Machine Learning, 39, 135–168, 2000.

BoosTexter: A Boosting-based System for Text Categorization

ROBERT E. SCHAPIRE schapire@research.att.com
AT&T Labs, Shannon Laboratory, 180 Park Avenue, Room A279, Florham Park, NJ 07932-0971, USA

YORAM SINGER singer@cs.huji.ac.il
School of Computer Science & Engineering, The Hebrew University, Jerusalem 91904, Israel

Editors: Jaime Carbonell and Yiming Yang

Abstract. This work focuses on algorithms which learn from examples to perform multiclass text and speech categorization tasks. Our approach is based on a new and improved family of boosting algorithms. We describe in detail an implementation, called BoosTexter, of the new boosting algorithms for text categorization tasks. We present results comparing the performance of BoosTexter and a number of other text-categorization algorithms on a variety of tasks. We conclude by describing the application of our system to automatic call-type identification from unconstrained spoken customer responses.

Keywords: text and speech categorization, multiclass classification problems, boosting algorithms

1. Introduction

Text categorization is the problem of classifying text documents into categories or classes. For instance, a typical problem is that of classifying news articles by topic based on their textual content. Another problem is to automatically identify the type of call requested by a customer; for instance, if the customer says, “Yes, I would like to charge this call to my Visa,” we want the system to recognize that this is a calling-card call and to process the call accordingly. (Although this is actually a speech-categorization problem, we can nevertheless apply a text-based system by passing the spoken responses through a speech recognizer.)

In this paper, we introduce the use of a machine-learning technique called boosting to the problem of text categorization. The main idea of boosting is to combine many simple and moderately inaccurate categorization rules into a single, highly accurate categorization rule. The simple rules are trained sequentially; conceptually, each rule is trained on the examples which were most difficult to classify by the preceding rules.

Our approach is based on a new and improved family of boosting algorithms which we have described and analyzed in detail in a companion paper (Schapire & Singer, 1998). This new family extends and generalizes Freund and Schapire’s AdaBoost algorithm (Freund & Schapire, 1997), which has been studied extensively and which has been shown to perform well on standard machine-learning tasks (Breiman, 1998; Drucker & Cortes, 1996; Freund

& Schapire, 1996, 1997; Maclin & Opitz, 1997; Margineantu & Dietterich, 1997; Quinlan, 1996; Schapire, 1997; Schapire et al., 1998). The purpose of the current work is to describe some ways in which boosting can be applied to the problem of text categorization, and to test its performance relative to a number of other text-categorization algorithms.

Text-categorization problems are usually *multiclass* in the sense that there are usually more than two possible categories. Although in some applications there may be a very large number of categories, in this work, we focus on the case in which there are a small to moderate number of categories. It is also common for text-categorization tasks to be *multi-label*, meaning that the categories are not mutually exclusive so that the same document may be relevant to more than one category. For instance, bibliographic medical articles are routinely given multiple Medical Subject Index (MeSH) categories when entered into Medline, the national bibliographic searchable archive which contains more than twenty million documents. While most machine-learning systems are designed to handle multiclass data, much less common are systems that can handle multi-label data. While numerous categorization algorithms, such as k-nearest neighbor, can be adapted to multi-label categorization problems, when machine-learning and other approaches are applied to text-categorization problems, a common technique has been to decompose the multi-class, multi-label problem into multiple, independent binary classification problems (one per category).

In this paper, we adopt a different approach in which we use two extensions of AdaBoost that were specifically intended for multiclass, multi-label data. In the first extension, the goal of the learning algorithm is to predict all and only all of the correct labels. Thus, the learned classifier is evaluated in terms of its ability to predict a good approximation of the set of labels associated with a given document. In the second extension, the goal is to design a classifier that *ranks* the labels so that the correct labels will receive the highest ranks. We next describe BoosTexter, a system which embodies four versions of boosting based on these extensions, and we discuss the implementation issues that arise in multi-label text categorization.

There has been voluminous work done on text categorization, including techniques based on decision trees, neural networks, nearest neighbor methods, Rocchio's method, support-vector machines, linear least squares, naive Bayes, rule-based methods and more. (See, for instance, Apté, Damerau, & Weiss (1994), Biebricher et al. (1988), Cohen & Singer (1996), Field (1975), Fuhr & Pfeifer (1994), Koller & Sahami (1997), Lewis & Ringuette (1994), Moulinier, Raškinis, & Ganascia (1996), Ng, Goh, & Low (1997) and Yang (1994)). It would be impossible for us to compare our algorithms to all of the previous methods. Instead, we compare to four very different methods which we believe are representative of some of the most effective techniques available, and report results on several different tasks. Our experiments show that, using a number of evaluation measures, our system's performance is generally better than the other algorithms, sometimes by a wide margin.

To further compare our algorithm to other methods, we tested the performance of BoosTexter on a standard benchmark problem so that performance could be compared directly with a large body of results reported in the literature. We specifically focus on a recent study by Yang (to appear) who conducted several experiments on this benchmark and who also surveyed many results reported by other authors. BoosTexter's performance places it at the very top of all the methods included in Yang's study.

Finally, we discuss the application of BoosTexter to an automatic speech-categorization task and compare the performance of our system to a previous algorithm which was specifically designed for this task.

2. Preliminaries

In this section, we describe the formal setting we use to study multi-label text categorization.

Let $\mathcal{X}$ denote the domain of possible text documents and let $\mathcal{Y}$ be a finite set of labels or classes. We denote the size of $\mathcal{Y}$ by $k = |\mathcal{Y}|$.

In the traditional machine-learning setting, each document $x \in \mathcal{X}$ is assigned a single class $y \in \mathcal{Y}$. The goal then, typically, is to find a classifier $H : \mathcal{X} \to \mathcal{Y}$ which minimizes the probability that $y \neq H(x)$ on a newly observed example (x, y). In the multi-label case, each document $x \in \mathcal{X}$ may be assigned multiple labels in $\mathcal{Y}$. For example, in a multiclass news-filtering problem in which the possible classes are `News`, `Finance` and `Sports`, a document may belong to both `News` *and* `Finance`. Thus, a labeled example is a pair (x, Y) where $Y \subseteq \mathcal{Y}$ is the set of labels assigned to x. The single-label case is a special case in which $|Y| = 1$ for all observations.

For $Y \subseteq \mathcal{Y}$, let us define $Y[\ell]$ for $\ell \in \mathcal{Y}$ to be

$$Y[\ell] = \begin{cases} +1 & \text{if } \ell \in Y \\ -1 & \text{if } \ell \notin Y. \end{cases}$$

In this paper, we will be primarily interested in classifiers which produce a *ranking* of the possible labels for a given document with the hope that the appropriate labels will appear at the top of the ranking. To be more formal, the goal of learning is to produce a function of the form $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ with the interpretation that, for a given instance x, the labels in $\mathcal{Y}$ should be ordered according to $f(x, \cdot)$. That is, a label ℓ_1 is considered to be ranked higher than ℓ_2 if $f(x, \ell_1) > f(x, \ell_2)$. If Y is the associated label set for x, then a successful learning algorithm will tend to rank labels in Y higher than those not in Y. Precise evaluation measures are discussed in Section 5.

Finally, to simplify the notation, for any predicate π, let $[\![\pi]\!]$ be 1 if π holds and 0 otherwise.

3. Boosting algorithms for multi-label multiclass problems

In a companion paper (Schapire & Singer, 1998), we introduced and analyzed two new boosting algorithms for multiclass, multi-label classification problems. Here, we review the two algorithms, discuss four versions of these algorithms, and describe an efficient implementation of the algorithms for the problem of text categorization.

The purpose of boosting is to find a highly accurate classification rule by combining many *weak* or *base hypotheses*, each of which may be only moderately accurate. We assume access to a separate procedure called the *weak learner* or *weak learning algorithm* for computing the weak hypotheses. The boosting algorithm finds a set of weak hypotheses by calling the weak learner repeatedly in a series of rounds. These weak hypotheses are then combined into a single rule called the *final* or *combined hypothesis*.

In the simplest version of AdaBoost for single-label classification, the boosting algorithm maintains a set of importance weights over training examples. These weights are used by the weak learning algorithm whose goal is to find a weak hypothesis with moderately low error with respect to these weights. Thus, the boosting algorithm can use these weights to force the weak learner to concentrate on the examples which are hardest to classify.

As we will see, for multiclass, multi-label problems, it is appropriate to maintain instead a set of weights over training examples *and labels*. As boosting progresses, training examples and their corresponding labels that are hard to predict correctly get incrementally higher weights while examples and labels that are easy to classify get lower weights. For instance, for the news classification problem, it might be easy to classify a document as a news item but hard to determine whether or not it belongs to the finance section. Then, as boosting progresses the weight of the label `News` for that document decreases while the weight of `Finance` increases. The intended effect is to force the weak learning algorithm to concentrate on examples and labels that will be most beneficial to the overall goal of finding a highly accurate classification rule.

3.1. AdaBoost.MH

Our first boosting algorithm for multiclass multi-label classification problems, called AdaBoost.MH, is shown in figure 1. Let S be a sequence of training examples $\langle(x_1, Y_1), \ldots, (x_m, Y_m)\rangle$ where each instance $x_i \in \mathcal{X}$ and each $Y_i \subseteq \mathcal{Y}$. As described above,

```
Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$ where $x_i \in \mathcal{X}$, $Y_i \subseteq \mathcal{Y}$
Initialize $D_1(i, \ell) = 1/(mk)$.
For $t = 1, \ldots, T$:
    • Pass distribution $D_t$ to weak learner.
    • Get weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.
    • Choose $\alpha_t \in \mathbb{R}$.
    • Update:
          $D_{t+1}(i, \ell) = \frac{D_t(i, \ell) \exp(-\alpha_t Y_i[\ell] h_t(x_i, \ell))}{Z_t}$
      where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
Output the final hypothesis:
    $f(x, \ell) = \sum_{t=1}^{T} \alpha_t h_t(x, \ell)$.
```

Figure 1. The algorithm AdaBoost.MH.

AdaBoost.MH maintains a set of weights as a distribution D_t over examples and labels. Initially, this distribution is uniform. On each round t, the distribution D_t (together with the training sequence S) is passed to the weak learner who computes a weak hypothesis h_t. The output of the weak learner is a hypothesis $h : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. We interpret the sign of $h(x, \ell)$ as a prediction as to whether the label ℓ is or is not assigned to x (i.e., a prediction of the value of $Y[\ell]$). The magnitude of the prediction $|h(x, \ell)|$ is interpreted as a measure of "confidence" in the prediction. The precise goal of the weak learner is described below, as are the weak learners used in our experiments.

A parameter α_t is then chosen and the distribution D_t is updated. We discuss the choice of α_t below. In the typical case that α_t is positive, the distribution D_t is updated in a manner that increases the weight of example-label pairs which are misclassified by h_t (i.e., for which $Y_i[\ell]$ and $h_t(x_i, \ell)$ differ in sign). The final hypothesis ranks documents using a weighted vote of the weak hypotheses.

This algorithm is derived using a natural reduction of the multiclass, multi-label data to binary data. Under this reduction, each example (x, Y) is mapped to k binary-labeled examples of the form $((x, \ell), Y[\ell])$ for all $\ell \in \mathcal{Y}$; that is, the instance or "document" part of each derived example is formally a pair (x, ℓ), and the binary label associated with this instance is $Y[\ell]$. In other words, we can think of each observed label set Y as specifying k binary labels (depending on whether a label ℓ is or is not included in Y), and we can then apply binary AdaBoost to the derived binary data. The algorithm that results from such a reduction is equivalent to AdaBoost.MH.

This view of AdaBoost.MH also leads to a simple analysis. Specifically, we have proved (Schapire & Singer, 1998) a bound on the empirical *Hamming loss* of this algorithm, i.e., the fraction of examples i and labels ℓ for which the sign of $f(x_i, \ell)$ differs from $Y_i[\ell]$. We showed that the Hamming loss of this algorithm is at most $\prod_{t=1}^{T} Z_t$, where Z_t is the normalization factor computed on round t. This upper bound can be used in guiding both our choice of α_t and the design of our weak learning algorithm. Together, these choices should be geared on each round t toward the minimization of

$$Z_t = \sum_{i=1}^{m} \sum_{\ell \in \mathcal{Y}} D_t(i, \ell) \exp(-\alpha_t \, Y_i[\ell] \, h_t(x_i, \ell)). \tag{1}$$

In Section 4, we describe the methods used for choosing α_t and the implementation of the weak learning algorithm for text categorization.

Note that the space and time-per-round requirements of AdaBoost.MH are $O(mk)$, not including the call to the weak learner.

3.2. AdaBoost.MR

We next describe our second boosting algorithm called AdaBoost.MR. Whereas AdaBoost.MH is designed to minimize Hamming loss, AdaBoost.MR is designed specifically to find a hypothesis which *ranks* the labels in a manner that hopefully places the correct labels at the top of the ranking.

```
Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$ where $x_i \in \mathcal{X}$, $Y_i \subseteq \mathcal{Y}$
Initialize $D_1(i, \ell_0, \ell_1) = \begin{cases} 1/(m \cdot |Y_i| \cdot |\mathcal{Y} - Y_i|) & \text{if } \ell_0 \notin Y_i \text{ and } \ell_1 \in Y_i \\ 0 & \text{else.} \end{cases}$
For $t = 1, \ldots, T$:

  • Train weak learner using distribution $D_t$.
  • Get weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.
  • Choose $\alpha_t \in \mathbb{R}$.
  • Update:

      $D_{t+1}(i, \ell_0, \ell_1) = \frac{D_t(i, \ell_0, \ell_1) \exp\left(\frac{1}{2}\alpha_t (h_t(x_i, \ell_0) - h_t(x_i, \ell_1))\right)}{Z_t}$

    where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).

Output the final hypothesis:

  $f(x, \ell) = \sum_{t=1}^{T} \alpha_t h_t(x, \ell)$.
```

Figure 2. The algorithm AdaBoost.MR.

With respect to a labeled observation (x, Y), we focus now only on the relative ordering of the *crucial pairs* ℓ_0, ℓ_1 for which $\ell_0 \notin Y$ and $\ell_1 \in Y$. A classification rule f *misorders* a crucial pair ℓ_0, ℓ_1 if $f(x, \ell_1) \leq f(x, \ell_0)$ so that f fails to rank ℓ_1 above ℓ_0. Our goal now is to find a function f with a small number of misorderings so that the labels in Y are ranked above the labels not in Y. Put another way, our goal is to minimize the average fraction of crucial pairs which are misordered, a quantity that we call the empirical *ranking loss*:

$$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{|Y_i|\,|\mathcal{Y} - Y_i|}\,|\{(\ell_0, \ell_1) \in (\mathcal{Y} - Y_i) \times Y_i : f(x, \ell_1) \leq f(x, \ell_0)\}|.$$

(We assume that Y_i is never empty nor equal to all of $\mathcal{Y}$ for any instance. If there are such instances in the training set we can simply discard them since there is no ranking problem to be solved in this case and they do not carry any information.)

AdaBoost.MR is shown in figure 2. We now maintain a distribution D_t over $\{1, \ldots, m\} \times \mathcal{Y} \times \mathcal{Y}$ and denote the weight for instance x_i and the pair ℓ_0, ℓ_1 by $D_t(i, \ell_0, \ell_1)$. This distribution is zero, however, except on the relevant triples (i, ℓ_0, ℓ_1) for which ℓ_0, ℓ_1 is a crucial pair relative to (x_i, Y_i).

As before, weak hypotheses have the form $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$; we think of these as providing a ranking of labels as described above. The update rule is a bit new. Let ℓ_0, ℓ_1 be a crucial pair relative to (x_i, Y_i) (recall that D_t is zero in all other cases). Assuming momentarily that $\alpha_t > 0$, this rule decreases the weight $D_t(i, \ell_0, \ell_1)$ if h_t gives a correct ranking ($h_t(x_i, \ell_1) > h_t(x_i, \ell_0)$), and increases this weight otherwise.

As for the Hamming loss, it can be shown (Schapire & Singer, 1998) that the empirical ranking loss of this algorithm is at most $\prod_{t=1}^{T} Z_t$. Thus, as before, our goal in choosing α_t and h_t should be minimization of

$$Z_t = \sum_{i,\ell_0,\ell_1} D_t(i, \ell_0, \ell_1) \exp\left(\tfrac{1}{2}\alpha(h_t(x_i, \ell_0) - h_t(x_i, \ell_1))\right) \tag{2}$$

We again defer the description of the technique used for this purpose to Section 4.

This algorithm is somewhat inefficient when there are many labels since, naively, we need to maintain $|Y_i| \cdot |\mathcal{Y} - Y_i|$ weights for each training example (x_i, Y_i), and each weight must be updated on each round. Thus, the space complexity and time-per-round complexity can be as bad as $\theta(mk^2)$. In fact, the same algorithm can be implemented using only $O(mk)$ space and time per round. By the nature of the updates, we can show (Schapire & Singer, 1998) that we only need to maintain weights v_t over $\{1, \ldots, m\} \times \mathcal{Y}$. To do this, we maintain the condition that if ℓ_0, ℓ_1 is a crucial pair relative to (x_i, Y_i), then

$$D_t(i, \ell_0, \ell_1) = v_t(i, \ell_0) \cdot v_t(i, \ell_1) \tag{3}$$

at all times. (Recall that D_t is zero for all other triples (i, ℓ_0, ℓ_1).) The pseudocode for this implementation is shown in figure 3. Note that all space requirements and all per-round computations are $O(mk)$, with the possible exception of the call to the weak learner which is discussed in the next section.

4. Weak hypotheses for text categorization

So far, we left unspecified the actual form and implementation of the weak learner, as well as the choice of the parameter α_t. In this section, we describe four implementations of weak learners, three for AdaBoost.MH and one for AdaBoost.MR. Our system for multi-label text categorization, called BoosTexter, can be used with any of the four methods described below.

Boosting is meant to be a general purpose method that can be combined with any classifier, and in practice it has been used, for instance, with decision trees and neural nets. In this paper, however, we focus on the use of boosting with very simple classifiers. Specifically, for all of the methods we use, the weak hypotheses have the same basic form as a one-level decision tree. The test at the root of this tree is a simple check for the presence or absence of a term in the given document. All words and pairs of adjacent words are potential terms.[1] Based only on the outcome of this test, the weak hypothesis outputs predictions and confidences that each label is associated with the document. For example, going back to the news categorization example, a possible term can be *Bill Clinton*, and the corresponding predictor is: “If the term *Bill Clinton* appears in the document then predict that the document belongs to `News` with high confidence, to `Finance` with low confidence, and that it does not belong to `Sports` with high confidence. If, on the other hand, the term does not appear in the document, then predict that it does not belong to any of the classes with low confidence.”

```
Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$ where $x_i \in \mathcal{X}$, $Y_i \subseteq \mathcal{Y}$
Initialize $v_1(i, \ell) = (m \cdot |Y_i| \cdot |\mathcal{Y} - Y_i|)^{-1/2}$
For $t = 1, \ldots, T$:

  • Train weak learner using distribution $D_t$ (as defined by Eq. (3))
  • Get weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.
  • Choose $\alpha_t \in \mathbb{R}$.
  • Update:

      $v_{t+1}(i, \ell) = \frac{v_t(i, \ell) \exp\left(-\frac{1}{2}\alpha_t Y_i[\ell] h_t(x_i, \ell)\right)}{\sqrt{Z_t}}$

    where

      $Z_t = \sum_i \left[ \left( \sum_{\ell \notin Y_i} v_t(i, \ell) \exp\left(\frac{1}{2}\alpha_t h_t(x_i, \ell)\right) \right) \left( \sum_{\ell \in Y_i} v_t(i, \ell) \exp\left(-\frac{1}{2}\alpha_t h_t(x_i, \ell)\right) \right) \right]$

Output the final hypothesis:

  $f(x, \ell) = \sum_{t=1}^{T} \alpha_t h_t(x, \ell).$
```

Figure 3. A more efficient version of AdaBoost.MR: on each round of boosting and for each example, the running time is linear in the number of labels ($O(k)$).

Figure 4 shows the first several weak hypotheses actually found by a version of AdaBoost on one of the datasets tested later in the paper.

Formally, denote a possible term by w, and let us define (abusively) $w \in x$ to mean that w occurs in document x. Based on the term, we will be interested in weak hypotheses h which make predictions of the form:

$$h(x, \ell) = \begin{cases} c_{0\ell} & \text{if } w \notin x \\ c_{1\ell} & \text{if } w \in x \end{cases}$$

where the $c_{j\ell}$'s are real numbers. The three weak learners we describe for AdaBoost.MH differ only with respect to possible restrictions which we place on the values of these numbers.

Our weak learners search all possible terms. For each term, values $c_{j\ell}$ are chosen as described below, and a score is defined for the resulting weak hypothesis. Once all terms have been searched, the weak hypothesis with the lowest score is selected and returned by the weak learner. For AdaBoost.MH, this score will always be an exact calculation of Z_t as defined in Eq. (1) since, as noted in Section 3.1, minimization of Z_t is a reasonable guiding principle in the design of the weak learning algorithm. For AdaBoost.MR, we know of no

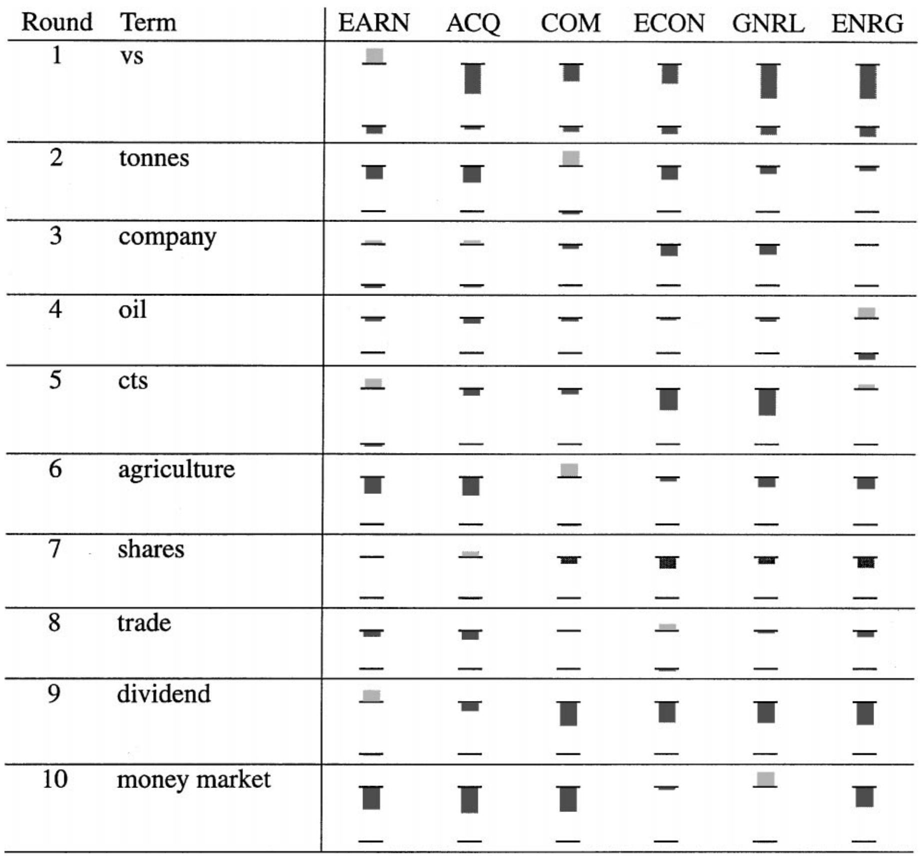

Figure 4. The first ten weak hypotheses found when real AdaBoost.MH (Section 4.1) is run on the entire Reuters-21450 dataset as described in Section 6.5. Each weak hypothesis has the following form and interpretation: if the term associated with the weak hypothesis occurs in the given document, then output the first row of values; otherwise, output the second row of values. Here, each value, represented graphically as a bar, gives the output of the weak hypothesis for one of the classes. For instance, the weak hypothesis found on the first round of boosting tests on the term *vs*. If present, a positive value is output for EARN and negative values are output for all of the other classes. If not present, weakly negative values are output for all classes.

analytical solution for the problem of minimizing Z_t. Instead, an approximation of Z_t is used as described below.

4.1. *AdaBoost.MH with real-valued predictions*

For our first weak learner, we permit unrestricted real-valued predictions $c_{j\ell}$. In our experiments, we call this version *real AdaBoost.MH*.

With minimization of Z_t in mind, the values $c_{j\ell}$ should be calculated as follows for a given term w: Let $X_0 = \{x : w \notin x\}$ and $X_1 = \{x : w \in x\}$. Given the current distribution D_t, we calculate the following for each possible label ℓ, for $j \in \{0, 1\}$, and for $b \in \{-1, +1\}$:

$$W_b^{j\ell} = \sum_{i=1}^{m} D_t(i, \ell)[\![x_i \in X_j \wedge Y_i[\ell] = b]\!]. \tag{4}$$

For readability of notation, we abbreviate the subscripts $+1$ and -1 in $W_{+1}^{j\ell}$ and $W_{-1}^{j\ell}$, writing instead $W_+^{j\ell}$ and $W_-^{j\ell}$. In words, $W_+^{j\ell}$ ($W_-^{j\ell}$) is the weight (with respect to the distribution D_t) of the documents in partition X_j which are (are not) labeled by ℓ.

It can be shown (Schapire & Singer, 1998) that Z_t is minimized for a particular term by choosing

$$c_{j\ell} = \frac{1}{2} \ln \left(\frac{W_+^{j\ell}}{W_-^{j\ell}} \right), \tag{5}$$

and by setting $\alpha_t = 1$. These settings imply that

$$Z_t = 2 \sum_{j \in \{0,1\}} \sum_{\ell \in \mathcal{Y}} \sqrt{W_+^{j\ell} W_-^{j\ell}}. \tag{6}$$

Thus, we choose the term w for which this value of Z_t is smallest.

In fact, it may well happen that $W_-^{j\ell}$ or $W_+^{j\ell}$ is very small or even zero, in which case $c_{j\ell}$ as defined in Eq. (5) will be very large or infinite in magnitude. In practice, such large predictions may cause numerical problems, and there may be theoretical reasons to suspect that large, overly confident predictions will increase the tendency to overfit. To limit the magnitudes of the predictions, in our implementation, we use instead the "smoothed" values

$$c_{j\ell} = \frac{1}{2} \ln \left(\frac{W_+^{j\ell} + \varepsilon}{W_-^{j\ell} + \varepsilon} \right). \tag{7}$$

In our experiments, we set $\varepsilon = 1/mk$. Since both $W_-^{j\ell}$ and $W_+^{j\ell}$ are bounded between 0 and 1, this has the effect of bounding $|c_{j\ell}|$ by roughly $\frac{1}{2}\ln(1/\varepsilon)$.

4.2. *AdaBoost.MH with real-valued predictions and abstaining*

The method described above assigns confidence values both when a term appears in a document and when it does not. Thus, it employs a tacit assumption that the absence of a term carries information about the possible classes a document may belong to. However, given our intuitive knowledge about the problem, we may wish to reject this assumption and force the weak hypothesis to abstain whenever the given term does not appear in a document. This can be accomplished simply by forcing each weak hypothesis to output a confidence value of zero for documents which do not contain the given term. In our experiments, we call this version *real abstaining AdaBoost.MH.*

For a given term w, this weak learner chooses predictions $c_{1\ell}$ for documents which contain w exactly as before. (In our implementation, we also smooth these values as before.) For the rest of the documents, the prediction values $c_{0\ell}$ are all set to zero. Hence, the term w has no influence on the classification if it does not appear in the document. As before, α_t is set to 1.

Let

$$W_0 = \sum_{i:x_i \in X_0} D_t(i, \ell)$$

be the weight of all the document that do *not* contain w. Then it can be shown (Schapire & Singer, 1998) that

$$Z_t = W_0 + 2 \sum_{\ell \in \mathcal{Y}} \sqrt{W_+^{1\ell} W_-^{1\ell}}, \tag{8}$$

and, as before, on each round we choose a term w for which the value Z_t is smallest.

One advantage of this weak learner over the first one is an improvement in the running time as we need to consider only the documents that include a given term w when computing Z_t. Since, typically, the number of documents that include a non-trivial term is only a small fraction of the training data, this version is in practice 15% faster than the previous one. Furthermore, in most of the experiments described in Section 6, the performance of the two versions is comparable.

4.3. AdaBoost.MH with discrete predictions

The next weak learner forces the predictions $c_{j\ell}$ of the weak hypotheses to be either $+1$ or -1. This is the more standard setting in which predictions do not carry confidences. We call this version *discrete AdaBoost.MH*.

With this restriction on the range of the weak hypotheses, we can still minimize Z_t for a given term w using the following method. With the same notation defined in Section 4.1, we set

$$c_{j\ell} = \operatorname{sign}\left(W_+^{j\ell} - W_-^{j\ell}\right)$$

which can be viewed as a (weighted) majority vote over examples in block X_j for each label ℓ. Let

$$r_t = \sum_{j \in \{0,1\}} \sum_{\ell \in \mathcal{Y}} \left| W_+^{j\ell} - W_-^{j\ell} \right|. \tag{9}$$

Then it can be shown (Schapire & Singer, 1998) that, for the purposes of minimizing Z_t, we should choose

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 + r_t}{1 - r_t}\right)$$

giving

$$Z_t = \sqrt{1 - r_t^2}.$$

4.4. AdaBoost.MR with discrete predictions

We next describe a weak learner for AdaBoost.MR. As noted in Section 3.2, we would like to minimize Z_t as defined in Eq. (2). Unfortunately, the exact minimization of this quantity is not as straightforward as it was for AdaBoost.MH. We therefore only consider discrete predictions in $\{-1, +1\}$, and we also use an approximation for Z_t as a score, rather than an exact computation. We call this *discrete AdaBoost.MR*.

For a given hypothesis h_t, let

$$r_t = \frac{1}{2} \sum_{i,\ell_0,\ell_1} D_t(i, \ell_0, \ell_1)(h(x_i, \ell_1) - h(x_i, \ell_0)).$$

Then, similar to the analysis for discrete AdaBoost.MH, it can be shown that $Z_t \leq \sqrt{1 - r_t^2}$ if we choose

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 + r_t}{1 - r_t}\right). \tag{10}$$

Since we do not know how to efficiently minimize Z_t exactly, we instead find a weak hypothesis which minimizes the upper bound $\sqrt{1 - r_t^2}$. We use this upper bound as our score in choosing the best weak hypothesis.

For efficiency, it is important to note that the quantity r_t can be computed efficiently in terms of the weights v_t (defined in Eq. (3)). Let

$$d_t(i, \ell) = \frac{1}{2} v_t(i, \ell) \sum_{\ell' :\, Y_i[\ell'] \neq Y_i[\ell]} v_t(i, \ell').$$

Then it can be shown (Schapire & Singer, 1998) that

$$r_t = \sum_{i,\ell} d_t(i, \ell)\, Y_i[\ell]\, h(x_i, \ell).$$

Thus, for a particular term w, we should choose

$$c_{j\ell} = \text{sign}\left(\sum_{i : x_i \in X_j} d_t(i, \ell)\, Y_i[\ell] \right)$$

which gives

Table 1. Summary of the properties of the four weak learners for multiclass multi-label text categorization.

Version	Loss	Prediction	α_t
Real MH	Hamming	$c_{j\ell} = \frac{1}{2}\ln\left(\frac{W_+^{j\ell}}{W_-^{j\ell}}\right)$ $(j \in \{0,1\})$	1
Real & abstaining MH	Hamming	$c_{0\ell} = 0$ $c_{1\ell} = \frac{1}{2}\ln\left(\frac{W_+^{1\ell}}{W_-^{1\ell}}\right)$	1
Discrete MH	Hamming	$c_{j\ell} = \mathrm{sign}\left(W_+^{j\ell} - W_-^{j\ell}\right)$	$\frac{1}{2}\ln\left(\frac{1+r_t}{1-r_t}\right)$ (r_t defined in Eq. (9))
Discrete MR	Ranking	$c_{j\ell} = \mathrm{sign}\left(\sum_{i:x_i \in X_j} d_t(i,\ell)\, Y_i[\ell]\right)$	$\frac{1}{2}\ln\left(\frac{1+r_t}{1-r_t}\right)$ (r_t defined in Eq. (11))

$$r_t = \sum_{j\in\{0,1\}} \sum_{\ell\in\mathcal{Y}} \left| \sum_{i:x_i\in X_j} d_t(i,\ell)\, Y_i[\ell] \right|. \tag{11}$$

We thus choose the term w which maximizes this quantity, and we assign predictions correspondingly. The parameter α_t is set as in Eq. (10).

The search for a good weak hypothesis can be very time consuming when the training corpus is large. We therefore use an inverted list that stores for each term (word, bigram, sparse n-gram, etc.) the list of documents in which it appears. On each round, when searching for a good weak hypothesis, we scan the inverted list and for each term we evaluate its prediction confidences $c_{j\ell}$ according to the version of AdaBoost that we use. A straightforward implementation would require scanning the entire collection for each term. However, precomputing certain values can save a significant amount of time. For AdaBoost.MH for instance, we first compute on each round once for all j the following values.

$$W^{j\ell} = \sum_{i:x_i\in X_j} D_t(i,\ell).$$

We now find for each term the values $W_+^{j\ell}$ by summing over the documents in which each term appears using the inverted list. We then set $W_-^{j\ell} = W^{j\ell} - W_+^{j\ell}$, and proceed to find $c_{j\ell}$ and the corresponding values for Z_t. Hence, the amount of time spent on each round searching for a weak hypothesis is proportional to the total number of occurrences of all the terms in the training collection. After a weak hypothesis is found, it takes $O(mk)$ time to update the distribution $D_t(i,\ell)$.

Our system for multi-label text categorization, called BoosTexter, can be used with any of the four implementations of weak learners described above. A brief summary of the different implementations is given in Table 1.

5. Evaluation measures

For evaluating the performance of our boosting algorithms we used three evaluation measures. The first one, one-error, is a simple generalization of classification error for multiclass multi-label problems. The one-error is also directly related to the training error (Schapire & Singer, 1998). The other two evaluation measures are based on measures used in information retrieval and used to evaluate the performance of the various classification algorithms in terms of their label rankings.

As noted earlier, we assume that a multi-label system induces an ordering of the possible labels for a given instance. That is, the output of the learning system is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ which ranks labels according to $f(x, \cdot)$ so that label ℓ_1 is considered to be ranked higher than ℓ_2 if $f(x, \ell_1) > f(x, \ell_2)$. With the exception of RIPPER, all the classification systems we tested in this paper can indeed be viewed in this way, where the ordering is defined by assigning a real number for each possible instance-label pair x, ℓ.

We will find it convenient to refer to the *rank* of a given label ℓ for instance x under f which we denote by $\text{rank}_f(x, \ell)$. That is, formally, $\text{rank}_f(x, \cdot)$ is a one-to-one mapping onto $\{1, \ldots, k\}$ such that if $f(x, \ell_1) > f(x, \ell_2)$ then $\text{rank}_f(x, \ell_1) < \text{rank}_f(x, \ell_2)$.

One-error. This measure evaluates how many times the top-ranked label was *not* in the set of possible labels. Thus, if the goal of a multiclass system is to assign a single label to a document, the one-error measures how many times the predicted label was not in Y. We call this measure the one-error of hypothesis H since it measures the probability of not getting even one of the labels correct. We denote the one-error of a hypothesis f by one-err(f). We can define a classifier $H : \mathcal{X} \rightarrow \mathcal{Y}$ that assigns a single label for a document x by setting $H(x) = \arg\max_{\ell \in \mathcal{Y}} f(x, y)$. Then, for a set of labeled documents $S = \langle (x_1, Y_1), \ldots, (x_m, Y_m) \rangle$, the one-error is

$$\text{one-err}_S(H) = \frac{1}{m} \sum_{i=1}^{m} [\![H(x_i) \notin Y_i]\!].$$

Note that, for single-label classification problems, the one-error is identical to ordinary error.

Coverage. While the one-error evaluates the performance of a system for the top-ranked label, the goal of the coverage measure is to assess the performance of a system for all the possible labels of documents. That is, coverage measures how far we need, on the average, to go down the list of labels in order to cover all the possible labels assigned to a document. Coverage is loosely related to precision at the level of perfect recall. Formally, we define the coverage of f with respect to $S = \langle (x_1, Y_1), \ldots, (x_m, Y_m) \rangle$ to be

$$\text{coverage}_S(H) = \frac{1}{m} \sum_{i=1}^{m} \max_{\ell \in Y_i} \text{rank}_f(x_i, \ell) - 1.$$

For single-label classification problems, coverage is the average rank of the correct label, and is zero if the system does not make any classification errors.

Average precision. The above measures are not complete for multi-label classification problems: We can achieve good (low) coverage but suffer high one-error rates, and vice versa. In order to assess the label ranking of a multiclass system as a whole we used the *non-interpolated* average precision, a performance measure frequently used for evaluation of information retrieval (IR) systems (Salton, 1991). Note, however, that non-interpolated average precision is typically used in IR systems to evaluate the *document* ranking performance for query retrieval. In contrast, in our experiments we use average precision for evaluating the effectiveness of the *label* rankings. Formally, we define average-precision for a ranking H with respect to a training set S, denoted avgprec ( ) for short, to be

$$\text{avgprec}_S(H) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{|Y_i|}\sum_{\ell\in Y_i}\frac{|\{\ell' \in Y_i \mid \text{rank}_f(x_i,\ell') \leq \text{rank}_f(x_i,\ell)\}|}{\text{rank}_f(x,\ell)}.$$

In words, this measure evaluates the average fraction of labels ranked above a particular label $\ell \in Y_i$ which actually are in Y_i. Note that $\text{avgprec}_S(f) = 1$ for a system f which ranks perfectly the labels for all documents so that there is no document x_i for which a label not in Y_i is ranked higher than a label in Y_i.

6. Text categorization experiments

In this section, we describe and analyze the experiments we performed using the four boosting algorithms for text categorization that were described in previous sections. The experiments were performed on an SGI Challenge with 20 MIPS R10000 processors running at 195 MHz. The timing information we give in this section is with respect to a single cpu.

6.1. Test corpora

Reuters-21450. The documents in this collection were collected from Reuters newswire in 1987. We used the modified Apte ("ModApte") split which contains 12,902 documents. A cleaned-up version of this dataset, called Reuters-21578, is publicly available from the web page `http://www.research.att.com/~lewis` by David Lewis, who originally compiled the collection. We performed the following pre-processing prior to the experiments: All words were converted to lower case, punctuation marks were removed, and "function words" from a standard stop-list were removed.[2] The average length of a document after pre-processing is 82 words. This corpus is divided into categories which in turn are sub-divided into sub-categories. The Reuters corpus has served as the benchmark for many text-categorization studies using various partitions of the corpus. See Yang's work (to appear) for an overview of the more common partitions and versions of this corpus as well as a summary of the text categorization algorithms that tested on this corpus. In this work, we considered several partitions of the Reuters corpus based on the broad topics at the top hierarchy (for further details see Tables A.1, A.7, and A.9). We used 3-fold cross validation in our experiments with these partitions. To compare our algorithm to previously published work, we also performed experiments with a partition that includes all topics in Reuters that have at least two relevant documents for training. This collection includes 93 topics and was studied extensively by Yang (to appear) and others. Yang referred to this partition as version-3 and compared the results to previously studied text-categorization algorithms. We devote a separate section, Section 6.5, to the description of our experiment with this widely tested partition of Reuters.

AP Titles. This is a corpus of AP newswire headlines (Lewis & Gale, 1994; Lewis & Catlett, 1994). As for the Reuters corpus, previous work concentrated on binary classification by tagging documents as being relevant or irrelevant to topics like “federal budget” and “Nielsens ratings.” The total number of documents in this corpus is 319,463. The headlines are an average of nine words long, with a total vocabulary of 67,331 words. No preprocessing of the text was done, other than to convert all words to lower case and remove punctuation marks. We performed two sets of experiments with this corpus based on two different labeling schemes available for this corpus.

UseNet data. This dataset consists of Usenet articles collected by Lang (1995) from 20 different newsgroups. One thousand articles were collected for each newsgroup so there are $20{,}000$ articles in the entire collection. This data was originally treated as single-labeled (see for instance Joachims (1997)). However, since people tend to post articles to multiple newsgroups, we found after examining the headers of the articles that about 4.5% of the articles are actually multi-labeled. Furthermore, we found 544 identical articles which were posted to more than one group. The total number of articles after relabeling the data based on the headers is $19{,}466$ with $20{,}347$ labels. Further description of this dataset is given in Table A.11. We used 3-fold cross validation in our experiments with the newsgroup data.

6.2. Other algorithms

As mentioned in the introduction, there has been immense work on text categorization using many different algorithms. Since it is impossible to implement and evaluate all previously published algorithms, we chose the following algorithms for comparison with the boosting algorithms:

RIPPER. This is Cohen’s (1995) rule-learning system as adapted to text categorization problems by Cohen and Singer (1996). RIPPER classifies a document by applying a set of boolean tests that check the absence (or presence) of words in the documents. RIPPER is not capable of dealing with multiple labels. RIPPER learns a classifier in the form of a boolean combination of simple terms. It does not provide a ranking of the possible labels for a given document. Therefore, the only performance measure we can use for comparison is the error rate.

Rocchio. We implemented a version of Rocchio’s algorithm (Rocchio, 1971), as adapted to text categorization by Ittner, Lewis, & Ahn (1995) and modified to multiclass problems. In Rocchio, we represent the data (both training and test documents) as vectors of numeric weights. The weight vector for the ith document is $\mathbf{v}^i = (v_1^i, v_2^i, \ldots, v_l^i)$, where l is the number of indexing terms used. We use single words as terms. We followed the TF-IDF weighting (Salton, 1991) and defined the weight v_k^i to be:

$$v_k^i = \frac{f_k^i \log(N_D/n_k)}{\sum_{j=1}^{l} f_j^i \log(N_D/n_j)},$$

Here, N_D is the number of documents, n_k is the number of documents in which the indexing term k appears. The weight f_k^i is $\log(m) + 1$, where m is the number of occurrences of the indexing term k in document i. We set $f_k^i = 0$ if $m = 0$. For each class ℓ we build a "prototype" vector which is the average weight vector over all documents x_i for which $\ell \in \mathcal{Y}_i$. Formally, let $X(\ell) = \{i \mid \ell \in \mathcal{Y}_i\}$. Then the prototype vector for class ℓ is

$$\frac{1}{|X(\ell)|} \sum_{i \in X(\ell)} \mathbf{v}^i.$$

Test documents are classified by calculating the dot-products between the weight vector representing the document and each of the prototype vectors. These dot-products induce a ranking of the possible labels. We use this ranking to evaluate the performance of the classifier for each of the measures discussed in Section 5.

Sleeping-experts. This is an algorithm originally proposed by Blum (1997), studied further by Freund et al. (1997), and first applied to text categorization by Cohen and Singer (1996). Briefly, this algorithm classifies a document by thresholding a score which is a weighted combination of "experts" which are based on the word-grams appearing in the text. This score can be used to rank the labels. The algorithm can be easily adapted to multiclass (and multi-label) settings by assigning mini-experts for each possible pair of class and sparse word-gram. We used words and word pairs as the set of experts in the experiments.

Naive-Bayes and probabilistic TF-IDF. These are probabilistic classifiers that assign, for each document, a probability vector of belonging to each of the possible labels. Like the algorithms above, these probability vectors can be viewed as rankings and thus used for evaluating the performance with respect to the measures discussed in Section 5. These algorithms are available as part of the publicly available *Rainbow* text-categorization system[3] which we used in our experiments. This system includes other classification methods but, in all of the experiments we performed, Naive-Bayes and probabilistic TF-IDF performed better than the other methods available in Rainbow. Further description of Naive-Bayes and probabilistic TF-IDF for text categorization is given in (Mitchell, 1997; Joachims, 1997). To handle multi-label data, we mapped to the single-label case by simply repeating each document once for each of its assigned labels.

6.3. *Experiments using single-label corpora*

In the first set of experiments, we partitioned the Reuters corpus into six disjoint classes. These classes roughly constitute the top categorization hierarchy. We discarded articles that do not belong to any of the classes and articles that belong to more than one class. A detailed description of this subset is given in Table A.1. The total number of articles in this experiment is $10{,}187$. We used three-fold cross-validation in the experiments. The results we report are averaged over the three folds. For all subsets of this dataset, we ran the real AdaBoost.MH and real-abstaining AdaBoost.MH for $5{,}000$ rounds and the discrete AdaBoost.MH and AdaBoost.MR for $20{,}000$.

We performed experiments with varying numbers of classes. We selected subsets of the data by taking the top k classes, in decreasing number of documents, from $k = 3$ to 6. For instance, for $k = 3$ we took 7,761 documents from the classes EARN, ACQ, and COM. We then created three different splits into training and test data and ran the various text categorization algorithms on each of the splits.

A summary of the results of the experiments with this dataset is given in Table A.2 and graphically in figure 5. The performance of the different multiclass versions of AdaBoost is comparable on this data set, with a small advantage to the real-valued versions of AdaBoost.MH (with and without abstaining). All the four versions of AdaBoost for multi-label problems clearly outperform all of the other classification algorithms. The error of AdaBoost.MH is almost 50% smaller than the error-rate of the best competing algorithm on this dataset (Naive-Bayes). Similar behavior is observed for coverage and average-precision.

The next set of experiments with single-label datasets is with the AP Titles corpus. In the first subset of AP titles, each headline is (possibly) labeled by a single topic from 20 possible classes. We extracted $29{,}841$ documents which belong to exactly one of the 20 classes. A description of this subset of AP titles is given in Table A.3. For the subsets in this dataset, we ran the real-valued version of AdaBoost.MH (with and without abstaining) for $10{,}000$ rounds and the discrete AdaBoost.MH and AdaBoost.MR for $40{,}000$ rounds.

As before, we tested the performance of the algorithms by extracting subsets with growing numbers of classes, where we ordered the classes by decreasing number of documents in each class. The results are summarized in Table A.4 and graphically in figure 6. Among the different boosting algorithms, real AdaBoost.MH exhibits the best performance: it is slightly better than real abstaining AdaBoost.MH and significantly better than the discrete AdaBoost.MH and discrete AdaBoost.MR where the latter is the worst performer among the four boosting algorithms. The main reason for this is that $40{,}000$ rounds were simply not enough for the discrete versions. For discrete AdaBoost.MR and AdaBoost.MH, the training error was still monotonically decreasing when we reached the maximal number of rounds. This improved performance in decreasing the training error of the real-valued versions of AdaBoost is even more vivid for large datasets, as we show subsequently.

The best competitor algorithm for this dataset is Sleeping-experts. In fact, Sleeping-experts slightly outperforms AdaBoost.MH when the number of classes is three. However, for subsets of at least eight classes, AdaBoost.MH significantly outperform Sleeping-experts with respect to all three performance measures. Note also the interesting fact that, in contrast to the results for the previous dataset, probabilistic TF-IDF outperforms Naive-Bayes, yet both algorithms are clearly inferior to AdaBoost.MH.

The last set of experiments with single-labeled multiclass problems is with the *entire* AP titles collection. In addition to the partial partition into twenty specific topics above, this corpus is also divided into six general categories[4] such that each article falls into exactly

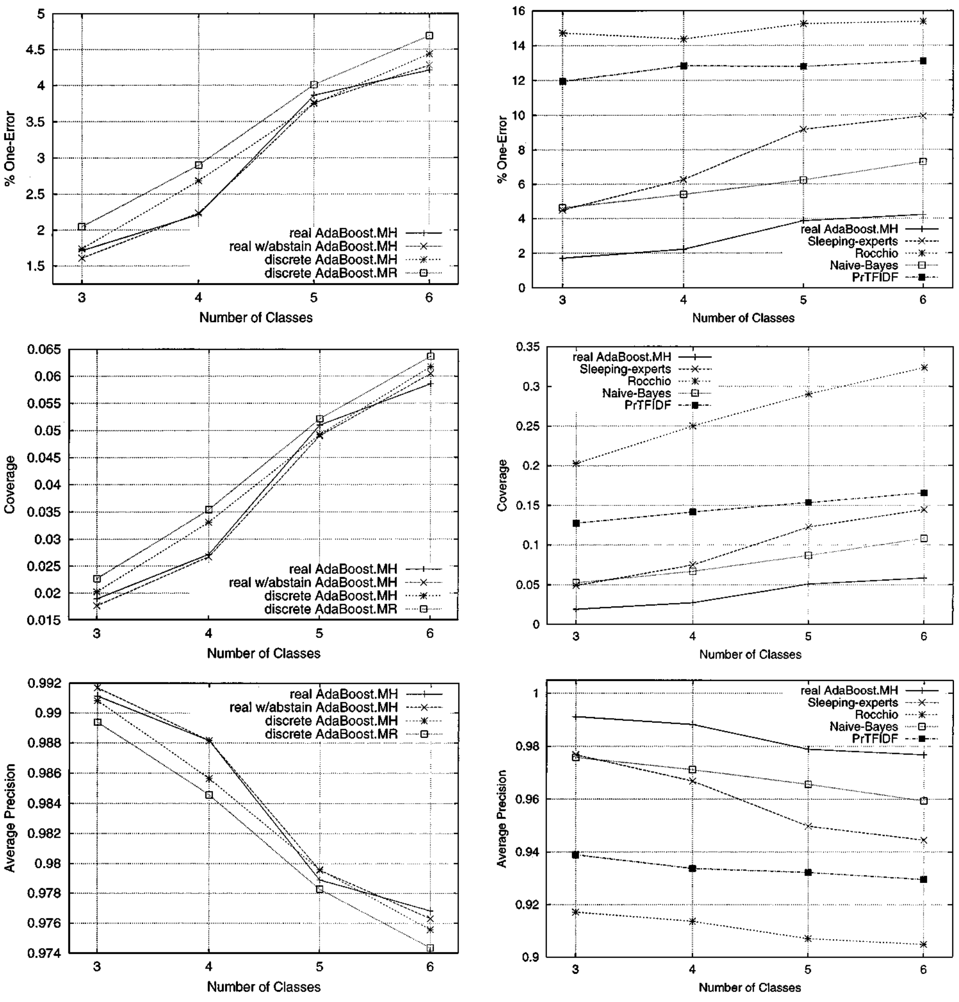

Figure 5. Left: Comparison of the various boosting algorithms for text categorization on the first single-label subset of Reuters-21450. Right: Comparison of real AdaBoost.MH with Naive-Bayes, probabilistic TF-IDF, Sleeping-experts, and Rocchio on the same subset.

one category. We removed all articles not belonging to any of the categories. The number of articles that remained is 209,700. Since this labeling scheme results in a very large corpus, we did not use cross-validation in the experiments. Instead, we used Lewis's chronological split into training and test sets. The training set for this split contains 142,727 headlines and the test set 66,973. A description of the classes is given in Table A.5 and a summary of the results is given in Table A.6.

Since Rainbow allocates a different file for each article, this dataset was too large to be converted into the format required for Rainbow. We therefore compared real AdaBoost.MH,

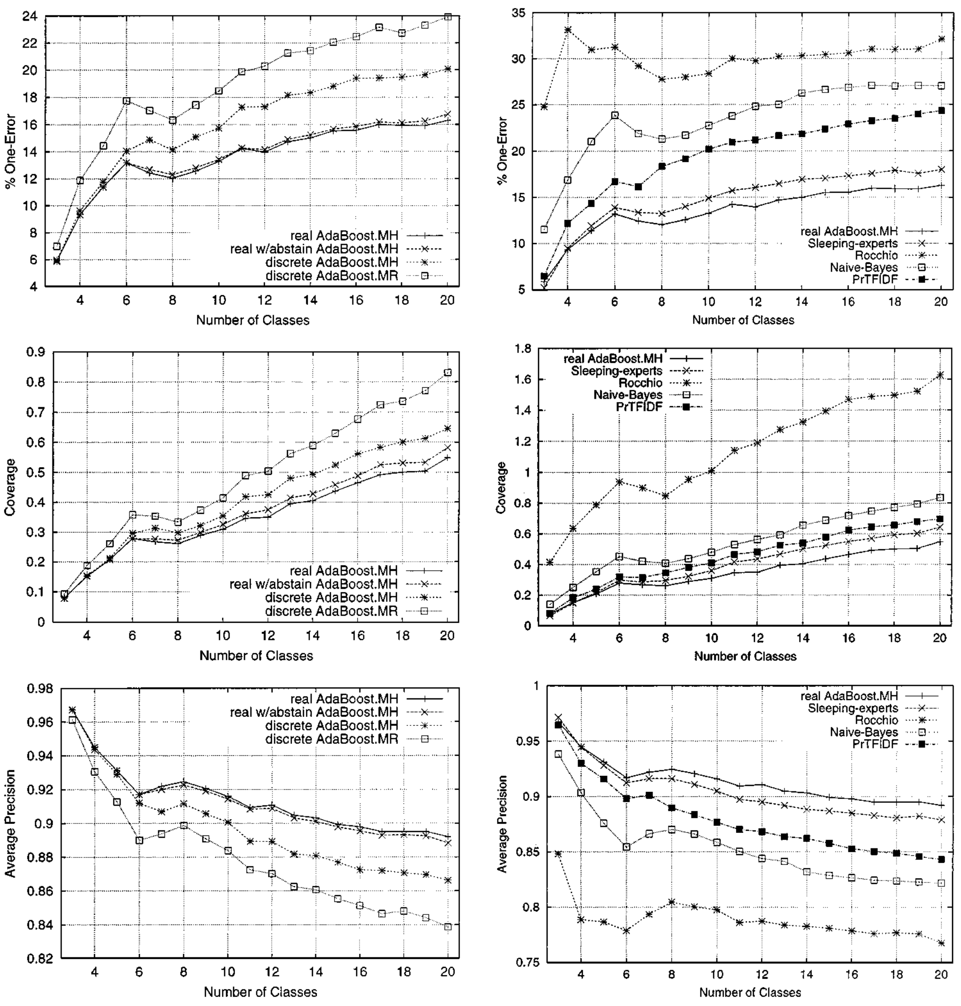

Figure 6. Left: Comparison of the various boosting algorithms for text categorization on the first single-label subset of AP titles. Right: Comparison of real AdaBoost.MH with Naive-Bayes probabilistic TF-IDF Sleeping-experts and Rocchio on the same dataset.

discrete AdaBoost.MH, and discrete AdaBoost.MR only with Sleeping-experts, Rocchio, and RIPPER.

Our main focus in the experiment with this dataset was the performance of the different boosting algorithms as a function of number of rounds. In figure 7, we show the training and test error of the algorithms as a function of the number of rounds. We see that the version of AdaBoost.MH which uses real-valued predictions dramatically outperforms the methods with predictions in $\{-1, +1\}$. After 180,000 rounds, discrete AdaBoost.MH reaches a training error of 32.2% while it took real AdaBoost.MH only 642 rounds to reach this training error—more than a two-hundred fold speed-up!

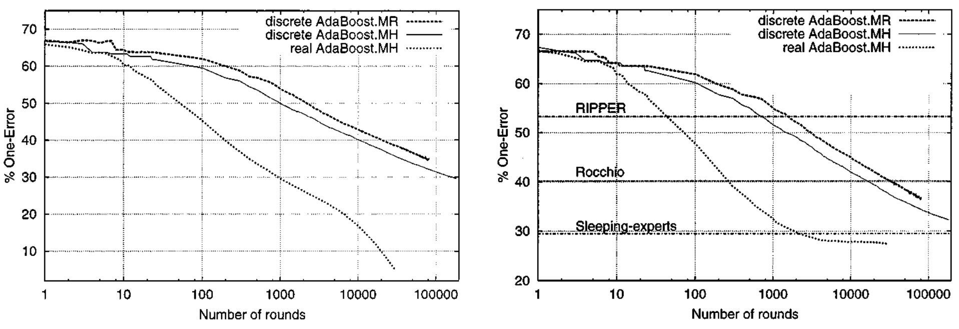

Figure 7. Comparison of the training (left) and test (right) error using three boosting methods on the second single-label subset of AP titles.

As with the previous experiments, discrete AdaBoost.MH seems to consistently outperform discrete AdaBoost.MR. This might be partially due to the approximation that is made of Z_t in lieu of its direct minimization. We fortunately do not observe overfitting with the AdaBoost algorithms so that the better performance in decreasing the training error results in lower error rates on the test data.

The best competitor algorithm for this dataset is Sleeping-experts. It takes about a thousand rounds for AdaBoost.MH to reach the test error rate of Sleeping-experts and after 30,000 rounds its test error is significantly lower. However, Sleeping-experts is much faster on this dataset, finishing in about a minute, roughly as long as it takes to run boosting for 25 rounds.

6.4. *Experiments using multi-label corpora*

For the first set of experiments with multi-labeled corpora, we used the Reuters dataset again. This time, we partitioned it into classes based on the nine topics constituting the top hierarchy. We discarded documents not belonging to any topic; however, articles belonging to more than one topic were assigned multiple labels. The total number of articles for this partition is 10,792 and the number of different labels is 11,588; about 7% of the articles are labeled with more than one label. We performed experiments by selecting a subset of the classes and the corresponding articles. The subsets were again selected by choosing the k classes with the largest number of articles for $k = 3, \ldots, 9$. Thus, once again, the difficulty of the classification problem increases with k. A description of the dataset is given in Table A.7. In all of the experiments with this data, we used three-fold cross validation. We ran the versions with real-valued prediction for 10,000 rounds and the discrete versions 40,000 rounds.

A summary of the results, averaged over the three folds, is given in Table A.7 and figure 8. The results for this multi-label dataset are similar to the previous single-label datasets. The different boosting methods are comparable in performance. AdaBoost.MR is slightly worse than the other three for one-error and average-precision. Real AdaBoost.MH again outperforms all the competitor algorithms with respect to the three performance evaluation

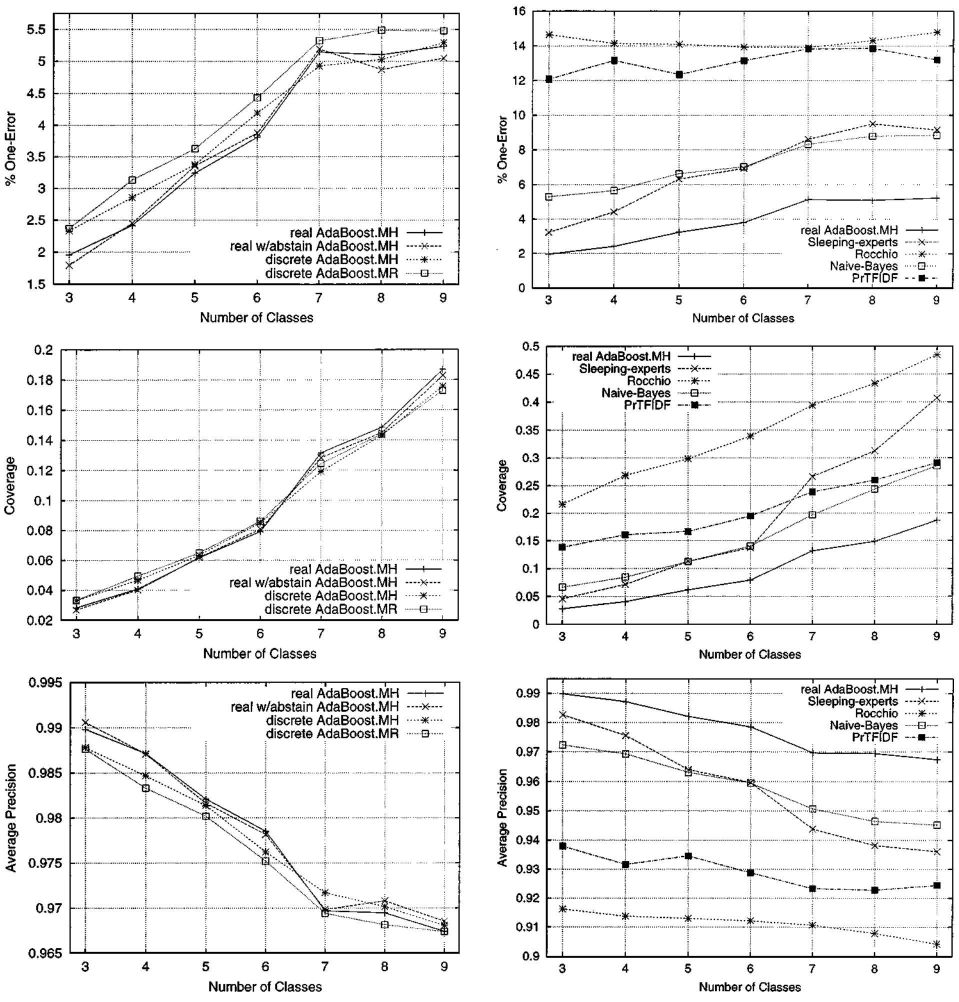

Figure 8. Left: Comparison of the various boosting algorithms for text categorization on the first multi-label subset of Reuters-21450. Right: Comparison of real AdaBoost.MH with Naive-Bayes, probabilistic TF-IDF, Sleeping-experts, and Rocchio (right) on the same dataset.

measures. Furthermore, there is no clear winner among the other algorithms: while Sleeping-experts is best for the subsets with a small number of classes ($k < 6$), Naive-Bayes is the best one for the large classification problems ($k > 6$). Nonetheless, AdaBoost.MH clearly outperforms both methods on all subsets.

In the second set of multi-label experiments with Reuters, we partitioned the dataset into the classes constituting the leaves of the hierarchy of topics. We chose all classes which include at least 100 articles. This subset includes 19 different classes which sum to 3,631 documents labeled by 5,173 different labels. In the full subset with 19 classes about 40% of

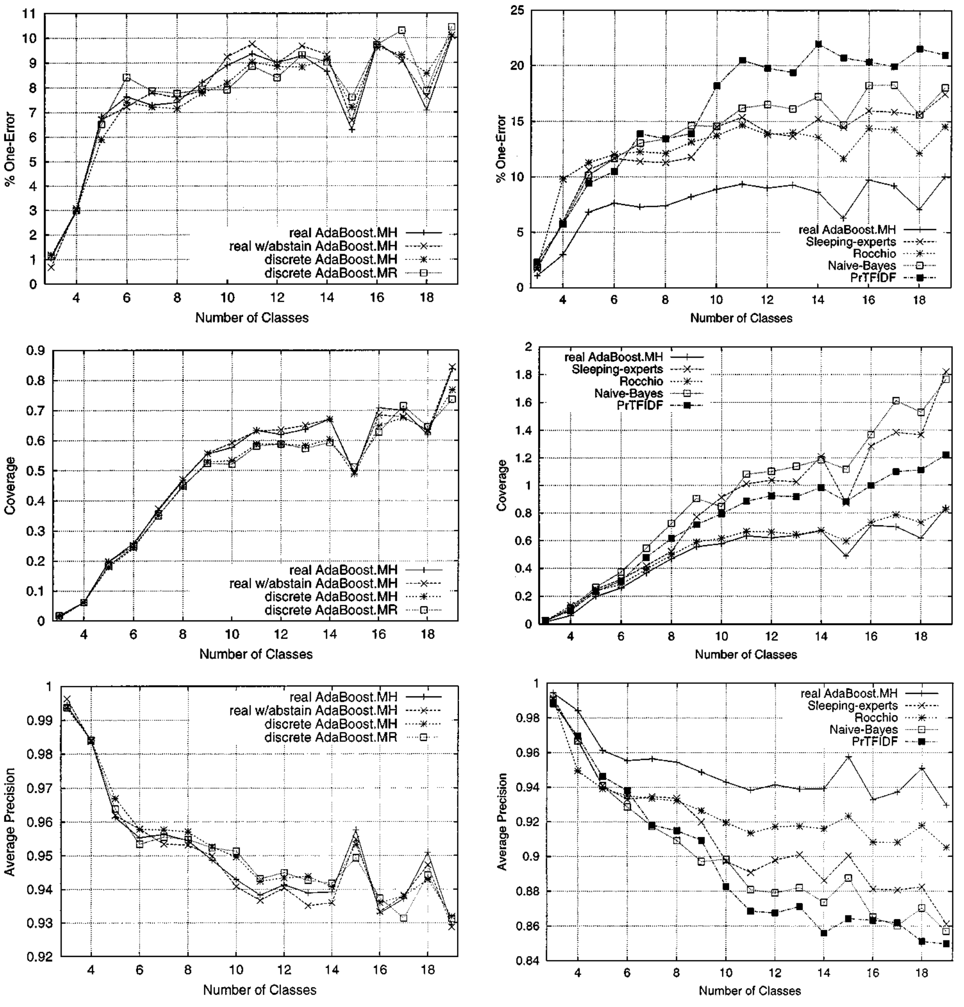

Figure 9. Left: Comparison of the various boosting algorithms for text categorization on the second multi-label subset of Reuters-21450. Right: Comparison of real AdaBoost.MH with Naive-Bayes probabilistic TF-IDF Sleeping-experts and Rocchio on the same dataset.

the articles have more than one label. As before, we performed experiments with subsets of growing size and classes, for $k = 3, \ldots, 19$. A detailed description of the dataset is given in Table A.9. As before we used 3-fold cross-validation to estimate the performance. Again, we ran the real-valued version for 10,000 rounds and the discrete for 40,000.

A summary of the results is given in Table A.10 and figure 9. Here again we see comparable performance of the different boosting algorithms. Also, real AdaBoost.MH is better than all competitor algorithms, especially with respect to one-error and average-precision. For this dataset, Rocchio seems to be the best alternative. In fact, it achieves coverage values which are comparable to real AdaBoost.MH on most, if not all, of the subsets.

The last experiment with multi-labeled text data was performed with newsgroup articles. Here we followed the experimental methodology used in previous studies with this dataset. We used 3-fold cross validation. For each fold we held the test set fixed and varied the size of the training set by sub-sampling the full training set for each fold. We ran the different algorithms for training sets of size 200, 500, 1000, 2000, 5000, 10000, and 12,977 (two thirds of the total number of articles available for this dataset). We compared real AdaBoost.MH with the two methods which in previous studies achieved the best results on this dataset, namely, Naive-Bayes and probabilistic TF-IDF. We allowed weak hypotheses (and features for Naive-Bayes and probabilistic TF-IDF) of single words and word pairs. We set the number of rounds for AdaBoost.MH to be twice the number of training documents. Hence, we ran AdaBoost.MH as little as 400 rounds and at most 26,000 rounds.

The results comparing real AdaBoost.MH, probabilistic TF-IDF and Naive-Bayes for the three evaluation measures as a function of the training set size are shown in figure 10.

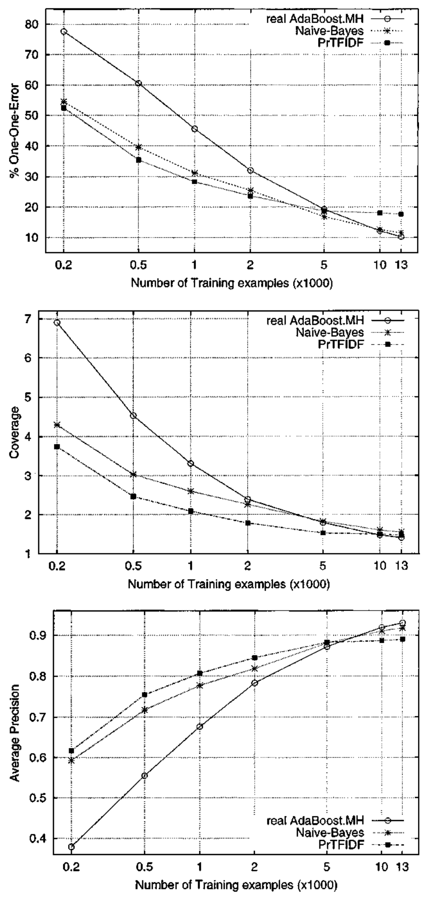

Figure 10. Comparison of real AdaBoost.MH Naive-Bayes and probabilistic TF-IDF as a function of the number of training examples on the UseNet data.

For training sets of size smaller than 10,000, real AdaBoost.MH is clearly inferior to probabilistic TF-IDF and Naive-Bayes. The performance of AdaBoost.MH is especially poor for training sets of size smaller than a thousand. When the training set is large enough, we again see that AdaBoost.MH outperforms both probabilistic TF-IDF and Naive-Bayes with respect to all three measures. However, the difference in performance is not as significant as in the previous datasets. One possible explanation for these results is that, in contrast to probabilistic TF-IDF and Naive-Bayes, AdaBoost.MH incorporates very little prior knowledge. Thus, although AdaBoost.MH is minimizing the Hamming loss on the training set, the generalization error is rather poor, as indeed implied by theoretical studies. Once there are enough examples, the prior knowledge, incorporated via the term weights in probabilistic TF-IDF and Naive-Bayes, is much less crucial and AdaBoost.MH does a better job in driving the training error down, and therefore also the generalization error decreases. These results suggest that the new boosting algorithms for text categorization would be best utilized in complex multiclass problems with a large number of training examples.

6.5. An experiment with a large number of classes

We conclude this section on text categorization experiments with a multi-label categorization experiment using a dataset that contains a large number of classes. In this experiment we used the partition of Reuters-21450 that was prepared by Apté, Damerau, and Weiss (1994) for their experiments with the SWAP-1 rule learner. The Reuters-21450 corpus was partitioned into a training set of 7,789 documents and a test set containing 3,309 document. This partition includes all classes with at least two documents in the training set and at least one document in the test set. There are 93 such classes. Yang (to appear), who refers to this partition of Reuters as version-3, performed extensive comparisons with various algorithms that were evaluated on this partition. Here we compare AdaBoost.MH with the two classification algorithms that achieved the best performance results according to Yang; these are a k-nearest-neighbor (KNN) classifier and a linear classifier based on a least squares fit of term weights to the class labels (LLSF).

We processed the text as described in Section 6.1. Each document was labeled with a subset of the 93 possible classes. The average number of labels per document is 1.24. This problem requires a vast amount of memory; to maintain the distribution $D_t(i, j)$, we need a table of size $mk = 7789 \times 93$ which amounts to over 700,000 numbers. As in previous experiments, we ran AdaBoost.MH for 10,000 rounds which took about 3 days of cpu time to complete. The smoothing value ε was set using 3-fold cross validation on the training set. We would like to note however that using the default value yielded only slightly worse results. (For instance, the 11-point average precision is 0.932 when using the default value for ε compared to 0.934 when using cross-validation to determine ε.) On the left hand-side of figure 11 we plot the one-error on the training and test data as a function of the number of rounds. Note that the one-error on the training set reaches its minimal value, which is very close to zero, after about 1000 rounds of boosting while the test error continues to decrease even after 10,000 rounds, apparently without overfitting. This behavior was also observed in other experiments with boosting algorithms and is partially motivated by theoretical analysis (Schapire et al., 1998).

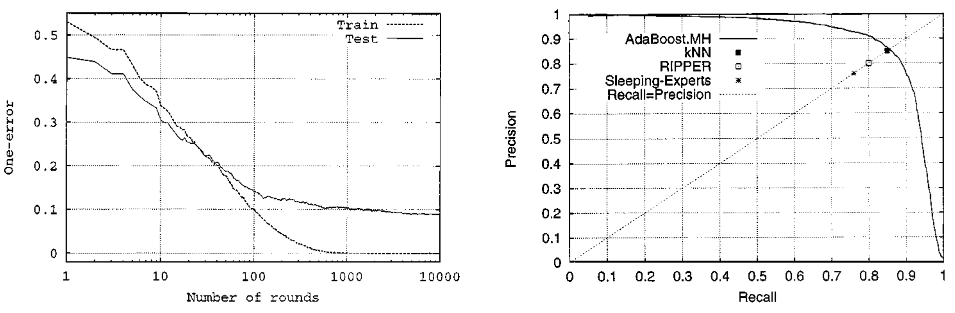

Figure 11. Left: The one-error on the training and test data for the Reuters partition with 93 classes. Right: Precision-Recall curve for AdaBoost.MH on the test collection of the same dataset.

Table 2. Summary of results obtained for Reuters-21450 with 93 classes.

Algorithm	11-pt Avg. precision	(Threshold adjusted on data) F_1	(Threshold = 0) F_1	Mircro-avg. BEP
AdaBoost.MH	0.934	0.853	0.851	0.86
kNN	0.924	0.852	–	0.85
LLSF	0.901	0.855	–	0.85

To make our results on this dataset comparable with previously published results, we used the three evaluation measures that were used by Yang (to appear) and others, namely, 11-point interpolated average precision (Salton & McGill, 1983), F_1 (van Rijsbergen, 1979), and micro-averaged break-even point. The first and the third performance measures asses the general quality of the label ranking while the second measure evaluates the classification quality. For further details on these evaluation measures see Yang (to appear). To use the F_1 measure we need to set a threshold for the label-rankings and decide which labels should be associated with a document. We evaluated the performance using two thresholds: the zero threshold and a threshold that was adjusted so as to maximize F_1 on the training data after AdaBoost.MH completed 10,000 rounds. In Table 2 we summarize the results and compare them to the best results obtained by Yang. (We would like to note parenthetically that in a very recent work, which was brought to out attention during the final preperations of this paper, Weiss et al. (1999) report a break-even point of 87% using the Reuters-21578 dataset.) We also give on the right hand side of figure 2 a precision-recall graph for AdaBoost.MH together with the break-even points of three other classification algorithms evaluated on this dataset. The performance of AdaBoost.MH is state-of-the-art: it achieves the highest 11-point interpolated average precision and break-even point and comes very close to the best F_1 value obtained on this partition of Reuters. However, it is difficult to asses the statistical significance of these results since the performance measures used are highly nonlinear and non-additive. Nonetheless, the good performance of AdaBoost.MH on this well-studied dataset provides further empirical evidence that boosting algorithms can serve as a viable alternative to existing algorithms for text categorization.

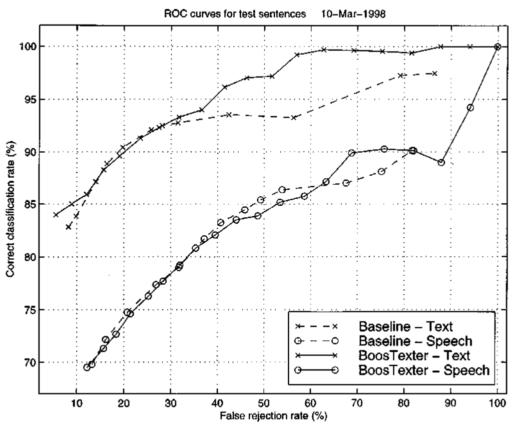

Figure 12. Results on a call-type identification task.

7. Speech categorization experiments

In the final set of experiments, we tested our system on a call-classification task. The purpose of this task is to automatically identify the type of call requested in response to the greeting, “How may I help you?” For instance, if the response is, “Yes, I would like to charge this call to my Visa card,” then the call should be classified as a calling-card call. There are fourteen call types, plus an ‘other’ category. Some calls can be of more than one type (for instance, a call can be both collect and person-to-person).

This task was previously studied by Gorin and others (Gorin, Riccardi, & Wright, 1997; Gorin et al., 1996; Riccardi et al., 1997; Wright, Gorin, & Riccardi, 1997), and we used the same data, namely, a collection of 8,000 training utterances and 1,000 test utterances. Both the training and test utterances were all transcribed by humans from actual spoken responses. The test utterances are also available in a form produced by an automatic speech recognizer; this, of course, is the only form that would be available in a real system.

Following others who have worked on this dataset, we present our results in the form of an ROC curve. For this, each algorithm needs to produce a confidence in its own predictions. The curve is then produced by varying a reject threshold which specifies that examples with confidence below the given threshold should be rejected (for this task, this would mean that the call would have to be handled by a human operator). We then plot the accuracy of the classifier on non-rejected examples as a function of the false rejection rate, which is the fraction of examples incorrectly rejected. A classification by the system of ‘other’ is also considered equivalent to rejection.

To get a confidence level for the predictions of AdaBoost.MH, we used the difference between the final scores of the first and second ranked labels. That is, if f is the final classifier produced by AdaBoost.MH, then the confidence assigned to the prediction of f on a test example x is $f(x, \ell_1) - f(x, \ell_2)$ where ℓ_1 and ℓ_2 are the first and second ranked labels according to $f(x, \cdot)$.

We trained real AdaBoost.MH on this data using 300 rounds of boosting, and allowing sparse word trigrams for the terms used in forming the weak hypotheses. We compared our system to the best previous published work on this dataset, namely, that of Wright, Gorin, and Riccardi (1997). The results are shown in figure 12 as a set of ROC curves. For the top set of curves, the algorithms were tested using human-transcribed test data. For the bottom set of curves, the test data were generated using an automatic speech recognizer (based on the same spoken utterances). The solid curves are for AdaBoost.MH, and the dashed curves are those of Wright, Gorin, and Riccardi (1997).

The performance of the two algorithms is strikingly similar for most reject levels. However, AdaBoost.MH does significantly better on the transcribed data for moderately large reject levels of 40% or more. These results indicate that for slightly less than half of the examples, AdaBoost.MH can produce predictions that are almost certainly correct.

Note that the training set is the same, whether we test on manually transcribed or automatically recognized data. AdaBoost.MH, like other learning algorithms, attempts to minimize the classification error on the training data and thus employs the tacit assumption that the test data are generated by the same source as the training data. This is clearly not true when we use the automatically transcribed data for testing. We believe that we can improve the performance of our system using training data that is automatically generated by a speech recognizer.

Appendix A: Description of text datasets and summary of results

Table A.1. Description of the classes constituting the single-label subset of Reuters-21450.

	Class	#Docs	Cum. #docs
1	Earnings and Earnings Forecasts (EARN)	3923	3923
2	Mergers/Acquisitions (ACQ)	2292	6215
3	Commodity Codes (COM)	1546	7761
4	Economic Indicator Codes (ECON)	997	8758
5	General Articles (GNRL)	871	9629
6	Energy Codes (ENRG)	558	10187

Table A.2. Results for the single-label subset of Reuters-21450 (Table A.1).

	Real AdaBoost.MH			Naive-Bayes			Rocchio			Sleeping-experts			RIPPER
k	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.	Error
3	1.71	0.018	0.991	4.63	0.052	0.975	14.71	0.203	0.917	4.50	0.049	0.977	8.52
4	2.22	0.027	0.988	5.40	0.066	0.971	14.36	0.250	0.914	6.27	0.075	0.967	10.78
5	3.86	0.051	0.978	6.24	0.086	0.965	15.25	0.290	0.907	9.18	0.123	0.950	14.35
6	4.21	0.058	0.976	7.29	0.108	0.959	15.37	0.323	0.905	9.93	0.145	0.945	14.81

Table A.3. Description of the classes constituting the first single-label subset of AP titles.

	Class	#Docs	Cum. #docs		Class	#Docs	Cum. #docs
1	japan	4272	4272	11	aparts	770	26262
2	bush	3738	8010	12	dukakis	702	26964
3	israel	3541	11551	13	yugoslavia	575	27539
4	britx	3370	14921	14	quayle	488	28027
5	gulf	3216	18137	15	ireland	457	28484
6	german	2321	20458	16	burma	432	28916
7	weather	1824	22282	17	bonds	384	29300
8	dollargold	1613	23895	18	nielsens	248	29548
9	hostages	800	24695	19	boxoffice	170	29718
10	budget	797	25492	20	tickertalk	123	29841

Table A.4. Results for the first single-label subset of AP titles (Table A.3).

	Real AdaBoost.MH			Naive-Bayes			Rocchio			Sleeping-experts			Ripper
k	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.	Error
3	5.87	0.0789	0.9673	11.53	0.1397	0.9383	24.79	0.4144	0.8483	5.20	0.0661	0.9716	10.71
4	9.34	0.1517	0.9450	16.90	0.2503	0.9035	33.15	0.6365	0.7886	9.48	0.1501	0.9446	18.76
5	11.39	0.2075	0.9313	21.04	0.3528	0.8757	30.98	0.7888	0.7867	11.92	0.2161	0.9281	21.78
6	13.19	0.2782	0.9171	23.86	0.4525	0.8545	31.28	0.9386	0.7789	13.90	0.2972	0.9123	23.80
7	12.43	0.2671	0.9220	21.89	0.4204	0.8664	29.23	0.9002	0.7935	13.36	0.2861	0.9162	22.81
8	12.04	0.2606	0.9246	21.29	0.4084	0.8701	27.77	0.8480	0.8047	13.25	0.2954	0.9161	21.93
9	12.57	0.2887	0.9207	21.71	0.4383	0.8661	28.00	0.9541	0.8004	13.99	0.3230	0.9109	22.25
10	13.27	0.3098	0.9160	22.77	0.4788	0.8587	28.38	1.0103	0.7977	14.89	0.3600	0.9049	23.29
11	14.24	0.3461	0.9094	23.80	0.5301	0.8504	30.02	1.1412	0.7861	15.75	0.4163	0.8974	25.69
12	13.97	0.3500	0.9108	24.82	0.5636	0.8441	29.76	1.1892	0.7873	16.08	0.4356	0.8953	25.41
13	14.71	0.3949	0.9049	25.03	0.5940	0.8415	30.24	1.2752	0.7838	16.49	0.4675	0.8921	26.03
14	15.01	0.4050	0.9033	26.25	0.6570	0.8322	30.32	1.3229	0.7828	16.97	0.5019	0.8884	25.83
15	15.53	0.4372	0.8993	26.65	0.6888	0.8289	30.44	1.3933	0.7811	17.08	0.5248	0.8869	26.74
16	15.58	0.4647	0.8979	26.87	0.7191	0.8268	30.63	1.4676	0.7786	17.33	0.5510	0.8849	26.74
17	16.00	0.4915	0.8950	27.09	0.7483	0.8245	31.04	1.4875	0.7760	17.59	0.5697	0.8829	26.86
18	15.93	0.5008	0.8950	27.02	0.7722	0.8239	30.98	1.4955	0.7768	17.93	0.5940	0.8806	26.55
19	15.91	0.5041	0.8951	27.07	0.7948	0.8229	31.04	1.5212	0.7759	17.58	0.6035	0.8824	26.90
20	16.29	0.5483	0.8920	27.04	0.8365	0.8218	32.11	1.6277	0.7674	18.01	0.6430	0.8788	27.52

Table A.5. Description of the classes constituting the second single-label subset of AP titles.

	Class	#Docs train	#Docs test
1	Domestic	46142	21605
2	International	44499	21398
3	Financial	22698	11410
4	Washington	22407	10656
5	Political	5739	1313
6	Entertainment	1242	591

Table A.6. Error rates for the different algorithms on the second single-label subset of AP titles (Table A.5).

(30,000 Rounds) real AdaBoost.MH	(180,000 Rounds) discrete AdaBoost.MH	RIPPER	Sleeping-experts	Rocchio
27.43	32.22	53.29	29.44	40.14

Table A.7. Description of the classes constituting the first multi-label subset of Reuters-21450.

	Class	#Docs	Cum. #docs	Avg. no. labels/docs
1	Earnings	3964	3964	–
2	Acquisitions	2369	6333	–
3	Commodity	1695	8028	1.0064
4	Economics	1140	9168	1.0116
5	Interest	717	9885	1.0171
6	Energy	701	10586	1.0220
7	Money-Fx	478	11064	1.0407
8	Shipping	286	11350	1.0534
9	Currency	238	11588	1.0738

Table A.8. Results for the first multi-labeled subset of Reuters-21450 (Table A.7).

	Real AdaBoost.MH			Naive-Bayes			Rocchio			Sleeping-experts		
k	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.
3	1.96	0.0283	0.9898	5.30	0.0666	0.9723	14.64	0.2164	0.9163	3.23	0.0459	0.9826
4	2.42	0.0408	0.9871	5.66	0.0846	0.9693	14.15	0.2681	0.9138	4.42	0.0714	0.9755
5	3.24	0.0616	0.9821	6.64	0.1122	0.9631	14.10	0.2985	0.9130	6.33	0.1127	0.9640
6	3.80	0.0792	0.9785	7.03	0.1402	0.9595	13.93	0.3392	0.9122	6.94	0.1370	0.9597
7	5.15	0.1315	0.9697	8.32	0.1969	0.9507	13.91	0.3940	0.9107	8.61	0.2660	0.9439
8	5.10	0.1486	0.9695	8.79	0.2433	0.9465	14.31	0.4336	0.9078	9.50	0.3126	0.9382
9	5.24	0.1868	0.9674	8.84	0.2856	0.9452	14.79	0.4843	0.9042	9.15	0.4073	0.9361

Table A.9. Description of the classes constituting the second multi-label subset of Reuters-21450.

	Class	#Docs	Cum. #docs	Avg. no. labels/docs		Class	#Docs	Cum. #docs	Avg. no. labels/docs
1	money-fx	717	717	–	11	oilseed	171	4167	1.3835
2	grain	582	1299	–	12	sugar	162	4329	1.3778
3	crude	578	1877	1.0032	13	coffee	139	4468	1.3693
4	trade	486	2363	1.0261	14	gnp	136	4604	1.3682
5	interest	478	2841	1.0956	15	oil	124	4728	1.2842
6	ship	286	3127	1.1309	16	gold	124	4852	1.3625
7	wheat	283	341	1.2319	17	soybean	111	4963	1.3937
8	corn	237	3647	1.3176	18	gas	105	5068	1.3544
9	dlr	175	3822	1.3773	19	bop	105	5173	1.4247
10	supply	174	3996	1.3629					

Table A.10. Results for the second multi-labeled subset of Reuters-21450 (Table A.9).

	Real AdaBoost.MH			Naive-Bayes			Rocchio			Sleeping-experts		
k	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.	Error	Cover	Prec.
3	1.07	0.0155	0.9944	1.87	0.0257	0.9901	1.82	0.0225	0.9907	1.66	0.0208	0.9915
4	2.95	0.0625	0.9842	5.77	0.1151	0.9667	9.81	0.1329	0.9496	5.99	0.1033	0.9674
5	6.83	0.1978	0.9613	10.14	0.2646	0.9409	11.30	0.2391	0.9392	10.64	0.2503	0.9408
6	7.63	0.2590	0.9554	11.68	0.3729	0.9285	12.04	0.2850	0.9351	11.68	0.3212	0.9330
7	7.30	0.3671	0.9564	13.04	0.5448	0.9174	12.28	0.3934	0.9334	11.42	0.4187	0.9345
8	7.41	0.4682	0.9545	13.44	0.7240	0.9093	12.14	0.4956	0.9321	11.31	0.5213	0.9337
9	8.22	0.5553	0.9487	14.63	0.9041	0.8971	13.15	0.5888	0.9263	11.78	0.7719	0.9199
10	8.90	0.5768	0.9429	14.56	0.8469	0.8984	13.71	0.6153	0.9195	14.53	0.9130	0.8973
11	9.36	0.6338	0.9383	16.20	1.0817	0.8809	14.67	0.6657	0.9136	15.37	1.0110	0.8909
12	9.01	0.6200	0.9414	16.52	1.1012	0.8790	13.81	0.6620	0.9173	13.97	1.0359	0.8979
13	9.29	0.6378	0.9389	16.12	1.1382	0.8822	14.01	0.6448	0.9175	13.67	1.0257	0.9011
14	8.65	0.6710	0.9392	17.24	1.1866	0.8736	13.58	0.6734	0.9161	15.22	1.2101	0.8862
15	6.30	0.4888	0.9576	14.69	1.1174	0.8876	11.68	0.5966	0.9232	14.47	0.8724	0.9006
16	9.74	0.7082	0.9329	18.23	1.3679	0.8652	14.38	0.7321	0.9083	15.95	1.2842	0.8813
17	9.21	0.7001	0.9372	18.25	1.6119	0.8602	14.27	0.7863	0.9081	15.84	1.3839	0.8807
18	7.08	0.6179	0.9510	15.59	1.5274	0.8704	12.16	0.7298	0.9178	15.56	1.3659	0.8826
19	10.02	0.8369	0.9295	18.01	1.7656	0.8570	14.51	0.8251	0.9052	17.43	1.8193	0.8613

Table A.11. List of the newsgroups and the number of articles posted to the newsgroups. (An article may be posted to multiple groups.).

Group	#Docs	Group	#Docs
alt.atheism	1114	rec.sport.hockey	1000
comp.graphics	1002	sci.crypt	1000
comp.os.ms-windows.misc	1000	sci.electronics	1000
comp.sys.ibm.pc.hardware	1028	sci.med	1001
comp.sys.mac.hardware	1002	sci.space	1000
comp.windows.x	1000	soc.religion.christian	997
misc.forsale	1005	talk.politics.guns	1008
rec.autos	1004	talk.politics.mideast	1000
rec.motorcycles	1000	talk.politics.misc	1163
rec.sport.baseball	1000	talk.religion.misc	1023

Acknowledgments

We would like to thank Allen Gorin, David Lewis, Andrew McCallum, Fernando Pereira, Amit Singhal and Jerry Wright for useful discussions and for help with our experiments.

Notes

1. Arbitrary long (sparse) n-grams but we restrict ourselves to words and word bigrams for comparison purposes.
2. "Function words" include high frequency but contentless words like 'of' and 'the'. We used the stop-list given by Lewis (1992).
3. http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html.
4. There are actually 15 categories but only 6 of them contain more than 40 articles. We discarded the 9 categories (and the corresponding articles) with only a few articles.

References

Apté, C., Damerau, F., & Weiss, S. M. (1994). Towards language independent automated learning of text categorization models. *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 23–30).

Biebricher, P., Fuhr, N., Lustig, G., Schwantner, M., & Knorz, G. (1988). The automatic indexing system AIR/PHYS—from research to application. *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 333–342).

Blum, A. (1997). Empirical support for winnow and weighted-majority based algorithms: results on a calendar scheduling domain. *Machine Learning, 26*, 5–23.

Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics, 26*(3), 801–849.

Cohen, W. (1995). Fast effective rule induction. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 115–123).

Cohen, W.W. & Singer, Y. (1996). Context-sensitive learning methods for text categorization. *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 307–315).

Drucker, H. & Cortes, C. (1996). Boosting decision trees. In *Advances in Neural Information Processing Systems 8* (pp. 479–485).

Field, B.J. (1975). Towards automatic indexing: automatic assignment of controlled-language indexing and classification from free indexing. *Journal of Documentation, 31*(4), 246–265.

Freund, Y. & Schapire, R.E. (1996). Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference* (pp. 148–156).

Freund, Y. & Schapire, R.E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*(1), 119–139.

Freund, Y., Schapire, R.E., Singer, Y., & Warmuth, M.K. (1997). Using and combining predictors that specialize. *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing* (pp. 334–343).

Fuhr, N. & Pfeifer, U. (1994). Probabilistic information retrieval as a combination of abstraction, inductive learning, and probabilistic assumptions. *ACM Transactions on Information Systems, 12*(1), 92–115.

Gorin, A.L., Parker, B.A., Sachs, R.M., & Wilpon, J.G. (1996). How may I help you?. *Proceedings Interactive Voice Technology for Telecommunications Applications (IVTTA)* (pp. 57–60).

Gorin, A.L., Riccardi, G., & Wright, J.H. (1997). How may I help you?. *Speech Communication, 23*(1-2), 113–127.

Ittner, D.J., Lewis, D.D., & Ahn, D.D. (1995). Text categorization of low quality images. *Symposium on Document Analysis and Information Retrieval* (pp. 301–315). Las Vegas, NV. ISRI; Univ. of Nevada, Las Vegas.

Joachims, T. (1997). A probabilistic analysis of the Rochhio algorithm with TFIDF for text categorization. *Machine Learning: Proceedings of the Fourteenth International Conference* (pp. 143–151).

Koller, D. & Sahami, M. (1997). Hierarchically classifying documents using very few words. *Machine Learning: Proceedings of the Fourteenth International Conference* (pp. 171–178).

Lang, K. (1995). Newsweeder: Learning to filter netnews. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 331–339).

Lewis, D. (1992). Representation and learning in information retrieval. Technical Report 91-93, Computer Science Department, University of Massachusetts at Amherst. Ph.D. Thesis.

Lewis, D. & Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. *Machine Learning: Proceedings of the Eleventh International Conference*.

Lewis, D. & Gale, W. (1994). Training text classifiers by uncertainty sampling. *Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Lewis, D.D. & Ringuette, M. (1994). A comparison of two learning algorithms for text categorization. *Third Annual Symposium on Document Analysis and Information Retrieval* (pp. 81–93).

Maclin, R. & Opitz, D. (1997). An empirical evaluation of bagging and boosting. *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (pp. 546–551).

Margineantu, D.D. & Dietterich, T.G. (1997). Pruning adaptive boosting. *Machine Learning: Proceedings of the Fourteenth International Conference* (pp. 211–218).

Mitchell, T.M. (1997). *Machine learning*. McGraw Hill.

Moulinier, I., Raškinis, G., & Ganascia, J.-G. (1996). Text categorization: a symbolic approach. *Fifth Annual Symposium on Document Analysis and Information Retrieval* (pp. 87–99).

Ng, H.T., Goh, W.B., & Low, K.L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 67–73).

Quinlan, J.R. (1996). Bagging, boosting, and C4.5. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 725–730).

Riccardi, G., Gorin, A.L., Ljolje, A., & Riley, M. (1997). Spoken language understanding for automated call routing. *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing* (pp. 1143–1146).

Rocchio, J. (1971). Relevance feedback information retrieval. In G. Salton, (Ed.), *The Smart retrieval system—experiments in automatic document processing* (pp. 313–323). Englewood Cliffs, NJ: Prentice-Hall.

Salton, G. (1991). Developments in automatic text retrieval. *Science, 253*, 974–980.

Salton, G. & McGill, M.J. (1983). *Introduction to modern information retrieval*. McGraw-Hill.

Schapire, R.E. (1997). Using output codes to boost multiclass learning problems. *Machine Learning: Proceedings of the Fourteenth International Conference* (pp. 313–321).

Schapire, R.E., Freund, Y., Bartlett, P., & Lee, W.S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics. 26*(5), 1651–1686.

Schapire, R.E. & Singer, Y. (1998). Improved boosting algorithms using confidence-rated predictions. *Proceedings of the Eleventh Annual Conference on Computational Learning Theory* (pp. 80–91). To appear, *Machine Learning*.

van Rijsbergen, C.J. (1979). *Information retrieval*. London: Butterworths.

Weiss, S., Apte, C., Damerau, F., Johnson, D., Oles, F., Goetz, T., & Hampp, T. (1999). Maximizing text-mining performance. *IEEE Intelligent Systems*.

Wright, J.H., Gorin, A.L., & Riccardi, G. (1997). Automatic acquisition of salient grammar fragments for call-type classification. *Proceedings of the 5th European Conference on Speech Communication and Technology* (pp. 1419–1422).

Yang, Y. (1994). Expert network: effective and efficient learning from human decisions in text categorization and retrieval. *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 13–22).

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Information Retrieval*, *1*, 69–90.

Received March 16, 1998
Accepted July 22, 1999
Final Manuscript July 22, 1999