On the Selection of Negative Examples for Hierarchical Text Categorization

Tiziano Fagni and Fabrizio Sebastiani

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

Abstract

Hierarchical text categorization (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 sibling to c in the hierarchy. However, this policy has always been taken for granted and never been subjected to careful scrutiny since first being proposed ten years ago. This paper 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 paper. We compare these policies on the hierarchical versions of two among the most important classes of supervised learning algorithms, boosting and support vector machines, by performing experiments on two standard TC datasets, REUTERS-21578 and RCV1-V2.

1. 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.

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 either be a tree or 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, 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.

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 and 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 et al., 1998; Koller and Sahami, 1997). This is fundamental when tackling classification tasks characterised by very high numbers of categories, as is the case e.g., of the OHSUMED dataset (Hersh et al., 1994), the WIPO-ALPHA dataset (Fall et al., 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 for achieving this is performing feature selection "locally" (Koller and 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 sibling to it 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 classification too).

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 are negative for c and positive for the categories sibling to c in the hierarchy. However, this policy (hereafter called the SIBLINGS policy) has always been taken for granted and never been subjected to careful scrutiny since first being proposed in (Wiener et al., 1995).

This paper 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 paper. We provide an intuitive basis for these policies and test them on the hierarchical versions of two among the most important classes of supervised learning algorithms, boosting and support vector machines, 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).

The paper is organized as follows. In Section 2. we outline the basic scheme for learning hierarchical text clas-

[1] Consistently with most mathematical literature we use the caret symbol (^) to indicate estimation.

sifiers that, in our experiments, we will instantiate with boosting and SVMs as base learners. Section 3. describes in detail the four policies for the selection of negative training examples, while Section 4. describes the comparative experiments we have run. Section 5. concludes.

2. 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 fix some notation. Let $H = \langle I, L\rangle$ be a tree-structured set of categories, where $I = \langle\langle i_1, Tr^+(i_1)\rangle, \ldots, \langle i_n, Tr^+(i_n)\rangle\rangle$ and $L = \langle\langle l_1, Tr^+(l_1)\rangle, \ldots, \langle l_m, Tr^+(l_m)\rangle\rangle$ 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 $r \in I$ 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 sibling categories of c_j

Table 1: Notational conventions.

We also assume that documents can belong to zero, one, or several leaf categories in L, and 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

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

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 Section 4.2. we will alternatively instantiate the pattern by a boosting-based learner or by an SVM-based learner, thus generating the TREEBOOST and TREESVM 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.

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

```
procedure TREELEARNER (H, r, np, learner)
begin
    if not (r is a leaf category) then
        foreach child in ↓(r) do
            Tr⁻(child) = getNegatives(H, r, child,
            np);
            train(child, learner);
            TREELEARNER(H, child, np, learner);
        end
    else
        do nothing;
    end
end
```

Figure 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.

3. Choosing negative examples in hierarchical text categorization

In this work we have tested 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. Moreover, for each proposed method, we give details about its computational cost by describing the cost of selecting negative documents for each category and the impact that the number of selected examples has on the learning phase.

3.1. 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*

[2]Throughout this paper we will always refer to tree-shaped hierarchies; however, all our arguments straightforwardly apply to DAG-shaped hierarchies (see 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 map H into an "extended" set of categories H' by appending to every internal node c_j of H an additional child (leaf) node c'_j, and by moving into c'_j all documents originally contained in c_j. This mapping, originally proposed in (Cheng et al., 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") which contains all documents belonging to the node but to none of its descendant leaves.

grounds alone. That is, it is composed of all the training documents which are not positive for c_j and positive for the categories sibling of c_j: i.e.[4],

$$Tr^-(c_j) = \left(\bigcup_{c \in \leftrightarrow(c_j)} Tr^+(c) \right) / \, Tr^+(c_j) \qquad (1)$$

There are two main intuitions behind this policy.

The first intuition is that, *if the classifier associated to $\uparrow(c_j)$ has generated no false positives*, the classifier associated to 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 multi-label problem of classifying documents into a hierarchy $H = \langle I, L \rangle$ is decomposed into several flat classification problems, one for each $i_j \in I$, in which the set of categories concerned is $\downarrow(i_j)$.

The SIBLINGS policy, originally proposed in (Wiener et al., 1995), was subsequently adopted in, e.g., (Chiang and Chen, 2001; Dumais and Chen, 2000; Esuli et al., 2006b; Liu et al., 2005; Ng et al., 1997; Ruiz and Srinivasan, 2002; Sun and Lim, 2001; Weigend et al., 1999), and quickly became the standard choice for HTC contexts.

3.2. 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.,

$$Tr^-(c_j) = Tr/Tr^+(c_j) \qquad (2)$$

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. This policy was used in what can be considered the very first HTC paper (Wiener et al., 1995), but was soon superseded by the SIBLINGS policy. However, 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 et al., 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 to $\uparrow(c_j)$ may indeed generate some false positives, typically corresponding to documents that belong neither to c_j nor to any of its siblings, but to some other category in H. 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 tested on.

The second intuition is that "the more 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.

3.3. 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 non-topological considerations. While it has never been used to date in a hierarchical context, BESTGLOBAL simply coincides with the "query zoning" selection strategy proposed in (Singhal et al., 1997) for flat classification, and subsequently used in (Schapire et al., 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 by[5]

$$\zeta(c_j) = \frac{1}{|Tr^+(c_j)|} \sum_{d_p \in Tr^+(c_j)} d_p \qquad (3)$$

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

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

where $\arg \min^{z}_{A} f$ indicates the bottom-ranked z elements of A according to 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 documents of c_j that are most similar to the positive documents of c_j are likely to be the positive documents of c_j's siblings. BESTGLOBAL instead equates similarity with closeness in the

[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 deriving from the multiple parents of c_j. Note that the three other policies we discuss in this paper do not use the hierarchical structure of the category set, hence they work for tree- and DAG-shaped hierarchies alike.

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

vector space in which the documents are represented. SIBLINGS is thus a policy specific to a hierarchical setting, while BESTGLOBAL is not.

3.4. 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 negative 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 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 neighbour* of document d_n to be

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

our policy selects the β_j negative training documents d_n closest to $\chi(d_n)$, i.e.

$$\arg \overset{\beta_j}{\min_{d_n \in Tr/Tr^+(c_j)}} \delta(\chi(d_n), d_n)$$

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* 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 neighbours* of document d_n to be

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

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.

$$\arg \overset{\beta_j}{\min_{d_n \in Tr/Tr^+(c_j)}} \sum_{d_p \in \chi^k(d_n)} \delta(d_p, d_n)$$

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

Similarly 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 vector in 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, while higher values of k bring about simple separating surfaces that fit the model less but tend to be more robust.

4. Experiments

4.1. 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 in (Toutanova et al., 2001) by the application of hierarchical agglomerative clustering on the 90 REUTERS-21578 categories that have at least one positive training example and one positive test example. The original REUTERS-21578 categories are thus "leaf" categories in the resulting hierarchy, and are clustered into four "macro-categories" (see Table 2) whose parent category is the root of the tree. Conforming to the experiments of (Toutanova et al., 2001), we have used (according to the ModApte split) the 7,770 training examples and 3,299 test examples that are labelled by at least one of the selected 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 examples per leaf category is 106.50, 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 text categorization dataset made available by Reuters and consisting of 804,414 news stories produced by Reuters from 20 Aug 1996 to 19 Aug 1997; all news stories are in English, and have 109 distinct terms per document on average (Rose et al., 2002). In our experiments we have used the "LYRL2004" split defined in (Lewis et al., 2004), in which the (chronologically) first 23,149 documents are used for training and the other 781,265 are used for testing. 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. The average number of leaf categories per document

[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 .

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

Table 2: REUTERS-21578 macro-categories and their member categories (from (Toutanova et al., 2001)).

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 729.67, ranging from a minimum of 1 to a maximum of 10,786 documents.

4.2. Experimental settings and evaluation measures

As the base learner for the TREELEARNER procedure we have decided to use an SVM-based learner and a boosting-based learner, since kernel machines and boosting are currently two among the classes of supervised learning devices that tend to obtain the best performance in a variety of learning tasks and, at the same time, have strong justifications from computational learning theory. The first learner is the SVM implementation embodied in the `svm_light` package (Joachims, 1998)[8], which we have run with a linear kernel and its parameters set at their default values. In the experiments this configuration will be referred to as TREESVM. The other learning algorithm we have used is MP-Boost (Esuli et al., 2006a), a learner based on boosting technology, which we have obtained from the authors. In all the experiments the algorithm has been run with a number of iterations fixed to 1,000. In the rest of the article this configuration will be referred to as TREEBOOST[9].

In all the experiments discussed in this section, punctuation has been removed, all letters have been converted to lowercase, numbers have been removed, stop words have been removed using the stop list provided in (Lewis, 1992, pages 117–118), 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 by the "ltc" variant (Salton and Buckley, 1988)) of the well-known $tfidf$ class of weighting functions, i.e.

$$tfidf(t_k, d_j) = tf(t_k, d_j) \cdot \log \frac{|Tr|}{\#_{Tr}(t_k)} \qquad (4)$$

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

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

where $\#(t_k, d_j)$ denotes the number of times t_k occurs in d_j. Weights obtained by Equation 4 are normalized through cosine normalization, i.e.

$$w_{kj} \;=\; \frac{tfidf(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} tfidf(t_s, d_j)^2}} \qquad (5)$$

The vectors provided as input to TREEBOOST are instead binary; this is a constraint imposed by the use of MP-Boost as base learner.

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

$$F_1 = \frac{2\pi\rho}{\pi + \rho} = \frac{2TP}{2TP + FP + FN} \qquad (6)$$

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 to equal 1.0, since the classifier has correctly classified all documents as negative examples.

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 Equation 6. F_1^M is obtained by first computing the F_1 values specific to the individual categories, and then averaging them across the c_i's. Of course, only leaf categories are considered in the evaluation.

4.3. Results

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) and BESTGLOBAL / BESTLOCAL(k), for these two latter policies we always choose the same number β_j of negative training examples as selected by the SIBLINGS policy. Of course, different β_j are thus chosen for different categories.

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

[9]Note that our TREEBOOST is different from the TREEBOOST.MH algorithm of (Esuli et al., 2006b); in fact, while we use MP-Boost as base learner, TREEBOOST.MH uses ADABOOST.MH (Schapire and Singer, 2000).

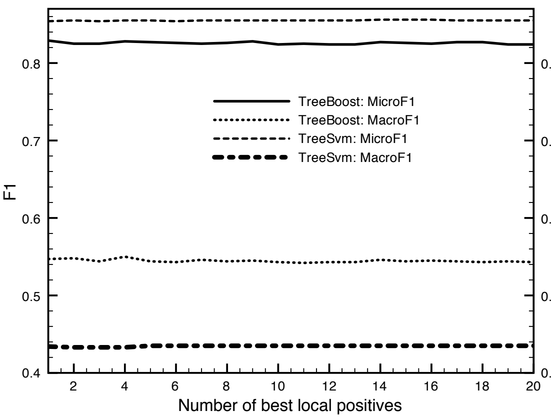

Figure 2: Influence of parameter k on the effectiveness of the BESTLOCAL(k) policy on REUTERS-21578.

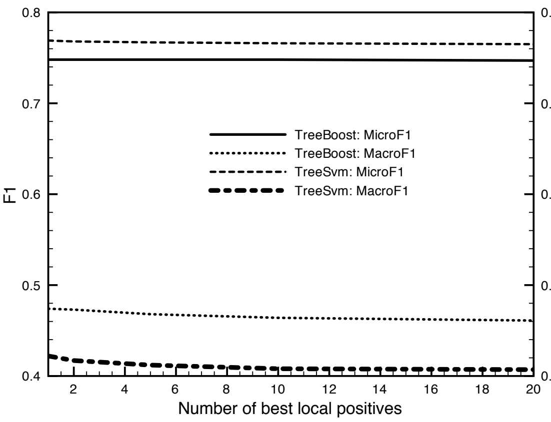

Figure 3: Influence of parameter k on the effectiveness of the BESTLOCAL(k) policy on RCV1-V2.

	π^μ	ρ^μ	F_1^μ	π^M	ρ^M	F_1^M
	TREEBOOST					
ALL	**.840**	.823	**.831**	**.835**	.525	**.547**
SIBLINGS	.810	**.842**	.826	.747	**.538**	.540
BESTGLOBAL	.818	.824	.821	.804	.532	.545
BESTLOCAL(1)	.830	.828	.829	.812	.528	**.547**
	TREESVM					
ALL	**.912**	.805	.855	**.961**	.376	.433
SIBLINGS	.898	**.825**	**.860**	.951	**.402**	**.458**
BESTGLOBAL	.906	.810	.855	.960	.379	.436
BESTLOCAL(1)	.902	.811	.854	.959	.379	.434

	π^μ	ρ^μ	F_1^μ	π^M	ρ^M	F_1^M
	TREEBOOST					
ALL	**.854**	.685	**.760**	**.690**	.389	.471
SIBLINGS	.771	**.726**	.748	.569	**.469**	**.492**
BESTGLOBAL	.777	.699	.736	.594	.408	.455
BESTLOCAL(1)	.794	.707	.748	.597	.427	.474
	TREESVM					
ALL	**.945**	.627	.754	**.892**	.229	.387
SIBLINGS	.881	**.694**	**.776**	.807	**.410**	**.479**
BESTGLOBAL	.902	.664	.765	.835	.332	.411
BESTLOCAL(1)	.925	.658	.769	.865	.336	.422

Table 3: Results on REUTERS-21578 (top) and RCV1-V2 (bottom).

	π^μ	ρ^μ	F_1^μ	π^M	ρ^M	F_1^M
ALL	**.894**	.705	.785	**.837**	.377	.446
SIBLINGS	.878	**.728**	**.792**	.822	**.397**	**.457**
BESTGLOBAL	.886	.716	.788	.830	.387	.452
BESTLOCAL(1)	.886	.716	.788	.830	.387	.452

Table 4: Results averaged across two datasets (REUTERS-21578 and RCV1-V2) and two hierarchical learners (TREEBOOST and TREESVM).

4.3.1. Effectiveness

Before comparing the four policies we need to analyze more in detail the BESTLOCAL(k) policy 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. We have tested all integer values of k up to 20 by 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 micro- 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. These results suggest setting k to a low value; we have thus fixed it to 1 for all our experiments.

Table 3 shows the results obtained with the four policies discussed, on REUTERS-21578 and on RCV1-V2, with TREEBOOST and with TREESVM; Table 4 summarizes these results by averaging across datasets and learners. The most important observation we can make from Table 3 is that ALL is always the winner in terms of precision and SIBLINGS is always the winner in terms of recall. 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. However, Table 4 shows that, in the average, (i) SIBLINGS performs better than ALL, and (ii) BESTGLOBAL and BESTLOCAL(k), while never the best performers, always perform fairly well, actually better than ALL on average.

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: more precisely, they are very small on REUTERS-21578 and slightly more marked in RCV1-V2, in particular for F_1^M.

4.3.2. Efficiency

In the absence of a clear winner in terms of effectiveness, efficiency considerations should also be considered. The computational cost that the different policies bring about depends on (i) the number of negative examples that are fed to the training phase, and (ii) the cost of selecting these negative examples.

In terms of issue (i), 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 two base learners we have used in our experiments: see (Joachims, 2006; Esuli et al., 2006a)) this translates in 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 (ii), 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 indicate with α_j and β_j the numbers of positive and (selected) negative training examples for c_j, BESTGLOBAL requires, for each category, $O(\alpha_j)$ sums of vectors for computing the centroid and $O(\beta_j \log \beta_j)$ vector similarity computations for ranking the set of negative training examples. BESTLOCAL(k) is even more expensive, requiring $O(\alpha_j \beta_j)$ vector similarity computations for obtaining the $\chi(d_n)$ values and $O(\beta_j \log \beta_j)$ comparisons for finally choosing the negative training examples.

All in all, it is clear that, on grounds of efficiency alone, SIBLINGS wins on the other three policies. Since it is also one of the two most effective policies, this means it should indeed be the policy of choice in HTC applications.

5. Conclusion

We have presented an extensive experimental comparison among four different policies for selecting negative examples in hierarchical text classification. Our aim was to test the conjecture according to which the best policy for selective negative examples for a category c_j is the SIBLINGS policy, namely, that of selecting the examples that are negative for c_j and positive for the categories that are sibling to c_j in the category hierarchy. This conjecture, although widespread, had never been subjected to careful scrutiny.

Our experiments, conducted by using hierarchical versions of two major learning algorithms on two popular text categorization datasets, have shown that, although no policy systematically outperforms all others, the SIBLINGS policy outperforms the others on average. Since SIBLINGS is, as we have argued, the policy which carries the smallest computational cost, we can conclude that is should indeed be the policy of choice in HTC applications.

6. References

Chakrabarti, Soumen, Byron E. Dom, Rakesh Agrawal, and Prabhakar Raghavan, 1998. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *Journal of Very Large Data Bases*, 7(3):163–178.

Cheng, Chun-Hung, Jian Tang, Ada Wai-Chee, and Irwin King, 2001. Hierarchical classification of documents with error control. In *Proceedings of the 5th Pacific-Asia Conferenece on Knowledge Discovery and Data Mining (PAKDD'01)*. Hong Kong, CN.

Chiang, Jung-Hsien and Yan-Cheng Chen, 2001. Hierarchical fuzzy-knn networks for news documents categorization. In *Proceedings of the 10th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'01)*, volume 2. Melbourne, AU.

Dumais, Susan T. and Hao Chen, 2000. Hierarchical classification of web content. In *Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR'00)*. Athens, GR.

Esuli, Andrea, Tiziano Fagni, and Fabrizio Sebastiani, 2006a. 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)*. Glasgow, UK.

Esuli, Andrea, Tiziano Fagni, and Fabrizio Sebastiani, 2006b. TreeBoost.MH: A boosting algorithm for multi-label hierarchical text categorization. In *Proceedings of the 13th International Symposium on String Processing and Information Retrieval (SPIRE'06)*. Glasgow, UK.

Fall, C. J., A. Törcsvári, K. Benzineb, and G. Karetka, 2003. Automated categorization in the International Patent Classification. *SIGIR Forum*, 37(1):10–25.

Hersh, William, Christopher Buckley, T.J. Leone, and David Hickman, 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)*. Dublin, IE.

Joachims, Thorsten, 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning (ECML'98)*. Chemnitz, DE.

Joachims, Thorsten, 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD'06)*. Philadelphia, US.

Kiritchenko, Svetlana, Stan Matwin, Richard Nock, and A. Fazel Famili, 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)*. Québec City, CA.

Koller, Daphne and Mehran Sahami, 1997. Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning (ICML'97)*. Nashville, US.

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

Lewis, David D., Fan Li, Tony Rose, and Yiming Yang, 2004. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

Liu, Tie-Yan, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma, 2005. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations*, 7(1):36–43.

Ng, Hwee T., Wei B. Goh, and Kok L. Low, 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)*. Philadelphia, US.

Rose, Tony, Mark Stevenson, and Miles Whitehead, 2002. The Reuters Corpus Volume 1 – from yesterday's news to tomorrow's language resources. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC'02)*. Las Palmas, ES.

Ruiz, Miguel and Padmini Srinivasan, 2002. Hierarchical text classification using neural networks. *Information Retrieval*, 5(1):87–118.

Salton, Gerard and Christopher Buckley, 1988. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.

Schapire, Robert E. and Yoram Singer, 2000. BOOSTEXTER: a boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

Schapire, Robert E., Yoram Singer, and Amit Singhal, 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)*. Melbourne, AU.

Singhal, Amit, Mandar Mitra, and Chris Buckley, 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)*. Philadelphia, US.

Sun, Aixin and Ee-Peng Lim, 2001. Hierarchical text classification and evaluation. In *Proceedings of the 1st IEEE International Conference on Data Mining (ICDM'01)*. San Jose, US.

Toutanova, Kristina, Francine Chen, Kris Popat, and Thomas Hofmann, 2001. Text classification in a hierarchical mixture model for small training sets. In *Proceedings of the 10th ACM International Conference on Information and Knowledge Management (CIKM'01)*. Atlanta, US.

Weigend, Andreas S., Erik D. Wiener, and Jan O. Pedersen, 1999. Exploiting hierarchy in text categorization. *Information Retrieval*, 1(3):193–216.

Wiener, Erik D., Jan O. Pedersen, and Andreas S. Weigend, 1995. 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.