Consistency of Probabilistic Classifier Trees

Krzysztof Dembczyński[1], Wojciech Kotłowski[1], Willem Waegeman[2], Róbert Busa-Fekete[3], and Eyke Hüllermeier[3]

[1] Poznań University of Technology, Poland
[kdembczynski,wkotlowski]@cs.put.poznan.pl
[2] Ghent University, Belgium
willem.waegeman@ugent.be
[3] University of Paderborn, Germany
[busarobi,eyke]@upb.de

Abstract. Label tree classifiers are commonly used for efficient multi-class and multi-label classification. They represent a predictive model in the form of a tree-like hierarchy of (internal) classifiers, each of which is trained on a simpler (often binary) subproblem, and predictions are made by (greedily) following these classifiers' decisions from the root to a leaf of the tree. Unfortunately, this approach does normally not assure consistency for different losses on the original prediction task, even if the internal classifiers are consistent for their subtask. In this paper, we thoroughly analyze a class of methods referred to as probabilistic classifier trees (PCTs). Thanks to training probabilistic classifiers at internal nodes of the hierarchy, these methods allow for searching the tree-structure in a more sophisticated manner, thereby producing predictions of a less greedy nature. Our main result is a regret bound for 0/1 loss, which can easily be extended to ranking-based losses. In this regard, PCTs nicely complement a related approach called filter trees (FTs), and can indeed be seen as a natural alternative thereof. We compare the two approaches both theoretically and empirically.

1 Introduction

Multi-class and multi-label classification problems are nowadays characterized not only by large sample sizes and feature spaces, but also by a large number of labels. In application fields like image classification [13], text classification [9], online advertising [4], and video recommendation [26], it is not uncommon to deal with tens or hundreds of thousands [12], or even millions of labels [1,23].

Label tree classifiers belong to the most efficient approaches for problems at this scale [3,13]. In this approach, a solution to the original problem is represented in the form of a hierarchy of classifiers, each of which is trained on a simpler subproblem. A prediction for a new example is then derived from the predictions of these (internal) classifiers, each of which corresponds to a node in the tree-like hierarchical structure; typically, each label in the original classification problem is uniquely represented by a path from the root to a leaf of that tree.

However, combining conventional training of the internal classifiers with greedy inference, namely, following a single root-to-leaf path in the tree, does not guarantee consistency of this approach [5,11]. Thus, even perfect (zero regret) classifiers in each node of the tree do not imply a perfect (global) classification of new examples. There are two ways to remedy this problem: adjusting training and adjusting inference. The first idea is to modify the training of the internal classifiers so as to assure the consistency of greedy inference later on. The second approach, while training more conventionally, guarantees consistency by searching the tree-structure for an optimal prediction in a less greedy way.

The first idea is realized by the *filter tree* (FT) approach [5]. By constructing label trees in a bottom-up manner, an internal classifier can anticipate the decisions of its successor classifiers, and exploit this information to properly condition its own behavior to these classifiers. In the case of 0/1 loss, this is accomplished thanks to a specific filter technique, which removes examples from the training data on which successor classifiers made incorrect predictions. For this training procedure, a regret bound connecting the global performance with the average performance of node classifiers can be proved [5]. This bound can be generalized from 0/1 loss to any cost-based loss function, albeit at the price of a more expensive training procedure; ranking-based losses, which require the ordering of labels, cannot be tackled by FTs. Since inference can be done in a greedy way, the complexity of prediction is only logarithmic in the number of labels. More recently, the training of FTs has been further improved in the context of multi-label classification [20].

The second approach ensures consistency thanks to more sophisticated search of label trees in the inference phase [11,19,21]. To this end, probabilistic classifiers in each node of the tree are required, which allow for assessing the usefulness of different search directions. Label trees with probabilistic classifiers have already been considered in multi-class classification under the name of conditional probability trees [4] and nested dichotomies [15,16]. In multi-label classification, a similar approach has been referred to as probabilistic classifier chains [10]. The same concept also appears in neural networks and natural language processing under the name of hierarchical softmax [22]. In the following we unify all these approaches and jointly refer to them as *probabilistic classifier trees* (PCTs).

We restrict to binary label trees, which are especially natural for multi-label classification; here, each level of the binary tree directly corresponds to one label. Higher order trees (including nodes with more than two children) are often used in multi-class classification. This usually improves the predictive performance at the cost of an increase in prediction time. We also assume the tree structure to be given beforehand, or to have been induced using any of the methods developed for this purpose [4,3,13,26], and focus on the (orthogonal) problem of how training and prediction should be performed to ensure consistency (given the tree structure).

The main contribution of the paper is a regret bound for PCT in the case of 0/1 loss, which is expressed in terms of the search error and the Kullback-Leibler (KL) divergence (i.e., log-loss regret) of the internal classifiers. The regret bound implies the consistency of the method, a good "sanity check" for any learning algorithm. Its form quantifies a trade-off between the computational complexity and the statistical accuracy. Moreover, we show that under log-loss we do not theoretically pay any price in terms of performance for representing the joint distribution over classes by a tree structure. Our regret analysis significantly extends and improves the results of [4] for the estimation error of conditional probability trees expressed in terms of squared error loss. We also point out that the bound can be further generalized to ranking-based losses, e.g., recall at k. We also generalize the tree search algorithms of [11] and [21] to get an anytime A^*-like algorithm and study its theoretical guarantees, extending the previous results given in [11]. Our theoretical contributions are complemented by a comparison of PCTs with filter trees, both conceptually and experimentally.

The paper is organized as follows. We formally state the problem in Section 2. Section 3 describes PCTs and gives a theoretical analysis of the generalized tree search algorithm. In Section 4, we prove the regret bound for 0/1 loss. Section 5 compares PCTs with other label tree approaches, particularly with conditional probability and filter trees. Section 6 discusses the use of PCTs for predicting top-k labels and its extension to multi-label classification. Section 7 presents experimental results, prior to concluding the paper in Section 8.

2 Problem statement

We formalize our problem in the setting of multi-class classification. Let $(\boldsymbol{x}, y)$ be an example coming from a probability distribution $P(\boldsymbol{X} = \boldsymbol{x}, Y = y)$ (later denoted $P(\boldsymbol{x}, y)$) on $\mathcal{X} \times \mathcal{Y}$, where $\boldsymbol{x} \in \mathcal{X} = \mathbb{R}^d$ and $y \in \mathcal{Y} = \{1, \ldots, m\}$. A classifier h predicts a label $\hat{y} = h(\boldsymbol{x}) \in \mathcal{Y}$ for each $\boldsymbol{x} \in \mathcal{X}$. The prediction accuracy of h can be measured in terms of 0/1 loss:[4]

$$\ell_{0/1}(y, h(\boldsymbol{x})) = [\![y \neq h(\boldsymbol{x})]\!]$$

We are interested in minimizing the expected loss, also referred to as the *risk*:

$$L_{0/1}(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim P}\left[\ell_{0/1}(y, h(\boldsymbol{x}))\right] = \int_{\mathcal{X}\times\mathcal{Y}} [\![y \neq h(\boldsymbol{x})]\!] \, dP(\boldsymbol{x}, y)$$

The *Bayes classifier*

$$h^* = \arg\min_h L_{0/1}(h)$$

minimizes the risk among all possible classifiers. While h^* may not be unique in general, the risk of h^*, denoted $L^*_{0/1}$, is unique, and is called the *Bayes risk*. Decomposing the risk over classes, i.e., writing $L_{0/1}(h)$ in the form

$$L_{0/1}(h) = \int_{\mathcal{X}} \underbrace{\left(\sum_{y\in\mathcal{Y}} [\![y \neq h(\boldsymbol{x})]\!] P(y|\boldsymbol{x}) \right)}_{=1-P(h(\boldsymbol{x})|\boldsymbol{x})} dP(\boldsymbol{x}) \,,$$

[4] We use $[\![P]\!]$ to denote a number that is 1 if condition P is satisfied, and 0 otherwise.

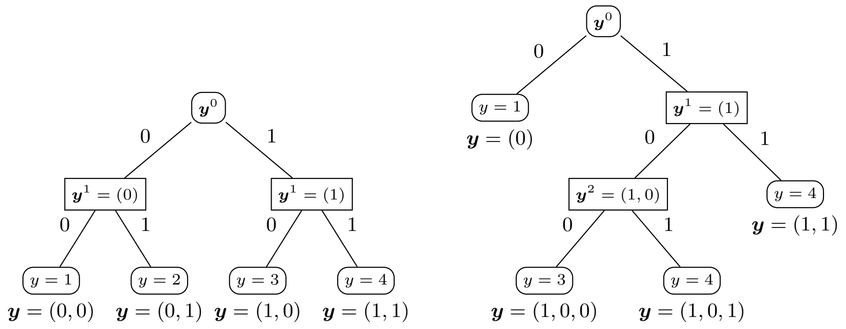

Fig. 1. Different binary codes in multi-class classification.

reveals that h^* minimizes risk in a pointwise manner, i.e., for every $\boldsymbol{x}$,

$$h^*(\boldsymbol{x}) = \arg\min_{y \in \mathcal{Y}} \left\{1 - P(y|\boldsymbol{x})\right\} = \arg\max_{y \in \mathcal{Y}} P(y \,|\, \boldsymbol{x}) \,.$$

Given a classifier h, the *regret* of h is defined as

$$\mathrm{reg}_{0/1}(h) \;=\; L_{0/1}(h) - L^*_{0/1} \;=\; \int_{\mathcal{X}} \Big(P(h^*(\boldsymbol{x})|\boldsymbol{x}) - P(h(\boldsymbol{x})|\boldsymbol{x}) \Big) dP(\boldsymbol{x}) \,. \quad (1)$$

The regret quantifies the suboptimality of h compared to the optimal classifier h^*. The goal is to train a classifier h with a small regret, ideally equal to zero.

In the following, we assume h to be represented as a label tree classifier. To this end, we encode the labels $\{1, \ldots, m\}$ using a prefix code. Any such code can be represented by a tree with 0/1 splits. Each path from the root to a leaf node then corresponds to a code word. Recall that codes of fixed length are also prefix codes. Figure 1 shows two examples of coding trees for multi-class classification with 4 classes. Under the coding, we represent each label y by a binary vector $\boldsymbol{y} = (y_1, \ldots, y_l)$, where l is the maximum length of the code. The set of all code words we denote by $\mathcal{C}$. As another special case, consider the problem of multi-label (instead of multi-class) classification, where the goal is to predict the set of labels assigned to a given instance $\boldsymbol{x}$. Such a set can be represented by a binary vector $\boldsymbol{y} = (y_1, \ldots, y_m)$, which in turn can be used as a prefix code.

In the label tree approach, we put a binary classifier in each non-leaf node of the tree. An internal node can be uniquely identified by the partial code word $\boldsymbol{y}^i = (y_1, \ldots, y_i)$. We denote the root node by $\boldsymbol{y}^0$, which is an empty vector (without any elements). The final prediction is determined by a sequence of decisions of internal classifiers. In the next section, we present a specific instance of the label tree approach that uses probabilistic classifiers in internal nodes of the tree.

3 Probabilistic classifier trees

Probabilistic classifier trees (PCTs) are designed to estimate probabilities $P(y \,|\, \boldsymbol{x})$ by following a path from the root to a leaf node, which corresponds to a code word $\boldsymbol{y} = (y_1, \ldots, y_l)$ assigned to label $y \in \mathcal{Y}$. Recalling the chain rule of probability, the process corresponds to computing

$$P(y \,|\, \boldsymbol{x}) = P(\boldsymbol{y} \,|\, \boldsymbol{x}) = \prod_{i=1}^{l} P(y_i | \boldsymbol{y}^{i-1}, \boldsymbol{x}) \,, \tag{2}$$

where $P(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x})$ are probabilities of $y_i \in \{0, 1\}$, estimated in non-leaf nodes $\boldsymbol{y}^{i-1}$. In the next two subsections, training and inference (classification of new examples) for PCT will be discussed in more detail.

3.1 Training

Training of PCT naturally decomposes into learning problems over non-leaf nodes of the tree. In each node $\boldsymbol{y}^{i-1}$, the task is to train a probabilistic classifier (e.g., logistic regression) to estimate $P(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x})$.

Looking at PCTs as a reduction technique, it is worth mentioning that its training complexity could be much lower than that of the 1-vs-all approach, since each example $(\boldsymbol{x}, y)$ is used in only l instead of m binary problems, where l is the height of the tree (i.e., $l = \lceil \log_2 m \rceil$ if the tree is balanced). To further improve the training time complexity, one can use online learning methods, such as stochastic gradient descent [6]. Moreover, internal classifiers in PCT can be trained independently of each other, thereby allowing for a massive parallelization of the training procedure. Let us also remark that the learning process can be defined as a single task; this is the so-called one-classifier trick [5], in which a node indicator is used as an additional feature. Alternatively, one can use a separate task for each level of the tree. This approach is used in multi-label classification, as will be discussed in Section 6.

3.2 Inference

The classification procedure in PCTs is more involved. To begin with, note that a probability estimate $Q(y \,|\, \boldsymbol{x})$ for any label y (given instance $\boldsymbol{x}$) is obtained quite easily, simply by following the corresponding path in the tree and applying the chain rule:

$$Q(y \,|\, \boldsymbol{x}) = Q(\boldsymbol{y} \,|\, \boldsymbol{x}) = \prod_{i=1}^{l} Q(y_i | \boldsymbol{y}^{i-1}, \boldsymbol{x})$$

However, being interested in the minimization of 0/1 loss, we actually seek to find

$$\hat{\boldsymbol{y}}^* = \operatorname*{arg\,max}_{\boldsymbol{y} \in \mathcal{C}} Q(\boldsymbol{y} \,|\, \boldsymbol{x}) \,, \tag{3}$$

preferably without computing the probability of each label first. A simple idea is to follow a single path in the tree, starting in the root and always choosing the branch $y_i \in \{0, 1\}$ for which $Q(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x}) > 0.5$. However, while being efficient, this approach is not guaranteed to find the optimal solution [5,11].

Algorithm 1 Inference with ϵ-approximate A^*

```
1: input: $\boldsymbol{x}$ (test example)
2: priority list $\mathcal{Q} \leftarrow \{\boldsymbol{y}_0\}$ (contains root node initially)
3: priority list $\mathcal{K} \leftarrow \{\}$ (contains nodes whose both children were not inserted to $\mathcal{Q}$)
4: $\epsilon \leftarrow 2^{-c}$ with $1 \leq c \leq m$
5: while $\mathcal{Q} \neq \emptyset$ do
6:     $\boldsymbol{v} \leftarrow$ pop first element in $\mathcal{Q}$
7:     if $\boldsymbol{v}$ is a leaf then delete all elements in $\mathcal{K}$ and break the while loop
8:     $\boldsymbol{v}_1 \leftarrow (\boldsymbol{v}, 1)$ (left child of $\boldsymbol{v}$) and $\boldsymbol{v}_0 \leftarrow (\boldsymbol{v}, 0)$ (right child of $\boldsymbol{v}$)
9:     compute $E(\boldsymbol{v}_1 \,|\, \boldsymbol{x})$ and $E(\boldsymbol{v}_0 \,|\, \boldsymbol{x})$ recursively from $E(\boldsymbol{v} \,|\, \boldsymbol{x})$ using Eqn. (4)
10:    if $E(\boldsymbol{v}_1 \,|\, \boldsymbol{x}) \geq \epsilon$ then add $(\boldsymbol{v}_1, E(\boldsymbol{v}_1 \,|\, \boldsymbol{x}))$ to $\mathcal{Q}$ sorted in descending order of $E$
11:    if $E(\boldsymbol{v}_0 \,|\, \boldsymbol{x}) \geq \epsilon$ then add $(\boldsymbol{v}_0, E(\boldsymbol{v}_0 \,|\, \boldsymbol{x}))$ to $\mathcal{Q}$ sorted in descending order of $E$
12:    if $\boldsymbol{v}_1$ and $\boldsymbol{v}_0$ are not inserted to $\mathcal{Q}$ then add $\boldsymbol{v}$ to $\mathcal{K}$ in descending order of $E$
13: $\theta \leftarrow 0$
14: while $\mathcal{K} \neq \emptyset$ do
15:     $\boldsymbol{v}' \leftarrow$ pop first element in $\mathcal{K}$
16:     $\boldsymbol{v}' \leftarrow$ apply greedy search downward on $\boldsymbol{v}'$
17:     if $Q(\boldsymbol{v}' \,|\, \boldsymbol{x}) \geq \theta$ then $\boldsymbol{v} \leftarrow \boldsymbol{v}'$ and $\theta \leftarrow Q(\boldsymbol{v}' \,|\, \boldsymbol{x})$
18: return $h_\epsilon(\boldsymbol{x}) = \hat{\boldsymbol{y}}_\epsilon = \boldsymbol{v}$
```

Better inference methods have been presented in recent years, based on search algorithms such as uniform-cost search [11], beam search [19], and A^* [21].

All three approaches allow for trading complexity against optimality, and hence for using PCTs in an anytime fashion, thanks to a hyper-parameter ϵ. This parameter controls the degree of optimality, i.e., of finding the true loss minimizer (3), as a function of the runtime (it finds a solution $\hat{\boldsymbol{y}}_\epsilon$ the conditional probability $Q(\hat{\boldsymbol{y}}_\epsilon \,|\, \boldsymbol{x})$ of which is not much worse than the probability of the optimal solution $\hat{\boldsymbol{y}}^*$ defined in Eqn. 3). In the analysis that follows, we will use this property to give a formal bound on the error made by such inference algorithms, with a particular focus on uniform-cost and A^* search. An extension of the analysis to beam search is straightforward and omitted due to lack of space. The pseudo code in Algorithm 1 unifies the approaches of [21] and [11]. This general algorithm, which we denote $h_\epsilon(\boldsymbol{x})$, is a variant of A^*. It fulfills the anytime property, i.e., the search can be stopped at any time and the algorithm will deliver a valid though possibly suboptimal solution.

Recall that each node in the tree is uniquely defined by a path from the root to this node, i.e., by the partial code word $\boldsymbol{y}^i$. We use $\boldsymbol{v}$ to denote the node currently visited by the algorithm, and associate with this node the following value:

$$E(\boldsymbol{v} \,|\, \boldsymbol{x}) = E(\boldsymbol{y}^i \,|\, \boldsymbol{x}) = Q(\boldsymbol{y}^i \,|\, \boldsymbol{x}) \times H(\boldsymbol{y}^i \,|\, \boldsymbol{x})$$

This value can be interpreted as an approximation of the maximal value of $Q(\boldsymbol{y} \,|\, \boldsymbol{x})$, in which $Q(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ is the part of the path that can be computed when moving from the root to node $\boldsymbol{v}$, and $H(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ is a heuristic that optimistically guesses the part of the path that has not yet been computed (in the considered case, $E(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ has to overestimate or to be the same as the maximal value of

$Q(\boldsymbol{y} \,|\, \boldsymbol{x})$). $Q(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ can be computed recursively as follows: $Q(\boldsymbol{y}^0 \,|\, \boldsymbol{x}) = 1$ and

$$Q(\boldsymbol{y}^i \,|\, \boldsymbol{x}) = Q(y_i = 1 | \boldsymbol{y}^{i-1}, \boldsymbol{x}) \times Q(\boldsymbol{y}^{i-1} \,|\, \boldsymbol{x}) \,. \tag{4}$$

In [21], a procedure for computing $H(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ is proposed for the specific case of logistic regression as a base learner, whereas the heuristic is simply $H(\boldsymbol{y}^i \,|\, \boldsymbol{x}) = 1$ in uniform-cost search used in [11]. The former approach has the advantage of providing a more accurate estimation of maximal $Q(\boldsymbol{y} \,|\, \boldsymbol{x})$, albeit with an additional computing cost, while the latter approach makes a more rough estimation without any additional cost. Interestingly, as shown in experiments in [21], the former approach is still more expensive in terms of the total search cost than the latter.

In a nutshell, Algorithm 1 starts from the root of the label tree, which is the single element of priority list $\mathcal{Q}$, sorted in descending order of E. In every iteration, the top element of the list is popped and the children $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ of the corresponding node $\boldsymbol{v}$ are visited. $E(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ is then recursively computed for the children of node $\boldsymbol{v}$, which are added to the list if this quantity exceeds the threshold $\epsilon = 2^{-c}$ with $1 \leq c \leq l$, where l is the maximal length of the path in the tree. Basically, they are inserted into the list at the appropriate position, so that the order imposed by $E(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ is respected. The first while-loop of the algorithm stops in two situations: (i) when the element popped from the list $\mathcal{Q}$ corresponds to a leaf of the tree, or (ii) when the list $\mathcal{Q}$ is empty. The label corresponding to the leaf is then returned in the former case, while in the latter case, inference by greedy search is applied to define a path from all nodes from the list $\mathcal{K}$. This list, also sorted in descending order of E, contains nodes for which none of their children has been added to $\mathcal{Q}$. The use of list $\mathcal{K}$ ensures that by decreasing the value of ϵ, the algorithm will always find a solution that is not worse than a solution that would be found with greater ϵ.

Algorithm 1 enjoys strong theoretical guarantees. Assuming the cost for computing $H(\boldsymbol{y}^i \,|\, \boldsymbol{x})$ to be constant, the following result immediately follows from a theorem proved in [11].

Theorem 1. *Let $1 \leq c \leq l$. Algorithm 1 with $\epsilon = 2^{-c}$ needs at most $\mathcal{O}(l\epsilon^{-1})$ iterations to find a prediction $h_\epsilon(\boldsymbol{x}) = \hat{\boldsymbol{y}}_\epsilon$ such that*

$$Q(\hat{\boldsymbol{y}}^* \,|\, \boldsymbol{x}) - Q(\hat{\boldsymbol{y}}_\epsilon \,|\, \boldsymbol{x}) \leq \epsilon - 2^{-l} \,.$$

From the theorem, we see that the quality of the solution found by the algorithm improves with the length of the running time. Consequently, the algorithm will always find the optimal solution, provided its probability mass is greater than ϵ. Reformulating the above, we can say that the algorithm finds the solution in time linear in $1/q_{\max}$, where $q_{\max}$ is the probability mass of the best solution in the estimated distribution Q. For problems with low noise (high values of $q_{\max}$), this method should work very fast.

The theorem also implies that the greedy search, which corresponds to the algorithm with $\epsilon = 0.5$, has very poor guarantees that approach the bound of 0.5 with $m \to \infty$.

4 Regret bounds for PCT

In this section, we are concerned with the generalization ability of the PCT classifier, measured by means of the regret (1). Assume for a moment that $Q(\cdot|\boldsymbol{x})$, the label distribution produced by PCT, coincides with the true conditional distribution $P(\cdot|\boldsymbol{x})$ for every $\boldsymbol{x}$. Then, if the ϵ-approximate inference algorithm is used for classification, Theorem 1 implies the regret of the PCT classifier is at most ϵ, i.e., the expected classification error of PCT is at most ϵ larger than the expected classification error of the Bayes classifier.

It is, however, unrealistic to assume that PCT is able to perfectly match the true data distribution, hence $Q(\cdot|\boldsymbol{x})$ and $P(\cdot|\boldsymbol{x})$ will differ in general. Thus, the question arises whether the expected classification error of PCT is still not much worse than the expected classification error of the Bayes classifier if $Q(\cdot|\boldsymbol{x})$ and $P(\cdot|\boldsymbol{x})$ do not coincide, but are *close* to each other in some sense. This section presents an affirmative answer to this question, delivering a regret bound on the classification error that takes into account the predictive performance of the internal classifiers. More precisely, we bound the PCT regret for 0/1 loss in terms of the difference between Q and P, quantified in terms of *log-loss regret*.

We start with a general definition of the log-loss. Consider a problem of estimating a probability distribution on some outcome space $\mathcal{S}$. The log-loss of probability estimate $Q(\cdot)$ on $\mathcal{S}$ when the observed outcome is $y \in \mathcal{S}$ is given by

$$\ell_{\log}(y, Q) = -\log Q(y)\,.$$

The log-loss is by far the most popular measure for quantifying the accuracy of probabilistic predictions, and plays an important role in information theory, data compression, and statistics [8] (we briefly analyze the other loss function, squared loss, in Section 5). The *log-loss risk* is the expected log-loss of $Q(\cdot)$:

$$L_{\log}(Q) = \mathbb{E}_{y \sim P}[\ell_{\log}(y, Q)]\,,$$

where $P(\cdot)$ is the true distribution of y. The log-loss is a *strictly proper loss*, which means that the unique minimizer of the risk is achieved at $Q(\cdot) \equiv P(\cdot)$ (see, e.g., [24]). We thus define the *log-loss regret* as:

$$\mathrm{reg}_{\log}(Q) = L_{\log}(Q) - L_{\log}(P) = \mathbb{E}_{y \sim P}\left[\log \frac{P(y)}{Q(y)}\right] = D(P \| Q),$$

where $D(\cdot\|\cdot)$ is the Kullback-Leibler (KL) divergence.

We now turn back to PCTs. Let us first fix an instance $\boldsymbol{x} \in \mathcal{X}$ and consider the distribution over code words $\boldsymbol{y} \in \mathcal{C}$. There are two ways in which log-loss can be used in this setting:

– To measure the quality of the estimate of the joint distribution of labels given $\boldsymbol{x}$, $Q(\boldsymbol{y}|\boldsymbol{x})$, i.e., the outcome space is $\mathcal{S} = \mathcal{C}$, and the log-loss is $\ell_{\log}(\boldsymbol{y}, Q(\cdot|\boldsymbol{x})) = -\log Q(\boldsymbol{y}|\boldsymbol{x})$. The log-loss regret is then the KL divergence between true joint conditional distribution $P(\boldsymbol{y}|\boldsymbol{x})$ and its estimate $Q(\boldsymbol{y}|\boldsymbol{x})$, $\mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{x})) = D(P(\cdot|\boldsymbol{x})\|Q(\cdot|\boldsymbol{x}))$.

– To measure the quality of individual classifiers in each node of the tree. Given a node $\boldsymbol{y}^{i-1} = (y_1, \ldots, y_{i-1})$, the probability estimate for label $y_i \in \{0,1\}$ at this node is $Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x})$. Thus, the outcome space is $\mathcal{S} = \{0,1\}$, and $\ell_{\log}(y_i, Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x})) = -\log Q(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x})$. The log-loss regret is then $\mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x})) = D(P(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x}) \| Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x}))$.

Both ways described above turn out to be equivalent. Indeed, we have

$$\ell_{\log}(\boldsymbol{y}, Q(\cdot|\boldsymbol{x})) = -\log Q(\boldsymbol{y}|\boldsymbol{x}) = \sum_{i=1}^{l} -\log Q(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x}) = \sum_{i=1}^{l} \ell_{\log}(y_i, Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x})) \,,$$

so that the log-loss of the joint distribution is equal to the sum of log-losses of individual node classifiers along the path from the root to leaf $\boldsymbol{y}$. Similarly,

$$\begin{aligned}\mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{x})) &= \mathbb{E}_{\boldsymbol{y}\sim P(\cdot|\boldsymbol{x})}\left[\log \frac{P(\boldsymbol{y}|\boldsymbol{x})}{Q(\boldsymbol{y}|\boldsymbol{x})}\right] = \mathbb{E}_{\boldsymbol{y}\sim P(\cdot|\boldsymbol{x})}\left[\sum_{i=1}^{l} \log \frac{P(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x})}{Q(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x})}\right] \\ &= \mathbb{E}_{\boldsymbol{y}\sim P(\cdot|\boldsymbol{x})}\left[\sum_{i=1}^{l} \mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x}))\right], \qquad (5)\end{aligned}$$

i.e., the log-loss regret of the joint distribution is equal to the sum of the regrets of node classifiers along the random path from the root to leaf $\boldsymbol{y}$, where $\boldsymbol{y}$ is drawn from $P(\cdot|\boldsymbol{x})$. This basically expresses the chain rule for KL divergence [8]. The consequence of the above is that under log-loss we theoretically do not pay any price in terms of performance for representing the joint distribution by a tree structure.

We are now ready to present the main result of this section, which states that the 0/1-regret of the PCT classifier is bounded by means of the sum of log-loss regrets along a random path from the root to the leaf (or, equivalently, by the log-loss regret of the joint distribution) and the search error ϵ of the inference procedure.

Theorem 2. *Consider PCT, which estimates the probability $Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x})$ in each non-leaf node $\boldsymbol{y}^{i-1}$, and let h_ϵ be the classifier which for any $\boldsymbol{x}$, outputs $\hat{\boldsymbol{y}}_\epsilon$ found by the ϵ-approximate inference procedure (Algorithm 1). Then, for any distribution P,*

$$\mathrm{reg}_{0/1}(h_\epsilon) \le \sqrt{2\mathrm{reg}_{\log}(Q)} + \epsilon - 2^{-l},$$

where $\mathrm{reg}_{\log}(Q) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim P}\left[\sum_{i=1}^{l} \mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{y}^{i}, \boldsymbol{x}))\right]$ is the expected sum of regrets at internal classifiers along a path from the root to the leaf.

Proof. We first condition everything on a fixed $\boldsymbol{x}$. Let $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x})$ be the mode of $P(\cdot|\boldsymbol{x})$, and let $\hat{\boldsymbol{y}}_\epsilon = h_\epsilon(\boldsymbol{x})$ be the output of Algorithm 1 for input $\boldsymbol{x}$. Moreover, we let $\hat{\boldsymbol{y}}^* = \arg\max_{\boldsymbol{y}} Q(\boldsymbol{y}|\boldsymbol{x})$ denote the mode of $Q(\cdot|\boldsymbol{x})$, and note that from Theorem 1,

$$Q(\hat{\boldsymbol{y}}^*|\boldsymbol{x}) - Q(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x}) \le \epsilon - 2^{-l}. \qquad (6)$$

According to (1), the 0/1-regret of $\hat{\boldsymbol{y}}_\epsilon$ conditioned at $\boldsymbol{x}$ is given by

$$\mathrm{reg}_{0/1}(\hat{\boldsymbol{y}}_\epsilon) = P(\boldsymbol{y}^*|\boldsymbol{x}) - P(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x}).$$

Note that the regret is 0 if $\boldsymbol{y}^* = \hat{\boldsymbol{y}}_\epsilon$, hence we assume $\boldsymbol{y}^* \neq \hat{\boldsymbol{y}}_\epsilon$ in what follows. From the definition of $\hat{\boldsymbol{y}}^*$, $Q(\hat{\boldsymbol{y}}^*|\boldsymbol{x}) - Q(\boldsymbol{y}^*|\boldsymbol{x}) \geq 0$, which together with (6) gives $Q(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x}) - Q(\boldsymbol{y}^*|\boldsymbol{x}) + \epsilon - 2^{-l} \geq 0$. Hence, we obtain the upper bound

$$\begin{aligned}
\mathrm{reg}_{0/1}(\hat{\boldsymbol{y}}_\epsilon) &\leq \Big(P(\boldsymbol{y}^*|\boldsymbol{x}) - Q(\boldsymbol{y}^*|\boldsymbol{x})\Big) + \Big(Q(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x}) - P(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x})\Big) + \epsilon - 2^{-l} \\
&\leq \big|P(\boldsymbol{y}^*|\boldsymbol{x}) - Q(\boldsymbol{y}^*|\boldsymbol{x})\big| + \big|Q(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x}) - P(\hat{\boldsymbol{y}}_\epsilon|\boldsymbol{x})\big| + \epsilon - 2^{-l} \\
&\leq \sum_{\boldsymbol{y}\in\mathcal{C}} \big|P(\boldsymbol{y}|\boldsymbol{x}) - Q(\boldsymbol{y}|\boldsymbol{x})\big| + \epsilon - 2^{-l},
\end{aligned}$$

where the last inequality is from $\boldsymbol{y}^* \neq \hat{\boldsymbol{y}}_\epsilon$. We now make use of Pinsker's inequality

$$\frac{1}{2}\sum_{\boldsymbol{y}\in\mathcal{C}} \big|P(\boldsymbol{y}\,|\,\boldsymbol{x}) - Q(\boldsymbol{y}\,|\,\boldsymbol{x})\big| \leq \sqrt{\frac{1}{2} D(P(\cdot\,|\,\boldsymbol{x}) \| Q(\cdot\,|\,\boldsymbol{x}))}\,,$$

which together with (5) implies

$$\mathrm{reg}_{0/1}(\hat{\boldsymbol{y}}_\epsilon) \leq \sqrt{2\mathbb{E}_{\boldsymbol{y}\sim P(\cdot|\boldsymbol{x})}\left[\sum_{i=1}^{l} \mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x}))\right]} + \epsilon - 2^{-l}. \tag{7}$$

Note that the 0/1-regret of h_ϵ, $\mathrm{reg}_{0/1}(h_\epsilon)$, is just the expectation of the left-hand side of (7) with respect to $\boldsymbol{x}$. Thus, taking expectation on both sides of (7), and using $\mathbb{E}\left[\sqrt{\cdot}\right] \leq \sqrt{\mathbb{E}\left[\cdot\right]}$ on the right-hand side (which is Jensen's inequality applied to the concave function $x \mapsto \sqrt{x}$) gives

$$\begin{aligned}
\mathrm{reg}_{0/1}(h_\epsilon) &\leq \sqrt{2\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim P}\left[\sum_{i=1}^{l} \mathrm{reg}_{\log}(Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x}))\right]} + \epsilon - 2^{-l} \\
&= \sqrt{2\mathrm{reg}_{\log}(Q)} + \epsilon - 2^{-l}.
\end{aligned}$$

□

Theorem 2 states that if the log-loss regret of node classifiers is small, the resulting ϵ-approximate classifier will be close to the Bayes classifier in terms of 0/1 loss. This suggests to use node classifiers which minimize log-loss on the training sample, examples of which include logistic regression, Gradient Boosting Machines, deep neural networks,[5] and many others [17]. One can show that the square-root dependence in the bound of Theorem 2 cannot be improved in general, since when the tree consists only of the root node, our bound essentially specializes to the bound in [2], which also exhibits square-root dependence.

[5] In this case, the log-loss if often referred to as "soft-max" function.

5 Relation to other label tree approaches

5.1 Conditional probability trees

Conditional probability trees (CPTs) [5] estimate a conditional probability distribution $P(y|\boldsymbol{x})$ in the multiclass setting and have the same structure as PCTs. What distinguishes this approach from ours is that CPTs are used for probability estimation, with squared loss $\ell_{\mathrm{sq}}(y_i, Q(\cdot|\boldsymbol{y}^{i-1}, \boldsymbol{x})) = \big(y_i - Q(y_i|\boldsymbol{y}^{i-1}, \boldsymbol{x})\big)^2$ as a performance measure, whence there is no inference phase to determine the mode of the conditional distribution. The main result in [5] relates the squared loss regret on the joint distribution to the expected squared loss over the nodes of the tree. This result is analogous to the identity (5), except that an additional $O(\sqrt{l})$ factor appears in the squared loss bound. Moreover, no result analogous to Theorem 2 is given, which would relate expected squared loss regret to the 0/1 classification regret.

In fact, we can show a *lower bound* on the 0/1 regret in terms of expected squared loss, which is at least a factor of $\Omega(\sqrt{l})$ worse than our bound. To be more precise, one can show that for any $l > 2$, there exists a true distribution P and an estimate Q with the following property: even when assuming that the inference algorithm can identify the mode of the distribution exactly, it holds that $\mathrm{reg}_{0/1}(h_\epsilon) > \sqrt{l \, \mathrm{reg}_{\mathrm{sq}}(Q)}$, where $\mathrm{reg}_{\mathrm{sq}}(Q)$ is the corresponding regret with log-loss replaced by squared loss.[6] In other words, using squared loss yields a bound for classification error that is at least a factor $\Omega(\sqrt{l})$ worse than the bound we obtained for log-loss.

5.2 Filter trees

The filter tree (FT) approach [4] is the first label tree algorithm for which a regret bound for the classification error has been proved. Interestingly, the specific training procedure used in FTs ensures that the greedy classification procedure is sufficient for obtaining consistent predictions.

FT uses the same tree structure as PCT, but with binary classifiers instead of class probability estimators in the non-leaf nodes of the tree. The method follows a bottom-up strategy, which can be interpreted as a single elimination tournament on the set of labels. A classifier in node $\boldsymbol{y}^{i-1}$ is trained to predict y_i, but FT implicitly transforms the underlying distribution of examples in the node. The transformation for 0/1 loss relies on filtering out all training examples that have been misclassified by successor classifiers on a path to a leaf. The learning algorithm starts with classifiers on the lowest non-leaf level of the tree. The correctly classified examples are then moved upward to nodes one level above. This process is repeated until the root node is reached.

In [4], a regret bound for 0/1 loss has been proved that is conceptually similar to the one given in Theorem 2. The difference is that the right side of the

[6] We skip the details of the construction of P and Q due to the space limit.

bound is expressed in terms of 0/1 loss of the binary classifiers in non-leaf nodes. Therefore, these two bounds are not directly comparable.

Another advantage of FTs is that they can be used with any cost-based loss function. An appropriate bound has also been proved in [4]. The classification procedure still follows a greedy search, but training is more demanding. It requires weighting of examples, the use of cost-sensitive learners, and each training example generally occurs in each internal classifier.

6 Extensions of PCTs

Since PCT estimates the entire conditional distribution over labels, it can be used with any loss function. This comes with no additional cost during training, but may lead to very costly inference. Actually, inference can be performed efficiently only for certain losses, such as 0/1 loss as discussed in Section 3.2, but also some ranking-based loss functions. As an example, consider recall at kth position defined as

$$R_{@k}(\boldsymbol{y}, \boldsymbol{x}, \mathcal{Y}_k) = [\![\boldsymbol{y} \in \mathcal{Y}_k]\!] \,,$$

where $\mathcal{Y}_k$ is a set of k labels predicted for $\boldsymbol{x}$. One can easily verify that an optimal $\mathcal{Y}_k$ should contain k top-labels with largest $P(y \,|\, \boldsymbol{x})$. This can be approximated by k top-labels with largest $Q(y \,|\, \boldsymbol{x})$, which are easily obtained by PCT and a small extension of the ϵ-approximate algorithm: it is enough to continue the search procedure until k leaves are visited. Moreover, the bound in Theorem 2 can be easily extended to this case.

As already mentioned, PCTs can also be used in multi-label classification. In this case, the tree is of height m and is fully balanced. Each path from the root to a leaf corresponds to one of possible label combinations. In principle, PCT contains a single classifier in each non-leaf node. In the case of multi-label classification, storing $2^m - 1$ classifiers for large m is unfortunately not feasible. One can, however, follow a trick used in probabilistic classifier chains [10] and condensed filter trees [20], which relies on using one binary classifier per tree level. In other words, prediction of the ith label corresponds to the prediction made by the classifier on level i with additional features that indicate a given node of the tree.

7 Experimental Results

We empirically evaluate PCTs and FTs in two scenarios: multi-label classification (MLC) and multi-class classification (MCC). We test the algorithms in terms of 0/1 loss and the computational costs of their training and testing procedures. For PCTs, we additionally report $R_{@k}$.

We conduct experiments on 3 multi-class and 3 multi-label datasets.[7] Table 1 provides a summary of basic statistics of the datasets. Notice that the number of

[7] Taken from the libsvm repository `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets` and the image net competition webpage `http://www.image-net.org/challenges/LSVRC/2010`.

leaf nodes is equal to m (the number of labels) in case of multi-class problems, and 2^m (the number of all possible label combinations) in case of multi-label problems. We therefore use multi-label datasets up to around 100 labels. For datasets with a greater number of labels, the 0/1 loss is usually very close to 1. We use the original split into a training and test set if available; otherwise, we use 90/10 train/test splits. For the ILSVR2010 dataset, we use the visual code words (sbow) vectors provided by the organizers of the challenge. Features were generated on the basis of the guidance contained in the ILSVR development kit.

Table 1. Multi-class (MCC) and multi-label (MLC) datasets and their properties: the number of training (#train) and test (#test) examples, the number of labels (m) and features (d).

	MCC				MLC				
Dataset	#train	#test	m	d	Dataset	#train	#test	m	d
Sector	6412	3207	105	55197	Yeast	1500	917	14	103
Aloi	97200	10800	1000	128	TMC	21519	7077	22	30438
ILSVR2010	1261406	150000	1000	1000	Mediamill	30993	12914	101	120

7.1 Implementation

We carefully implemented PCTs and FTs in Java. As internal classifiers, we use L_2 linear logistic regression trained by a variant of stochastic gradient descent (SGD) introduced in [14]. To deal with a large number of weights, we use feature hashing [25] shared over all tree nodes using hashes up to size of 2^{24}. We use a random complete binary tree to code class labels in the MCC scenarios and train a classifier in each node of the tree. For MLC problems we take the original order of the labels to obtain the code words. We use one classifier per tree level. We tune the hyper-parameters of SGD in a 80/20 simple validation on the training set. We applied an off-the-shelf hyper-parameter optimizer [18] with a wide range of parameters. We tune PCTs to optimize the log-loss as suggested by our theoretical analysis. FTs are tuned to perform well on 0/1 loss.

We use PCTs with the ϵ-approximate inference algorithm with different values of $\epsilon \in \{0, 0.25, 0.5\}$. The variant with $\epsilon = 0.5$ corresponds to greedy search, while the algorithm with $\epsilon = 0$ will always find the optimal solution, but may visit all nodes of the tree in the worst case (in fact, ϵ should be set to 2^{-l} instead to 0 to be concordant with the description of the algorithm; to keep the notation simple, we use 0 to indicate the smallest possible value of ϵ for a given dataset).

7.2 Results

The results are given in Table 2. We can observe that the results of PCTs improve with decreasing value of ϵ. PCT with $\epsilon = 0.5$ gets worse results than FT, which confirms the theoretical results, i.e., the filtering of misclassified examples during training in FT improves the results for the greedy inference. For $\epsilon = 0.25$,

Table 2. Experimental results for 0/1 loss (in percentages), 1-$R_{@5}$ (in percentages), train (t_{trn}) and test (t_{test}) running times (in seconds), and the average (A) number of inner products computed during classification of a test example. The *Top 1* column indicates the results for top-1 prediction (standard prediction of the algorithms), while column *Top 5* the results for top-5 prediction (only for PCT with $\epsilon < 0.5$). The best results are indicated in bold (except for wall-clock times which can be affected by many factors). The value in subscript of PCT corresponds to the value of ϵ.

	MCC							MLC						
		Top 1			Top 5				Top 1			Top 5		
	t_{trn}	0/1	t_{test}	A	1-R$_{@5}$	t_{test}	A	t_{trn}	0/1	t_{test}	A	1-R$_{@5}$	t_{test}	A
	Sector, $m = 105$							Yeast, $m = 14$						
FT	11.75	13.43	0.144	6.81		–		2.49	78.73	0.07	**14**		–	
PCT$_{.5}$	11.56	17.18	0.154	6.81		–		3.12	80.15	0.04	**14**		–	
PCT$_{.25}$	11.56	13.68	0.16	7.04	12.61	**0.24**	**7.5**	3.12	79.28	0.05	17.15	76.22	**0.12**	**21.3**
PCT$_0$	11.56	**13.28**	0.198	7.13	**7.23**	0.48	18.2	3.12	**78.62**	0.09	23.82	**58.77**	0.17	64.6
	Aloi, $m = 105$							TMC, $m = 22$						
FT	15.11	88.98	0.14	**9.97**		–		30.7	77.06	0.47	**22**		–	
PCT$_{.5}$	13.43	88.99	0.14	**9.97**		–		34.3	75.06	0.39	**22**		–	
PCT$_{.25}$	13.43	**88.95**	0.15	9.98	88.64	**0.21**	**10.2**	34.3	73.74	0.45	27.97	68.50	**0.57**	**34.0**
PCT$_0$	13.43	**88.95**	0.21	9.98	**76.19**	0.55	26.1	34.3	**73.18**	0.73	33.50	**41.18**	1.29	87.9
	ILSVR2010, $m = 1000$							Mediamill, $m = 101$						
FT	1710	95.10	10.12	**8.39**		–		220	90.79	2.24	**101**		–	
PCT$_{.5}$	1825	99.96	10.13	**8.39**		–		274	90.78	2.22	**101**		–	
PCT$_{.25}$	1825	95.30	13.23	10.03	95.30	**20.10**	**14.4**	274	90.06	2.79	107	89.14	**3.02**	**129**
PCT$_0$	1825	**94.76**	15.20	10.57	**92.33**	44.31	34.3	274	**89.65**	5.23	274	**74.22**	9.50	529

the results are already very competitive to FT. For $\epsilon = 0$, PCT consistently outperforms FT, but the difference is not always large.

From a computational perspective, FTs achieve better performance. The training time of both approaches is very similar, but the testing time is in favor of FTs (and PCTs with $\epsilon = 0.5$). To give a deeper insight into the time costs we also report the average number of inner products computed by internal classifiers per test example. Interestingly, PCT with $\epsilon = 0$ always finds the solution in a reasonable time. Its testing time is never longer than three times that of FT. Similarly, the number of inner products is only up to three times greater than that of FT or PCT with $\epsilon = 0.5$.

Recall at kth position ($R_{@k}$) can be measured only for PCTs. There is no way to deliver top-k predictions in FTs, since this algorithm uses binary decisions in non-leaf nodes, so the search process results in only a single path from the root to a leaf node. Based on the results we can observe that PCT efficiently finds topmost results. The positive label appears more often in the top-5 predictions than in the top-1. Similarly as for 0/1 loss, $R_{@5}$ improves with decreasing value of ϵ. Unfortunately, predicting top-k labels increases test time. Therefore, the label tree search for $\epsilon = 0$ requires about 2-3 times more steps to find top-5 labels.

8 Conclusion

In this paper, we analyzed probabilistic classifier trees for efficient multi-class and multi-label classification. In particular, we proved a regret bound for 0/1 loss, which provides a strong theoretical foundation of PCTs, and which can also be extended to ranking-based losses. Moreover, we compared PCTs with the closely related filter tree method. We conclude the paper by summarizing the main theoretical and empirical results of FTs and PCTs, pointing out advantages and disadvantages of both approaches.

An unquestionable advantage of FTs is their prediction time, which is logarithmic in the number of classes or possible label combinations. FT can be used with any type of binary classifier as base learner and relies on simple 0/1 predictions. However, to guarantee the consistency of greedy inference, it requires more demanding training. In the naïve implementation, classifiers are trained sequentially in a bottom-up manner. The most important disadvantage is a significant reduction of the number of training examples in the top levels of the tree, which is caused by filtering examples in each level from bottom to top. This sparsity of training data may deteriorate predictive performance. However, thanks to filtering, an internal classifier is aware of errors of the successor classifiers. FT can be used with any cost-based loss function, but it is not able to predict top-k labels.

Prediction with PCTs requires search techniques, whence it is usually more demanding than FTs (yet significantly faster than 1-vs-all). Moreover, anytime algorithms can be used for searching the tree. The time complexity of PCT strongly depends on the noise contained in the data. If the signal-to-noise ratio is high, we can expect prediction time to be small. However, learning is much simpler for PCT than for FT, and can be easily parallelized. There is no filtering of training examples, so all examples are used for training on each level of the tree. The probabilistic nature of PCTs allows for delivering a list of top-labels and to work efficiently for $R_{@k}$.

The results we obtained for FTs are comparable with those reported in [7]. We stress that better results can be obtained by other algorithms, for example LomTrees introduced in the same paper. This is mainly because LomTrees train the tree structure online, along with the internal classifiers, whereas PCTs and FTs use random trees/coding. Interestingly, LomTrees are not consistent. Thus, an important challenge for future research is to find an algorithm that is able to train the tree structure online while ensuring consistency.

Acknowledgments. The work of Krzysztof Dembczyński and Wojciech Kotłowski has been supported by the Polish National Science Centre under grant no. 2013/09/D/ST6/03917 and 2013/11/D/ST6/03050, respectively.

References

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

2. Bartlett, P.L., Jordan, M.I., McAuliffe, J.D.: Convexity, classification, and risk bounds. Journal of the American Statistical Association 101(473), 138–156 (2006)
3. Bengio, S., Weston, J., Grangier, D.: Label embedding trees for large multi-class tasks. In: NIPS 23. pp. 163–171. Curran Associates, Inc. (2010)
4. Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G.B., Strehl, A.L.: Conditional probability tree estimation analysis and algorithms. In: UAI. pp. 51–58 (2009)
5. Beygelzimer, A., Langford, J., Ravikumar, P.D.: Error-correcting tournaments. In: ALT. pp. 247–262 (2009)
6. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: COMPSTAT. pp. 177–187. Springer (2010)
7. Choromanska, A., Langford, J.: Logarithmic time online multiclass prediction. In: NIPS 29 (2015)
8. Cover, T., Thomas, J.: Elements of Information Theory. Wiley (1991)
9. Dekel, O., Shamir, O.: Multiclass-multilabel learning when the label set grows with the number of examples. In: AISTATS (2010)
10. Dembczyński, K., Cheng, W., Hüllermeier, E.: Bayes optimal multilabel classification via probabilistic classifier chains. In: ICML. pp. 279–286. Omnipress (2010)
11. Dembczyński, K., Waegeman, W., Cheng, W., Hüllermeier, E.: An analysis of chaining in multi-label classification. In: ECAI (2012)
12. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Li, F.F.: ImageNet: A large-scale hierarchical image database. In: CVPR. pp. 248—255 (2009)
13. Deng, J., Satheesh, S., Berg, A.C., Li, F.F.: Fast and balanced: Efficient label tree learning for large scale object recognition. In: NIPS. pp. 567–575 (2011)
14. Duchi, J., Singer, Y.: Efficient online and batch learning using forward backward splitting. JMLR 10, 2899–2934 (2009)
15. Fox, J.: Applied regression analysis, linear models, and related methods. Sage (1997)
16. Frank, E., Kramer, S.: Ensembles of nested dichotomies for multi-class problems. In: ICML (2004)
17. Friedman, J.H., Hastie, T., Tibshirani, R.: Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2009)
18. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization. Springer (2011)
19. Kumar, A., Vembu, S., Menon, A.K., Elkan, C.: Beam search algorithms for multilabel learning. Machine Learning 92(1), 65–89 (2013)
20. Li, C.L., Lin, H.T.: Condensed filter tree for cost-sensitive multi-label classification. In: ICML. pp. 423–431 (2014)
21. Mena, D., Montañés, E., Quevedo, J.R., del Coz, J.J.: Using A* for inference in probabilistic classifier chains. In: IJCAI. pp. 3707–3713 (2015)
22. Morin, F., Bengio, Y.: Hierarchical probabilistic neural network language model. In: AISTATS. pp. 246–252 (2005)
23. Prabhu, Y., Varma, M.: Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In: KDD. pp. 263–272. ACM (2014)
24. Reid, M.D., Williamson, R.C.: Composite binary losses. JMLR 11, 2387–2422 (2010)
25. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: ICML. pp. 1113–1120. ACM (2009)
26. Weston, J., Makadia, A., Yee, H.: Label partitioning for sublinear ranking. In: ICML. pp. 181–189 (2013)