A Boosting Algorithm for Label Covering in Multilabel Problems

Yonatan Amit
mitmit@cs.huji.ac.il
School of Comp. Sci. and Eng.
Hebrew University
Jerusalem, Israel

Ofer Dekel
oferd@cs.huji.ac.il
School of Comp. Sci. and Eng.
Hebrew University
Jerusalem, Israel

Yoram Singer
singer@google.com
Google Inc.
1600 Amphitheatre Pkwy.
94043, Mountain View, CA

Abstract

We describe, analyze and experiment with a boosting algorithm for multilabel categorization problems. Our algorithm includes as special cases previously studied boosting algorithms such as Adaboost.MH. We cast the multilabel problem as multiple binary decision problems, based on a user-defined covering of the set of labels. We prove a lower bound on the progress made by our algorithm on each boosting iteration and demonstrate the merits of our algorithm in experiments with text categorization problems.

1 Introduction

This paper is concerned with the problem of learning a multilabel categorization function from a labeled set of examples. Each example consists of an instance and an associated set of relevant labels. Our goal is to learn a categorization function which accurately predicts the relevant set of labels for a given instance. This task naturally emerges in problems such as text classification and collaborative filtering.

Multilabel problems are often recast as ranking problems, where the goal is to order the entire set of labels according to their relevance to the instance at hand (see [5, 2, 4, 3, 7]). The ranking approach may be useful in applications where the goal is to prioritize the relevance of the topics. Such approach, however, is inappropriate in settings where the set of relevant labels must be uniquely identified. For example, consider the task of automatic email routing. A large company receives an email which is not addressed to a specific department and an automated system should decide to which subset of departments to route the email. A ranking function would order the list of all the departments according to their relevance to the topic of the email. However, some emails should be routed to a single department whereas others should be routed to multiple departments, and the ranking function does not provide such information. In this example, we are clearly interested in the precise subset of relevant departments.

Another approach to the multilabel categorization problem is to decompose it into independent binary classification problems, one problem for each label. Such an approach was tacitly applied, for instance, by a multilabel version of AdaBoost called AdaBoost.MH [8]. Breaking a multilabel problem into multiple independent problems greatly simplifies the learning task and enables the usage of off-the-shelf binary classification learning algorithms. However, any such decomposition is unlikely to capture apparent correlations between the labels. In this paper we enhance this approach and attempt to overcome its shortcomings by devising a general *boosting* algorithm for the multilabel categorization problem. Our algorithm requires the user to specify how prediction errors should be evaluated, and then attempts to minimize this user-defined criterion. For instance, the user can specify that predictions should be penalized according to the size of the symmetric difference between the correct set of labels and the predicted set of labels. In this case, our algorithm simply reduces to AdaBoost.MH. Other criteria for evaluating the quality of a prediction lead to the new algorithmic framework described in the sequel.

The user-defined evaluation criterion is relayed through a covering of the set of labels. Namely, the user predefines subsets of labels, which are not necessarily disjoint, but whose union equals the entire set of labels. Hence, we refer to our approach as `label covering` and abbreviate the resulting algorithm by AdaBoost.LC. Each set in the cover corresponds to a decision task. Failing to predict one or more of the labels in a set counts as one error. Thus, the maximal number of errors that can be associated with a single prediction equals the number of sets in the cover.

One extreme is to place each label in its own unique set, which amounts to counting the number of incorrectly classified labels. The other extreme is to define a single set which includes the entire set of labels. This definition implies that a single incorrectly classified label receives the same penalty as multiple incorrectly classified labels. The more interesting label-covering schemes lie between the two extremes and capture some prior knowledge about the problem.

The label covering mechanism becomes especially handy when the set of labels can be naturally partitioned into subsets. For instance, consider a text categorization task, where the instances of the problem are textual documents and the label set includes the labels FOOTBALL, BASKETBALL, FINANCIAL MARKETS and CURRENCIES. This label set splits very naturally into two broad-topic categories: SPORTS and ECONOMY. In this example, we are not especially interested in the exact number of incorrectly predicted labels. Instead, we are interested in the number of incorrectly predicted broad-topic categories. An algorithm which minimizes the number of incorrectly predicted labels does not necessarily minimize the number of incorrectly predicted high-level-categories.

The covering mechanism is in fact even more flexible. A covering may differ between instances, and may depend on the labeling of the instance on hand. For example, in the textual categorization task described above, a BASKETBALL related article may be evaluated by summing the prediction mistakes in the SPORTS category, while any mistake in the ECONOMY category should simply be considered as a unit loss. Our AdaBoost.LC algorithm lets the user choose the label covering that suits best her needs, and then devises a specific boosting procedure that is tailored to the covering that was chosen.

The core of our algorithm is based on recent advances in boosting algorithms. As mentioned above our approach generalizes the AdaBoost.MH algorithm and a few other variants described in [8, 1, 3]. Other large margin algorithms can also be adapted to solve the label covering problem. For instance, both the kernel-based approaches for label ranking discussed in [2, 4] and the large margin methods for structured prediction [10, 11] can be exported and used for the label covering task. One of our future research goals is to explore ways to combine the label covering approach presented in this paper with kernel methods.

The paper is organized as follows. We formally describe of our settings in Sec. 2. In Sec. 3 we describe and analyze the learning algorithm for label categorization. We conclude with an empirical evaluation of our algorithm in Sec. 4.

2 Problem Setting

In this section we introduce the notation used throughout this paper and describe our problem setting. We denote scalars by lower case letters (e.g. x) and vectors by bold face letters (e.g. $\mathbf{x}$). We denote by v^l the l'th coordinate of a vector $\mathbf{v}$. Sets are denoted by upper case Latin letters (e.g. A). Elements of a set are indexed by subscripting (e.g. a_s for scalars and $\mathbf{a}_s$ for vectors). For any natural number k, we denote the set $\{1, 2, \ldots, k\}$ by $[k]$. We identify a set $A \subset [k]$ with a binary vector $\mathbf{a}$ where $a^l = 1 \iff l \in A$. Last, we denote by $[\![x]\!]$ the predicate which takes a value of 1 when x is true, and is 0 otherwise.

The problem we are exploring is concerned with multiclass multilabel classification. Let $\mathcal{X}$ denote the instance space and $\mathcal{Y} = \{-1, 1\}^k$ denote the set of possible labelings. A labeling for $\mathbf{x} \in \mathcal{X}$ is a vector $\mathbf{y} \in \mathcal{Y}$ where we consider the label $l \in [k]$ to be relevant to $\mathbf{x}$ iff $y^l = 1$.

We next introduce the notion of a non-disjoint covering of a set of labels. A cover-element is a set $A \subset [k]$. We consider the labels $l \in A$ as tied together in the sense that failing to predict a *single* label in A amounts to the same error as failing to predict *all* of them. As described above, we identify a cover-element A with the binary vector $\mathbf{a}$. Thus, for a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ and a pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$, the prediction error associated with a cover-element $\mathbf{a}$ is defined as

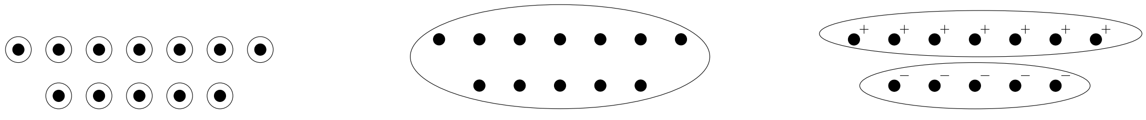
$$[\![\exists l \text{ s.t. } a^l = 1 \wedge f^l(\mathbf{x}) \neq y^l]\!] \quad .$$

A covering $\mathcal{A}$ is a mapping from a labeling vector to a multiset of cover-elements. We denote the size of $\mathcal{A}(\mathbf{y})$ by $n(\mathbf{y}) = |\mathcal{A}(\mathbf{y})|$ and the elements of $\mathcal{A}(\mathbf{y})$ as $\{\mathbf{a}_s\}_{s=1}^{n(\mathbf{y})}$. We analogously define the prediction error associated with the covering $\mathcal{A}$ as the sum of errors with respect to cover-elements of $\mathcal{A}(\mathbf{y})$,

$$err(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathcal{A}) = \sum_{\mathbf{a} \in \mathcal{A}(\mathbf{y})} [\![\exists l : \text{ s.t. } a^l = 1 \wedge f^l(\mathbf{x}) \neq y^l]\!] \quad .$$

For abbreviation, we call this the *covering error*. As we now illustrate, this notion of generalized error is flexible and includes several previously studied error models for multilabel problems

Fig. 1 illustrates a few possible choices for $\mathcal{A}$. The covering described in (a) puts each label in a separate set $a_s^l = \delta_{ls}$ and translates to an error function which is equivalent to the Hamming distance between the true labels and the predicted labels (similar to Adaboost.MH of [8]). The covering described in (b) ties all labels into a single set. The covering error in this



(a)

(b)

(c)

Figure 1: Examples of different coverings.

case generalizes the $0-1$ error and assigns a 0 error if a perfect prediction is made, and a unit loss otherwise. The covering described in (c) amounts to tying together all relevant labels and, separately, tying all irrelevant ones, associating a unit loss when failing to predict *any* of the relevant labels as relevant, and equivalently a unit loss when failing to predict correctly *any* of the irrelevant. We would like to emphasize that the above covering depends on the correct labeling of the instances. The last example underscores the fact that the covering may depend on the input instance and its label and therefore may change as a function of the example being handled.

The covering error is a combinatorial measure of the accuracy of a predictor. Thus, finding a hypothesis f which minimizes the covering error is difficult. We thus expand the image of f to the range $\mathbb{R}^k$ allowing the magnitude $|f^l|$ to denote the confidence of a prediction. We further employ a smooth, convex upper bound on the covering error, and construct an hypothesis $\mathbf{f}$ that minimizes the upper bound, and thus bounds the covering error. Formally, Given a covering $\mathcal{A}$ and an hypothesis $\mathbf{f} : \mathcal{X} \rightarrow \mathbb{R}^k$, the covering loss, denoted by L, on an example $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ is

$$L(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathcal{A}) = \sum_{s=1}^{n(y)} \log\left(1 + \sum_{l=1}^{k} a_s^l \exp\left(-y^l f^l(\mathbf{x})\right)\right) .$$

Given a training set

$$S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)\} \subset \mathcal{X} \times \mathcal{Y} \ ,$$

we define the empirical loss L as the average loss over the training set, namely,

$$\begin{aligned} L(\mathbf{f}, S, \mathcal{A}) &= \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{f}, (\mathbf{x}_i, \mathbf{y}_i), \mathcal{A}) \\ &= \frac{1}{m} \sum_{i=1}^{m} \sum_{s=1}^{n(\mathbf{y}_i)} \log\left(1 + \sum_{l=1}^{k} a_s^l \exp\left(-y_i^l f^l(\mathbf{x}_i)\right)\right) . \end{aligned}$$

We refer to the product $y_i^l f^l(\mathbf{x}_i)$ as the (signed) *margin* of example i on label l. Denoting $z^l = -y_i^l f^l(\mathbf{x}_i)$ and $z^* = \max_l z^l$, we can upper bound z^* by

$$\begin{aligned} \log\left(\sum e^{z^l}\right) &= \log\left(e^{z^*} \sum e^{z^l - z^*}\right) \\ &= z^* \log e + \log\left(\underbrace{\sum e^{z^l - z^*}}_{\geq 1}\right) \geq z^* \ . \end{aligned}$$

Note that a prediction $f^l(\mathbf{x})$ is incorrect iff z^* is positive, thus minimizing the multilabel covering-loss translates to a bound on the number of prediction mistakes. We can therefore view the multilabel covering-loss as a proxy for minimizing the $0-1$ loss on each set of labels defined by the covering $\mathcal{A}$.

We generalize techniques presented in [1] and propose an iterative algorithm that utilizes a family of base hypotheses (also referred to as weak hypotheses) and generates a linear combination of these hypotheses. Let $\mathcal{H} = \{h_1 \ldots h_H\}$ denote the set of base hypotheses, where $h \in \mathcal{H}$ is a function $h : \mathcal{X} \rightarrow [-1, 1]$. On each iteration, the algorithm chooses hypotheses from $\mathcal{H}$, and updates the weights of these hypotheses in the combination. The update is governed by a set of templates $\mathcal{B} \in \mathbb{R}_+^H$, where each $\mathbf{b} \in \mathcal{B}$ defines a subset of hypotheses from $\mathcal{H}$ along with appropriate scaling. On every iteration, the algorithm selects a $\mathbf{b} \in \mathcal{B}$ and updates the weights of all hypotheses for which $b_i \neq 0$. For instance, we may set $\mathcal{B}$ to $\{\mathbf{e}_1, \ldots, \mathbf{e}_H\}$, where e_i^j is 1 if $i = j$ and 0 otherwise. In this manner, on each iteration we select a single base hypothesis and thus obtain a sequential update algorithm which is analogous to Adaboost.MH from [8]. We emphasize that while the base hypotheses we employ are class independent binary classifiers our algorithm ties the hypotheses and combines them into a single multi label predictor.

3 AdaBoost.LC

Our algorithm, presented in Fig. 2, is a boosting algorithm for label-covering which uses the covering-loss. The algorithm updates its prediction function on every iteration in order to improve on the loss function. On each iteration t, the algorithm assigns weights to training examples in accordance with their relative contribution to the cumulative loss: for each example $(\mathbf{x}_i, \mathbf{y}_i)$ and each vector in the covering $\mathbf{a}_s \in \mathcal{A}(\mathbf{y}_i)$, the algorithm defines k weight variables $\{q_t(i, s, l)\}_{l=1}^{k}$. Each

```
Input:
    $S = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^m,\ \mathcal{A}\ ,\ \mathcal{B} \subset \mathbb{R}_+^H$ s.t. $\forall b \in \mathcal{B} : \|b\|_1 \le 1$
Output:
    $f_T^l(\mathbf{x}) = \sum_{j=1}^{H} \alpha_T(j,l)\, h_j(\mathbf{x})$    $(l \in [k])$
For $t = 1, 2, \ldots, T$:
        $q_t(i,s,l) = \dfrac{a_s^l \exp\left(-y_i^l\, f_{t-1}^l(\mathbf{x}_i)\right)}{1 + \sum_{r=1}^k a_s^r \exp\left(-y_i^r\, f_{t-1}^r(\mathbf{x}_i)\right)}$    $\big(i \in [m], s \in [n(\mathbf{y}_i)], l \in [k]\big)$
        $W_{j,l}^+ = \sum_{i=1}^{m} \sum_{s=1}^{n(\mathbf{y}_i)} \Big[q_t(i,s,l)\ y_i^l\, h_j(\mathbf{x}_i)\Big]_+$    $\big(j \in [H], l \in [k]\big)$
        $W_{j,l}^- = \sum_{i=1}^{m} \sum_{s=1}^{n(\mathbf{y}_i)} \Big[-q_t(i,s,l)\ y_i^l\, h_j(\mathbf{x}_i)\Big]_+$    $\big(j \in [H], l \in [k]\big)$
        $d_{j,l} = \frac{1}{2} \ln\left(\frac{W_{j,l}^+}{W_{j,l}^-}\right)$    $\big(j \in [H], l \in [k]\big)$
        $b_l = \arg\max_{\mathbf{b} \in \mathcal{B}} \sum_{j=1}^{H} b^j \left(\sqrt{W_{j,l}^+} - \sqrt{W_{j,l}^-}\right)^2$    $\big(l \in [k]\big)$
        $\alpha_t(j,l) = \alpha_{t-1}(j,l) + b_l^j d_{j,l}$    $\big(j \in [H], l \in [k]\big)$
```

Figure 2: Adaboost.LC

variable assesses the relative contribution of the l'th label to the loss. This defines an un-normalized distribution over the training examples (similar to AdaBoost's importance weights typically denoted as $D_t(i)$). The algorithm then assesses the potential decrease of the loss by calculating the success and failure scores (denoted $W_{j,l}^+$ and $W_{j,l}^-$ respectively) associated with each base hypothesis. The scores are the sum on all training examples and all vectors in the covering $\mathcal{A}$ weighted according to $q_t(i,s,l)$. The ratio of $W_{j,l}^+$ and $W_{j,l}^-$ assesses the predictability potential of the j'th hypothesis with respect to the l'th label. The algorithm then selects the template $b \in \mathcal{B}$ which maximizes the predictability potential and updates the multilabel predictor.

The following theorem provides a lower bound on the decrease in the loss on every iteration of the algorithm.

Theorem 3.1. *Let $\mathcal{A}$ denote a covering of the set of labels. Let $\mathcal{B} \subset \mathbb{R}_+^H$ denote a set of hypotheses templates. Let $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$ be a training set of m examples. Denote by f_t the multi-class predictor constructed after t iterations by the algorithm described in Fig. 2. Then the decrease in the training loss L on every iteration is bounded below as follows*

$$\begin{aligned} & L(f_t, S, \mathcal{A}) - L(f_{t+1}, S, \mathcal{A}) \\ & \qquad \ge \frac{1}{|S|} \sum_{l=1}^{k} \max_{b \in \mathcal{B}} \sum_{j=1}^{H} b^j \left(\sqrt{W_{j,l}^+} - \sqrt{W_{j,l}^-}\right)^2 \quad . \end{aligned}$$

Proof. Throughout the proof we use the following notation,

$$\pi_{i,s,l}^t \overset{\text{def}}{=} a_s^l \exp\left(-y_i^l\, f_t^l(\mathbf{x}_i)\right)\ ,\ \phi_{i,s}^t \overset{\text{def}}{=} \sum_{l=1}^{k} \pi_{i,s,l}^t$$

and

$$\Delta_{i,s}^t \overset{\text{def}}{=} \log(1 + \phi_{i,s}^t) - \log(1 + \phi_{i,s}^{t+1}) \quad .$$

We abbreviate $q_{t+1}(i,s,l)$ by $q_{i,s,l}$ and

$$m_{i,j,l} = b_l^j\ y_i^l\, h_j(\mathbf{x}_i) \quad .$$

For brevity and clarity, we omit the ranges of summations. Summations over i are from 1 through m, summations over s range from 1 through $n\left(\mathbf{y}_i\right)$, summations over j are from 1 through H, and over l are from 1 through k.

We start the proof by expanding $\Delta_{i,s}^t$ to get that

$$\begin{aligned} \Delta_{i,s}^t &= \log(1 + \phi_{i,s}^t) - \log(1 + \phi_{i,s}^{t+1}) \\ &= -\log\left(\frac{1 + \phi_{i,s}^{t+1}}{1 + \phi_{i,s}^t}\right) = -\log\left(1 - \frac{\phi_{i,s}^t - \phi_{i,s}^{t+1}}{1 + \phi_{i,s}^t}\right) \\ &\ge \frac{\phi_{i,s}^t - \phi_{i,s}^{t+1}}{1 + \phi_{i,s}^t} \quad . \qquad (1) \end{aligned}$$

where we used the fact that $-\log(1-x) \ge x$ to obtain the last inequality. We now rewrite the difference in the loss attained by f_t and f_{t+1} as follows,

$$|S|\left(L(\mathbf{f}_t, S, \mathcal{A}) - L(\mathbf{f}_{t+1}, S, \mathcal{A})\right) = \sum_i \sum_s \Delta_{i,s}^t \quad . \qquad (2)$$

Using the bound on $\Delta^t_{i,s}$ from Eq. (1) in Eq. (2), we further bound Eq. (2) by the following expression,

$$\sum_i \sum_s \frac{\sum_l \pi^t_{i,s,l} - \sum_l \pi^{t+1}_{i,s,l}}{1+\sum_{r=1}^k \pi^t_{i,s,r}} \ . \tag{3}$$

The update $\alpha_{t+1}(j,l) = \alpha_t(j,l) + b^j_l\, d_{j,l}$ at the end of iteration $t+1$ implies that $f^l_{t+1} = f^l_t + \sum_j b^j_l d_{j,l} h_j$ and thus

$$\begin{aligned} y^l_i f^l_{t+1}(\mathbf{x}_i) &= y^l_i f^l_t(\mathbf{x}_i) + y^l_i \sum_j b^j_l\, d_{j,l}\, h^j(\mathbf{x}_i) \\ &= y^l_i f^l_t(\mathbf{x}_i) + \sum_j d_{j,l}\, m_{i,j,l} \ . \end{aligned}$$

We therefore obtain that the following equality which holds for all i whenever $a^l_s = 1$,

$$\begin{aligned} \pi^{t+1}_{i,s,l} &= \exp\left(-y^l_i f^l_{t+1}(\mathbf{x}_i)\right) \\ &= \pi^t_{i,s,l} \exp\left(-\sum_j d_{j,l} m_{i,j,l}\right) \ . \end{aligned} \tag{4}$$

We note that if $a^l_s = 0$ then $\pi^t_{i,s,l} = 0 = \pi^{t+1}_{i,s,l}$. Therefore, Eq. (4) holds whether a^l_s is either zero or one. Eq. (3) can thus be rewritten as

$$\sum_{i,s,l}\left[\frac{\pi^t_{i,s,l}}{1+\sum_{r=1}^k \pi^t_{i,s,r}}\left(1-\exp\Big(-\sum_j d_{j,l} m_{i,j,l}\Big)\right)\right]$$

Plugging the definition of $q_{i,s,l}$, we can rewrite the above equation as follows,

$$\sum_{i,s,l}\left[q_{i,s,l}\left(1-\exp\Big(-\sum_j d_{j,l} m_{i,j,l}\Big)\right)\right] \ . \tag{5}$$

Note that $d_{j,l} m_{i,j,l}$ can be written as

$$d_{j,l}\,|m_{i,j,l}|\,\sigma(m_{i,j,l})$$

where $\sigma(x) = \mathrm{sign}(x)$. Furthermore, note that $\mathbf{b}_l \in \mathcal{B}$ implies that,

$$\sum_j |m_{i,j,l}| \leq \sum_j |b^j_l| = \|\mathbf{b}_l\|_1 \leq 1 \ . \tag{6}$$

We now apply *Jensen's* Inequality to the concave function $1-e^x$ and further bound Eq. (5) by the following term,

$$\sum_{i,s,l}\left[q_{i,s,l}\sum_j |m_{i,j,l}|\left(1-e^{-d_{j,l}\sigma(m_{i,j,l})}\right)\right] \ .$$

We now rearrange terms in the above equation, decompose $|m_{i,j,l}|$ into $|b^j_l| \cdot |y^l_i\, h_j(\mathbf{x}_i)|$ and obtain the following lower bound on the decrease in the loss,

$$\sum_{i,s} \Delta^t_{i,s} \geq \tag{7}$$

$$\sum_{l,j} |b^j_l| \left(\sum_{i,s}\left[q_{i,s,l}|y^l_i\, h_j(\mathbf{x}_i)|\left(1-e^{-d_{j,l}\sigma(m_{i,j,l})}\right)\right]\right) \ .$$

Since $b^j_l \geq 0$, we may omit the absolute value. We now use the definition of $W^+_{j,l}$ and $W^-_{j,l}$ and rewrite the above lower bound as follows,

$$\sum_l \sum_j b^j_l \left[W^+_{j,l}\left(1-e^{-d_{j,l}}\right) + W^-_{j,l}\left(1-e^{+d_{j,l}}\right)\right] \ . \tag{8}$$

Now, using the definition of $d_{j,l}$ we obtain

$$e^{-d_{j,l}} = \exp\left(-\log\frac{\sqrt{W^+_{j,l}}}{\sqrt{W^-_{j,l}}}\right) = \left(\frac{\sqrt{W^-_{j,l}}}{\sqrt{W^+_{j,l}}}\right) \ ,$$

and equivalently

$$e^{d_{j,l}} = \left(\frac{\sqrt{W^+_{j,l}}}{\sqrt{W^-_{j,l}}}\right) \ .$$

Thus, we can rewrite Eq. (8) as follows,

$$\begin{aligned} &\sum_l \sum_j b^j_l \left(\sqrt{W^+_{j,l}} - \sqrt{W^-_{j,l}}\right)^2 \\ &= \sum_l \max_{b\in\mathcal{B}} \sum_j b^j \left(\sqrt{W^+_{j,l}} - \sqrt{W^-_{j,l}}\right)^2 \ , \end{aligned} \tag{9}$$

where the last equality is the criterion applied by the algorithm to select b_l. □

We have thus shown the algorithm is guaranteed to decrease the loss on each iteration. A non-zero improvement is guaranteed whenever there exists a template in $\mathcal{B}$ whose predictability scores, which constitutes the criterion for choosing a template, is non-zero. It is easy to verify that the template choosing criterion,

$$\left(\sqrt{W^+_{j,l}} - \sqrt{W^-_{j,l}}\right)^2$$

is zero for all templates iff none of the templates can contribute to the decrease in the loss. This situation seldom happens in practice when the number templates is very large.

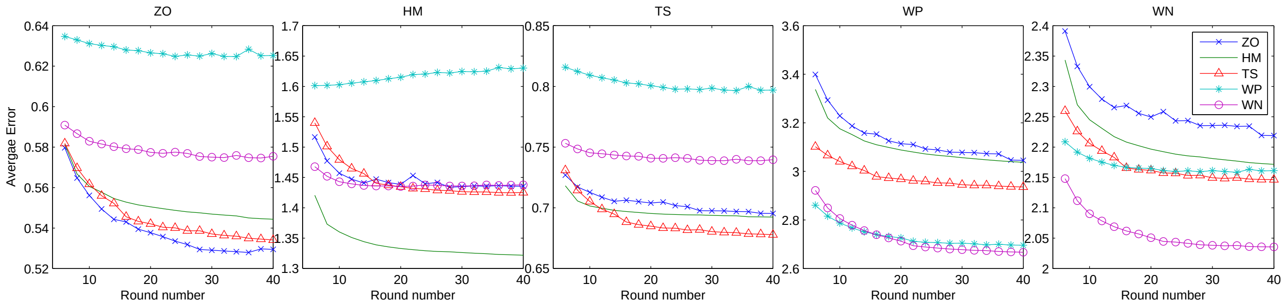


Figure 3: The covering-error on the training data as a function of the number of boosting iterations. Each graph depicts the performance of the classifiers trained with different covering while being evaluated using the same covering-error. The covering-errors used for evaluation are, from left to right, ZO, HM, TS, WP, and WN.

4 Experiments

In this section we describe experiments that we conducted in order to asses the performance of the AdaBoost.LC algorithm with various coverings. The task that we chose for our experiments is text categorization for which there exist publically available datasets which match our algorithmic setting. One of the main goal of the experiments is to demonstrate how proper adaptation of the covering to the task at hand affects the performance of the learning algorithm. We begin with a description of the datasets that are used in the experiments and our preprocessing of the data. We then describe the evaluation measures and the different coverings that we tested and conclude with a discussion of the results we obtained.

Datasets and data representation: We tested the AdaBoost.LC algorithm on the Reuters Corpus Volume 1 (RCV1). This corpus is available from http://about.reuters.com/researchandstandards . The corpus consists of approximately $800,000$ news articles. The articles were collected over a period of 12 months. The articles in the corpus were hand labeled from a set of 103 labels with most articles being associated with 2-4 relevant labels. About three quarters of the articles, from August 1996 through early May 1997, were used to train the classifiers (about $590,000$ documents). The rest of the articles were used to evaluate the generalization performance. Each document was represented by a vector of terms. The weight of terms not appearing in a document was set to zero. The weight of each term appearing in the corpus was determined using the pivoted length normalization method from [9]. We now briefly describe the pivoted length normalization.

Let d_i^l denote the number of times a term, whose index is l, appears in document i. We denote the number of unique terms appearing in document i by m_i, that is, $m_i = |\{l : d_i^l > 0\}|$. Let u_l denote the number of times the term l appears throughout the corpus. Finally, we denote the total number of documents in the corpus by m. The *idf* (inverse document frequency) weight of a term l is $\log(m/u_l)$. The average frequency of the terms in document i is denoted by $\text{avg}\left(d_i^l\right) = \frac{\sum_l d_i^l}{m_i}$, and the average frequency of the number of unique terms in a document by $\text{avg}(m_i) = \frac{1}{m}\sum_{i=1}^m m_i$. The *tf* (term frequency) is now defined as

$$\left(\frac{1+\log(d_i^l)}{1+\log\left(\text{avg}(d_i^l)\right)}\right) \Big/ \left(1.0-\text{slope}+\text{slope}\cdot\frac{m_i}{\text{avg}(m_i)}\right),$$

where *slope* was set to 0.3 as in [9]. Feature selection was used to select a subset of the terms appearing with highest Rocchio score (as adapted in [6]) for each topic. We selected the 100 highest ranked terms for each topic, and ended up with $6,270$ unique terms. Each document is therefore represented as a 6270 dimension vector where the j'th entry of the vector corresponds to the *tf-idf* value of the j'th selected term. Note that if a term does not appear in a document, its *tf-idf* value is zero. The average number of nonzero terms per document is 22.76. Recall that AdaBoost.LC is given a set of templates which we denote by $\mathcal{B}$. Since we performed feature selection prior to running the boosting algorithm we chose the set $\mathcal{B}$ to consist of a single template which contains all features/hypotheses. In the case of binary classification, this choice of a single template vector is also known as the *parallel* update [1].

Evaluation measures: Since many of the multilabel text categorization algorithms output an ordering of the labels, we used two sets of performance measures. First, we evaluated the AdaBoost.LC algorithm with respect to different coverings. We also used a set of performance measures commonly employed in document retrieval systems. Before we formally describe these measures, we need the following definitions. Focusing on a single instance (document) $\mathbf{x}$, its label $\mathbf{y}$, and a set of classifiers, the *rank* of topic r, denoted

$rank(\mathbf{x}, r)$, is defined as the rank, or position, of the topic r in the list of topics sorted according to the values attained by applying the classifiers to $\mathbf{x}$. Thus, the rank order of the top ranked topic is 1 while the rank of the lowest ranked topic is k. The *precision* at r is defined as the number of topics from the set of relevant topics, as indicated by $\mathbf{y}$, whose rank is at most r divided by the position r. The evaluation measures that we used for a given document $\mathbf{x}$ with label $\mathbf{y}$ are defined as follows.

OneErr The one-error (abbreviate OneErr) generalizes the zero-one error for the top ranked label. It designates whether the top ranked label, l, is relevant. Thus it takes the values of 0 when the $y^l = 1$, and is 1 otherwise.

Coverage The coverage loss measures how far down the list of labels we need to go in order to include all relevant labels, namely, Coverage $= \max_{r:y^r=1} rank(\mathbf{x}, r) - 1$.

AvgP The average precision (abbreviated AvgP) evaluates, as the name implies, the average precision taken at positions according to relevant topics.

$$\text{AvgP} = \frac{\sum_{r:y^r=1} \frac{\left|\left\{r' : y^{r'}=1 \wedge rank(\mathbf{x},r') \leq rank(\mathbf{x},r)\right\}\right|}{rank(\mathbf{x},r)}}{|\,\{r : y^r = 1\}\,|} \ .$$

Coverings: We next move our focus of attention to the different coverings employed by AdaBoost.LC. We used 5 different coverings in the training phase, each of the coverings defines an error model and translates into a different classifier. The first three coverings are illustrated in Fig. 1.

Zero-One (ZO) The zero-one covering ties all labels into a single set. This covering can be viewed as an extension of the binary zero-one error to the multilabel case.

Hamming-Distance (HM) The Hamming distance covering places each label in a separate set. This covering translates to an error function which is equivalent to the Hamming distance between the predicted and correct set of labels.

Two-Sets (TS) The two sets covering puts together all positive labels in a single set, and all negative labels in a second set. This covering assigns a unit loss when failing to predict *any* of the positive labels as positive, and, equivalently, when failing to predict any of the negative labels as negative.

Weighted-Positive (WP) The weighted positive covering ties together all positive labels in a single set. This set is then replicated several times (in our experiments we used weight of 6). Each of the remaining negative labels are placed in a separate set. Thus, failing to predict any positive label correctly results in a preset penalty, while the number of failed negative predictions are added to the penalty linearly.

Weighted-Negative (WN) The weighted negative covering is similar to the weighted positive covering, with the roles of positive and negative reversed. In our experiments we used a replication factor of 4.

We finally focus our attention on the empirical results achieved by the different label covering approaches. In order to fully grasp the effect of the covering on the algorithm, we evaluated each of the classifiers trained with the coverings above with respect to all coverings. While performing both the training and testing phases using the same covering is expected to yield the optimal result, the quantitative difference emphasizes the importance of a proper selection. We especially note that the Hamming-Distance covering can be viewed as an adaptation of AdaBoost.MH to the log-loss function. Since AdaBoost.MH is a rather popular and thoroughly used multilabel extension of AdaBoost [8], it is particularly interesting to compare the performance of the different label covering to the Hamming-Distance covering.

Let us first examine the behavior of the algorithm with respect to different coverings during the training phase. We ran AdaBoost.LC using the five coverings described above on the training data. After each boosting iteration, we calculated the error rate of all five instantiations with respect to the different covering errors. The goal here is to verify that the log-loss relaxation serves as the correct proxy for the label covering error. In Fig. 3 we show five graphs each of which corresponds to evaluation using one of the five coverings. It is clear from the graphs that there is a one-to-one correspondence between the instantiations which results in the best training error and the covering used for evaluation. For instance, when evaluating the performance during training using the WN covering error (right-most figure), the best instantiation of AdaBoost.LC is indeed the one trained using the WN covering and its performance exceeds all other four instantiations by at least 0.2 units, which is overwhelmingly statistically significant. Therefore, we can conclude that employing a specific covering for training indeed plays an important role in the performance of the algorithm. A very similar phenomenon is exhibited on test data. Last, we would like to note that the convergence rate is rather fast and after about 20 iterations the performance reaches an asymptote.

We next focus on the performance of the different instantiations of AdaBoost.LC on test data. In Fig. 4 we show the results of the five AdaBoost.LC instanti-

train \ *test*	ZO	HM	TS	WP	WN	OneErr	Coverage	AvgP
ZO	**0.567**	1.587	0.759	3.300	2.407	0.100	4.493	0.865
HM	0.579	**1.492**	0.757	3.297	2.365	**0.091**	4.378	0.870
TS	0.574	1.599	**0.747**	3.209	2.356	0.094	4.376	0.870
WP	0.652	1.792	0.857	**3.022**	2.403	0.094	**4.177**	**0.872**
WN	0.610	1.614	0.817	3.051	**2.314**	0.094	4.227	0.872

Figure 4: Average test losses: each row represents the test loss using a different covering for training while being evaluated with respect to all the losses defined by the various label coverings.

ations with respect to eight evaluation measures. The first five measures correspond to the coverings used during training while the last three are commonly used evaluation measures in IR and do not correspond to any of the coverings used for training. Each row in the table corresponds to a specific covering instantiation used for training while being evaluated on all the eight performance measures. Thus, a column corresponds to the evaluation of a single performance measure on all of the instantiations of AdaBoost.LC. It is clear from the table that for each of the first five performance measures, which are defined by a label covering, the best performing AdaBoost.LC instantiation is the one that was trained using the same label covering. The differences between the results appearing on the diagonal and all the off-diagonal results were found to be statistically significant when using the paired Wald test with a p-value of at most 0.01. However, there is no clear winner when evaluating the different label coverings using IR evaluation measure. The instantiation trained with the HM covering achieves the best performance with respect to One-Error while the instantiation trained with the WP covering is the best performer with respect to Coverage and Average-Precision. Moreover, both instantiations trained with the WP and WN coverings perform substantially better than the three other coverings for the latter two measures. Since Coverage and AvgP are highly non-linear performance measures without any close resemblance to any of the coverings, it is difficult to draw any general conclusion on the usage of any covering as a proxy for obtaining good performance with respect to IR-related losses. Devising approximate IR measures by a label covering is left to future research.

One conclusion that we can draw is that popular multilabel constructions such as AdaBoost.MH are not likely to be suitable for decision settings which correspond to complex label coverings. It is therefore essential to tailor the specific covering to the requirements of the application on hand. To recap, the generality of our construction enables us to cope with a rather broad spectrum of requirements while retaining the simplicity and power of AdaBoost.

References

[1] M. Collins, R.E. Schapire, and Y. Singer. Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 47(2/3):253–285, 2002.

[2] K. Crammer and Y. Singer. A new family of online algorithms for category ranking. *Jornal of Machine Learning Research* 1025–1058, 2003

[3] O. Dekel, C. Manning, and Y. Singer. Log-linear models for label ranking. In *Advances in Neural Information Processing Systems 16*, 2003.

[4] A. Elisseeff and J. Weston. A kernel method for multi-labeled classification. In *Advances in Neural Information Processing Systems 14*, 2001.

[5] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[6] D. J. Ittner, D. D. Lewis, and D. D. Ahn. Text categorization of low quality images. In *Symposium on Document Analysis and Information Retrieval*, pages 301–315, 1995.

[7] G. Lebanon and J. Lafferty. Conditional models on the ranking poset. In *Advances in Neural Information Processing Systems 15*, 2002.

[8] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):1–40, 1999.

[9] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Res. and Dev. in Information Retrieval*, pages 21–29, 1996.

[10] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems 17*, 2003.

[11] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. *Proc. of 21st Intl. Conf. on Machine Learning*, 2004.