Boosting products of base classifiers

Balázs Kégl[1] BALAZS.KEGL@GMAIL.COM
Róbert Busa-Fekete[1,2] BUSAROBI@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 show how to boost products of simple base learners. Similarly to trees, we call the base learner as a subroutine but in an iterative rather than recursive fashion. The main advantage of the proposed method is its simplicity and computational efficiency. On benchmark datasets, our boosted products of decision stumps clearly outperform boosted trees, and on the MNIST dataset the algorithm achieves the second best result among no-domain-knowledge algorithms after deep belief nets. As a second contribution, we present an improved base learner for nominal features and show that boosting the product of two of these new subset indicator base learners solves the maximum margin matrix factorization problem used to formalize the collaborative filtering task. On a small benchmark dataset, we get experimental results comparable to the semi-definite-programming-based solution but at a much lower computational cost.

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 an incremental fashion by adding simple classifiers to a pool, and using their weighted "vote" to determine the final classification. ADABOOST was later extended to multi-class classification problems (Schapire & Singer, 1999). Although various other attempts have been made to deal directly with the multi-class setting, ADABOOST.MH has become the gold standard of multi-class boosting due to its simplicity and versatility.

In practice, boosting simple learners like decision stumps has often been found to be sub-optimal. Indeed, the class of linear combinations of stumps is not a universal approximator (e.g., the XOR problem is not in the class), and it is easy to find even toy examples for which there is a mismatch between the function class and the data distribution. The most common solution for overcoming this problem is to use trees as base learners that call the simple base learner in a recursive fashion to partition the input space. As the main contribution of this paper, we propose and explore another possibility of learning *products* of simple base learners. The main advantage of the proposed method is its simplicity and computational efficiency. Similarly to trees, we call the base learner as a subroutine but in an iterative rather than recursive fashion. In the optimization loop we fix all but one of the base classifier terms, temporarily re-label the points, and call the base learner using the "virtual" labels. On benchmark datasets, ADABOOST.MH using products of decision stumps definitely outperforms boosted trees. As a highlight, note that the 1.26% test error on the MNIST dataset is the best reported error rate among no-domain-knowledge algorithms after Hinton and Salakhutdinov's (2007) deep belief nets (1.00%); it is significantly better than the error rates of support vector machines (1.4%), randomly initialized back-propagation neural nets (1.6%), or boosted trees (1.53%).

As a second contribution, we propose an improved base learner for nominal features. In the standard approach a nominal base learner selects only one value to test in each boosting iteration. Here we show how we can optimize a nominal base learner that acts as a subset indicator using the same computational effort. We will also show that boosting the product of two indicator base learners solves the maximum margin matrix factorization (MMMF) problem (Srebro et al., 2005) used to formalize and solve the collaborative filtering problem. We carried out experiments on a small benchmark dataset and compared the method to the semi-definite-programming-based (SDP) solution (Srebro et al., 2005). The results are slightly worse, but the computational effort needed to produce them was an order of magnitude smaller than in the case of SDP MMMF. Since ADABOOST.MH scales linearly with the data size, the approach has a greater prospective on large collaborative filtering problems.

Appearing in *Proceedings of the 26^{th} International Conference on Machine Learning*, Montreal, Canada, 2009.

The paper is organized as follows. First we describe the algorithm in Section 2, then we present the experimental results in Section 3. Lastly we draw some brief conclusions and make a few pertinent remarks.

2. The algorithm

In Section 2.1 we first describe ADABOOST.MH to the extent that is necessary to understand our contribution. For further details we refer the reader to the original paper (Schapire & Singer, 1999). Section 2.2 contains details on the base learning, including our contribution on the subset indicator base learner. Then we describe the new product base learner in Section 2.3.

2.1. ADABOOST.MH

For the formal description let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the $n \times d$ *observation matrix*, where the elements $x_i^{(j)}$ of the d-dimensional observation vectors $\mathbf{x}_i \in \mathcal{X}$ are either real numbers, or they come from an unordered set of cardinality M. In this latter case, without loss of generality, we will assume that $x_i^{(j)} \in \mathcal{I} = \{1, \ldots, M\}$. The column vectors of $\mathbf{X}$ will be denoted by $\mathbf{x}^{(j)}$, $j = 1, \ldots, d$. We are also given a *label matrix* $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ 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 $\mathbf{y}_1$ 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 $\ell(\mathbf{x}_i)$.

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*

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

by minimizing its upper bound (the exponential margin loss)

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

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*

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

Using the weight update formula of line 5 (Figure 1), it can be shown that

$$R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \prod_{t=1}^{T} E\big(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\big),$$

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*)

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

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

2.2. Learning the base classifier

In this section we will first describe the details of multi-class base learning, then we will briefly define decision stumps used on numerical features, and finally we will introduce a new technique for optimizing a nominal base learner. There are two generic learning schemes available that are used to transform a scalar binary base classifier $\varphi : \mathbb{R} \to \{+1, -1\}$ (or $\varphi : \mathcal{I} \to \{+1, -1\}$) into a (real) vector-valued base classifier $\mathbf{h}$. In *discrete* ADABOOST.MH, $\mathbf{h}(\mathbf{x})$ is represented by

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

where $\alpha \in \mathbb{R}^+$ is the *base coefficient* and $\mathbf{v} \in \{+1, -1\}^K$ is the *vote vector*, whereas in *real* ADABOOST.MH the vote vector $\mathbf{v}$ is real-valued and $\alpha = 1$ (therefore it can be omitted). In discrete ADABOOST.MH it can be shown that for a given φ, (3) is minimized by

$$v_\ell = \begin{cases} 1 & \text{if } \mu_{\ell+} > \mu_{\ell-} \\ -1 & \text{otherwise,} \end{cases} \quad \ell = 1, \ldots, K, \tag{4}$$

and

$$\alpha = \frac{1}{2} \ln \frac{\sum_{\ell=1}^{K} \Big(\mu_{\ell+} \mathbb{I}\{v_\ell = +1\} + \mu_{\ell-} \mathbb{I}\{v_\ell = -1\}\Big)}{\sum_{\ell=1}^{K} \Big(\mu_{\ell-} \mathbb{I}\{v_\ell = +1\} + \mu_{\ell+} \mathbb{I}\{v_\ell = -1\}\Big)},$$

ADABOOST.MH$(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}, \text{BASE}(\cdot,\cdot,\cdot), T)$

1 **for** $t \leftarrow 1$ **to** T
2 $\quad\big(\alpha^{(t)}, \mathbf{v}^{(t)}, \varphi^{(t)}(\cdot)\big) \leftarrow \text{BASE}\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$
3 $\quad\mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot)$
4 $\quad$**for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K
5 $\quad\quad w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \dfrac{\exp\left(-h_\ell^{(t)}(\mathbf{x}_i) y_{i,\ell}\right)}{\sum_{i'=1}^{n}\sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} \exp\left(-h_{\ell'}^{(t)}(\mathbf{x}_{i'}) y_{i',\ell'}\right)}$
6 **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, $\text{BASE}(\cdot,\cdot,\cdot)$ 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.

where

$$\mu_{\ell-} = \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\left\{\varphi(\mathbf{x}_i) \neq y_{i,\ell}\right\}, \quad \ell = 1, \ldots, K \tag{5}$$

is the *weighted per-class error rate*, and

$$\mu_{\ell+} = \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\left\{\varphi(\mathbf{x}_i) = y_{i,\ell}\right\} \quad \ell = 1, \ldots, K. \tag{6}$$

The goal of the scalar base learner is to return a φ that maximizes the *edge*

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

In real ADABOOST.MH, φ is found in the same way by maximizing the edge (7) using the discrete votes (4), but then we set

$$v_\ell = \frac{1}{2} \ln \frac{\mu_{\ell+}}{\mu_{\ell-}}, \quad \ell = 1, \ldots, K.$$

This procedure finds the minimizer of (3) in discrete ADABOOST.MH. In real ADABOOST.MH it is suboptimal (corresponding to the two-stage greedy functional gradient descent approach of (Mason et al., 2000)); however, it ensures that $E^{(t)}(\mathbf{h}) < 1$, so the algorithm will converge.

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

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

where j is the index of the selected feature and b is the decision threshold. If the features are pre-ordered before the first boosting iteration, a decision stump maximizing the edge (7) can be found very efficiently in $\Theta(ndK)$ time (making the total running time $\Theta\big(nd(\log n + KT)\big)$).

To handle nominal features $x_i^{(j)} \in \mathcal{I}^{(j)} = \{1, \ldots, M^{(j)}\}$ with a possibly large number of values $M^{(j)}$, we introduce a novel *subset indicator* base learner (Figure 2). The standard procedure (Schapire & Singer, 1999) is equivalent to using stumps on one-hot encoded nominal features: only *one* of the nominal values $\iota \in \mathcal{I}^{(j)}$ is tested in each iteration using base learners of the form

$$\varphi_{j,\iota}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} = \iota, \\ -1 & \text{otherwise.} \end{cases} \tag{8}$$

Optimizing this *selector* base learner takes $\Theta\left(ndK + K\Sigma\right)$ time, where $\Sigma = \sum_{j=1}^{d} M^{(j)}$ is the number of all the different feature values. Hence, to obtain a strong learner that potentially uses (tests) $\Omega(M^{(j)})$ values for each feature, we will need $\Omega\big((ndK + K\Sigma)\Sigma\big)$ operations, which can be prohibitive if Σ is large. On the other hand, we saw that we can optimize a base learner that makes a decision on *all* values of $\mathcal{I}^{(j)}$ in each iteration, using essentially the same number $\Theta\left(ndK + K\Sigma\right)$ of operations per iteration as the selector base learner. Formally, we consider base learners of the form

$$\varphi_{j,\mathbf{u}}(\mathbf{x}) = \mathbf{u}_{x^{(j)}}, \tag{9}$$

where $\mathbf{u}$ is a vector over the index set $\mathcal{I}^{(j)}$. In the base learner, we initialize $\mathbf{u}$ randomly, and then we optimize $\mathbf{v}$ and $\mathbf{u}$ in an alternating iteration until convergence is achieved. Convergence is guaranteed since $E\big(\alpha\mathbf{v}\varphi_{j,\mathbf{u}}, \mathbf{W}\big)$ is bounded from below by zero, it must decrease in each iteration, and the decrease is bounded away from zero because the weights $w_{i,\ell}$ are bounded away from zero. In practice INDICATORBASE always stopped after a few iterations. There is no guarantee that the global minimum of the base objective will be found, but this is not a problem in ADABOOST.MH: the boosting loop can continue with any base learner $\mathbf{h}$ with $E\big(\mathbf{h}, \mathbf{W}^{(t)}\big) < 1$.

Another advantage of the new indicator base learner (besides being faster) is that the vote vector $\mathbf{v}$ is shared among

INDICATORBASE($\mathbf{X}, \mathbf{Y}, \mathbf{W}$)

for $j \leftarrow 1$ **to** d ▷ *all (nominal) features*
  $(\alpha_j, \mathbf{v}_j, \mathbf{u}_j) \leftarrow$ BESTINDICATOR$(\mathbf{x}^{(j)}, \mathbf{Y}, \mathbf{W}, \mathcal{I}^{(j)})$ ▷ $\mathbf{x}^{(j)} \triangleq \left(x_1^{(j)}, \ldots, x_n^{(j)}\right)$
$j^* \leftarrow \arg\min_j E\left(\alpha_j \mathbf{v}_j \varphi_{j,\mathbf{u}_j}, \mathbf{W}\right)$
return $\left(\alpha_{j^*}, \mathbf{v}_{j^*}, \varphi_{j^*,\mathbf{u}_{j^*}}(\cdot)\right)$

BESTINDICATOR($\mathbf{x}, \mathbf{Y}, \mathbf{W}, \mathcal{I}$)

1 **for** $\iota \in \mathcal{I}$
2 **for** $\ell \leftarrow 1$ **to** K
3 $\gamma^+_{\iota,\ell} \leftarrow \gamma^-_{\iota,\ell} \leftarrow 0$
4 $u_\iota \leftarrow$ RANDOM(± 1)
5 **for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K
6 **if** $w_{i,\ell} y_{i,\ell} > 0$ **then**
7 $\gamma^+_{x_i,\ell} \leftarrow \gamma^+_{x_i,\ell} + w_{i,\ell} y_{i,\ell}$
8 **else**
9 $\gamma^-_{x_i,\ell} \leftarrow \gamma^-_{x_i,\ell} - w_{i,\ell} y_{i,\ell}$
10 $\alpha \leftarrow 0$, $\mathbf{v} \leftarrow \mathbf{0}$
11 **while** TRUE
12 $\alpha_{\text{prev}} \leftarrow \alpha$, $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{v}$ ▷ *save current optimal* α *and* $\mathbf{v}$
13 **for** $\ell \leftarrow 1$ **to** K
14 $v_\ell \leftarrow \text{sign}\left(\sum_{\iota \in \mathcal{I}} (\gamma^+_{\iota,\ell} - \gamma^-_{\iota,\ell}) u_\iota\right)$ or $v_\ell \leftarrow \frac{1}{2} \ln \frac{\sum_{\iota \in \mathcal{I}} \left(\gamma^+_{\iota,\ell} \mathbb{I}\{u_\iota > 0\} + \gamma^-_{\iota,\ell} \mathbb{I}\{u_\iota < 0\}\right)}{\sum_{\iota \in \mathcal{I}} \left(\gamma^-_{\iota,\ell} \mathbb{I}\{u_\iota > 0\} + \gamma^+_{\iota,\ell} \mathbb{I}\{u_\iota < 0\}\right)}$
15 $\alpha \leftarrow \frac{1}{2} \ln \frac{\sum_{\ell=1}^K \sum_{\iota \in \mathcal{I}} \left(\gamma^+_{\iota,\ell} \mathbb{I}\{u_\iota > 0\} + \gamma^-_{\iota,\ell} \mathbb{I}\{u_\iota < 0\}\right)}{\sum_{\ell=1}^K \sum_{\iota \in \mathcal{I}} \left(\gamma^-_{\iota,\ell} \mathbb{I}\{u_\iota > 0\} + \gamma^+_{\iota,\ell} \mathbb{I}\{u_\iota < 0\}\right)}$ or $\alpha \leftarrow 1$
16 **if** $E\left(\alpha \mathbf{v} \varphi_{\mathbf{u}}, \mathbf{W}\right) \geq E\left(\alpha_{\text{prev}} \mathbf{v}_{\text{prev}} \varphi_{\mathbf{u}}, \mathbf{W}\right)$ **then**
17 **return** $(\alpha_{\text{prev}}, \mathbf{v}_{\text{prev}}, \mathbf{u})$
18 $\alpha_{\text{prev}} \leftarrow \alpha$, $\mathbf{u}_{\text{prev}} \leftarrow \mathbf{u}$ ▷ *save current optimal* α *and* $\mathbf{u}$
19 **for** $\iota \in \mathcal{I}$
20 $u_\iota \leftarrow \text{sign}\left(\sum_{\ell=1}^K (\gamma^+_{\iota,\ell} - \gamma^-_{\iota,\ell}) v_\ell\right)$ or $u_\iota \leftarrow \frac{1}{2} \ln \frac{\sum_{\ell=1}^K \left(\gamma^+_{\iota,\ell} \mathbb{I}\{v_\ell > 0\} + \gamma^-_{\iota,\ell} \mathbb{I}\{v_\ell < 0\}\right)}{\sum_{\ell=1}^K \left(\gamma^-_{\iota,\ell} \mathbb{I}\{v_\ell > 0\} + \gamma^+_{\iota,\ell} \mathbb{I}\{v_\ell < 0\}\right)}$
21 $\alpha \leftarrow \frac{1}{2} \ln \frac{\sum_{\iota \in \mathcal{I}} \sum_{\ell=1}^K \left(\gamma^+_{\iota,\ell} \mathbb{I}\{v_\ell > 0\} + \gamma^-_{\iota,\ell} \mathbb{I}\{v_\ell < 0\}\right)}{\sum_{\iota \in \mathcal{I}} \sum_{\ell=1}^K \left(\gamma^-_{\iota,\ell} \mathbb{I}\{v_\ell > 0\} + \gamma^+_{\iota,\ell} \mathbb{I}\{v_\ell < 0\}\right)}$ or $\alpha \leftarrow 1$
22 **if** $E\left(\alpha \mathbf{v} \varphi_{\mathbf{u}}, \mathbf{W}\right) \geq E\left(\alpha_{\text{prev}} \mathbf{v} \varphi_{\mathbf{u}_{\text{prev}}}, \mathbf{W}\right)$ **then**
23 **return** $(\alpha_{\text{prev}}, \mathbf{v}, \mathbf{u}_{\text{prev}})$

Figure 2. The pseudocode of the subset indicator base learner. Between lines 12 and 17 we optimize $\mathbf{v}$ and α given a fixed $\mathbf{u}$, and between lines 18 and 23 we optimize $\mathbf{u}$ and α given a fixed $\mathbf{v}$. The two alternatives in lines 14, 15, 20, and 21 refer to discrete and real ADABOOST.MH, respectively.

the selectors while in the standard learner we learn a vote vector for each selector. This means that the capacity of the base classifier is considerably reduced (compared to the sum of Σ selectors), suggesting that the algorithm is less susceptible to overfitting. Sharing the vote vector may also help to "pick up" dependencies between feature values in the case where there is a structure in the feature space (e.g., using discrete ADABOOST.MH, each indicator $\varphi_{j,\mathbf{u}}$ clusters the feature values $\mathcal{I}^{(j)}$ into two groups).

In a broader sense, the proposed learner is related to Cohen's (1996) model which allows set-valued features as well as the common real-valued and nominal features. The main difference is that in (Cohen, 1996) it is the *features of the observations* that can take a subset of a large set of possible values but the decision functions are still *selectors*

of the form (8), whereas what we propose here are subset-indicator *decision functions* (9). In another related algorithm (SLIPPER), Cohen and Singer (1999) build rules that are *conjunctions of selectors* (8) using a growing/pruning procedure. The selectors in the same base rule can use different *features*, whereas INDICATORBASE acts on different *values* of *one* feature. Although superficially similar to our method, both approaches use different base classifiers and different algorithms to learn the classifiers. On the other hand, they can be incorporated into our approach with just a few modifications to INDICATORBASE.

2.3. The product base learner

The goal of the procedure is to optimize 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, and the number of terms m is a complexity parameter that should be validated (similarly to the size of decision trees). We suppose that $\varphi_j(\cdot)$ is a simple base classifier equipped with a base learner that returns the optimal α, $\mathbf{v}_j$, and $\varphi_j(\cdot)$, as described in Section 2.2. To maximize the edge (7), we follow a simple iterative approach (Figure 3). In each iteration we fix each base learner except for one φ_j, and maximize the edge with respect to φ_j. Because of the product form of the edge (7), we can carry out this maximization by simply calling the base learner of φ_j with "virtual" labels defined as the sign of the product of the real labels and the outputs of the remaining base learners (line 7 in Figure 3). The procedure is guaranteed to converge since in each batch of m iterations at least one "virtual" sign $y'_{i,\ell}$ must change (otherwise the base learner in line 8 returns the same set of m classifiers, and we have equality in line 9) and the number of different sign vectors is finite. As with subset indicator base learners, we found that in practice PRODUCTBASE always returns after a few iterations.

REMARK 1: PRODUCTS OF STUMPS. In experiments (see Section 3) we found that ADABOOST.MH with products of stumps achieves excellent test results on benchmark data sets, for which, at this point, we can only provide a partial explanation. First, it is easy to see that the algorithm solves the XOR problem: a product of two stumps can obviously implement the XOR function. As an extension, the product of d stumps can implement the parity function on any subsequence of the (binary or thresholded real-valued) feature vector up to d elements. It can also be shown that the class of sums of products of d stumps is a universal approximator if and only if the observation space $\mathcal{X}$ is at most d-dimensional.[2] Second, it is clear that the class of sums of products of stumps is a subclass of decision trees: there are constraints on how homogeneous regions are created, and each base learner term of the product has a "global" contribution. Nevertheless, we also found that *one* product of stumps does not have enough capacity to achieve a low test error even if m is set to infinity: PRODUCTBASE underfits the data, probably due to the greediness of the optimization loop. Hence, unlike trees, products cannot be used as standalone strong learners. On the other hand, boosting products is less susceptible to overfitting than boosting trees, for which we also provide some experimental evidence.

REMARK 2: PRODUCTS OF SUBSET INDICATORS. Boosting products of indicator base classifiers (9) is a solution of maximum margin matrix factorization (MMMF) (Srebro et al., 2005), used to formalize and solve the collaborative filtering problem. Taking the simple case when the observations consist of two index features $x^{(1)} \in \mathcal{I}^{(1)}$ and $x^{(2)} \in \mathcal{I}^{(2)}$ (for example, movie id's and user id's), and the classification is binary (for example, $y = +1$ if user$_{x^{(1)}}$ liked movie$_{x^{(2)}}$ and $y = -1$ if he or she disliked it), individual base classifiers will take the form

$$\mathbf{h}^{(t)}_{\mathbf{u}^{(t)},\mathbf{z}^{(t)}}\left(x^{(1)}, x^{(2)}\right) = \alpha^{(t)} u^{(t)}_{x^{(1)}} z^{(t)}_{x^{(2)}},$$

where $\mathbf{u}^{(t)}$ and $\mathbf{z}^{(t)}$ are vectors over $\mathcal{I}^{(1)}$ and $\mathcal{I}^{(2)}$, respectively. The strong classifier $\mathbf{f}$ can then be used to classify *every* pair of $\mathcal{I}^{(1)} \times \mathcal{I}^{(2)}$. The resulting $M^{(1)} \times M^{(2)}$ matrix can be expressed as the product of two matrices, an $M^{(1)} \times T$ "user feature" matrix containing the column vectors $\sqrt{\alpha^{(t)}}\mathbf{u}^{(t)}$, and a $T \times M^{(2)}$ "movie feature" matrix containing the row vectors $\sqrt{\alpha^{(t)}}\mathbf{z}^{(t)}$. The maximization of the exponential margin loss (1) leads to a large margin solution, and all the relevant results on the generalization error (e.g., (Schapire et al., 1998)) apply immediately. Finding two low-rank (or otherwise low-complexity) matrices whose product produces large margins on the training data is the exact goal of MMMF (Srebro et al., 2005). In a certain sense, boosting products of two index learners is related to the semi-definite-programming-based solution of MMMF as the classical binary ADABOOST is related to support vector machines.

The algorithm can be applied directly to multi-valued preferences by adding the optimization of the vote vectors $\mathbf{v}^{(t)}$. In this case, the preference prediction matrix becomes a hyper-matrix with vector-valued elements. The formulation also permits one to boost products of more than two indices (or to use two or more times a base learner on the same index) which goes beyond the matrix-decomposition-based formulation of the collaborative filtering problem. It is also quite straightforward to combine collaborative features with "traditional" numerical or nominal covariates by allowing mixed products. The main shortcoming of the

[2] Here we shall only give an outline of the constructive proof: $\mathbf{f}$ can be set to an arbitrary value on any hypercube in $\mathbb{R}^d$, independently of the rest of the space by carefully selecting and weighting products of stumps placed at the corners of the hypercube.

PRODUCTBASE($\mathbf{X}, \mathbf{Y}, \mathbf{W}, \text{BASE}(\cdot,\cdot,\cdot), m$)

1 **for** $j \leftarrow 1$ **to** m
2 $\quad\varphi_j(\cdot) \leftarrow 1, \mathbf{v}_j \leftarrow \mathbf{1}$ ▷ *terms are initialized to the constant* 1 *function*
3 $\alpha \leftarrow 1$
4 **while** TRUE **for** $j \leftarrow 1$ **to** m
5 $\quad\varphi^*(\cdot) \leftarrow \prod_{j'=1}^{m} \varphi_{j'}(\cdot), \mathbf{v}^* \leftarrow \prod_{j'=1}^{m} \mathbf{v}_{j'}, \alpha^* \leftarrow \alpha$ ▷ *save current optimal classifier*
6 $\quad$**for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K
7 $\quad\quad y'_{i,\ell} \leftarrow \text{sign}\left(y_{i,\ell} \dfrac{v^*_\ell \varphi^*(\mathbf{x}_i)}{v_{j\ell}\varphi_j(\mathbf{x}_i)}\right)$ ▷ *"virtual" labels*
8 $\quad\left(\alpha, \mathbf{v}_j, \varphi_j(\cdot)\right) \leftarrow \text{BASE}(\mathbf{X}, \mathbf{Y}', \mathbf{W})$
9 $\quad$**if** $E\left(\alpha \prod_{j'=1}^{m} \mathbf{v}_{j'}\varphi_{j'}, \mathbf{W}\right) \geq E\left(\alpha^*\mathbf{v}^*\varphi^*, \mathbf{W}\right)$ **then**
10 $\quad\quad$**return** $\left(\alpha^*, \mathbf{v}^*, \varphi^*(\cdot)\right)$

Figure 3. The pseudocode of the product base learner. m is the number of base classifier terms to be optimized. The vote vectors are multiplied element-wise in lines 5 and 9. The $\text{sign}(\cdot)$ operator in line 7 can be omitted in the case of discrete ADABOOST.MH.

method at this point is that the exponential margin loss (1) is designed for multi-class or multi-label classification, and it is sub-optimal if the preferences come from an ordered set. One of our aims for future study is to examine the possibility of using product learners in regression or ranking.

3. Experiments

To test the algorithm[3], we carried out two sets of experiments. In the first we boosted products of decision stumps on five benchmark datasets[4] using the standard train/test cuts. We used discrete ADABOOST.MH with multi-class initial weights (2). We found no overfitting whatsoever in terms of the number of boosting iterations T (Figure 4), so instead of validating T and performing an early stopping, we decided to run the algorithm for a long time after convergence ($T = 10^5$ in each experiment), and measure the average test error $R\left(\mathbf{f}^{(T)}\right)$ on the last $T/2$ iterations. The advantage of this approach is that this estimate is more robust in terms of random fluctuations after convergence than the error at a given iteration. It is also a pessimistic estimate of the error when there is a slight overfitting (since the average is always an upper bound of the minimum). For the two image datasets we also report results using ADABOOST.MH with stumps on Haar filters (Viola & Jones, 2004).[5]

We conducted experiments using a range of values for the number of terms m for which we report the test errors in Table 1. The optimal number of terms (error rate in bold face) was selected by a 80%-20% simple validation on the training set. We compared our algorithm with two versions of boosted decision trees. We implemented a tree learner similar to ID3 in our AdaBoost.MH version which uses the boosting's base objective (3) (instead of the entropy-based criterion) to learn stumps in a recursive fashion. Since this algorithm was more prone to overfitting (last plot in Figure 4), we validated both the number of leaves N and the number of boosting iterations T. Table 1 indicates that, with one exception, boosting products is significantly better than boosting trees. We also tested the Weka (Witten & Frank, 2005) implementation of boosted trees. However the comparison is slightly unfair for two reasons: first, the Weka implementation uses ADABOOST.M1 which is known to be inferior to ADABOOST.MH for multi-class boosting, and second, the high computational complexity did not allow us a complete exploration of the hyperparameter space, so we cannot claim that we have found the optimal tree size and number of iterations T. Nevertheless, these results provide a reasonable idea about what the novice user's experience with AdaBoost.

On an absolute scale the results are on a par with the state-of-the-art results reported in the literature on the same data sets. In particular, they are the best results obtained by any ensemble method. We would especially like to underline the 1.26% test error on the MNIST dataset, which is the best reported error rate among generic classification al-

[3]The multi-platform C++ source code is available at `http://users.web.lal.in2p3.fr/kegl/research/multiboost`.

[4]The data sets are available at `yann.lecun.com/exdb/mnist` (MNIST), `www.kernel-machines.org/data.html` (USPS), `www.ics.uci.edu/~mlearn/MLRepository.html` (letter, pendigit, isolet), and `www.cs.umn.edu/Research/GroupLens` (MovieLens).

[5]It is not the subject of this paper, however, we give some details for reproducibility. We used five filter types ("bw" and "bwb", vertical and horizontal, and four-field checkerboard). Due to the large number of possible filters (order of 10^5), we selected just the best of a random 100 in each boosting iteration.

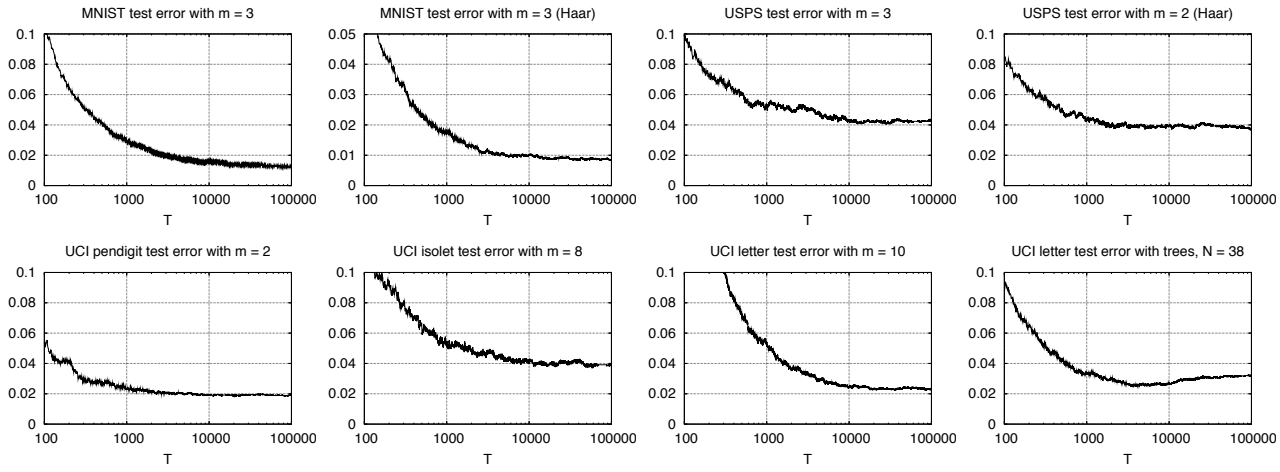

Figure 4. The "winning" learning curves on the five benchmark datasets indicate no overfitting in terms of the number of boosting iterations T when products are used. With a tree base learner (last plot), on the other hand, we do observe slight overfitting sometimes.

learner \ data set	MNIST	USPS	UCI pendigit	UCI isolet	UCI letter
Stump ($m = 1$)	7.71 (0.05)	6.48 (0.05)	4.97 (0.00)	4.88 (0.07)	14.74 (0.06)
Product / $m = 2$	1.56 (0.03)	4.92 (0.06)	**1.89 (0.02)**	3.91 (0.10)	3.65 (0.04)
Product / $m = 3$	**1.26 (0.02)**	**4.24 (0.04)**	2.07 (0.03)	3.97 (0.04)	2.71 (0.02)
Product / $m = 4$	1.38 (0.02)	4.72 (0.07)		3.95 (0.05)	2.54 (0.06)
Product / $m = 5$				3.87 (0.07)	2.41 (0.04)
Product / $m = 6$				**3.92 (0.08)**	2.40 (0.03)
Product / $m = 8$				3.89 (0.04)	2.38 (0.04)
Product / $m = 10$				4.11 (0.04)	**2.35 (0.04)**
Haar ($m = 1$) (Viola & Jones, 2004)	1.02 (0.02)	4.29 (0.05)			
Haar / Product / $m = 2$	0.84 (0.02)	**3.84 (0.07)**			
Haar / Product / $m = 3$	**0.87 (0.02)**	4.03 (0.06)			
Haar / Product / $m = 4$	0.90 (0.02)	4.05 (0.04)			
Tree	1.53 (0.02)	4.73 (0.04)	2.14 (0.04)	3.69 (0.04)	2.62 (0.04)
Haar / Tree	1.08 (0.02)	4.98 (0.05)			
AdaBoost.M1 / C4.5	4.05	5.98	2.66	4.81	2.88

Table 1. Test error percentages $100\frac{2}{T}\sum_{t=T/2}^{T} R\big(\mathbf{f}^{(t)}\big)$ on benchmark datasets using discrete ADABOOST.MH with products of stumps. The results with the optimal number of terms m selected by 80%-20% simple validation on the training set are shown in bold. Tree and Haar/Tree are "in-house" implementations of a tree base learner. In this case both T and the number of leaves N were validated. For AdaBoost.M1/C4.5 we used the Weka implementation of it.

gorithms[6] after Hinton, G. E. and Salakhutdinov's (2007) deep belief nets (1.00%); it is significantly better than the error rates of support vector machines (1.4%) and randomly initialized two-layer back-propagation neural nets using cross-entropy loss (1.6%).

It seems also quite surprising that we were able to improve on boosting stumps over the feature space generated by Haar filters. Boosting stumps over Haar filters is already one of the best semi-generic method[7], achieving similar error rates to convolutional neural nets (LeCun et al., 1998). Boosting these feature extractors also outperforms boosting stumps and even boosting products of stumps (on MNIST). The feature space has a large dimension (of order 10^5) and the Haar filters are well-adapted to natural images, so one would expect that a simple linear combination over this space can achieve the best results. While this intuition is reaffirmed when using *trees* over the Haar space, we were genuinely surprised to see that using the *product* of a small number of Haar filters as a base learner can significantly outperform boosting single Haar filters.

[6]Algorithms that do not make explicit use of the fact that the observation vectors represent images.

[7]Algorithms that do make explicit use of the fact that the observation vectors represent images, but not of the fact that the images depict characters.

set	WLRA (Marlin, 2004)		MMMF SDP (Srebro et al., 2005)		ADABOOST	
1	rank 2	57.5	max-norm, $C = 0.0012$	**56.2**	$T = 7275$	56.3
2	rank 2	56.2	trace norm, $C = 0.24$	55.2	$T = 9940$	**54.5**
3	rank 1	54.3	max-norm, $C = 0.0012$	**52.7**	$T = 475$	53.9
4	rank 2	55.3	max-norm, $C = 0.0012$	**55.0**	$T = 9515$	56.6
Avg		55.8		**54.8**		55.3

Table 2. Test error percentages on the MovieLens dataset using the experimental setup of (Srebro et al., 2005). The errors in columns 3 and 5 were taken directly from (Srebro et al., 2005).

In the second set of experiments we tested the algorithm on a small subset of the MovieLens collaborative filtering database using the same experimental settings as Srebro et al. (2005): the data is divided into four sets, for each of the four test sets the algorithms are trained and validated on the remaining three in a 3-fold cross validation. The two best of the three learners are selected and tested on the hold-out test set. In the case of ADABOOST, we validated the real/discrete version, the weight initialization, the number of terms m and the number of iterations T. In each experiment, using real ADABOOST.MH with uniform weight initialization $w_{i,\ell}^{(1)} = 1/(nK)$ and $m = 2$ proved to be the best approach. The results in Table 2 place this approach between WLRA (Marlin, 2004) and semi-definite-programming-based MMMF (Srebro et al., 2005). The computational effort needed to produce the results was an order of magnitude smaller than in the case of SDP MMMF (minutes vs. hours). Since ADABOOST.MH scales linearly with the data size, this approach has a greater prospective on large collaborative filtering problems.

4. Conclusions

In this paper we described and tested ADABOOST.MH with products of simple base learners, introduced a new nominal base learner, and demonstrated that boosting the products of this new base learners solves the MMMF problem. We found that boosting products outperforms boosting trees, it is less prone to overfitting, and it is even able to improve boosting stumps in such complex feature spaces where boosting stumps is expected to be the state-of-the-art. The main issues that we foresee to attack in the near future are 1) the extension of the algorithm to regression and ranking, and 2) investigating the initializing of the subset indicators in a more sophisticated way than the random initialization used here.

Acknowledgments

This work was supported by the ANR-07-JCJC-0052 grant of the French National Research Agency.

References

Cohen, W. (1996). Learning trees and rules with set-valued features. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 709–716).

Cohen, W., & Singer, Y. (1999). A simple, fast, and effective rule learner. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 335–342).

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*, 2278–2324.

Marlin, B. (2004). Collaborative filtering: a machine learning perspective. Master's thesis, University of Toronto.

Mason, L., Bartlett, P., Baxter, J., & Frean, M. (2000). Boosting algorithms as gradient descent. *Advances in Neural Information Processing Systems* (pp. 512–518). The MIT Press.

Salakhutdinov, R., & Hinton, G. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. *International Conference on Artificial Intelligence and Statistics* (pp. 412–419).

Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, *26*, 1651–1686.

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, *37*, 297–336.

Srebro, N., Rennie, J. D. M., & Jaakkola, T. (2005). Maximum margin matrix factorization. *Advances in Neural Information Processing Systems* (pp. 1329–1336). The MIT Press.

Viola, P., & Jones, M. (2004). Robust real-time face detection. *International Journal of Computer Vision*, *57*, 137–154.

Witten, I., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.