Tracking the Best Hyperplane with a Simple Budget Perceptron

Nicolò Cesa-Bianchi[1] and Claudio Gentile[2]

[1] Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy
cesa-bianchi@dsi.unimi.it
[2] Dipartimento di Informatica e Comunicazione
Università dell'Insubria, Varese, Italy
claudio.gentile@uninsubria.it

Abstract. Shifting bounds for on-line classification algorithms ensure good performance on any sequence of examples that is well predicted by a sequence of smoothly 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.

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 for tracking abilities 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 (e.g., [19, 13, 2, 14, 15], 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 change gradually with time. If the algorithm is kernel-based, then 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 to encode a classifier (borrowing the Support Vector Machine [22, 21] terminology, we call these data "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 RKHS induced by the kernel under consideration. To address this specific issue, variants of the Perceptron algorithm have been proposed [6, 23] and analyzed [7] that work using a fixed *budget* of support vectors. 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. Our eviction rule, at the basis of the 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 $U = \Theta\big(\sqrt{B}\big)$ between the largest norm U of a classifier in the comparison sequence and the required budget B. This improves on $U = O\big(\sqrt{B/(\ln B)}\big)$ obtained in [7], via a more complicated algorithm.

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 our simple Randomized Budget Perceptron algorithm is described and analyzed. Finally, Section 4 is devoted to conclusions and open problems.

All of our algorithms are kernel-based. For notational simplicity, we define and analyze them without using kernels.

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 [1, 17] 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 [3, 20]. At each trial $t = 1, 2, \ldots$ this algorithm predicts y_t through the linear-threshold function $\widehat{y}_t = \textsc{sgn}\big(\boldsymbol{w}^\top \boldsymbol{x}_t\big)$, 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 the general problem of 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* [8, 11]. 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 $\textsc{sgn}\big(\boldsymbol{u}^\top \boldsymbol{x}\big) \neq y$.

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 Figure 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.

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

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.

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 $\boldsymbol{w}_{k+1} = \alpha_0\, y_{t_0} \boldsymbol{x}_{t_0} + \alpha_1\, y_{t_1} \boldsymbol{x}_{t_1} + \cdots + \alpha_k\, y_{t_k} \boldsymbol{x}_{t_k}$ with[3]

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

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.

Now, since we are facing a shifting target problem, it is reasonable to expect that the optimal degree λ 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 [3, 17, 20].

[3] 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 (e.g., [7, 14, 15]), 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 Figure 1, we have*

$$\|\boldsymbol{w}_{k+1}\| \le e\sqrt{\frac{\lambda+k+2}{2\lambda+1}}$$

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.[4] 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:

$$D = \sum_{k=0}^{m-1} d(\boldsymbol{u}_k; (\boldsymbol{x}_{t_k}, y_{t_k})), \quad S = \sum_{k=1}^{m-1} \|\boldsymbol{u}_k - \boldsymbol{u}_{k-1}\|, \quad U = \max_{t=0,\ldots,n-1} \|\boldsymbol{u}_t\| \ . \tag{2}$$

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 [15]) 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 1. *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 Figure 1 makes a number m of mistakes bounded by*

$$m \le D + K^2 + K\sqrt{D + \lambda + 1} \ , \tag{3}$$

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

$$m \le D + e\sqrt{(8SU + U^2)\, D} + e^2 (8SU + U^2) + e\,(2S + 3U) \ . \tag{4}$$

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

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

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

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

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

Recalling that $\boldsymbol{w}_0 = \boldsymbol{0}$, we sum the above inequality over[5] $k = 0, 1, \ldots, m-1$, then we rearrange and overapproximate. This results in

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

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

$$\begin{aligned}
\text{(I)} &\leq S \max_{k=1,\ldots,m-1}\Big((1-\lambda_k)\left\|\boldsymbol{w}_k\right\|\Big) \\
&\leq S\,\frac{e\,(m-1)}{\lambda+m-1}\sqrt{\frac{\lambda+m}{2\lambda+1}} \qquad \text{(from Lemma 1 and the definition of } \lambda_k) \\
&\leq e\,S\sqrt{\frac{\lambda+m}{2\lambda+1}}\ . \qquad (5)
\end{aligned}$$

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

$$\begin{aligned}
\text{(II)} &\leq U \sum_{k=1}^{m-1}\lambda_k\left\|\boldsymbol{w}_k\right\| \\
&\leq U \sum_{k=1}^{m-1}\frac{e\lambda}{\lambda+k}\sqrt{\frac{\lambda+k+1}{2\lambda+1}} \\
&\leq U\,\frac{4e\lambda}{\sqrt{2\lambda+1}}\sum_{k=1}^{m-1}\Big(\sqrt{\lambda+k+1}-\sqrt{\lambda+k}\Big) \\
&= U\,\frac{4e\lambda}{\sqrt{2\lambda+1}}\Big(\sqrt{\lambda+m}-\sqrt{\lambda+1}\Big). \qquad (6)
\end{aligned}$$

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

Finally, again from Lemma 1, we derive

$$\text{(III)} \leq e \, \|\boldsymbol{u}_{m-1}\| \sqrt{\frac{\lambda+m+1}{2\lambda+1}} \; . \tag{7}$$

At this point, in order to ease the subsequent calculations, we compute upper bounds on (5), (6) and (7) so as to obtain expressions having a similar dependence[6] on the relevant quantities around. We can write

$$(5) \leq e\,S \sqrt{\frac{\lambda+m+1}{2\lambda+1}} \; , \quad (6) \leq 4\,e\,\lambda\,U \sqrt{\frac{\lambda+m+1}{2\lambda+1}} \; , \quad (7) \leq e\,U \sqrt{\frac{\lambda+m+1}{2\lambda+1}} \; .$$

Putting together gives

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

Solving for m and overapproximating once again gets

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

where $K = K(\lambda) = \dfrac{e}{\sqrt{2\lambda+1}} \left(S + (4\,\lambda+1)\,U\right)$. This is the claimed bound (3).

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

$$K(\lambda) = e\,U \, \frac{(4c+1)\,S/U + 1}{\sqrt{2cS/U+1}} \leq e \sqrt{\frac{(4c+1)^2}{2c}\,SU + U^2}, \tag{8}$$

where we used $\dfrac{\alpha\,r+1}{\sqrt{\beta\,r+1}} \leq \sqrt{\dfrac{\alpha^2}{\beta} r + 1}$, $\alpha, r \geq 0$, $\beta > 0$, with $\alpha = 4c+1$, $\beta = 2c$, and $r = S/U$. We minimize (8) w.r.t. c by selecting $c = 1/4$. Plugging back into (3) and overapproximating once more gives (4). □

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 Figure 1. In the special case $\lambda_k = \lambda$ for all $k \geq 1$, 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 [7], 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

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

Forgetron analysis can be extended to the shifting model. In this section, we turn our attention to a way of combining shifting and budgeted 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 [6, 7, 23], 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 is based on discarding the oldest support. The exponential coefficients $(1-\lambda)^k$ 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 $k \geq B$. 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 $\|\boldsymbol{u}\| = O\big(\sqrt{B/(\ln B)}\big)$. In this section we show that a completely random policy of discarding support vectors achieves a mistake bound without imposing on $\|\boldsymbol{u}\|$ any constraint stronger than $\|\boldsymbol{u}\| = O\big(\sqrt{B}\big)$, 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 $\mathbb{P}\big(Q_k = y_{i_j}\boldsymbol{x}_{i_j}\big) = 1/B$ 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 Figure 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 2). This in turn implies that, at any *fixed* point in time, the expected norm of the current weight is $O(\sqrt{B})$. The hard part of the proof (Lemma 3) is showing that the sum of the norms of all distinct weights generated during a run has expected value $O(\sqrt{B})\,\mathbb{E}\,M + O\big(B^{3/2}\ln B\big)$, 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. 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 3 below.

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

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.

Lemma 2. *With the notation introduced in this section, we have*

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

Moreover, using Jensen's inequality,

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

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 $(\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-1})$ 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). (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., the event $\{M = k\}$ is measurable w.r.t. the σ-algebra generated by $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{k-1}$).

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 3. *With the notation and the assumptions introduced so far, we have, for any constant $\varepsilon > 0$,*

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

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

$$\begin{aligned}
\mathbb{E}\left[\sum_{k=B}^{M} \|\boldsymbol{w}_k\|\right] &= \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \|\boldsymbol{w}_k\|\right] \\
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \mathbb{E}_{k-1}\Big[\{M \geq k\} \|\boldsymbol{w}_k\|\Big]\right] \\
&= \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \mathbb{E}_{k-1} \|\boldsymbol{w}_k\|\right] \\
&\quad \text{(since } \{M \geq k\} \text{ is determined by } \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{k-1}) \\
&\leq \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho \|\boldsymbol{w}_{k-1}\|^2 + 2}\right] \qquad \text{(from Lemma 2)} \\
&\leq \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k-1\} \sqrt{\rho \|\boldsymbol{w}_{k-1}\|^2 + 2}\right] \\
&= \mathbb{E}\left[\sum_{k=B-1}^{\infty} \{M \geq k\} \sqrt{\rho \|\boldsymbol{w}_k\|^2 + 2}\right] \\
&\leq \sqrt{\rho B + 2} + \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k\} \sqrt{\rho \|\boldsymbol{w}_k\|^2 + 2}\right] \qquad (9)
\end{aligned}$$

the last inequality following from Lemma 2, which implies $\|\boldsymbol{w}_{B-1}\|^2 \leq B$.

Now, (9) can be treated in a similar fashion. We have

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

the last inequality following again from $\|\boldsymbol{w}_{B-1}\|^2 \leq B$. Iterating for a total of i times we obtain that (9) is at most

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

where for the first term we used $\sum_{\ell=0}^{j} \rho^\ell = \frac{1-\rho^{j+1}}{1-\rho} = B(1-\rho^{j+1})/2$, and for the second term we used the trivial upper bound $\left\|\boldsymbol{w}_k\right\|^2 \leq B^2$ for all $k \geq 1$ and $\sum_{\ell=0}^{j} \rho^\ell \leq \frac{1}{1-\rho} = B/2$. We thus obtain

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

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 $i \geq \frac{B}{2} \ln \frac{B^2}{2\varepsilon}$, yielding the claimed inequality. □

Theorem 2. *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 Figure 2 makes a number M of mistakes whose expectation is bounded as*

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

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 1. 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

$$\mathbb{E}M \leq 2\, \mathbb{E}\, D + 18\, U\big(S_{\text{tot}} + U^2\big) + 12\, U^2 \ln(3\, U) .$$

The dependence on S_{tot} is linear as in (4), which is the best bound we could prove on Perceptron-like algorithms without imposing a budget.

Remark 2. In the nonshifting case our bound reduces to

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

This is similar to the (deterministic) Forgetron bound shown in [7], 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 $\|\boldsymbol{u}\| = O\big(\sqrt{B/(\ln B)}\big)$, our result just requires $\|\boldsymbol{u}\| = O\big(\sqrt{B}\big)$. This is basically optimal, since it was shown in [7] that the condition $\|\boldsymbol{u}\| < \sqrt{B+1}$ is necessary for any on-line algorithm working on a budget B.

Remark 3. From a computational standpoint, our simple randomized policy compares favourably with other eviction strategies that need to check the properties of *all* support vectors in the currect storage, such as those in [6, 23]. Thus, in this context, randomization exhibits a clear computational advantage.

Proof (of Theorem 2). We proceed as in the proof of Theorem 1 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 example is a random variable. Without loss of generality, in what follows we assume $\boldsymbol{w}_k = \boldsymbol{w}_M$ for all $k > M$. We can write

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

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

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

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 $\|\boldsymbol{w}_k\| \leq B$ for all k, we have (I) $\leq U\, B$. To bound (II), we write

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

Hence

$$\begin{aligned}
\text{(II)} &\leq \frac{U}{B}\, \mathbb{E}\left[\sum_{k=B}^{\infty} \{M \geq k+1\}\, \|\boldsymbol{w}_k\| \right] \leq \frac{U}{B}\, \mathbb{E}\left[\sum_{k=B}^{M} \|\boldsymbol{w}_k\| \right] \\
&\leq \frac{U\sqrt{B}}{2}\, \ln \frac{B^2}{2\varepsilon} + (1+\varepsilon) \frac{U}{\sqrt{B}}\, \mathbb{E}\, M \qquad (\text{from Lemma 3}).
\end{aligned}$$

Next, we bound (III) as follows

$$\begin{aligned}\mathbb{E}\left[\sum_{k=1}^{M-1}(\boldsymbol{u}_{k-1}-\boldsymbol{u}_k)^\top \boldsymbol{w}_k\right] &= \mathbb{E}\left[\sum_{k=1}^{M-1}\sum_{t=t_{k-1}+1}^{t_k}(\boldsymbol{u}_{t-1}-\boldsymbol{u}_t)^\top \boldsymbol{w}_k\right] \\ &\le \mathbb{E}\left[\sum_{k=1}^{M-1}\sum_{t=t_{k-1}+1}^{t_k}\|\boldsymbol{u}_{t-1}-\boldsymbol{u}_t\|\,\|\boldsymbol{w}_k\|\right] \\ &\le \mathbb{E}\left[\sum_{t=1}^{n-1}\|\boldsymbol{u}_{t-1}-\boldsymbol{u}_t\|\,\|\boldsymbol{w}_t\|\right]\end{aligned}$$

where $\boldsymbol{w}_t$ is the random weight used by the algorithm at time t. A simple adaptation of Lemma 2 and an easy induction argument together imply that $\mathbb{E}\,\|\boldsymbol{w}_t\| \le \sqrt{B}$ for all t. Thus we have

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

Piecing together gives

$$\mathbb{E}\,M \le \mathbb{E}\,D + (1+\varepsilon)\frac{U}{\sqrt{B}}\,\mathbb{E}\,M + S_{\mathrm{tot}}\sqrt{B} + U\,B + \frac{U\sqrt{B}}{2}\,\ln\frac{B^2}{2\varepsilon}\ .$$

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

4 Conclusions and ongoing research

In this paper we have shown that simple changes to the standard (kernel) Perceptron algorithm suffice to obtain efficient shifting and memory bounded algorithms. 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 favourably with the existing kernel-based algorithms working in the on-line shifting framework. Many of the results we have proven here can easily be extended to the family of p-norm algorithms [12, 10], to large margin on-line algorithms (e.g., [16, 9]) and to other Perceptron-like algorithms, such as the second-order Perceptron algorithm [5].

A few issues we are currently working on are the following. The bound exhibited in Theorem 2 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. In fact, we believe this argument is within reach. Finally, we are trying to see whether our statement also holds with high probability, rather than just in expectation.

This paper introduces new on-line learning technologies which, as we said, can be combined with several existing techniques. We are planning to make experiments to give evidence of the theoretical behavior of algorithms resulting from such combinations.

References

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.
2. P. Auer and M. Warmuth. Tracking the best disjunction. *Machine Learning*, 3:127–150, 1998.
3. H.D. Block. The Perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962.
4. A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *JMLR*, 6:1579–1619, 2005.
5. N. Cesa-Bianchi, A. Conconi, and C. Gentile. A second-order Perceptron algorithm. *SIAM Journal of Computing*, 34(3):640–668, 2005.
6. K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In *Proc. 16th NIPS*, 2003.
7. O. Dekel, S. Shalev-Shwartz, Y. Singer, The Forgetron: a kernel-based Perceptron on a fixed budget. In Proc. 19th NIPS, 2005.
8. Y. Freund, and R. Schapire. Large margin classification using the Perceptron algorithm. *Journal of Machine Learning*, 37(3)277–296, 1999.
9. C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.
10. C. Gentile. The robustness of the p-norm algorithms. *Machine Learning*, 53:265–299, 2003.
11. C. Gentile and M. Warmuth. Linear hinge loss and average margin. In *Advances in Neural Information Processing Systems 10*, MIT Press, pp. 225–231, 1999.
12. A. J. Grove, N. Littlestone and D. Schuurmans. General convergence results for linear discriminant updates. *Journal of Machine Learning*, 43(3):173–210.
13. M. Herbster and M. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.
14. M. Herbster and M. Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.
15. J. Kivinen, A.J. Smola, and R.C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2004.
16. Y. Li, and P. Long. The relaxed online maximum margin algorithm. *Journal of Machine Learning*, 46(1):361–387, 2002.
17. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
18. N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. PhD thesis, University of California Santa Cruz, 1989.
19. N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.
20. A.B.J. Novikov. On convergence proofs on Perceptrons. In *Proc. of the Symposium on the Mathematical Theory of Automata, vol. XII*, pages 615–622, 1962.
21. B. Schölkopf and A. Smola, *Learning with kernels*, MIT Press, 2002.
22. V. Vapnik. *Statistical learning theory*. J. Wiley & Sons, New York, 1998.
23. J. Weston, A. Bordes, and L. Bottou. Online (and offline) on an even tighter budget. In *Proc. 10th AIStat*, pp. 413-420, 2005.

A Proof of Lemma 1

Let $t = t_k$ be the trial at the end of which $\boldsymbol{w}_k$ is updated. The update rule of Figure 1 along with the condition $y_t \boldsymbol{w}_k^\top \boldsymbol{x}_t \leq 0$ allow us to write for each $k \geq 0$

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

Unwrapping the recurrence yields $\|\boldsymbol{w}_{k+1}\|^2 \le \sum_{i=0}^{k} \prod_{j=i+1}^{k} (1-\lambda_j)^2$, where the product is meant to be 1 if $i+1 > k$. The above, in turn, can be bounded as follows.

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

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

B Proof of Lemma 2

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 (e.g., [3, 20]) Perceptron weight bound $\|\boldsymbol{w}_k\| \le \sqrt{k}$, $k = 1, \ldots, B$. In the case $k \ge B$ the update rule in Figure 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:

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

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. □