The return of ADABoost.MH: multi-class Hamming trees

Balázs Kégl
LAL/LRI, University of Paris-Sud, CNRS, 91898 Orsay, France

Abstract

Within the framework of ADABoost.MH, we propose to train vector-valued decision trees to optimize the multi-class edge without reducing the multi-class problem to $K$ binary one-against-all classifications. The key element of the method is a vector-valued decision stump, factorized into an input-independent vector of length $K$ and label-independent scalar classifier. At inner tree nodes, the label-dependent vector is discarded and the binary classifier can be used for partitioning the input space into two regions. The algorithm retains the conceptual elegance, power, and computational efficiency of binary ADABoost. In experiments it is on par with support vector machines and with the best existing multi-class boosting algorithm AOSOLOGITBOOST, and it is significantly better than other known implementations of ADABoost.MH.

1. Introduction

ADABoost (Freund & Schapire, 1997) is one of the most influential supervised learning algorithms of the last twenty years. It has inspired learning theoretical developments and also provided a simple and easily interpretable modeling tool that proved to be successful in many applications (Caruana & Niculescu-Mizil, 2006). It is especially the method of choice when any-time solutions are required on large data sets, so it has been one of the most successful techniques in recent large-scale classification and ranking challenges (Dror et al., 2009; Chapelle et al., 2011).

The original ADABoost paper of Freund and Schapire (Freund & Schapire, 1997), besides defining binary ADABoost, also described two multi-class extensions, ADABoost.M1 and ADABoost.M2. Both required a quite strong performance from the base learners, partly defeating the purpose of boosting, and saw limited practical success. The breakthrough came with Schapire and Singer’s seminal paper (Schapire & Singer, 1999), which proposed, among other interesting extensions, ADABoost.MH. The main idea of this approach is to use vector-valued base classifiers to build a multi-class discriminant function of $K$ outputs (for $K$-class classification). The weight vector, which plays a crucial role in binary ADABoost, is replaced by a weight matrix over instances and labels. The simplest implementation of the concept is to use $K$ independent one-against-all classifiers in which base classifiers are only loosely connected through the common normalization of the weight matrix. This setup works well with single decision stumps, but in most of the practical problems, boosting stumps is suboptimal compared to boosting more complex base classifiers such as trees. Technically, it is possible to build $K$ one-against-all binary decision trees in each iteration, but this approach, for one reason or another, has not produced state-of-the-art results. As a consequence, several recent papers concentrate on replacing the boosting objective and the engine that optimizes this objective (Li, 2009a;b; Zhu et al., 2009; Sun et al., 2012; Mukherjee & Schapire, 2013).

The main misconception that comes back in several papers is that ADABoost.MH has to train $K$ parallel one-against-all classifiers in each iteration. It turns out that the original setup is more general. For example, staying within the classical ADABoost.MH framework, Kégl & Busa-Fekete (2009) trained products of simple classifiers and obtained state-of-the-art results on several data sets. In this paper, we describe multi-class Hamming trees, another base learner that optimizes the multi-class edge without reducing the problem to $K$ binary classifications. The key idea is to factorize general vector-valued classifiers into an input-independent vector of length $K$, and label-independent scalar classifier. It turns out that optimizing such base classifiers using decision stumps as the scalar component is almost as simple as optimizing simple binary stumps on binary data. The technique can be intuitively understood as optimizing a binary cut and an output code at the same time. The main consequence of the setup is that now it is easy to build trees of these classifiers by simply discarding the label-dependent vector and using the binary classifier for partitioning the input space into two regions.
The algorithm retains the conceptual elegance, power, and computational efficiency of binary ADABoost. Algorithmically it cannot fail (the edge is always positive) and in practice it almost never overfits. Inheriting the flexibility of ADABoost.MH, it can be applied directly (without any modification) to multi-label and multi-task classification. In experiments (carried out using an open source package of Benbouzid et al. (2012) for reproducibility) we found that ADABoost.MH with Hamming trees performs on par with the best existing multiclass boosting algorithm AOSLOGITBoost (Sun et al., 2012) and with support vector machines (SVMs; Boser et al. 1992). It is also significantly better than other known implementations of ADABoost.MH (Zhu et al., 2009; Mukherjee & Schapire, 2013).

The paper is organized as follows. In Section 2 we give the formal multi-class setup used in the paper and ADABoost.MH, and show how to train factorized base learners in general. The algorithm to build Hamming trees is described in Section 3. Experiments are described in Section 4 before a brief conclusion in Section 5.

2. ADABoost.MH

In this section we first introduce the general multi-class learning setup (Section 2.1), then we describe ADABoost.MH in detail (Section 2.2). We proceed by explaining the general requirements for base learning in ADABoost.MH, and introduce the notion of the factorized vector-valued base learner (Section 2.4). Finally, we explain the general objective for factorized base learners and the algorithmic setup to optimize that objective. (Section 2.5).

2.1. The multi-class setup: single-label and multi-label/multi-task

For the formal description of ADABoost.MH, let the training data be \( D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \), where \( x_i \in \mathbb{R}^d \) are observation vectors, and \( y_i \in \{\pm 1\}^K \) are label vectors. Sometimes we will use the notion of an \( n \times d \) observation matrix of \( X = (x_1, \ldots, x_n) \) and an \( n \times K \) label matrix \( Y = (y_1, \ldots, y_n) \) instead of the set of pairs \( D \). In multi-class classification, the single label \( \ell(x) \) of the observation \( x \) comes from a finite set. Without loss of generality, we will suppose that \( \ell \in \mathcal{L} = \{1, \ldots, K\} \). The label vector \( y \) is a one-hot representation of the correct class: the \( \ell \)th element of \( y \) will be 1 and all the other elements will be -1. Besides expressing faithfully the architecture of a multi-class neural network or multi-class ADABoost,

\[ w_\ell = \begin{cases} 1 \quad & \text{if } \ell = \ell(x) \text{ (i.e., if } y_\ell = 1) \, , \\ \frac{1}{2(K-1)} \quad & \text{otherwise } \text{(i.e., if } y_\ell = -1) \, . \end{cases} \]

The idea behind this scheme is that it will create \( K \) well-balanced one-against-all binary classification problems: if

\[ \ell(x) \text{ is true and } \ell \text{ the label index of } x, \text{ respectively. For emphasizing the distinction between multi-class and multi-label classification, we will use the term single-label for the classical multi-class setup, and reserve multi-class to situations when we talk about the three setups in general.}

The goal of learning is to infer a vector-valued multi-class discriminant function \( f : \mathcal{X} \rightarrow \mathbb{R}^K \). The single-label output of the algorithm is then \( \ell(x) = \arg \max_\ell f_\ell(x) \). The classical measure of the performance of the multi-class discriminant function \( f \) is the single-label one-loss

\[ L_1(f, (x, \ell)) = \mathbb{I}\{\ell \neq \ell(x)\} \, , \]

Another, perhaps more comprehensive, way to measure the performance of \( f \) is by computing the weighted Hamming loss \( L_w(f, (x, y), w) = \sum_{\ell=1}^K w_\ell \mathbb{I}\{\text{sign}(f_\ell(x)) \neq y_\ell\} \) where \( w = [w_\ell] \) is an \( \mathbb{R}^K \)-valued “user-defined” weight vector over labels. The corresponding empirical risk (training error) is

\[ \hat{R}_w(f, W) = \frac{1}{n} \sum_{i=1}^n \sum_{\ell=1}^K w_\ell \mathbb{I}\{\text{sign}(f_\ell(x_i)) \neq y_{i,\ell}\} \, , \]

where \( W = [w_{i,\ell}] \) is an \( n \times k \) weight matrix over data points and labels.

In the multi-label/multi-task setup, when, for example, it is equally important to predict that a song is “folk” as predicting that it is sung by a woman, the Hamming loss with uniform weights \( w_\ell = 1/K, \ell = 1, \ldots, K \) is a natural measure of performance: it represents the uniform error rate of missing any class sign \( y_\ell \) of a given observation \( x \). In single-label classification, \( w \) is usually set asymmetrically to

\[ w_\ell = \frac{1}{2} \quad \text{if } \ell (x) \text{ (i.e., if } y_\ell = 1) \, , \quad \text{otherwise } \text{(i.e., if } y_\ell = -1) \, . \]

\[ 1 \text{We will use bold capitals X for matrices, bold small letters } x_i \text{ and } x_{i,j} \text{ for its row and column vectors, respectively, and italic for its elements } x_{i,j} \text{.} \]

\[ ^2\text{Instead of the original notation of (Schapire & Singer, 1999) where both x and } \ell \text{ are inputs of a function } f(x, \ell) \text{ outputting a single real-valued score, we use the notation } f(x) = (f_1(x), \ldots, f_K(x)) \text{ since we feel it expresses better that x is (in general) continuous and } \ell \text{ is a discrete index.} \]

\[ ^3\text{The indicator function } \mathbb{I}\{A\} \text{ is 1 if its argument } A \text{ is true and 0 otherwise.} \]
we start with a balanced single-label multi-class problem, that is, if each of the \( K \) classes have \( n/K \) examples in \( D \), then for each class \( \ell \), the sum of the weights of the positive examples in the column \( w_{\cdot,\ell} \) of the weight matrix \( W \) will be equal to the sum of the weights of the negative examples. Note that both schemes boil down to the classical uniform weighting in binary classification.

### 2.2. AdaBoost.MH

The goal of the AdaBoost.MH algorithm (Schapire & Singer 1999; Figure 1) is to return a vector-valued discriminant function \( f^{(T)} : \mathbb{R}^d \to \mathbb{R}^K \) with a small Hamming loss \( \hat{R}_h(f, W) \) (2) by minimizing the weighted multi-class exponential margin-based error

\[
\hat{R}_\text{exp}(f^{(T)}, W) = \frac{1}{n} \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp \left( -f^{(T)}_{\ell}(x_i) y_{i,\ell} \right).
\]

Since \( \exp(-\rho) \geq 1 \{ \rho < 0 \} \), (4) upper bounds the Hamming loss \( \hat{R}_h(f^{(T)}, W) \) (2). AdaBoost.MH builds the final discriminant function \( f^{(T)}(x) = \sum_{t=1}^{T} h^{(t)}(x) \) as a sum of \( T \) base classifiers \( h^{(t)} : \mathcal{X} \to \mathbb{R}^K \), returned by a base learner algorithm \( \text{Base}(X, Y, W^{(t)}) \) in each iteration \( t \).

#### AdaBoost.MH(X, Y, W, Base(·, ·, ·), T)

1. \( W^{(1)} \leftarrow \frac{1}{n} W \)
2. for \( t \leftarrow 1 \) to \( T \)
   3. \( (\alpha^{(t)}, v^{(t)}, \varphi^{(t)}(\cdot)) \leftarrow \text{Base}(X, Y, W^{(t)}) \)
   4. \( h^{(t)}(\cdot) \leftarrow \alpha^{(t)} v^{(t)} \varphi^{(t)}(\cdot) \)
   5. for \( i \leftarrow 1 \) to \( n \) for \( \ell \leftarrow 1 \) to \( K \)
     6. \( w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} e^{h^{(t)}_{\ell}(x_i) y_{i,\ell}} \)
     \( \sum_{\ell'=1}^{K} w_{i,\ell'}^{(t)} e^{-h^{(t)}_{\ell'}(x_i) y_{i,\ell'}} \) \( Z(h^{(t)}, W^{(t)}) \)
3. return \( f^{(T)}(\cdot) = \sum_{t=1}^{T} h^{(t)}(\cdot) \)

Figure 1. The pseudocode of the AdaBoost.MH algorithm with factorized base classifiers (17). \( X \) is the \( n \times d \) observation matrix, \( Y \) is the \( n \times K \) label matrix, \( W \) is the user-defined weight matrix used in the definition of the weighted Hamming error (2) and the weighted exponential margin-based error (4), \( \text{Base}(\cdot, \cdot, \cdot) \) is the base learner algorithm, and \( T \) is the number of iterations. \( \alpha^{(t)} \) is the base coefficient, \( \varphi^{(t)}(\cdot) \) is the vote vector, \( \varphi^{(t)}(\cdot) \) is the scalar base (weak) classifier, \( h^{(t)}(\cdot) \) is the vector-valued base classifier, and \( f^{(T)}(\cdot) \) is the final (strong) discriminant function.

### 2.3. Algorithmic convergence of AdaBoost.MH

In this section we derive the algorithmic convergence of the training error of AdaBoost.MH, and show the relationship of AdaBoost.MH and binary AdaBoost. None of these results are new, they were derived by Freund & Schapire (1997) and Schapire & Singer (1999), but they help understanding the objective of base learning. Readers not interested in these technical details can skip the section after (5). The main novelty of the paper is an efficient method of how (5) can be minimized; the rest of this section describes why (5) is the base objective.

The following derivation shows that

\[
\hat{R}_\text{exp}(f^{(T)}, W) = \prod_{t=1}^{T} Z(h^{(t)}, W^{(t)}),
\]

where

\[
Z(h, W^{(t)}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp \left( -h_{\ell, y_{i,\ell}} \right)
\]

is the base objective. This means that the goal of the base learner in iteration \( t \) is to minimize \( Z(h, W^{(t)}) \) in \( h \). Minimizing the base objective (5) in each iteration is therefore equivalent to minimizing the weighted multi-class exponential margin-based error (4) in an iterative greedy fashion. Indeed,

\[
\hat{R}_\text{exp}(f^{(T)}, W) = \frac{1}{n} \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp \left( -f^{(T)}_{\ell}(y_{i,\ell}) \right)
\]

\[
= \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp \left( -f^{(T)}_{\ell}(y_{i,\ell}) \right)
\]

\[
= Z(h^{(1)}, W^{(1)}) \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp \left( -f^{(T)}_{\ell}(y_{i,\ell}) \right)
\]

\[
= \prod_{t=1}^{T} Z(h^{(t)}, W^{(t)}) \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp \left( -f^{(T)}_{\ell}(y_{i,\ell}) \right)
\]

(8)

(9)

(10)

(11)

In (6) we use the definition of the weighted exponential margin-based error (4). (7) follows from the weight initialization of line 1. In (8)-(9) we repeatedly apply the weight
Multi-class Hamming trees

update formula of line 6. In (10) we use the definition of
the strong classifier in line 7. Finally, (11) follows from
the fact that, by lines 1 and 6, the weight matrix \( W^{(1)} \) remains
normalized
\[
\sum_{i=1}^{n} \sum_{t=1}^{K} w_{i,t}^{(t)} = 1
\]
in each iteration \( t \).

In binary ADABoost (Figure 2) with binary base clas-
sifiers \( h(x) \in \{ \pm 1 \} \) and coefficients \( \alpha, (5) \) simplifies to

\[
Z(h, \alpha) = \sum_{i=1}^{n} w_i \exp(-\alpha h(x_i) y_i) = \sum_{i=1}^{n} w_i \mathbb{I}\{h(x_i) = y_i\} \exp(-\alpha) + \sum_{i=1}^{n} w_i \mathbb{I}\{h(x_i) \neq y_i\} \exp(\alpha) = (1 - \epsilon) \exp(-\alpha) + \epsilon \exp(\alpha),
\]

(12)

where

\[
\epsilon = \epsilon(h, w) = \sum_{i=1}^{n} w_i \mathbb{I}\{h(x_i) \neq y_i\}
\]
is the weighted error of the base classifier \( h \). It is easy to
see that for a given \( h \), \( Z(h, \alpha) \) is minimized by

\[
\alpha = \frac{1}{2} \log \frac{1 - \epsilon}{\epsilon} = \frac{1}{2} \log \frac{1 + \gamma}{1 - \gamma},
\]

(13)

where

\[
\gamma = \gamma(h, w) = 1 - 2\epsilon = \sum_{i=1}^{n} w_i h(x_i) y_i
\]
is the so called edge of the base classifier \( h(x_i) \). Plugging
(13) back into (12), we have

\[
Z(h, \alpha) = (1 - \epsilon) \sqrt{\frac{\epsilon}{1 - \epsilon}} + \epsilon \sqrt{\frac{1 - \epsilon}{\epsilon}} = 2 \sqrt{\epsilon(1 - \epsilon)} = \sqrt{1 - \gamma^2}.
\]

Minimizing \( Z(h, \alpha) \) is therefore equivalent to maximizing
the edge \( \gamma \) or minimizing the weighted error \( \epsilon \). The chain
of minimizing the exponential error \( \hat{R}_{\text{exp}}(f^{(T)}) \) to
minimizing the base objective \( Z(h, \alpha) \) to minimizing the weighted
error \( \epsilon(h, w) \) to setting \( \alpha \) to (13) explains the formulas of
binary ADABoost that might have been arbitrary-looking
at the first sight. It is also easy to see that if

\[
\gamma^{(t)} \triangleq \gamma(h^{(t)}, W^{(t)}) \geq \delta > 0
\]
(or, equivalently, if \( \epsilon^{(t)} < \frac{1}{2} - \frac{\delta}{2} \), the so-called weak-
learning condition), then the binary margin-based exponential error can be bounded from above by

\[
\hat{R}_{\text{exp}}(f^{(T)}) \leq \sqrt{1 - \delta^2} \leq \exp\left(-\frac{\delta^2 T}{2}\right),
\]

(16)

where the second inequality follows from \( 1 - x \leq \exp(-x) \). This means that \( \hat{R}_{\text{exp}}(f^{(T)}) \) becomes smaller than \( \frac{1}{n} \) after at most

\[
T^* = \left\lceil \frac{2 \log n}{\delta^2} \right\rceil + 1
\]

iterations. Since \( \hat{R}_{\text{exp}}(f^{(T)}) \) is an upper bound of the training
error \( \hat{R}_t(f^{(T)}) \), and since the smallest non-zero value of \( \hat{R}_t(f^{(T)}) \) is \( \frac{1}{n} \), after \( T^* \) iterations \( f^{(T)} \) cannot commit
any error on the data set \( D \). To summarize, if the base clas-
sifiers \( h^{(t)} \) are slightly better than a random guess, the final
classifier \( f^{(T)} \) has zero training error in a number of steps
that is logarithmic in the size \( n \) of the data set \( D \).

**ADABoost** (\( D_n, \text{Base}(\cdot, \cdot), T \))

1. \( w^{(1)} \leftarrow (1/n, \ldots, 1/n) \) \( \triangleright \) initial weights
2. for \( t \leftarrow 1 \) to \( T \)
3. \( h^{(t)} \leftarrow \text{Base}(D_n, w^{(t)}) \) \( \triangleright \) base classifier
4. \( \epsilon^{(t)} \leftarrow \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\{h^{(t)}(x_i) \neq y_i\} \) \( \triangleright \) base error
5. \( \alpha^{(t)} \leftarrow \frac{1}{2} \ln \left( \frac{1 - \epsilon^{(t)}}{\epsilon^{(t)}} \right) \) \( \triangleright \) base coefficient
6. for \( i \leftarrow 1 \) to \( n \) \( \triangleright \) re-weighting the points
7. if \( h^{(t)}(x_i) \neq y_i \) then \( \triangleright \) error
8. \( w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\epsilon^{(t)}} \) \( \triangleright \) weight increases
9. else \( \triangleright \) correct classification
10. \( w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2(1 - \epsilon^{(t)})} \) \( \triangleright \) weight decreases
11. return \( f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot) \) \( \triangleright \) strong classifier

Figure 2. The pseudocode of binary ADABoost. \( D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \) is the training set, \( \text{Base}(\cdot, \cdot) \) is the base
learner, and \( T \) is the number of iterations.

In the general multi-class setup, base learning is more com-
plicated (although still elementary) so we devote a sepa-
rate section for the technical details (Section 2.4). But
even without understanding how \( h \) is found, it can be seen,
similarly to the binary case, that if \( Z(h, W^{(t)}) \leq \sqrt{1 - \delta^2} \), then the exponential error \( \hat{R}_{\text{exp}}(f^{(T)}, W) \) (4) is
also bounded above by (16). The smallest nonzero multi-class Hamming loss $\hat{R}_t(f, W)$ (2) is $w_{\text{min}}/n$ where

$$w_{\text{min}} = \min_{i, t : w_{i, t} \neq 0} w_{i, t}$$

is the smallest weight in the initial weight matrix $W$, so $\hat{R}_t(f, W)$ will be zero after at most

$$T^* = \left\lceil \frac{2 \log(n/w_{\text{min}}/\delta^2)}{\delta^2} \right\rceil + 1$$

iterations. If $W$ is set to (3), then $w_{\text{min}} = \frac{1}{2(K-1)}$, so

$$T^* = \left\lceil \frac{2 \log(2n/(K-1))}{\delta^2} \right\rceil + 1.$$ 

This result can be tightened: it can be shown (Schapire & Singer, 1999) that with the single-label initialization (3),

$$\hat{R}_t(f^{(T)}) \leq \sqrt{K-1} \prod_{t=1}^{T} Z(h(t), W^{(t)})$$

which implies the limit

$$T^* = \left\lceil \frac{2 \log (n\sqrt{K-1})}{\delta^2} \right\rceil + 1.$$ 

This result is another motivation for setting the weights to (3). Note, however, that these bounds have very few practical implications: on the one hand, the training error usually becomes 0 much before $T^*$, and on the other hand, ADABOOST is usually trained much longer after the training error becomes 0.

2.4. Base learning for ADABOOST.MH

In general, any vector-valued multi-class learning algorithm can be used to minimize (5). Although this goal is clearly defined in (Schapire & Singer, 1999), efficient base learning algorithms have never been described in detail. In most recent papers (Zhu et al., 2009; Mukherjee & Schapire, 2013) where ADABOOST.MH is used as baseline, the base learner is a classical single-label decision tree which has to be grown rather large to satisfy the weak-learning condition, and, when boosted, yields suboptimal results (Section 4). The reason why methods for learning multi-class $\{-1, 1\}^K$-valued base classifiers had not been developed before is because they have to be boosted: since they do not select a single label, they cannot be used as stand-alone multi-class classifiers.

Although it is not described in detail, it seems that the base classifier used in the original paper of Schapire & Singer (1999) is a vector of $K$ independent decision stumps $h(x) = (h_1(x), \ldots, h_K(x))$. These stumps cannot be used as node classifiers to grow decision trees since they do not define a single cut that depends only on the input (see Section 3 for a more detailed discussion). To overcome this problem, we propose base learning algorithms that factorize $h(x)$ into

$$h(x) = \alpha v \varphi(x),$$

where $\alpha \in \mathbb{R}^+$ is a positive real valued base coefficient, $\varphi(x)$ is a label-independent vote vector of length $K$, and $\varphi(x)$ is a label-independent scalar classifier. In discrete ADABOOST.MH, both components are binary, that is, $v \in \{-1\}^K$ and $\varphi(x) : \mathbb{R}^d \rightarrow \{-1\}$. The setup can be extended to real-valued classifiers $\varphi(x) : \mathbb{R}^d \rightarrow \mathbb{R}$, also known as confidence-rated classifiers, and it is also easy to make the vote vector $v$ real-valued (in which case, without the loss of generality, $\alpha$ would be set to 1). Both variants are known under the name of real ADABOOST.MH. Although there might be slight differences in the practical performance of real and discrete ADABOOST.MH, here we decided to stick to the discrete case for the sake of simplicity.

2.5. Casting the votes

To start, we show how to set $\alpha$ and $v$ in general if the scalar base classifier $\varphi$ is given. The intuitive semantics of (17) is the following. The binary classifier $\varphi(x)$ cuts the input space into a positive and a negative region. In binary classification this is the end of the story: we need $\varphi(x)$ to be well-correlated with the binary class labels $y$. In multi-class classification it is possible that $\varphi(x)$ correlates with some of the class labels $y_\ell$ and anti-correlates with others. This free choice is expressed by the binary “votes” $v_\ell \in \{-1\}$. We say that $\varphi(x)$ votes for class $\ell$ if $v_\ell = +1$ and it votes against class $\ell$ if $v_\ell = -1$. As in binary classification, $\alpha$ expresses the overall quality of the classifier $v_\ell \varphi(x)$: $\alpha$ is monotonically decreasing with respect to the weighted error of $v_\ell \varphi(x)$.

The advantage of the setup is that, given the binary classifier $\varphi(x)$, the optimal vote vector $v$ and the coefficient $\alpha$ can be set in an efficient way. To see this, first let us define the weighted per-class error rate

$$\mu_{\ell^-} = \sum_{i=1}^{n} w_{i, \ell} \mathbb{I} \{\varphi(x_i) \neq y_i, \ell\},$$

and the weighted per-class correct classification rate

$$\mu_{\ell^+} = \sum_{i=1}^{n} w_{i, \ell} \mathbb{I} \{\varphi(x_i) = y_i, \ell\}$$

for each class $\ell = 1, \ldots, K$. With this notation, $Z(h, W)$
Multi-class Hamming trees

simplifies to (see Appendix A)

\[ Z(h, W) = \frac{e^\alpha + e^{-\alpha}}{2} - \frac{e^\alpha - e^{-\alpha}}{2} \sum_{\ell=1}^{K} v_\ell (\mu_{\ell^+} - \mu_{\ell^-}). \]  

(20)

The quantity

\[ \gamma_\ell = v_\ell (\mu_{\ell^+} - \mu_{\ell^-}) = \sum_{i=1}^{n} w_{i,\ell} v_\ell \varphi(x_i)y_{i,\ell} \]  

(21)

is called the classwise edge of \( h(x) \). The full multi-class edge of the classifier is then

\[ \gamma = \gamma(v, \varphi, W) = \sum_{\ell=1}^{K} \gamma_\ell = \sum_{\ell=1}^{K} v_\ell (\mu_{\ell^+} - \mu_{\ell^-}) \]

(22)

\[ = \sum_{i=1}^{n} \varphi(x_i) \sum_{\ell=1}^{K} w_{i,\ell} v_\ell y_{i,\ell}. \]

With this notation, the classical (Freund & Schapire, 1997) binary coefficient \( \alpha \) is recovered: it is easy to see that (20) is minimized when

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

(23)

With this optimal coefficient, similarly to the binary case (15), (20) becomes \( Z(h, W) = \sqrt{1 - \gamma^2} \), so \( Z(h, W) \) is minimized when \( \gamma \) is maximized. From (22) it then follows that \( Z(h, W) \) is minimized if \( v_\ell \) agrees with the sign of \( (\mu_{\ell^+} - \mu_{\ell^-}) \), that is,

\[ v_\ell = \begin{cases} 1 & \text{if } \mu_{\ell^+} > \mu_{\ell^-} \\ -1 & \text{otherwise} \end{cases} \]  

(24)

for all classes \( \ell = 1, \ldots, K \).

The setup of factorized base classification (17) has another important consequence: the preservation of the weak-learning condition. Indeed, if \( \varphi(x) \) is slightly better than a coin toss, \( \gamma \) will be positive. Another way to look at it is to say that if a \((\varphi, v)\) combination has a negative edge \( \gamma < 0 \), then the edge of its complement (either \((-\varphi, v)\) or \((\varphi, -v)\)) will be \(-\gamma > 0\). To understand the significance of this, consider a classical single-label base classifier \( h : X \rightarrow L = \{-1, 1\} \), required by ADABoost.M1. Now if \( h(x) \) is slightly better than a coin toss, all one can hope for is an error rate slightly lower than \( \frac{1}{K-1} \) (which is equivalent to an edge slightly higher than \( \frac{2}{K} \)) to achieve the error of \( \frac{1}{2} \) (zero edge), required for continuing boosting, one has to come up with a base learner which is significantly better than a coin toss.

There is a long line of research on output codes similar in spirit to our setup. The boosting engine in these works is usually slightly different from ADABoost.MH since it attempts to optimize the multi-class hinge loss, but the factorization of the multi-class base classifier is similar to (17). Formally, the vote vector \( v \) in this framework is one column in an output code matrix. In the simplest setup this matrix is fixed beforehand by maximizing the error correcting capacity of the matrix (Dietterich & Bakiri, 1995; Allwein et al., 2001). A slightly better solution (Schapire, 1997; Guruswami & Sahai, 1999; Sun et al., 2005) is to wait until the given iteration to pick \( v \) by maximizing

\[ v^* = \arg \max_v \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \{ v_\ell \neq v_\ell(x_i) \}, \]

and then to choose the optimal binary classifier \( \varphi \) with this fixed vote (or code) vector \( v^* \) (although in practice it seems to be better to fix \( v \) to a random binary vector; Sun et al., 2005). The state of the art in this line of research is to iterate between optimizing \( \varphi \) with a fixed \( v \) and then picking the best \( v \) with a fixed \( \varphi \) (Li, 2006; Kégl & Busa-Fekete, 2009; Gao & Koller, 2011).

It turns out that if \( \varphi \) is a decision stump, exhaustive search for both the best binary cut (threshold) and the best vote vector can be carried out using one single sweep in \( \Theta(nK) \) time. The algorithm is a simple extension of the classical binary decision stump learner; for the sake of completeness, we provide the pseudocode in Appendix B. The computational efficiency of this learning algorithm combined with the factorized form (17) of the classifier allows us to build multiclass Hamming trees in an efficient manner, circumventing the problem of global maximization of the edge with respect to \( \varphi \) and \( v \).

3. Hamming trees

Classification trees (Quinlan, 1986) have been widely used for multivariate classification since the 80s. They are especially efficient when used as base learners in ADABoost (Caruana & Niculescu-Mizil, 2006; Quinlan, 1996). Their main disadvantage is their variance with respect to the training data, but when averaged over \( T \) different runs, this problem largely disappears. The most commonly used tree learner is C4.5 of Quinlan (1993). Whereas this tree implementation is a perfect choice for binary ADABoost, it is suboptimal for ADABoost.MH since it outputs a single-label classifier with no guarantee of a positive multi-class edge (22). Although this problem can be solved in practice by building large trees, it seems that using these large single-class trees is suboptimal (Section 4).

The main technical difficulty of building trees out of generic \( \{\pm 1\}^K \)-valued multi-class classifiers \( h(x) \) is that they do not necessarily implement a binary cut \( x \mapsto \{\pm 1\} \).
and partitioning the data into all the possibly $2^K$ children at a tree node leads to rapid overfitting. Factorizing the multi-class classifier $h(x)$ into an input-independent vote vector $v$ and a label-independent binary classifier $\varphi(x)$ as in (17) solves this problem. Base classifiers are trained as usual at each new tree leaf. In case this leaf remains a leaf, the full classifier $h_j(x)$ is used for instances $x$ that arrive to this leaf. If it becomes an inner node, the vote vector $v$ is discarded, and the partitioning of the data set is based solely on the binary classifier $\varphi(x)$. An advantage of this formalization is that we can use any multi-class base classifier of the form (17) for the tree cuts, so the Hamming tree algorithm can be considered as a “meta learner” which can be used on the top of any factorized base learner.

Formally, a binary classification tree with $N$ inner nodes ($N + 1$ leaves) consists of a list of $N$ base classifiers $\mathbf{h} = (h_1, \ldots, h_N)$ of the form $h_j(x) = v_j \varphi_j(x)$ and two index lists $I = (I_1, \ldots, I_N)$ and $\mathbf{r} = (r_1, \ldots, r_N)$ with $I, \mathbf{r} \in (\mathbb{N} \cup \{\text{NULL}\})^N$. $I_j$ and $r_j$ represent the indices of the left and right children of the $j$th node of the tree, respectively. The node classifier in the $j$th node is defined recursively as:

$$h_j(x) = \begin{cases} 
-v_j & \text{if } \varphi_j(x) = -1 \land I_j = \text{NULL} \\
v_j & \text{if } \varphi_j(x) = +1 \land r_j = \text{NULL} \\
h_{I_j}(x) & \text{if } \varphi_j(x) = -1 \land I_j \neq \text{NULL} \\
h_{r_j}(x) & \text{if } \varphi_j(x) = +1 \land r_j \neq \text{NULL}
\end{cases}$$

(25)

The final tree classifier $h_{3, I, r}(x) = \alpha h_j(x)$ itself is not a factorized classifier (17). In particular, $h_{3, I, r}(x)$ uses the local vote vectors $v_j$ determined by each leaf instead of a global vote vector. On the other hand, the coefficient $\alpha$ is unique and it is determined in the standard way

$$\alpha = \frac{1}{2} \log \frac{1 + \gamma(h_{I_1}, \mathbf{W})}{1 - \gamma(h_{I_1}, \mathbf{W})}$$

based on the edge of the tree classifier $h_{I_1}$. The local coefficients $\alpha_j$ returned by the base learners are discarded (along with the vote vectors in the inner nodes).

Finding the optimal $N$-inner-node tree is a difficult combinatorial problem. Most tree-building algorithms are therefore sub-optimal by construction. For AdaBoost this is not a problem: we can continue boosting as long as the edge is positive. Classification trees are usually built in a greedy manner: at each stage we try to cut all the current leaves $j$ by calling the base learner of the data points reaching the $j$th leaf, then select the best node to cut, convert the old leaf into an inner node, and add two new leaves. The difference between the different algorithms is in the way the best node is selected. Usually, we select the node that improves a gain function the most. In AdaBoost the natural gain is the edge (22) of the base classifier. Since the data set $(\mathbf{X}, \mathbf{Y})$ is different at each node, we include it explicitly in the argument of the full multi-class edge

$$\gamma(v, \varphi, \mathbf{X}, \mathbf{Y}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{t=1}^{K} \mathbb{I}\{x_i \in \mathbf{X}\} w_{i,t} v_t \varphi(x_i) y_{i,t}.$$

Note that in this definition we do not require that the weights of the selected points add up to 1. Also note that this gain function is additive on subsets of the original data set, so the local edges in the leaves add up to the edge of the full tree. This means that any improvement in the local edge directly translates to an improvement of the tree edge. This is a crucial property: it assures that the edge of the tree is always positive as long as the local edges in the inner nodes are positive, so any weak binary classifier $\phi(x)$ can be used to define the inner cuts and the leaves.

The basic operation when adding a tree node with a scalar binary classifier (cut) $\varphi$ is to separate the data matrices $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{W}$ according to the sign of classification $\varphi(x_i)$ for all $x_i \in \mathbf{X}$. The pseudocode is straightforward, but for the sake of completeness, we include it in the supplementary material (Appendix C, Figure 6).

Building a tree is usually described in a recursive way but we find the iterative procedure easier to explain, so our pseudocode in Figure 3 contains this version. The main idea is to maintain a priority queue, a data structure that allows inserting objects with numerical keys into a set, and extracting the object with the maximum key (Cormen et al., 2009). The key will represent the improvement of the edge when cutting a leaf. We first call the base learner on the full data set (line 1) and insert it into the priority queue using the difference between the best edge improvement among all the leaves in the priority queue (line 7), we partition the data set (line 11), call the base learners on the two new leaves (line 12), and insert them into the priority queue using the difference between the old edge on the partitioned data sets and the new edges of the base classifiers in the two new leaves (line 13). When inserting a leaf into the queue, we also save the sign of the cut (left or right child) and the index of the parent, so the index vectors $\mathbf{I}$ and $\mathbf{r}$ can be set properly in line 8.

When the priority queue is implemented as a heap, both the insertion and the extraction of the maximum takes $O(\log N)$ time (Cormen et al., 2009), so the total running time of the procedure is $O(N(T_{\text{BASE}} + n + \log N))$, where $T_{\text{BASE}}$ is the running time of the base learner. Since $N$ cannot be more than $n$, the running time is $O(N(T_{\text{BASE}} + n))$.

Multi-class Hamming trees
TreeBase(X, Y, W, Base(·, ·), N)
1 (α, v, φ(·)) ← Base(X, Y, W)
2 S ← PriorityQueue▷O(log N) insertion and extraction of maximum key
3 Insert(S, (v, φ(·), X, Y, NULL, 0), γ(v, φ, X, Y, W))▷key = edge γ
4 Sj ← ()▷initialize classifier list
5 for j ← 1 to N
6 Ij ← τj ← NULL▷initialize child indices
7 (vj, φj(·), Xj, Yj, •, τj) ← ExtractMax(S)▷best node in the priority queue
8 if • = − then Ij ← j else if • = + then τj ← j▷child index of parent
9 Sj ← Append((Sj, vjφj(·)))▷adding hj(·) = vjφj(·) to Sj
10 (X−, Y−, W−, X+, Y+, W+) ← CutDataSet(Xj, Yj, W, φj(·))
11 for • ∈ {−, +}▷insert children into priority queue
12 (α•, v•, φ•(·)) ← Base(X•, Y•, W•)
13 Insert(S, (v•, φ•(·), X•, Y•, •, τ), γ(v•, φ•X•Y•W•) − γ(vj, φjXjYjWj))▷key = edge improvement over parent edge
14 α = 1/2 log 1 + γ(hj1, W)1 − γ(hj1, W)▷standard coefficient of the full tree classifier h1 (25)
15 return (α, Sj, 1, τ)

Figure 3. The pseudocode of the Hamming tree base learner. N is the number of inner nodes. The algorithm returns a list of base classifiers Sj, two index lists I and τ, and the base coefficient α. The tree classifier is then defined by (25).

Table 2. Test error percentages on small benchmark data sets.

<table>
<thead>
<tr>
<th></th>
<th>AB-MH</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>6.0 ± 4.0</td>
<td>10.0 ± 2.0</td>
</tr>
<tr>
<td>blood</td>
<td>22.0 ± 4.0</td>
<td>21.0 ± 5.0</td>
</tr>
<tr>
<td>wdbc</td>
<td>3.0 ± 2.0</td>
<td>2.0 ± 3.0</td>
</tr>
<tr>
<td>breast</td>
<td>34.0 ± 13.0</td>
<td>37.0 ± 8.0</td>
</tr>
<tr>
<td>ecoli</td>
<td>15.0 ± 6.0</td>
<td>16.0 ± 6.0</td>
</tr>
<tr>
<td>iris</td>
<td>7.0 ± 6.0</td>
<td>5.0 ± 6.0</td>
</tr>
<tr>
<td>pima</td>
<td>24.0 ± 5.0</td>
<td>24.0 ± 4.0</td>
</tr>
<tr>
<td>sonar</td>
<td>13.0 ± 10.0</td>
<td>14.0 ± 8.0</td>
</tr>
<tr>
<td>wine</td>
<td>2.0 ± 3.0</td>
<td>3.0 ± 4.0</td>
</tr>
</tbody>
</table>

4. Experiments

Full reproducibility was one of the key motivations when we designed our experimental setup. All experiments were done using the open source multiboost software of Benbouzid et al. (2012), version 1.2. In addition, we will make public all the configuration files, train/test/validation cuts, and the scripts that we used to set up the hyperparameter validation.

We carried out experiments on five mid-sized (isotet, letter, optdigits, pendigits, and USPS) and nine small (balance, blood, wdbc, breast, ecoli, iris, pima, sonar, and wine) data sets from the UCI repository. The five sets were chosen to overlap with the selections of most of the recent multi-class boosting papers (Kégl & Busa-Fekete, 2009; Li, 2009a;b; Zhu et al., 2009; Sun et al., 2012; Mukherjee & Schapire, 2013). The small data sets were selected for comparing AdaBoost.MH with SVMs using Gaussian kernels, taking the results of a recent paper (Duch et al., 2012) whose experimental setup we adopted. All numerical results (multi-class test errors H(f) (1) and test learning curves) are given in Appendix D, one experiment per page for clarity. Tables 1 and 2 contain summaries of the results.

Hyperparameter optimization is largely swept under the rug in papers describing alternative multi-class boosting methods. Some report results with fixed hyperparameters (Zhu et al., 2009; Sun et al., 2012) and others give the full table of test errors for a grid of hyperparameters (Kégl & Busa-Fekete, 2009; Li, 2009a;b; Mukherjee & Schapire, 2013). Although the following procedure is rather old, we feel the need to detail it for promoting a more scrupulous comparison across papers.

For the small data sets we ran 10×10 cross-validation (CV) to optimize the hyperparameters and the estimate the generalization error. For the number of inner nodes we do a grid search (we also considered using the “one sigma” rule for biasing the selection towards smaller trees, but the sim-
ple minimization proved to be better). For robustly estimating the optimal stopping time we use a smoothed test error. For the formal description, let \( \hat{R}(t) \) be the average test error \( (1) \) of the ten validation runs after \( t \) iterations. We run ADA\-BOOST.MH for \( T_{\text{max}} \) iterations, and compute the optimal stopping time using the minimum of the smoothed test error using a linearly growing sliding window, that is,

\[
T^* = \arg \min_{T: T_{\text{min}} < T \leq T_{\text{max}}} \frac{1}{T - \lfloor 0.8T \rfloor} \sum_{t = \lfloor 0.8T \rfloor}^{T} \hat{R}(t),
\]

(26)

where \( T_{\text{min}} \) was set to a constant 50 to avoid stopping too early due to fluctuations. For selecting the best number of inner nodes \( N \), we simply minimized the smoothed test error over a predefined grid

\[
N^* = \min_{N \in N} \hat{R}(T_N^*)(N)
\]

where \( T_N^* \) and \( \hat{R}(t)(N) \) are the optimal stopping time (26) and the test error, respectively, in the run with \( N \) inner nodes, and \( N \) is the set of inner nodes participating in the grid search. Then we re-run ADA\-BOOST.MH on the joined training/validation set using the selected hyperparameters \( N^* \) and \( T_N^* \). The error \( \hat{R}_i \) in the \( t \)th training/test fold is then computed on the held-out test set. In the tables we report the mean error and the standard deviation. On the medium-size data sets we ran \( 1 \times 5 \) CV (using the designated test sets where available) following the same procedure. In this case we report the binomial standard deviation \( \sqrt{\hat{R}(1 - \hat{R})/n} \). Further details and the description and explanation of some slight variations of this experimental setup are found in Appendix D.

On the small data sets, Duch et al. (2012) used the exact same protocol, so, although the folds are not the same, the results are directly comparable. The error bars represent the standard deviation of the test errors over the ten test folds not divided by \( \sqrt{10} \), contrary to common practice, since the training set of the folds are highly correlated. The large error bars are the consequence of the small size and the noisiness of these sets. They make it difficult to establish any significant trends. We can safely state that ADA\-BOOST.MH is on par with SVM (it is certainly not worse, “winning” on six of the nine sets), widely considered one of the the best classification methods for small data sets.

Even though on the mid-sized data sets there are dedicated test sets used by most of the experimenters, comparing ADA\-BOOST.MH to alternative multi-class boosting techniques is somewhat more difficult since none of the papers do proper hyperparameter tuning. Most of the papers report results with a table of errors given for a set of hyperparameter choices, without specifying which hyperparameter choice would be picked by proper validation. For methods that are non-competitive with ADA\-BOOST.MH (SAMME of Zhu et al. (2009) and ADA\-BOOST.MM of Mukherjee & Schapire (2013)) we report the post-validated best error which may be significantly lower than the error corresponding to the hyperparameter choice selected by proper validation. For methods where this choice would unfairly bias the comparison (AOSOLOGITBOOST (Sun et al., 2012), ABCLOGITBOOST, LOGITBOOST, and ABCMART (Li, 2009a;b)), we chose the best overall hyperparameter \( J = 20 \) and \( \nu = 0.1 \), suggested by the Li (2009a;b). In Appendix D (but not in Table 1) we give both errors for some of the methods. Proper hyperparameter-validation should put the correct test error estimates between those two limits. Since ADA\-BOOST.MH with decision products (Kégl & Busa-Fekete, 2009) is also implemented in multiboost (Benbouzid et al., 2012), for this method we re-ran experiments with the protocol described above.

The overall conclusion is that AOSOLOGITBOOST (Sun et al., 2012) and ADA\-BOOST.MH with Hamming trees are the best algorithms (ADA\-BOOST.MH winning on all the five data sets but within one standard deviation). ADA\-BOOST.MH with decision products (Kégl & Busa-Fekete, 2009) and ABCLOGITBOOST are slightly weaker, as also noted by (Sun et al., 2012). SAMME (Zhu et al., 2009) and ADA\-BOOST.MM (Mukherjee & Schapire, 2013) perform below the rest of the methods on the two data sets shared among all the papers (even though

### Table 1. Test error percentages on mid-sized benchmark data sets.

<table>
<thead>
<tr>
<th>Method</th>
<th>islet</th>
<th>letter</th>
<th>optdigits</th>
<th>pendants</th>
<th>USPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADA-BOOST.MH w Hamming trees</td>
<td>3.5 ± 0.5</td>
<td>2.1 ± 0.2</td>
<td>2.0 ± 0.3</td>
<td>2.1 ± 0.3</td>
<td>4.5 ± 0.5</td>
</tr>
<tr>
<td>ADA-BOOST.MH w Hamming prod. (Kégl &amp; Busa-Fekete, 2009)</td>
<td>4.2 ± 0.5</td>
<td>2.5 ± 0.2</td>
<td>2.1 ± 0.4</td>
<td>2.1 ± 0.2</td>
<td>4.4 ± 0.5</td>
</tr>
<tr>
<td>AOSOLOGITBOOST ( J = 20, \nu = 0.1 ) (Sun et al., 2012)</td>
<td>3.5 ± 0.5</td>
<td>2.3 ± 0.2</td>
<td>2.1 ± 0.3</td>
<td>2.4 ± 0.3</td>
<td>4.9 ± 0.5</td>
</tr>
<tr>
<td>ABCLOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>4.2 ± 0.5</td>
<td>2.2 ± 0.2</td>
<td>3.1 ± 0.4</td>
<td>2.9 ± 0.3</td>
<td>4.9 ± 0.5</td>
</tr>
<tr>
<td>ABCMART ( J = 20, \nu = 0.1 ) (Li, 2009a)</td>
<td>5.0 ± 0.6</td>
<td>2.5 ± 0.2</td>
<td>2.6 ± 0.4</td>
<td>3.0 ± 0.3</td>
<td>5.2 ± 0.5</td>
</tr>
<tr>
<td>LOGITBoost ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>4.7 ± 0.5</td>
<td>2.8 ± 0.3</td>
<td>3.6 ± 0.4</td>
<td>3.1 ± 0.3</td>
<td>5.8 ± 0.5</td>
</tr>
<tr>
<td>SAMME w single-label trees (Zhu et al., 2009)</td>
<td>2.3 ± 0.2</td>
<td>2.5 ± 0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA-BOOST.MH w single-label trees (Zhu et al., 2009)</td>
<td>2.6 ± 0.3</td>
<td>2.8 ± 0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA-BOOST.MM (Mukherjee &amp; Schapire, 2013)</td>
<td>2.5 ± 0.2</td>
<td>2.7 ± 0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA-BOOST.MH w single-label trees (Mukherjee &amp; Schapire, 2013)</td>
<td>9.0 ± 0.5</td>
<td>7.0 ± 0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multi-class Hamming trees

We give post-validated results. Another important conclusion is that ADABoost.MH with Hamming trees is significantly better than other implementations of ADABoost.MH in (Zhu et al., 2009; Mukherjee & Schapire, 2013), assumably implemented using single-label trees (the errors reported by Mukherjee & Schapire (2013) are especially conspicuous).

ADABoost.MH with Hamming trees also achieves good results on image recognition problems. On MNIST, boosting trees of stumps over pixels with eight inner nodes and about 50000 iterations has a test error of 1.25%, making it one of the best no-domain-knowledge “shallow” classifiers. Using stumps over Haar filters (Viola & Jones, 2004), boosted trees with four inner nodes and 10000 iterations achieves a test error of 0.85%, comparable to classical convolutional nets (LeCun et al., 1998).

ADABoost.MH with Hamming trees, usually combined with calibration (Platt, 2000; Niculescu-Mizil & Caruana, 2005) and model averaging, has been also successful in recent data challenges. On the Kaggle emotions data challenge, although not competitive with deep learning techniques, out-of-the-box ADABoost.MH with Hamming trees over Haar filters finished 17th place with a test error of 57%. In the Yahoo! Learning-to-Rank Challenge (Chapelle et al., 2011; Busa-Fekete et al., 2013) it achieved top ten performances with results not significantly different from the winning scores. Finally, in the recent INTERSPEECH Challenge it won the Emotion sub-challenge and it was runner up in the Social Signals sub-challenge.

5. Conclusion

In this paper we introduced Hamming trees that optimize the multi-class edge prescribed by ADABoost.MH without reducing the multi-class problem to $K$ binary one-against-all classifications. We showed that without this restriction, often considered mandatory, ADABoost.MH is one of the best off-the-shelf multi-class classification algorithms. The algorithm retains the conceptual elegance, power, and computational efficiency of binary ADABoost.

Using decision stumps at the inner nodes and at the leaves of the tree is a natural choice due to the efficiency of the learning algorithm, nevertheless, the general setup described in this paper allows for using any binary classifier. One of the avenues investigated for future work is to try stronger classifiers, such as SVMs, as binary cuts. The formal setup described in Section 2.1 does not restrict the algorithm to single-label problems; another direction for future work is to benchmark it on standard multi-label and sequence-to-sequence classification problems (Dietterich et al., 2008).

Acknowledgments

This work was supported by the ANR-2010-COSI-002 grant of the French National Research Agency.

References


Duch, W., Jankowski, N., and Maszczyk, T. Make it cheap: Learning with $O(nd)$ complexity. In International Joint Conference on Neural Networks (IJCNN), pp. 1–4, 2012.


A. Showing (20)

\[
Z(\mathbf{h}, \mathbf{W}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp(-h_{\ell}(\mathbf{x}_i) y_{i,\ell}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp(-\alpha \psi_{\ell}(\mathbf{x}_i) y_{i,\ell})
\]

(27) comes from the definition (17) of \( h \) and (28) follows from the definitions (18) and (19) of \( \mu_{\ell-} \) and \( \mu_{\ell+} \). In the final step (29) we used the fact that

\[
\sum_{\ell=1}^{K} (\mu_{\ell+} + \mu_{\ell-}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} = 1.
\]

B. Multi-class decision stumps

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

\[
\psi_{j,\ell}(\mathbf{x}) = \begin{cases} 
1 & \text{if } x^{(j)} \geq b, \\
-1 & \text{otherwise},
\end{cases}
\]

where \( j \) is the index of the selected feature and \( b \) is the decision threshold. If the feature values \( (x^{(1)}, \ldots, x^{(n)}) \) are pre-ordered before the first boosting iteration, a decision stump maximizing the edge (22) (or minimizing the energy (27))\(^3\) can be found very efficiently in \( \Theta(nK) \) time.

The pseudocode of the algorithm is given in Figure 4. STUMPBase first calculates the edge vector \( \gamma^{(0)} \) of the constant classifier \( \mathbf{h}^{(0)}(\mathbf{x}) = 1 \) which will serve as the initial edge vector for each featurewise edge-maximizer. Then it loops over the features, calls BESTSTUMP to return the best featurewise stump, and then selects the best of the best by minimizing the energy (27). BESTSTUMP loops over all (sorted) feature values \( s_1, \ldots, s_{n-1} \). It considers all thresholds \( b \) halfway between two non-identical feature values \( s_i \neq s_{i+1} \). The main trick (and, at the same time, the bottleneck of the algorithm) is the update of the classwise edges in lines 4-5: when the threshold moves from \( b = \frac{s_{i-1} + s_i}{2} \) to \( b = \frac{s_{i} + s_{i+1}}{2} \), the classwise edge \( \gamma_{\ell} \) of \( 1 \psi(\mathbf{x}) \) (that is, \( \psi_{\ell}(\mathbf{x}) \) with \( v = 1 \)) can only change by \( \pm w_{i,\ell} \), depending on the sign \( y_{i,\ell} \) (Figure 5). The total edge of \( \psi_{\ell}(\mathbf{x}) \) with optimal votes (24) is then the sum of the absolute values of the classwise edges of \( 1 \psi(\mathbf{x}) \) (line 7).

\(^3\)Note the distincton: for full binary \( v \) the two are equivalent, but for ternary or real valued \( v \) and/or real valued \( \phi(\mathbf{x}) \) they are not. In Figure 4 we are maximizing the edge within each feature (line 7 in BESTSTUMP) but across features we are minimizing the energy (line 7 in STUMPBase). Updating the energy inside the inner loop (line 4) could not be done in \( \Theta(K) \) time.
Multi-class Hamming trees

STUMPASE(X, Y, W)

1 for ℓ ← 1 to K → for all classes
2 γ(0) ℓ ← \sum_{i=1}^{n} w_i y_i ℓ → classwise edges (21) of constant classifier \( h^{(0)}(x) \equiv 1 \)
3 for j ← 1 to d → all (numerical) features
4 s ← SORT(x(j) 1, \ldots, x(j) n) → sort the jth column of X
5 (v_j, b_j, γ_j) ← BESTSTUMP(s, Y, W, γ(0)) → best stump per feature
6 \( \alpha_j \) ← \frac{1}{2} \log 1 + γ_j 1 - γ_j → base coefficient (23)
7 j∗ ← arg min_j Z(\alpha_j v_j ϕ_j, b_j, W) → best stump across features
8 return (\alpha_j∗, v_j∗, ϕ_j∗, b_j∗)

BESTSTUMP(s, Y, W, γ(0))

1 γ∗ ← γ(0) → best edge vector
2 γ ← γ(0) → initial edge vector
3 for i ← 1 to n - 1 → for all points in order \( s_1 \leq \ldots \leq s_{n-1} \)
4 for ℓ ← 1 to K → for all classes
5 γ ℓ ← γ ℓ - 2w_i y_i ℓ → update classwise edges of stump with v = 1
6 if s_i \neq s_i+1 then → no threshold if identical coordinates \( s_i = s_i+1 \)
7 if \sum_{ℓ=1}^{K} |γ ℓ | > \sum_{ℓ=1}^{K} |γ ℓ | then → found better stump
8 γ∗ ← γ → update best edge vector
9 b∗ ← \frac{s_i + s_i+1}{2} → update best threshold
10 for ℓ ← 1 to K → for all classes
11 v∗ ℓ ← sign(γ ℓ ) → set vote vector according to (24)
12 if γ∗ = γ(0) → did not beat the constant classifier
13 return (v∗, -∞, ||γ∗||1) → constant classifier with optimal votes
14 else
15 return (v∗, b∗, ||γ∗||1) → best stump

Figure 4. Exhaustive search for the best decision stump. BESTSTUMP receives a sorted column (feature) s of the observation matrix X. The sorting in line 4 can be done once for all features outside of the boosting loop. BESTSTUMP examines all thresholds b halfway between two non-identical coordinates \( s_i \neq s_i+1 \) and returns the threshold \( b^* \) and vote vector \( v^* \) that maximizes the edge \( γ(v, ϕ, b, W) \). STUMPBASE then sets the coefficient \( \alpha_j \) according to (23) and chooses the stump across features that minimizes the energy (5).

C. Cutting the data set

The basic operation when adding a tree node with a scalar binary classifier (cut) \( ϕ \) is to separate the data matrices X, Y, and W according to the sign of the classification \( ϕ(x_i) \) for all \( x_i \in X \). Figure 6 contains the pseudocode of this simple operation.
Figure 5. Updating the edge $\gamma_\ell$ in line 5 of BESTSTUMP. If $y_{i,\ell} = 1$, then $\gamma_\ell$ decreases by $2w_{i,\ell}$, and if $y_{i} = -1$, then $\gamma_\ell$ increases by $2w_{i,\ell}$.

\begin{verbatim}
cutDataSet(X, Y, W, \varphi(\cdot))
    1  X_- \leftarrow Y_- \leftarrow W_- \leftarrow X_+ \leftarrow Y_+ \leftarrow W_+ \leftarrow ( ) \triangleright empty vectors
    2  for i \leftarrow 1 to n
    3      if x_i \in X then
    4          if \varphi(x_i) = -1 then
    5              X_- \leftarrow APPEND(X_-, x_i)
    6              Y_- \leftarrow APPEND(Y_-, y_i)
    7              W_- \leftarrow APPEND(W_-, w_i)
    8          else
    9              X_+ \leftarrow APPEND(X_+, x_i)
   10              Y_+ \leftarrow APPEND(Y_+, y_i)
   11              W_+ \leftarrow APPEND(W_+, w_i)
   12  return (X_-, Y_-, W_-, X_+, Y_+, W_+)
\end{verbatim}

Figure 6. The basic operation when adding a tree node is to separate the data matrices $X$, $Y$, and $W$ according to the sign of classification $\varphi(x_i)$ for all $x_i \in X$.

D. Detailed results on data sets
Multi-class Hamming trees

The abalone data set

**Source:** [http://archive.ics.uci.edu/ml/datasets/Abalone](http://archive.ics.uci.edu/ml/datasets/Abalone)

**Data set:** number of training instances $n = 3342$, number of test instances $n = 835$, number of features $d = 10$, number of classes $K = 28$.

**Experimental setup:** $1 \times 5$ cross-validation, $T_{\text{max}} = 100000$

number of tree nodes $N \in \{1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

It is a “bad” data set: low complexity and high noise. After less than 10 iterations, boosting identifies the useful features. After this point the error just fluctuates around 74%. There is very little difference between the algorithms. We included it since (Mukherjee & Schapire, 2013) used it.

**Test error results:**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAABOOST.MH with greedy single-label trees (Mukherjee &amp; Schapire, 2013)</td>
</tr>
<tr>
<td>ADAABOOST.MM (Mukherjee &amp; Schapire, 2013)</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming trees/decision stumps</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAABOOST.MH with greedy single-label trees (Mukherjee &amp; Schapire, 2013)</td>
</tr>
<tr>
<td>ADAABOOST.MM (Mukherjee &amp; Schapire, 2013)</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming trees/decision stumps</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAABOOST.MH with greedy single-label trees (Mukherjee &amp; Schapire, 2013)</td>
<td>73.0±2.0</td>
</tr>
<tr>
<td>ADAABOOST.MM (Mukherjee &amp; Schapire, 2013)</td>
<td>73.0±2.0</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming trees/decision stumps</td>
<td>74.0±2.0</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>76.0±1.0</td>
</tr>
</tbody>
</table>

**ADAABOOST.MH learning curves:**

![Learning Curves Plot](image_url)
Multi-class Hamming trees

The adult data set

Source: [http://archive.ics.uci.edu/ml/datasets/Adult](http://archive.ics.uci.edu/ml/datasets/Adult)

Data set: number of training instances $n = 30162$, number of test instances $n = 15060$, number of features $d = 105$, number of classes $K = 2$.

Experimental setup: $1 \times 5$ cross-validation, $T_{\text{max}} = 100000$

number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABoost.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>13.4±0.3</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming trees/decision stumps</td>
<td>13.5±0.3</td>
</tr>
</tbody>
</table>

ADABoost.MH learning curves:
The chesskrk data set

**Source:** [http://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King%29](http://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King%29)

**Data set:** number of training instances \( n = 22000 \), number of test instances \( n = 6056 \), number of features \( d = 48 \), number of classes \( K = 18 \).

**Experimental setup:** \( 1 \times 5 \) cross-validation, \( T_{\text{max}} = 100000 \)

number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100, 120, 150, 200\} \)

number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

This is an interesting data set: it needs both high-complexity base classifiers and a large number of boosting iterations. In fact, after an initial optimization run up to \( N = 100 \) nodes, we had to open up the search space to go up to \( N = 200 \).

**Test error results:**

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABOOST.MH with Hamming trees/decision stumps</td>
<td>9.3±0.4</td>
</tr>
<tr>
<td>ADABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>11.2±0.4</td>
</tr>
</tbody>
</table>

**ADABOOST.MH learning curves:**

![Learning curves graph](image-url)
The covtype data set

Source: http://archive.ics.uci.edu/ml/datasets/Covertype

Data set: number of training instances \( n = 15000 \), number of test instances \( n = 566012 \), number of features \( d = 54 \), number of classes \( K = 7 \).

Experimental setup: \( 1 \times 5 \) cross-validation, \( T_{\text{max}} = 100000 \)

number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\} \)

number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

It is a huge set with a relatively small number of features. It would be interesting to study the asymptotic behavior of algorithms (as the training data grows), but in this study the goal is to compare the methods on moderately large sets, so we use 15 000 training points and 566 012 test points. We do 5CV for validation, as usual. The setup is also comparable to some of the only publications on the data and to (Mukherjee & Schapire, 2013) (15 120 training points), but not (directly) to (Li, 2009a;b).

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABOOST.MH with Hamming trees/decision stumps</td>
<td>14.84±0.05</td>
</tr>
<tr>
<td>ADABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>15.08±0.05</td>
</tr>
<tr>
<td>ADABOOST.MM (Mukherjee &amp; Schapire, 2013)</td>
<td>24.0±0.4</td>
</tr>
<tr>
<td>ADABOOST.MH with greedy single-label trees (Mukherjee &amp; Schapire, 2013)</td>
<td>30.0±0.4</td>
</tr>
</tbody>
</table>

ADABOOST.MH learning curves:
The islet data set

**Source:** [http://archive.ics.uci.edu/ml/datasets/ISOLET](http://archive.ics.uci.edu/ml/datasets/ISOLET)

**Dataset:** number of training instances \( n = 6238 \), number of test instances \( n = 1559 \), number of features \( d = 617 \), number of classes \( K = 26 \).

**Experimental setup:** \( 1 \times 4 \) cross-validation, \( T_{\text{max}} = 100000 \)
number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\} \)
number of product terms \( m \in \{1, 2, 3, 4, 5\} \)

The test set contains data generated by independent writers, so the train and test distributions are very different. Since data produced by a single writer is in a consecutive block, we do a 4CV using a non-random (order) permutation. With this setup, the validation error matches the test error.

The error rates of ABCMART, LOGITBOOST, and ABCLOGITBOOST were post-validated (minimized on the test set with respect to the hyperparameters) since (Li, 2009a;b) did not carry out proper cross-validation to determine the best hyperparameters \( (J, \nu, \text{and } T) \). The same holds for AOSOLOGITBOOST (Sun et al., 2012) but here we only report results with the best overall hyper-parameter \( J = 20 \) and \( \nu = 0.1 \), suggested by the authors of (Li, 2009a;b).

**Test error results:**

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAUOOST.MH with Hamming trees/decision stumps</td>
<td>3.5±0.5</td>
</tr>
<tr>
<td>ABCLOGITBOOST post-validated (Li, 2009b)</td>
<td>3.5±0.5</td>
</tr>
<tr>
<td>AOSOLOGITBOOST ( J = 20, \nu = 0.1 ) (Sun et al., 2012)</td>
<td>3.5±0.5</td>
</tr>
<tr>
<td>LOGITBOOST post-validated (Li, 2009b)</td>
<td>4.0±0.5</td>
</tr>
<tr>
<td>ABCMART post-validated (Li, 2009a)</td>
<td>4.1±0.5</td>
</tr>
<tr>
<td>ADAUOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>4.2±0.5</td>
</tr>
<tr>
<td>ABCLOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>4.2±0.5</td>
</tr>
<tr>
<td>LOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>4.7±0.5</td>
</tr>
<tr>
<td>ABCMART ( J = 20, \nu = 0.1 ) (Li, 2009a)</td>
<td>5.0±0.6</td>
</tr>
</tbody>
</table>

ADAUOOST.MH learning curves:
Multi-class Hamming trees

The letter data set


Data set: number of training instances $n = 16000$, number of test instances $n = 4000$, number of features $d = 16$, number of classes $K = 26$.

Experimental setup: $1 \times 5$ cross-validation, $T_{\text{max}} = 100000$
number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$
number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30, 50, 70, 100\}$

The error rates of ABCMART, LOGITBOOST, and ABCLOGITBOOST were post-validated (minimized on the test set with respect to the hyperparameters) since (Li, 2009a;b) did not carry out proper cross-validation to determine the best hyperparameters ($J$, $\nu$, and $T$). The same holds for AOSOLOGITBOOST (Sun et al., 2012) but here we only report results with the best overall hyper-parameter $J = 20$ and $\nu = 0.1$, suggested by the authors of (Li, 2009a;b).

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADA BOOST.MH with Hamming trees/decision stumps</td>
<td>2.1±0.2</td>
</tr>
<tr>
<td>ABC LOGITBOOST post-validated (Li, 2009b)</td>
<td>2.2±0.2</td>
</tr>
<tr>
<td>ABC LOGITBOOST $J = 20$, $\nu = 0.1$ (Li, 2009b)</td>
<td>2.2±0.2</td>
</tr>
<tr>
<td>AOSO LOGITBOOST $J = 20$, $\nu = 0.1$ (Sun et al., 2012)</td>
<td>2.3±0.2</td>
</tr>
<tr>
<td>SAMM with single-label trees (Zhu et al., 2009)</td>
<td>2.3±0.2</td>
</tr>
<tr>
<td>ADA BOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>2.5±0.2</td>
</tr>
<tr>
<td>ADA BOOST.MM (Mukherjee &amp; Schapire, 2013)</td>
<td>2.5±0.2</td>
</tr>
<tr>
<td>ABCMART post-validated (Li, 2009a)</td>
<td>2.5±0.2</td>
</tr>
<tr>
<td>ABCMART $J = 20$, $\nu = 0.1$ (Li, 2009a)</td>
<td>2.5±0.2</td>
</tr>
<tr>
<td>ADA BOOST.MH with single-label trees (Zhu et al., 2009)</td>
<td>2.6±0.3</td>
</tr>
<tr>
<td>LOGITBOOST post-validated (Li, 2009b)</td>
<td>2.7±0.3</td>
</tr>
<tr>
<td>LOGITBOOST $J = 20$, $\nu = 0.1$ (Li, 2009b)</td>
<td>2.8±0.3</td>
</tr>
<tr>
<td>ADA BOOST.MH with greedy single-label trees (Mukherjee &amp; Schapire, 2013)</td>
<td>9.0±0.5</td>
</tr>
</tbody>
</table>

ADA BOOST.MH learning curves:
Multi-class Hamming trees

The optdigits data set


Data set: number of training instances $n = 3823$, number of test instances $n = 1797$, number of features $d = 64$, number of classes $K = 10$.

Experimental setup: $1 \times 5$ cross-validation, $T_{\text{max}} = 100000$

number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

The test set contains data generated by independent writers, so the train and test distributions are different. In particular, the validation error is about 1.3% whereas test error is above 2%. Writer IDs are unknown, so we did random 5CV.

The error rates of ABCMART, LOGITBOOST, and ABCLOGITBOOST were post-validated (minimized on the test set with respect to the hyperparameters) since (Li, 2009a;b) did not carry out proper cross-validation to determine the best hyperparameters ($J$, $\nu$, and $T$). The same holds for AOSOLOGITBOOST (Sun et al., 2012) but here we only report results with the best overall hyper-parameter $J = 20$ and $\nu = 0.1$, suggested by the authors of (Li, 2009a;b).

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>$\text{error} \pm \text{std}$ (%)</th>
</tr>
</thead>
</table>
| ADA
dOOST.MH with Hamming trees/decision stumps | $2.0 \pm 0.3$ |
| ADA
dOOST.MH with Hamming products/decision stumps (Kégl & Busa-Fekete, 2009) | $2.1 \pm 0.3$ |
| AOSOLOGITBOOST $J = 20$, $\nu = 0.1$ (Sun et al., 2012) | $2.1 \pm 0.3$ |
| ABCLOGITBOOST post-validated (Li, 2009b) | $2.3 \pm 0.4$ |
| ABCMART post-validated (Li, 2009a) | $2.4 \pm 0.4$ |
| ABCMART $J = 20$, $\nu = 0.1$ (Li, 2009a) | $2.6 \pm 0.4$ |
| LOGITBOOST post-validated (Li, 2009b) | $2.7 \pm 0.4$ |
| ABCLOGITBOOST $J = 20$, $\nu = 0.1$ (Li, 2009b) | $3.1 \pm 0.4$ |
| LOGITBOOST $J = 20$, $\nu = 0.1$ (Li, 2009b) | $3.6 \pm 0.4$ |

ADABOOST.MH learning curves:
Multi-class Hamming trees

The pendigits data set

Source: http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

Data set: number of training instances \( n = 7494 \), number of test instances \( n = 3498 \), number of features \( d = 16 \), number of classes \( K = 10 \).

Experimental setup: \( 1 \times 5 \) cross-validation, \( T_{\text{max}} = 100000 \)
number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\} \)
number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

The test set contains data generated by independent writers, so the train and test distributions are very different. In particular, the validation error is about 0.7% whereas test error is above 2%. If the parameters are validated using random 5CV on the training set, both trees and products are more complex than they should be, resulting in a suboptimal behavior on the test data. We could recover the writer ids from the pendigits-orig files (second number after .COMMENT), and so we created 5-fold writer-independent CV. The validation and test errors are closer, but they vary a lot fold-to-fold, and the complexity is still slightly overestimated.

The error rates of ABCMART, LOGITBOOST, and ABCLOGITBOOST were post-validated (minimized on the test set with respect to the hyperparameters) since (Li, 2009a;b) did not carry out proper cross-validation to determine the best hyperparameters \((J, \nu, \text{and } T)\). The same holds for AOSOLOGITBOOST (Sun et al., 2012) but here we only report results with the best overall hyper-parameter \( J = 20 \) and \( \nu = 0.1 \), suggested by the authors of (Li, 2009a;b). ABCMART (Zhu et al., 2009) did use proper cross validation for setting the size of the trees but the number of iterations is still post-validated. With post-validation, boosted products with \( T = 100000 \) iterations and \( m = 2 \) terms achieve an error rate of 1.9% (Kégl & Busa-Fekete, 2009).

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABooST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>2.1±0.2</td>
</tr>
<tr>
<td>ADABooST.MH with Hamming trees/decision stumps</td>
<td>2.1±0.2</td>
</tr>
<tr>
<td>AOSOLOGITBOOST ( J = 20, \nu = 0.1 ) (Sun et al., 2012)</td>
<td>2.4±0.3</td>
</tr>
<tr>
<td>SAMME with single-label trees (Zhu et al., 2009)</td>
<td>2.5±0.3</td>
</tr>
<tr>
<td>ABCLOGITBOOST post-validated (Li, 2009b)</td>
<td>2.6±0.3</td>
</tr>
<tr>
<td>ADABooST-MM (Mukherjee &amp; Schapire, 2013)</td>
<td>2.7±0.3</td>
</tr>
<tr>
<td>ADABooST.MH with single-label trees (Zhu et al., 2009)</td>
<td>2.8±0.3</td>
</tr>
<tr>
<td>ABCLOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>2.9±0.3</td>
</tr>
<tr>
<td>ABCMART post-validated (Li, 2009a)</td>
<td>2.9±0.3</td>
</tr>
<tr>
<td>ABCMART ( J = 20, \nu = 0.1 ) (Li, 2009a)</td>
<td>3.0±0.3</td>
</tr>
<tr>
<td>LOGITBOOST post-validated (Li, 2009b)</td>
<td>3.1±0.3</td>
</tr>
<tr>
<td>LOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>3.1±0.3</td>
</tr>
<tr>
<td>ADABooST.MH with greedy single-label trees (Mukherjee &amp; Schapire, 2013)</td>
<td>7.0±0.4</td>
</tr>
</tbody>
</table>

ADABooST.MH learning curves:
Multi-class Hamming trees

\[ \text{Product, } m = 4., \quad T = 45715 \]
\[ \text{Tree, } N = 20., \quad T = 14802 \]

\begin{tabular}{c|c|c|c|c|c}
        & 1 & 10 & 100 & 1000 & 10^4 \\
\hline
\text{test error} & 0.0 & 0.2 & 0.4 & 0.6 & 0.8 & 1.0 \\
\end{tabular}

\text{pendigits}
The satimage data set

**Source:** [http://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)](http://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite))

**Data set:** number of training instances \( n = 4435 \), number of test instances \( n = 2000 \), number of features \( d = 36 \), number of classes \( K = 7 \).

**Experimental setup:** \( 1 \times 5 \) cross-validation, \( T_{\text{max}} = 100000 \)

number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100, 120, 150, 200\} \)

number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

**Test error results:**

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADA-BOOST.MH with Hamming trees/decision stumps</td>
<td>8.4±0.6</td>
</tr>
<tr>
<td>ADA-BOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>8.8±0.6</td>
</tr>
</tbody>
</table>

**ADA-BOOST.MH learning curves:**

\[ e \sim f \sim \sim \sim OE \]

\[ m \sim q \sim k \sim P \sim M \sim V \sim U \]

\[ q \sim k \sim M \sim V \sim U \]
The USPS data set

Source: http://

Data set: number of training instances \( n = 7291 \), number of test instances \( n = 2007 \), number of features \( d = 256 \), number of classes \( K = 10 \).

Experimental setup: \( 1 \times 5 \) cross-validation, \( T_{\text{max}} = 100000 \)

number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\} \)

number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>4.4±0.5</td>
</tr>
<tr>
<td>ADBOOST.MH with Hamming trees/decision stumps</td>
<td>4.5±0.5</td>
</tr>
<tr>
<td>ABCLOGITBOOST post-validated (Li, 2009b)</td>
<td>4.6±0.5</td>
</tr>
<tr>
<td>ABCLOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>4.8±0.5</td>
</tr>
<tr>
<td>AOSOLOGITBOOST ( J = 20, \nu = 0.1 ) (Sun et al., 2012)</td>
<td>4.9±0.5</td>
</tr>
<tr>
<td>ABCMART post-validated (Li, 2009a)</td>
<td>5.0±0.5</td>
</tr>
<tr>
<td>ABCMART ( J = 20, \nu = 0.1 ) (Li, 2009a)</td>
<td>5.2±0.5</td>
</tr>
<tr>
<td>LOGITBOOST post-validated (Li, 2009b)</td>
<td>5.3±0.5</td>
</tr>
<tr>
<td>LOGITBOOST ( J = 20, \nu = 0.1 ) (Li, 2009b)</td>
<td>5.8±0.5</td>
</tr>
</tbody>
</table>

ADB propagation learning curves:
Multi-class Hamming trees

The balance data set


Data set: number of instances $n = 625$, number of features $d = 4$, number of classes $K = 3$.

Experimental setup: $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$

number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Small but decent set: low complexity in terms of tree size but needs a couple of hundred iterations to converge.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABOOST.MH with Hamming trees/decision stumps</td>
<td>6.0±4.0</td>
</tr>
<tr>
<td>ADABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>8.0±3.0</td>
</tr>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>10.0±2.0</td>
</tr>
</tbody>
</table>

ADABoost.MH learning curves:

\[
\tilde{A} \hat{a} = \tilde{A} \hat{E}
\]

\[
n \hat{Q} \hat{I} \hat{a} = N \hat{k} \pm M \hat{q} = COT \pm OTS
\]

\[
q \hat{K} \hat{I} \hat{R} = N \hat{R} \pm M \hat{R} \hat{I} \hat{P} \hat{I} \hat{q} = SRP \pm QPM
\]
The blood data set

**Source:** [http://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center](http://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center)

**Data set:** number of instances $n = 748$, number of features $d = 4$, number of classes $K = 2$.

**Experimental setup:** $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$

- number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$
- number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Particularly bad data set: there is no significant improvement over one single stump.

**Test error results:**

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM with Gaussian kernel (<a href="http://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center">Duch et al., 2012</a>)</td>
<td>21.0±5.0</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming products/decision stumps (<a href="http://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center">Kégl &amp; Busa-Fekete, 2009</a>)</td>
<td>22.0±4.0</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming trees/decision stumps</td>
<td>22.0±4.0</td>
</tr>
</tbody>
</table>

**ADAABOOST.MH learning curves:**

![ADAABOOST.MH learning curve](image-url)
The wdbc data set

Source: http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+Diagnostic

Data set: number of instances $n = 569$, number of features $d = 30$, number of classes $K = 2$.

Experimental setup: $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$

- number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$
- number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Small but decent set: low complexity in terms of tree size but needs a couple of hundred iterations to converge.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABooST.MH with Hamming products/decision stumps</td>
<td>2.0±2.0</td>
</tr>
<tr>
<td>(Kégl &amp; Busa-Fekete, 2009)</td>
<td></td>
</tr>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>2.0±3.0</td>
</tr>
<tr>
<td>ADABooST.MH with Hamming trees/decision stumps</td>
<td>3.0±2.0</td>
</tr>
</tbody>
</table>

ADABooST.MH learning curves:

![Learning curves for wdbc dataset](image)
Multi-class Hamming trees

The breast data set

Source: http://archive.ics.uci.edu/ml/datasets/Breast+Tissue

Data set: number of instances $n = 106$, number of features $d = 9$, number of classes $K = 6$.

Experimental setup: $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$

number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Extremely small and not too interesting set: it likes small trees (but not stumps) and converges after about $T = 20$ iterations.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABoost.MH with Hamming trees/decision stumps</td>
<td>34.0±13.0</td>
</tr>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>37.0±8.0</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>38.0±18.0</td>
</tr>
</tbody>
</table>

ADABoost.MH learning curves:

Product, $m = 2.4 \pm 3.1$, $T = 69 \pm 69$

Tree, $N = 2.9 \pm 6.$, $T = 122 \pm 181$
Multi-class Hamming trees

The ecoli data set

Source: http://archive.ics.uci.edu/ml/datasets/Ecoli
Data set: number of instances \( n = 336 \), number of features \( d = 7 \), number of classes \( K = 8 \).
Experimental setup: \( 10 \times 10 \) cross-validation, \( T_{\text{max}} = 1000 \)
number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\} \)
number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

Quite interesting set for its size: it likes bigger trees and need about \( T = 100 \) iterations to converge.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABOOST.MH with Hamming trees/decision stumps</td>
<td>15.0±6.0</td>
</tr>
<tr>
<td>ADABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>15.0±6.0</td>
</tr>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>16.0±6.0</td>
</tr>
</tbody>
</table>

ADABOOST.MH learning curves:
Multi-class Hamming trees

The iris data set

**Source:** [http://archive.ics.uci.edu/ml/datasets/Iris](http://archive.ics.uci.edu/ml/datasets/Iris)

**Data set:** number of instances $n = 150$, number of features $d = 4$, number of classes $K = 3$.

**Experimental setup:** $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$

number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Small and uninteresting set: it basically needs two stumps to converge.

**Test error results:**

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM with Gaussian kernel ([Duch et al., 2012])</td>
<td>5.0±6.0</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming products/decision stumps ([Kégl &amp; Busa-Fekete, 2009])</td>
<td>7.0±6.0</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming trees/decision stumps</td>
<td>7.0±6.0</td>
</tr>
</tbody>
</table>

**ADABoost.MH learning curves:**

\[
\hat{c} = m\hat{c} + \hat{a} = N\hat{O} \pm MS\hat{P}I \quad q = PO \pm OR
\]

\[
q\hat{d} = k = NK \pm MP\hat{O}I \quad q = OQ \pm V
\]
The pima data set

Source: http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

Data set: number of instances $n = 768$, number of features $d = 8$, number of classes $K = 2$.

Experimental setup: $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$

number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$

number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Another terrible set: no significant improvement over one single stump.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>24.0±4.0</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>24.0±5.0</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming trees/decision stumps</td>
<td>24.0±5.0</td>
</tr>
</tbody>
</table>

ADABoost.MH learning curves:
The sonar data set


Data set: number of instances $n = 208$, number of features $d = 60$, number of classes $K = 2$.

Experimental setup: $10 \times 10$ cross-validation, $T_{\text{max}} = 1000$
number of tree nodes $N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\}$
number of product terms $m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\}$

Quite interesting set for its size: it likes small trees (but bigger than stumps) and needs a couple of hundred iterations to converge.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error±std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADABoost.MH with Hamming trees/decision stumps</td>
<td>13.0±10.0</td>
</tr>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>14.0±8.0</td>
</tr>
<tr>
<td>ADABoost.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>15.0±8.0</td>
</tr>
</tbody>
</table>

ADABoost.MH learning curves:

\[ \text{error} = NQ \pm M R \quad \text{q} = N C R \pm N M N \]

\[ \text{test error} = N Q \pm M R \quad \text{q} = O V T \pm N O V \]
The wine data set

Source: http://archive.ics.uci.edu/ml/datasets/Wine

Data set: number of instances \( n = 178 \), number of features \( d = 13 \), number of classes \( K = 3 \).

Experimental setup: 10 \( \times \) 10 cross-validation, \( T_{\text{max}} = 1000 \)
number of tree nodes \( N \in \{1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100\} \)
number of product terms \( m \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 30\} \)

Small but decent data set: it likes stumps but needs about a hundred iterations to converge.

Test error results:

<table>
<thead>
<tr>
<th>Method</th>
<th>error ± std (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAABOOST.MH with Hamming products/decision stumps (Kégl &amp; Busa-Fekete, 2009)</td>
<td>2.0 ± 3.0</td>
</tr>
<tr>
<td>ADAABOOST.MH with Hamming trees/decision stumps</td>
<td>2.0 ± 3.0</td>
</tr>
<tr>
<td>SVM with Gaussian kernel (Duch et al., 2012)</td>
<td>3.0 ± 4.0</td>
</tr>
</tbody>
</table>

ADAABOOST.MH learning curves: