Consistent Label Tree Classifiers for Extreme Multi-Label Classification

Kalina Jasinska KJASINSKA@CS.PUT.POZNAN.PL
Institute of Computing Science, Poznań University of Technology
Piotrowo 2, 60-965 Poznań, Poland

Krzysztof Dembczyński KDEMBCZYNSKI@CS.PUT.POZNAN.PL
Institute of Computing Science, Poznań University of Technology
Piotrowo 2, 60-965 Poznań, Poland

Abstract

One way of solving multi-label classification is to reduce the original problem to a number of simple binary problems, for which we can use any existing learning algorithm. The simplest reduction for multi-label classification under Hamming loss is the so-called binary relevance (BR), in which an independent binary classifier is trained for each label. This approach is statistically consistent, but its train and test time complexity is linear (in the number of labels) which can be too expensive in many applications. Many reduction approaches exist that improve over the complexity of BR, but quite often they sacrifice statistical consistency. We introduce and analyze two label-tree-based reduction techniques that are consistent and significantly improve the test time complexity. As the main theoretical result we prove the regret bounds for these algorithms. Moreover, the empirical studies show competitive results in comparison to the state-of-the-art methods.

1. Introduction

Nowadays learning problems are characterized not only by a large number of examples and features, but also by a large number of labels. In these problems we often deal with tens or hundreds of thousands (Deng et al., 2009), or even millions of labels (Agrawal et al., 2013). Applications of that kind can be found in image classification (Deng et al., 2011), text document classification (Dekel & Shamir, 2010), on-line advertising (Beygelzimer et al., 2009a), and video recommendation (Weston et al., 2013).

Proceedings of the 31st International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37. Copyright 2015 by the author(s).

The simplest reduction for multi-label classification under Hamming loss is the so-called binary relevance (BR), in which an independent binary classifier is trained for each label. This approach is statistically consistent, i.e., by solving the resulting binary classification problems optimally, we get an optimal solution for the multi-label problem. However, the linear (in the number of labels) train and test time complexity of BR can be too expensive in many applications. Many reduction approaches exist that improve over the complexity of BR, but quite often they sacrifice statistical consistency. The most popular approaches are PLST (Tai & Lin, 2012), compressed sensing (Hsu et al., 2009), and robust Bloom filters (R-BF) (Cisse et al., 2013).

In this paper we introduce and analyze two algorithms, referred to as probabilistic trees (PT) and BR-trees, that are based on the label tree approach which organizes classifiers in a tree structure. Classification of a test example relies then on a sequence of decisions made by these classifiers, leading the test example from the root to the leaves of the tree. The label tree approaches aim mainly in improving the test time complexity and belong to the most efficient ones for the problems with a large number of labels (Beygelzimer et al., 2009b; Bengio et al., 2010; Deng et al., 2011). It is usually assumed that learning can be more costly and performed off-line (Bengio et al., 2010). This assumption is reasonable in many large-scale machine learning applications in which classification/test time is of the main interest. A similar assumption is also made in other domains, like in databases, where index structures are built for efficient data access, or in information retrieval, where expensive learning of hash codes is used to optimize their performance.

Both algorithms introduced in this paper significantly improve the test time complexity over BR. In the best case scenario, we can get logarithmic complexity in the number of labels. The first algorithm, PT, can also be faster in training than BR, while the second algorithm, BR-trees needs two times more time than BR for training. The space complexity of both algorithms is twice the complexity of

BR. From statistical point of view, the main advantage of the introduced algorithms is their statistical consistency. As the main theoretical contribution of this paper we prove the regret bounds for these two algorithms. We also perform a large experiment, in which we show that our approach is competitive to R-BF in terms of time complexity and predictive performance under the Hamming loss, micro- and macro-F-measure.

There are only few related approaches suited for multi-label classification. A similar tree structure is considered in (probabilistic) classifier chains (Read et al., 2009; Dembczyński et al., 2010) and condensed filter trees (Li & Lin, 2014), but the leaf nodes in these approaches correspond to label combinations, not to single labels as in our case. Therefore the test time complexity of these approaches is linear in the number of labels. PT can be in fact treated as a specific variant of conditional probability trees (Beygelzimer et al., 2009b) or probabilistic classifier chains suited for Hamming loss minimization. PT are also similar to Homer (Tsoumakas et al., 2008), which uses the same transformation of training examples, but does not have probabilistic interpretation. We discuss thoroughly the difference between PT and Homer further in the paper. BR-trees, in turn, adapt to some extent the underlying idea of filter trees (Beygelzimer et al., 2009a) to the problem of Hamming loss minimization. There is, however, no tournament between labels, as they are all predicted simultaneously. There is also a similarity of our work to hierarchical multi-label classification (Vens et al., 2008), but in our case we assume that the hierarchical structure is not given and do not consider any hierarchy-based loss function.

The paper is organized as follows. We formally define the problem in Section 2. Section 3 defines the label tree classifiers in a more precise way. PT are introduced and analyzed in Section 4, while BR-trees in Section 5. Section 6 presents the experimental results. Section 7 concludes the paper.

2. Problem statement

Let $\mathcal{L} = \{\lambda_1, \lambda_2, \dots, \lambda_m\}$ be a finite set of labels. In multi-label classification each instance $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ denotes a feature space, is (non-deterministically) associated with a subset of labels $\mathcal{L}_+ \in 2^{\mathcal{L}}$; this subset is often called the set of relevant (positive) labels, while the complement $\mathcal{L} \setminus \mathcal{L}_+$ is considered as irrelevant (negative) for $\boldsymbol{x}$. We identify a set $\mathcal{L}_+$ of relevant labels with a binary vector $\boldsymbol{y} = (y_1, y_2, \dots, y_m)$, in which $y_i = 1$ iff $\lambda_i \in \mathcal{L}_+$. We refer to vector $\boldsymbol{y}$ as a label vector. The set of all possible label combinations is denoted by $\mathcal{Y} = \{0,1\}^m$ and its cardinality is 2^m. We assume that observations $(\boldsymbol{x}, \boldsymbol{y})$ are generated independently and randomly according to a probability distribution $P(\boldsymbol{x}, \boldsymbol{y})$ on $\mathcal{X} \times \mathcal{Y}$.

The goal in multi-label classification is to train a classifier $\boldsymbol{h}(\boldsymbol{x})$ whose predictions $\hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_m)$ are as close as possible to true label vectors $\boldsymbol{y}$ of instances $\boldsymbol{x} \in \mathcal{X}$. By $\boldsymbol{h}_i(\boldsymbol{x})$ we denote the prediction of $\boldsymbol{h}$ for the i-th label.

There is a multitude of loss functions that can be considered for assessing the predictive performance of a multi-label classifier. In the following, we focus on Hamming loss defined as:

$$\ell_H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{m}\sum_{i=1}^{m} [\![y_i \neq \hat{y}_i]\!]$$

where $[\![\cdot]\!]$ is the indicator function. Remark that $[\![y_i \neq \hat{y}_i]\!]$ is the typical 0/1 loss used in binary classification, which we denote by $\ell_{0/1}$. The expected loss, also referred to as *risk*, is defined by:

$$L(\boldsymbol{h}, P) = \mathbb{E}\left[\ell(\boldsymbol{Y}, \boldsymbol{X})\right] = \int \ell(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x}))\, dP(\boldsymbol{x}, \boldsymbol{y})\,,$$

For the Hamming loss, the risk gets the following form:

$$\begin{aligned} L_H(\boldsymbol{h}, P) &= \frac{1}{m}\int \sum_{i=1}^{m} \ell_{0/1}(y_i, \boldsymbol{h}_i(\boldsymbol{x})) dP(\boldsymbol{x}, \boldsymbol{y}) \\ &= \frac{1}{m}\sum_{i=1}^{m} L_{0/1}(\boldsymbol{h}_i, P_i)\,, \end{aligned}$$

i.e., it is a sum of m binary classification risks.

The optimal classifier, commonly referred to as *Bayes classifier*, minimizes the risk. It suffices to minimize the risk pointwise:

$$\boldsymbol{h}^*(\boldsymbol{x}) = \operatorname*{arg\,min}_{\boldsymbol{h} \in \mathcal{Y}} L(\boldsymbol{h}, P \,|\, \boldsymbol{x})\,.$$

We note that $\boldsymbol{h}^*$ is in general not unique. However, the risk of $\boldsymbol{h}^*$, denoted $L^*(P)$, is unique, and is called the *Bayes risk*. It is easy to check that the Bayes classifier for Hamming loss $\boldsymbol{h}_H^* = (\boldsymbol{h}_1^*, \boldsymbol{h}_2^*, \dots, \boldsymbol{h}_m^*)$ is given by:

$$\boldsymbol{h}_i^*(\boldsymbol{x}) = \operatorname*{arg\,max}_{b \in \{0,1\}} P(y_i = b \,|\, \boldsymbol{x})$$

The regret of $\boldsymbol{h}$ on $P(\boldsymbol{X}, \boldsymbol{Y})$ is defined as:

$$\mathrm{reg}(\boldsymbol{h}, P) = L(\boldsymbol{h}, P) - L^*(P)$$

The goal is to train a classifier $\boldsymbol{h}$ with a small regret, ideally equal to zero. Obviously we cannot measure the regret, but we can analyze learning and classification algorithms in terms of regret. We say that an algorithm is statistically consistent (or calibrated) if its regret can be reduced down to zero.

The regret for the Hamming loss can be given as an average over m regrets of 0/1 loss:

$$\mathrm{reg}_H(\boldsymbol{h}, P) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{reg}_{0/1}(\boldsymbol{h}_i, P_i),$$

where:

$$\mathrm{reg}_{0/1}(\boldsymbol{h}_i, P) = L_{0/1}(\boldsymbol{h}_i, P_i) - L^*(P_i) = \\ = \int \underbrace{\ell_{0/1}(y_i, \boldsymbol{h}_i(\boldsymbol{x})) - \ell_{0/1}(y_i, \boldsymbol{h}_i^*(\boldsymbol{x})) dP(y_i \,|\, \boldsymbol{x})}_{\mathrm{reg}_{0/1}(\boldsymbol{h}_i, P \,|\, \boldsymbol{x})} dP(\boldsymbol{x})$$

The pointwise regret $\mathrm{reg}_{0/1}(h_i, P \,|\, \boldsymbol{x})$ can be readily expressed by:

$$\mathrm{reg}_{0/1}(\boldsymbol{h}_i, P \,|\, \boldsymbol{x}) = P(y_i = \boldsymbol{h}_i^*) - P(y_i = \boldsymbol{h}_i). \quad (1)$$

The above analysis suggests that one can solve a multi-label problem by reducing it to a series of m binary classification tasks, in which each $\boldsymbol{h}_i$ corresponds to a simple binary classifier h. Such an approach is usually referred to as binary relevance (BR). Let us denote the binary relevance classifier by BR and its prediction for i-th label by BR_i. Its regret is then:

$$\mathrm{reg}_H(\mathrm{BR}, P) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{reg}_{0/1}(\mathrm{BR}_i, P_i)\,. \quad (2)$$

The remaining problem is to build a consistent binary classifier h. The popular choice is to use algorithms that minimize margin-based loss functions, for example, logistic loss. Let $y \in \{0, 1\}$ be a label to be predicted, and q the estimate of $p = P(y = 1 \,|\, \boldsymbol{x})$. Logistic loss is then defined as:

$$\ell_{\log}(y, q) = y \log(q) + (1 - y) \log(1 - q). \quad (3)$$

The conditional regret of logistic loss is given by:

$$\mathrm{reg}_{\log}(q, P \,|\, \boldsymbol{x}) = p \log \frac{p}{q} + (1 - p) \log \frac{1-p}{1-q} \quad (4)$$

Based on the results from (Bartlett et al., 2006), we can upper bound $\mathrm{reg}_{0/1}(h, P)$ by a function of $\mathrm{reg}_{\log}(q, P)$.

Concluding the above discussion, we note that by solving each binary problem optimally, we get zero regret for the multi-label problem. Thus, BR is consistent for Hamming loss. Unfortunately, computational complexity of this reduction is linear in the number of labels for both training and classification. Below we consider two other algorithms that are not only consistent, but also they improve computational complexity of the classification procedure.

3. Label Tree Classifiers

We consider a label tree classifier that uses a tree T of classifiers to compute prediction for a test instance $\boldsymbol{x}$. In the following we assume that the tree is binary and its structure is given prior to learning classifiers. For simplicity, we assume that the number of labels is always a power of 2. Then, for $m = 2^k$, the depth of the tree is $k+1$. In general, there are $2m - 1$ classifiers, one classifier in each node.

The root of the tree T is denoted by $r(T)$. The leaves of tree T correspond to labels. We denote a set of leaves of a (sub)tree rooted in node n as $L(n)$. If n is a root of T then we write L. Similarly, we denote a set of internal nodes of the tree rooted in node n as $N(n)$. If n is a root of T then we write N. We assume that a root node is also an internal node, i.e., $n \in N(n)$. The parent node of n is denoted by $\mathrm{pa}(n)$, and the left and the right child by $\mathrm{l}(n)$ and $\mathrm{r}(n)$, respectively. The path from the root $r(T)$ to the i-th leaf is denoted by $\mathrm{Path}(i)$.

An internal node classifier decides whether to go down the tree to both child nodes. A leaf node classifier makes a final decision regarding the prediction of a label associated with this leaf. These two aspects make the main difference in comparison to the label tree algorithms used for multi-class classification. If classes are mutually exclusive, then we can follow only one path to a leaf node in the tree. This is not a case of multi-label classification.

To assign label λ_i to a test instance $\boldsymbol{x}$ all classifiers on $\mathrm{Path}(i)$ need to pass $\boldsymbol{x}$ down to the i-th leaf node. Then, in the leaf node, a final decision is made whether λ_i is assigned or not to $\boldsymbol{x}$. In other words, the prediction procedure starts with a vector of all zeros and traverses a tree from a root to leaves to predict positive labels for a given test instance. This procedure can be much cheaper than m independent queries to BR, since many subtrees are not explored. If there is only one label to predict than in the optimal scenario the label tree needs to call $2k - 1$ classifiers corresponding to a path from the root to a leaf plus all sibling node classifiers. If the tree predicts two positive labels, than the cost will be the same if these labels are direct siblings. Of course, in the worse-case scenario the entire tree might be explored, but then there is no more than $2m - 1$ calls required. In case of sparse label sets, the label tree classifiers can significantly speed up the classification procedure. The expected cost of this prediction procedure depends on the tree structure and accuracy of binary classifiers.

4. Probabilistic Trees

Probabilistic trees (PT) use a path from a root to the i-th leaf to compute the conditional probability $P(y_i \,|\, \boldsymbol{x})$. In other words, we divide the process of estimating $P(y_i \,|\, \boldsymbol{x})$

to $k+1$ stages, each corresponding to a level of the tree T. In each node j, we associate a label z_j with a training instance $\boldsymbol{x}$ such that:

$$z_j = \begin{cases} 1, & \text{if } \sum_{i \in L(j)} y_i \geq 1\,, \\ 0, & \text{otherwise.} \end{cases}$$

Recall that $L(j)$ is a set of all leaves of a subtree with the root in the j-th node. Notice that in leaf nodes the labels z_i, $i \in L$, correspond to original labels λ_i.

Consider the leaf node i and the path from the root to this leaf node. Using the chain rule of probability, we can express $P(y_i = 1 \,|\, \boldsymbol{x})$ in the following way:

$$P(y_i = 1 \,|\, \boldsymbol{x}) = \prod_{j \in \mathrm{Path}(i)} P(z_j = 1 \,|\, z_{\mathrm{pa}(j)} = 1, \boldsymbol{x})\,, \quad (5)$$

where for the root node r_T we have $P(z_{r_T} = 1 \,|\, z_{\mathrm{pa}(r_T)} = 1, \boldsymbol{x}) = P(z_{r_T} = 1 \,|\, \boldsymbol{x})$.

The learning algorithm of PT is given in Algorithm 1. Let $\mathcal{S}$ be a training set of multi-label examples $(\boldsymbol{x}, \boldsymbol{y})$. To learn classifiers in all nodes of tree T we need to properly filter training examples $(\boldsymbol{x}, \boldsymbol{y})$ to estimate $P(z_j = 1 \,|\, z_{\mathrm{pa}(j)} = 1, \boldsymbol{x})$. Moreover, we need to use a learning algorithm which trains a probability estimation classifier. We denote by $q_j(\boldsymbol{x})$ a probability estimation function trained by such a classifier in node j. In the theoretical analysis of PT we will focus on logistic loss minimization to obtain q_j.

The computational complexity of learning PT can be expressed in terms of the number of nodes in which an original training example $(\boldsymbol{x}, \boldsymbol{y})$ is used. Since the training example is used in a node j only if j is the root or $z_{\mathrm{pa}(j)} = 1$, this number is upper bounded by $s(2k+1)$, where s is the number of positive labels in $\boldsymbol{y}$. Therefore PT can be faster than BR. Notice also that learning can be performed simultaneously for all nodes.

Prediction with probabilistic trees relies on estimating (5) by traversing the tree from the root to leaf nodes. However, if the intermediate value of this product in node j, denoted by p, is smaller than a given threshold t, then the subtree starting with this node j is no longer explored. Formally, we can express this by a function $h_j(\boldsymbol{x}) = \mathrm{sgn}(p \geq t)$. For minimization of Hamming loss it is reasonable to take $t = 0.5$. The procedure is given in Algorithm 2. For sake of completeness, we shortly describe this procedure. We start with setting $\hat{\boldsymbol{y}} = \mathbf{0}_m$. In order to traverse a tree we initialize a queue Q and add the root node r_T to it. In the while loop we iteratively pop a node from Q and compute p and $h_j(\boldsymbol{x})$. For $h_j(\boldsymbol{x}) = 1$, we either set $\hat{y}_j = 1$ if j is a leaf, or add child nodes of j to Q, otherwise. If Q is empty, we stop the search and return $\hat{\boldsymbol{y}}$. With properly estimated probabilities, the algorithm will not explore a large part of the tree.

Algorithm 1 Learning of a Probabilistic Tree

```
input: a label tree $T$, a learning algorithm $A$, and a training set $\mathcal{S}$
output: a set of probability estimation classifiers $\mathcal{Q}$
$\mathcal{Q} = \emptyset$
for each node $j \in T$ do
  $\mathcal{S}_j = \emptyset$
  for each training instance $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}$ do
    if $j$ is root or $\sum_{i \in L(\mathrm{pa}(j))} y_i \geq 1$ then
      $z_j = [\![ \sum_{i \in L(j)} y_i \geq 1 ]\!]$
      $\mathcal{S}_j = \mathcal{S}_j \cup (\boldsymbol{x}_i, z_j)$
    end if
  end for
  $q_j = A(\mathcal{S}_j)$, $\mathcal{Q} = \mathcal{Q} \cup q_j$
end for
return a set of probability estimation classifiers $\mathcal{Q}$.
```

Algorithm 2 Prediction with a Probabilistic Tree

```
input: a label tree $T$, a set of probability estimation classifiers $\mathcal{Q}$, a test example $\boldsymbol{x}$, a threshold $t$
input: a label vector $\hat{\boldsymbol{y}}$
$\hat{\boldsymbol{y}} = \mathbf{0}_m$, $Q = \mathit{initializeQueue}()$, $\mathrm{add}(Q, (\mathrm{root}, 1))$
while $Q \neq \emptyset$ do
  $(j, p) = \mathrm{pop}(Q)$
  $p = p \cdot q_j(\boldsymbol{x})$
  $h_j(\boldsymbol{x}) = \mathrm{sgn}(p \geq t)$
  if $h_j(\boldsymbol{x}) = 1$ then
    if $j$ is a leaf node then
      $\hat{y}_j = 1$
    else
      $\mathrm{add}(Q, (\mathrm{l}(j), p))$, $\mathrm{add}(Q, (\mathrm{r}(j), p))$
    end if
  end if
end while
return $\hat{\boldsymbol{y}}$.
```

As the main theoretical result we prove the regret bound for PT. This regret bound is expressed in terms of the logistic regret (4) averaged over all m paths from the root node to leaves of T.

Theorem 4.1. *Let* PT *be a probabilistic tree classifier based on label tree T with each node associated with a probability estimation function q_j. Moreover, let $\mathrm{reg}_{\log}(q, P_T)$ be the average logistic regret over m paths from the root to the leaves in tree T:*

$$\mathrm{reg}_{\log}(q, P_T) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j \in \mathrm{Path}(i)} \mathrm{reg}_{\log}(q_j, P_j)\,.$$

where P_j is distribution induced in the j-th internal node. For any distribution P, label trees T and PT*,*

$$\mathrm{reg}_H(\mathrm{PT}, P) \leq \sqrt{2\mathrm{reg}_{\log}(q, P_T)}$$

The regret bound has a drawback that is computed over m paths from the root to leaves and therefore the regret of upper level nodes is counted multiple times, i.e., the error made higher in the tree has more impact. The square root on the right side of the bound results from using logistic loss and is unavoidable (Bartlett et al., 2006).

PT shares some similarities with Homer (Tsoumakas et al., 2008). Both algorithms use the same reduction of the training instances for the node classifiers. Classification of test examples is different, since Homer does not have probabilistic interpretation. It uses binary classifiers and output either 0 or 1 in each node of the tree. In consequence, the regret bound does not apply to this algorithm. Just to perform a simple analysis, assume that the probabilistic models in each node of the tree are perfectly trained and that each node classifier predicts 1 if $P(z_j = 1 \,|\, z_{\mathrm{pa}(j)} = 1, \boldsymbol{x}) > 0.5$ (as typically in binary classification). Then, we predict label λ_i to be relevant, if $P(y_i = 1 \,|\, \boldsymbol{x}) > 0.5^{k+1}$. This is certainly a wrong strategy for Hamming loss, for which the optimal threshold is 0.5. The authors of Homer have remarked that their algorithm may perform worse under Hamming loss, but gets better results in terms of F-measure. This is not surprising, as the optimal threshold for the F-measure is usually less than 0.5 (Zhao et al., 2013).

Note that in PT we can set threshold to any value. Moreover, it does not have to be the same in each node. By setting it appropriately we can obtain different thresholds for each label. In this way we can easily tune PT for the F-measure and other more complex performance measures. PT can also be easily suited for prediction of top labels if Q would be changed to a priority queue and the search stopped after a given number of top labels.

5. BR-trees

The second algorithm, we refer to as *BR-trees*, follows a bottom-up strategy. It first trains a regular BR, i.e., m independent binary classifiers, one for each label. These classifiers are then associated with the leaves of the label tree. In the next steps, internal node classifiers are trained over the predictions of BR. Each internal node classifier is trained to decide whether to explore ($h_n(\boldsymbol{x}) = 1$) or not ($h_n(\boldsymbol{x}) = 0$) its subtree. From this perspective, the structure of BR-trees is similar to hierarchical indexes known from database systems. The difference is that regular indexes are built on static data, not over classifiers that for each new test instance may predict different values.

The logic behind BR-trees can be explained in the following way. If BR predicts at least one positive label in the leaves of a given subtree, then the algorithm will expand the subtree. However, to reduce the computational complexity, we do not want to explore subtrees with no positive labels in leaves. There are several ways of training internal node classifiers to implement this logic. Below we consider one of them that is characterized by good theoretical guarantees in terms of predictive and computational performance.

Algorithm 3 Learning of BR-trees

```
input: a label tree $T$, a learning algorithm $A$, and a training set $\mathcal{S}$
output: a set of binary classifiers $\mathcal{H}$
$\mathcal{H} = \mathrm{BR}(\mathcal{S})$ {train BR}
for each internal node $n$ in order from leaves to root do
  $\mathcal{S}_n = \emptyset$
  for each training instance $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}$ do
    $v_n = v_{\mathrm{l}(n)} \hat{z}_{\mathrm{l}(n)} + v_{\mathrm{r}(n)} \hat{z}_{\mathrm{r}(n)}$
    $(\boldsymbol{x}, \boldsymbol{y}) \to (\boldsymbol{x}, z_n = [\![ v_n > 0 ]\!], w_n = \max(1, v_n))$
    $\mathcal{S}_n = \mathcal{S}_n \cup (\boldsymbol{x}, z_n, w_n)$
  end for
  $h_n = A(S_n)$, $\mathcal{H} = \mathcal{H} \cup h_n$
end for
return a set of classifiers $\mathcal{H}$.
```

For each example in an internal node n we compute the following recursive value:

$$v_n = v_{\mathrm{l}(n)} \hat{z}_{\mathrm{l}(n)} + v_{\mathrm{r}(n)} \hat{z}_{\mathrm{r}(n)}$$

where $\hat{z}_n = h_n(\boldsymbol{x})$, and $v_l(\boldsymbol{x}) = 1$ for each leaf node $l \in L$. It is easy to see that v_n corresponds to the number of positive labels returned by BR in the leaves accessible from n (i.e., for which exists a path with all internal node classifiers predicting 1). We train a classifier $h_n(\boldsymbol{x})$ in internal node n using importance-weighted training examples of the form:

$$(\boldsymbol{x}, z_n = [\![v_n > 0]\!], w_n = \max(1, v_n))$$

where z_n is the output variable and w_n the weight of the training example. In other words, examples $(\boldsymbol{x}, z_n, w_n)$ are coming from a new distribution P_n induced by our procedure in each node n of the tree.

The learning algorithm is summarized in Algorithm 3. Let $\mathcal{S}$ be a training set of multi-label instances $(\boldsymbol{x}, \boldsymbol{y})$. We learn $2m - 1$ classifiers, one in each node of the tree. In the i-th leaf node, we use a training set consisting of examples $(\boldsymbol{x}, y_i)$, while in each internal node n a training set of examples $(\boldsymbol{x}, z_n, w_n)$. Basically, we change only the labels and weights of the examples and the features remain unchanged. Since each example is used $2m - 1$ times, the overall cost of learning is approximately twice the cost of BR. The benefits, however, are in the classification procedure.

The classification procedure for BR-trees is straightforward and resembles the procedure for PT. For sake of completeness, we shortly describe this procedure and

Algorithm 4 Prediction with BR-trees

```
input: a label tree $T$, a set of classifiers $\mathcal{H}$, a test example $\boldsymbol{x}$
output: a label vector $\hat{\boldsymbol{y}}$
$\hat{\boldsymbol{y}} = \mathbf{0}_m$, $Q = initializeQueue()$, $\mathrm{add}(Q, r_T)$
while $Q \neq \emptyset$ do
  $n = \mathrm{pop}(Q)$
  if $n$ is a leaf node then
    $\hat{y}_n = h_n(\boldsymbol{x})$
  else
    if $h_n(\boldsymbol{x}) = 1$ then
      $\mathrm{add}(Q, \mathrm{l}(n))$, $\mathrm{add}(Q, \mathrm{r}(n))$
    end if
  end if
end while
return $\hat{\boldsymbol{y}}$.
```

present the pseudo-code in Algorithm 4. We set $\hat{\boldsymbol{y}} = \mathbf{0}_m$. In order to traverse a tree we initialize a queue Q and add the root node r_T to it. In the while loop we iteratively pop a node from Q. If the node is a leaf then $\hat{\boldsymbol{y}}$ is updated by its prediction. Otherwise we check whether we should explore the subtree rooted in n. If $h_n(\boldsymbol{x}) = 1$, we add child nodes of n to Q. If Q is empty, we stop the search and return $\hat{\boldsymbol{y}}$.

We can easily notice that minimization of weighted false negatives in each node n,

$$\mathrm{WFN}(h_n \,|\, P_n) = \mathbb{E}_{P_n}\left[v_n[\![z_n = 1 \wedge \hat{z}_n = 0]\!]\right],$$

should lead to better predictive performance of BR-trees, as more positive leaves would be visited in the classification procedure. This statement can be formalized in the following theorem.

Theorem 5.1. *Let* BRT *be a BR-tree based on a label tree T with each node j associated with classifier h_j. Moreover, let* $\mathrm{WFN}(h, N)$ *be the weighted false negatives averaged over the number of leaves m in tree T:*

$$\mathrm{WFN}(h, N) = \frac{1}{m}\sum_{n \in N} \mathrm{WFN}(h_n, P_n).$$

where P_n is distribution induced in the n-th internal node. For any distribution P, label tree T and BRT*, we have*

$$\mathrm{reg}_H(\mathrm{BRT}, P) \leq \mathrm{WFN}(h, N) + \mathrm{reg}_H(\mathrm{BR}, P),$$

where $\mathrm{reg}_H(\mathrm{BR}, P)$ *is regret given in (2) of a binary relevance classifier* BR *containing leaf classifiers h_i, $i \in L$.*

This regret bound has an advantage that the error (weighted false negatives) is computed in each node only once. This would not be a case, if positive examples in internal nodes were not weighted by v_n. The regret bound of PT does not posses this advantages and the regret is summed over all paths, not the nodes as in this case.

Let us also notice that we can control computational performance of the classification procedure by minimizing false positives in each node n (for negative examples $w_n = 1$, so there is no need to use weighted false positives)

$$\mathrm{FP}(h_n \,|\, P_n) = \mathbb{E}_{P_n}\left[[\![z_n = 0 \wedge \hat{z}_n = 1]\!]\right],$$

These two quantities, WFN and FP, are minimized in each node by the training algorithms. In order to control the trade-off between WFN and FP, i.e., predictive and computational performance, we can assigned additional costs to positive and negative classes. Alternatively, we can control this trade-off in the classification procedure if we use a scoring-based binary classifier $h_n(\boldsymbol{x}) = \mathrm{sgn}(f_n(\boldsymbol{x}) > t)$. Then, the value of threshold t can be used to control the trade-off between WFN and FP. The tree search is expanded by lower values of t and narrowed for higher values. As we show in experiments, careful tunning of the threshold improves the performance measures adding only small additional costs of the search.

Table 1. Main characteristics and differences between datasets used in our experiment and the experiment described in (Cisse et al., 2013) (R-BF). The last line shows Hamming loss (%) of all-zero classifier (predicting all zeros)

	Reuters		Wikipedia	
	R-BF	Our	R-BF	Our
#labels	303	296	1000	933
#features		47236		1617899
#total examples	72334	72335	110530	108738
avg. cardinality	1.73	1.3	1.11	1.71
max cardinality	30	30	5	14
cardinality >2	20%	20%	10%	41%
all-zero		0.4392		0.1833

6. Experiments

We conduct experiments with a setting similar to the one presented in (Cisse et al., 2013) in order to compare the introduced algorithms with other methods, particularly with the robust Bloom filters (R-BF).[1] Experiments are performed on two large datasets, Reuters and Wikipedia, extracted from two readly available datasets as described in aforementioned article. Properties of obtained datasets and differences between them and the datasets described in (Cisse et al., 2013) are presented in Table 1.

We evaluate performance of the algorithms in terms of Hamming loss and the average number of calls to node classifiers during testing. We also compute micro- and macro-F-measure. As the base binary classifier we use L_2-regularized logistic regression. For each LR classifier, we

[1]Unfortunately, there is no publicly available code of R-BF.

tune the regularization constant on a validation set. We average the results over 10 random splits for train, validation, and test datasets, each respectively consisting of 50/25/25% of examples. The standard deviations of performance measures values are smaller than 1% of their average. Experiments were performed on a computer cluster using scikit-learn package (Pedregosa et al., 2011).

BR-trees and PT are tested on two binary tree structures, called naive and spectral. In the naive tree labels are assigned to nodes randomly. Spectral tree tries to assign labels to leaves in such a way that labels that co-occur most frequently have many common nodes on their paths to the root. It is built by splitting the set of labels into two branches recursively, minimizing the sum of similarities between labels in different clusters. In each iteration we solve a graph-cut problem for a graph defined as a label affinity matrix. The similarity measure is defined as the labels co-occurrence count. This optimization problem is solved with spectral clustering, using the algorithm described in (Ng et al., 2001). In first step we use the affinity matrix as mentioned before and in the clustering step we ensure that clusters do not differ in size more than one.

The results concerning prediction performance are given in Table 2, while the average number of calls to the base classifiers is given in Table 3. We compare BR-trees and PT with several algorithms. The results in the tables for both algorithms are reported for the standard value of threshold, $t = 0.5$. Later in this section we show how proper tuning of t improves the results. As a baseline we use BR and truncated-BR that uses only top most popular labels. We set top in a way corresponding to the number of calls of BR-trees and PT. In Table 1 we also report Hamming loss of a hypothetical all-zero classifier that predicts all zeros. The main competitors of our algorithms are PT with a classification procedure of Homer (Tsoumakas et al., 2008) and robust Bloom filters (R-BF). The results of R-BF are taken from (Cisse et al., 2013), where two variants of them have been tested. In the first one, the original problem is reduced to 150/240 binary classifiers respectively for Reuters/Wikipedia datasetes. The second one uses the reduction to 200/500 binary classifiers. Due to the differences between datasets presented in Table 1, we include the results of BR from the R-BF paper to allow a more fairly comparison between the algorithms. In Table 3 we also present the number of R-BF classifiers for a more comprehensive analysis. This number can be fairly compared with the average number of calls to node classifiers as in both experiments the base classifiers are the same.

From the results we observe that BR-trees and PT (both with standard threshold $t = 0.5$) get worse results with respect to Hamming loss and F-measures than BR and R-BF, but obtain a significant speed-up. From Table 3 we may observe that the prediction times are 10 times faster than R-BF, and 20 (Reuters) and 30-40 (Wikipedia) times faster than BR. Truncated-BR with a similar prediction complexity gets incomparably worse results. Let us also report that prediction times scale almost linearly with the average number of calls to classifiers. For example, prediction times for Reuters are 730.7s for BR and 42.99s for BR-tree.

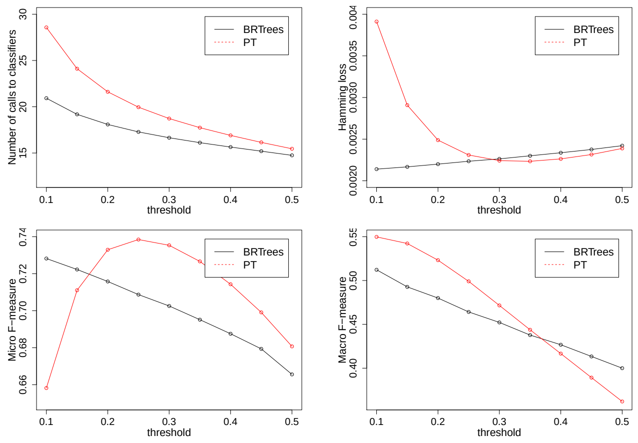

Figure 1. Impact of changing a threshold in the classification procedure. Top: the average number of calls to node classifiers (left) and Hamming loss (right). Bottom: micro-F (left) and macro-F (right).

BR-trees and PT perform quite similarly. It is worth to underline that PT outperforms PT-Homer in terms of Hamming loss due to a correct use of probabilistic inference. As expected, PT-Homer gets better results in terms of micro- and macro-F, but as shown below, PT with a tuned threshold outperforms PT-Homer in this regard as well. Interestingly, there is almost no difference between naive and spectral label trees. Proper construction of a label tree is certainly an interesting problem for future research.

To give a deeper insight in the nature of BR-trees and PT

Table 3. The average number of calls to base classifiers for a single test example.

Classifier	Reuters	Wikipedia
BR	296	933
truncated-BR 15	15	15
truncated-BR 35	35	35
BR-tree (spectral)	14.75	18.28
BR-tree (naive)	14.28	18.92
PT (spectral)	15.46	20.06
PT (naive)	15.37	21.38
PT-Homer (spectral)	18.80	26.41
PT-Homer (naive)	19.14	32.63
R-BF 150/240	150	240
R-BF 200/500	200	500

Table 2. Experimental results for Hamming loss (in %), micro- and macro-F-scores of binary relevance (BR), truncated-BR, BR-trees, probabilistic tress (PT), probabilistic trees with a classification procedure of Homer (PT-Homer), and robust Bloom filters (R-BF). For BR-trees and PT we use two variants of label trees: naive and spectral. R-BF are used with a reduction to 150/240 and 240/500 binary classifiers, respectively for both datasets, Reuters/Wikipedia. The results of R-BF are reported from (Cisse et al., 2013). We also include results of BR reported in this paper, because of small differences in datasets used in their and our experiments.

	Reuters			Wikipedia		
Classifier	Hamming loss	micro-F	macro-F	Hamming loss	micro-F	macro-F
BR	0.1991	75.58	56.07	0.1085	66.41	31.05
truncated-BR 15	0.3394	45.92	4.18	0.1550	40.27	1.21
truncated-BR 35	0.2951	57.12	9.25	0.1432	47.68	2.84
BR-tree (spectral)	0.2422	66.56	40.00	0.1166	60.60	22.06
BR-tree (naive)	0.2350	65.67	41.73	0.1211	57.76	18.38
PT (spectral)	0.2388	68.07	36.20	0.1151	63.25	20.78
PT (naive)	0.2396	67.94	36.65	0.1198	60.87	17.98
PT-Homer (spectral)	0.2684	70.69	52.21	0.1464	62.94	34.35
PT-Homer (naive)	0.2715	70.53	53.64	0.1696	58.93	31.83
BR in R-BF	0.2000	72.43	47.82	0.0711	55.96	34.70
R-BF 150/240	0.2100	71.31	43.44	0.0728	55.85	34.65
R-BF 200/500	0.2050	71.86	44.57	0.0705	57.31	36.85

we plot the results for different values of threshold t on the Reuters dataset (Figure 1). We vary the threshold from 0.1 to 0.5 (recall that results in Table 2 are given for $t = 0.5$). In case of BR-trees, we can see that prediction performance approaches the results of BR, but the costs of the classification procedure are still very low. For $t = 0.1$, BR-trees achieve Hamming loss of 0.214, micro-F of 72.81, and macro-F of 51.22 with less than 21 calls to classifiers per example on average, which is 12 times faster than BR. Moreover, this result is very close to R-BF 150 (worse on Hamming loss, but better in terms of F-measures), but requires much less time. As already mention, PT with a tuned threshold obtains better results than PT-Homer, and gets better micro- ad macro-F than R-BF being very close to BR under these measures.

7. Conclusions

We introduced two label tree classifiers for Hamming loss minimization in multi-label classification. These algorithms are characterized by very efficient classification procedures. Moreover, as the main theoretical contribution we proved the regret bound for both algorithms. We verified empirically both algorithms in a large experimental study, showing their potential for consistent and effective multi-label classification.

References

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*, May 2013.

Bartlett, P., Jordan, M., and Mcauliffe, J. Convexity, classification and risk bounds. *JASA*, 101:138–156, 2006.

Bengio, S., Weston, J., and Grangier, D. Label embedding trees for large multi-class tasks. In *NIPS*, pp. 163–171. Curran Associates, Inc., 2010.

Beygelzimer, Alina, Langford, John, Lifshits, Yury, Sorkin, Gregory B., and Strehl, Alexander L. Conditional probability tree estimation analysis and algorithms. In *UAI*, pp. 51–58, 2009a.

Beygelzimer, Alina, Langford, John, and Ravikumar, Pradeep D. Error-correcting tournaments. In *ALT*, pp. 247–262, 2009b.

Cisse, Moustapha M, Usunier, Nicolas, Artières, Thierry, and Gallinari, Patrick. Robust bloom filters for large multilabel classification tasks. In *Advances in NIPS 26*, pp. 1851–1859, 2013.

Dekel, O. and Shamir, O. Multiclass-multilabel learning when the label set grows with the number of examples. In *Artificial Intelligence and Statistics (AISTATS)*, 2010.

Dembczyński, K., Cheng, W., and Hüllermeier, E. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, pp. 279–286. Omnipress, 2010.

Deng, J., Dong, W., Socher, R., Li, Li-Jia, Li, K., and Li, Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 248—255, 2009.

Deng, J., Satheesh, S., Berg, A. C., and Li, Fei Fei F. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, pp. 567–575, 2011.

Hsu, D., Kakade, S., Langford, J., and Zhang, T. Multi-label prediction via compressed sensing. In *NIPS*, 2009.

Li, Chun-Liang and Lin, Hsuan-Tien. Condensed filter tree for cost-sensitive multi-label classification. In *ICML*, pp. 423–431, 2014.

Ng, Andrew Y., Jordan, Michael I., and Weiss, Yair. On spectral clustering: Analysis and an algorithm. In *Advances in NIPS*, pp. 849–856. MIT Press, 2001.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. Classifier chains for multi-label classification. In *ECML/PKDD*, pp. 254–269, 2009.

Tai, F. and Lin, H.-T. Multi-label classification with principal label space transformation. In *Neural Computat.*, volume 9, pp. 2508–2542, 2012.

Tsoumakas, G., Katakis, I., and Vlahavas, I. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 2008.

Vens, C., Struyf, J., Schietgat, L., Dzeroski, S., and Blockeel, H. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

Weston, Jason, Makadia, Ameesh, and Yee, Hector. Label partitioning for sublinear ranking. In *ICML*, volume 28, pp. 181–189, 2013.

Zhao, Ming-Jie, Edakunni, Narayanan, Pocock, Adam, and Brown, Gavin. Beyond Fano's inequality: Bounds on the Optimal F-Score, BER, and Cost-Sensitive Risk and Their Implications. *Journal of Machine Learning Research*, pp. 1033–1090, 2013.