Selecting Negative Examples for Hierarchical Text Classification: An Experimental Comparison

Tiziano Fagni and Fabrizio Sebastiani
Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Via Giuseppe Moruzzi 1–56124 Pisa, Italy. E-mail: {tiziano.fagni, fabrizio.sebastiani}@isti.cnr.it

Hierarchical text classification (HTC) approaches have recently attracted a lot of interest on the part of researchers in human language technology and machine learning, since they have been shown to bring about equal, if not better, classification accuracy with respect to their "flat" counterparts while allowing exponential time savings at both learning and classification time. A typical component of HTC methods is a "local" policy for selecting negative examples: Given a category *c*, its negative training examples are by default identified with the training examples that are negative for *c* and positive for the categories which are siblings of *c* in the hierarchy. However, this policy has always been taken for granted and never been subjected to careful scrutiny since first proposed 15 years ago. This article proposes a thorough experimental comparison between this policy and three other policies for the selection of negative examples in HTC contexts, one of which (BESTLOCAL(*k*)) is being proposed for the first time in this article. We compare these policies on the hierarchical versions of three supervised learning algorithms (boosting, support vector machines, and naïve Bayes) by performing experiments on two standard TC datasets, REUTERS-21578 and RCV1-v2.

Introduction

Given a set of textual documents D and a predefined set of *categories* (aka *labels* or *classes*) $C = \{c_1, \ldots, c_m\}$, *multi-label* (aka n-of-m) *text classification* (TC) is the task of approximating, or estimating, an unknown *target function* $\Phi: D \times C \rightarrow \{-1, +1\}$ that describes how documents ought to be classified, by means of a function $\hat{\Phi}: D \times C \rightarrow \{-1, +1\}$, called the "*classifier.*"[1] Here, "multi-label" indicates that the same document can belong to zero, one, or several categories at the same time.

Received May 14, 2010; revised June 22, 2010; accepted June 28, 2010

[1]Consistent with most mathematical literature, we use the caret symbol (ˆ) to indicate estimation.

 DOI: 10.1002/asi.21411

Hierarchical text classification (HTC) refers to a variant of the TC task, namely, that in which the set C of the categories is organized into a hierarchy; this may be a tree or, more generally, a directed acyclic graph (DAG). HTC approaches have recently attracted a lot of interest on the part of researchers in human language technology and machine learning (see e.g. Gauch, Ranganathan and Chandramouli, 2009; Liu, 2009; Yoon, Lee, & Lee, 2006), since they have been shown to bring about equal, if not better, classification accuracy with respect to their "flat" counterparts while allowing exponential time savings at both learning and classification time (Esuli, Fagni, & Sebastiani, 2008; Liu et al., 2005).

Multi-label HTC is usually implemented by generating a binary classifier for each nonroot node in the hierarchy (be it an internal or a leaf node); the role of this classifier is to decide whether the test document belongs or not to the category associated with the node. Classification is then performed in "Pachinko machine" style (Koller & Sahami, 1997): The test document is first submitted to the classifiers corresponding to the top-level nodes, and recursively percolates down to (i.e. is submitted to the classifiers corresponding to the nodes in) the lower levels of the hierarchy only if the classifiers at the higher levels have deemed that the document belong to their associated category. In this way, entire subtrees are pruned from consideration, which allows exponential savings at classification time (Chakrabarti, Dom, Agrawal, & Raghavan, 1998; Koller & Sahami, 1997). This is fundamental when tackling classification tasks characterized by very high numbers of categories, as is the case e.g. of the OHSUMED dataset (Hersh, Buckley, Leone, & Hickman, 1994), the WIPO-ALPHA dataset (Fall, Törcsvári, Benzineb, & Karetka, 2003), and the YAHOO dataset (Liu et al., 2005), which all contain tens of thousands of categories.

Exponential savings can also be accomplished at learning time. One way of achieving this is performing feature selection "locally" (Koller & Sahami, 1997), i.e. selecting, for a classifier corresponding to category c, only the features that are most useful in discriminating among c and the categories that are its siblings in the hierarchy; in this way the

vector space in which the documents are represented can be much smaller, thus bringing about speedier learning (and also classification).

A second way of speeding up learning in HTC is adopting a "local" policy for selecting negative examples: Given a category c, its negative training examples are identified with the training examples that, aside from being negative for c, are also positive for the categories which are siblings of to c in the hierarchy. With respect to a policy in which all examples negative for c are used, this results in faster learning, since it reduces (sometimes dramatically) the overall number of training examples considered, and since practically all learning methods are at least linear in such number.

However, this policy (hereafter called the SIBLINGS policy) has always been taken for granted, and never been subjected to careful scrutiny since first being used by Wiener, Pedersen, and Weigend (1995) and explicitly advocated by Ng, Goh, and Low (1997).

This article proposes a thorough experimental comparison between this policy and three other policies for the selection of negative examples in HTC, one of which (BESTLOCAL(k)) is being proposed for the first time in this article. We provide an intuitive basis for these policies and test them on the hierarchical versions of three important supervised learning algorithms (namely: boosting, support vector machines, and naïve Bayes) by performing experiments on two standard TC datasets: (a hierarchical version of) a small dataset consisting of approximately 11,000 documents (REUTERS-21578) and a very large dataset of more than 800,000 documents (RCV1-v2).

We stress that, while the BESTLOCAL(k) policy is proposed here for the first time, this should not be understood as an essential contribution of this article: The main goal of our work was instead to clarify the relative merits of policies which, to date, had never been compared experimentally in a systematic way.

The article is organized as follows: In the second section we outline the basic scheme for learning hierarchical text classifiers that, in our experiments, we will instantiate with boosting, support vector machines (SVMs) and naïve Bayes as base learners. The third section describes in detail the four policies for the selection of negative training examples. The fourth section describes the experimental settings that we have adopted in our experiments. The results of these experiments are reported in the fifth section and further analyzed in the next section while the final section concludes.

A Pattern for Multi-Label HTC

In this section, we describe the basic pattern to which we will conform in building a hierarchical classifier as a hierarchy of standard binary classifiers. Let us first define our notation. Let $H = \langle I, L\rangle$ be a tree-structured set of categories, where $I = \{\langle i_1, Tr^+(i_1)\rangle, \ldots, \langle i_n, Tr^+(i_n)\rangle\}$, and $L = \{\langle l_1, Tr^+(l_1)\rangle, \ldots, \langle l_m, Tr^+(l_m)\rangle\}$ are the sets of categories of H corresponding to the internal nodes (hereafter: *internal categories*) and the leaf nodes (*leaf categories*) of H, respectively, together with their sets of positive training examples, and where $r \in I$ is the root category of H.[2] For each category $c_j \in H$, we will use the following abbreviations:

Symbol	Meaning
$Tr^+(c_j)$	The set of positive training documents of c_j
$Tr^-(c_j)$	The set of negative training documents of c_j
$\uparrow(c_j)$	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 categories which are siblings of c_j

We here assume that documents can belong to zero, one, or several leaf categories in L. Further, we assume that the set of positive examples of an internal category i_j is always given by the union of the positive examples of its descendant leaf categories; in other words, an internal category can contain no documents that do not belong to at least one of its descendant leaf categories. This is a common constraint in many HTC applications, but the assumption is not restrictive anyway.[3] When it comes to training examples, it thus follows that

$$Tr^+(c_j) = \bigcup_{l \in \Downarrow(c_j)} Tr^+(l) \quad (1)$$

We assume that all training examples belong to at least one leaf category $l_j \in L$; the training set Tr thus coincides with $\cup_{l_j \in L} Tr^+(l_j)$.

Figure 1 describes the basic scheme (called TREELEARNER) to which we conform in building a hierarchical classifier. A base learner that generates binary classifiers is passed as a parameter to TREELEARNER; in the section Learning Methods, Weighting Method, and Evaluation Measures, we will alternatively instantiate this general scheme via a boosting-based learner, an SVM-based learner, and a naïve Bayes learner, thus generating the TREEBOOST, TREESVM, and TREENB hierarchical learners. Also the policy for the selection of negative examples is passed as a parameter to TREELEARNER; this will allow us to compare experimentally the four different policies mentioned in the Introduction.

[2]Throughout this article we will always refer to tree-shaped hierarchies; however, all our arguments straightforwardly apply to DAG-shaped hierarchies (see also Footnote 4).

[3]In fact, without loss of generality, given a hierarchically structured set of categories H in which internal categories can indeed contain documents that do not belong to any of their descendant leaf categories, we can transform H into an "extended" set of categories H' by appending to every internal node i_j of H an additional child (leaf) node l_j, and by moving into l_j all documents originally contained in i_j. This mapping, originally proposed by Cheng, Tang, Wai-Chee, and King (2001) produces a hierarchy H' semantically equivalent to H in which all documents are indeed contained in leaf categories only. 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") that contains all documents belonging to the node but to none of its descendant leaves.

DOI: 10.1002/asi

```
procedure TreeLearner(H, r, np, learner)
begin
  if not (r is a leaf category) then
    foreach child in ↓(r) do
    begin
      Tr⁻(child) ← GetNegatives(H, r, child, np);
      Train(child, learner);
      TreeLearner(H, child, np, learner);
    end
  else
    do nothing
  endif
end
```

FIG. 1. The TreeLearner scheme; *H*, *r*, *np*, and *learner* indicate the hierarchy, its root, the chosen policy for the selection of negative training examples, and the chosen learner, respectively.

The scheme is defined as a recursive procedure that, for each nonroot (internal or leaf) category c_j, generates a binary classifier from $Tr^+(c_j)$ and the chosen $Tr^-(c_j)$.

Choosing Negative Examples in HTC

In this work, we present experimental comparisons of four different strategies for selecting negative training examples for a given category. In the following, we give a description of the strategies used and we try to explain the key ideas behind each policy. In the section Efficiency, we will then give details about the computational cost of each such strategy by describing the cost of selecting negative documents for each category and by analyzing the impact that the number of selected examples has on the learning phase.

The Siblings Policy

According to the Siblings policy, the set of negative training documents for category c_j is chosen among the training documents that are not positive for c_j *and* may be assumed to be most correlated to c_j *on topological grounds alone*. That is, it is composed of all the training documents that are not positive for c_j and are positive for the categories sibling of c_j: i.e.[4]

$$Tr^-(c_j) = \left(\bigcup_{c \in \leftrightarrow(c_j)} Tr^+(c) \right) \setminus Tr^+(c_j) \qquad (2)$$

There are two main intuitions behind this policy.

The first intuition is that, *if the classifier associated with* $\uparrow(c_j)$ *has generated no false positives*, the classifier associated with c_j will only be asked to classify documents that belong to c_j and/or one or more among its siblings. If this is the case, it is clear that including in $Tr^-(c_j)$ documents that are neither positive for c_j nor for any of its siblings would distract the classifier from focusing on the only distinction that matters in this context, i.e. that between c_j and its siblings.

The second intuition is that this is the policy that most closely conforms to the *divide et impera* view of HTC at the base of the TreeLearner scheme, in which the multilabel problem of classifying documents into a hierarchy $H = <I, L>$ is decomposed into several flat classification problems, one for each $i_j \in I$, in which the set of categories concerned is $\downarrow(i_j)$.

It can be a bit controversial to say who proposed the Siblings policy first. The first authors to explicitly mention it in contrast with the All policy (and to claim "intellectual ownership" of it) are Ng, Goh, and Low (1997). However, a careful reading of Wiener, Pedersen, & Weigend (1995) reveals that Siblings was already used in this article, even if discussed only between the lines and in somewhat unclear terms. After being proposed in these two articles, the Siblings policy was subsequently adopted in, e.g. Chiang and Chen (2001), Dumais and Chen (2000); Esuli, Fagni, and Sebastiani (2008), Liu et al. (2005), Ruiz and Srinivasan (2002), Sun, Lim, and Ng (2003), Weigend, Wiener, and Pedersen (1999), and quickly became the standard choice for HTC contexts.

However, it is important to note that Wiener, Pedersen, and Weigend (1995) nor Ng, Goh, and Low (1997) nor the researchers who have followed in their footsteps have submitted this policy to careful experimental scrutiny, i.e. tested Siblings against All (or against other policies, for that matter) in any comparative experiment.

The All Policy

According to the All policy, the set $Tr^-(c_j)$ of negative training documents for category c_j is simply the entire training set minus the positive training documents of c_j, i.e.

$$Tr^-(c_j) = Tr \setminus Tr^+(c_j) \qquad (3)$$

In a sense, All is a "brute force" policy that disregards the hierarchical structure of the set of categories, treating the HTC problem as a flat classification problem in which no particular selection criterion is used. However, All is still a frequently used policy whenever the HTC classification problem is *not* decomposed into recursively smaller flat classification problems (as in, e.g. Kiritchenko, Matwin, Nock, & Famili, 2006).

Again, there are two main intuitions behind the All policy.

The first intuition is that it is generally the case that the classifier associated with $\uparrow(c_j)$ may indeed generate some false positives, i.e. documents that belong neither to c_j nor to any of its siblings. In this case, if the classifier for c_j had been trained (according to the Siblings policy) only with training examples belonging to $\uparrow(c_j)$, it might be unequipped to correctly recognize (i.e. reject) documents that are very different from the ones it has been trained on.

[4]Note that, if the hierarchy is tree-shaped, each category c_j has a single set of siblings, but if the hierarchy is DAG-shaped c_j has in general several such sets, since it has several parent categories. In the DAG case, the set of categories sibling of c_j to be considered in the Siblings policy is simply the union of the various sets of siblings of c_j as derived from the multiple parents of c_j. Note that the three other policies we discuss in this article do not use the hierarchical structure of the category set; hence, they work for tree- and DAG-shaped hierarchies alike.

DOI: 10.1002/asi

The second intuition is, somehow crudely, that "the more the training data, the better", i.e. that using additional (albeit negative) training examples may only bring about equally or more accurate classifiers, provided efficiency is not an issue.

The BestGlobal Policy

The third policy we discuss, dubbed BestGlobal, has similarities to Siblings in that it tries to substantially limit the size of $Tr^-(c_j)$, and has similarities to All in that it disregards the hierarchical structure of the category set, thus basing the selection process on nontopological considerations. While it has never been used to date in a hierarchical context, BestGlobal simply coincides with the "query zoning" selection strategy, proposed by Singhal, Mitra, and Buckley (1997) for flat classification and subsequently used in Schapire, Singer, and Singhal (1998).

In order to implement BestGlobal one first computes the centroid of $Tr^+(c_j)$, i.e. the document $\zeta(c_j)$ whose vectorial representation is obtained as[5]

$$\zeta(c_j) = \frac{1}{|Tr^+(c_j)|} \sum_{d_p \in Tr^+(c_j)} d_p \tag{4}$$

The $Tr^-(c_j)$ set is then defined as the set of the β_j documents in $Tr \backslash Tr^+(c_j)$ that minimize the distance from this centroid, according to some measure δ of vector distance; i.e.

$$Tr^-(c_j) = \arg \overset{\beta_j}{\min}_{d_n \in Tr \backslash Tr^+(c_j)} \delta(\zeta(c_j), d_n) \tag{5}$$

where $\arg \overset{z}{\min}_{A} f$ denotes the z elements of A that minimize function f.

The rationale behind this policy is that the documents thus selected may be viewed as "near-positives" for c_j, i.e. documents that tend to lie *just* outside the region where the positive examples lie. As such, they tend to be the most informative negative training documents, since they allow a learner to fine-tune the choice of a classifier, i.e. of a surface that separates the above region from that of the negative examples. In this, the notion of a "near-positive training example" is akin to the notion of support vector in kernel machines.

Note that also the Siblings policy may be viewed as a policy for the selection of near-positives. The difference with BestGlobal is that Siblings makes this choice based on topological considerations alone, i.e. by making the assumption that the negative examples of c_j that are most similar to the positive examples of c_j are likely to be the positive examples of c_j's siblings. BestGlobal instead equates similarity with closeness in the vector space in which the documents are represented. Siblings is thus a policy specific to a hierarchical setting, whereas BestGlobal is not.

The BestLocal(k) Policy

We here propose a fourth selection policy (dubbed BestLocal(k)) that essentially consists in a variant of BestGlobal aimed at improving the selection of negative training examples for categories that are not linearly separable.

The disadvantage of the BestGlobal policy is that the centroid of $Tr^+(c_j)$ may be too coarse a representation of the region of the positive examples of c_j. If c_j is linearly separable, the centroid is an optimal such representation; if c_j is not (i.e. if the separating surface in the vector space has a complex geometrical form), the BestGlobal policy will select some negative examples that are in fact far away from the separating surface, and will miss some negative examples that are instead close to it.

A solution to this problem might be that of selecting the β_j negative training examples whose distance from *any element of* $Tr^+(c_j)$ is minimum. In other words, if we define the *closest c_j-positive training neighbor* of document d_n to be

$$\chi(d_n) = \arg \min_{d_p \in Tr^+(c_j)} \delta(d_n, d_p) \tag{6}$$

our policy selects the β_j negative training documents d_n that are closest to any positive training example of c_j, i.e.

$$\arg \overset{\beta_j}{\min}_{d_n \in Tr \backslash Tr^+(c_j)} \delta(d_n, \chi(d_n)) \tag{7}$$

We call this policy BestLocal(1). This policy avoids selecting examples that, while close to the centroid of $Tr^+(c_j)$, are too far from the separating surface, *and* avoids missing examples that, while far from the centroid of $Tr^+(c_j)$, are very close to the separating surface.

A generalization of this policy is obtained by selecting the β_j negative training examples d_n for which the sum of the distances from d_n and its closest k elements of $Tr^+(c_j)$ is minimum. In other words, if we define the *k closest c_j-positive training neighbors* of document d_n to be

$$\chi^k(d_n) = \arg \overset{k}{\min}_{d_p \in Tr^+(c_j)} \delta(d_n, d_p) \tag{8}$$

our policy selects the β_j negative training documents d_n for whom the sum of the distances between d_n and each of the $\chi^k(d_n)$ is minimum, i.e.

$$\arg \overset{\beta_j}{\min}_{d_n \in Tr \backslash Tr^+(c_j)} \sum_{d_p \in \chi^k(d_n)} \delta(d_p, d_n) \tag{9}$$

We call this policy BestLocal(k). This policy trades the specificity (i.e. the ability to individuate documents extremely close to the separating surface) of BestLocal(1) for the robustness (i.e. the ability to avoid outliers) of BestGlobal, and may be seen as an attempt to "smooth" BestLocal(1) by

[5] In order to simplify the notation, in this article we will indicate by the same symbol d_i both a document and its vectorial representation; the intended meaning will be clear from the context.

DOI: 10.1002/asi

TABLE 1. REUTERS-21578 macro-categories and their member categories from Toutanova, Chen, Popat, and Hofmann (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

insisting that, in order to be selected, a negative example must be close not to just one, but to several elements of $Tr^{+}(c_j)$.

Similar to what happens for the BESTGLOBAL policy, also the negative examples selected by BESTLOCAL(k) allow a learner to fine-tune the choice of a surface that separates the positive region from the negative region, and in this case too these examples play a role akin to the support vectors of kernel machines. In this case, the k parameter is used to trade the fit of the model for its simplicity, i.e. its generalization capability: Lower numbers of k bring about complex separating surfaces that may tend to overfit the training data, whereas higher values of k bring about simpler separating surfaces, i.e. surfaces that fit the training data less accurately but tend to be more robust.

Experiments

The Datasets

The first dataset we have used in our experiments is the "REUTERS-21578, Distribution 1.0" corpus, one of the most widely used datasets in TC research.[6] In origin, the REUTERS-21578 category set is not hierarchically structured, and is thus not suitable "as is" for HTC experiments; we have thus used a hierarchical version of it generated by Toutanova, Chen, Popat, and Hofmann (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" (i.e. internal nodes—see Table 1) whose parent category is the root of the tree. Conforming to the experiments of Toutanova, Chen, Popat, and Hofmann (2001), we have used (according to the ModApte split) the 7,770 training examples and 3,299 test examples that are labeled by at least one of the selected leaf categories; the average number of leaf categories per document is 1.23, ranging from a minimum of 1 to a maximum of 15. The average number of positive training examples per leaf category is 106.5, ranging from a minimum of 1 to a maximum of 2,877.

The second dataset we have used is REUTERS CORPUS VOLUME 1 version 2 (RCV1-v2),[7] a more recent TC dataset made available by Reuters and consisting of 804,414 news stories produced by Reuters from Aug 20, 1996 to Aug 19, 1997; all news stories are in English, and have 109 distinct terms per document on average (Rose, Stevenson, & Whitehead, 2002). In our experiments, we have used the "LYRL2004" split defined in Lewis, Yang, Rose, and Li (2004), in which the (chronologically) first 23,149 documents are used for training and the other 781,265 are used for testing. Out of the 82 "Topic" leaf categories, in our experiments we have restricted our attention to the 80 such 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 levels 2 and 3. The average number of leaf categories per document is 3.18, ranging from a minimum of 1 to a maximum of 14. The average number of positive training examples per leaf category is 347.2, ranging from a minimum of 1 to a maximum of 10,786.

Learning Methods, Weighting Method, and Evaluation Measures

As the base learner for the TREELEARNER procedure, we have decided to use a boosting-based learner, an SVM-based learner, and a naïve Bayes learner.

The first learning algorithm we have used is our own implementation of MP-Boost (Esuli, Fagni, & Sebastiani, 2006), a learner based on boosting-by-reweighting technology. In all the experiments, the algorithm has been run with a number of iterations fixed to 1,000. In the remainder of the article, this configuration will be referred to as TREEBOOST.[8]

The second learner is the implementation[9] of SVMs available in the `svm_light` package (Joachims, 1999, Chapter 11), which we have run with a linear kernel and the parameters set at their default values. In the experiments, this configuration will be referred to as TREESVM.

The third learner we have used is our own implementation of a multinomial version of naïve Bayes (McCallum & Nigam, 1998), in which we use the Laplace smoothing in order to smooth parameter estimates obtained via maximum likelihood estimation. In the experiments, this configuration will be referred to as TREENB.

In all the experiments discussed in this section, punctuation has been removed, all letters have been converted to

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

[7] Freely available from http://trec.nist.gov/data/reuters/~reuters.html

[8] Note that TREEBOOST should not be confused with the TREEBOOST.MH algorithm we have presented in (Esuli, Fagni, and Sebastiani, 2008); in fact, while TREEBOOST uses MP-Boost as base learner, TREEBOOST.MH uses ADABOOST.MH (Schapire & Singer, 2000).

[9] Freely downloadable from http://svmlight.joachims.org/

DOI: 10.1002/asi

lowercase, numbers have been removed, stop words have been removed using the stop list provided in Lewis (1992), and stemming has been performed by means of Porter's stemmer. All remaining terms that occur at least once in *Tr* have thus been used as dimensions of our vectorial representations of documents. No feature selection has been performed.

The vectors provided as input to TREESVM have been obtained via the "ltc" variant (Salton & Buckley, 1988) of the well-known *tfidf* class of weighting functions, i.e.

$$tfidf(t_k, d_i) = tf(t_k, d_i) \cdot \log \frac{|Tr|}{\#_{Tr}(t_k)} \tag{10}$$

where $\#_{Tr}(t_k)$ denotes the number of documents in Tr in which t_k occurs at least once and

$$tf(t_k, d_i) = \begin{cases} 1 + \log \#(t_k, d_i) & \text{if } \#(t_k, d_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where $\#(t_k, d_i)$ denotes the number of times t_k occurs in d_i. Weights obtained by Equation (10) are normalized through cosine normalization, i.e.

$$w_{ki} = \frac{tfidf(t_k, d_i)}{\sqrt{\sum_{s=1}^{|\mathcal{T}|} tfidf(t_s, d_i)^2}} \tag{12}$$

The vectors provided as input to TREEBOOST and TREENB are instead binary; this is a constraint imposed by the use of MP-Boost and naïve Bayes, respectively, as base learners.

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

$$F_1 = \frac{2\pi\rho}{\pi + \rho} = \frac{2TP}{2TP + FP + FN} \tag{13}$$

which corresponds to the harmonic mean of precision and recall; here 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 we take F_1 equal to 1, since the classifier has correctly classified all documents as negative examples.

In order to obtain F_1 figures global to an entire dataset, i.e. that cut across the entire set L of leaf categories in the dataset, we compute both *microaveraged* F_1 (denoted by F_1^{μ}) and *macroaveraged* F_1 (F_1^{M}). F_1^{μ} is obtained by (1) computing the category-specific values TP_j for all leaf categories $l_j \in L$, (2) obtaining TP as the sum of all the TP_j's (same for FP and FN), and then (3) applying Equation (13). F_1^{M} is obtained by first computing the F_1 values specific to the individual leaf categories, and then averaging them across all $l_j \in L$. Microaveraged and macroaveraged notions of precision and recall (indicated by π^{μ}, ρ^{μ}, π^{M}, ρ^{M}) are obtained analogously.

Setting the Number of Negative Examples

While the number $\beta_j = |Tr^{-}(c_j)|$ of negative training examples chosen for each category c_j is not under user control for the ALL and SIBLINGS policies, it can be set by the user for BESTGLOBAL and BESTLOCAL(k). In order to allow a fair comparison between SIBLINGS (as argued in the Introduction, the main focus of our comparative study), BESTGLOBAL, and BESTLOCAL(k), we always choose the same number β_j of negative training examples for the two latter policies as selected by the SIBLINGS policy. Of course, different values of β_j are thus chosen for different $l_j \in L$.

Experimental Results

Parameter Optimization

Before comparing the four policies experimentally, we need to analyze more in detail BESTLOCAL(k) and how it depends on the k parameter. In Figures 2 and 3, we show how BESTLOCAL(k) behaves as a function of k on REUTERS-21578 and RCV1-v2, respectively, as resulting from preliminary experiments we have run in order to optimize this value. In these experiments we have tested all integer values of k up to 20 by performing five-fold cross-validation on the training set.

As evident from these plots, BESTLOCAL(k) proves fairly insensitive to the value of k, both for microaveraged and macroaveraged F_1, and on both datasets; only on the RCV1-v2 dataset macroaveraged F_1 seems to decrease slightly as the value of k increases. In other words, these results suggest that BESTLOCAL(k) performs fairly stably notwithstanding the stand we take on the tradeoff between specificity and robustness mentioned at the end of the section The BESTLOCAL(k) Policy, and that in some cases specificity is slightly more important than robustness.

Overall, these findings thus indicate that k should preferably be set to a low value. This would also have beneficial effects on computational cost, as will be evident from the discussion in the section Efficiency. We have thus set k to 1 for all our experiments.

The Results

Table 2 shows the results obtained with the three learners and four policies discussed on the REUTERS-21578 dataset, whereas Table 3 does the same for the RCV1-v2 data set. Table 4 summarizes these results by averaging across datasets and learners.

One general observation that we can make from Tables 2 and 3 is that TREEBOOST and TREESVM are much more effective than TREENB, regardless of the dataset and of the effectiveness measure used. This is not surprising, and simply confirms the evidence, both theoretical and experimental, that has accumulated over the last 10 years in the machine learning/information retrieval literature concerning the relative merits of boosting, SVMs, and naïve Bayes.

The relative merits of TREEBOOST and TREESVM, as derived from our experiments, are instead less clearcut. In general, TREEBOOST outperforms TREESVM in terms of F_1^{M} (this is so by a large margin on REUTERS-21578). The opposite is instead true for F_1^{μ}, where TREESVM usually outperforms

DOI: 10.1002/asi

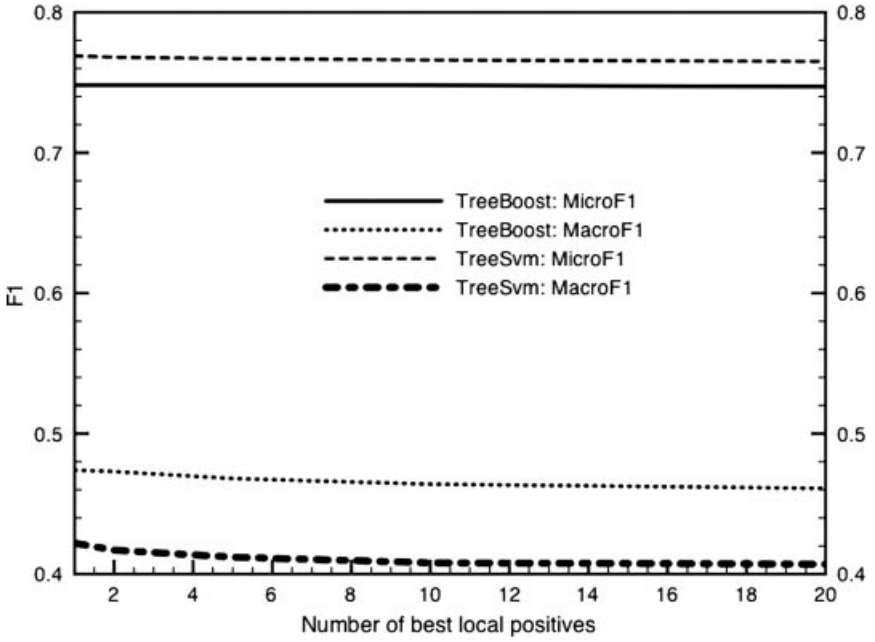

FIG. 2. Influence of parameter k on the effectiveness of the BESTLOCAL(k) policy on REUTERS-21578.

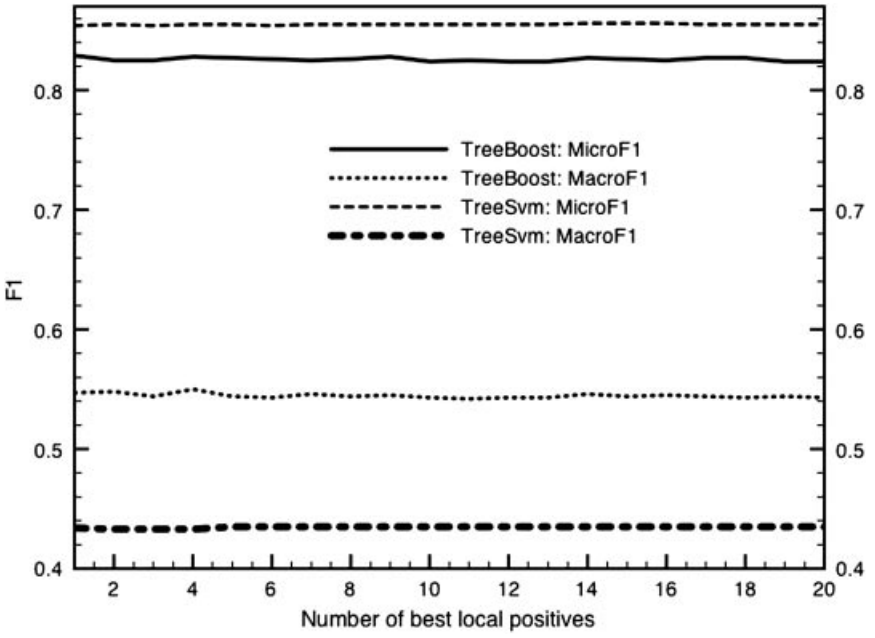

FIG. 3. Influence of parameter k on the effectiveness of the BESTLOCAL(k) policy on RCV1-v2.

TREEBOOST, although by a smaller margin than in the previous case. These two facts indicate that TREESVM is somewhat more effective than its competitor on the more frequent categories, whereas TREEBOOST is decidedly more effective on the less frequent ones; this observation is based on the well-known fact that F_1^{μ} results are dominated by the performance of the classifiers on the more frequent categories, whereas a good behavior on the more infrequent ones is needed in order to make an impact on F_1^{M} (Sebastiani, 2002).

Concerning the differences between our four strategies, the most important observation we can make from Tables 2 and 3 is that ALL is often the winner in terms of precision (it *always* is with TREEBOOST and TREESVM) whereas SIBLINGS is often the winner in terms of recall (again, it *always* is with TREEBOOST and TREESVM). When it comes to balancing precision and recall into F_1, however, the situation is more uncertain, with SIBLINGS and ALL winning out as best performers in approximately the same number of cases.

Although there is no clear winner, there are two clear losers, namely, BESTGLOBAL and BESTLOCAL(k). In fact, neither the former nor the latter ever stand out as the best performer (at least for our two stronger learning methods,

DOI: 10.1002/asi

TABLE 2. Results on REUTERS-21578.

	π^{μ}	ρ^{μ}	F_1^{μ}	π^{M}	ρ^{M}	F_1^{M}
TREEBOOST						
ALL	**0.840**	0.823	**0.831**	**0.835**	0.525	**0.547**
SIBLINGS	0.810	**0.842**	0.826	0.747	**0.538**	0.540
BESTGLOBAL	0.818	0.824	0.821	0.804	0.532	0.545
BESTLOCAL(1)	0.830	0.828	0.829	0.812	0.528	**0.547**
TREESVM						
ALL	**0.912**	0.805	0.855	**0.961**	0.376	0.433
SIBLINGS	0.898	**0.825**	**0.860**	0.951	**0.402**	**0.458**
BESTGLOBAL	0.906	0.810	0.855	0.960	0.379	0.436
BESTLOCAL(1)	0.902	0.811	0.854	0.959	0.379	0.434
TREENB						
ALL	0.680	**0.764**	**0.720**	0.405	**0.218**	0.218
SIBLINGS	0.708	0.635	0.670	0.560	0.193	**0.219**
BESTGLOBAL	0.708	0.658	0.682	**0.689**	0.186	0.205
BESTLOCAL(1)	**0.740**	0.634	0.683	0.665	0.176	0.205

Note. Boldface indicates the best performer.

TABLE 3. Results on RCV1-v2.

	π^{μ}	ρ^{μ}	F_1^{μ}	π^{M}	ρ^{M}	F_1^{M}
TREEBOOST						
ALL	**0.854**	0.685	**0.760**	**0.690**	0.389	0.471
SIBLINGS	0.771	**0.726**	0.748	0.569	**0.469**	**0.492**
BESTGLOBAL	0.777	0.699	0.736	0.594	0.408	0.455
BESTLOCAL(1)	0.794	0.707	0.748	0.597	0.427	0.474
TREESVM						
ALL	**0.945**	0.627	0.754	**0.892**	0.229	0.387
SIBLINGS	0.881	**0.694**	**0.776**	0.807	**0.410**	**0.479**
BESTGLOBAL	0.902	0.664	0.765	0.835	0.332	0.411
BESTLOCAL(1)	0.925	0.658	0.769	0.865	0.336	0.422
TREENB						
ALL	0.636	**0.721**	**0.676**	0.386	**0.342**	**0.327**
SIBLINGS	**0.746**	0.563	0.642	0.566	0.227	0.282
BESTGLOBAL	0.727	0.580	0.646	0.593	0.244	0.298
BESTLOCAL(1)	0.747	0.562	0.641	**0.615**	0.223	0.277

Note. Boldface indicates the best performer.

TABLE 4. Results averaged across two data sets (REUTERS-21578 and RCV1-v2) and three hierarchical learners (TREEBOOST, TREESVM, and TREENB).

	π^{μ}	ρ^{μ}	F_1^{μ}	π^{M}	ρ^{M}	F_1^{M}
ALL	0.811	**0.738**	**0.766**	0.695	0.347	0.397
SIBLINGS	0.802	0.714	0.754	0.700	**0.373**	**0.412**
BESTGLOBAL	0.806	0.706	0.751	0.746	0.347	0.392
BESTLOCAL(1)	**0.823**	0.700	0.754	**0.752**	0.345	0.393

Note. Boldface indicates the best performer.

TREEBOOST and TREESVM), neither for precision nor for recall nor for F_1. This observation is interesting, because it decrees the inferiority of distance-based methods (i.e. methods that are based on the computation of the distance $\delta(d', d'')$ between documents, such as BESTGLOBAL and BESTLOCAL(k)) with respect to the other two.

The average results reported in Table 4 confirm the lack of a clear superiority of one of ALL and SIBLINGS over the other. While ALL is the best performer according to microaveraged F_1, SIBLINGS is the best according to macroaveraged F_1, which shows a better propensity of SIBLINGS to work well with the less highly populated categories.

However, the key observation to be made is that the differences in effectiveness (both for F_1^{μ} and for F_1^{M}) among the four methods are pretty small anyway: For instance, for microaveraged F_1 the average difference (as from Table 4) in performance between the best technique (ALL) and the worst technique (BESTGLOBAL) is 0.015, which is very small. In the absence of a clear winner in terms of effectiveness, efficiency considerations should thus also be considered in order to make an informed choice between our two best techniques. The aim of the section Efficiency will exactly be that of discussing how efficiency may have an impact on the final choice while uninteresting negative documents can bring about better results.

Discussion

It may be interesting to analyze why, as observed in the second section, ALL and SIBLINGS are often (more precisely: They are always in the case of our two stronger learning methods, TREEBOOST and TREESVM) the winners in terms of precision and recall, respectively.

We conjecture that this difference may be due to the fact that ALL makes the classification problem more *imbalanced* (see e.g. Chawla, Japkowicz, & Kolcz, 2004) than SIBLINGS does. In fact, the result of applying ALL is that, when training a binary classifier for class c_j, the size of $Tr^{-}(c_j)$ is much, much larger than the size of $Tr^{+}(c_j)$; for instance, given a REUTERS-21578 leaf category l_j, the fraction of training examples that are positive for l_j is, on average, equal to 0.013. This imbalance between the numbers of positive and negative training examples of a given class is not as severe when SIBLINGS is applied, since SIBLINGS selects much fewer negative examples than ALL does.

Now, the key observation is that "standard" learning methods (such as the ones we have adopted in this work) were not originally devised to handle imbalance, and as a result they tend to emphasize precision at the expense of recall when datasets are imbalanced. In fact, these learning methods were usually devised to optimize *accuracy*, defined as the fraction of classification decisions that are correct (which is an effectiveness measure substantially different from F_1). In a severely imbalanced setting, when negative examples by far outnumber positive ones, a trivial way to optimize accuracy is to "always reject," i.e. to return a negative decision for all test examples: If the positive test examples are, say, 1% of the entire lot (a percentage which, as discussed above, is roughly that of REUTERS-21578), accuracy will be 0.99, i.e. extremely high. A side effect of always rejecting is that precision has a value of 1, since no false positive is generated, but recall has a value of 0, since no true positive is generated.

More realistically, a genuinely trained classifier will likely not *always* reject, but will anyway reject *very often*, since it

DOI: 10.1002/asi

has been trained to optimize accuracy, and being conservative (i.e. tending to reject rather than to accept) is a policy that tends to increase accuracy in imbalanced settings. As a result, since ALL generates a very imbalanced training set, and since our learning methods were originally devised to optimize accuracy, precision tends to be high and recall tends to be low.

Efficiency

The computational cost that the different policies bring about depends on (1) the number of negative examples that are fed to the training phase, and (2) the cost of selecting these negative examples.

In terms of issue (1), ALL is clearly more expensive than SIBLINGS. While the average number of negative training examples per category (averaged across *all* categories, internal, and leaf) generated by the ALL policy was 7583.6 on REUTERS-21578 and 22419.3 on RCV1-v2, the SIBLINGS policy generated 2435.7 on REUTERS-21578 and 4383.8 on RCV1-v2 (i.e. 68% less on REUTERS-21578 and 80% less on RCV1-v2). Since the computational cost of training is, for most supervised learning algorithms, at least linear in the number of training examples (it is certainly so for the three base learners we have used in our experiments (see Esuli, Fagni, & Sebastiani, 2006; Joachims, 2006; McCallum & Nigam, 1998), this translates into a considerable advantage for SIBLINGS at training time. Concerning BESTGLOBAL and BESTLOCAL(k), nothing can be said concerning this aspect, since the number of negative training examples that are selected is chosen by the user. However, BESTGLOBAL and BESTLOCAL(k) are akin in spirit to SIBLINGS, in that their very aim is the *reduction* of the number of negative training examples to be selected; we may thus consider them on a par with SIBLINGS.

In terms of issue (2), however, ALL and SIBLINGS are the clear winners, since they do not require any extra time for individuating the negative training examples. For this, BESTGLOBAL and BESTLOCAL(k) instead require considerable additional time. If we take $\alpha_j = |Tr^+(c_j)|$ and $\gamma_j = |Tr \backslash Tr^+(c_j)|$, BESTGLOBAL requires, for each category, a sum of $O(\alpha_j)$ vectors for computing the centroid and $O(\gamma_j \log \gamma_j)$ vector similarity computations for ranking the set of negative training examples. BESTLOCAL(k) is even more expensive, requiring $O(\alpha_j \gamma_j)$ vector similarity computations and further $O(\gamma_j \alpha_j \log \alpha_j)$ operations for obtaining the $\chi(d_n)$ values of Equation (8), $O(k\gamma_i)$ sums for computing the values from Equation (9), and $O(\gamma_i \log \gamma_i)$ comparisons for finally choosing the negative training examples.

To summarize, SIBLINGS, BESTGLOBAL, and BESTLOCAL(k) are the winners in terms of issue (1), whereas ALL and SIBLINGS are optimal in terms of issue (2). Overall, SIBLINGS is a thus a very clear winner over all the other three policies, at least on counts of efficiency alone. Since it is also one of the two most effective policies, we can conclude that it should indeed be the policy of choice in HTC applications.

Conclusion

We have presented an extensive experimental comparison of four different policies for selecting negative examples in HTC. The comparison has involved hierarchical variants of three popular supervised learning algorithms, and two standard datasets, one of which contains almost 800,000 test documents, which lends robustness and credibility to the conclusions drawn. Our aim was to test the conjecture according to which the best policy for selecting negative examples for a category c is the SIBLINGS policy, namely, that of selecting the examples that are negative for c and positive for the categories that are sibling to c in the category hierarchy. This conjecture, although widespread, had never been subjected to careful scrutiny.

Our experiments have shown that, although no policy systematically outperforms all others, on average the SIBLINGS policy outperforms the others when macroaveraged F_1 is used as the effectiveness measure, and closely trails the best-performing technique when microaveraged F_1 is used instead. Since SIBLINGS is, as we have argued in detail, the policy that carries the smallest computational cost, we have concluded that it should indeed be the policy of choice in HTC applications.

Acknowledgments

This work was carried out in the context of the "Advanced Search Services and Enhanced Technological Solutions for the European Digital Library" (ASSETS) project, funded by the Commission of the European Communities under the ICT Policy Support Programme (ICT PSP). We thank Andrea Esuli for useful discussions on the theme of this article.

References

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.

Chawla, N.V., Japkowicz, N., & Kolcz, A. (2004). Editorial: Special issue on learning from imbalanced data sets. ACM SIGKDD Explorations, 6(1), 1–6.

Cheng, C.-H., Tang, J., Wai-Chee, A., & King, I. (2001). Hierarchical classification of documents with error control. In Proceedings of the Fifth Pacific–Asia Conference on Knowledge Discovery and Data Mining (PAKDD'01). Lecture Notes in Artificial Intelligence, 2035, 433–443.

Chiang, J.-H., & Chen, Y.-C. (2001). Hierarchical fuzzy-KNN networks for news documents categorization. In Proceedings of the 10th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'01), (Vol. 2, pp. 720–723). Washington, DC: IEEE.

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). New York: ACM Press.

Esuli, A., Fagni, T., & Sebastiani, F. (2006). MP-Boost: A multiple-pivot boosting algorithm and its application to text categorization. In Proceedings of the 13th International Symposium on String Processing and Information Retrieval (SPIRE'06). Lecture Notes in Computer Science, 4209, 1–12.

Esuli, A., Fagni, T., & Sebastiani, F. (2008). Boosting multi-label hierarchical text categorization. Information Retrieval, 11(4), 287–313.

DOI: 10.1002/asi

Fall, C.J., Törcsvári, A., Benzineb, K., & Karetka, G. (2003). Automated categorization in the international patent classification. SIGIR Forum, 37(1), 10–25.

Gauch, S., Ranganathan, S., & Chandramouli, A. (2009). Training a hierarchical classifier using inter-document relationships. Journal of the American Society for Information Science and Technology, 60(1), 47–58.

Hersh, W., Buckley, C., Leone, T., & Hickman, D. (1994). OHSUMED: An interactive retrieval evaluation and new large text collection for research. In Proceedings of the 17th ACM International Conference on Research and Development in Information Retrieval (SIGIR'94) (pp. 192–201). New York: ACM Press.

Joachims, T. (1999). Making large-scale SVM learning practical. In B. Schölkopf, C.J. Burges, & A.J. Smola (Eds.), Advances in Kernel Methods–Support Vector Learning (pp. 169–184). Cambridge, MA: The MIT Press.

Joachims, T. (2006). Training linear SVMs in linear time. In Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD'06) (pp. 217–226). New York: ACM Press.

Kiritchenko, S., Matwin, S., Nock, R., & Famili, A.F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In Proceedings of the 19th Canadian Conference on Artificial Intelligence (AI'06). Lecture Notes in Artificial Intelligence, 4013, 395–406.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. Paper presented at Proceedings of the 14th International Conference on Machine Learning (ICML'97) (pp. 170–178). San Francisco, Morgan Kauffman.

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

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

Liu, R.-L. (2009). Context recognition for hierarchical text classification. Journal of the American Society for Information Science and Technology, 60(4), 803–813.

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

McCallum, A.K., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In Proceedings of the AAAI Workshop on Learning for Text Categorization (pp. 41–48). Menlo Park, CA: AAAI.

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). New York: ACM Press.

Rose, T., Stevenson, M., & Whitehead, M. (2002). The reuters corpus volume 1—From yesterday's news to tomorrow's language resources. Paper presented at Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02) (pp. 827–832). Paris: ELDA.

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

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5), 513–523.

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). New York: ACM Press.

Sebastiani, F. (2002). Machine learning in automated text categorization. ACM Computing Surveys, 34(1), 1–47.

Singhal, A., Mitra, M., & Buckley, C. (1997). Learning routing queries in a query zone. In Proceedings of the 20th ACM International Conference on Research and Development in Information Retrieval (SIGIR'97) (pp. 25–32). New York: ACM Press.

Sun, A., Lim, E.-P., & Ng, W.K. (2003). Performance measurement framework for hierarchical text classification. Journal of the American Society for Information Science and Technology, 54(11), 1014–1028.

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

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 Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95), Las Vegas, NV (pp. 317–332).

Yoon, Y., Lee, C., & Lee, G.G. (2006). An effective procedure for constructing a hierarchical text classification system. Journal of the American Society for Information Science and Technology, 57(3), 431–442.

DOI: 10.1002/asi