TreeBoost.MH: A Boosting Algorithm for Multi-label Hierarchical Text Categorization

Andrea Esuli, Tiziano Fagni, and Fabrizio Sebastiani

Istituto di Scienza e Tecnologia dell'Informazione
Consiglio Nazionale delle Ricerche
Via Giuseppe Moruzzi 1 – 56124 Pisa, Italy
{andrea.esuli, tiziano.fagni, fabrizio.sebastiani}@isti.cnr.it

Abstract. In this paper we propose TreeBoost.MH, an algorithm for multi-label *Hierarchical Text Categorization* (HTC) consisting of a hierarchical variant of AdaBoost.MH. 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". We present the results of experimenting TreeBoost.MH on two HTC benchmarks, and discuss analytically its computational cost.

1 Introduction

Hierarchical text categorization (HTC) is the task of generating text classifiers that operate on classification schemes endowed with a hierarchical structure. Notwithstanding the fact that most large-sized classification schemes for text (e.g. the ACM Classification Scheme[1]) 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. These algorithms, once applied to a hierarchical classification problem, are not capable of taking advantage of the information inherent in the class hierarchy. 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. Many of these heuristics have been used in close association with a specific learning algorithm; the most popular choices in this respect have been naïve Bayesian methods [1, 2, 3, 4, 5, 6], neural networks [7, 8, 9], support vector machines [10, 11], and example-based classifiers [11].

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

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

F. Crestani, P. Ferragina, and M. Sanderson (Eds.): SPIRE 2006, LNCS 4209, pp. 13–24, 2006.

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, a very well-known 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 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 Section 2 we give a concise description of boosting and the AdaBoost.MH algorithm. Section 3 describes TreeBoost.MH. In Section 4 we present experiments comparing AdaBoost.MH and TreeBoost.MH on two well-known HTC benchmarks. Section 5 discusses related work. Section 6 concludes.

2 An Introduction to Boosting and AdaBoost.MH

AdaBoost.MH [12] 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*). The input to the algorithm is a set of pairs $C = \{\langle c_1, Tr^+(c_1)\rangle, \ldots, \langle c_m, Tr^+(c_m)\rangle\}$ each consisting of a category and its set of positive training examples. For each $c_j \in C$, we define the set $Tr^-(c_j)$ of its negative training examples simply as the union of the sets of the positive training examples of the other categories, minus $Tr^+(c_j)$.

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 $\hat{\Phi}_s : D \times C \rightarrow \mathbb{R}$. We interpret the sign of $\hat{\Phi}_s(d_i, c_j)$ 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. We instead interpret the absolute value of $\hat{\Phi}_s(d_i, c_j)$ (indicated by $|\hat{\Phi}_s(d_i, c_j)|$) 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 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 forces 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

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

where the *target function* $\Phi(d_i, c_j)$ is defined to be 1 if $c_j \in C_i$ and -1 otherwise, and $Z_s = \sum_{i=1}^{g} \sum_{j=1}^{m} D_s(d_i, c_j) \exp(-\Phi(d_i, c_j) \cdot \hat{\Phi}_s(d_i, c_j))$ is a normalization factor chosen so that $\sum_{i=1}^{g} \sum_{j=1}^{m} D_{s+1}(d_i, c_j) = 1$, i.e. so that D_{s+1} is in fact a distribution. 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.

2.1 Choosing the Weak Hypotheses

In AdaBoost.MH each document d_i is represented as a vector $\langle w_{1i}, \ldots, w_{ri} \rangle$ of r binary weights, where $w_{ki} = 1$ (resp. $w_{ki} = 0$) means that term t_k occurs (resp. does not occur) in d_i; $T = \{t_1, \ldots, t_r\}$ is the set of terms that occur in at least one document in Tr.

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

$$\hat{\Phi}_s(d_i, c_j) = \begin{cases} a_{0j} & \text{if } w_{ki} = 0 \\ a_{1j} & \text{if } w_{ki} = 1 \end{cases} \tag{2}$$

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 [13] have proven that 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 Z_s.

Schapire and Singer [12] define three different variants of AdaBoost.MH, corresponding to three different methods for making these choices. In this paper we concentrate on one of them, AdaBoost.MH *with real-valued predictions* (hereafter simply called AdaBoost.MH), since it is the one that, in [12], has been experimented most thoroughly and has given the best results:

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 [13] have proven that, given term t_k and category c_j,

$$\hat{\Phi}_{best(k)}(d_i, c_j) = \begin{cases} \frac{1}{2} \ln \frac{W_{+1}^{0jk}}{W_{-1}^{0jk}} & \text{if } w_{ki} = 0 \\ \frac{1}{2} \ln \frac{W_{+1}^{1jk}}{W_{-1}^{1jk}} & \text{if } w_{ki} = 1 \end{cases} \tag{3}$$

where $W_b^{xjk} = \sum_{i=1}^{g} D_s(d_i, c_j) \cdot [\![w_{ki} = x]\!] \cdot [\![\Phi(d_i, c_j) = b]\!]$ 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 function that returns 1 if π is true and 0 otherwise.

3 A Hierarchical Boosting Algorithm 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. TreeBoost.MH is fully illustrated in Figure 1.

Let us first fix some notation and definitions. Let H be a tree-structured set of categories, let r be its root category, and let $L = \langle\langle l_1, Tr^+(l_1)\rangle, \ldots, \langle l_m, Tr^+(l_m)\rangle\rangle$ be the set of leaf categories of H together with their sets of positive training examples. For each category $c_j \in H$, 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

We assume that documents can belong to zero, one, or several leaf categories in L, and that leaf categories are the only categories to which documents can belong (so that categories corresponding to internal nodes are just aggregations of "real" categories). The notion of set of positive/negative training examples is naturally extended to nonleaf categories via the following definition.

Definition 1. *Given a nonleaf category c_j, its set of positive training examples $Tr^+(c_j)$ is defined as $Tr^+(c_j) = \bigcup_{c \in \Downarrow(c_j)} Tr^+(c)$, i.e. as the union of the sets of positive training examples of all its descendant (leaf) categories.*

Input: A triplet $\langle H, r, L\rangle$ where
  H is a tree-structured set of categories,
  r is the root category of H,
  $L = \langle\langle l_1, Tr^+(l_1)\rangle, \ldots, \langle l_m, Tr^+(l_m)\rangle\rangle$ is the (possibly empty) set of leaf categories of H
    together with their sets of positive training examples;
Body: **if** r is a leaf category **then** do nothing
else begin
  let $\downarrow(r) = \{\langle\downarrow_1(r), Tr^+(\downarrow_1(r))\rangle \ldots, \langle\downarrow_{k(r)}(r), Tr^+(\downarrow_{k(r)}(r))\rangle\}$ be the $k(r)$ children categories of r
    together with their sets of positive training examples;
  run a ML feature selection algorithm on $\downarrow(r)$;
  run AdaBoost.MH on $\downarrow(r)$;
  for $q = 1, \ldots, k(r)$ **do**
    begin
      let T_q be the subtree of H rooted at $\downarrow_q(r)$;
      let $L_q = \{\langle l_{q(1)}, Tr^+(l_{q(1)})\rangle, \ldots, \langle l_{q(z)}, Tr^+(l_{q(z)})\rangle\}$ be the (possibly empty)
        set of leaf categories of T_q together with their sets of positive training examples;
      run TreeBoost.MH on $\langle T_q, \downarrow_q(r), L_q\rangle$;
    end
  end
Output: For each nonleaf category $c_t \in H$, 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. 1. The TreeBoost.MH algorithm

Definition 2. *Given a nonleaf category c_j, its set of negative training examples $Tr^-(c_j)$ is defined as $Tr^-(c_j) = \left(\bigcup_{c \in \leftrightarrow(c_j)} Tr^+(c)\right) - Tr^+(c_j)$, i.e. as the union of the sets of positive training examples of all its sibling (leaf or nonleaf) categories, minus its own positive training examples.*

3.1 The Rationale

TreeBoost.MH 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 leaf 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. Each document may thus reach zero, one, or several leaf 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 correctly, $\hat{\Phi}_j$ 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 union of the positive training examples of the categories in $\leftrightarrow(c_j)$ (with the obvious exception of the

documents that are also positive training examples of c_j); in particular, training documents that only belong to leaf categories other than those in $\Downarrow(c_j)$ need not be used. The rationale of this choice is that the chosen documents are "quasi-positive" examples of c_j [14], 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. This intuition lies at the basis of Definition 2 above; in a similar form, it had first been presented in [15].

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 correctly, $\hat{\Phi}_j$ 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 among them, and only among them; 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 a different form, was first presented in [2].

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.

3.2 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 $c_j \in H$, a binary classifier $\hat{\Phi}$ for c_j, by means of which hierarchical classification can be performed as described in Section 3.1.

Learning in TreeBoost.MH proceeds by first identifying whether a leaf category has been reached (line 6 of Figure 1), 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 H) and "local" policy (in which a different set of features is chosen for each different category). The glocal policy selects a different set of features for each maximal set of sibling categories in H, thus implementing a view of feature selection as described in Section 3.1[2]. Any of the standard feature scoring functions (e.g. information gain, chi-square) can be used, as well as any of the standard feature score globalization methods (e.g. max, weighted average, Forman's [16] round robin). Note that all these functions require a precise notion of what the positive and negative training examples of a category are; this notion is well-defined for leaf categories (see beginning of Section 2), and is catered for by Definitions 1 and 2 for internal node categories.

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 $\downarrow(c_j)$; here too, what counts as a positive and as a negative training example of a category comes from Definitions 1 and 2, which implements the "quasi-positive" policy for the choice of negative training examples discussed in Section 3.1. Note that restricting the AdaBoost.MH call to the categories in $\downarrow(c_j)$ implements the view, discussed in Section 3.1, 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.

In an extended version of this paper [17] we discuss the computational cost of TreeBoost.MH, proving that (at least in the idealized case of a "fully grown", perfectly balanced tree of constant ariety a):

- at training time TreeBoost.MH is $O(grah)$, while AdaBoost.MH is $O(grm)$;
- at testing time TreeBoost.MH is $O(Sah)$, while AdaBoost.MH is $O(Sm)$.

Since $m = a^h$, this means that TreeBoost.MH is cheaper than AdaBoost.MH by an exponential factor, both at training time and at testing time.

4 Experiments

4.1 Experimental Setting

The first benchmark we have used in our experiments is the "Reuters-21578, Distribution 1.0" corpus[3]. In origin, the Reuters-21578 category set is not

[2] 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 $\downarrow(c_j)$.

[3] `http://www.daviddlewis.com/resources/testcollections/~reuters21578/`

hierarchically structured, and is thus not suitable "as is" for HTC experiments; we have thus used a hierarchical version of it generated in [5] 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 [5], 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 second benchmark we have used is REUTERS CORPUS VOLUME 1 version 2 (RCV1-v2)[4], consisting of 804,414 news stories. In our experiments we have used the "LYRL2004" split defined in [18], in which the (chronologically) first 23,149 documents are used for training and the other 781,265 are used for testing. Out of the 103 "Topic" categories, in our experiments we have restricted our attention to the 101 categories with at least one positive training example. 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.

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, and stemming has been performed by means of Porter's stemmer. 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 $F_1 = \frac{2\pi\rho}{\pi+\rho} = \frac{2TP}{2TP+FP+FN}$, where TP, FP, and FN stand for the numbers of true positives, false positives, and false negatives, respectively. We compute 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 the $F_1 = \frac{2\pi\rho}{\pi+\rho}$ formula. F_1^M is obtained by first computing the category-specific F_1 values and then averaging them across the c_i's.

4.2 Results

The results of our experiments are reported in Table 1.

In a first experiment we have compared ADABOOST.MH and TREEBOOST.MH using a full feature set. We have then switched to reduced feature sets, obtained according to a "global" feature selection policy in which (i) feature-category pairs have been scored by means of *information gain*, defined as $IG(t_k, c_i) = \sum_{c\in\{c_i,\overline{c}_i\}} \sum_{t\in\{t_k,\overline{t}_k\}} P(t,c) \cdot \log \frac{P(t,c)}{P(t)\cdot P(c)}$ 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_i, the v features with the highest $IG(t_k, c_i)$ value, and pooling all of them together into a category-independent set [16]. This set thus contains a number of features $q \leq vm$, where m is the number of categories; it usually contains strictly fewer than them, since 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

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

Table 1. AdaBoost.MH and TreeBoost.MH on Reuters-21578 (top 5 rows) and RCV1-v2 (bottom 5 rows). In each square, the first figure from top is F_1^{μ}, the second is F_1^{M}, the third is training time (inclusive of the time required to perform feature selection, if any), and the fourth is testing time.

	5 iterations	10 iterations	20 iterations	50 iterations	100 iterations	200 iterations	500 iterations	1000 iterations
AdaBoost.MH (Full)	.533 .033 34.0 11.1	.597 .075 68.1 14.3	.664 .160 136.3 18.2	.724 .255 340.7 35.1	.783 .332 681.5 66.2	.798 .361 1362.9 129.9	.804 .377 3407.3 274.0	.808 .379 6814.6 464.3
TreeBoost.MH (Full)	.596 (+11.9%) .100 (+197.4%) 16.9 (-50.4%) 13.0 (+16.8%)	.699 (+17.2%) .187 (+148.4%) 33.8 (-50.4%) 12.6 (-11.8%)	.745 (+12.2%) .286 (+78.9%) 67.6 (-50.4%) 17.2 (-5.5%)	.795 (+9.9%) .416 (+62.6%) 169.0 (-50.4%) 20.6 (-41.4%)	.810 (+3.5%) .425 (+28.2%) 337.9 (-50.4%) 29.9 (-54.9%)	.827 (+3.7%) .454 (+25.6%) 675.8 (-50.4%) 48.5 (-62.7%)	.830 (+3.1%) .460 (+22.0%) 1689.4 (-50.4%) 96.6 (-64.7%)	.826 (+2.3%) .479 (+26.4%) 3378.9 (-50.4%) 151.3 (-67.4%)
AdaBoost.MH (Global)	.533 .034 24.6 8.8	.597 .075 49.2 12.4	.664 .160 98.4 17.0	.724 .256 246.1 32.8	.783 .332 492.1 59.6	.799 .354 984.3 112.2	.811 .373 2460.8 255.2	.801 .362 4921.5 386.0
TreeBoost.MH (Global)	.596 (+11.9%) .100 (+197.4%) 11.5 (-53.4%) 9.1 (+3.2%)	.699 (+17.2%) .187 (+148.4%) 23.0 (-53.4%) 9.6 (-22.1%)	.744 (+11.9%) .285 (+78%) 45.9 (-53.4%) 11.3 (-33.5%)	.800 (+10.5%) .437 (+70.8%) 114.7 (-53.4%) 17.2 (-47.7%)	.809 (+3.4%) .427 (+28.7%) 229.5 (-53.4%) 26.3 (-55.9%)	.815 (+2.0%) .457 (+28.8%) 459.0 (-53.4%) 42.4 (-62.2%)	.828 (+2.1%) .457 (+22.4%) 1147.4 (-53.4%) 82.3 (-67.7%)	.821 (+2.5%) .473 (+30.6%) 2294.7 (-53.4%) 131.3 (-66.0%)
TreeBoost.MH (Glocal)	.596 (+11.9%) .100 (+197.4%) 12.1 (-50.7%) 10.7 (+21.6%)	.699 (+17.2%) .187 (+148.4%) 24.2 (-50.7%) 14.9 (+20.9%)	.744 (+11.9%) .285 (+77.9%) 48.5 (-50.7%) 14.1 (-16.7%)	.794 (+9.7%) .401 (+56.9%) 121.2 (-50.7%) 23.0 (-29.8%)	.812 (+3.8%) .430 (+29.8%) 242.5 (-50.7%) 28.3 (-52.4%)	.817 (+2.2%) .460 (+29.8%) 485.0 (-50.7%) 46.2 (-58.9%)	.824 (+1.6%) .465 (+24.7%) 1212.4 (-50.7%) 94.6 (-62.9%)	.825 (+3.0%) .465 (+28.3%) 2424.8 (-50.7%) 142.9 (-63.0%)
AdaBoost.MH (Full)	.361 .037 181.3 3346.2	.406 .070 362.7 3576.4	.479 .131 725.3 5168.2	.587 .239 1813.3 9524.7	.650 .333 3626.5 16827.2	.701 .396 7253.1 33608.9	.735 .435 18132.7 83951.2	.745 .442 36265.5 170720.3
TreeBoost.MH (Full)	.391 (+8.4%) .097 (+159.0%) 78.3 (-56.8%) 2774.5 (-17.1%)	.460 (+13.3%) .128 (+81.5%) 156.5 (-56.8%) 2813.5 (-21.3%)	.543 (+13.5%) .211 (+60.8%) 313.1 (-56.8%) 3081.1 (-40.4%)	.658 (+12.1%) .341 (+42.8%) 782.7 (-56.8%) 3963.2 (-58.4%)	.705 (+8.4%) .409 (+22.7%) 1565.3 (-56.8%) 6044.2 (-64.1%)	.734 (+4.6%) .447 (+12.9%) 3130.6 (-56.8%) 9328.3 (-72.2%)	.752 (+2.3%) .476 (+9.4%) 7826.6 (-56.8%) 21847.4 (-74.0%)	.761 (+2.1%) .486 (+9.8%) 15653.1 (-56.8%) 38342.9 (-77.5%)
AdaBoost.MH (Global)	.361 .037 107.8 1598.4	.406 .070 215.5 2223.7	.479 .131 431.1 3741.0	.587 .239 1077.7 8012.8	.650 .332 2155.5 16206.9	.700 .398 4310.9 31907.7	.736 .443 10777.3 74292.3	.749 .457 21554.7 147354.1
TreeBoost.MH (Global)	.391 (+8.4%) .097 (+159.0%) 33.8 (-68.7%) 1830.2 (+14.5%)	.460 (+13.3%) .128 (+81.6%) 67.5 (-68.7%) 1422.0 (-36.1%)	.545 (+13.8%) .213 (+62.6%) 135.1 (-68.7%) 1901.0 (-49.2%)	.657 (+12.0%) .340 (+42.4%) 337.6 (-68.7%) 2772.1 (-65.4%)	.702 (+8.0%) .409 (+23.4%) 675.3 (-68.7%) 4836.0 (-70.2%)	.732 (+4.6%) .448 (+12.5%) 1350.5 (-68.7%) 8156.6 (-74.4%)	.753 (+2.3%) .484 (+9.4%) 3376.4 (-68.7%) 18384.5 (-75.3%)	.760 (+1.4%) .495 (+8.3%) 6752.8 (-68.7%) 33648.3 (-77.2%)
TreeBoost.MH (Glocal)	.391 (+8.4%) .097 (+159.0%) 41.3 (-61.7%) 2374.9 (+48.6%)	.460 (+13.3%) .128 (+81.5%) 82.6 (-61.7%) 2432.7 (+9.4%)	.543 (+13.5%) .211 (+61.1%) 165.3 (-61.7%) 2499.9 (-33.2%)	.658 (+12.1%) .340 (+42.6%) 413.1 (-61.7%) 3645.9 (-54.5%)	.703 (+8.1%) .408 (+23.0%) 826.3 (-61.7%) 5020.3 (-69.0%)	.735 (+5.1%) .450 (+12.9%) 1652.6 (-61.7%) 8372.9 (-73.8%)	.753 (+2.3%) .476 (+7.6%) 4131.4 (-61.7%) 18173.9 (-75.5%)	.762 (+1.7%) .490 (+7.2%) 8268.9 (-61.7%) 33149.0 (-77.5%)

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 Section 3.2, consisting in selecting a different feature subset (of the same cardinalities as in the global policy) for the set of children of each different internal node. Note that the results obtained by means of this policy are reported only for TreeBoost.MH, since this policy obviously is not applicable to AdaBoost.MH.

We will now comment on the Reuters-21578 results[5]; the RCV1-v2 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

[5] 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 [12] for the same algorithm ($F_1^{\mu} = .851$). There are several reasons for this: (a) [12] actually uses a different, much older version of this collection, called Reuters-21450 [19]; (b) [12] 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.

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 anymore [2, 4]. 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.

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 Section 3.2 (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 wrt 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.

5 Related Work

HTC was first tackled in [9], in the context of a TC system based on neural networks. The intuition that it could be useful to perform feature selection locally by exploiting the topology of the tree is originally due to [2]. However, this work dealt with 1-of-n text categorization, which means that feature selection was performed relative to the set of children of each internal node; given that we are in a m-of-n classification context, we instead do it relative to each individual 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 close to c_j in the tree is due to [15]. The notion that, in a m-of-n classification context, classifiers at internal nodes act as "filters" informs much of the HTC literature, and is explicitly discussed at least in [7], which proposes a HTC system based on neural networks.

Other works in HTC focus on other specific aspects of the learning task. For instance, the "shrinkage" method presented in [4] attempts to improve parameter estimation for data-sparse leaf categories in a 1-of-n HTC system based on a naïve Bayesian method. 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 [1, 3, 5, 6].

6 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 two 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 wrt 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.

References

1. Chakrabarti, S., Dom, B.E., Agrawal, R., Raghavan, P.: Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. Journal of Very Large Data Bases **7**(3) (1998) 163–178
2. Koller, D., Sahami, M.: Hierarchically classifying documents using very few words. In: Proceedings of the 14th International Conference on Machine Learning (ICML'97), Nashville, US (1997) 170–178
3. Gaussier, É., Goutte, C., Popat, K., Chen, F.: A hierarchical model for clustering and categorising documents. In: Proceedings of the 24th European Colloquium on Information Retrieval Research (ECIR'02), Glasgow, UK (2002) 229–247
4. McCallum, A.K., Rosenfeld, R., Mitchell, T.M., Ng, A.Y.: Improving text classification by shrinkage in a hierarchy of classes. In: Proceedings of the 15th International Conference on Machine Learning (ICML'98), Madison, US (1998) 359–367
5. Toutanova, K., Chen, F., Popat, K., Hofmann, T.: 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), Atlanta, US (2001) 105–113
6. Vinokourov, A., Girolami, M.: A probabilistic framework for the hierarchic organisation and classification of document collections. Journal of Intelligent Information Systems **18**(2/3) (2002) 153–172
7. Ruiz, M., Srinivasan, P.: Hierarchical text classification using neural networks. Information Retrieval **5**(1) (2002) 87–118
8. Weigend, A.S., Wiener, E.D., Pedersen, J.O.: Exploiting hierarchy in text categorization. Information Retrieval **1**(3) (1999) 193–216

9. Wiener, E.D., Pedersen, J.O., Weigend, A.S.: A neural network approach to topic spotting. In: Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95), Las Vegas, US (1995) 317–332
10. Dumais, S.T., Chen, H.: Hierarchical classification of web content. In: Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR'00), Athens, GR (2000) 256–263
11. Yang, Y., Zhang, J., Kisiel, B.: 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), Toronto, CA (2003) 96–103
12. Schapire, R.E., Singer, Y.: BoosTexter: a boosting-based system for text categorization. Machine Learning **39**(2/3) (2000) 135–168
13. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine Learning **37**(3) (1999) 297–336
14. Schapire, R.E., Singer, Y., Singhal, A.: Boosting and Rocchio applied to text filtering. In: Proceedings of the 21st ACM International Conference on Research and Development in Information Retrieval (SIGIR'98), Melbourne, AU (1998) 215–223
15. Ng, H.T., Goh, W.B., Low, K.L.: 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), Philadelphia, US (1997) 67–73
16. Forman, G.: 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 (2004)
17. Esuli, A., Fagni, T., Sebastiani, F.: TreeBoost.MH: A boosting algorithm for multi-label hierarchical text categorization. Technical Report 2006-TR-56, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, IT (2006) Submitted for publication.
18. Lewis, D.D., Li, F., Rose, T., Yang, Y.: RCV1: A new benchmark collection for text categorization research. Journal of Machine Learning Research **5** (2004) 361–397
19. Apté, C., Damerau, F.J., Weiss, S.M.: Automated learning of decision rules for text categorization. ACM Transactions on Information Systems **12**(3) (1994) 233–251