Online probabilistic label trees

Kalina Jasinska-Kobus[*]
ML Research at Allegro.pl, Poland
Poznan University of Technology, Poland
kjasinska@cs.put.poznan.pl

Marek Wydmuch[*]
Poznan University of Technology, Poland
mwydmuch@cs.put.poznan.pl

Devanathan Thiruvenkatachari
Yahoo Research, New York, USA

Krzysztof Dembczyński
Yahoo Research, New York, USA
Poznan University of Technology, Poland

[*] Equal contribution

Abstract

We introduce *online probabilistic label trees* (OPLTs), an algorithm that trains a label tree classifier in a fully online manner without any prior knowledge about the number of training instances, their features and labels. OPLTs are characterized by low time and space complexity as well as strong theoretical guarantees. They can be used for online multi-label and multi-class classification, including the very challenging scenarios of one- or few-shot learning. We demonstrate the attractiveness of OPLTs in a wide empirical study on several instances of the tasks mentioned above.

1 Introduction

In modern machine learning applications, the label space can be enormous, containing even millions of different labels. Problems of such scale are often referred to as *extreme classification*. Some notable examples of such problems are tagging of text documents (Dekel and Shamir, 2010), content annotation for multimedia search (Deng et al., 2011), and different types of recommendation, including webpages-to-ads (Beygelzimer et al., 2009), ads-to-bid-words (Agrawal et al., 2013; Prabhu and Varma, 2014), users-to-items (Weston et al., 2013; Zhuo et al., 2020), queries-to-items (Medini et al., 2019), or items-to-queries (Chang et al., 2020). In these practical applications, learning algorithms run in rapidly changing environments. Hence, the space of labels and features might grow over time as new data points arrive. Retraining the model from scratch every time a new label is observed is computationally expensive, requires storing all previous data points, and introduces long retention before the model can predict new labels. Therefore, it is desirable for algorithms operating in such a setting to work in an incremental fashion, efficiently adapting to the growing label and feature space.

To tackle extreme classification problems efficiently, we consider a class of label tree algorithms that use a hierarchical structure of classifiers to reduce the computational complexity of training and prediction. The tree nodes contain classifiers that direct the test examples from the root down to the leaf nodes, where each leaf corresponds to one label. We focus on a subclass of label tree algorithms that uses probabilistic classifiers. Examples of such algorithms for multi-class classification include hierarchical softmax (HSM) (Morin and Bengio, 2005), implemented, for example, in fastText (Joulin et al., 2016), and conditional probability estimation tree (CPET) (Beygelzimer et al., 2009). The generalization of this idea to multi-label classification is known under the name of probabilistic label trees (PLTs) (Jasinska et al., 2016), and has been recently implemented in several state-of-the-art algorithms: Parabel (Prabhu et al., 2018), extremeText (Wydmuch et al., 2018), Bonsai Tree (Khandagale et al., 2019), and AttentionXML (You et al., 2019). While some of the above algorithms use incremental procedures to train node classifiers, only CPET allows for extending the model with new labels, but it only works for multi-class classification. For all the other algorithms, a label tree needs to be given before training of the node classifiers.

In this paper, we introduce *online probabilistic label*

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

trees (OPLTs), an algorithm for multi-class and multi-label problems, which trains a label tree classifier in a *fully* online manner. This means that the algorithm does not require any prior knowledge about the number of training instances, their features and labels. The tree is updated every time a new label arrives with a new example, in a similar manner as in CPET (Beygelzimer et al., 2009), but the mechanism used there has been generalized to multi-label data. Also, new features are added when they are observed. This can be achieved by feature hashing (Weinberger et al., 2009) as in the popular Vowpal Wabbit package (Langford et al., 2007). We rely, however, on a different technique based on recent advances in the implementation of hash maps, namely the Robin Hood hashing (Celis et al., 1985).

We require the model trained by OPLT to be equivalent to a model trained as a tree structure would be known from the very beginning. In other words, the node classifiers should be exactly the same as the ones trained on the same sequence of training data using the same incremental learning algorithm, but with the tree produced by OPLT given as an input parameter before training them. We refer to an algorithm satisfying this requirement as a *proper* online PLT. If the incremental tree can be built efficiently, then we additionally say that the algorithm is also *efficient*. These properties are important as a proper and efficient online PLT algorithm possesses similar guarantees as PLTs in terms of computational complexity (Busa-Fekete et al., 2019) and statistical performance (Wydmuch et al., 2018).

To our best knowledge, the only algorithm that also addresses the problem of fully online learning in the extreme multi-class and multi-label setting is the recently introduced contextual memory tree (CMT) (Sun et al., 2019), which is a specific online key-value structure that can be applied to a wide spectrum of online problems. More precisely, CMT stores observed examples in the near-balanced binary tree structure that grows with each new example. The problem of mapping keys to values is converted into a collection of classification problems in the tree nodes, which predict which sub-tree contains the best value corresponding to the key. CMT has been empirically proven to be useful for the few-shot learning setting in extreme multi-class classification, where it has been used directly as a classifier, and for extreme multi-label classification problems, where it has been used to augment an online one-versus-rest (OVR) algorithm. In the experimental study, we compare OPLT with its offline counterparts and CMT on both extreme multi-label classification and few-shot multi-class classification tasks.

Some other existing extreme classification approaches can be tried to be used in the fully online setting, but the adaptation is not straightforward and there does not exist any such algorithm. For example, the efficient OVR approaches (e.g., DiSMEC (Babbar and Schölkopf, 2017), PPDSparse (Yen et al., 2017), ProXML (Babbar and Schölkopf, 2019)) work only in the batch mode. Interestingly, one way of obtaining a fully online OVR is to use OPLT with a 1-level tree. Popular decision-tree-based approaches, such as FastXML (Prabhu and Varma, 2014), also work in the batch mode. An exception is LOMTree (Choromanska and Langford, 2015)), which is an online algorithm. It can be adapted to the fully online setting, but as shown in (Sun et al., 2019) its performance is worse than the one of CMT. Recently, the idea of soft trees, closely related to the hierarchical mixture of experts (Jordan and Jacobs, 1994), has gained increasing attention in the deep learning community (Frosst and Hinton, 2017; Kontschieder et al., 2015; Hehn et al., 2020). However, it has been used neither in the extreme nor in the fully online setting.

The paper is organized as follows. In Section 2, we define the problem of extreme multi-label classification (XMLC). Section 3 recalls the PLT model. Section 4 introduces the OPLT algorithm, defines the desired properties and shows that the introduced algorithm satisfies them. Section 5 presents experimental results. The last section concludes the paper.

2 Extreme multi-label classification

Let $\mathcal{X}$ denote an instance space and $\mathcal{L}$ be a finite set of m labels. We assume that an instance $\boldsymbol{x} \in \mathcal{X}$ is associated with a subset of labels $\mathcal{L}_{\boldsymbol{x}} \subseteq \mathcal{L}$ (the subset can be empty); this subset is often called the set of relevant or positive labels, while the complement $\mathcal{L} \setminus \mathcal{L}_{\boldsymbol{x}}$ is considered as irrelevant or negative for $\boldsymbol{x}$. We identify the set $\mathcal{L}_{\boldsymbol{x}}$ of relevant labels with the binary vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$, in which $y_j = 1 \Leftrightarrow j \in \mathcal{L}_{\boldsymbol{x}}$. By $\mathcal{Y} = \{0,1\}^m$ we denote the set of all possible label vectors. We assume that observations $(\boldsymbol{x}, \boldsymbol{y})$ are generated independently and identically according to a probability distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$ defined on $\mathcal{X} \times \mathcal{Y}$. Notice that the above definition of multi-label classification includes multi-class classification as a special case in which $\|\boldsymbol{y}\|_1 = 1$ ($\|\cdot\|$ denotes a vector norm). In case of XMLC, we assume m to be a large number but the size of the set of relevant labels $\mathcal{L}_{\boldsymbol{x}}$ is usually much smaller than m, that is $|\mathcal{L}_{\boldsymbol{x}}| \ll m$.

3 Probabilistic label trees

We recall the definition of probabilistic label trees (PLTs), introduced in (Jasinska et al., 2016). PLTs follow a label-tree approach to efficiently solve the problem of estimation of marginal probabilities of labels in

Algorithm 1 IPLT.TRAIN($T, A_{\text{online}}, \mathcal{D}$)

$H_T = \emptyset$ ▷ Initialize a set of node probabilistic classifiers
for $v \in V_T$ **do** $\hat{\eta}(v) = \text{NEWCLASSIFIER}()$, $H_T = H_T \cup \{\hat{\eta}(v)\}$ ▷ Initialize binary classifier for each node in the tree
for $i = 1 \to n$ **do** ▷ For each observation in the training sequence
  $(P, N) = \text{ASSIGNTONODES}(T, \boldsymbol{x}_i, \mathcal{L}_{\boldsymbol{x}_i})$ ▷ Compute its positive and negative nodes
  for $v \in P$ **do** $A_{\text{online}}.\text{UPDATE}(\hat{\eta}(v), (\boldsymbol{x}_i, 1))$ ▷ Update all positive nodes with a positive update with $\boldsymbol{x}$
  for $v \in N$ **do** $A_{\text{online}}.\text{UPDATE}(\hat{\eta}(v), (\boldsymbol{x}_i, 0))$ ▷ Update all negative nodes with a negative update with $\boldsymbol{x}$
return H_T ▷ Return the set of node probabilistic classifiers

multi-label problems. They reduce the original problem to a set of binary problems organized in the form of a rooted, leaf-labeled tree with m leaves. We denote a single tree by T, a root node by r_T, and the set of leaves by L_T. The leaf $l_j \in L_T$ corresponds to the label $j \in \mathcal{L}$. The set of leaves of a (sub)tree rooted in node v is denoted by L_v. The set of labels corresponding to all leaf nodes in L_v is denoted by $\mathcal{L}_v$. The parent node of v is denoted by $\text{pa}(v)$, and the set of child nodes by $\text{Ch}(v)$. The path from node v to the root is denoted by $\text{Path}(v)$. The length of the path is the number of nodes on the path, which is denoted by len_v. The set of all nodes is denoted by V_T. The degree of a node $v \in V_T$, being the number of its children, is denoted by $\deg_v = |\text{Ch}(v)|$.

PLT uses tree T to factorize conditional probabilities of labels, $\eta_j(\boldsymbol{x}) = \mathbf{P}(y_j = 1|\boldsymbol{x}) = \mathbf{P}(j \in \mathcal{L}_{\boldsymbol{x}}|\boldsymbol{x})$. To this end let us define for every $\boldsymbol{y}$ a corresponding vector $\boldsymbol{z}$ of length $|V_T|$,whose coordinates, indexed by $v \in V_T$, are given by:

$$z_v = [\![\textstyle\sum_{\ell_j \in L_v} y_j \geq 1]\!] \,. \tag{1}$$

In other words, the element z_v of $\boldsymbol{z}$, corresponding to the node $v \in V_T$, is set to one iff $\boldsymbol{y}$ contains at least one label corresponding to leaves in L_v. With the above definition, it holds for any node $v \in V_T$ that:

$$\eta_v(\boldsymbol{x}) = \mathbf{P}(z_v = 1 \,|\, \boldsymbol{x}) = \prod_{v' \in \text{Path}(v)} \eta(\boldsymbol{x}, v') \,, \tag{2}$$

where $\eta(\boldsymbol{x}, v) = \mathbf{P}(z_v = 1|z_{\text{pa}(v)} = 1, \boldsymbol{x})$ for non-root nodes, and $\eta(\boldsymbol{x}, v) = \mathbf{P}(z_v = 1 \,|\, \boldsymbol{x})$ for the root (see, e.g., Jasinska et al. (2016)). Notice that for leaf nodes we get the conditional probabilities of labels, i.e.,

$$\eta_{l_j}(\boldsymbol{x}) = \eta_j(\boldsymbol{x}) \,, \quad \text{for } l_j \in L_T \,. \tag{3}$$

For a given T it suffices to estimate $\eta(\boldsymbol{x}, v)$, for $v \in V_T$, to build a PLT model. To this end one usually uses a function class $\mathcal{H} : \mathbb{R}^d \mapsto [0, 1]$ which contains probabilistic classifiers of choice, for example, logistic regression. We assign a classifier from $\mathcal{H}$ to each node of the tree T. We index this set of classifiers by the elements of V_T as $H = \{\hat{\eta}(v) \in \mathcal{H} : v \in V_T\}$. Training is performed usually on a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ consisting of n tuples of feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and label vector $\boldsymbol{y}_i \in \{0, 1\}^m$. Because of factorization (2), node classifiers can be trained as independent tasks.

The quality of the estimates $\hat{\eta}_j(\boldsymbol{x})$, $j \in \mathcal{L}$, can be expressed in terms of the L_1-estimation error in each node classifier, i.e., by $|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|$. PLTs obey the following bound (Wydmuch et al., 2018).

Theorem 1. For any tree T and $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$ the following holds for $v \in V_T$:

$$|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \leq \sum_{v' \in \text{Path}(l_j)} \eta_{\text{pa}(v')}(\boldsymbol{x}) \, |\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')| \,,$$

where for the root node $\eta_{\text{pa}(r_T)}(\boldsymbol{x}) = 1$.

Prediction for a test example $\boldsymbol{x}$ relies on searching the tree. For metrics such as precision@k, the optimal strategy is to predict k labels with the highest marginal probability $\eta_j(\boldsymbol{x})$. To this end, the prediction procedure traverses the tree using the uniform-cost search to obtain the top k estimates $\hat{\eta}_j(\boldsymbol{x})$ (see Appendix B for the pseudocode).

4 Online probabilistic label trees

A PLT model can be trained incrementally, on observations from $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, using an incremental learning algorithm A_{online} for updating the tree nodes. Such *incremental* PLT (IPLT) is given in Algorithm 1. In each iteration, it first identifies the set of *positive* and *negative nodes* using the ASSIGNTONODES procedure (see Appendix A for the pseudocode and description). The positive nodes are those for which the current training example is treated as positive (i.e, $(\boldsymbol{x}, z_v = 1)$), while the negative nodes are those for which the example is treated as negative (i.e., $(\boldsymbol{x}, z_v = 0)$). Next, IPLT appropriately updates classifiers in the identified nodes. Unfortunately, the incremental training in IPLT requires the tree structure T to be given in advance.

To construct a tree, at least the number m of labels needs to be known. More advanced tree construction procedures exploit additional information like feature values or label co-occurrence (Prabhu et al., 2018; Khandagale et al., 2019). In all such algorithms, the tree is built prior to the learning of node classifiers.

Algorithm 2 OPLT.Init()

$r_T = \text{NewNode}()$, $V_T = \{r_T\}$ ▷ Create the root of the tree
$\hat{\eta}(r_T) = \text{NewClassifier}()$, $H_T = \{\hat{\eta}_T(r_T)\}$ ▷ Initialize a new classifier in the root
$\hat{\theta}(r_T) = \text{NewClassifier}()$, $\Theta_T = \{\theta(r_T)\}$ ▷ Initialize an auxiliary classifier in the root

Algorithm 3 OPLT.Train($\mathcal{S}, A_{\text{online}}, A_{\text{policy}}$)

for $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t}) \in \mathcal{S}$ **do** ▷ For each observation in $\mathcal{S}$
  if $\mathcal{L}_{\boldsymbol{x}_t} \setminus \mathcal{L}_{t-1} \neq \emptyset$ **then** UpdateTree($\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t}, A_{\text{policy}}$) ▷ If the obser. contains new labels, add them to the tree
  UpdateClassifiers($\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t}, A_{\text{online}}$) ▷ Update the classifiers
  send $H_t, T_t = H_T, V_T$ ▷ Send the node classifiers and the tree structure

Here, we analyze a different scenario in which an algorithm operates on a possibly infinite sequence of training instances, and the tree is constructed online, simultaneously with incremental training of node classifiers, without any prior knowledge of the set of labels or training data.

Let us denote a sequence of observations by $\mathcal{S} = \{(\boldsymbol{x}_i, \mathcal{L}_{\boldsymbol{x}_i})\}_{i=1}^{\infty}$ and a subsequence consisting of the first t instances by $\mathcal{S}_t$. We use here $\mathcal{L}_{\boldsymbol{x}_i}$ instead of $\boldsymbol{y}_i$ as the number of labels m, which is also the length of $\boldsymbol{y}_i$, increases over time in this online scenario.[1] Furthermore, let the set of labels observed in $\mathcal{S}_t$ be denoted by $\mathcal{L}_t$, with $\mathcal{L}_0 = \emptyset$. An online algorithm returns at step t a tree structure T_t constructed over labels in $\mathcal{L}_t$ and a set of node classifiers H_t. Notice that the tree structure and the set of classifiers change in each iteration in which one or more new labels are observed. Below we discuss two properties that are desired for such online algorithms, defined in relation to the IPLT algorithm given above.

Definition 1 (A proper online PLT algorithm)**.** Let T_t and H_t be respectively a tree structure and a set of node classifiers trained on a sequence $\mathcal{S}_t$ using an online algorithm B. We say that B is a *proper online* PLT *algorithm*, when for any $\mathcal{S}$ and t we have that

- $l_j \in L_{T_t}$ iff $j \in \mathcal{L}_t$, i.e., leaves of T_t correspond to all labels observed in S_t,
- and H_t is exactly the same as $H = \text{IPLT.Train}(T_t, A_{\text{online}}, \mathcal{S}_t)$, i.e., node classifiers from H_t are the same as the ones trained incrementally by Algorithm 1 on $\mathcal{D} = \mathcal{S}_t$ and tree T_t given as input parameter.

In other words, we require that whatever tree the online algorithm produces, the node classifiers should be trained in the same way as if the tree was known from the very beginning of training. Thanks to that, we can control the quality of each node classifier, as we are not missing any update. Since the model produced by a proper online PLT is the same as of IPLT, the same statistical guarantees apply to both of them.

The above definition can be satisfied by a naive algorithm that stores all observations seen so far, uses them in each iteration to build a tree and train node classifiers with the IPLT algorithm from scratch. This approach is costly. Therefore, we also demand an online algorithm to be space and time-efficient in the following sense.

Definition 2 (An efficient online PLT algorithm)**.** Let T_t and H_t be respectively a tree structure and a set of node classifiers trained on a sequence $\mathcal{S}_t$ using an online algorithm B. Let C_s and C_t be the space and time training cost of IPLT trained on sequence $\mathcal{S}_t$ and tree T_t. An online algorithm is an *efficient online* PLT *algorithm* when for any S and t we have its space and time complexity to be in constant factor of C_s and C_t, respectively.

In this definition, we abstract from the actual implementation of IPLT. In other words, the complexity of an efficient online PLT algorithm depends directly on design choices for IPLT. The space complexity is upper bounded by $2m - 1$ (i.e., the maximum number of node models), but it also depends on the chosen type of node models and the way of storing them. Let us also notice that the definition implies that the update of a tree structure has to be in a constant factor of the training cost of a single instance.

4.1 Online tree building and training of node classifiers

Below we describe an online PLT algorithm that, as we show in subsection 4.3, satisfies both properties defined above. It is similar to CPET (Beygelzimer et al., 2009), but extends it to multi-label problems and trees of any shape. We refer to this algorithm as OPLT.

The pseudocode is presented in Algorithms 2-7. In a nutshell, OPLT proceeds observations from $\mathcal{S}$ se-

[1] The same applies to $\boldsymbol{x}_i$ as the number of features also increases. However, we keep the vector notation in this case, as it does not impact the algorithm's description.

Algorithm 4 OPLT.UpdateTree($\boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}}, A_{\text{policy}}$)

for $j \in \mathcal{L}_{\boldsymbol{x}} \setminus \mathcal{L}_{t-1}$ **do** ▷ For each new label in the observation
  if $\mathcal{L}_T = \emptyset$ **then** Label(r_T) $= j$ ▷ If no labels have been seen so far, assign label j to the root node
  else ▷ If there are already labels in the tree
    v, $insert = A_{\text{policy}}(\boldsymbol{x}, j, \mathcal{L}_{\boldsymbol{x}})$ ▷ Select a variant of extending the tree
    if $insert$ **then** InsertNode(v) ▷ Insert an additional node if needed
    AddLeaf(j, v) ▷ Add a new leaf for label j

Algorithm 5 OPLT.InsertNode(v)

$v' =$ NewNode(), $V_T = V_T \cup \{v'\}$ ▷ Create a new node and add it to the tree nodes
if IsLeaf(v) **then** Label(v') = Label(v), Label(v) = Null ▷ If node v is a leaf reassign label of v to v'
else ▷ Otherwise
  $\text{Ch}(v') = \text{Ch}(v)$ ▷ All children of v become children of v'
  for $v_{\text{ch}} \in \text{Ch}(v')$ **do** $\text{pa}(v_{\text{ch}}) = v'$ ▷ And v' becomes their parent
$\text{Ch}(v) = \{v'\}$, $\text{pa}(v') = v$ ▷ The new node v' becomes the only child of v
$\hat{\eta}(v') =$ Copy($\hat{\theta}(v)$), $H_T = H_T \cup \{\hat{\eta}(v')\}$ ▷ Create a classifier
$\hat{\theta}(v') =$ Copy($\hat{\theta}(v)$), $\Theta_T = \Theta_T \cup \{\hat{\theta}(v')\}$ ▷ And an auxiliary classifier

quentially, updating node classifiers. For new incoming labels, it creates new nodes according to a chosen tree building policy which is responsible for the main logic of the algorithm. Each new node v is associated with two classifiers, a regular one $\hat{\eta}(v) \in H_T$, and an *auxiliary* one $\hat{\theta}(v) \in \Theta_T$, where H_T and Θ_T denote the corresponding sets of node classifiers. The task of the auxiliary classifiers is to accumulate positive updates. The algorithm uses them later to initialize classifiers associated with new nodes added to a tree. They can be removed if a given node will not be used anymore to extend the tree. A particular criterion for removing an auxiliary classifier depends, however, on a tree building policy.

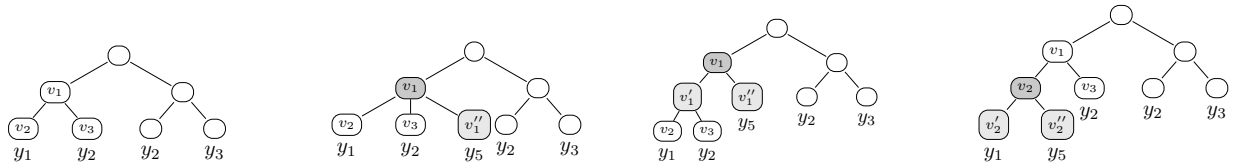
The algorithm starts with OPLT.Init procedure, presented in Algorithm 2, that initializes a tree with only a root node v_{r_T} and corresponding classifiers, $\hat{\eta}(v_{r_T})$ and $\hat{\theta}(v_{r_T})$. Notice that the root has both classifiers initialized from the very beginning without a label assigned to it. Thanks to this, the algorithm can properly estimate the probability of $\mathbf{P}(\boldsymbol{y} = \mathbf{0} \,|\, \boldsymbol{x})$. From now on, OPLT.Train, outlined in Algorithm 3, administrates the entire process. In its main loop, observations from $\mathcal{S}$ are proceeded sequentially. If a new observation contains one or more new labels then the tree structure is appropriately extended by calling UpdateTree. The node classifiers are updated in UpdateClassifiers. After each iteration t, the algorithm sends H_T along with the tree structure T, respectively as H_t and T_t, to be used outside the algorithm for prediction tasks. We assume that tree T along with sets of its all nodes V_T and leaves L_T, as well as sets of classifiers H_T and Θ_T, are accessible to all the algorithms discussed below.

Algorithm 4, UpdateTree, builds the tree structure. It iterates over all new labels from $\mathcal{L}_{\boldsymbol{x}}$. If there were no labels in the sequence $\mathcal{S}$ before, the first new label taken from $\mathcal{L}_{\boldsymbol{x}}$ is assigned to the root node. Otherwise, the tree needs to be extended by one or two nodes according to a selected tree building policy. One of these nodes is a leaf to which the new label will be assigned. There are, in general, three variants of performing this step illustrated in Figure 1. The first one relies on selecting an internal node v whose number of children is lower than the accepted maximum, and adding to it a child node v'' with the new label assigned to it. In the second one, two new child nodes, v' and v'', are added to a selected internal node v. Node v' becomes a new parent of child nodes of the selected node v, i.e., the subtree of v is moved down by one level. Node v'' is a leaf with the new label assigned to it. The third variant is a modification of the second one. The difference is that the selected node v is a leaf node. Therefore there are no children nodes to be moved to v', but label of v is reassigned to v'. The A_{policy} method encodes the tree building policy, i.e., it decides which of the three variants to follow and selects the node v. The additional node v' is inserted by the InsertNode method. Finally, a leaf node is added by the AddLeaf method. We discuss the three methods in more detail below.

A_{policy} returns the selected node v and a Boolean variable $insert$, which indicates whether an additional node v' has to be added to the tree. For the first variant, v is an internal node, and $insert$ is set to false. For the second variant, v is an internal node, and $insert$ is set to true. For the third variant, v is a leaf node, and $insert$ is set to true. In general, the policy can be as simple as selecting a random node or a node based on the current tree size to construct a complete tree. It can also be much more complex, guided in general by $\boldsymbol{x}$, current label j, and set $\mathcal{L}_{\boldsymbol{x}}$ of all labels of $\boldsymbol{x}$. Nevertheless, as mentioned before, the complexity of this

Algorithm 6 OPLT.AddLeaf(j, v)

$v'' = \text{NewNode}()$, $V_T = V_T \cup \{v''\}$ ▷ Create a new node and add it to the tree nodes
$\text{Ch}(v) = \text{Ch}(v) \cup \{v''\}$, $\text{pa}(v'') = v$, $\text{Label}(v'') = j$ ▷ Add this node to children of v and assign label j to the node v''
$\hat{\eta}(v'') = \text{InverseClassifier}(\hat{\theta}(v))$, $H_T = H_T \cup \{\hat{\eta}(v'')\}$ ▷ Initialize a classifier for v''
$\hat{\theta}(v'') = \text{NewClassifier}()$, $\Theta_T = \Theta_T \cup \{\hat{\theta}(v'')\}$ ▷ Initialize an auxiliary classifier for v''



(a) Tree T_{t-1} after $t-1$ iterations.

(b) Variant 1: A leaf node v_1'' for label j added as a child of an internal node v_1.

(c) Variant 2: A leaf node v_1'' for label j and an internal node v_1' (with all children of v_1 reassigned to it) added as children of v_1.

(d) Variant 3: A leaf node v_2'' for label j and a leaf node v_2' (with a reassigned label of v_2) added as children of v_2.

Figure 1: Three variants of tree extension for a new label j.

step should be at most proportional to the complexity of updating the node classifiers for one label, i.e., it should be proportional to the depth of the tree. We propose two such policies in the next subsection.

The InsertNode and AddLeaf procedures involve specific operations initializing classifiers in the new nodes. InsertNode is given in Algorithm 5. It inserts a new node v' as a child of the selected node v. If v is a leaf, then its label is reassigned to the new node. Otherwise, all children of v become the children of v'. In both cases, v' becomes the only child of v. Figure 1 illustrates inserting v' as either a child of an internal node (c) or a leaf node (d). Since, the node classifier of v' aims at estimating $\eta(\boldsymbol{x}, v')$, defined as $\mathbf{P}(z_{v'} = 1 \,|\, z_{\text{pa}(v')} = 1, \boldsymbol{x})$, its both classifiers, $\hat{\eta}(v')$ and $\hat{\theta}(v')$, are initialized as copies (by calling the Copy function) of the auxiliary classifier $\hat{\theta}(v)$ of the parent node v. Recall that the task of auxiliary classifiers is to accumulate all positive updates in nodes, so the conditioning $z_{\text{pa}(v')} = 1$ is satisfied in that way.

Algorithm 6 outlines the AddLeaf procedure. It adds a new leaf node v'' for label j as a child of node v. The classifier $\hat{\eta}(v'')$ is created as an "inverse" of the auxiliary classifier $\hat{\theta}(v)$ from node v. More precisely, the InverseClassifier procedure creates a wrapper inverting the behavior of the base classifier. It predicts $1 - \hat{\eta}$, where $\hat{\eta}$ is the prediction of the base classifier, and flips the updates, i.e., positive updates become negative and negative updates become positive. Finally, the auxiliary classifier $\hat{\theta}(v'')$ of the new leaf node is initialized.

The final step in the main loop of OPLT.Train updates the node classifiers. The regular classifiers, $\hat{\eta}(v) \in H_T$, are updated exactly as in IPLT.Train given in Algorithm 1. The auxiliary classifiers, $\theta(v) \in \Theta_T$, are updated only in positive nodes according to their definition and purpose.

Notice that OPLT.Train can also be run without prior initialization with OPLT.Init if only a tree with properly trained node and auxiliary classifiers is provided. One can create such a tree using a set of already available observations $\mathcal{D}$ and then learn node and auxiliary classifiers using the same OPLT.Train algorithm. Because all labels from $\mathcal{D}$ should be present in the created tree, it is not updated by the algorithm. From now on, OPLT.Train can be used again to correctly update the tree for new observations.

4.2 Random and best-greedy policy

We discuss two policies A_{policy} for OPLT that can be treated as non-trivial generalization of the policy used in CPET to the multi-label setting. CPET builds a binary balanced tree by expanding leaf nodes, which corresponds to the use of the third variant of the tree structure extension only. As the result, it gradually moves away labels that initially have been placed close to each other. Particularly, labels of the first observed examples will finally end in leaves at the opposite sides of the tree. This may result in lowering the predictive performance and increasing training and prediction times. To address these issues, we introduce a solution, inspired by (Prabhu et al., 2018; Wydmuch et al., 2018), in which pre-leaf nodes, i.e., parents of leaf nodes, can be of much higher arity than the other internal nodes. In general, we guarantee that the arity of each pre-leaf node is upper bounded by $b_{\max}$, while all other internal nodes by b, where $b_{\max} \geq b$.

Both policies, presented jointly in Algorithm 8, start

Algorithm 7 OPLT.UpdateClassifiers($\boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}}, A_{\text{online}}$)

$(P, N) = \text{AssignToNodes}(T, \boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}})$ ▷ Compute its positive and negative nodes
for $v \in P$ **do** ▷ For all positive nodes
  $A_{\text{online}}.\text{Update}(\hat{\eta}(v), (\boldsymbol{x}, 1))$ ▷ Update classifiers with a positive update with $\boldsymbol{x}$
  if $\hat{\theta}(v) \in \Theta$ **then** $A_{\text{online}}.\text{Update}(\hat{\theta}(v), (\boldsymbol{x}, 1))$ ▷ If aux. classifier exists, update it with a positive update with $\boldsymbol{x}_i$
for $v \in N$ **do** $A_{\text{online}}.\text{Update}(\hat{\eta}(v), (\boldsymbol{x}, 0))$ ▷ Update all negative nodes with a negative update with $\boldsymbol{x}$

Algorithm 8 Random and Best-greedy $A_{\text{policy}}(\boldsymbol{x}, j, \mathcal{L}_{\boldsymbol{x}})$

if RunFirstFor($\boldsymbol{x}$) **then** ▷ If the algorithm is run for the first time for the current observation $\boldsymbol{x}$
  $v = r_T$ ▷ Set current node v to root node
  while $\text{Ch}(v) \nsubseteq L_T \wedge \text{Ch}(v) = b$ **do** ▷ While the node's children are not only leaf nodes and arity is equal to b
    if Random policy **then** $v = \text{SelectRandomly}(\text{Ch}(v))$ ▷ If Random policy, randomly choose child node
    else if Best-greedy policy **then** ▷ In the case of Best-greedy policy
      $v = \arg\max_{v' \in \text{Ch}(v)} (1-\alpha)\hat{\eta}(\boldsymbol{x}, v') + \alpha |L_{v'}|^{-1} (\log |L_v| - \log |\text{Ch}(v)|)$ ▷ Select child node with the best score
else $v = \text{GetSelectedNode}()$ ▷ If the same $\boldsymbol{x}$ is observed as the last time, select the node used previously
if $|\text{Ch}(v) \cap L_T| = 1$ **then** $v = v' \in \text{Ch}(v) : v' \in L_T$ ▷ If node v has only one leaf, change the selected node to this leaf
SaveSelectedNode(v) ▷ Save the selected node v
return $(v, |\text{Ch}(v)| = b_{\max} \vee v \subseteq L_T)$ ▷ Return node v, if num. of v's children reached the max. or v is a leaf, insert a new node

with selecting one of the pre-leaves. The first policy traverses a tree from top to bottom by randomly selecting child nodes. The second policy, in turn, selects a child node using a trade-off between the balancedness of the tree and fit of $\boldsymbol{x}$, i.e., the value of $\hat{\eta}(\boldsymbol{x}, v)$:

$$\text{score}_v = (1-\alpha)\hat{\eta}(\boldsymbol{x}, v) + \alpha \frac{1}{|L_v|} \log \frac{|L_{\text{pa}(v)}|}{|\text{Ch}(\text{pa}(v))|},$$

where α is a trade-off parameter. It is worth to notice that both policies work in logarithmic time of the number of internal nodes. Moreover, we run this selection procedure only once for the current observation, regardless of the number of new labels. If the selected node v has fewer leaves than b_{max}, both policies follow the first variant of the tree extension, i.e., they add a new child node with the new label assigned to node v. Otherwise, the policies follow the second variant, in which additionally, a new internal node is added as a child of v with all its children inherited. In case the selected node has only one leaf node among its children, which only happens after adding a new label with the second variant, the policy changes the selected node v to the previously added leaf node.

The above policies have two advantages over CPET. Firstly, new labels coming with the same observation should stay close to each other in the tree. Secondly, the policies allow for efficient management of auxiliary classifiers, which basically need to reside only in pre-leaf nodes, with the exception of leaf nodes added in the second variant. The original CPET algorithm needs to maintain auxiliary classifiers in all leaf nodes.

4.3 Theoretical analysis of OPLT

The OPLT algorithm has been designed to satisfy the properness and efficiency property. The theorem below states this fact formally.

Theorem 2. OPLT is a proper and efficient online PLT algorithm.

We present the proof in Appendix C. To show the properness, it uses induction for both the outer and inner loop of the algorithm, where the outer loop iterates over observations $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$, while the inner loop over new labels in $\mathcal{L}_{\boldsymbol{x}_t}$. The key elements to prove this property are the use of the auxiliary classifiers and the analysis of the three variants of the tree structure extension. The efficiency is proved by noticing that the algorithm creates up to two new nodes per new label, each node having at most two classifiers. Therefore, the number of updates is no more than twice the number of updates in IPLT. Moreover, any node selection policy in which cost is proportional to the cost of updating IPLT classifiers for a single label meets the efficiency requirement. Notably, the policies presented above satisfy this constraint. Note that training of IPLT can be performed in logarithmic time in the number of labels under the additional assumption of using a balanced tree with constant nodes arity (Busa-Fekete et al., 2019). Because presented policies aim to build trees close to balanced, the time complexity of the OPLT training should also be close to logarithmic in the number of labels.

5 Experiments

In this section, we empirically compare OPLT and CMT on two tasks, extreme multi-label classification and few-shot multi-class classification. We implemented OPLT in C++, based on recently introduced

Table 1: Datasets used for experiments on extreme multi-label classification task and few-shot multi-class classification task. Notation: N – number of samples, m – number of labels, d – number of features, S – shot

Dataset	N_{train}	N_{test}	m	d
AmazonCat	1186239	306782	13330	203882
Wiki10	14146	6616	30938	101938
WikiLSHTC	1778351	587084	325056	1617899
Amazon	490449	153025	670091	135909
ALOI	97200	10800	1001	129
WikiPara-S	$S \times 10000$	10000	10000	188084

napkinXC (Jasinska-Kobus et al., 2020).[2] We use online logistic regression to train node classifiers with the AdaGrad (Duchi et al., 2011) updates.

For CMT, we use a Vowpal Wabbit (Langford et al., 2007) implementation (also in C++), provided by courtesy of its authors. It uses linear models also incrementally updated by AdaGrad, but all model weights are stored in one large continuous array using the hashing trick. However, it requires at least some prior knowledge about the size of the feature space since the size of the array must be determined beforehand, which can be hard in a fully online setting. To address the problem of unknown features space, we store weights of OPLT in an easily extendable hash map based on Robin Hood Hashing (Celis et al., 1985), which ensures very efficient insert and find operations. Since the model sparsity increases with the depth of a tree for sparse data, this solution might be much more efficient in terms of used memory than the hashing trick and does not negatively impact predictive performance.

For all experiments, we use the same, fixed hyper-parameters for OPLT. We set learning rate to 1, AdaGrad's ϵ to 0.01 and the tree balancing parameter α to 0.75, since more balanced trees yield better predictive performance (see Appendix F for empirical evaluation of the impact of parameter α on precision at k, train and test times, and the tree depth). The only exception is the degree of pre-leaf nodes, which we set to 100 in the XMLC experiment, and to 10 in the few-shot multi-class classification experiment. For CMT we use hyper-parameters suggested by the authors. According to the appendix of (Sun et al., 2019), CMT achieves the best predictive performance after 3 passes over training data. For this reason, we give all algorithms the maximum of 3 such passes and report the best results (see Appendix D and E for the detailed results after 1 and 3 passes). We repeated all the experiments 5 times, each time shuffling the training set and report the mean performance. We performed all the experiments on an Intel Xeon E5-2697 v3 2.6GHz machine with 128GB of memory.

[2]Repository with the code and scripts to reproduce the experiments: https://github.com/mwydmuch/napkinXC

5.1 Extreme multi-label classification

In the XMLC setting, we compare performance in terms of precision at $\{1, 3, 5\}$ and the training time (see Appendix D for prediction times and propensity scored precision at $\{1, 3, 5\}$) on four benchmark datasets: AmazonCat, Wiki10, WikiLSHTC and Amazon, taken from the XMLC repository (Bhatia et al., 2016). We use the original train and test splits. Statistics of these datasets are included in Table 1. In this setting, CMT has been originally used to augment an online one-versus-rest (OVR) algorithm. In other words, it can be treated as a specific index that enables fast prediction and speeds up training by performing a kind of negative sampling. In addition to OPLT and CMT we also report results of IPLT and Parabel (Prabhu et al., 2018). IPLT is implemented similarly to OPLT, but uses a tree structure built-in offline mode. Parabel is, in turn, a fully batch variant of PLT. Not only the tree structure, but also node classifiers are trained in the batch mode using the LIBLINEAR library (Fan et al., 2008). We use a single tree variant of this algorithm. Both IPLT and Parabel are used with the same tree building algorithm, which is based on a specific hierarchical 2-means clustering of labels (Prabhu et al., 2018). Additionally, we report the results of an OPLT with warm-start (OPLT-W) that is first trained on a sample of 10% of training examples and a tree created using hierarchical 2-means clustering on the same sample. After this initial phase, OPLT-W is trained on the remaining 90% of data using the Best-greedy policy (see Appendix G for the results of OPLT-W trained with different sizes of the warm-up sample and comparison with IPLT trained only on the same warm-up sample).

Results of the comparison are presented in Table 2. Unfortunately, CMT does not scale very well in the number of labels nor in the number of examples, resulting in much higher memory usage for massive datasets. Therefore, we managed to obtain results only for Wiki10 and AmazonCat datasets using all available 128GB of memory. OPLT with both extension policies achieves results as good as Parabel and IPLT and significantly outperforms CMT on AmazonCat and Wiki10 datasets. For larger datasets OPLT with Best-greedy policy outperforms the Random policy but obtains worse results than its offline counterparts, with trees built with hierarchical 2-means clustering, especially on the WikiLSHTC dataset. OPLT-W, however, achieves results almost as good as IPLT what proves that good initial structure, even with only some labels, helps to

Table 2: Mean precision at $\{1, 3, 5\}$ (%) and CPU train time of PARABEL, IPLT, CMT, OPLT for XMLC tasks.

Algo.	AmazonCat				Wiki10				WikiLSHTC				Amazon			
	P@1	P@3	P@5	t_{train}	P@1	P@3	P@5	t_{train}	P@1	P@3	P@5	t_{train}	P@1	P@3	P@5	t_{train}
PARABEL	92.58	78.53	63.90	10.8m	84.17	72.12	63.30	4.2m	62.78	41.22	30.27	14.4m	43.13	37.94	34.00	7.2m
IPLT	93.11	78.72	63.98	34.2m	84.87	74.42	65.31	18.3m	60.80	39.58	29.24	175.1m	43.55	38.69	35.20	79.4m
CMT	89.43	70.49	54.23	168.2m	80.59	64.17	55.25	35.1m	-	-	-	-	-	-	-	-
OPLT_R	92.66	77.44	62.52	99.5m	84.34	73.73	64.31	30.3m	47.76	30.97	23.37	330.1m	38.42	34.33	31.32	134.2m
OPLT_B	92.74	77.74	62.91	84.1m	84.47	73.73	64.39	27.7m	54.69	35.32	26.31	300.0m	41.09	36.65	33.42	111.9m
OPLT-W	93.14	78.68	63.92	43.7m	85.22	74.68	64.93	28.2m	59.23	38.39	28.38	205.7m	42.21	37.60	34.25	98.3m

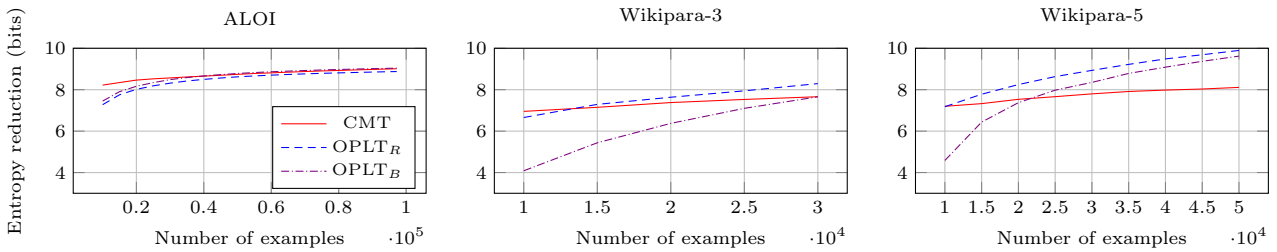


Figure 2: Online progressive performance of CMT and OPLT with respect to the number of samples on few-shot multi-class classification tasks.

Table 3: Mean accuracy of prediction (%) and train CPU time of CMT, OPLT for few-shot multi-class classification tasks.

Algo.	ALOI		Wikipara-3		Wikipara-5	
	Acc.	t_{train}	Acc.	t_{train}	Acc.	t_{train}
CMT	71.98	207.1s	3.28	63.1s	4.21	240.9s
OPLT_R	66.50	20.3s	27.34	16.4s	40.67	27.6s
OPLT_B	67.26	18.1s	24.66	15.6s	39.13	27.2s

build a good tree in an online way. In terms of training times, OPLT, as expected, is slower than IPLT due to additional auxiliary classifiers and worse tree structure, both leading to a larger number of updates.

5.2 Few-shot multi-class classification

In the second experiment, we compare OPLT with CMT on three few-shot learning multi-class datasets: ALOI (Geusebroek et al., 2005), 3 and 5-shot versions of WikiPara. Statistics of these datasets are also included in Table 1. CMT has been proven in (Sun et al., 2019) to perform better than two other logarithmic-time online multi-class algorithms, LOMTREE (Choromanska and Langford, 2015) and RECALL TREE (Daumé et al., 2017) on these specific datasets. We use here the same version of CMT as used in a similar experiment in the original paper (Sun et al., 2019).

Since OPLT and CMT operate online, we compare their performance in two ways: 1) using online progressive validation (Blum et al., 1999), where each example is tested ahead of training and 2) using offline evaluation on the test set after seeing the whole training set. Figure 2 summarizes the results in terms of progressive performance. In the same fashion as in (Sun et al., 2019), we report entropy reduction of accuracy from the constant predictor, calculated as $\log_2(\text{Acc}_{\text{algo}}) - \log_2(\text{Acc}_{\text{const}})$, where Acc_{algo} and $\text{Acc}_{\text{const}}$ is mean accuracy of the evaluated algorithm and the constant predictor. In Table 3, we report results on the test datasets. In online and offline evaluation, OPLT performs similar to CMT on ALOI dataset, while it significantly dominates on the WikiPara datasets.

6 Conclusions

In this paper, we introduced online probabilistic label trees, an algorithm that trains a label tree classifier in a fully online manner, without any prior knowledge about the number of training instances, their features and labels. OPLTs can be used for both multi-label and multi-class classification. They outperform CMT in almost all experiments, scaling at the same time much more efficiently on tasks with a large number of examples, features and labels.

Acknowledgments

Computational experiments have been performed in Poznan Supercomputing and Networking Center.

References

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. (2013). Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 13–24. International World Wide Web Conferences Steering Committee / ACM.

Babbar, R. and Schölkopf, B. (2017). Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*, pages 721–729. ACM.

Babbar, R. and Schölkopf, B. (2019). Data scarcity, robustness and extreme multi-label classification. *Machine Learning*, 108.

Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G. B., and Strehl, A. L. (2009). Conditional probability tree estimation analysis and algorithms. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 51–58. AUAI Press.

Bhatia, K., Dahiya, K., Jain, H., Mittal, A., Prabhu, Y., and Varma, M. (2016). The extreme classification repository: Multi-label datasets and code.

Blum, A., Kalai, A., and Langford, J. (1999). Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, COLT '99, page 203–208, New York, NY, USA. Association for Computing Machinery.

Busa-Fekete, R., Dembczynski, K., Golovnev, A., Jasinska, K., Kuznetsov, M., Sviridenko, M., and Xu, C. (2019). On the computational complexity of the probabilistic label tree algorithms.

Celis, P., Larson, P.-A., and Munro, J. I. (1985). Robin hood hashing. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, page 281–288, USA. IEEE Computer Society.

Chang, W., Yu, H., Zhong, K., Yang, Y., and Dhillon, I. S. (2020). Taming pretrained transformers for extreme multi-label text classification. In Gupta, R., Liu, Y., Tang, J., and Prakash, B. A., editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3163–3171. ACM.

Choromanska, A. and Langford, J. (2015). Logarithmic time online multiclass prediction. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 55–63.

Daumé, III, H., Karampatziakis, N., Langford, J., and Mineiro, P. (2017). Logarithmic time one-against-some. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 923–932, International Convention Centre, Sydney, Australia. PMLR.

Dekel, O. and Shamir, O. (2010). Multiclass-multilabel classification with more classes than examples. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 137–144. JMLR.org.

Deng, J., Satheesh, S., Berg, A. C., and Li, F. (2011). Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 567–575.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Fan, R., Chang, K., Hsieh, C., Wang, X., and Lin, C. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Frosst, N. and Hinton, G. E. (2017). Distilling a neural network into a soft decision tree. *CoRR*, abs/1711.09784.

Geusebroek, J.-M., Burghouts, G., and Smeulders, A. (2005). The amsterdam library of object images. *Int. J. Comput. Vision*, 61(1):103–112.

Hehn, T. M., Kooij, J. F. P., and Hamprecht, F. A. (2020). End-to-end learning of decision trees and forests. *Int. J. Comput. Vis.*, 128(4):997–1011.

Jain, H., Prabhu, Y., and Varma, M. (2016). Extreme multi-label loss functions for recommendation, tagging, ranking and other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 935–944, New York, NY, USA. Association for Computing Machinery.

Jasinska, K., Dembczynski, K., Busa-Fekete, R., Pfannschmidt, K., Klerx, T., and Hüllermeier, E. (2016). Extreme F-measure maximization using sparse probability estimates. In *Proceedings of the*

33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1435–1444. JMLR.org.

Jasinska-Kobus, K., Wydmuch, M., Dembczyński, K., Kuznetsov, M., and Busa-Fekete, R. (2020). Probabilistic label trees for extreme multi-label classification. *CoRR*, abs/2009.11218.

Jordan, M. and Jacobs, R. (1994). Hierarchical mixtures of experts and the. *Neural computation*, 6:181–.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

Khandagale, S., Xiao, H., and Babbar, R. (2019). Bonsai - diverse and shallow trees for extreme multi-label classification. *CoRR*, abs/1904.08249.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bulò, S. R. (2015). Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1467–1475. IEEE Computer Society.

Langford, J., Strehl, A., and Li, L. (2007). Vowpal wabbit. https://vowpalwabbit.org.

Medini, T. K. R., Huang, Q., Wang, Y., Mohan, V., and Shrivastava, A. (2019). Extreme classification in log memory using count-min sketch: A case study of amazon search with 50m products. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 13265–13275. Curran Associates, Inc.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*. Society for Artificial Intelligence and Statistics.

Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., and Varma, M. (2018). Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 993–1002. ACM.

Prabhu, Y. and Varma, M. (2014). Fastxml: a fast, accurate and stable tree-classifier for extreme multi-label learning. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 263–272. ACM.

Sun, W., Beygelzimer, A., Iii, H. D., Langford, J., and Mineiro, P. (2019). Contextual memory trees. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6026–6035, Long Beach, California, USA. PMLR.

Weinberger, K. Q., Dasgupta, A., Langford, J., Smola, A. J., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 1113–1120. ACM.

Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 181–189. JMLR.org.

Wydmuch, M., Jasinska, K., Kuznetsov, M., Busa-Fekete, R., and Dembczynski, K. (2018). A no-regret generalization of hierarchical softmax to extreme multi-label classification. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 6355–6366. Curran Associates, Inc.

Yen, I. E., Huang, X., Dai, W., Ravikumar, P., Dhillon, I. S., and Xing, E. P. (2017). Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 545–553. ACM.

You, R., Zhang, Z., Wang, Z., Dai, S., Mamitsuka, H., and Zhu, S. (2019). Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 5812–5822. Curran Associates, Inc.

Zhuo, J., Xu, Z., Dai, W., Zhu, H., Li, H., Xu, J., and Gai, K. (2020). Learning optimal tree models under beam search. In *Proceedings of the 37th International Conference on Machine Learning*, Vienna, Austria. PMLR.

A Training in PLT

The pseudocode with a brief description of the training algorithm for IPLTs is given in the main text. Here we discuss it in more detail. The algorithm first initializes all node classifiers. The training relies on a proper assignment of training examples to nodes. To train probabilistic classifiers $\hat{\eta}(v)$, $v \in V_T$, in all nodes of a tree T, we need to properly filter training examples as given in (2). In each iteration the algorithm identifies for a given training example a set of *positive* and *negative nodes*, i.e., the nodes for which a training example is treated respectively as positive (i.e, $(\boldsymbol{x}, z_v = 1)$), or negative (i.e., $(\boldsymbol{x}, z_v = 0)$). The AssignToNodes method is given in Algorithm 9. It initializes the positive nodes to the empty set and the negative nodes to the root node (to deal with $\boldsymbol{y}$ of all zeros). Next, it traverses the tree from the leaves corresponding to the labels of the training example to the root adding the visited nodes to the set of positive nodes. It also removes each visited node from the set of negative nodes, if it has been added to this set before. All children of the visited node, which are not in the set of positive nodes, are then added to the set of negative nodes. If the parent node of the visited node has already been added to positive nodes, the traversal on this path stops.

By using the positive and negative nodes, the procedure updates the corresponding classifiers with an online learner A_{online} of choice. Note that all node classifiers are trained in fact independently as updates in any node do not influence the training of the other nodes. The output of the algorithm is a set of probabilistic classifiers H.

Algorithm 9 IPLT/OPLT.AssignToNodes$(T, \boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}})$

$P = \emptyset$, $N = \{r_T\}$ ▷ Initialize sets of positive and negative nodes
for $j \in \mathcal{L}_{\boldsymbol{x}}$ **do** ▷ For all labels of the training example
  $v = \ell_j$ ▷ Set v to a leaf corresponding to label j
  while $v \neq$ Null **and** $v \notin P$ **do** ▷ On a path to the root or the first positive node (excluded)
    $P = P \cup \{v\}$ ▷ Assign a node to positive nodes
    $N = N \setminus \{v\}$ ▷ Remove the node from negative nodes if added there before
    for $v' \in \text{Ch}(v)$ **do** ▷ For all its children
      if $v' \notin P$ **then** $N = N \cup \{v'\}$ ▷ If a child is not a positive node assign it to negative nodes
    $v = \text{pa}(v)$ ▷ Move up along the path
return (P, N) ▷ Return a set of positive and negative nodes for the training example

B Prediction in PLT

Algorithm 10 outlines the prediction procedure for PLTs that returns top k labels. It is based on the uniform-cost search. Alternatively, one can use beam search.

Algorithm 10 IPLT/OPLT.PredictTopLabels$(T, H, k, \boldsymbol{x})$

$\hat{\boldsymbol{y}} = \boldsymbol{0}$, $\mathcal{Q} = \emptyset$, ▷ Initialize prediction vector to all zeros and a priority queue, ordered descending by $\hat{\eta}_v(\boldsymbol{x})$
$k' = 0$ ▷ Initialize counter of predicted labels
$\mathcal{Q}$.add$((r_T, \hat{\eta}(\boldsymbol{x}, r_T)))$ ▷ Add the tree root with the corresponding estimate of probability
while $k' < k$ **do** ▷ While the number of predicted labels is less than k
  $(v, \hat{\eta}_v(\boldsymbol{x})) = \mathcal{Q}$.pop() ▷ Pop the top element from the queue
  if IsLeaf(v) **then** ▷ If the node is a leaf
    $\hat{y}_{\text{Label}(v)} = 1$ ▷ Set the corresponding label in the prediction vector
    $k' = k' + 1$ ▷ Increment the counter
  else ▷ If the node is an internal node
    for $v' \in \text{Ch}(v)$ **do** ▷ For all child nodes
      $\hat{\eta}_{v'}(\boldsymbol{x}) = \hat{\eta}_v(\boldsymbol{x}) \times \hat{\eta}(\boldsymbol{x}, v')$ ▷ Compute $\hat{\eta}_{v'}(\boldsymbol{x})$ using $\hat{\eta}(v') \in H$
      $\mathcal{Q}$.add$((v', \hat{\eta}_{v'}(\boldsymbol{x})))$ ▷ Add the node and the computed probability estimate
return $\hat{\boldsymbol{y}}$ ▷ Return the prediction vector

C The proof of the result from Section 4.3

Theorem 2 concerns two properties, properness and efficiency, of an OPLT algorithm. We first prove that the OPLT algorithm satisfies each of the properties in two separate lemmas. The final proof of the theorem is then straightforward.

Lemma 1. OPLT is a proper OPLT algorithm.

Proof. We need to show that for any $\mathcal{S}$ and t the two of the following hold. Firstly, that the set L_{T_t} of leaves of tree T_t built by OPLT correspond to $\mathcal{L}_t$, the set of all labels observed in S_t. Secondly, that the set H_t of classifiers trained by OPLT is exactly the same as $H = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_t)$, i.e., the set of node classifiers trained incrementally by Algorithm 1 on $\mathcal{D} = \mathcal{S}_t$ and tree T_t given as input parameter. We will prove it by induction with the base case for $\mathcal{S}_0$ and the induction step for $\mathcal{S}_t$, $t \geq 1$, with the assumption that the statement holds for $\mathcal{S}_{t-1}$.

For the base case of $\mathcal{S}_0$, tree T_0 is initialized with the root node r_T with no label assigned and set H_0 of node classifiers with a single classifier assigned to the root. As there are no observations, this classifier receives no updates. Now, notice that IPLT.Train, run on T_0 and $\mathcal{S}_0$, returns exactly the same set of classifiers H that contains solely the initialized root node classifier without any updates (assuming that initialization procedure is always the same). There are no labels in any sequence of 0 observations and also T_0 has no label assigned.

The induction step is more involved as we need to take into account the internal loop, which extends the tree with new labels. Let us consider two cases. In the first one, observation $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ does not contain any new label. This means that that the tree T_{t-1} will not change, i.e., $T_{t-1} = T_t$. Moreover, node classifiers from H_{t-1} will get the same updates for $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ as classifiers in IPLT.Train, therefore $H_t = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_t)$. It also holds that $l_j \in L_{T_t}$ iff $j \in \mathcal{L}_t$, since $\mathcal{L}_{t-1} = \mathcal{L}_t$. In the second case, observation $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ has $m' = |\mathcal{L}_{\boldsymbol{x}_t} \setminus \mathcal{L}_{t-1}|$ new labels. Let us make the following assumption for the UpdateTree procedure, which we later prove that it indeed holds. Namely, we assume that the set $H_{t'}$ of classifiers after calling the UpdateTree procedure is the same as the one being returned by $\text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$, where T_t is the extended tree. Moreover, leaves of T_t correspond to all observed labels seen so far. If this is the case, the rest of the induction step is the same as in the first case. All updates to classifiers in $H_{t'}$ for $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ are the same as in IPLT.Train. Therefore $H_t = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_t)$.

Now, we need to show that the assumption for the UpdateTree procedure holds. To this end we also use induction, this time on the number m' of new labels. For the base case, we take $m' = 1$. The induction step is proved for $m' > 1$ with the assumption that the statement holds for $m' - 1$.

For $m' = 1$ we need consider two scenarios. In the first scenario, the new label is the first label in the sequence. This label will be then assigned to the root node r_T. So, the structure of the tree does not change, i.e., $T_{t-1} = T_t$. Furthermore, the set of classifiers also does not changed, since the root classifier has already been initialized. It might be negatively updated by previous observations. Therefore, we have $H_{t'} = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$. Furthermore, all observed labels are appropriately assigned to the leaves of T_t. In the second scenario, set $\mathcal{L}_{t-1}$ is not empty. We need to consider in this scenario the three variants of tree extension illustrated in Figure 1.

In the first variant, tree T_{t-1} is extended by one leaf node only without any additional ones. AddNode creates a new leaf node v'' with the new label assigned to the tree. After this operation the tree contains all labels from $\mathcal{S}_t$. The new leaf v'' is added as a child of the selected node v. This new node is initialized as $\hat{\eta}(v'') = \textsc{InverseClassifier}(\hat{\theta}(v))$. Recall that InverseClassifier creates a wrapper that inverts the behavior of the base classifier. It predicts $1 - \hat{\eta}$, where $\hat{\eta}$ is the prediction of the base classifier, and flips the updates, i.e., positive updates become negative and negative updates become positive. From the definition of the auxiliary classifier, we know that $\hat{\theta}(v)$ has been trained on all positives updates of $\hat{\eta}(v)$. So, $\hat{\eta}(v'')$ is initialized with a state as if it was updated negatively each time $\hat{\eta}(v)$ was updated positively in sequence S_{t-1}. Notice that in S_{t-1} there is no observation labeled with the new label. Therefore $\hat{\eta}(v'')$ is the same as if it was created and updated using IPLT.Train. There are no other operations on T_{t-1}, so we have that $H_{t'} = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$.

In the second variant, tree T_{t-1} is extended by internal node v' and leaf node v''. The internal node v' is added in InsertNode. It becomes a parent of all child nodes of the selected node v and the only child of this node. Thus, all leaves of the subtree of v does not change. Since v' is the root of this subtree, its classifier $\hat{\eta}(v')$ should be initialized as a copy of the auxiliary classifier $\hat{\theta}(v)$, which has accumulated all updates from and only from observations with labels assigned to the leaves of this subtree. The addition of the leaf node v'' can be analyzed as in the first variant. Since nothing else has changed in the tree and in the node classifiers, we have that $H_{t'} = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$. Moreover, the tree contains the new label, so the statement holds.

The third variant is similar to the second one. Tree T_{t-1} is extended by two leaf nodes v' and v'' being children of the selected node v. Insertion of leaf v' is similar to insertion of node v' in the second variant, with the difference that v does not have any children and its label has to be reassigned to v'. The new classifier in v' is initialized

as a copy of the auxiliary classifier $\hat{\theta}(v)$, which contains all updates from and only from observations with the label assigned previously to v. Insertion of v'' is exactly the same as in the second variant. From the above, we conclude that $H_{t'} = \text{IPLT.}\textsc{Train}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$ and that T_t contains all labels from T_{t-1} and the new label. In this way, we prove the base case.

The induction step is similar to the second scenario of the base case. The only difference is that we do not extent tree T_{t-1}, but an intermediate tree with $m' - 1$ new labels already added. Because of the induction hypothesis, the rest of the analysis of the three variants of tree extension is exactly the same. This ends the proof that the assumption for the inner loop holds. At the same time, it finalizes the entire proof. □

Lemma 2. OPLT is an efficient OPLT algorithm.

Proof. The OPLT maintains one additional classifier per each node in comparison to IPLT. Hence, for a single observation, there is at most one update more for each positive node. Furthermore, the time and space cost of the complete tree building policy is constant per a single label, if implemented with an array list. In this case, insertion of any new node can be made in amortized constant time, and the space required by the array list is linear in the number of nodes. Concluding the above, the time and space complexity of OPLT is in constant factor of C_t and C_s, the time and space complexity of IPLT respectively. This proves that OPLT is an efficient OPLT algorithm. □

Theorem 2. OPLT is a proper and efficient online PLT algorithm.

Proof. The theorem directly follows from Lemma 1 and Lemma 2. □

D Detailed results of OPLT and CMT on extreme multi-label classification tasks

In Table 4, complementary to Table 2, we report performance on propensity scored precision at $\{1, 3, 5\}$ (Jain et al., 2016) defined as:

$$\text{PSP@}k = \frac{1}{k} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} q_j [\![\tilde{y}_j = 1]\!] ,$$

where $q_j = 1 + C(N_j + B)^{-A}$ is inverse propensity of label j, where N_j is the number of data points annotated with label j in the observed ground truth dataset of size N, A, B are dataset specific parameters and $C = (\log N - 1)(B + 1)^A$. For Wiki10 and AmazonCat datasets we use $A = 0.55, B = 1.5$, for WikiLSHTC $A = 0.5, B = 0.4$, and for Amazon $A = 0.6, B = 2.6$ as recommended in (Jain et al., 2016). Since values of q_j are higher for infrequent labels, propensity scored precision at k promotes the accurate prediction on harder to predict tail labels. As in the case of the results in terms of standard precision at k, OPLT outperforms CMT, being slightly worse than its offline counterparts IPLT and Parabel, especially on the WikiLSHTC dataset.

Table 4: Mean propensity weighted precision at $\{1, 3, 5\}$ (%) of Parabel, IPLT, CMT, OPLT for XMLC tasks. Notation: PP@k – propensity weighted precision at k-position, R – Random policy, B – Best-greedy policy.

	AmazonCat-13K			Wiki10-31K			WikiLSHTC-320K			Amazon-670K		
Algorithm	PP@1	PP@3	PP@5	PP@1	PP@3	PP@5	PP@1	PP@3	PP@5	PP@1	PP@3	PP@5
Parabel	50.89	63.60	71.27	11.73	12.70	13.68	25.91	31.50	34.77	25.39	28.57	31.21
IPLT	49.97	63.13	70.77	12.26	13.75	14.80	24.16	29.09	32.29	25.25	28.85	32.12
CMT	47.48	54.25	56.03	9.68	9.66	9.67	-	-	-	-	-	-
OPLT_R	48.98	60.67	67.51	11.64	13.07	14.12	16.12	19.70	22.68	20.28	24.04	27.35
OPLT_B	49.30	61.55	68.64	11.59	13.19	14.26	20.69	24.87	28.14	23.41	27.26	30.49
OPLT-W	49.86	62.91	70.44	11.69	13.41	14.43	22.56	27.10	30.23	23.65	27.62	30.95

In Table 5, we show detailed results of the empirical study we performed to evaluate OPLTs comprehensively. In addition to precision at $\{1, 3, 5\}$ and train times, we also report test times. We present the results for every algorithm that is using an incremental learning algorithm, updating the tree nodes (IPLT, CMT, and OPLT)

after 1 and 3 passes over the training dataset. We also present the results of OPLT-W with a warm-start sample of size 5, 10, and 15% of the training dataset (warm-up dataset). In all cases, the initial tree was created using hierarchical 2-means clustering on the warm-up sample and later extended using BEST-GREEDY policy. The results show that IPLT and OPLT achieve good results after just one pass over datasets. Prediction times of OPLT are slightly worse than IPLT, probably due to the worst tree structure, the prediction times of OPLT with warm-start are better than OPLT build from scratch. The reported prediction times of PARABEL are given for the batch prediction. The prediction times seem to be faster, but PARABEL needs to decompress the model during prediction, which makes it less suitable for online prediction. It is only efficient when the batches are sufficiently large.

Table 5: Mean precision at $\{1, 3, 5\}$ (%), train and test CPU time (averaged over 5 runs) of PARABEL, IPLT, CMT, OPLT for XMLC tasks. Notation: P@k – precision at k-position, T – CPU time, R – RANDOM policy, B – BEST-GREEDY policy, w – percentage of the training dataset sampled for warm-start, p – number of passes over train dataset, $*$ – offline prediction, depends on the test dataset size. Highlighted values are presented in the main paper in Table 2.

	AmazonCat					Wiki10				
Algorithm	P@1	P@3	P@5	t_{train}	t_{test}/N	P@1	P@3	P@5	t_{train}	t_{test}/N
PARABEL	92.58	78.53	63.90	11.51m	*0.3ms	84.17	72.12	63.30	4.00m	*0.9ms
IPLT$_{p=1}$	93.11	78.72	63.98	34.2m	0.6ms	84.87	74.42	65.31	18.3m	10.1ms
IPLT$_{p=3}$	93.19	78.64	63.92	115.7m	0.6ms	84.37	73.90	64.70	46.8m	10.4ms
CMT$_{p=1}$	87.51	69.49	53.99	45.8m	1.1ms	80.59	64.17	55.05	35.1m	16.5ms
CMT$_{p=3}$	89.43	70.49	54.23	168.3m	1.5ms	79.86	64.72	55.25	120.9m	20.4ms
OPLT$_{R,p=1}$	92.66	77.44	62.52	99.5m	1.7ms	84.34	73.73	64.41	30.3m	14.1ms
OPLT$_{R,p=3}$	92.65	77.43	62.59	278.3m	1.4ms	83.09	72.25	63.33	86.5m	18.2ms
OPLT$_{B,p=1}$	92.74	77.74	62.91	84.1m	1.5ms	84.47	73.73	64.39	27.7m	16.4ms
OPLT$_{B,p=3}$	92.77	77.70	62.93	251.9m	1.4ms	83.39	72.50	63.48	86.7m	18.2ms
OPLT-W$_{w=5\%,p=1}$	93.14	78.67	63.85	40.6m	0.8ms	85.10	74.44	64.77	28.7m	16.6ms
OPLT-W$_{w=5\%,p=3}$	93.11	78.50	63.75	134.3m	0.8ms	83.93	73.34	63.75	75.9m	17.0ms
OPLT-W$_{w=10\%,p=1}$	93.14	78.68	63.92	43.7m	0.9ms	85.22	74.68	64.93	28.2m	17.8ms
OPLT-W$_{w=10\%,p=3}$	93.10	78.51	63.79	133.9m	0.8ms	84.75	74.02	64.33	83.9m	18.3ms
OPLT-W$_{w=15\%,p=1}$	93.17	78.75	63.95	50.1m	1.0ms	85.42	74.50	64.94	28.2m	17.5ms
OPLT-W$_{w=15\%,p=3}$	93.15	78.54	63.82	121.6m	0.8ms	84.69	73.90	64.29	79.1m	19.5ms
	WikiLSHTC					Amazon				
Algorithm	P@1	P@3	P@5	t_{train}	t_{test}/N	P@1	P@3	P@5	t_{train}	t_{test}/N
PARABEL	62.78	41.22	30.27	15.4m	*0.3ms	43.13	37.94	34.00	7.6m	*0.3ms
IPLT$_{p=1}$	58.14	37.94	28.14	69.0m	2.1ms	40.78	35.88	32.28	33.2m	5.1ms
IPLT$_{p=3}$	60.80	39.58	29.24	175.1m	2.6ms	43.55	38.69	35.20	79.4m	6.2ms
OPLT$_{R,p=1}$	46.36	29.85	22.53	103.0m	6.4ms	36.27	32.13	29.01	46.1m	17.0ms
OPLT$_{R,p=3}$	47.76	30.97	23.37	330.1m	7.9ms	38.42	34.33	31.32	134.2m	18.2ms
OPLT$_{B,p=1}$	53.32	34.22	25.52	88.6m	4.8ms	38.42	34.03	30.73	36.7m	10.0ms
OPLT$_{B,p=3}$	54.69	35.32	26.31	300.0m	5.8ms	41.09	36.65	33.42	111.9m	13.0ms
OPLT-W$_{w=5\%,p=1}$	57.46	36.95	27.35	82.5m	3.6ms	39.45	34.85	31.44	35.3m	9.1ms
OPLT-W$_{w=5\%,p=3}$	58.51	37.92	28.04	249.8m	4.6ms	41.96	37.39	34.07	100.0m	11.1ms
OPLT-W$_{w=10\%,p=1}$	58.20	37.50	27.73	71.9m	3.0ms	39.71	35.03	31.57	36.6m	9.1ms
OPLT-W$_{w=10\%,p=3}$	59.23	38.39	28.38	205.7m	3.4ms	42.21	37.60	34.25	98.3m	11.3ms
OPLT-W$_{w=15\%,p=1}$	58.68	37.85	27.97	72.1m	2.8ms	40.11	35.35	31.84	34.0m	8.3ms
OPLT-W$_{w=15\%,p=3}$	59.66	38.67	28.57	196.0m	3.1ms	42.41	37.70	34.29	85.1m	9.1ms

E Detailed results of OPLT and CMT on few-shot multi-class classification tasks

In Table 6 we present additional results for experiments on few-shot multi-class classification. We report results on the test set after 1 and 3 passes. In addition to accuracy and train times, we also report test times after 1 and 3 passes over the training dataset.

Table 6: Mean accuracy of prediction (%) and train and test CPU time of CMT, OPLT for few-shot multi-class classification tasks. Notation: Acc – accuracy, T – CPU time, N – number of samples in test set, R – RANDOM policy, B – BEST-GREEDY policy, p – number of passes over train dataset. Highlighted values are presented in the main paper in Table 3.

	ALOI			Wikipara 1-shot			Wikipara 3-shot			Wikipara 5-shot		
Algorithm	Acc.	t_{train}	t_{test}/N	Acc.	t_{train}	t_{test}/N	Acc.	t_{train}	t_{test}/N	Acc.	t_{train}	t_{test}/N
$\text{CMT}_{p=1}$	17.63	37.8s	0.78ms	2.22	7.1s	0.51ms	3.22	20.9s	0.76ms	4.15	81.1s	3.22ms
$\text{OPLT}_{R,p=1}$	62.58	6.0s	0.12ms	1.51	2.6s	0.82ms	13.31	6.7s	1.83ms	26.06	11.6s	2.61ms
$\text{OPLT}_{B,p=1}$	63.91	6.1s	0.12ms	0.80	2.2s	0.55ms	11.40	6.4s	1.67ms	23.12	10.3s	2.48ms
$\text{CMT}_{p=3}$	71.98	207s	0.57ms	2.48	21.3s	0.37ms	3.28	63.1s	0.49ms	4.21	240.9s	1.22ms
$\text{OPLT}_{R,p=3}$	66.50	29.3s	0.11ms	8.99	4.6s	1.14ms	27.34	16.4s	2.64ms	40.67	27.6s	2.83ms
$\text{OPLT}_{B,p=3}$	67.26	18.1s	0.10ms	3.31	4.4s	0.87ms	24.66	15.6s	2.37ms	39.13	27.2s	2.54ms

F Performance of OPLT with BEST-GREEDY policy and different α values

In Table 7, OPLT with BEST-GREEDY policy and different values of tree balancing parameter α is compared to IPLT and OPLT with RANDOM policy. All other parameters of the model were set as described in Section 5.1. It shows that for $\alpha \geq 0.75$, OPLT achieves the tree depth close or the same as perfectly balanced tree build for IPLT, at the same time having the best predictive performance and the shortest training and prediction time among OPLT variants.

G Performance analysis of OPLT with warm-start

In Table 8, we compare OPLT with warm-start (OPLT-W) with two variants of IPLT. In the first variant, IPLT is trained only on the warm-up training dataset, and in the second variant, IPLT uses a tree created on warm-up, but then updated with all examples from the training set but without updating the tree structure. In both variants, IPLT cannot predict labels that are not present in the initial warm-up dataset. All other parameters of the model were set as described in Section 5.1. This experiment shows the significant gain in predictive performance on WikiLSHTC and Amazon datasets by extending a tree with newly observed labels over the IPLT variants that do not take new labels into account.

Table 7: Mean precision at $\{1, 3, 5\}$ (%), train time, test time and tree depth (averaged over 5 runs) of OPLT with BEST-GREEDY policy with different α values compared with IPLT and OPLT with RANDOM policy. Notation: P@k – precision at k-position, t – CPU time, d(T) – tree depth, R – RANDOM policy, B – BEST-GREEDY policy, α – tree balancing parameter, percentage of the training dataset sampled for warm-start, p – number of passes over train dataset.

	AmazonCat						Wiki10					
Algorithm	P@1	P@3	P@5	t_{train}	t_{test}/N	d(T)	P@1	P@3	P@5	t_{train}	t_{test}/N	d(T)
IPLT$_{p=3}$	93.10	78.52	63.81	115.7m	0.6ms	10.0	83.49	73.06	64.12	46.8m	10.4ms	11.0
OPLT$_{R,p=3}$	92.65	77.43	62.59	278.3m	1.4ms	10.0	83.09	72.25	63.33	86.5m	18.2ms	11.0
OPLT$_{B,\alpha=0.25,p=3}$	92.77	77.77	62.99	303.1m	1.5ms	30.0	82.93	72.41	63.38	88.2m	20.0ms	24.6
OPLT$_{B,\alpha=0.375,p=3}$	92.80	77.72	62.93	250.2m	1.5ms	18.0	83.10	72.43	63.37	84.6m	18.9ms	17.2
OPLT$_{B,\alpha=0.5,p=3}$	92.79	77.74	62.95	257.6m	1.4ms	14.0	83.09	72.48	63.45	86.0m	18.7ms	14.0
OPLT$_{B,\alpha=0.625,p=3}$	92.77	77.74	62.96	245.2m	1.4ms	12.0	83.25	72.50	63.48	87.3m	19.6ms	12.6
OPLT$_{B,\alpha=0.75,p=3}$	92.77	77.70	62.93	248.9m	1.4ms	11.0	83.39	72.50	63.48	86.7m	18.2ms	12.0
OPLT$_{B,\alpha=0.875,p=3}$	92.73	77.64	62.85	207.7m	1.2ms	10.0	83.41	72.50	63.44	82.2m	17.6ms	11.0
	WikiLSHTC						Amazon					
Algorithm	P@1	P@3	P@5	t_{train}	t_{test}/N	d(T)	P@1	P@3	P@5	t_{train}	t_{test}/N	d(T)
IPLT$_{p=3}$	60.80	39.58	29.24	175.1m	2.6ms	14.0	43.55	38.69	35.20	79.4m	6.2ms	15.0
OPLT$_{R,p=3}$	47.76	30.97	23.37	330.1m	7.9ms	14.8	38.42	34.33	31.32	134.2m	18.2ms	16.0
OPLT$_{B,\alpha=0.25,p=3}$	53.14	34.36	25.69	337.8m	6.3ms	68.4	38.83	34.72	31.69	139.2m	16.4ms	38.4
OPLT$_{B,\alpha=0.375,p=3}$	53.47	34.57	25.81	309.1m	6.9ms	30.8	39.17	35.04	31.99	126.6m	16.2ms	21.2
OPLT$_{B,\alpha=0.5,p=3}$	53.77	34.74	25.93	294.3m	6.0ms	21.8	39.47	35.26	32.17	123.5m	14.1ms	17.8
OPLT$_{B,\alpha=0.625,p=3}$	54.01	34.90	26.04	298.7m	6.1ms	18.0	40.06	35.79	32.65	118.9m	14.0ms	16.0
OPLT$_{B,\alpha=0.75,p=3}$	54.69	35.32	26.31	297.0m	5.8ms	16.0	41.09	36.65	33.42	111.9m	13.0ms	16.0
OPLT$_{B,\alpha=0.875,p=3}$	54.56	35.22	26.25	285.4m	5.8ms	15.0	41.18	36.75	33.53	110.1m	12.7ms	15.0

Table 8: Mean precision at $\{1, 3, 5\}$ (%, averaged over 5 runs) of IPLT trained only on warm-start training dataset and IPLT with a tree created on warm-start training dataset (IPLT-U) and updated with all examples without updating the tree and OPLT with warm-start (OPLT-W). Notation: P@k – precision at k-position, w – percentage of the training dataset sampled for warm-start, p – number of passes over train dataset.

	AmazonCat			Wiki10			WikiLSHTC			Amazon		
Algorithm	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
IPLT$_{w=5\%,p=3}$	88.49	70.48	55.49	80.69	61.68	51.85	38.95	22.74	16.37	10.98	9.55	8.77
IPLT-U$_{w=5\%,p=3}$	93.15	78.54	63.62	83.95	72.99	63.20	54.70	34.54	25.27	29.54	22.65	18.22
OPLT-W$_{w=5\%,p=3}$	93.11	78.50	63.75	83.93	73.34	63.75	58.51	37.92	28.04	41.96	37.39	34.07
IPLT$_{w=10\%,p=3}$	89.92	72.97	57.99	80.79	65.46	55.38	44.59	26.71	19.35	15.77	13.70	12.45
IPLT-U$_{w=10\%,p=3}$	93.12	78.58	63.79	84.38	73.52	63.88	57.47	36.81	27.05	35.41	28.84	24.06
OPLT-W$_{w=10\%,p=3}$	93.10	78.51	63.79	84.75	74.02	64.33	59.23	38.39	28.38	42.21	37.60	34.25
IPLT$_{w=15\%,p=3}$	90.69	74.17	59.25	81.61	67.28	57.18	47.63	29.03	21.11	19.38	16.79	15.24
IPLT-U$_{w=15\%,p=3}$	93.13	78.62	63.85	84.65	73.93	64.24	58.55	37.71	27.77	37.90	31.87	27.19
OPLT-W$_{w=15\%,p=3}$	93.15	78.54	63.82	84.69	73.90	64.29	59.66	38.67	28.57	42.41	37.70	34.29