Inf Retrieval
DOI 10.1007/s10791-008-9047-y

Boosting multi-label hierarchical text categorization

Andrea Esuli · Tiziano Fagni · Fabrizio Sebastiani

Received: 19 January 2008 / Accepted: 22 January 2008

Abstract *Hierarchical Text Categorization* (HTC) is the task of generating (usually by means of supervised learning algorithms) text classifiers that operate on hierarchically structured classification schemes. Notwithstanding the fact that most large-sized classification schemes for text have a hierarchical structure, so far the attention of text classification researchers has mostly focused on algorithms for "flat" classification, i.e. algorithms that operate on non-hierarchical classification schemes. These algorithms, once applied to a hierarchical classification problem, are not capable of taking advantage of the information inherent in the class hierarchy, and may thus be suboptimal, in terms of efficiency and/or effectiveness. In this paper we propose TreeBoost.MH, a multi-label HTC algorithm consisting of a hierarchical variant of AdaBoost.MH, a very well-known member of the family of "boosting" learning algorithms. TreeBoost.MH embodies several intuitions that had arisen before within HTC: e.g. the intuitions that both feature selection and the selection of negative training examples should be performed "locally", i.e. by paying attention to the topology of the classification scheme. It also embodies the novel intuition that the weight distribution that boosting algorithms update at every boosting round should likewise be updated "locally". All these intuitions are embodied within TreeBoost.MH in an elegant and simple way, i.e. by defining TreeBoost.MH as a recursive algorithm that uses AdaBoost.MH as its base step, and that recurs over the tree structure. We present the results of experimenting TreeBoost.MH on three HTC benchmarks, and discuss analytically its computational cost.

Keywords Hierarchical text classification · Boosting

1 Introduction

Hierarchical text categorization (HTC) is the task of generating (usually by means of supervised learning algorithms) text classifiers that operate on classification schemes

A. Esuli · T. Fagni · F. Sebastiani (✉)
Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche,
Via Giuseppe Moruzzi, 1, 56124 Pisa, Italy
e-mail: fabrizio.sebastiani@isti.cnr.it

endowed with a hierarchical structure. Notwithstanding the fact that most large-sized classification schemes for text (e.g. the ACM Classification Scheme[1] the MESH thesaurus[2] the NASA thesaurus[3]) indeed have a hierarchical structure, so far the attention of text classification (TC) researchers has mostly focused on algorithms for "flat" classification, i.e. algorithms that operate on non-hierarchical classification schemes.[4] These algorithms, once applied to a hierarchical classification problem, are not capable of taking advantage of the information inherent in the class hierarchy, and may thus be suboptimal, in terms of efficiency and/or effectiveness. On the contrary, many researchers have argued that by leveraging on the hierarchical structure of the classification scheme, heuristics of various kinds can be brought to bear that make the classifier more efficient and/or more effective.

An important intuition is that, by viewing classification as the identification of the paths that, starting from the root, funnel the document down to the subtrees where it belongs (in "Pachinko machine" style), entire other subtrees can be pruned from consideration. That is, when the classifier corresponding to an internal node outputs a negative response, the classifiers corresponding to its descendant nodes need not be invoked any more, thus reducing the computational cost of classifier invocation exponentially (Chakrabarti et al. 1998; Koller and Sahami 1997).

A second important intuition is that, by training a binary classifier for an internal node category on a well-selected subset of training examples of local interest only, the resulting classifier may be made more attuned to recognizing the subtle distinctions between documents belonging to that node and those belonging to neighbouring nodes (Ng et al. 1997; Wiener et al. 1995). While this technique promises to bring about more effective classifiers, it is also going to improve efficiency, since a smaller set of examples is used in training, thereby making classifier learning speedier.

Many of these intuitions have been used in close association with a specific learning algorithm; the most popular choices in this respect have been naïve Bayesian methods (Chakrabarti et al. 1998; Gaussier et al. 2002; Koller and Sahami 1997; McCallum et al. 1998; Toutanova et al. 2001; Vinokourov and Girolami 2002), neural networks (Ruiz and Srinivasan 2002; Weigend et al. 1999; Wiener et al. 1995), support vector machines (Cai and Hofmann 2004; Dumais and Chen 2000; Liu et al. 2005; Yang et al. 2003), and example-based classifiers (Yang et al. 2003).

Within this literature, the absence of "boosting" methods is conspicuous: to the best of our knowledge, we do not know of any HTC method belonging to the boosting family. This is somehow surprising, (i) because of the high applicative interest of HTC, (ii) because boosting algorithms are well-known for their interesting theoretical properties and for their high accuracy, and (iii) because, given their relatively high computational cost, they would definitely benefit by the added efficiency that consideration of the hierarchical structure can bring about.

In this paper we try to fill this gap by proposing TreeBoost.MH, a multi-label HTC algorithm that consists of a hierarchical variant of AdaBoost.MH, the most important member of the family of boosting algorithms; here, *multi-label* (ML) means that a document can belong to zero, one, or several categories at the same time. TreeBoost.MH

[1] http://info.acm.org/class/1998/ccs98.html

[2] http://www.nlm.nih.gov/mesh/meshhome.html

[3] http://www.sti.nasa.gov/nasa-thesaurus.html

[4] This is the result of the fact that the very first publicly available TC benchmarks (e.g. the Reuters-21578 benchmark—see Sect. 5.1) had no hierarchical structure; the TC literature ended up in overfitting, at least to some degree, the available benchmarks.

embodies several intuitions that had arisen before within HTC: e.g. the intuitions that both feature selection and the selection of negative training examples should be performed "locally", i.e. by paying attention to the topology of the classification scheme. TreeBoost.MH also incorporates the novel intuition that the weight distribution that boosting algorithms update at every boosting round should likewise be updated "locally". All these intuitions are embodied within TreeBoost.MH in an elegant and simple way, i.e. by defining TreeBoost.MH as a recursive algorithm that uses AdaBoost.MH as its base step, and that recurs over the tree structure.

The paper is structured as follows. In Sect. 2 we give a concise description of boosting and the AdaBoost.MH algorithm. Section 3 describes TreeBoost.MH, our hierarchical version of AdaBoost.MH. Section 4 goes the analytical way, comparing the computational costs of AdaBoost.MH and TreeBoost.MH, and showing that the latter obtains exponential savings over the former both at classifier-learning time and at classification time. In Sect. 5 we present experiments comparing AdaBoost.MH and TreeBoost.MH on three well-known HTC benchmarks, including a hierarchical version of the Reuters-21578 benchmark defined in Toutanova et al. (2001). Section 6 discusses related work, pointing out the differences between existing approaches and ours. Section 7 concludes.

2 An introduction to boosting and AdaBoost.MH

AdaBoost.MH (Schapire and Singer 2000) (see Fig. 1) is a *boosting* algorithm, i.e. an algorithm that generates a highly accurate classifier $\hat{\Phi}$ (also called *final hypothesis*) by combining a set of moderately accurate classifiers $\hat{\Phi}_1, \ldots, \hat{\Phi}_S$ (also called *weak hypotheses*).[5] The input to the algorithm is a training set $Tr = \{\langle d_1, C_1\rangle, \ldots, \langle d_g, C_g\rangle\}$, where $C_i \subseteq C$ is the set of categories to each of which d_i belongs. For each $c_j \in C$, by $Tr^+(c_j)$ we denote the set of the positive training examples of c_j. Furthermore, for each $c_j \in C$ we define the set $Tr^-(c_j)$ of its negative training examples simply as the set difference between Tr and $Tr^+(c_j)$.

AdaBoost.MH works by iteratively calling a *weak learner* to generate a sequence $\hat{\Phi}_1, \ldots, \hat{\Phi}_S$ of weak hypotheses; at the end of the iteration the final hypothesis $\hat{\Phi}$ is obtained as a sum $\hat{\Phi} = \sum_{s=1}^{S} \hat{\Phi}_s$ of these weak hypotheses. A weak hypothesis is a function $\hat{\Phi}_s : D \times C \rightarrow \mathbb{R}$ such that $sign(\hat{\Phi}_s(d_i, c_j))$ can be interpreted as the prediction of $\hat{\Phi}_s$ on whether d_i belongs to c_j (i.e. $\hat{\Phi}_s(d_i, c_j) > 0$ means that d_i is believed to belong to c_j while $\hat{\Phi}_s(d_i, c_j) < 0$ means it is believed not to belong to c_j), and the absolute value of $\hat{\Phi}_s(d_i, c_j)$ (indicated by $|\hat{\Phi}_s(d_i, c_j)|$) can be interpreted as the strength of this belief.

At each iteration s AdaBoost.MH tests the effectiveness of the newly generated weak hypothesis $\hat{\Phi}_s$ on the training set and uses the results to update a distribution D_s of weights on the training pairs $\langle d_i, c_j\rangle$. The updated weight $D_{s+1}(d_i, c_j)$ is meant to capture how effective $\hat{\Phi}_1, \ldots, \hat{\Phi}_s$ have been in correctly predicting whether the training document d_i belongs to category c_j or not. By passing (together with the training set Tr) this distribution to the weak learner, AdaBoost.MH asks this latter to generate a new weak hypothesis $\hat{\Phi}_{s+1}$ that concentrates on the pairs with the highest weight, i.e. those that had proven harder to classify for the previous weak hypotheses.

The initial distribution D_1 is uniform. At each iteration s all the weights $D_s(d_i, c_j)$ are updated to $D_{s+1}(d_i, c_j)$ according to the rule

[5] Consistently with most mathematical literature we use the caret symbol (ˆ) to indicate estimation. In fact, a classifier $\hat{\Phi}$ can be understood as an estimation, or approximation, of an unknown "target function" Φ.

Input: A training set $Tr = \{\langle d_1, C_1\rangle, \ldots, \langle d_g, C_g\rangle\}$

where $C_i \subseteq C = \{c_1, \ldots, c_m\}$ for all $i = 1, \ldots, g$.

Body: Set $D_1(d_i, c_j) = \dfrac{1}{gm}$ for all $i = 1, \ldots, g$ and for all $j = 1, \ldots, m$;

for $s = 1, \ldots, S$ **do**

begin

Pass distribution $D_s(d_i, c_j)$ to the weak learner;

Get the weak hypothesis $\hat{\Phi}_s$ from the weak learner;

Set $D_{s+1}(d_i, c_j) = \dfrac{D_s(d_i, c_j)\exp(-\Phi(d_i, c_j)\cdot\hat{\Phi}_s(d_i, c_j))}{Z_s}$

where $Z_s = \sum_{i=1}^{g}\sum_{j=1}^{m} D_s(d_i, c_j)\exp(-\Phi(d_i, c_j)\cdot\hat{\Phi}_s(d_i, c_j))$

is a normalization factor chosen so that $\sum_{i=1}^{g}\sum_{j=1}^{m} D_{s+1}(d_i, c_j) = 1$

end

Output: A final hypothesis $\hat{\Phi}(d, c) = \sum_{s=1}^{S}\hat{\Phi}_s(d, c)$ for $c \in C = \{c_1, \ldots, c_m\}$

Fig. 1 The AdaBoost.MH algorithm

$$D_{s+1}(d_i, c_j) = \frac{D_s(d_i, c_j)\exp(-\Phi(d_i, c_j)\cdot\hat{\Phi}_s(d_i, c_j))}{Z_s} \tag{1}$$

where the *target function* $\Phi(d_i, c_j)$ is defined to be 1 if $d_i \in c_j$ and -1 otherwise, and

$$Z_s = \sum_{i=1}^{g}\sum_{j=1}^{m} D_s(d_i, c_j)\exp(-\Phi(d_i, c_j)\cdot\hat{\Phi}_s(d_i, c_j)) \tag{2}$$

is a normalization factor chosen so that D_{s+1} is in fact a distribution, i.e. so that $\sum_{i=1}^{g}\sum_{j=1}^{m} D_{s+1}(d_i, c_j) = 1$. Equation 1 is such that the weight assigned to a pair $\langle d_i, c_j\rangle$ misclassified by $\hat{\Phi}_s$ is increased, as for such a pair $\Phi(d_i, c_j)$ and $\hat{\Phi}_s(d_i, c_j)$ have different signs and the factor $\Phi(d_i, c_j)\cdot\hat{\Phi}_s(d_i, c_j)$ is thus negative; likewise, the weight assigned to a pair correctly classified by $\hat{\Phi}_s$ is decreased. Weights are increased or decreased to a larger extent if the absolute value of $\hat{\Phi}_s(d_i, c_j)$ is higher, to reflect the fact that classification decisions taken with high confidence must have a higher impact in the process.

2.1 Choosing the weak hypotheses

In AdaBoost.MH each document d_i is represented as a vector $\mathbf{d}_i = \langle w_{1i}, \ldots, w_{ri}\rangle$ of r binary weights, where $w_{ki} = 1$ (resp. $w_{ki} = 0$) is normally interpreted to mean that term t_k occurs (resp. does not occur) in d_i; accordingly, $T = \{t_1, \ldots, t_r\}$ is the set of terms that occur in at least one document in *Tr*. Of course, AdaBoost.MH does not make any assumption on

what constitutes a term; single words, stems of words, phrases, or character *n*-grams are all plausible choices.

In AdaBoost.MH the weak hypotheses generated by the weak learner at iteration s are *decision stumps* of the form

$$\hat{\Phi}_s(d_i, c_j) = \begin{cases} a_{0j} & \text{if } w_{ki} = 0 \\ a_{1j} & \text{if } w_{ki} = 1 \end{cases} \tag{3}$$

where t_k (called the *pivot term* of $\hat{\Phi}_s$) belongs to T, and a_{0j} and a_{1j} are real-valued constants. The choices for t_k, a_{0j} and a_{1j} are in general different for each iteration s, and are made according to an error-minimization policy described in the rest of this section.

Schapire and Singer (1999) have proven that the *Hamming loss* of the final hypothesis $\hat{\Phi}$, defined as the percentage of pairs $\langle d_i, c_j \rangle$ for which $sign(\Phi(d_i, c_j)) \neq sign(\hat{\Phi}(d_i, c_j))$, is at most $\Pi_{s=1}^{S} Z_s$. The Hamming loss of a hypothesis is a measure of its classification (in)effectiveness; therefore, a reasonable (although suboptimal) way to maximize the effectiveness of the final hypothesis $\hat{\Phi}$ is to "greedily" choose each weak hypothesis $\hat{\Phi}_s$ (and thus its parameters t_k, a_{0j} and a_{1j}) in such a way as to minimize the normalization factor Z_s.

Schapire and Singer (2000) define three different variants of AdaBoost.MH, corresponding to three different methods for making these choices:

1. AdaBoost.MH *with real-valued predictions* (here nicknamed AdaBoost.MH$^{\mathrm{R}}$);
2. AdaBoost.MH *with real-valued predictions and abstaining* (AdaBoost.MH$^{\mathrm{RA}}$);
3. AdaBoost.MH *with discrete-valued predictions* (AdaBoost.MH$^{\mathrm{D}}$).

In this paper we concentrate on AdaBoost.MH$^{\mathrm{R}}$, since it is the one that, in the experiments of Schapire and Singer (2000), has been experimented most thoroughly and has given the best results; however, everything we say in this paper about AdaBoost.MH$^{\mathrm{R}}$ straightforwardly applies to AdaBoost.MH$^{\mathrm{RA}}$ and AdaBoost.MH$^{\mathrm{D}}$.

At iteration s, AdaBoost.MH$^{\mathrm{R}}$ (from now on simply called AdaBoost.MH) chooses a weak hypothesis of the form described in Eq. 3 by the following algorithm.

Algorithm 1 (The AdaBoost.MH **weak learner**)

1. For each term $t_k \in \{t_1, \ldots, t_r\}$ select, among all weak hypotheses $\hat{\Phi}$ that have t_k as the "pivot term", the one (indicated by $\hat{\Phi}_{best(k)}$) for which Z_s is minimum.
2. Among all the hypotheses $\hat{\Phi}_{best(1)}, \ldots, \hat{\Phi}_{best(r)}$ selected for the r different terms in Step 1, select the one (indicated by $\hat{\Phi}_s$) for which Z_s is minimum.

Step 1 is clearly the key step, since there are a non-enumerable set of weak hypotheses with t_k as the pivot. Schapire and Singer (1999) have proven that, given term t_k and category c_j,

$$\hat{\Phi}_{best(k)}(d_i, c_j) = \begin{cases} \frac{1}{2} \ln \frac{W_{+1}^{0jk}}{W_{-1}^{0jk}} & \text{if } w_{ki} = 0 \\ \frac{1}{2} \ln \frac{W_{+1}^{1jk}}{W_{-1}^{1jk}} & \text{if } w_{ki} = 1 \end{cases} \tag{4}$$

where

$$W_b^{xjk} = \sum_{i=1}^{g} D_s(d_i, c_j) \cdot [\![w_{ki} = x]\!] \cdot [\![\Phi(d_i, c_j) = b]\!] \tag{5}$$

for $b \in \{1, -1\}$, $x \in \{0, 1\}$, $j \in \{1,\ldots,m\}$ and $k \in \{1,\ldots,r\}$, and where $[\![\pi]\!]$ indicates the characteristic function of predicate π (i.e. the function that returns 1 if π is true and 0 otherwise). For term t_k and for these values of a_{xj} we obtain

$$Z_s = 2\sum_{j=1}^{m}\sum_{x=0}^{1}(W_{+1}^{xjk}W_{-1}^{xjk})^{\frac{1}{2}} \tag{6}$$

Choosing $\frac{1}{2}\ln\frac{W_{+1}^{xjk}}{W_{-1}^{xjk}}$ as the value for a_{xj} has the effect that $\hat{\Phi}_s(d_i, c_j)$ outputs a positive real value in the two following cases:

1. $w_{ki} = 1$ (i.e. t_k occurs in d_i) and the majority of the training documents in which t_k occurs belong to c_j;
2. $w_{ki} = 0$ (i.e. t_k does not occur in d_i) and the majority of the training documents in which t_k does not occur belong to c_j.

In all the other cases $\hat{\Phi}_s$ outputs a negative real value. Here, "majority" has to be understood in a weighted sense, i.e. by bringing to bear the weight $D_s(d_i, c_j)$ associated to the training pair $\langle d_i, c_j\rangle$. The larger this majority is, the higher the absolute value of $\hat{\Phi}_s(d_i, c_j)$ is; this means that this absolute value represents a measure of the confidence that $\hat{\Phi}_s$ has in its own prediction (Schapire and Singer 1999).

In practice, the value $a_{xj} = \frac{1}{2}\ln\frac{W_{+1}^{xjk}+\epsilon}{W_{-1}^{xjk}+\epsilon}$ is chosen in place of $a_{xj} = \frac{1}{2}\ln\frac{W_{+1}^{xjk}}{W_{-1}^{xjk}}$, since this latter may produce outputs with a very large or infinite absolute value when the denominator is very small or zero.[6]

The output of the final hypothesis is the value

$$\hat{\Phi}(d_i, c_j) = \sum_{s=1}^{S}\hat{\Phi}_s(d_i, c_j) \tag{7}$$

obtained by summing the outputs of the weak hypotheses.

2.2 Implementing AdaBoost.MH

Following Sebastiani et al. (2000), in our implementation of AdaBoost.MH we have further optimized the final hypothesis $\hat{\Phi}(d_i, c_j) = \sum_{s=1}^{S}\hat{\Phi}_s(d_i, c_j)$ by "combining" the weak hypotheses $\hat{\Phi}_1, \ldots, \hat{\Phi}_S$ according to their pivot term t_k. In fact, note that if $\{\hat{\Phi}_1, \ldots, \hat{\Phi}_S\}$ contains a subset $\{\hat{\Phi}_1^{(k)}, \ldots, \hat{\Phi}_{q(k)}^{(k)}\}$ of weak hypotheses that all hinge on the same term t_k and are of the form

$$\hat{\Phi}_r^{(k)}(d_i, c_j) = \begin{cases} a_{0j}^r & \text{if } w_{ki} = 0 \\ a_{1j}^r & \text{if } w_{ki} = 1 \end{cases} \tag{8}$$

for $r = 1,\ldots,q(k)$, the collective contribution of $\hat{\Phi}_1^{(k)}, \ldots, \hat{\Phi}_{q(k)}^{(k)}$ to the final hypothesis is the same as that of a "combined hypothesis"

[6] In Schapire and Singer (2000) the value for ε is chosen by 3-fold cross validation on the training set, but this procedure is reported to give only marginal improvements with respect to the default choice of $\epsilon = \frac{1}{gm}$, which we adopt in this work.

$$\hat{\Phi}^{(k)}(d_i, c_j) = \begin{cases} \sum_{r=1}^{q(k)} a_{0j}^r & \text{if } w_{ki} = 0 \\ \sum_{r=1}^{q(k)} a_{1j}^r & \text{if } w_{ki} = 1 \end{cases} \quad (9)$$

In the implementation we have thus replaced $\sum_{s=1}^{S} \hat{\Phi}_s(d_i, c_j)$ with $\sum_{k=1}^{\Delta} \hat{\Phi}^{(k)}(d_i, c_j)$, where Δ is the number of different terms that act as pivot for the weak hypotheses in $\{\hat{\Phi}_1, \ldots, \hat{\Phi}_S\}$.

This modification brings about a considerable efficiency gain in the application of the final hypothesis to a test example. For instance, the final hypothesis we obtained on REUTERS-21578 with ADABOOST.MH when $S = 1{,}000$ consists of 1,000 weak hypotheses, but the number of different pivot terms is only 766. The reduction in the size of the final hypothesis which derives from this modification is usually larger when high reduction factors have been applied in a feature selection phase, since in this case the number of different terms that can be chosen as the pivot is smaller. A large reduction is also obtained when the total number of iterations S is high, since in this case the terms chosen as pivot in the last iterations tend to be ones that have been chosen already in previous iterations.

In this work we further implement two important optimizations for reducing classification time.

The first optimization consists in building the vectorial representations $\mathbf{d}_i$ of the test documents *after* the final hypothesis $\hat{\Phi} = \sum_{k=1}^{\Delta} \hat{\Phi}^{(k)}$ has been built, so that only the $|\Delta|$ features that act as pivot for some weak hypothesis in $\hat{\Phi}$ are actually included in the vectorial representations $\mathbf{d}_i$ of the test documents; the other terms can be discarded, since they play no role in the classification. Since it is usually the case that $|\Delta| \ll r$, this brings about a substantial reduction in the space occupied by the $\mathbf{d}_i$'s. For instance, the size of the vectors that we have obtained by this method on RCV1-v2 with ADABOOST.MH for $S = 1{,}000$ is 950, while the length r of the original vectors (see Sect. 5.3) was equal to 55,051.

The second optimization consists in sorting the final hypothesis $\hat{\Phi}$ (here viewed for convenience as a *sequence* $\hat{\Phi}^{(1)}, \ldots, \hat{\Phi}^{(\Delta)}$ of weak hypotheses) so that the terms that act as pivot appear in the same order as they appear in the vectorial representations of the documents. As a consequence, we can indeed view the final hypothesis as consisting of the $2m$ vectors $\mathbf{a}_{0j} = \langle a_{0j}^1, \ldots, a_{0j}^{\Delta} \rangle$ and $\mathbf{a}_{1j} = \langle a_{1j}^1, \ldots, a_{1j}^{\Delta} \rangle$ (for $j = 1,\ldots,m$) that contain the constants output by the compressed hypotheses of Eq. 9. Classification thus amounts to computing

$$\hat{\Phi}(d_i, c_j) = \sum_{k=1}^{\Delta} w_{ki} a_{1j}^k + \sum_{k=1}^{\Delta} (1 - w_{ki}) a_{0j}^k$$

Since one of w_{ki} and $(1 - w_{ki})$ is always 0, this amounts to a sum of Δ real numbers. This is extremely cheap, also due to the fact that Δ is typically small. Note for example that performing classification with any linear classifier (such as those generated by support vector machines) requires a dot product of length r, which is much more expensive than a sum of $\Delta \ll r$ reals. This makes our system even more classification-time efficient than other leading-edge technologies.

3 A hierarchical version of AdaBoost.MH for multi-label TC

In this section we describe a version of AdaBoost.MH, called TreeBoost.MH, that is explicitly designed to work on tree-structured sets of categories, and is capable of leveraging on the information inherent in this structure.

3.1 Notation, definitions, and the semantics of hierarchies

Before discussing the intuitions on which TreeBoost.MH is based, let us first fix some notation and definitions. Let C be a tree-structured set of categories, and let r be its root category. For each category $c_j \in C$, we will use the following abbreviations:

Symbol	Meaning
$\uparrow (c_j)$	The parent category of c_j
$\downarrow (c_j)$	The set of children categories of c_j
$\Uparrow (c_j)$	The set of ancestor categories of c_j
$\Downarrow (c_j)$	The set of descendant categories of c_j
$\leftrightarrow (c_j)$	The set of sibling categories of c_j

When discussing an HTC application it is always important to specify what the *semantics of the hierarchy* is, i.e., to specify the semantic constraints that a supposedly perfect classifier would enforce; which constraints are in place has important consequences on which algorithms we might want to apply to this task, and, more importantly, on how we should evaluate these algorithms.

For instance, one should specify whether a document can belong to zero, one, or several categories in C (which is indeed the case of this paper) or whether it always belongs to one and only one category in C.

No less importantly, one should specify whether it is the case that

1. a document d that is a positive example of a category c_j is also a positive example of all its ancestor categories $\Uparrow (c_j)$. We assume this to be the case. We say that, for the categories in $\Uparrow (c_j)$, d is a *bubbled-up* positive example (in the sense that it has bubbled up to c_j from somewhere down below).
2. a document d can in principle be a positive example of an internal node category c_j and at the same time *not* be a positive example of any of its descendant categories $\Downarrow (c_j)$. We assume this to be the case. We say that d is an *own* positive example of c_j.

Assumption 2 is indeed useful for tackling datasets, such as RCV1-v2 (see Sect. 5.1) in which documents with these characteristics do occur, while at the same time not preventing us to deal with datasets with the opposite characteristics. A consequence of these two assumptions is that

$$Tr^{+}(c_j) \supseteq \bigcup_{c \in \Downarrow (c_j)} Tr^{+}(c) \tag{10}$$

i.e., the set $Tr^{+}(c_j)$ of the positive training examples of a nonleaf category c_j is a (possibly proper) superset of the union of the sets of positive training examples of all its descendant (leaf) categories.

3.2 The rationale

TreeBoost.MH (which is fully illustrated in Fig. 2) embodies several intuitions that had arisen before within HTC.

The first, fairly obvious intuition (which lies at the basis of practically all HTC algorithms proposed in the literature) is that, in a hierarchical context, the classification of a document d_i is to be seen as a descent through the hierarchy, from the root to the (internal or leaf node) categories where d_i is deemed to belong. In ML classification this means that each nonroot category c_j has an associated binary classifier $\hat{\Phi}_j$ which acts as a "filter" that prevents unsuitable documents to percolate to lower levels. All test documents that a classifier $\hat{\Phi}_j$ deems to belong to c_j are passed as input to all the binary classifiers corresponding to the categories in $\downarrow(c_j)$, while the documents that $\hat{\Phi}_j$ deems not to belong to c_j are "blocked" and analysed no further. Note that it may well be the case that a document d_i is deemed to belong to c_j by $\hat{\Phi}_j$ and is then rejected by all the binary classifiers corresponding to the categories in $\downarrow(c_j)$; this is indeed consistent with assumption (b) above. In the end, each document may thus reach zero, one, or several (leaf or internal node) categories, and is thus classified under them.

The second intuition is that the training of $\hat{\Phi}_j$ should be performed "locally", i.e. by paying attention to the topology of the classification scheme. To see this note that, during classification, if the classifier for $\uparrow(c_j)$ has performed reasonably well, $\hat{\Phi}_j$ will only (or mostly) be presented with documents that belong to the subtree rooted in $\uparrow(c_j)$, i.e. with documents that belong to c_j and/or to some of the categories in $\leftrightarrow(c_j)$. As a result, the training of $\hat{\Phi}_j$ should be performed by using, as negative training examples, the positive training examples of $\uparrow(c_j)$, with the obvious exception of the documents that are also positive training examples of c_j. In particular, training documents that only belong to categories other than those in $\Downarrow(\uparrow(c_j))$ need not be used. The rationale of this choice is

1 *Input:* A pair $\langle C, r\rangle$ where C is a tree-structured set of categories and r is the root category of C

2 *Body:* **if** r is a leaf category **then** do nothing

3 **else begin**

4 Let $\downarrow(r) = \{\delta_1(r), \ldots, \delta_{k(r)}(r)\}$ be the $k(r)$ children categories of r;

5 Optionally run a ML feature selection algorithm on $\downarrow(r)$;

6 Run AdaBoost.MH on $\downarrow(r)$;

7 **for** $q = 1, \ldots, k(r)$ **do**

8 **begin**

9 Let T_q be the subtree of C rooted at $\delta_q(r)$;

10 Run TreeBoost.MH on $\langle T_q, \delta_q(r)\rangle$;

11 **end**

12 **end**

13 *Output:* For each nonleaf category $c_t \in C$, a final hypothesis $\hat{\Phi}^{(t)}(d,c) = \sum_{s=1}^{S} \hat{\Phi}_s^{(t)}(d,c)$ for $c \in \downarrow(c_t)$

Fig. 2 The TreeBoost.MH algorithm

that the negative training examples thus selected are "quasi-positive" examples of c_j (Schapire et al. 1998), i.e. are the negative examples that are closest to the boundary between the positive and the negative region of c_j (a notion akin to that of "support vectors" in SVMs), and are thus the most informative negative examples that can be used in training. This is beneficial also from the standpoint of (both training and classification time) efficiency, since fewer training examples and fewer features are involved. In a similar form, this intuition (which we discuss at large in Fagni and Sebastiani 2007) had first been presented in Ng et al. (1997) and Wiener et al. (1995).

The third intuition is similar, i.e. that feature selection should also be performed "locally", by paying attention to the topology of the classification scheme. As above, if the classifier for $\uparrow(c_j)$ has performed reasonably well, $\hat{\Phi}_j$ will only (or mostly) be presented with documents that belong to the subtree rooted in $\uparrow(c_j)$. As a consequence, for the classifiers corresponding to c_j and its siblings, it is cost-effective to employ features that are useful in discriminating (only) among themselves and $\uparrow(c_j)$; features that discriminate among categories lying outside the subtree rooted in $\uparrow(c_j)$ are too general, and features that discriminate among the subcategories of c_j, or among the subcategories of one of its siblings, are too specific. This intuition, albeit in the slightly different context of single-label classification, was first presented in Koller and Sahami (1997).

TreeBoost.MH also embodies the novel intuition that the weight distribution that boosting algorithms update at every boosting round should likewise be updated "locally". In fact, the two previously discussed intuitions indicate that hierarchical ML classification is best understood as consisting of several independent (flat) ML classification problems, one for each internal node of the hierarchy: for each such node c_j we must generate a number of binary classifiers, one for each $c_q \in \downarrow(c_j)$. In a boosting context, this means that several independent distributions, each one "local" to an internal node and its children, should be generated and updated by the process. In this way, the "difficulty" of a category c_q will only matter *relative* to the difficulty of its sibling categories. As discussed in Sect. 4, this intuition is of key importance in allowing TreeBoost.MH to obtain exponential savings in the cost of training over AdaBoost.MH.

3.3 The algorithm

TreeBoost.MH incorporates these four intuitions by factoring the hierarchical ML classification problem into several "flat" ML classification problems, one for every internal node in the tree. TreeBoost.MH learns in a recursive fashion, by identifying internal nodes c_j and calling AdaBoost.MH to generate a ML (flat) classifier for the set of categories $\downarrow(c_j)$. Alternatively (and more conveniently), this process may be viewed as generating, for each nonroot category $c_j \in C$, a binary classifier $\hat{\Phi}$ for c_j, by means of which hierarchical classification can be performed as described in Sect. 3.2.

Learning in TreeBoost.MH proceeds by first identifying whether a leaf category has been reached (line 6 of Fig. 2), in which case nothing is done, since the classifiers are generated only at internal nodes.

If an internal node c_j has been reached, a ML feature selection process may (optionally) be run (line 10) to generate a reduced feature set on which the ML classifier for $\downarrow(c_j)$ will operate. This may be dubbed a "glocal" feature selection policy, since it takes an intermediate stand between the well-known "global" policy (in which the same set of features is selected for all the categories in C) and "local" policy (in which a different set of features is chosen for each different category in $\downarrow(c_j)$). The glocal policy selects a different

set of features for each (maximal) set of sibling categories in C, thus implementing a view of feature selection as described in Sect. 3.2.[7] Any of the standard feature scoring functions (e.g. information gain, chi-square, odds ratio) can be used, as well as any of the standard feature score globalization methods (e.g. max, weighted average, Forman's (2004) round robin). Note that all these functions require a precise notion of what the positive and negative training examples of a category are; here, consistently with the "locality" principle discussed in Sect. 3.2, the negative training examples of a category c are taken to be the set $Tr^+(\uparrow(c)) - Tr^+(c)$.

After the reduced feature set has been identified, TreeBoost.MH calls upon AdaBoost.MH (line 11) to solve a ML (flat) classification problem for the categories in $\downarrow(c_j)$; again, in order to implement the "quasi-positive" policy discussed in Sect. 3.2, the negative training examples of a category c are taken to be the set $Tr^+(\uparrow(c)) - Tr^+(c)$. Note that restricting the AdaBoost.MH call to the categories in $\downarrow(c_j)$ implements the view, discussed in Sect. 3.2, of several independent, "local" distributions being generated and updated during the boosting process.

Finally, after the ML classifier for $\downarrow(c_j)$ has been generated, for each category $c_q \in \downarrow(c_j)$ a recursive call to TreeBoost.MH is issued (lines 12–18) that processes the subtree rooted in c_q in the same way. The final result is a hierarchical ML classifier in the form of a tree of binary classifiers, one for each nonroot node, each consisting of a committee of S decision stumps.

Note that the generated classifiers would allow us to implement another, alternative view of what the hierarchical ML classifier consists of: instead of a tree of committees, we might have a committee of trees, with each tree T_s having a single decision stump (the one generated at iteration s) at each nonroot node. In this paper we concentrate on the former view, leaving the latter for future investigation.

4 The computational cost of TreeBoost.MH

We now analyse the computational costs of AdaBoost.MH and TreeBoost.MH, and show that the latter is computationally cheaper than the former, allowing exponential savings at both training and testing time with respect to the former.

Let us first discuss the cost of classifier training. The key steps of AdaBoost.MH are (i) computing, for each $t_k \in T$, the Z_s factor resulting from $\hat{\Phi}_{best(k)}$, and (ii) computing the minimum, over all t_k, of such Z_s factors. By inspecting Eqs. 5 and 6 we can clearly see that, for each t_k, Step (i) requires $O(gm)$ operations for each t_k, where g is the number of training documents and m is the number of categories; since there are r such terms, the entire step requires $O(gmr)$ operations.

The cost of classifier training in TreeBoost.MH heavily depends on the topology of the tree and on the distribution of positive training examples across the nodes of the tree; in particular, it depends from factors such as the ariety (i.e. branching factor) of each individual internal node, the depth h_j of each individual node c_j, and the number $Tr^+(c_j)$ of positive training examples in each such node. We will thus limit our analysis to the best case and the worst case, since they are more easily identifiable; the cost of the other cases will be intermediate between these two. The worst possible case is that of a "flat",

[7] Note that a local policy would also implement this view, but is not made possible by AdaBoost.MH, since this latter uses the same set of features for all the categories involved in the ML classification problem. This means that we need to use the same set of features for all categories in $\downarrow(c_j)$.

degenerate tree of height 1, i.e. a tree in which all leaf categories are children of the root category and there are no internal nodes aside from the root itself. In this case, TreeBoost.MH calls AdaBoost.MH exactly once, and on the entire category set, which means that the two algorithms coincide, and have thus the same cost. The best possible case is more interesting, and coincides with the "fully grown" case of a perfectly balanced tree of constant ariety a (in this case the height of the tree is $h = \log_a m$) in which leaf categories have all the same frequency and each document belongs to exactly one leaf category. At each level $l = 1,\ldots,h$ of such a tree (the root is conventionally assigned level 0) TreeBoost.MH calls AdaBoost.MH exactly a^{l-1} times. Since, as from the analysis above, AdaBoost.MH is $O(gmr)$ in the general case, this means that in this case each call to AdaBoost.MH requires $O(\frac{g}{a^{l-1}}ar)$ operations, given that (i) the training examples involved are not g but only $\frac{g}{a^{l-1}}$ (since we have made the hypothesis that leaf categories are evenly populated and each training example belongs to exactly one leaf category) and (ii) only a (instead of m) categories are involved. This means that the number of operations required by TreeBoost.MH is

$$O\left(\sum_{l=1}^{h} a^{l-1} \cdot \frac{g}{a^{l-1}} ar\right) = O\left(\sum_{l=1}^{h} gar\right) = O(garh)$$

This means that, for each of the g training examples and for each of the r terms, TreeBoost.MH performs $O(ah)$ operations and AdaBoost.MH performs $O(m)$ operations. Given that $m = a^h$, this means that, at training time, TreeBoost.MH is cheaper than AdaBoost.MH by an exponential factor.

Let us then discuss the cost of testing (i.e. applying) the generated classifiers. Again, in the "flat" worst case discussed above the two algorithms are trivially the same. Let us then only analyse the "fully grown" best case, in the understanding that the cost of the other cases will be intermediate between these two. In AdaBoost.MH, each test document must be given as input to $O(S)$ weak hypotheses, each of which performs 1 test and m additions, one per category; the cost is thus $O(Sm)$.[8] In TreeBoost.MH, each test document is input to $O(h)$ classifiers (corresponding to one or more—complete or incomplete—paths downwards from the root), each of them consisting of $O(S)$ weak hypotheses each of which performs 1 test and a additions, one per category; the cost is thus $O(Sah)$. Recalling that $m = a^h$, we can see that TreeBoost.MH is cheaper than AdaBoost.MH by an exponential factor at testing time too.

5 Experimental results

5.1 Datasets

The first benchmark we have used in our experiments is the "Reuters-21578, Distribution 1.0" corpus, one of the most widely used benchmarks in TC research.[9] In origin, the Reuters-21578 category set is not hierarchically structured, and is thus not suitable "as is"

[8] Our analysis is here in terms of the S original weak hypotheses rather than in terms of the Δ combined weak hypotheses; this is because different internal nodes have different values of Δ, which would needlessly complicate the notation and the discussion; the conclusions are not affected anyway.

[9] Reuters-21578 is freely available for experimentation purposes from http://www.daviddlewis.com/resources/testcollections/~reuters21578/.

Table 1 REUTERS-21578 macro-categories and their member categories (from Toutanova et al. 2001))

Macrocategory	Member categories
Commodities	Barley, carcass, castor-oil, cocoa, coconut, coconut-oil, coffee, copra-cake, corn cotton, cotton-oil, grain, groundnut, groundnut-oil, hog, l-cattle, lin-oil, livestock, lumber, meal-feed, oat, oilseed, orange, palm-oil, palmkernel, pet-chem, potato, rape-oil, rapeseed, rice, rubber, rye, ship, sorghum, soy-meal, soy-oil, soybean, sugar, sun-meal, sun-oil, sunseed, tea, veg-oil, wheat
Financial	acq, bop, cpi, cpu, dfl, dlr, dmk, earn, gnp, housing, income, instal-debt, interest, ipi, jobs, lei, money-fx, money-supply, nkr, nzdlr, rand, reserves, retail, trade, wpi, yen
Metals	Alum, copper, gold, iron-steel, lead, nickel, palladium, platinum, silver, strategic-metal, tin, zinc
Energy	Crude, fuel, gas, heat, jet, naphtha, nat-gas, propane

for HTC experiments; we have thus used a hierarchical version of it generated in Toutanova et al. (2001) by the application of hierarchical agglomerative clustering on the 90 REUTERS-21578 categories that have at least one positive training example and one positive test example. The original REUTERS-21578 categories are thus "leaf" categories in the resulting hierarchy, and are clustered into four "macro-categories" whose parent category is the root of the tree. Conforming to the experiments of Toutanova et al. (2001), we have used (according to the ModApte split) the 7,770 training examples and 3,299 test examples that are labelled by at least one of the selected categories; the average number of categories per document is 1.23, ranging from a minimum of 1 to a maximum of 15. The average number of positive examples per category is 106.50, ranging from a minimum of 1 to a maximum of 2,877 (Table 1).

The second benchmark we have used is REUTERS CORPUS VOLUME 1 version 2 (RCV1-v2),[10] a more recent text categorization benchmark made available by Reuters and consisting of 804,414 news stories produced by Reuters from 20 Aug 1996 to 19 Aug 1997; all news stories are in English, and have 109 distinct terms per document on average (Rose et al. 2002). In our experiments we have used the "LYRL2004" split defined in Lewis et al. (2004), in which the (chronologically) first 23,149 documents are used for training and the other 781,265 are used for testing. The documents are classified according to three different, orthogonal classification schemes: "topics, "industries" and "regions"; consistently with Lewis et al. (2004) and all the literature that has followed, we focus on the "topics" classification scheme. Out of the 103 "topics" categories, in our experiments we have restricted our attention to the 101 categories (21 internal node and 80 leaf node categories) with at least one positive training example. Of the three benchmarks we use in this work, RCV1-v2 is the only one containing (both training and test) examples that are "own" documents of internal node categories.; each of the 21 internal node categories has at least one own document. The RCV1-v2 hierarchy is four levels deep (including the root, to which we conventionally assign level 0); there are four internal nodes at level 1, and the leaves are all at the levels 2 and 3. The 80 leaf categories instead have 347.2 positive training examples on average.

The third benchmark we have used is the ICCCFT from the 2007 "International Challenge on Classifying Clinical Free Text Using Natural Language Processing",[11]

[10] http://trec.nist.gov/data/reuters/reuters.html.

[11] http://computationalmedicine.org/challenge/index.php

organized by the Computational Medicine Center of the Cincinnati Children's Hospital Medical Center and the University of Cincinnati Medical Center. The documents are short discharge reports classified according to the ICD-9-CM classification scheme,[12] the official system of assigning codes to diagnoses and procedures associated with hospital utilization in the United States. The experiments we present here use only the training set of the Challenge, since the labels of the test documents are not available to participants; unlike with the previous two benchmarks we thus compute effectiveness by leave-one-out cross-validation. There are only 978 documents in the training set, with an average length of 13.3 words. We restrict our experiments to the 79 categories that have at least one positive training document; of these 79 categories, 45 are leaf node categories and the other 34 are internal node categories, none of which has "own" documents. The ICD-9-CM hierarchy (or, at least, that part of it that is used for labelling our training data) is again four levels deep (including the root, to which we conventionally assign level 0); there are seven internal nodes at level 1, and the leaves are all at the levels 2 and 3. The average number of positive examples per leaf category is 27.1, ranging from a minimum of 1 and a maximum of 266. One peculiar feature of this dataset is that some nodes are single children, i.e., have no siblings. This makes it impossible to adopt our standard policy of choosing, as negative training examples of a category, the positive training documents of the father category. In these cases we merge father and child categories into a single node, since they always have the same positive and negative examples.

5.2 Averaging effectiveness across categories

As a measure of effectiveness that combines the contributions of *precision* (π) and *recall* (ρ) we have used the well-known F_1 function, defined as

$$F_1 = \frac{2\pi\rho}{\pi + \rho} = \frac{2TP}{2TP + FP + FN} \tag{11}$$

which corresponds to the harmonic mean of precision and recall, where TP stands for true positives, FP for false positives, and FN for false negatives. Note that F_1 is undefined when $TP = FP = FN = 0$; in this case, consistently with most other works in the literature, we take F_1 to equal 1.0, since the classifier has correctly classified all documents (as negative examples).

In text classification it is customary to average the category-specific F_1 scores by computing both microaveraged F_1 (denoted by F_1^{μ}) and macroaveraged F_1 (F_1^{M}). F_1^{μ} is obtained by (i) computing the category-specific values TP_i, (ii) obtaining TP as the sum of the TP_i's (same for FP and FN), and then (iii) applying Eq. 11. F_1^{M} is obtained by first computing the F_1 values specific to the individual categories, and then averaging them across the c_j's.

However, in HTC one should specify exactly which categories the average is computed across. Should this average be computed across leaf categories only, or should it involve internal node categories too? It might seem reasonable that also the internal nodes that have "own" positive examples (see Sect. 3.1) are considered, since these nodes are not to be viewed as merely routing documents to the subtrees below them. However, the presence of "bubbled-up" positive test examples within internal node categories means that, if averages involve these categories too, they will involve classification decisions that are not

[12] http://www.cdc.gov/nchs/about/otheract/icd9/abticd9.htm

independent of each other. For instance, the fact that test document d_i has been correctly classified under a leaf category c_j entails that d_i has also been correctly classified under all the categories in $\Uparrow(d_i)$, which means that this decision will count as n true positives (where n is the depth of c_j) instead of a single true positive.

The approach we take to averaging is intermediate between these two extremes, and involves distinguishing, for each internal node c_j, the roles of "own" and "bubbled-up" positive test examples. In turn, this corresponds to distinguishing the roles of internal nodes as "routers" towards its subordinate nodes or as "repositories" of documents in their own right (a distinction already addressed in Ruiz and Srinivasan 2002]).

The approach consists in

1. mapping the original hierarchy C into a modified hierarchy C' such that, for each internal node category $c_j \in C$ with "own" positive training examples, C' contains a "dummy" child (leaf) node c'_j appended to c_j;
2. moving into c'_j all of c_j's "own" positive test examples.

This simple mapping, originally proposed in Cheng et al. (2001), produces a hierarchy C' semantically equivalent to C in which all documents are contained in at least one leaf category.[13] For evaluation purposes we then use the modified hierarchy C' instead of the original hierarchy C (even if learning and classification have indeed used C), and average across leaf nodes only. The net effect is that we do take into consideration the ability of the system to correctly classify documents as "own" documents of internal nodes (in the modified hierarchy, this is represented by the system's effectiveness on "dummy" nodes), and at the same time we remove the dependence between classification decisions due to inherited examples.

5.3 The experiments

In all the experiments discussed in this section, punctuation has been removed, all letters have been converted to lowercase, numbers have been removed, stop words have been removed using the stop list provided in Lewis (1992, pp. 117–118), and stemming has been performed by means of Porter's stemmer.

In a first experiment we have compared AdaBoost.MH and TreeBoost.MH using a full feature set.

We have then performed a number of experiments using feature selection; however, these have been run on Reuters-21578 and RCV1-v2 only, due to the fact that the original feature set of ICCCFT has a very limited size already (1,294 terms only). Reduced feature sets were obtained according to a "global" feature selection policy in which (i) feature-category pairs have been scored by means of *information gain*, defined as

$$IG(t_k, c_i) = \sum_{c \in \{c_i, \bar{c}_i\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t, c) \cdot \log \frac{P(t, c)}{P(t) \cdot P(c)}$$

and (ii) the final set of features has been chosen according to Forman's *round robin* technique, which consists in picking, for each category c_j, the v features with the highest $IG(t_k, c_j)$ value, and pooling all of them together into a category-independent set (Forman

[13] Note that many real-world classification schemes (e.g. the ACM Classification Scheme) are of this latter type, since their internal nodes usually have a special child category (called General, or Other) which contains all documents belonging to the node but to none of its descendant leaves.

2004). This set thus contains a number of features $q \leq vm$, where m is the number of categories; it usually contains strictly fewer than vm, since it is usually the case that some features are among the best v features for more than one category. We have set v to 60 for REUTERS-21578 and to 43 for RCV1-v2, which are the values that, for each corpora, best approximate a total number of features of 2,000; in fact, the reduced feature sets consist of 2,012 features for REUTERS-21578 (11% of the 18,177 original ones) and 2,029 for RCV1-v2 (3.7% of the 55,051 original ones).

We have also run an experiment in which we have used the "glocal" feature selection policy described in Sect. 3.3, consisting in selecting a different subset of features (of the same cardinalities as in the global policy) for the set of children of each different internal node. Note that, for each corpus, the results obtained by means of this policy are reported only for TREEBOOST.MH, since this policy obviously is not applicable to ADABOOST.MH.

5.4 Effectiveness

The results of our experiments are reported in Table 2.

We will now comment on the REUTERS-21578 results;[14] the RCV1-v2 and ICCCFT results are qualitatively similar. The first observation we can make is that, in switching from ADABOOST.MH to TREEBOOST.MH, effectiveness improves substantially. F_1^{μ} improves from +2.3% to +17.2%, depending on the number S of boosting iterations. F_1^{M} improves even more substantially, from +22.0% to +197.4%; this means that TREEBOOST.MH is especially suited to categorization problems in which the distribution of training examples across the categories is highly skewed. For both F_1^{μ} and F_1^{M}, the improvements tend to be more substantial for low values of S, showing that TREEBOOST.MH converges to optimum performance more rapidly than ADABOOST.MH. Altogether, these effectiveness improvements are somehow surprising, since it is well-known that hierarchical TC can introduce a deterioration of effectiveness due to classification errors made high up in the hierarchy, which cannot be recovered at the lower levels (Koller and Sahami 1997; McCallum et al. 1998). The improvements thus show that the "filters" placed at the internal nodes work well, likely due to the fact that they their training benefits from using only the "quasi-positive" examples of local interest as negative training examples.

Concerning the RCV1-v2 dataset, note that the results obtained by both ADABOOST.MH and TREEBOOST.MH are inferior to the ones reported in Lewis et al. (2004) and obtained, on the same dataset, by SVM-based classification systems. One of the reasons is certainly the fact that F_1 is (micro- and macro-)averaged across different sets of categories. In fact, while the authors of Lewis et al. (2004) choose to include internal node categories in the average, as mentioned in Sect. 5.2 we only include their associated dummy nodes. By doing so we avoid "watering down" the evaluation by considering nodes (the internal ones) that are both "easy" (since they typically have many training examples, of the "bubbled-up" type) and scarcely significant from an application point of view (since their

[14] The reader might notice that the best performance we have obtained from ADABOOST.MH on REUTERS-21578 ($F_1^{\mu} = .808$) is inferior to the one reported in Schapire and Singer (2000) for the same algorithm ($F_1^{\mu} = .851$). There are several reasons for this: (a) Schapire and Singer (2000) actually uses a different, much older version of this collection, called REUTERS-21450 (Apté et al. 1994); (b) Schapire and Singer (2000) only uses the 93 categories which have at least 2 positive training examples and 1 positive test example, while we also use the categories that have just 1 positive training example and those that have no positive test example. This makes the two sets of ADABOOST.MH results difficult to compare.

Table 2 AdaBoost.MH and TreeBoost.MH on Reuters-21578 (top 5 rows), RCV1-v2 (mid 5 rows) and ICCCFT (bottom two rows)

		5 Iterations	10 Iterations	20 Iterations	50 Iterations	100 Iterations	200 Iterations	500 Iterations	1,000 Iterations
Reuters-21578	AdaBoost.MH (full feature set)	.533	.597	.664	.724	.783	.798	.804	.808
		.033	.075	.160	.255	.332	.361	.377	.379
		34.0	68.1	136.3	340.7	681.5	1362.9	3407.3	6814.6
		11.1	14.3	18.2	35.1	66.2	129.9	274.0	464.3
	TreeBoost.MH (full feature set)	.596 (+11.9%)	.699 (+17.2%)	.745 (+12.2%)	.795 (+9.9%)	.810 (+3.5%)	.827 (+3.7%)	**.830** (+3.1%)	.826 (+2.3%)
		.100 (+197.4%)	.187 (+148.4%)	.286 (+78.9%)	.416 (+62.6%)	.425 (+28.2%)	.454 (+25.6%)	.460 (+22.0%)	**.479** (+26.4%)
		16.9 (−50.4%)	33.8 (−50.4%)	67.6 (−50.4%)	169.0 (−50.4%)	337.9 (−50.4%)	675.8 (−50.4%)	1689.4 (−50.4%)	3378.9 (−50.4%)
		13.0 (+16.8%)	12.6 (−11.8%)	17.2 (−5.5%)	20.6 (−41.4%)	29.9 (−54.9%)	48.5 (−62.7%)	96.6 (−64.7%)	151.3 (−67.4%)
	AdaBoost.MH (global FS)	.533	.597	.664	.724	.783	.799	.811	.801
		.034	.075	.160	.256	.332	.354	.373	.362
		24.6	49.2	98.4	246.1	492.1	984.3	2460.8	4921.5
		8.8	12.4	17.0	32.8	59.6	112.2	255.2	386.0
	TreeBoost.MH (global FS)	.596 (+11.9%)	.699 (+17.2%)	.744 (+11.9%)	.800 (+10.5%)	.809 (+3.4%)	.815 (+2.0%)	.828 (+2.1%)	.821 (+2.5%)
		.100 (+197.4%)	.187 (+148.4%)	.285 (+78%)	.437 (+70.8%)	.427 (+28.7%)	.457 (+28.8%)	.457 (+22.4%)	.473 (+30.6%)
		11.5 (−53.4%)	23.0 (−53.4%)	45.9 (−53.4%)	114.7 (−53.4%)	229.5 (−53.4%)	459.0 (−53.4%)	1147.4 (−53.4%)	2294.7 (−53.4%)
		9.1 (+3.2%)	9.6 (−22.1%)	11.3 (−33.5%)	17.2 (−47.7%)	26.3 (−55.9%)	42.4 (−62.2%)	82.3 (−67.7%)	131.3 (−66.0%)
	TreeBoost.MH (glocal FS)	.596 (+11.9%)	.699 (+17.2%)	.744 (+11.9%)	.794 (+9.7%)	.812 (+3.8%)	.817 (+2.2%)	.824 (+1.6%)	.825 (+3.0%)
		.100 (+197.4%)	.187 (+148.4%)	.285 (+77.9%)	.401 (+56.9%)	.430 (+29.8%)	.460 (+29.8%)	.465 (+24.7%)	.465 (+28.3%)
		12.1 (−50.7%)	24.2 (−50.7%)	48.5 (−50.7%)	121.2 (−50.7%)	242.5 (−50.7%)	485.0 (−50.7%)	1212.4 (−50.7%)	2424.8 (−50.7%)
		10.7 (+21.6%)	14.9 (+20.9%)	14.1 (−16.7%)	23.0 (−29.8%)	28.3 (−52.4%)	46.2 (−58.9%)	94.6 (−62.9%)	142.9 (−63.0%)

Table 2 continued

		5 Iterations	10 Iterations	20 Iterations	50 Iterations	100 Iterations	200 Iterations	500 Iterations	1,000 Iterations
RCV1-v2 (ORIGINAL LEAF NODES + DUMMY NODES)	AdaBoost.MH (full feature set)	.026	.105	.187	.331	.415	.471	.521	.537
		.005	.021	.072	.165	.239	.296	.333	.340
		181.3	362.7	725.3	1813.3	3626.5	7253.1	18132.7	36265.5
		3346.2	3576.4	5168.2	9524.7	16827.2	33608.9	83951.2	170720.3
	TreeBoost.MH (full feature set)	.160 (+519.3%)	.274 (+161.9%)	.360 (+93.1%)	.488 (+47.5%)	.542 (+30.9%)	.579 (+23.0%)	.604 (+16.1%)	**.615** (+14.4%)
		.045 (+762.0%)	.093 (+335.0%)	.154 (+113.9%)	.270 (+63.8%)	.330 (+37.9%)	.367 (+23.7%)	.388 (+16.6%)	.399 (+17.3%)
		78.3 (−56.8%)	156.5 (−56.8%)	313.1 (−56.8%)	782.7 (−56.8%)	1565.3 (−56.8%)	3130.6 (−56.8%)	7826.6 (−56.8%)	15653.1 (−56.8%)
		2774.5 (−17.1%)	2813.5 (−21.3%)	3081.1 (−40.4%)	3963.2 (−58.4%)	6044.2 (−64.1%)	9328.3 (−72.2%)	21847.4 (−74.0%)	38342.9 (−77.5%)
	AdaBoost.MH (global FS)	.026	.105	.187	.331	.419	.473	.526	.545
		.005	.021	.072	.165	.243	.300	.341	.354
		107.8	215.5	431.1	1077.7	2155.5	4310.9	10777.3	21554.7
		1598.4	2223.7	3741.0	8012.8	16206.9	31907.7	74292.3	147354.1
	TreeBoost.MH (global FS)	.160 (+519.3%)	.273 (+161.5%)	.360 (+92.7%)	.486 (+46.9%)	.540 (+28.8%)	.578 (+22.2%)	.604 (+14.9%)	.608 (+11.7%)
		.045 (+762.0%)	.093 (+333.9%)	.155 (+114.4%)	.273 (+65.1%)	.334 (+37.0%)	.371 (+23.5%)	.402 (+17.7%)	**.406** (+14.6%)
		33.8 (−68.7%)	67.5 (−68.7%)	135.1 (−68.7%)	337.6 (−68.7%)	675.3 (−68.7%)	1350.5 (−68.7%)	3376.4 (−68.7%)	6752.8 (−68.7%)
		1830.2 (+14.5%)	1422.0 (−36.1%)	1901.0 (−49.2%)	2772.1 (−65.4%)	4836.0 (−70.2%)	8156.6 (−74.4%)	18384.5 (−75.3%)	33648.3 (−77.2%)
	TreeBoost.MH (glocal FS)	.160 (+519.1%)	.275 (+162.8%)	.365 (+95.5%)	.487 (+47.1%)	.537 (+28.3%)	.578 (+22.2%)	.604 (+14.9%)	.613 (+12.7%)
		.045 (+762.0%)	.093 (+337.6%)	.159 (+119.7%)	.264 (+60.1%)	.326 (+33.9%)	.364 (+21.1%)	.389 (+14.1%)	.401 (+13.4%)
		41.3 (−61.7%)	82.6 (−61.7%)	165.3 (−61.7%)	413.1 (−61.7%)	826.3 (−61.7%)	1652.6 (−61.7%)	4131.4 (−61.7%)	8268.9 (−61.7%)
		2374.9 (+48.6%)	2432.7 (+9.4%)	2499.9 (−33.2%)	3645.9 (−54.5%)	5020.3 (−69.0%)	8372.9 (−73.8%)	18173.9 (−75.5%)	33149.0 (−77.5%)

Table 2 continued

		5 Iterations	10 Iterations	20 Iterations	50 Iterations	100 Iterations	200 Iterations	500 Iterations	1,000 Iterations
ICCCFT	AdaBoost.MH (full feature set)	.630	.704	.806	.813	.827	.823	.827	.819
		.084	.159	.303	.345	.409	.416	.432	.433
		602.8	1205.7	2411.3	6028.3	12056.6	24113.2	60282.9	120565.9
		5.9	9.8	11.7	17.6	21.5	33.3	48.9	78.2
	TreeBoost.MH (full feature set)	.704 (+11.7%)	.757 (+7.5%)	.803 (−0.4%)	.824 (+1,4%)	.830 (+0.4%)	**.831** (+1.0%)	.828 (+0.1%)	.813 (−0.7%)
		.212 (+152.4%)	.284 (+78.6%)	.365 (+20.5%)	.403 (+16.8%)	.429 (+4.9%)	.471 (+13.2%)	**.470** (+8.8%)	.469 (+8.3%)
		317.4 (−47.3%)	634.9 (−47.3%)	1269.7 (−47.3%)	3174.3 (−47.3%)	6348.7 (−47.3%)	12697.4 (−47.3%)	31743.4 (−47.3%)	63486.9 (−47.3%)
		2.9 (−50%)	3.9 (−60%)	8,8 (−25.0%)	9.8 (−44.4%)	10.8 (−50.0%)	12.7 (−61.8%)	15.6 (−68.0%)	20.5 (−73.8%)

In each square, the first figure from top is F_1^{μ}, the second is F_1^{M}, the third is training time $\tau(Tr)$ (inclusive of the time required to perform feature selection, if any), and the fourth is testing time $\tau(Te)$. The best effectiveness results obtained on each collection are indicated in boldface

most important role is as routers towards the leaves, rather than as categories in their own right).

Obviously, this means that the classification problem we tackle is more difficult that the one tackled in Lewis et al. (2004); in fact, easy nodes are now removed from consideration, while "hard" ones (i.e., the dummy ones, which have very few training examples) are introduced. This is best appreciated by looking at Tables 3 and 4, in which the effectiveness of the classifiers is computed separately for original leaves (Table 3) and dummy nodes (Table 4). The fact that effectiveness is very, very low on dummy nodes shows that including them in the averages from which Table 2 was computed has considerably reduced the resulting values of effectiveness.

5.5 Significance testing

In order to check whether these results are statistically significant we have subjected them to thorough statistical significance testing, by applying to the results reported in Table 2 (we use those for $S = 1{,}000$ boosting iterations) all the significance tests defined for text classification systems in Yang and Liu (1999), Sect. 4) i.e.:

1. the *s-test*: a sign test (Spiegel and Stephens 1999, Chapter 17) which compares two classifiers $\hat{\Phi}_1$ and $\hat{\Phi}_2$ by analysing their binary decisions on each document/category pair;
2. the *S-test* on F_1: a sign test which compares two classifiers $\hat{\Phi}_1$ and $\hat{\Phi}_2$ by analysing the paired F_1 scores on individual categories;
3. the *T-test* on F_1: a *t*-test (Spiegel and Stephens 1999, Chapter 11) which compares two classifiers $\hat{\Phi}_1$ and $\hat{\Phi}_2$ by analysing the paired F_1 values on individual categories;
4. the *T′-test* on F_1: a "*t*-test after rank transformation" which compares two classifiers $\hat{\Phi}_1$ and $\hat{\Phi}_2$ by analysing the rank positions (1st or 2nd) that the two systems have obtained, in terms of F_1, on each individual category;
5. the *p-test* on π^{μ} and ρ^{μ}: a *t*-test which compares two classifiers $\hat{\Phi}_1$ and $\hat{\Phi}_2$ by analysing the microaveraged precision and recall values that the two systems have obtained.

Tests 1 and 5 are designed to evaluate the two systems at the ("micro") level of individual classification decisions, while Tests 2, 3 and 4 are designed to evaluate them at the ("macro") level of individual categories, i.e., by analysing the performance scores that the two systems have obtained on such categories. We refer the interested reader to (Yang and Liu 1999, Sect. 4] for full mathematical definitions and for a discussion of the strengths and weaknesses of these five tests; we here only note, along with (Yang and Liu 1999, p. 47), that "none of the tests is 'perfect' for all the performance measures, or for performance analysis with respect to a skewed category distribution, so using them jointly instead of using one test alone would be a better choice".

Table 5 clearly shows that the results reported in Table 2 are statistically significant. In fact, in 33 out of 42 tests (6 significance tests times the 7 different scenarios in which TreeBoost.MH and AdaBoost.MH are compared) TreeBoost.MH turns out to be statistically significantly better than AdaBoost.MH at a p-value ≤ 0.01 AdaBoost.MH (this corresponds to the cells marked ("$\ll$"), while in other 4 tests this holds only at a p-value ≤ 0.05 (cells marked "$<$"); only in the remaining 5 tests no statistically significant difference is found at p-values > 0.05 (cells marked "$\sim$").

Note that this result becomes even stronger if we restrict ourselves to the largest of the tested collection (namely, RCV1-v2), on which TreeBoost.MH turns out to be statistically

Table 3 AdaBoost.MH and TreeBoost.MH on RCV1-v2; averaging is performed across all original leaf categories (i.e., "dummy" categories are not considered)

		5 Iterations	10 Iterations	20 Iterations	50 Iterations	100 Iterations	200 Iterations	500 Iterations	1,000 Iterations
RCV1-v2 (original leaf categories only)	AdaBoost.MH (full feature set)	.008	.086	.211	.393	.492	.548	.597	.614
		.002	.018	.073	.183	.272	.335	.375	.383
	TreeBoost.MH (full feature set)	.165 (+2056.8%)	.288 (+234.8%)	.398 (+88.8%)	.533 (+35.5%)	.589 (+19.7%)	.622 (+13.4%)	.644 (+7.9%)	**.654** (+6.4%)
		.048 (+2632.1%)	.104 (+471.9%)	.172 (+136.2%)	.303 (+65.5%)	.369 (+35.7%)	.407 (+21.6%)	.432 (+15.3%)	.444 (+16.1%)
	AdaBoost.MH (global FS)	.008	.086	.211	.393	.495	.553	.602	.621
		.002	.018	.073	.183	.277	.340	.385	.398
	TreeBoost.MH (global FS)	.165 (+2056.8%)	.285 (+232.1%)	.397 (+88.2%)	.533 (+35.6%)	.586 (+18.4%)	.621 (+12.3%)	.645 (+7.1%)	.648 (+4.5%)
		.048 (+2632.1%)	.104 (+470.6%)	.172 (+136.1%)	.305 (+66.6%)	.375 (+35.2%)	.415 (+21.9%)	.449 (+16.5%)	**.453** (+13.7%)
	TreeBoost.MH (glocal FS)	.165 (+2056.3%)	.286 (+233.3%)	.398 (+88.6%)	.532 (+35.3%)	.582 (+17.7%)	.621 (+12.4%)	.645 (+7.2%)	.654 (+5.4%)
		.048 (+2632.0%)	.105 (+475.1%)	.173 (+137.6%)	.296 (+61.4%)	.365 (+31.7%)	.406 (+19.3%)	.434 (+12.8%)	.448 (+12.5%)

In each square, the first figure from top is F_1^{μ} and the second is F_1^{M}. The best effectiveness results obtained are indicated in boldface

Table 4 AdaBoost.MH and TreeBoost.MH on RCV1-v2; averaging is performed across "dummy" categories only

		5 Iterations	10 Iterations	20 Iterations	50 Iterations	100 Iterations	200 Iterations	500 Iterations	1,000 Iterations
RCV1-v2 (dummy categories only)	AdaBoost.MH (full feature set)	.058	.133	.146	.205	.234	.270	.315	.328
		.018	.033	.070	.095	.115	.150	.172	.178
	TreeBoost.MH (full feature set)	.145 (+148.1%)	.235 (+76.3%)	.261 (+78.4%)	.347 (+69.3%)	.388 (+65.7%)	.424 (+56.7%)	.454 (+44.1%)	**.465** (+41.7%)
		.030 (+63.5%)	.049 (+48.0%)	.087 (+24.8%)	.144 (+51.1%)	.181 (+57.8%)	.213 (+41.6%)	.219 (+27.8%)	**.227** (+27.1%)
	AdaBoost.MH (global FS)	.058	.133	.146	.205	.239	.268	.320	.337
		.018	.033	.070	.095	.116	.147	.174	.185
	TreeBoost.MH (global FS)	.145 (+148.1%)	.240 (+80.1%)	.261 (+78.6%)	.340 (+66.0%)	.385 (+61.0%)	.422 (+57.2%)	.450 (+40.5%)	.455 (+35.2%)
		.030 (+63.5%)	.049 (+47.5%)	.089 (+27.8%)	.147 (+54.3%)	.177 (+53.2%)	.203 (+38.1%)	.223 (+28.0%)	.225 (+21.7%)
	TreeBoost.MH (glocal FS)	.145 (+148.1%)	.242 (+82.2%)	.278 (+90.3%)	.347 (+69.3%)	.387 (+62.0%)	.421 (+57.0%)	.450 (+40.6%)	.458 (+36.2%)
		.030 (+63.5%)	.049 (+49.3%)	.103 (+48.1%)	.144 (+50.5%)	.178 (+54.1%)	.202 (+37.3%)	.218 (+25.2%)	.224 (+20.9%)

In each square, the first figure from top is F_1^{μ} and the second is F_1^{M}. The best effectiveness results obtained are indicated in boldface

Table 5 Statistical significance tests on the results of Table 2

	$\hat{\Phi}_1$	$\hat{\Phi}_2$	s-test	S-test on F_1	T-test on F_1	T'-test on F_1	p-test on π^{μ}	p-test on ρ^{μ}
REUTERS-21578	ADABOOST.MH (full)	TREEBOOST.MH (full)	$<$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	ADABOOST.MH (glob)	TREEBOOST.MH (glob)	$\sim$	$\ll$	$<$	$<$	$\ll$	$\ll$
	ADABOOST.MH (glob)	TREEBOOST.MH (gloc)	$\sim$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	TREEBOOST.MH (gloc)	TREEBOOST.MH (glob)	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
RCV1-v2	ADABOOST.MH (full)	TREEBOOST.MH (full)	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	ADABOOST.MH (glob)	TREEBOOST.MH (glob)	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	ADABOOST.MH (glob)	TREEBOOST.MH (gloc)	$\ll$	$\ll$	$\ll$	$\ll$	$<$	$\ll$
	TREEBOOST.MH (gloc)	TREEBOOST.MH (glob)	$\ll$	$\sim$	$\sim$	$\sim$	$\ll$	$\ll$
ICCCFT	ADABOOST.MH (full)	TREEBOOST.MH (full)	$\sim$	$\ll$	$\ll$	$<$	$<$	$\ll$

All tests are conducted on the results obtained with $S = 1{,}000$ boosting iterations; "full", "glob" and "gloc" stand for no feature selection, global feature selection, and glocal feature selection, respectively; "$\ll$" (resp., "$<$") means that classifier $\hat{\Phi}_2$ is statistically significantly better than classifier $\hat{\Phi}_1$ at a p-value ≤ 0.01 (resp., ≤ 0.05); "$\sim$" means that classifier $\hat{\Phi}_2$ is *not* statistically significantly better than classifier $\hat{\Phi}_1$ at p-values > 0.05

significantly better than AdaBoost.MH at a p-value ≤ 0.01 in 17 out of 18 tests. The strength of these results is also witnessed by the fact that these tests have been run on the results obtained with $S = 1{,}000$ boosting iterations, i.e., the scenario in which (see Table 2) the difference between the two systems is smallest.

Finally, note (see 4th and 8th rows of Table 5) that no statistically significant difference is found between the global and "glocal" feature selection policies (however, note that global consistently scores better than glocal on micro-level tests on RCV1-v2), thereby reinforcing the impression that no significant advantage is to be gained by using the latter instead of the former.

5.6 Efficiency

In terms of efficiency, we can observe that training time is +50.4% smaller, irrespectively of the number of iterations, a reduction that confirms the theoretical findings discussed in Sect. 4 (and that might likely be even more substantial in classification problems characterized by a deeper, more articulated hierarchy). Classification time is also generally reduced; aside from an isolated case in which it increases by 16.8%, it is reduced from +5.5% to +67.4%, with higher reductions being obtained for high values of S; this is likely due to the fact that, since high values of S bring about more effective classifiers, the classifiers placed at internal nodes are more effective at "blocking" unsuitable documents from percolating down to leaves which would reject them anyway.

The experiments run after global feature selection qualitatively confirm the results above. Note that the effectiveness values are practically unchanged with respect to the full feature set experiment; this is especially noteworthy for the RCV1-v2 experiments, in which more than 96% of the original features have been discarded with no loss in effectiveness. Effectiveness does not change also when using "glocal" feature selection. This is somehow surprising, since an effectiveness improvement might have been expected here, due to the generation of feature sets customized to each internal node. It is thus likely that the values of v chosen when applying the global policy were large enough to allow the inclusion, for each internal node, of enough features customized to it. We plan to carry out further experiments in order to check whether, at more aggressive levels of reduction (i.e. smaller values of v), the glocal policy will indeed prove superior to the global one.

6 Related work

HTC was first tackled in Wiener et al. (1995), in the context of a TC system based on neural networks and latent semantic indexing. The intuition that it could be useful to perform feature selection locally by exploiting the topology of the tree is originally due to Koller and Sahami (1997). However, this work dealt with 1-of-n text categorization, which means that feature selection was performed "collectively", i.e., relative to the set of children of each internal node; given that we are in an m-of-n classification context, we instead do it "individually", i.e., relative to each child of any internal node. The intuition that the negative training examples for training the classifier for category c_j could be limited to the positive training examples of categories topologically close to c_j is due to Ng et al. (1997) and Wiener et al. (1995). The notion that, in an m-of-n classification context, classifiers at internal nodes act as "routers" informs much of the HTC literature, and is explicitly discussed at least in Ruiz and Srinivasan (2002), which proposes a HTC system based on neural networks.

Other works in hierarchical text categorization have focused on other specific aspects of the learning task. For instance, the "shrinkage" method presented in McCallum et al. (1998) is aimed at improving parameter estimation for data-sparse leaf categories in a 1-of-*n* HTC system based on a naïve Bayesian method; the underlying intuitions are specific to naïve Bayesian methods, and do not easily carry over to other contexts. Incidentally, the naïve Bayesian approach seems to have been the most popular among HTC researchers, since several other HTC models are hierarchical variations of naïve Bayesian learning algorithms (Chakrabarti et al. 1998; Gaussier et al. 2002; Toutanova et al. 2001; Vinokourov and Girolami 2002); SVMs have also recently gained popularity in this respect (Cai and Hofmann 2004; Dumais and Chen 2000; Liu et al. 2005; Yang et al. 2003).

Evaluation measures from flat classification are the most popular choices for evaluating HTC systems. Some other researchers (Ceci and Malerba 2007; Sun and Lim 2001) have proposed that evaluation measures specific to the hierarchical case should be used in HTC, so that credit is given to "partially correct" classification, i.e., to the misclassification of a document into a category topologically close to the correct one. We think that these measures are difficult to judge in the abstract, since whether a user would gain any more benefit from a "partially correct" classification than from a "completely wrong" classification remains open to question, and fundamentally dependent on the particular application. We also believe that such proposals may be interesting, if at all, only in 1-of-*m* classification, in which there is such a notion as *the* correct category to which a document belongs. For the evaluation of our systems we have thus stuck to using "traditional" evaluation measures.

7 Conclusion

We have presented TreeBoost.MH, a recursive algorithm for hierarchical text categorization that uses AdaBoost.MH as its base step and that recurs over the category tree structure. We have given complexity results in which we show that TreeBoost.MH, by leveraging on the hierarchical structure of the category tree, is exponentially cheaper to train and to test than AdaBoost.MH. These theoretical intuitions have been confirmed by thorough empirical testing on three standard benchmarks, on which TreeBoost.MH has brought about substantial savings at both learning time and classification time. TreeBoost.MH has also shown to bring about substantial improvements in effectiveness with respect to AdaBoost.MH, especially in terms of macroaveraged effectiveness; this feature makes it extremely suitable to categorization problems characterized by a skewed distribution of the positive training examples across the categories.

Acknowledgements This work has been partially supported by Project "Networked Peers for Business" (NeP4B), funded by the Italian Ministry of University and Research (MIUR) under the "Fondo per gli Investimenti della Ricerca di Base" (FIRB) funding scheme.

References

Apté, C., Damerau, F. J., & Weiss, S. M. (1994). Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems, 12*(3), 233–251.

Cai, L., & Hofmann, T. (2004). Hierarchical document categorization with support vector machines. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM'04)*, pp. 78–87.

Ceci, M., & Malerba, D. (2007). Classifying Web documents in a hierarchy of categories: A comprehensive study. *Journal of Intelligent Information Systems, 28*(1), 37–78.

Chakrabarti, S., Dom, B. E., Agrawal, R., & Raghavan, P. (1998). Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *Journal of Very Large Data Bases, 7*(3), 163–178.

Cheng, C.-H., Tang, J., Wai-Chee, A., & King, I. (2001). Hierarchical classification of documents with error control. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'01)* (pp. 433–443). Hong Kong, CN.

Dumais, S. T., & Chen, H. (2000). Hierarchical classification of web content. In *Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR'00)* (pp. 256–263). Athens, GR.

Fagni, T., & Sebastiani, F. (2007). On the selection of negative examples for hierarchical text categorization. In *Proceedings of the 3rd Language & Technology Conference (LTC'07)* (pp. 24–28). Poznań, PL.

Forman, G. (2004). A pitfall and solution in multi-class feature selection for text classification. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. Banff, CA.

Gaussier, É., Goutte, C., Popat, K., & Chen, F. (2002). A hierarchical model for clustering and categorising documents. In *Proceedings of the 24th European Colloquium on Information Retrieval Research (ECIR'02)* (pp. 229–247). Glasgow, UK.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning (ICML'97)* (pp. 170–178). Nashville, US.

Lewis, D. D. (1992). *Representation and learning in information retrieval*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, US.

Lewis, D. D., Li, F., Rose, T., & Yang, Y. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research, 5*, 361–397.

Liu, T. Y., Yang, Y., Wan, H., Zeng, H. J., Chen, Z., & Ma, W. Y. (2005). Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations, 7*(1), 36–43.

McCallum, A. K., Rosenfeld, R., Mitchell, T. M., Ng, A. Y. (1998). Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)* (pp. 359–367). Madison, US.

Ng, H. T., Goh, W. B., Low, K. L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the 20th ACM International Conference on Research and Development in Information Retrieval (SIGIR'97)* (pp. 67–73). Philadelphia, US.

Rose, T., Stevenson, M., & Whitehead, M. (2002). The Reuters Corpus Volume 1—from yesterday's news to tomorrow's language resources. In *Proceedings of the 3rd International Conference on Language (LREC'02) Resources and Evaluation* (pp. 827–832). Las Palmas, ES.

Ruiz, M., & Srinivasan, P. (2002). Hierarchical text classification using neural networks. *Information Retrieval, 5*(1), 87–118.

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning, 37*(3), 297–336.

Schapire, R. E., & Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine Learning, 39*(2/3), 135–168.

Schapire, R. E., Singer, Y., & Singhal, A. (1998). Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st ACM International Conference on Research and Development in Information Retrieval (SIGIR'98)* (pp. 215–223). Melbourne, AU.

Sebastiani, F., Sperduti, A., & Valdambrini N. (2000). An improved boosting algorithm and its application to automated text categorization. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM'00)* (pp. 78–85). McLean, US.

Spiegel, M. R., & Stephens, L. J. (1999). *Statistics* (3rd ed.). New York, US: McGraw-Hill.

Sun, A., & Lim, E.-P. (2001). Hierarchical text classification and evaluation. In *Proceedings of the 1st IEEE International Conference on Data Mining (ICDM-01)* (pp. 521–528). San Jose, US.

Toutanova, K., Chen, F., Popat, K., & Hofmann, T. (2001). Text classification in a hierarchical mixture model for small training sets. In *Proceedings of the 10th ACM International Conference on Information and Knowledge Management (CIKM'01)* (pp. 105–113). Atlanta, US.

Vinokourov, A., & Girolami, M. (2002). A probabilistic framework for the hierarchic organisation and classification of document collections. *Journal of Intelligent Information Systems, 18*(2/3), 153–172.

Weigend, A. S., Wiener, E. D., & Pedersen, J. O. (1999). Exploiting hierarchy in text categorization. *Information Retrieval, 1*(3), 193–216.

Wiener, E. D., Pedersen, J. O., & Weigend, A. S. (1995). A neural network approach to topic spotting. In *Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95)* (pp. 317–332). Las Vegas, US.

Yang, Y., & Liu, X. (1999). A re-examination of text categorization methods. In *Proceedings of the 22nd ACM International Conference on Research and Development in Information Retrieval (SIGIR'99)* (pp. 42–49). Berkeley, US.

Yang, Y., Zhang, J., & Kisiel, B. (2003). A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th ACM International Conference on Research and Development in Information Retrieval (SIGIR'03)* (pp. 96–103). Toronto, CA.