Fast boosting using adversarial bandits

Róbert Busa-Fekete[1,2] BUSAROBI@GMAIL.COM
Balázs Kégl[1] BALAZS.KEGL@GMAIL.COM
[1]LAL/LRI, University of Paris-Sud, CNRS, 91898 Orsay, France

[2]Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged, Aradi vértanúk tere 1., H-6720 Szeged, Hungary

Abstract

In this paper we apply multi-armed bandits (MABs) to improve the computational complexity of AdaBoost. AdaBoost constructs a strong classifier in a stepwise fashion by selecting simple base classifiers and using their weighted "vote" to determine the final classification. We model this stepwise base classifier selection as a sequential decision problem, and optimize it with MABs where each arm represents a subset of the base classifier set. The MAB gradually learns the "usefulness" of the subsets, and selects one of the subsets in each iteration. AdaBoost then searches only this subset instead of optimizing the base classifier over the whole space. The main improvement of this paper over a previous approach is that we use an adversarial bandit algorithm instead of stochastic bandits. This choice allows us to prove a weak-to-strong-learning theorem, which means that the proposed technique remains a boosting algorithm in a formal sense. We demonstrate on benchmark datasets that our technique can achieve a generalization performance similar to standard AdaBoost for a computational cost that is an order of magnitude smaller.

1. Introduction

AdaBoost (Freund & Schapire, 1997) is one of the best off-the-shelf learning methods developed in the last decade. It constructs a classifier in a stepwise fashion by adding simple classifiers (called *base classifiers*) to a pool, and using their weighted "vote" to determine the final classification. The simplest base learner used in practice is the *decision stump*, a one-decision two-leaf decision tree. Learning a decision stump means selecting a feature and a threshold, so the running time of AdaBoost with stumps is proportional to the number of data points n, the number of attributes d, and the number of boosting iterations T. When *trees* (Quinlan, 1993) or *products* (Kégl & Busa-Fekete, 2009) are constructed over the set of stumps, the computational cost is multiplied by an additional factor of the number of tree levels $\log N$ (where N is the number of leaves) or the number of terms m. Although the running time is linear in each of these factors, the algorithm can be prohibitively slow if the data size n and/or the number of features d is large.

There are essentially two ways to accelerate AdaBoost in this setting: one can either limit the number of data points n used to train the base learners, or one can cut the search space by using only a subset of the d features. Although both approaches increase the number of iterations T needed for convergence, the net computational time can still be significantly decreased. The former approach has a basic version when the base learner is not trained on the whole weighted sample, rather on a small subset selected randomly using the weights as a discrete probability distribution (Freund & Schapire, 1997). A recently proposed algorithm of this kind is FilterBoost (Bradley & Schapire, 2008), which assumes that an oracle can produce an unlimited number of labeled examples, one at a time. In each boosting iteration, the oracle generates sample points that the base learner can either accept or reject, and then the base learner is trained on a small set of accepted points. The latter approach was proposed by (Escudero et al., 2000) which introduces several feature selection and ranking methods used to accelerate AdaBoost. In particular, the LazyBoost algorithm chooses a fixed-size random subset

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

of the features in each boosting iteration, and trains the base learner using only this subset. This technique was successfully applied to face recognition where the number of features can be extremely large (Viola & Jones, 2004).

Recently, (Busa-Fekete & Kégl, 2009) proposed an improvement of LazyBoost by replacing the random selection by a biased selection towards features that proved to be useful in previous iterations. Learning the usefulness was achieved by using multi-armed bandit (MAB) algorithms. They used a *stochastic* MAB (UCB (Auer et al., 2002a)) which assumes that the rewards are generated randomly from a stationary distribution. The algorithm was successful in accelerating AdaBoost in practice, but due to the inherently non-stochastic nature of the rewards, (Busa-Fekete & Kégl, 2009) could not state anything on the algorithmic convergence of the technique, and so the connection between AdaBoost and bandits remained slightly heuristic.

In this paper we propose to use *adversarial* bandits instead of stochastic bandits in a similar setup. In this model the rewards are not assumed to be drawn from a stationary distribution, and they can depend arbitrarily on the past. Using adversarial bandits, we can prove a weak-to-strong-learning theorem which means that the proposed technique remains a boosting algorithm in a formal sense. Furthermore, the new algorithm also appears good in practice: if anything, it seems better in terms of generalization performance, and, above all, in computational complexity.

The paper is organized as follows. We start by reviewing the technical details of the AdaBoost.MH algorithm and the base learners we will use in the experiments in Section 2. Section 3 contains our main results where we describe the adversarial MAB algorithm and its interaction with AdaBoost.MH, and state our weak-to-strong learning result for the proposed algorithm. We present our experimental results in Section 4 where first we demonstrate on synthetic data sets that the bandit based algorithms are indeed able to discover useful features, then we furnish results on benchmark data sets that which indicate that the proposed algorithm improves the running time of AdaBoost.MH by at least an order of magnitude, without losing much of its generalization capacity.

2. AdaBoost.MH

For the formal description, let $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ be the $n \times d$ *observation matrix*, where $x_i^{(j)}$ are the elements of the d-dimensional observation vectors $\mathbf{x}_i \in \mathbb{R}^d$. We are also given a *label matrix* $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ of dimension $n \times K$ where $\mathbf{y}_i \in \{+1, -1\}^K$. In *multi-class* classification one and only one of the elements of $\mathbf{y}_i$ is $+1$, whereas in *multi-label* (or *multi-task*) classification $\mathbf{y}_i$ is arbitrary, meaning that the observation $\mathbf{x}_i$ can belong to several classes at the same time. In the former case we will denote the index of the correct class by $\ell(\mathbf{x}_i)$.

AdaBoost.MH$(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}, \textsc{Base}(\cdot,\cdot,\cdot), T)$

1 **for** $t \leftarrow 1$ **to** T

2 $\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot) \leftarrow \textsc{Base}\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$

3 $\quad$ **for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K

4 $\quad\quad w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \dfrac{\exp\big(-h_\ell^{(t)}(\mathbf{x}_i) y_{i,\ell}\big)}{\sum_{i'=1}^{n}\sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} e^{-h_{\ell'}^{(t)}(\mathbf{x}_{i'}) y_{i',\ell'}}}$

5 **return** $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\cdot)$

Figure 1. The pseudocode of the AdaBoost.MH algorithm. $\mathbf{X}$ is the observation matrix, $\mathbf{Y}$ is the label matrix, $\mathbf{W}^{(1)}$ is the initial weight matrix, $\textsc{Base}(\cdot,\cdot,\cdot)$ is the base learner algorithm, and T is the number of iterations. $\alpha^{(t)}$ is the base coefficient, $\mathbf{v}^{(t)}$ is the vote vector, $\varphi^{(t)}(\cdot)$ is the scalar base classifier, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) classifier.

The goal of the AdaBoost.MH algorithm ((Schapire & Singer, 1999), Figure 1) is to return a vector-valued classifier $\mathbf{f} : \mathcal{X} \to \mathbb{R}^K$ with a small *Hamming loss*

$$R_{\mathrm{H}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \mathbb{I}\left\{\operatorname{sign}\big(f_\ell^{(T)}(\mathbf{x}_i)\big) \neq y_{i,\ell}\right\}^1$$

by minimizing its upper bound (the exponential margin loss)

$$R_{\mathrm{e}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\big(-f_\ell^{(T)}(\mathbf{x}_i) y_{i,\ell}\big), \tag{1}$$

where $f_\ell(\mathbf{x}_i)$ is the ℓth element of $\mathbf{f}(\mathbf{x}_i)$. The user-defined weights $\mathbf{W}^{(1)} = \big[w_{i,\ell}^{(1)}\big]$ are usually set either uniformly to $w_{i,\ell}^{(1)} = 1/(nK)$, or, in the case of multi-class classification, to

$$w_{i,\ell}^{(1)} = \begin{cases} \frac{1}{2n} & \text{if } \ell = \ell(\mathbf{x}_i) \text{ (i.e., if } y_{i,\ell} = 1), \\ \frac{1}{2n(K-1)} & \text{otherwise (i.e., if } y_{i,\ell} = -1) \end{cases} \tag{2}$$

to create K well-balanced one-against-all classification problems. AdaBoost.MH builds the final classifier $\mathbf{f}$ as a sum of *base classifiers* $\mathbf{h}^{(t)} : \mathcal{X} \to \mathbb{R}^K$ returned by a *base learner* algorithm $\textsc{Base}(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)})$ in each

iteration t. In general, the base learner should seek to minimize the *base objective*

$$E\big(\mathbf{h}, \mathbf{W}^{(t)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp\big(-h_\ell(\mathbf{x}_i) y_{i,\ell}\big). \quad (3)$$

Using the weight update formula of Line 4 (Figure 1), it can be shown that

$$R_{\mathrm{e}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \prod_{t=1}^{T} E\big(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\big), \quad (4)$$

so minimizing (3) in each iteration is equivalent to minimizing (1) in an iterative greedy fashion. By obtaining the multi-class prediction $\widehat{\ell}(\mathbf{x}) = \arg\max_\ell f_\ell^{(T)}(\mathbf{x})$, it can also be proven that the "traditional" multi-class loss (or *one-error*)

$$R\big(\mathbf{f}^{(T)}\big) = \sum_{i=1}^{n} \mathbb{I}\left\{\ell(\mathbf{x}_i) \neq \widehat{\ell}(\mathbf{x}_i)\right\} \quad (5)$$

has an upper bound $K R_{\mathrm{e}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big)$ if the weights are initialized uniformly, and $\sqrt{K-1} R_{\mathrm{e}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big)$ with the multi-class initialization (2). This justifies the minimization of (1).

2.1. Learning the base classifier

In this paper we use *discrete* AdaBoost.MH in which the vector-valued base classifier $\mathbf{h}(\mathbf{x})$ is represented as

$$\mathbf{h}(\mathbf{x}) = \alpha \mathbf{v} \varphi(\mathbf{x}), \quad (6)$$

where $\alpha \in \mathbb{R}^+$ is the *base coefficient*, $\mathbf{v} \in \{+1, -1\}^K$ is the *vote vector*, and $\varphi(\mathbf{x}) : \mathbb{R}^d \to \{+1, -1\}$ is a *scalar* base classifier. It can be shown that to minimize (3), one has to choose a $\mathbf{v}$ and a φ that maximize the *edge*

$$\gamma = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}. \quad (7)$$

The optimal coefficient is then

$$\alpha = \frac{1}{2} \log \frac{1+\gamma}{1-\gamma}.$$

It is also well known that the base objective (3) can be expressed as

$$E\big(\mathbf{h}, \mathbf{W}\big) = \sqrt{(1+\gamma)(1-\gamma)} = \sqrt{1-\gamma^2}. \quad (8)$$

The simplest scalar base learner used in practice is the *decision stump*, a one-decision two-leaf decision tree of the form

$$\varphi_{j,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise,} \end{cases} \quad (9)$$

where j is the index of the selected feature and b is the decision threshold.

Although boosting decision stumps often yields satisfactory results, the state-of-the-art performance of AdaBoost is usually achieved by using *decision trees* as base learners, parametrized by their number of leaves N. We also test our approach using a recently proposed base learner that seems to outperform boosted trees on large benchmarks (Kégl & Busa-Fekete, 2009). The goal of this learner is to optimize *products* of simple base learners of the form $\mathbf{h}(\cdot) = \alpha \prod_{j=1}^{m} \mathbf{v}_j \varphi_j(\cdot)$, where the vote vectors $\mathbf{v}_j$ are multiplied element-wise. The base learner is parametrized by the number of terms m.

3. Reducing the search space of the base learners using multi-armed bandits

This section contains our main results. In Section 3.1 we provide some details on MAB algorithms necessary for understanding our approach. Section 3.2 describes the concrete MAB algorithm Exp3.P and its interaction with AdaBoost.MH. Section 3.3 gives our weak-to-strong learning result for the proposed algorithm AdaBoost.MH.Exp3.P. Finally, in Section 3.4 we elaborate some of the algorithmic aspects of the generic technique.

3.1. The multi-armed bandit problem

In the classical stochastic bandit problem the decision maker pulls an arm out of M arms at discrete time steps. Selecting an arm $j^{(t)}$ in iteration t results in a random reward $r_{j^{(t)}}^{(t)} \in [0,1]$ coming from a stationary distribution. The goal of the decision maker is to maximize the expected sum of the rewards received. Intuitively, the decision maker has to strike a balance between using arms with large past rewards (exploitation) and trying arms that have not been tested enough times (exploration). Formally, let $G^{(T)} = \sum_{t=1}^{T} r_{j^{(t)}}^{(t)}$ be the total reward that the decision maker receives up to the Tth iteration. Then the performance of the decision maker can be evaluated in terms of the *regret* with respect to the best arm retrospectively, defined as $T \max_{1 \leq j \leq M} \mu_j - G^{(T)}$ where μ_j is the (unknown) expected reward of the jth arm.

Contrary to the classical stochastic MABs, in the adversarial setup (Auer et al., 1995) there is a second, non-random player that chooses a reward vector $\mathbf{r}^{(t)} \in \mathbb{R}^M$ in each iteration. There is no restriction on the series of reward vectors $\mathbf{r}^{(t)}$, in particular, they can be influenced by the decision maker's previous ac-

tions. Only the reward $r_{j^{(t)}}^{(t)}$ of the chosen arm $j^{(t)}$ is revealed to the decision maker. Since the rewards are not drawn from a stationary distribution, any kind of regret can only be defined with respect to a particular sequence of actions. The most common performance measure is the *weak regret*

$$\max_j \sum_{t=1}^{T} r_j^{(t)} - G^T \tag{10}$$

where the decision maker is compared to the best fixed arm retrospectively. We will denote this arm by $j^* = \arg\max_j \sum_{t=1}^{T} r_j^{(t)}$ and its regret by G^*.

3.2. Accelerating AdaBoost using multi-armed bandits

The general idea is to partition the base classifier space $\mathcal{H}$ into (not necessarily disjunct) subsets $\mathcal{G} = \{\mathcal{H}_1, \ldots, \mathcal{H}_M\}$ and use MABs to learn the usefulness of the subsets. Each arm represents a subset, so, in each iteration, the bandit algorithm selects a subset. The base learner then finds the best base classifier *in the subset* (instead of searching through the *whole space* $\mathcal{H}$), and returns a reward based on this optimal base learner.

The upper bound (4) along with (8) suggest the use of $r_j^{(t)} = -\log\sqrt{1-{\gamma_{\mathcal{H}_j}^{(t)}}^2}$ as a reward function where $\gamma_{\mathcal{H}_j}^{(t)}$ is the edge (7) of the classifier chosen by the base learner from $\mathcal{H}_j$ in th tth iteration. Since $\gamma \in [0,1]$, this reward is unbounded. To overcome this technical problem, we cap $r_j^{(t)}$ by 1 and define the reward as

$$r_j^{(t)} = \min\left(1, -\log\sqrt{1-{\gamma_{\mathcal{H}_j}^{(t)}}^2}\right). \tag{11}$$

This is equivalent of capping the edge $\gamma_{\mathcal{H}_j}^{(t)}$ at $\gamma_{\max} = 0.93$ which is rarely exceeded in practice, so this constraint does not affect the practical performance of the algorithm.

(Busa-Fekete & Kégl, 2009) used this setup recently in the stochastic settings (with a slightly different reward). There are many applications where stochastic bandits have been applied successfully in a non-stationary environment, e.g., in a performance tuning problem (De Mesmay et al., 2009) or in the SAT problem (Maturana et al., 2009). The UCB (Auer et al., 2002a) and the UCBV (Audibert et al., 2009) algorithms work well in practice for accelerating AdaBoost, but the mismatch between the inherently adversarial nature of AdaBoost (the edges $\gamma_{j^{(t)}}$ are deterministic) and the stochastic setup of UCB made it impossible to derive weak-to-strong-learning-type performance guarantees on AdaBoost, and so in (Busa-Fekete & Kégl, 2009) the connection between AdaBoost and bandits remained slightly heuristic.

In this paper we propose to use an adversarial MAB algorithm that belongs to the class of Exponentially Weighted Average Forecaster (EWAF) methods (Cesa-Bianchi & Lugosi, 2006). In general, an EWAF algorithm maintains a probability distribution $\mathbf{p}^{(t)}$ over the arms and draws a random arm from this distribution in each iteration. The probability value of an arm increases exponentially with the average of past rewards. In particular, we chose the Exp3.P algorithm (Auer et al., 2002b) (Figure 2) because the particular form of the high-probability bound on the weak regret allows us to derive a weak-to-strong-learning result for AdaBoost.MH. We call the combined algorithm AdaBoost.MH.Exp3.P.

Exp3.P(η, λ, T)

1 **for** $j \leftarrow 1$ **to** M ▷ *initialization*
2 $\quad \omega_j^{(1)} \leftarrow \exp\left(\frac{\eta\lambda}{3}\sqrt{\frac{T}{M}}\right)$
3 **for** $t \leftarrow 1$ **to** T
4 $\quad$ **for** $j \leftarrow 1$ **to** M
5 $\quad\quad p_j^{(t)} \leftarrow (1-\lambda)\frac{\omega_j^{(t)}}{\sum_{j'=1}^{M}\omega_{j'}^{(t)}} + \frac{\lambda}{M}$
6 $\quad j^{(t)} \leftarrow \text{Random}\left(p_1^{(t)}, \ldots, p_M^{(t)}\right)$
7 $\quad$ Receive reward $r_{j^{(t)}}^{(t)}$
8 $\quad$ **for** $j \leftarrow 1$ **to** M
9 $\quad\quad \widehat{r}_j^{(t)} \leftarrow \begin{cases} r_j^{(t)}/p_j^{(t)} & \text{if } j = j^{(t)} \\ 0 & \text{otherwise} \end{cases}$
10 $\quad\quad \omega_j^{(t+1)} \leftarrow \omega_j^{(t)} \exp\left(\frac{\lambda}{3M}\left(\widehat{r}_j^{(t)} + \frac{\eta}{p_j^{(t)}\sqrt{MT}}\right)\right)$

Figure 2. The pseudocode of the Exp3.P algorithm. $\eta > 0$ and $0 < \lambda \leq 1$ are "smoothing" parameters: the larger they are, the more uniform is the probability vector $\mathbf{p}^{(t)}$, and so the more the algorithm explores (vs. exploits). T is the number of iterations. Exp3.P sends $j^{(t)}$ (Line 6) to AdaBoost.MH, and receives its reward in Line 7 from AdaBoost.MH's base learner via (11).

3.3. A weak-to-strong-learning result using Exp3.P with AdaBoost.MH

A sufficient condition for an algorithm to be called boosting is that, given a Base learner which always returns a classifier $\mathbf{h}^{(t)}$ with edge $\gamma^{(t)} \geq \rho$ for given $\rho > 0$, it returns a strong classifier $\mathbf{f}^{(T)}$ with zero training error after a logarithmic number of iterations $T = O(\log n)$. It is well-known that AdaBoost.MH

satisfies this condition (Theorem 3 in (Schapire & Singer, 1999)). In the following theorem we prove a similar result for AdaBoost.MH.Exp3.P.

Theorem 1. *Let $\mathcal{H}$ be the class of base classifiers and $\mathcal{G} = \{\mathcal{H}_1, \ldots, \mathcal{H}_M\}$ be an arbitrary partitioning of $\mathcal{H}$. Suppose that there exists a subset $\mathcal{H}_{j^\dagger}$ in $\mathcal{G}$ and a constant $0 < \rho \leq \gamma_{\max}$ such that for any weighting over the training data set $\mathcal{D}$, the base learner returns a base classifier from $\mathcal{H}_{j^\dagger}$ with an edge $\gamma_{\mathcal{H}_{j^\dagger}} \geq \rho$. Then, with probability at least $1-\delta$, the training error $R\left(\mathbf{f}^{(T)}\right)$ of* AdaBoost.MH.Exp3.P *will become 0 after at most*

$$T = \max\left(\log^2 \frac{M}{\delta}, \left(\frac{4C}{\rho^2}\right)^4, \frac{4\log\left(n\sqrt{K-1}\right)}{\rho^2}\right) \quad (12)$$

iterations, where $C = \sqrt{32M} + \sqrt{27M\log M} + 16$, and the input parameters of AdaBoost.MH.Exp3.P *are set to*

$$\lambda = \min\left(\frac{3}{5}, 2\sqrt{\frac{3}{5}\frac{M\log M}{T}}\right), \eta = 2\sqrt{\log\frac{MT}{\delta}}. \quad (13)$$

Proof. The proof is based on Theorem 6.3 of (Auer et al., 2002b) where the weak regret (10) of Exp3.P is bounded from above by

$$4\sqrt{MT\log\frac{MT}{\delta}} + 4\sqrt{\frac{5}{3}MT\log M} + 8\log\frac{MT}{\delta}$$

with probability $1-\delta$. Since this bound does not depend on the number of data points n, it is relatively easy to obtain the most important third term of (12) that ensures the logarithmic dependence of T on n. The technical details of the proof are included as supplementary material. □

Remark 1 The fact that Theorem 1 provides a large probability bound does not affect the PAC strong learning property of the algorithm since T is sub-polynomial in $1/\delta$ (first term of (12)).

Remark 2 Our weak-learning condition is stronger than in the case of AdaBoost.MH: we require ρ-weak-learnability in the *best* subset $\mathcal{H}_{j^\dagger}$, as opposed to the whole base classifier set $\mathcal{H}$. In a certain sense, a smaller ρ (and hence a larger number of iterations in principle) is the price we pay for doing less work in each iteration. The three terms of (12) comprise an interesting interplay between the number of subsets M, the size of the subsets (in terms of the running time of the base learner), the quality of the subsets (in terms of ρ), and the number of iterations T. In principle, it should be possible to optimize these parameters so as to minimize the total running time of the algorithm (to achieve zero training error) in a similar framework to what was proposed by (Bottou & Bousquet, 2008). In practice, however such an optimization is limited since, on the one hand, the quality of the subsets is unknown beforehand, and on the other, the bound (12) is quite loose in an absolute sense.

Remark 3 Applying Theorem 6.3 of (Auer et al., 2002b) requires that we formally set λ and η to the values in (13). However, it is a general practice to validate these parameters. In our experiments we found that both of them should be set to relatively small values ($\lambda \approx 0.15, \eta \approx 0.3$), which accords with the suggestions of (Kocsis & Szepesvári, 2005).

3.4. Partitioning the base classifier set

In the case of simple decision stumps (9) the most natural partitioning is to assign a subset to each feature: $\mathcal{H}_j = \{\varphi_{j,b}(\mathbf{x}) : b \in \mathbb{R}\}$. All our experiments were carried out using this setup. One could think about making a coarser partitioning (more then one feature per subset), however, it makes no sense to split the features further since the computational time of finding the best threshold b on a feature is the same as that for evaluating a given stump on the data set.

The situation is more complicated in the case of trees or products. In principle, one could set up a non-disjoint partitioning where every subset of N or m features is assigned to an arm. This would make M very big (d^N or d^m). Theorem 1 is still valid but in practice the algorithm would spend all its time in the exploration phase, making it practically equivalent to LazyBoost. To overcome this problem, we followed the setup proposed by (Busa-Fekete & Kégl, 2009) in which trees and products are modeled as sequences of decisions over the smaller partitioning used for stumps. The algorithm performs very well in practice using this setup. Finding a more accurate theoretical framework for this model is an interesting future task.

4. Experiments

4.1. Synthetic datasets

The goal of these experiments was to verify whether the bandit-based algorithms can indeed find useful features on data sets where the usefulness of the features is entirely under our control. We used two baselines methods in each test: AdaBoost.MH with stumps (Full) and AdaBoost.MH that searches only a random subset of base classifiers (Random), equivalent to LazyBoost. We tested the methods on two binary classification tasks. In both tasks $\mathbf{x}$ are generated uniformly in the d-dimensional unit cube, and only the first J features are relevant in determining the label.

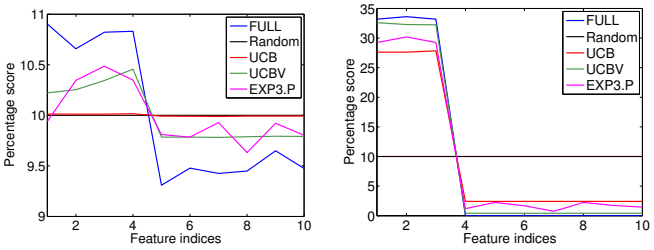
In the first task DIAGONAL(d, J, q), used by (Mease & Wyner, 2007), the classes are separated by a diagonal cut on the first J features, and labels are perturbed by a small random noise q. Formally,

$$P(Y = 1|\mathbf{x}) = q + (1 - 2q)\mathbb{I}\left\{\sum_{j=1}^{J} x^{(j)} > \frac{J}{2}\right\}.$$

In the second task CHESS(d, J, L), the labels are generated according to a chess board with L fields per dimension using the first J features. Formally,

$$y = \begin{cases} 1 & \text{if } \sum_{j=1}^{J} \lfloor Lx^{(j)} \rfloor \text{ is pair} \\ -1 & \text{otherwise} \end{cases}$$

We run the algorithms on DIAGONAL$(10, 4, 0.1)$ and CHESS$(10, 3, 3)$ with $n = 1000$ data points. The parameters of ADABOOST.MH.EXP3.P were set to $T = 10000$ and $\lambda = \eta = 0.3$. Figure 3 shows the percentage of the iterations when a certain feature was selected, averaged over 100 runs. The results are not very surprising: the bandit-based algorithms succeeded in identifying the useful features most of the time, and chose them almost as frequently as full ADABOOST.MH. The only surprise was the performance of UCB on the DIAGONAL task: it stuck in an exploration mode and failed to discover useful features.



(a) The DIAGONAL data set (b) The CHESS data set

Figure 3. The percentage score of the iterations when a certain feature was selected on (a) DIAGONAL$(10, 4, 0.1)$ and (b) CHESS$(10, 3, 3)$ by the different algorithms.

4.2. Benchmark datasets

We tested the proposed method on five benchmark datasets using the standard train/test cuts[2]. We compared ADABOOST.MH.EXP3.P (EXP3.P) with full ADABOOST.MH (FULL), ADABOOST.MH with random feature selection (RANDOM), and two stochastic-bandit-based techniques of (Busa-Fekete & Kégl, 2009) (UCB and UCBV). In each run, we validated the number of iterations T using 80%-20% simple validation. We smoothed the test error (5) on a window of $T/5$ iterations to obtain $\overline{R}^{(T)} = \frac{5}{T}\sum_{t=4T/5}^{6T/5} R\big(\mathbf{f}^{(t)}\big)$, and minimized $\overline{R}^{(T)}$ in T. The advantage of this approach is that this estimate is more robust in terms of random fluctuations than the raw error $R\big(\mathbf{f}^{(T)}\big)$. In the case of trees and products, we also validated the hyperparameters N and m using full ADABOOST.MH, and used the validated values in all the algorithms. Finally, for ADABOOST.MH.EXP3.P we also validated the hyperparameters λ and η using a grid search in the range of $[0, 0.6]$ with a resolution of 0.05.

Table 1 shows the test errors on the benchmark datasets. The first observation is that full ADABOOST.MH wins most of the time although the differences are rather small. The few cases where RANDOM or UCB/UCBV/EXP3.P beats full ADABOOST.MH could be explained by statistical fluctuations or the regularization effect of randomization. The second is that EXP3.P seems slightly better than UCB/UCBV although the differences are even smaller.

Our main goal was not to beat full ADABOOST.MH in test performance, but to improve its computational complexity. So we were not so much interested in the *asymptotic* test errors but rather the speed by which *acceptable* test errors are reached. As databases become larger, it is not unimaginable that certain algorithms cannot be run with their statistically optimal hyperparameters (T in our case) because of computational limits, so managing *underfitting* (an algorithmic problem) is more important than managing overfitting (a statistical problem) (Bottou & Bousquet, 2008). To illustrate how the algorithms behave in terms of computational complexity, we plotted the smoothed test error curves $\overline{R}^{(T)}$ versus *time* for selected experiments in Figure 4. The improvement in terms of computational time over full ADABOOST.MH is often close to an order of magnitude (or two in the case of MNIST), and RANDOM is also significantly worse than the bandit-based techniques. The results also indicate that EXP3.P also wins over UCB/UCBV most of the time, and it is never significantly worse than the stochastic-bandit-based approach.

5. Discussion and Conclusions

In this paper we introduced a MAB based approach for accelerating ADABOOST. Using an adversarial setup, we were able to prove a high-probability weak-to-strong-learning bound, a result that was lacking from previous, more heuristic approaches using stochastic bandits. At the same time, in practice, the proposed

[2]The data sets (selected based on their wide usage and their large sizes) are available at `yann.lecun.com/exdb/mnist` (MNIST), `www.kernel-machines.org/data.html` (USPS), and `www.ics.uci.edu/~mlearn/MLRepository.html` (letter, pendigit, isolet).

Table 1. Smoothed test error percentage scores ($100\overline{R}^{(T)}$) on benchmark datasets.

learner \ data set	MNIST	USPS	UCI pendigit	UCI isolet	UCI letter
Stump/FULL	7.56	6.28	4.96	4.52	14.57
RANDOM	6.94	6.13	4.57	4.43	14.60
UCB	7.07	6.03	4.63	4.30	14.65
UCBV	7.02	6.08	4.63	4.43	14.77
EXP3.P	6.96	6.08	4.63	4.43	14.62
Product (m)	3	3	2	6	10
FULL	1.26	4.37	1.89	3.86	2.31
RANDOM	1.73	4.58	1.83	3.59	2.25
UCB	1.82	4.43	1.80	3.53	2.25
UCBV	1.85	4.53	1.92	3.53	2.20
EXP3.P	1.70	4.53	1.94	3.40	2.30
Tree (N)	17	15	19	8	38
FULL	1.52	4.87	2.12	3.85	2.48
RANDOM	1.82	5.13	2.06	3.78	3.08
UCB	2.30	4.98	2.09	3.85	3.10
UCBV	3.06	5.18	2.20	3.78	3.08
EXP3.P	1.85	4.93	2.06	3.72	3.02

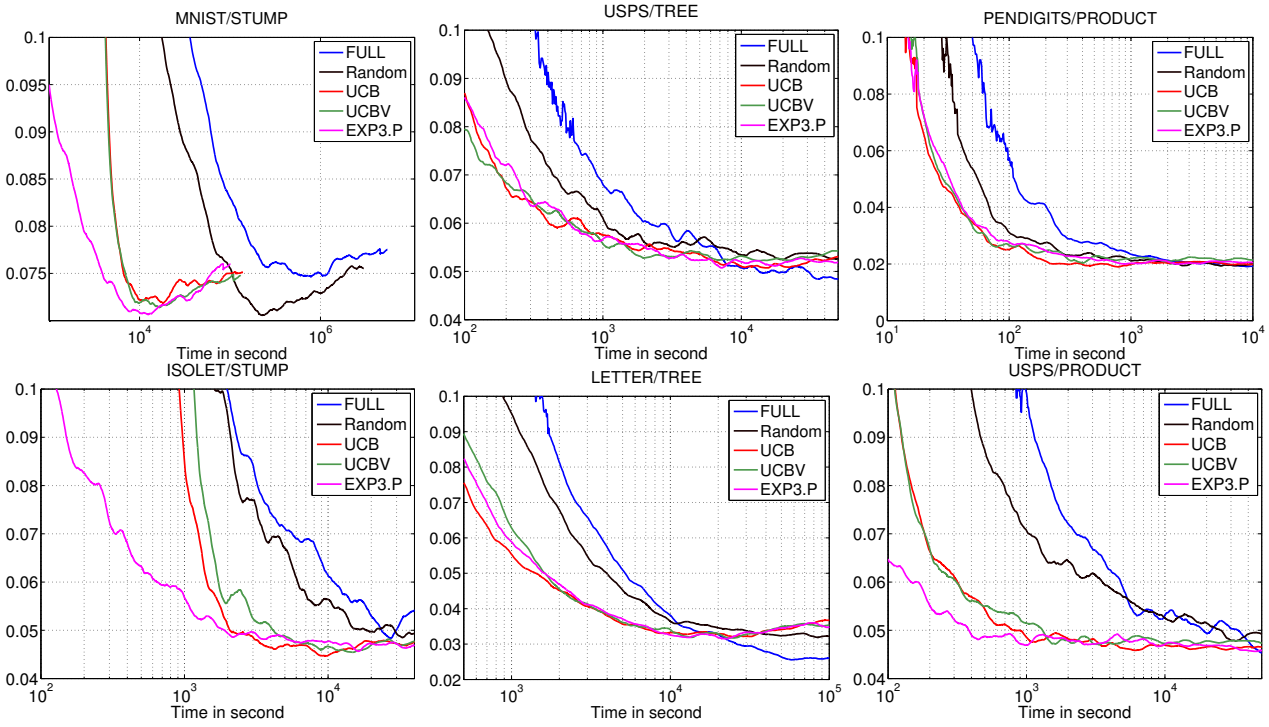


Figure 4. Smoothed test errors $\overline{R}^{(T)}$ vs. total computational time.

algorithm performs at least as well as its stochastic counterpart in terms of both its generalization error and its computational efficiency. Although it seems that the asymptotic test error of ADABOOST with full search is hard to beat if we have enough computational resources, in *large scale learning* (recently described in a seminal paper by (Bottou & Bousquet, 2008)), where we stay in an *underfitting* regime so fast op-

timization becomes more important than asymptotic statistical optimality, our MAB-optimized AdaBoost has its place.

To keep the project within manageable limits, we consciously did not explore all the possible avenues in testing all possible on-line optimization algorithms. There are two particular ideas that we would like to investigate in the near future. First, multi-armed bandits assume a *stateless* system, whereas AdaBoost has a natural state descriptor: the weight matrix $\mathbf{W}^{(t)}$. In this setup a Markov decision process would be more natural given that we can find a particular algorithm that can exploit a high dimensional, continuous state space and, at the same time, compete with bandits in computational complexity and convergence speed. The second avenue to explore is within the MAB framework: there are several successful applications of AdaBoost where the number of features is huge but the feature space has an a-priori known structure (for example, Haar filters in image processing (Viola & Jones, 2004) or word sequences in natural language processing (Escudero et al., 2000)). Classical bandits are hard to use in these cases but more recent MAB algorithms were developed for exactly this scenario (Coquelin & Munos, 2007).

Acknowledgements

We would like to thank Peter Auer, András György, and Csaba Szepesvári for their useful comments. This work was supported by the ANR-07-JCJC-0052 grant of the French National Research Agency.

References

Audibert, J.-Y., Munos, R., and Szepesvári, Cs. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.*, 410(19):1876–1902, 2009.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R.E. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of the 36th Ann. Symp. on Foundations of Computer Science*, pp. 322–331, 1995.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002a.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R.E. The non-stochastic multi-armed bandit problem. *SIAM J. on Computing*, 32(1):48–77, 2002b.

Bottou, L. and Bousquet, O. The tradeoffs of large scale learning. In *NIPS*, volume 20, pp. 161–168, 2008.

Bradley, J.K. and Schapire, R.E. FilterBoost: Regression and classification on large datasets. In *NIPS*, volume 20, 2008.

Busa-Fekete, R. and Kégl, B. Accelerating AdaBoost using UCB. In *KDDCup 2009 (JMLR W&CP)*, volume 7, pp. 111–122, 2009.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Coquelin, P-A. and Munos, R. Bandit algorithms for tree search. In *UAI*, volume 23, 2007.

De Mesmay, F., Rimmel, A., Voronenko, Y., and Püschel, M. Bandit-based optimization on graphs with application to library performance tuning. In *ICML*, volume 26, pp. 729–736, 2009.

Escudero, G., Màrquez, L., and Rigau, G. Boosting applied to word sense disambiguation. In *ECML*, volume 11, pp. 129–141, 2000.

Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. of Comp. and Sys. Sci.*, 55:119–139, 1997.

Kégl, B. and Busa-Fekete, R. Boosting products of base classifiers. In *ICML*, volume 26, pp. 497–504, 2009.

Kocsis, L. and Szepesvári, Cs. Reduced-Variance Payoff Estimation in Adversarial Bandit Problems. In *ECML Work. on Reinforcement Learning in Non-Stationary Environments*, 2005.

Maturana, J., Fialho, Á., Saubion, F., Schoenauer, M., and Sebag, M. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *IEEE ICEC*, pp. 365–372, 2009.

Mease, D. and Wyner, A. Evidence contrary to the statistical view of boosting. *JMLR*, 9:131–156, 2007.

Quinlan, J. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

Schapire, R. E. and Singer, Y. Improved boosting algorithms using confidence-rated predictions. *Mach. Learn.*, 37(3):297–336, 1999.

Viola, P. and Jones, M. Robust real-time face detection. *Int. J. of Comp. Vis.*, 57:137–154, 2004.