{x}_t$ used by the algorithm in Fig. 1. As mentioned in the previous section, in the special case $\lambda_k = \lambda$ for all $k \ge 1$, this corresponds to associating with each support vector $\boldsymbol{x}_t$ a coefficient decreasing exponentially with the number of additional mistakes made. This exponential decay is at the core of many algorithms in the on-line learning literature, and has the immediate consequence of keeping bounded the norm of weight vectors. This same idea is used by the Forgetron of Dekel, Shalev-Shwartz, and Singer (2006), a recently proposed variant of the Perceptron algorithm that learns using a fixed budget of support vectors. In fact, it is not hard to show that the Forgetron analysis can be extended to the shifting model. In this section, we turn our attention to a way of combining shifting and budget algorithms by means of randomization, with no explicit weighting on the support vectors. As we show, this alternative approach yields a simple algorithm and a crisp analysis.

Consider a generic Perceptron algorithm with bounded memory. The algorithm has at its disposal a fixed number B of "support vectors", in the sense that, at any given trial, the weight vector $\boldsymbol{w}$ maintained by the algorithm is a linear combination of $y_{i_1}\boldsymbol{x}_{i_1}, y_{i_2}\boldsymbol{x}_{i_2}, \ldots, y_{i_B}\boldsymbol{x}_{i_B}$ where $i_1, \ldots, i_B$ is a subset of past trials where a mistake was made. Following (Crammer, Kandola, & Singer, 2004; Dekel, Shalev-Shwartz, & Singer, 2006; Weston, Bordes, & Bottou, 2005), we call B the algorithm's *budget*. As in the standard Perceptron algorithm, each example on which the algorithm makes a mistake becomes a support vector. However, in order not to exceed the budget, before adding a new support the algorithm has to discard an old one.

The analysis of the Forgetron algorithm is based on discarding the oldest support. The exponential coefficients $(1 - \lambda)^{-i}$ assigned to supports guarantee that, when λ is properly chosen as a function of B, the norm of the discarded vector is at most $1/\sqrt{B}$. In addition, it can be proven that the norm of $\boldsymbol{w}_k$ is at most $\sqrt{B/(\ln B)}$ for all $k \ge B$.

These facts can be used to prove a mistake bound in terms of the hinge loss of the best linear classifier $\boldsymbol{u}$ in hindsight, as long as $\|\boldsymbol{u}\| = O(\sqrt{B/(\ln B)})$. In this section we show that a purely random policy of discarding support vectors achieves a mistake bound without imposing on $\|\boldsymbol{u}\|$ any constraint stronger than $\|\boldsymbol{u}\| = O(\sqrt{B})$, which must be provably obeyed by any algorithm using budget B.

More precisely, suppose $\boldsymbol{w}_k$ makes a mistake on example $(\boldsymbol{x}_t, y_t)$. If the current number of support vectors is less than B, then our algorithm performs the usual additive update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_t \boldsymbol{x}_t$ (with no exponential scaling). Otherwise the algorithm chooses a random support vector Q_k, where $\mathbb{P}(Q_k = y_{i_j} \boldsymbol{x}_{i_j}) = 1/B$ for $j = 1, \ldots, B$, and performs the update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_t \boldsymbol{x}_t - Q_k$. Note that Q_k satisfies $\mathbb{E}_k Q_k = \boldsymbol{w}_k / B$ where $\mathbb{E}_k[\,\cdot\,]$ denotes the conditional expectation $\mathbb{E}[\,\cdot \mid \boldsymbol{w}_0, \ldots, \boldsymbol{w}_k]$. The resulting algorithm, called Randomized Budget Perceptron (RBP), is summarized in Fig. 2.

The main idea behind this algorithm is the following: by removing a random support we guarantee that, in expectation, the squared norm of the weight $\boldsymbol{w}_{k+1}$ increases by at most $2 - (2/B)\,\|\boldsymbol{w}_k\|^2$ each time we make an update (Lemma 3). This in turn implies that, at any *fixed* point in time, the *expected* norm of the current weight vector is $O(\sqrt{B})$. The hard part of the proof (Lemma 4) is showing that the sum of the norms of all distinct weights generated during a run has expected value $O(\sqrt{B})\,\mathbb{E}\,M + O(B^{3/2} \ln B)$, where M is the random number of mistakes.

3.1 Analysis

Similarly to Section 2.1, we state a simple lemma (whose proof is deferred to the appendix) that bounds in a suitable way the norm of the algorithm's weight vector.

Algorithm: Randomized Budget Perceptron.
Parameters: Budget $B \in \mathbb{N}$, $B \geq 2$;
Initialization: $\boldsymbol{w}_0 = \boldsymbol{0}$, $s = 0$, $k = 0$.
For $t = 1, 2, \ldots$

1. Get instance vector $\boldsymbol{x}_t \in \mathbb{R}^d$, $\|\boldsymbol{x}_t\| = 1$;
2. Predict with $\widehat{y}_t = \textsc{sgn}(\boldsymbol{w}_k^\top \boldsymbol{x}_t) \in \{-1, +1\}$;
3. Get label $y_t \in \{-1, +1\}$;
4. **If** $\widehat{y}_t \neq y_t$ **then**
   a) **If** $s < B$ **then**

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_t \boldsymbol{x}_t, \qquad k \leftarrow k + 1, \qquad s \leftarrow s + 1$$

   b) **else** let Q_k be a random support vector of $\boldsymbol{w}_k$ and perform the assignment

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_t \boldsymbol{x}_t - Q_k, \qquad k \leftarrow k + 1\ .$$

Fig. 2 The randomized budget Perceptron algorithm

Unlike Lemma 1, here we do not solve the recurrence involved. We rather stop earlier at a bound expressed in terms of conditional expectations, to be exploited in the proof of Lemma 4 below.

Lemma 3. *Let $B \geq 2$. With the notation introduced in this section, we have*

$$\mathbb{E}_k \, \|\boldsymbol{w}_{k+1}\|^2 \leq \begin{cases} k+1 & \text{for } k = 0, \ldots, B-1 \\ \left(1 - \frac{2}{B}\right) \|\boldsymbol{w}_k\|^2 + 2 & \text{for } k \geq B. \end{cases}$$

Moreover, using Jensen's inequality,

$$\mathbb{E}_k \, \|\boldsymbol{w}_{k+1}\| \leq \begin{cases} \sqrt{k+1} & \text{for } k = 0, \ldots, B-1 \\ \sqrt{\left(1 - \frac{2}{B}\right) \|\boldsymbol{w}_k\|^2 + 2} & \text{for } k \geq B. \end{cases}$$

The main result of this section bounds the expected number of mistakes, $\mathbb{E}\, M$, made by RBP in the shifting case.

For any sequence $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^d \times \{-1, +1\}$ of examples and any sequence $(\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-1})$ of comparison vectors, this bound is expressed in terms of the expectations of the cumulative hinge loss D, the shift S, and the maximal norm U of the sequence, defined in (2), (3) and (4), respectively. (All expectations are understood with respect to the algorithm's randomization.) Following the notation of previous sections, t_k denotes the (random) trial where $\boldsymbol{w}_k$ is updated and $\boldsymbol{u}_k$ is the comparison vector in trial t_k. Moreover, in what follows, we assume the underlying sequence of examples and the sequence $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots$ of linear classifiers are fixed and arbitrary. This implies that the value of the random variable $\{M = k\}$ is determined given $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{k-1}$ (i.e., $\{M = k\}$ is measurable w.r.t. the σ-algebra generated by $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{k-1}$), which is fairly essential for the ensuing analysis.

The next lemma is our key tool for proving expectation bounds. It may be viewed as a simple extension of Wald's equation to certain dependent processes.

Lemma 4. *With the notation and the assumptions introduced so far, we have, for any constant $\varepsilon > 0$, and any integer $B \geq 2$,*

$$\mathbb{E}\left[\sum_{k=B}^{M} \|\boldsymbol{w}_k\|\right] \leq \frac{B^{3/2}}{2} \ln \frac{B}{2\varepsilon} + (1+\varepsilon)\sqrt{B}\, \mathbb{E}[\max\{0, M+1-B\}].$$

Proof: Set for brevity $\rho = 1 - 2/B$. We can write

$$\mathbb{E}\left[\sum_{k=B}^{M} \|\boldsymbol{w}_k\|\right] = \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \, \|\boldsymbol{w}_k\|\right]$$

$$
\begin{aligned}
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \mathbb{E}_{k-1}[\{M \geq k\} \|\boldsymbol{w}_k\|]\right] \\
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \mathbb{E}_{k-1} \|\boldsymbol{w}_k\|\right] \\
&\quad \text{(since } \{M \geq k\} \text{ is determined by } \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{k-1}) \\
&\leq \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho \|\boldsymbol{w}_{k-1}\|^2 + 2}\right] \qquad (11) \\
&\quad \text{(from Lemma 3)} \\
&\leq \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k-1\} \sqrt{\rho \|\boldsymbol{w}_{k-1}\|^2 + 2}\right] \\
&= \mathbb{E}\left[\sum_{k=B-1}^{\infty} \{M \geq k\} \sqrt{\rho \|\boldsymbol{w}_k\|^2 + 2}\right] \\
&< \sqrt{\rho B + 2} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho \|\boldsymbol{w}_k\|^2 + 2}\right] \qquad (12)
\end{aligned}
$$

the last inequality following from Lemma 3, which implies $\|\boldsymbol{w}_{B-1}\|^2 \leq B - 1$ (recall that the algorithm proceeds deterministically in the first B steps, so here no expectation is needed).

Now, (12) can be treated in a similar fashion. We have

$$
\begin{aligned}
(12) &= \sqrt{\rho B + 2} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \mathbb{E}_{k-1}\left[\sqrt{\rho \|\boldsymbol{w}_k\|^2 + 2}\right]\right] \\
&\quad \text{(since, as before, } \{M \geq k\} \text{ is determined by } \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{k-1}) \\
&\leq \sqrt{\rho B + 2} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho(\rho \|\boldsymbol{w}_{k-1}\|^2 + 2) + 2}\right] \\
&\quad \text{(from Jensen's inequality and Lemma 3)} \\
&\leq \sqrt{\rho B + 2} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k-1\} \sqrt{\rho(\rho \|\boldsymbol{w}_{k-1}\|^2 + 2) + 2}\right] \\
&= \sqrt{\rho B + 2} + \mathbb{E}\left[\sum_{k=B-1}^{\infty} \{M \geq k\} \sqrt{\rho(\rho \|\boldsymbol{w}_k\|^2 + 2) + 2}\right] \\
&< \sqrt{\rho B + 2} + \sqrt{\rho(\rho B + 2) + 2} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho(\rho \|\boldsymbol{w}_k\|^2 + 2) + 2}\right]
\end{aligned}
$$

the last inequality following again from $\|\boldsymbol{w}_{B-1}\|^2 \leq B-1$. Iterating for a total of i times we obtain that (12) is less than

$$\sum_{j=0}^{i-1} \sqrt{\rho^{j+1} B + 2 \sum_{\ell=0}^{j} \rho^{\ell}} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho^i \|\boldsymbol{w}_k\|^2 + 2 \sum_{j=0}^{i-1} \rho^j} \right]$$
$$\leq \sum_{j=0}^{i-1} \sqrt{\rho^{j+1} B + B - \rho^{j+1} B} + \sqrt{\rho^i B^2 + B}\ \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \right],$$

where for the first term we used

$$\sum_{\ell=0}^{j} \rho^{\ell} = \frac{1-\rho^{j+1}}{1-\rho} = \frac{B}{2}(1-\rho^{j+1})$$

and for the second term we used

$$\sum_{j=0}^{i-1} \rho^j \leq \frac{1}{1-\rho} = \frac{B}{2}$$

and the trivial upper bound $\|\boldsymbol{w}_k\|^2 \leq B^2$ for all $k \geq 1$. We thus obtain

$$\mathbb{E}\left[\sum_{k=B}^{M} \|\boldsymbol{w}_k\| \right] < i\sqrt{B} + \sqrt{\rho^i B^2 + B}\ \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \right]$$
$$= i\sqrt{B} + \sqrt{\rho^i B^2 + B}\ \mathbb{E}[\max\{0, M+1-B\}].$$

We are free to choose the number i of iterations. We set i in a way that the factor $\sqrt{\rho^i B^2 + B}$ gets as small as $(1+\varepsilon)\sqrt{B}$. Since $\rho^i \leq e^{-2i/B}$ and $\sqrt{1+x} \leq 1 + x/2$ for any $x \geq 0$, it suffices to pick $i \geq \frac{B}{2} \ln \frac{B}{2\varepsilon}$, yielding the claimed inequality. □

Theorem 5. *Given any $\varepsilon \in (0,1)$, any $n \in \mathbb{N}$, any sequence of examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^d \times \{-1,+1\}$ such that $\|\boldsymbol{x}_t\| = 1$ for each t, the algorithm in Fig. 2 makes a number M of mistakes whose expectation is bounded as*

$$\mathbb{E}M \leq \frac{1}{\varepsilon}\mathbb{E}\,D + \frac{S_{\text{tot}}\sqrt{B}}{\varepsilon} + \frac{UB}{\varepsilon} + \frac{U\sqrt{B}}{2\varepsilon} \ln \frac{B}{2\varepsilon}$$

for any sequence of comparison vectors $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1} \in \mathbb{R}^d$, with expected hinge loss $\mathbb{E}\,D$, total shift S_{tot}, and such that $\max_t \|\boldsymbol{u}_t\| = U \leq \frac{1-\varepsilon}{1+\varepsilon}\sqrt{B}$.

Remark 6. Note the role played by the free parameter $\varepsilon \in (0,1)$. If ε is close to 0, then the comparison vectors $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1}$ are chosen from a large class, but the bound is loose. On the other hand, if ε is close to 1, our bound gets sharper but applies to a smaller comparison class. We can rewrite the above bound in terms of $U = \frac{1-\varepsilon}{1+\varepsilon}\sqrt{B}$.

For instance, setting $\varepsilon = 1/2$ results in

$$\mathbb{E}M \leq 2\,\mathbb{E}D + 18\,U(S_{\text{tot}} + U^2) + 6\,U^2 \ln(3\,U).$$

The dependence on S_{tot} is linear as in (6), which is the best bound we could prove on Perceptron-like algorithms without imposing a budget.

Remark 7. In the nonshifting case our bound reduces to

$$\mathbb{E}M \leq \frac{1}{\varepsilon}\,\mathbb{E}D + \frac{UB}{\varepsilon} + \frac{U\sqrt{B}}{2\varepsilon}\,\ln\frac{B}{2\varepsilon}.$$

This is similar to the (deterministic) Forgetron bound shown by Dekel, Shalev-Shwartz, and Singer (2006), though we have a better dependence on D and a worse dependence on U and B. However, and more importantly, whereas the Forgetron bound can be proven only for $\|\boldsymbol{u}\| = O(\sqrt{B/(\ln B)})$, our result just requires $\|\boldsymbol{u}\| = O(\sqrt{B})$. Note that this relationship between $\|\boldsymbol{u}\|$ and B achieved by our algorithm is optimal. Indeed, via a simple generalization of the argument presented by Dekel, Shalev-Shwartz, and Singer (2006), it can be proven that any[5] randomized algorithm using budget B makes a mistake with probability at least $1/(2(B+1))$ on each example $(\boldsymbol{x}_t, y_t)$ of an infinite sequence $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots$ such that $y_t\,\boldsymbol{u}^\top \boldsymbol{x}_t \geq 1$ for all $t = 1, 2, \ldots$, where $\boldsymbol{u} \in \mathbb{R}^{B+1}$ satisfies $\|\boldsymbol{u}\| = \sqrt{B+1}$.

Remark 8. From a computational standpoint, our simple randomized policy compares favorably with other eviction strategies that need to check the properties of *all* support vectors in the currect storage, such as those of Crammer, Kandola, and Singer (2004) and Weston, Bordes, and Bottou (2005). Thus, in this context, randomization exhibits a clear computational advantage.

Proof of Theorem 5: We proceed as in the proof of Theorem 2 and adopt the same notation used there. Note, however, that the weights $\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots$ are now the realization of a random process on $\mathbb{R}^d$ and that the number M of mistakes on a given sequence of examples is a random variable. Without loss of generality, in what follows we may set $\boldsymbol{w}_k = \boldsymbol{w}_M$ for all $k > M$. We can write

$$\begin{aligned}
\boldsymbol{u}_k^\top \boldsymbol{w}_{k+1} &= \boldsymbol{u}_k^\top (\boldsymbol{w}_k + y_{t_k}\boldsymbol{x}_{t_k} - \{k \geq B\}\,Q_k) \\
&= (\boldsymbol{u}_k - \boldsymbol{u}_{k-1})^\top \boldsymbol{w}_k + \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k + y_{t_k}\boldsymbol{u}_k^\top \boldsymbol{x}_{t_k} - \{k \geq B\}\,\boldsymbol{u}_k^\top Q_k \\
&\geq (\boldsymbol{u}_k - \boldsymbol{u}_{k-1})^\top \boldsymbol{w}_k + \boldsymbol{u}_{k-1}^\top \boldsymbol{w}_k + 1 - d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k})) - \{k \geq B\}\,\boldsymbol{u}_k^\top Q_k.
\end{aligned}$$

[5] To be precise, the argument applies to any randomized learning algorithm whose probability p of predicting $+1$ when the current instance is orthogonal to the span of its supports is a fixed and deterministic quantity.

We rearrange, sum over $k = 0, \ldots, M-1$, recall that $\boldsymbol{w}_0 = \mathbf{0}$, and take expectations on both sides of the resulting inequality,

$$\mathbb{E}M \le \mathbb{E}\left[\sum_{k=0}^{M-1} d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k}))\right] + \underbrace{\mathbb{E}\big[\boldsymbol{u}_{M-1}^\top \boldsymbol{w}_M\big]}_{\text{(I)}} + \underbrace{\mathbb{E}\left[\sum_{k=B}^{M-1} \boldsymbol{u}_k^\top Q_k\right]}_{\text{(II)}}$$
$$+ \underbrace{\mathbb{E}\left[\sum_{k=1}^{M-1} (\boldsymbol{u}_{k-1} - \boldsymbol{u}_k)^\top \boldsymbol{w}_k\right]}_{\text{(III)}}.$$

The first term in the right-hand side equals $\mathbb{E}\, D$. Thus we need to find suitable upper bounds on (I), (II), and (III). Recalling that $U = \max_t \|\boldsymbol{u}_t\|$, and noting that $\|\boldsymbol{w}_k\| \le B$ for all k, we have (I) $\le U\, B$. To bound (II), we write

$$\begin{aligned}
\text{(II)} &= \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \ge k+1\}\, \boldsymbol{u}_k^\top Q_k\right] \\
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \mathbb{E}_k\big[\{M \ge k+1\}\, \boldsymbol{u}_k^\top Q_k\big]\right] \\
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \ge k+1\} \boldsymbol{u}_k^\top \mathbb{E}_k\, Q_k\right] \\
&\quad \text{(since } \{M \ge k+1\} \text{ and } \boldsymbol{u}_k \text{ are determined given } \boldsymbol{w}_0, \ldots, \boldsymbol{w}_k) \\
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \ge k+1\} \frac{\boldsymbol{u}_k^\top \boldsymbol{w}_k}{B}\right] \\
&\quad \text{(since } \mathbb{E}_k\, Q_k = \boldsymbol{w}_k / B).
\end{aligned}$$

Hence

$$\begin{aligned}
\text{(II)} &\le \frac{U}{B}\, \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \ge k+1\}\, \|\boldsymbol{w}_k\|\right] \\
&\le \frac{U}{B}\, \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \ge k\}\, \|\boldsymbol{w}_k\|\right] = \frac{U}{B}\, \mathbb{E}\left[\sum_{k=B}^{M} \|\boldsymbol{w}_k\|\right] \\
&\le \frac{U\sqrt{B}}{2}\, \ln \frac{B}{2\varepsilon} + (1+\varepsilon) \frac{U}{\sqrt{B}}\, \mathbb{E}M \\
&\quad \text{(from Lemma 4 and the assumption } B \ge 2).
\end{aligned}$$

Next, we bound (III) as follows

$$\text{(III)} = \mathbb{E}\left[\sum_{k=1}^{M-1} \sum_{t=t_{k-1}+1}^{t_k} (\boldsymbol{u}_{t-1} - \boldsymbol{u}_t)^\top \boldsymbol{w}_k\right]$$

$$\leq \mathbb{E}\left[\sum_{k=1}^{M-1}\sum_{t=t_{k-1}+1}^{t_k}\|\boldsymbol{u}_{t-1}-\boldsymbol{u}_t\|\,\|\boldsymbol{w}_k\|\right]$$

$$\leq \mathbb{E}\left[\sum_{t=1}^{n-1}\|\boldsymbol{u}_{t-1}-\boldsymbol{u}_t\|\,\|\boldsymbol{w}_t\|\right]$$

where $\boldsymbol{w}_t$ is the random weight used by the algorithm at time t. A simple adaptation of Lemma 3 and an easy induction argument together imply that $\mathbb{E}\,\|\boldsymbol{w}_t\| \leq \sqrt{B}$ for all t. Thus we have

$$\mathbb{E}\left[\sum_{t=1}^{n-1}\|\boldsymbol{u}_{t-1}-\boldsymbol{u}_t\|\,\|\boldsymbol{w}_t\|\right] = \sum_{t=1}^{n-1}\|\boldsymbol{u}_{t-1}-\boldsymbol{u}_t\|\,\mathbb{E}\,\|\boldsymbol{w}_t\| \leq S_{\text{tot}}\sqrt{B}.$$

Piecing together gives

$$\mathbb{E}M \leq \mathbb{E}D + (1+\varepsilon)\frac{U}{\sqrt{B}}\,\mathbb{E}M + S_{\text{tot}}\sqrt{B} + UB + \frac{U\sqrt{B}}{2}\,\ln\frac{B}{2\varepsilon}.$$

The condition $U \leq \frac{1-\varepsilon}{1+\varepsilon}\sqrt{B}$ implies the desired result. □

4 Experiments

We tested the empirical performance of our algorithms by conducting a number of experiments on a collection of datasets derived from the first 20,000 newswire stories in the Reuters Corpus Volume 1 (RCV1, Reuters, 2000). A standard TF-IDF bag-of-words encoding was used to transform each news story into a normalized vector of real attributes (see Cesa-Bianchi, Conconi, & Gentile, 2003, for details on preprocessing).

In order to evaluate the tracking properties of our algorithms, we generated a collection of shifting binary datasets based on the most frequent categories (recall that each RCV1 article is labelled with one or more elements from a set of 101 semantic categories). Different shifting datasets were obtained by "binarizing" consecutive equally-sized chunks of RCV1 against different target categories. The binarization of a chunk against a given category is done in the most natural way: by replacing the original set of labels of each story with the single label $+1$ if the target category is in the set and with the label -1 otherwise.

4.1 Datasets

In our experiments we used the following four target categories: 70, 101, 4, and 59. These correspond to the 2nd, 3rd, 4th, and 5th most frequent categories, respectively, in the first 20,000 news stories of RCV1 (we did not use the most frequent category because of its significant overlap with the other ones).

These categories were used to generate four groups of binary datasets with increasing number of shifts (0, 1, 3, and 7). Each group contained four different datasets with the same number of shifts but binarized against a different sequence of categories (see

details below). The experiments in each group are reported as averages over the four datasets in the group.

The four groups of datasets are costructed as follows.

1. The datasets in the first group have no shifts. Each of the four datasets within this group is obtained through binarization against one of the four target categories.
2. The datasets in the second group have one shift. This amounts to splitting each dataset into two equally-sized chunks (thus each chunk has 10,000 examples) and associating different target categories with the two chunks. For each dataset, the two target categories are randomly selected.
3. The datasets in the third group have three shifts. Each dataset is obtained through binarization against a random permutation of the four target categories. Each chunk has 5000 examples.
4. The datasets in the fourth group have seven shifts. Each dataset is obtained through binarization against a random permutation of the four target categories (repeated twice). This results in datasets having 8 chucks, with 2500 examples each.

Since the newswire stories from RCV1 are chronologically ordered, our collection of datasets can naturally be viewed as modelling the task of detecting the "topic of the day". In this interpretation, a change in the target category (i.e., a "shift") reflects a change in the topic of the day, which must be detected by the learning algorithm.

4.2 Algorithms

Besides the Shifting Perceptron Algorithm (SPA) and the Randomized Budget Perceptron (RBP) algorithm, we evaluated the behavior of two non-budget baseline algorithms: the original Perceptron (PERC) algorithm, and the Second-Order Perceptron (SOP) algorithm, as defined in Cesa-Bianchi, Conconi, and Gentile (2005). In addition, we also tested the following first and second-order budget Perceptron algorithms.

- The Forgetron Algorithm (FA), introduced by Dekel, Shalev-Shwartz, and Singer (2006).
- A variant, here referred to as LBP (Least recent Budget Perceptron), of the Randomized Budget Perceptron algorithm where we always remove the oldest support whenever the budget is exceeded. Note that this algorithm differs from the Forgetron algorithm, since the latter may also decide to scale the contribution of old supports. In this sense LBP can be regarded as an aggressive variant of FA.
- The Randomized Budget Second-Order Perceptron algorithm (here abbreviated as RBSOP). It is a budget version of SOP in which the support to be discarded is chosen at random among the current set of supports in the cache.
- The variant (here referred to as LBSOP, Least recent Budget Second-Order Perceptron) of RBSOP where we always remove the oldest support vector.
- The (original) Budget Perceptron, here abbreviated as CKS, as described by Crammer, Kandola, and Singer (2004).

All algorithms were run with a polynomial kernel of degree four. We empirically observed that, on our datasets, the error rates exhibited by budget and non-budget algorithms dropped the most when we switched from a linear to a polynomial kernel of degree four. Little or no improvement was achieved for higher degrees.

Table 1 Average number of support vectors (rounded to the closest integer) used by non-budget perceptron algorithms on chunks of different size. Standard deviations are given in parentheses

Chunk size	PERC	SOP
20000	1368 (±152,29)	1358 (±134,68)
10000	772 (±84,60)	751 (±75,58)
5000	445 (±50,70)	419 (±43,17)
2500	260 (±32,98)	241 (±26,76)

Finally, in all second-order variants we set the parameter a to 1 (see Cesa-Bianchi, Conconi, & Gentile, 2005, for details), while for the Shifting Perceptron algorithm we manually tuned the decay parameter λ for each group of datasets.

4.3 Results

In order to provide a better insight into the tracking properties of budget algorithms, we first tested the performance (in terms of number of support vectors) of PERC and SOP on all individual chunks appearing in our shifting datasets. Average results are reported in Table 1.

The main experimental results are summarized in Figs. 3–5. For clarity, we plotted first and second-order algorithms separately. Because SPA and PERC exhibited the same performance in low shifting settings, we did not plot SPA error rates in Fig. 3 and in the first two plots of Fig. 4.

The two plots in Fig. 3 depict the (average) performance of the algorithms on non-shifting datasets, as the budget B increases.

As expected, on this task the budget versions tend to perform worse than their non-budget counterparts. Specifically, the second-order algorithms have a slight advantage over the first-order ones (had we used a linear kernel this advantage would have been considerably higher) especially for small budget sizes. Moreover, randomized and non-randomized budget versions of each base algorithm exhibit similar performance results. Unsurprisingly, CKS shows a much higher error rate than any other algorithm for

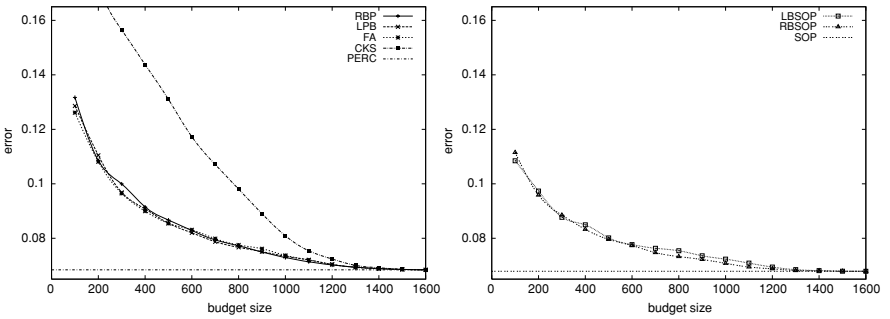

Fig. 3 Average error rates on non-shifting datasets, as the budget size B increases. The two horizontal lines show the error performance of the corresponding baseline ($B = \infty$) algorithms

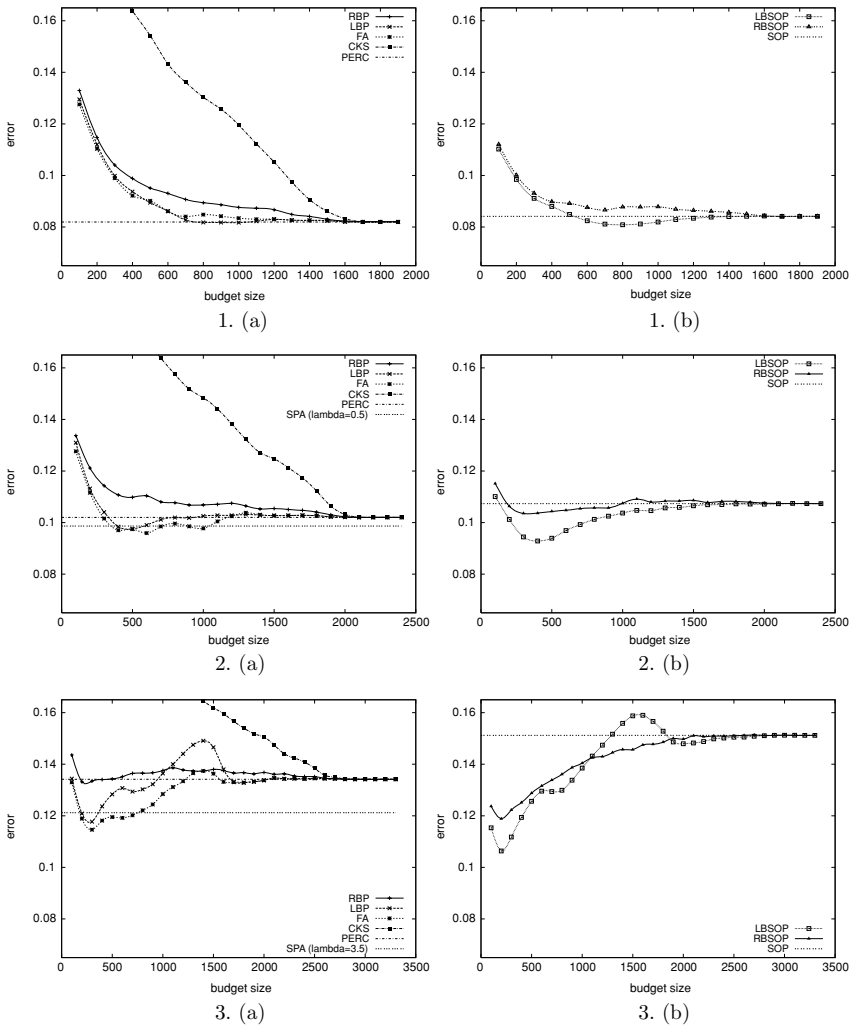

Fig. 4 Average error rates on 1-shift (1(a) and (b)), 3-shifts (2(a) and (b)) and 7-shifts (3(a) and (b)) tasks

the same budget size. In fact the RCV1 dataset is relatively noisy, and the performance of CKS is known to degrade quickly in these situations (see, e.g., Weston, Bordes, & Bottou, 2005).

As the number of shifts increases (see Fig. 4) each baseline moves higher, with the Second-Order Perceptron getting worse faster then the standard Perceptron algorithm. This may be due to the fact that SOP has a stronger dependence on the spectral structure of a dataset. When this changes, as is the case when a shift happens, the cost to re-adapt to a new dataset is higher.

A comparison between the results plotted in Fig. 4 and those in Table 1 reveals that the task of learning on a shifting dataset is considerably harder than the task of separately learning all the chunks in the same dataset. This can be explained by saying that every time a shift takes place, the target hypothesis changes significantly.

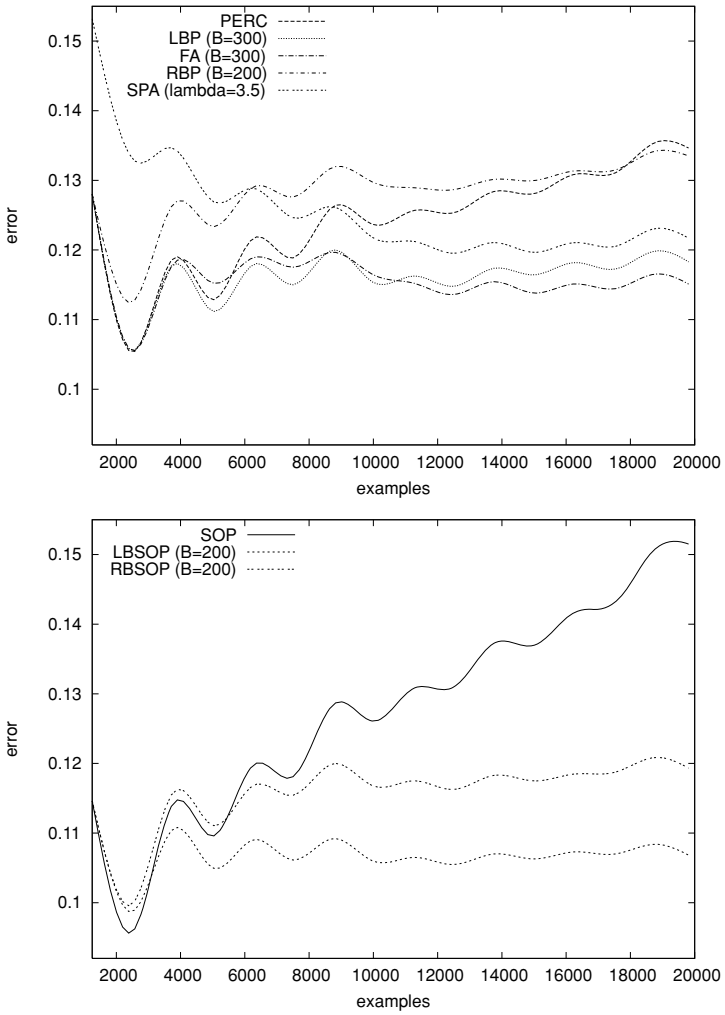

Fig. 5 Average instantaneous error rates on 7-shifts datasets

In all our experiments the randomized algorithms exhibited a substantially smooth convergence to the error rates of their non-budget counterparts. In high shifting settings the budget algorithms (both randomized and deterministic) tend to outperform the non-budget ones.

As for LBP, LBSOP and FA, they all manage to outperform their respective baselines for certain budget sizes. In particular, on any given dataset, they achieve the highest advantage when the budget size is close to the number of support vectors stored, on average, by their non-budget counterparts on each chunk of the dataset (compare to the results in Table 1). This makes sense because, as seen before, in order to learn the next hyperplane an algorithm has to basically unlearn the previous one. By limiting the budget to the number of support vectors shown in Table 1, for an appropriate chunk size, the effort of changing the current hypothesis is greatly diminished. In fact,

when these algorithms make a mistake exactly one example related to the previous hypothesis is removed from the active set and a new one is added while, at the same time, the budget is large enough to maintain a good hypothesis for the current target.

Randomized algorithms are a bit less capable of exploiting a "good" budget size to their advantage. It is likely that at some point, after a shift occurs, they start discarding recent examples more frequently than older ones. On the other hand, their performance is not dramatically changing for moderately smaller/larger budget sizes.

The budget versions of the second-order algorithms, both randomized and non-randomized, show lower error rates than their counterparts for small to medium budget sizes. When the budget size is kept small, the effort to track a shifting hyperplane appears to be easier, especially for second-order algorithms (compare to the above discussion on non-budget second-order algorithms on shifting datasets). Additionally, under this condition, the second-order variants can exploit their ability to learn a single chunk faster.

As Theorem 2 suggests, upon setting the decay parameter λ, the Shifting Perceptron algorithm was able to outperform the standard Perceptron algorithm. In addition, the error rate of SPA is close to the lowest error rate achieved by both LBP and FA.

A remark about the performance exhibited on the 7-shifts datasets. The graphs of LBP, LBSOP, and FA show a "bump" for budget values around 1500. This is actually an artifact of our dataset construction. Recall that the seven shifts are obtained by repeating twice the four target catergories on eight equally-sized chunks. A cache size of 1500 is just large enough to maintain examples learned on a chunk from the first half of the dataset while processing the same chunk from the second half. Hence, for values of the cache size smaller than 1500, the algorithms perform well because of the "least recent" eviction policy: when a mistake occurs while processing a chunk in the second half of the dataset, the supports belonging to the corresponding chunk in the first half have already been evicted, and thus an irrelevant support gets eliminated from the cache while a relevant one is added. For values of the cache size bigger than 1500, the algorithms also perform well because they are able to process a chunk in the second half without evicting vectors learned from the corresponding chunk in the first half. On the other hand, when the cache size is around 1500, the algorithms perform worse because on each mistake they add to the cache a support vector from a chunk while eliminating a relevant vector previously learned from the same chunk.

This argument also explains why the "bump" is above the baseline (corresponding to the performance of a standard Perceptron). Indeed, just like the budget algorithms using a large cache, the Perceptron, whose budget is unlimited, is able to take advantage of the chunk repetitions in the datasets.[6]

Note also that, on the same datasets, the randomized algorithms exhibit a more robust behavior. This is witnessed by the smooth changes in their performance as the budget varies. In fact, thanks to randomization, these algorithms always retain some fraction of the support vectors learned on all previous chunks.

[6] We noticed that using additional RCV1 categories (besides 70, 101, 4, and 59) to avoid the repetitions of chunks in experiments with high shift did not give good results. This because the remaining categories have significantly lower frequencies and, for this reason, they do not induce enough mistakes in shifting experiments.

Table 2 Order of magnitude of the time complexities involved in each update and evaluation step for the algorithms used in the experiments. B denotes the budget and m denotes the number of mistakes already made when the step takes place. The $O(1)$ required for the update step of SPA and FA is assuming a lazy update scheme, where the new scaling factors are computed within the evaluation phase. The $O(1)$ in the update step of RBP implicitely assumes the availability of an oracle that in unit time returns a random element from the set $\{1, \ldots, B\}$

Algorithm	Update	Evaluation
PERC	1	m
SPA	1	m
CKS	B	B
FA	1	B
RBP/LBP	1	B
SOP	m^2	m
RBSOP/LBSOP	B^2	B

Figure 5 shows the instantaneous error rates exhibited on 7-shifts datasets by both budget and non-budget algorithms. For the former we only plotted error rates for the best budget size. In order to get smoother curves, the error rate is sampled every 1250 examples.

In all our experiments we observed that the variance caused by the internal randomization of RBP and RBSOP did not have a significant impact on their performance.

In Table 2 we summarized the time complexity of the algorithms used in our experiments. In particular, the evaluation step requires $O(B)$ operations for all budget algorithms. On the other hand, the update step needs $O(1)$ operations for all first-order budget algorithms, except for CKS, for which the update is $O(B)$, and the second-order budget algorithms, whose update is quadratic in B.

5 Conclusions and open problems

In this paper we have shown that simple changes to the standard (kernel) Perceptron algorithm suffice to obtain efficient memory bounded algorithms with good theoretical shifting performance. Our elaborations deliver robust on-line procedures which we expect to be of practical relevance in many real-world data-intensive learning settings.

From the theoretical point of view, we have shown that these simple algorithms compare favorably with the existing kernel-based algorithms working in the on-line shifting framework. We expect that many of the results proven here could be extended to the family of p-norm algorithms studied by Grove, Littlestone, and Schuurmans (1997) and Gentile (2003), and to large margin on-line algorithms (see, e.g., Li & Long, 2002; Gentile, 2001).

In order to complement the theoretical analysis, we experimentally compared our algorithms to various first-order and second-order Perceptron algorithms, including both randomized and non randomized learners, in low and high shifting settings. The experiments showed that, in high shifting regimes, RBP achieves a performance comparable (and sometimes better) to that of the standard Perceptron

algorithm, even for remarkably small values of the budget. On the other hand, two out of three deterministic eviction policies achieve mistake rates smaller than RBP.[7] Indeed, both LBP and FA outperform the Perceptron algorithm for small budget values and even when the shifting is moderate. In particular, the eviction policy of FA gives the overall best results. The second-order budget algorithms also behave well. In particular, LBSOP achieves the lowest mistake rates among all algorithms and, in the 7-shift case, this minimum rate is achieved using a budget smaller than the budget used by the other algorithms to obtain their best performance. The experiments also included SPA. As suggested by our theory, with a suitable tuning of its parameter this algorithm actually outperforms the standard Perceptron algorithm.

A few issues left open by our theoretical analysis are the following. First, the bound exhibited in Theorem 5 shows an unsatisfactory dependence on U. This is due to the technical difficulty of finding a more sophisticated argument than the crude upper bound we use to handle expression (I) occurring in the proof. Second, our shifting analysis only depends on the *total* target shift. We conjecture that a better bound can be obtained under more specific shifting assumptions, like the smoothness condition $\|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\| \leq c$ for all t and for some constant $c > 0$. Third, motivated by the low variance observed in our experiments, it might be worth trying to see whether the result of Theorem 5 also holds with high probability, rather than just in expectation. Finally, it is not clear how our budget analysis could be extended to RBSOP, the second-order budget algorithm introduced in the experimental section. A direct combination of the analysis by Cesa-Bianchi, Conconi, and Gentile (2005) with the techniques of Section 3.1 does not seem to lead to interesting bounds.

Appendix A: Proof of Lemma 1

The lemma holds trivially for $k = 0$. For $k \geq 1$, let $t = t_{k-1}$ be the trial at the end of which $\boldsymbol{w}_{k-1}$ is updated. The update rule of Fig. 1 along with the condition $y_t \boldsymbol{w}_{k-1}^\top \boldsymbol{x}_t \leq 0$ allow us to write

$$\begin{aligned}\|\boldsymbol{w}_k\|^2 &= (1-\lambda_{k-1})^2 \|\boldsymbol{w}_{k-1}\|^2 + 2(1-\lambda_{k-1}) y_t \boldsymbol{w}_{k-1}^\top \boldsymbol{x}_t + \|\boldsymbol{x}_t\|^2 \\ &\leq (1-\lambda_{k-1})^2 \|\boldsymbol{w}_{k-1}\|^2 + 1.\end{aligned}$$

Unwrapping the recurrence yields

$$\|\boldsymbol{w}_k\|^2 \leq \sum_{i=0}^{k-1} \prod_{j=i+1}^{k-1} (1-\lambda_j)^2,$$

[7] This can be explained noting that our real-world datasets are presumably far from the worst-case data sequences implicitely taken into account by the theoretical analysis of RBP.

where the product is meant to be 1 if $i+1 > k-1$. The above, in turn, can be bounded as follows.

$$\begin{aligned}
\sum_{i=0}^{k-1}\prod_{j=i+1}^{k-1}(1-\lambda_j)^2 &\le \sum_{i=0}^{k-1}\exp\left(-2\sum_{j=i+1}^{k-1}\lambda_j\right)\\
&= \sum_{i=0}^{k-1}\exp\left(-2\lambda\sum_{j=i+1}^{k-1}\frac{1}{\lambda+j}\right)\\
&\le \sum_{i=0}^{k-1}\exp\left(-2\lambda\int_{i+1}^{k}\frac{dx}{\lambda+x}\right) = \sum_{i=0}^{k-1}\left(\frac{\lambda+i+1}{\lambda+k}\right)^{2\lambda}\\
&\le \frac{1}{(\lambda+k)^{2\lambda}}\int_{\lambda+1}^{\lambda+k+1}x^{2\lambda}dx \le \frac{1}{2\lambda+1}\frac{(\lambda+k+1)^{2\lambda+1}}{(\lambda+k)^{2\lambda}}\\
&= \left(\frac{\lambda+k+1}{2\lambda+1}\right)\left[\left(1+\frac{1}{\lambda+k}\right)^{\lambda+k}\right]^{\frac{2\lambda}{\lambda+k}} \le \left(\frac{\lambda+k+1}{2\lambda+1}\right)e^2,
\end{aligned}$$

where the last inequality uses $(1+1/x)^x \le e$ for all $x > 0$, and $\frac{2\lambda}{\lambda+k} \le 2$. Taking the square root completes the proof. □

Appendix B: Proof of Lemma 3

Let $t = t_k$ be the trial where $\boldsymbol{w}_k$ gets updated. We distinguish the two cases $k < B$ and $k \ge B$. In the first case no randomization is involved, and we have the standard (see, e.g., Block, 1962; Novikoff, 1962) Perceptron weight bound $\|\boldsymbol{w}_k\| \le \sqrt{k}$, $k = 1, \ldots, B$. In the case $k \ge B$ the update rule in Fig. 2 allows us to write

$$\begin{aligned}
\|\boldsymbol{w}_{k+1}\|^2 &= \|\boldsymbol{w}_k + y_t\boldsymbol{x}_t - Q_k\|^2\\
&= \|\boldsymbol{w}_k\|^2 + \|\boldsymbol{x}_t\|^2 + \|Q_k\|^2 - 2\boldsymbol{w}_k^\top Q_k + 2y_t(\boldsymbol{w}_k - Q_k)^\top\boldsymbol{x}_t\\
&\le \|\boldsymbol{w}_k\|^2 + 2 - 2\boldsymbol{w}_k^\top Q_k + 2y_t(\boldsymbol{w}_k - Q_k)^\top\boldsymbol{x}_t.
\end{aligned}$$

Recalling $\mathbb{E}_k\, Q_k = \boldsymbol{w}_k/B$, we take conditional expectation $\mathbb{E}_k$ on both sides:

$$\begin{aligned}
\mathbb{E}_k\,\|\boldsymbol{w}_{k+1}\|^2 &\le \|\boldsymbol{w}_k\|^2 + 2 - 2\frac{\boldsymbol{w}_k^\top\boldsymbol{w}_k}{B} + 2\left(1-\frac{1}{B}\right)y_t\,\boldsymbol{w}_k^\top\boldsymbol{x}_t\\
&\le \left(1-\frac{2}{B}\right)\|\boldsymbol{w}_k\|^2 + 2
\end{aligned}$$

the last step following from $y_t\,\boldsymbol{w}_k^\top\boldsymbol{x}_t \le 0$. This gives the desired bound on $\mathbb{E}_k\,\|\boldsymbol{w}_{k+1}\|^2$. The bound on $\mathbb{E}_k\,\|\boldsymbol{w}_{k+1}\|$ is a direct consequence of Jensen's inequality. □

References

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, *2*(4), 319–342.

Auer, P., & Warmuth, M. (1998). Tracking the best disjunction. *Machine Learning*, *32*(2), 127–150.

Block, H. (1962). The Perceptron: A model for brain functioning. *Review of Modern Physics*, *34*, 123–135.

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2003). Learning probabilistic linear-threshold classifiers via selective sampling. In *Proceedings of the 16th Annual Conference on Learning Theory* (pp. 373–386).

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2005). A second-order Perceptron algorithm. *SIAM Journal on Computing*, *43*(3), 640–668.

Crammer, K., Kandola, J., & Singer, Y. (2004). Online classification on a budget. In *Advances in Neural Information Processing Systems 16*.

Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2006). The Forgetron: A Kernel-based Perceptron on a fixed budget. In *Advances in Neural Information Processing Systems, 18* (pp. 259–266).

Freund, Y., & Schapire, R. (1999). Large margin classification using the Perceptron algorithm. *Machine Learning* (pp. 277–296).

Gentile, C. (2001). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, *2*, 213–242.

Gentile, C. (2003). The robustness of the *p*-norm algorithms. *Machine Learning*, *53*(3), 265–299.

Gentile, C., & Warmuth, M. (1999). Linear hinge loss and average margin. In *Advances in Neural Information Processing Systems, 10* (pp. 225–231).

Grove, A., Littlestone, N., & Schuurmans, D. (1997). General convergence results for linear discriminant updates. In *Proceedings of the 10th Annual Conference on Computational Learning Theory* (pp. 171–183).

Herbster, M., & Warmuth, M. (1998). Tracking the best expert. *Machine Learning*, *32*(2), 151–178.

Herbster, M., & Warmuth, M. (2001). Tracking the best linear predictor. *Journal of Machine Learning Research*, *1*, 281–309.

Kivinen, J., Smola, A., & Williamson, R. (2004). Online learning with Kernels. *IEEE Transactions on Signal Processing*, *52*(8), 2165–2176.

Li, Y., & Long, P. (2002). The relaxed online maximum margin algorithm. *Machine Learning*, *46*(1/3), 361–387.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*(4), 285–318.

Littlestone, N., & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation*, *108*, 212–261.

Novikoff, A. (1962). On Convergence proofs of Perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata* (Vol. XII. pp. 615–622).

Reuters (2000). `http://about.reuters.com/researchandstandards/corpus/`.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. MIT Press.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley.

Weston, J., Bordes, A., & Bottou, L. (2005). Online (and offline) on an even tighter budget. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics* (pp. 413–420).