

Consistent, Balanced, and Overlapping Label Trees for Extreme Multi-label Learning

Zhiqi Ge*
Yuanyuan Guan*
zhiqige2000@gmail.com
guanyy16@gmail.com
College of Computer Science and Technology, Jilin University
China

Ximing Li†‡
College of Computer Science and Technology, Jilin University
China
liximing86@gmail.com

Bo Fu
School of Computer and Information Technology, Liaoning Normal University
China
fubo@lnnu.edu.cn

ABSTRACT

The emerging eXtreme Multi-label Learning (XML) aims to induce multi-label predictive models from big datasets with extremely large numbers of instances, features, and especially labels. To meet the great efficiency challenge of XML, one flexible solution is the methodology of label tree, which, as its name suggests, is technically defined as a tree hierarchy of label subsets, partitioning the original large-scale XML problem into a number of small-scale sub-problems (*i.e.,* denoted by leaf nodes) and then reducing the complexity to logarithmic time. Notably, the expected label trees should accurately find the right leaf nodes for future instances (*i.e.,* effectiveness) and generate balanced leaf nodes (*i.e.,* efficiency). To achieve this, we propose a novel generic method of label tree, namely Consistent, Balanced, and Overlapping Label Tree (CBOLT). To enhance the precision, we employ the weighted clustering to partition non-leaf nodes and allow overlapping label subsets, enabling to alleviate the inconsistent path and disjoint label subset issues. To improve the efficiency, we propose a new concept of balanced problem scale and implement it with a balanced regularization for non-leaf nodes partition. We conduct extensive experiments on several benchmark XML datasets. Empirical results demonstrate that CBOLT is superior to the existing methods of label trees, and it can be applied to existing XML methods and achieve competitive performance with strong baselines.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**; *Classification and regression trees.*

*Both authors contributed equally to this research
†Corresponding author
‡Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, China

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '22, October 17–21, 2022, Atlanta, GA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557261

KEYWORDS

extreme multi-label learning, label tree, consistent path, balanced regularization

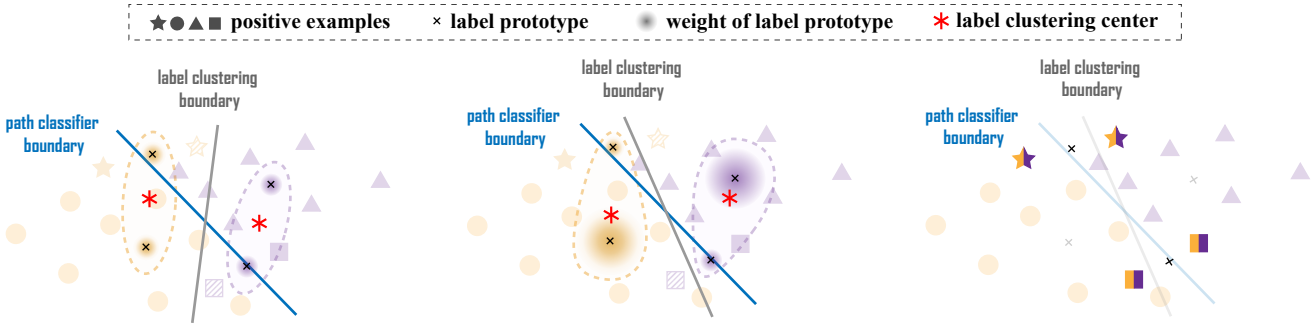
ACM Reference Format:
Zhiqi Ge, Yuanyuan Guan, Ximing Li, and Bo Fu. 2022. Consistent, Balanced, and Overlapping Label Trees for Extreme Multi-label Learning. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557261

1 INTRODUCTION

In modern machine learning applications, *e.g.,* advertising, searching, and recommendation, the datasets can be extremely huge even with millions of instances, features, and especially labels [1, 30, 32]. To handle such kind of big datasets, more attention has been paid to the emerging e**X**treme **M**ulti-label **L**earning (**XML**). Its goal is to efficiently train multi-label predictive models over big datasets, and predict each future instance its relevant subset of labels from the huge label set with limited computational resource [19, 21, 40, 42].

To meet the great efficiency challenges of XML, many attempts have been made as surveyed in [28], where one key idea is the **problem partition**, *i.e.,* dividing the original large-scale classification problem into a set of small-scale sub-problems and then solving for them. To our knowledge, the most prevalent way of problem partition is to generate a tree hierarchy of labels, called **label tree**, where each non-root node represents a label subset. Specifically, each leaf node defines a sub-problem with its corresponding label subset, and it trains a sub-model, called **base classifier**. Given any future instance, it traverses the learned label tree to select a few **candidate leaf nodes** for final prediction [9, 26, 31, 36, 43]. The label tree hierarchy can reduce the computational complexity to logarithmic time for training and prediction simultaneously. Nowadays, generating label trees has accordingly become a generic step in many most recent XML arts [10, 24, 41].

Formally, the label tree is generated by recursively performing the following two steps until the termination condition is reached. (1) **New label subset generation:** For each non-leaf node, the labels within the current subset are represented by label prototypes. A conventional clustering method such as k-means is applied to divide the current subset into several new subsets, *i.e.,* new nodes. (2) **Path classifier induction.** For each non-leaf node, each of its child nodes is treated as a meta-label, and its positive instances are defined as the ones whose relevant labels are within its label subset. A classifier, dubbed as **path classifier**, is then trained over those



(a) Problems (b) Consistent (c) Overlapping

Figure 1: A toy example of a non-leaf node with 4 labels (a) First, two label subsets are generated by clustering with label prototypes, however, the label clustering boundary is inconsistent with the boundary of the path classifier trained by the corresponding positive instances. Second, in light of the path classifier boundary, some marginal labels, especially infrequent ones, tend to belong to both label subsets, however, they have to be one side under the disjoint setting. (b) To solve the inconsistent path problem, we suggest a weighted k-means clustering method to divide the current label subset, where each label prototype is weighed by its positive instance number. (c) We compute the predictive scores of instances with the path classifiers, and allow the labels with similar predictive scores for both subsets (*i.e.,* the ones with bi-color instances) to belong to both subsets.

meta-labels. The path classifiers are used to traverse the label tree for future instances during prediction.

Unfortunately, the aforementioned label tree methodology suffers from two issues. (1) **Inconsistent path problem:** The label subsets are generated with label prototypes while the path classifiers are trained with the corresponding positive instances. This results in an inconsistency between label subsets and path classifiers, especially for infrequent labels. (2) **Disjoint label subset problem:** For non-leaf nodes, the hard clustering methods are often applied to generate new child nodes, whose label subsets are *disjoint*. When traversing the tree during prediction, such disjoint partition tends to miss the relevant marginal labels whose positive instances are close to the boundary, also especially for infrequent labels. To clearly show them, we illustrate toy examples in Fig.1(a).

To remedy the aforementioned problems, in this paper we propose a novel generic method of label tree for XML tasks, namely **C**onsistent, **B**alanced, and **O**verlapping **L**able **T**ree (**CBOLT**). *First*, to solve the inconsistent path problem, during new label subset generation we suggest a weighted k-means clustering method to divide the current label subset, where each label prototype is weighed by its positive instance number. The intuition is that we generate new label subsets by further incorporating the numbers of positive instances that are captured for path classifier induction, so as to make the label subsets and path classifiers relatively consistent. *Second*, to solve the disjoint label subset problem, we suggest an overlapping labels selection step. We feed the instances into the path classifier to compute the total predictive scores of labels, and then arrange a number of marginal labels, *i.e.,* the labels whose total predictive scores of multiple label subsets are close to each other, to belong to different label subsets, leading to several overlapping ones. For clarity, we illustrate a toy example in Fig.1(b)(c). Additionally, to improve the efficiency, we consider that the sub-problems of leaf nodes should be balanced. Beyond the prior art that considers the balance with respect to labels [31], we propose a new concept of **balanced problem scale**, taking the numbers of labels and instances into account. This can be achieved by incorporating a balanced regularization into the weighted k-means clustering for the new label subset generation.

We have conducted extensive experiments to thoroughly verify CBOLT on benchmark XML datasets. Empirical results demonstrate that CBOLT is superior to the existing label tree methods, and it can be more efficient with balanced problem scales. Additionally, ablative studies have proven the positive impact of each idea of CBOLT,

In a nutshell, the main contributions of this paper are outlined below.

- We propose a novel generic label tree method of XML named CBOLT, which alleviates the existing inconsistent path and disjoint label subset problems.
- To improve the efficiency, we propose a new concept of balanced problem scale achieved by incorporating a balanced regularization into the new label subset generation.
- Empirical results of specific experiments indicate the superior performance of CBOLT compared against the existing label tree methods.

2 RELATED WORK

As surveyed in the literature [28], a well-recognized taxonomy of XML divides existing methods into four types: embedding-based, tree-based, One-vs-All (OvA), and deep learning-based methods. However, many recent XML techniques adopt various mixture tricks and actually belong to multiple XML types. In this section, we briefly review several related XML methods with label tree and embedding learning.

Table 1: Summary of notations.

Notation	Meaning
n	number of instances
d	number of features
l	number of labels
$\Omega = \{(\mathbf{x}_g, \mathbf{y}_g)\}_{g=1}^{g=n}$	training dataset of XML
$\mathbf{x} \in \mathbb{R}^d$	feature vector of instance
$\mathbf{y} \in \{0,1\}^l$	label indicator of instance
h	depth of label tree
$\mathcal{T} = \{(\Pi^{i\|j}, C^{i\|j})\}_{i=0,j=1}^{i=h,j=k^i}$	k-ary label tree
$\Pi^{i\|j}$	label subset of the j-th node at the i-th level
$C^{i\|j}$	path classifier of the j-th node at the i-th level
$\mathcal{P}^{i\|j} = \{\mathbf{p}_s\}_{s\in\Pi^{i\|j}}$	label prototype set of the j-th node at the i-th level
w_s	number of positive instances of the s-th label
$S^{i\|j}$	problem scale of the j-th node at the i-th level

2.1 XML Methods with Label Tree

The label tree is technically defined as a tree hierarchy of label subsets, partitioning the original large-scale XML problem into a number of small-scale sub-problems. It has been widely applied to embedding-based [5, 37], OvA [26, 31], and deep learning-based methods [9, 43]. Commonly, the k-means clustering family is often employed to recursively decompose label subsets, and several tricks such as balanced constraint [31] and compression operation [43] are used to achieve balanced or shallower tree structures. Besides the tree structures, another central problem is the way to train path classifiers for non-leaf nodes, and base classifiers for leaf nodes. For example, the path and base classifiers of SLEEC [5], Parabel [31], and AttentionXML [43] are kNNs, OvA with SVMs, and attention-based neural networks, respectively. Although various attempts of label trees have been made, they still suffer from inconsistent path and disjoint label subset problems. In contrast, our Cbolt can alleviate both problems by retraining and overlapping techniques, so as to enhance the accuracy of leaf recall and further the final performance when predicting future texts.

2.2 XML Methods with Embedding Learning

In the context of XML, the target of embedding learning is to learn lower dimensional and more discriminative features for efficient training. Several early attempts mainly concentrate on linear feature projection [5, 20, 23, 37]. For example, SLEEC [5] and AnnexML [37] train feature projection matrix by preserving the text similarity matrix computed by label co-occurrences. To some extent, the deep learning-based XML methods can be interpreted as nonlinear feature projection by applying the well-established architectures of CNN, LSTM, and Transformer [9, 11, 13, 24, 27, 40, 42–44].

Besides the aforementioned XML arts, another interesting spirit of embedding learning is to induce lower-dimensional features by Random Projection (RP) techniques, which have been extensively used in clustering [7, 18], classification [6, 8, 34], and numerical approximation [12, 15, 33]. The essential idea of RP is based on the Johnson-Lindenstrauss lemma [14, 25], which clarifies that in certain circumstances, pairwise distances are well preserved after RP transformations. Later researches have studied the preservation of other data characteristics including margin separability [29, 35] and generalized error bounds of certain classifiers [8, 16, 17]. Built on those theoretical supports, the CRAFTML [36] efficiently builds trees by RP features, and the work of [38] directly handles XML datasets by ensemble kNN classifiers with RP features.

3 THE PROPOSED CBOLT APPROACH

In this section, we introduce the proposed label tree method for XML tasks named **Cbolt**. We describe the generative process details of Cbolt and briefly show how to find the candidate leaf nodes for future instances with Cbolt. Finally, we discuss the time complexity.

To clearly describe Cbolt, we first formulate XML by the following notations. Consider a dataset $\Omega = \{(\mathbf{x}_g, \mathbf{y}_g)\}_{g=1}^{g=n}$, where, for each instance, $\mathbf{x}_g \in \mathbb{R}^d$ and $\mathbf{y}_g \in \{0,1\}^l$ denote its feature and indicator vector of relevant labels, respectively. Let n, d, and l denote the numbers of instances, features, and labels, where in the context of XML they are all extremely large (*e.g.*, $> 10^7$). The task aims at efficiently inducing a multi-label predictive model from Ω to predict future instances. For clarity, we outline all important notations in Table 1.

3.1 Generative Process of Cbolt

Given Ω, the proposed Cbolt method concentrates on building a k-ary label tree $\mathcal{T} = \{(\Pi^{i|j}, C^{i|j})\}_{i=0,j=1}^{i=h,j=k^i}$, where h is the depth of the tree, and $\Pi^{i|j}$ denotes the label subset of the j-th node at the i-th level, and $C^{i|j}$ is the corresponding path classifier set.[1] Formally, Cbolt recursively generates new nodes by performing 3 key steps, *i.e.,* **N**ew label subsets generation (**N-Step**), **P**ath classifier induction (**P-Step**), and **O**verlapping labels selection (**O-Step**), until a predefined tree depth is reached. We now describe the details of those steps.

[1] Specially, we note the leaf nodes have no path classifiers. We overlook this to make the notations simple.

[N-Step]. Given the label subset $\Pi^{i|j}$ of any non-leaf node (*i.e.,* $i < h$), we divide it into k new label subsets by applying the clustering techniques with the prototypes of labels within $\Pi^{i|j}$. Each label prototype is the normalized average vector of its positive instances, described below:

$$\mathbf{p}_s = \frac{\sum_{g=1}^{n} y_{gs}\mathbf{x}_g}{\| \sum_{g=1}^{n} y_{gs}\mathbf{x}_g \|}, \qquad s \in \Pi^{i|j}, \tag{1}$$

where $\| \cdot \|$ denotes the ℓ_2 norm, and s is a label within $\Pi^{i|j}$.

Commonly, the prior arts [24, 26] divide $\Pi^{i|j}$ into k new label subsets by clustering $\mathcal{P}^{i|j} = \{\mathbf{p}_s\}_{s\in\Pi^{i|j}}$ with the conventional k-means method. However, this may result in an inconsistency between its child label subsets and its path classifier (*i.e.,* generated in the next P-Step), which are actually trained with the corresponding positive instances, instead of $\mathcal{P}^{i|j}$. To remedy this, we propose to divide $\Pi^{i|j}$ by applying the following weighted clustering objective:

$$\min_{\boldsymbol{\mu},\mathbf{F}} \; - \sum_{s\in\Pi^{i|j}} \sum_{u=1}^{k} F_{su} w_s \mathbf{p}_s^\top \boldsymbol{\mu}_u, \tag{2}$$

where $\boldsymbol{\mu}$ denotes the set of clustering centers; $F_{su} \in \{0, 1\}$ denotes the cluster assignment indicator of the s-th label to the u-th cluster; and most importantly w_s is the number of positive instances of the s-th label. Our intuition is that we divide $\Pi^{i|j}$ by considering the numbers of positive instances, enabling to make its child label subsets and path classifier (trained with positive instances) relatively consistent.

To further improve the efficiency, we consider the label subsets at the same level should be balanced. The prior art [31] has considered the case of balanced labels, where it constrains each new label subset has to include the same number of labels in the N-Step. However, this may be problematic since in XML there are many infrequent labels containing very few positive instances. Therefore, even the label subsets contain the same number of labels, they may contain quite different numbers of instances. To alleviate this, we propose a new concept of balanced problem scale, taking both numbers of labels and instances into account. For a label subset $\Pi^{i|j}$, its problem scale is technically defined as $S^{i|j} = |\Pi^{i|j}| \sum_{s\in\Pi^{i|j}} w_s$. Specifically, we implement the balanced problem scale by incorporating a balanced regularization to the weighted clustering objective as follows:

$$\min_{\boldsymbol{\mu},\mathbf{F}} \; - \sum_{s\in\Pi^{i|j}} \sum_{u=1}^{k} F_{su} w_s \mathbf{p}_s^\top \boldsymbol{\mu}_u + \beta \sum_{u=1}^{k} S^{i+1|jk+u}, \tag{3}$$

where β is the balanced coefficient used to control the balanced degree of new label subsets. By solving Eq.3, we can directly achieve consistent and balanced new label subsets in this N-Step. Since it is intractable to optimize, we resort to a heuristic iterative approach, which empirically converges fast in our early experiments. We briefly introduce its update rules each loop. (I) By fixing the cluster assignments $\mathbf{F}$, the clustering centers $\boldsymbol{\mu}$ can be updated as follows:

$$\boldsymbol{\mu}_u = \frac{\sum_{s\in\Pi^{i|j}} F_{su} w_s \mathbf{p}_s}{\| \sum_{s\in\Pi^{i|j}} F_{su} w_s \mathbf{p}_s \|}, \quad u \in 1, \cdots, k \tag{4}$$

(II) By fixing $\boldsymbol{\mu}$, we update $\mathbf{F}$ with a greedy update process. We consider each label one by one in the descending order of the values of $w_s \mathbf{p}_s^\top \boldsymbol{\mu}_{\hat{u}_s}$, where

$$\hat{u}_s = \underset{u\in 1,\cdots,k}{\operatorname{argmax}} \; w_s \mathbf{p}_s^\top \boldsymbol{\mu}_u, \quad s \in \Pi^{i|j} \tag{5}$$

For each label, we assign it to the $\widetilde{u}_s$-th cluster, *i.e.,* $F_{s\widetilde{u}_s} = 1$, by using the following formula:

$$\widetilde{u}_s = \underset{u\in 1,\cdots,k}{\operatorname{argmax}} \; w_s \mathbf{p}_s^\top \boldsymbol{\mu}_u - \beta \frac{S^{i+1|jk+u}}{S^{i|j}}, \quad s \in \Pi^{i|j}, \tag{6}$$

We then update the problem scale $S^{i+1|jk+\widetilde{u}_s}$. For clarity, we summarize the update process in **Algorithm 1**.

[P-Step]. In this step, we train the path classifier $C^{i|j}$ by treating the obtained label subsets $\{\Pi^{i+1|jk+u}\}_{u=1}^{u=k}$ as k meta-labels denoted by $\{\widehat{\mathbf{y}}_u^{i|j}\}_{u=1}^{u=k}$. For each meta-label $\widehat{\mathbf{y}}_u^{i|j}$, its positive instances are defined as the instances whose relevant labels are within the corresponding subset, *i.e.,* $\mathcal{X}_u^{i|j} = \{\mathbf{x}_g | y_{gs} = 1 \;\&\; s \in \Pi^{i+1|jk+u}, g \in [n]\}$, Accordingly, this leads to a pseudo-training dataset $\Omega^{i|j} = \{(\mathcal{X}_u^{i|j}, \widehat{\mathbf{y}}_u^{i|j})\}_{u=1}^{u=k}$. We train the path classifier $C^{i|j}$ over $\Omega^{i|j}$.

[O-Step]. Another problem is that there are many marginal labels, especially infrequent labels, whose predictive scores of multiple meta-labels tend to be similar. To protect such labels, we allow them to belong to multiple label subsets, leading to overlapping subsets. To be specific, we feed the instances[2] $\{\mathcal{X}_u^{i|j}\}_{u=1}^{u=k}$ into the trained path classifier $C^{i|j}$. For each label within $\Pi^{i|j}$, we compute its predictive scores for meta-labels as follows:

$$\lambda_u^{i|j}(s) = \sum_{\mathbf{x}_g \in \{\mathcal{X}_u^{i|j}\}_{u=1}^{u=k}} y_{gs} C^{i|j}(\mathbf{x}_g)_u, \quad s \in \Pi^{i|j}, \tag{7}$$

where $C^{i|j}(\mathbf{x}_g)_u$ denotes the predictive score of $\mathbf{x}_g$ belonging to the meta-label $\widehat{\mathbf{y}}_u$. Its meta-label with top-1 predictive score is denoted as follows:

$$u_s^* = \underset{u\in 1,\cdots,k}{\operatorname{argmax}} \; \lambda_u^{i|j}(s), \quad s \in \Pi^{i|j} \tag{8}$$

Given those predictive results, we allow that the label can be also assigned to the label subsets as follows:

$$s \to \Pi^{i+1|jk+u}, \quad \text{if } \lambda_u^{i|j}(s) > \rho \lambda_{u_s^*}^{i|j}(s), \quad s \in \Pi^{i|j}, \tag{9}$$

where ρ denotes the overlapping coefficient. To avoid produce too big label subsets, we define an overlapping upper bound parameter $\gamma \in [0, 1]$. Specifically, for each label subset $\Pi^{i+1|jk+u}$, we ingest overlapping labels satisfying Eq.9 in descending order of the predictive scores $\lambda_u^{i|j}(s)$, and constrain its problem scale can not exceed $(1+\gamma)S^{i+1|jk+u}$.

For clarity, we summarize the full generative process of Cbolt in **Algorithm 2**.

3.2 Training Base Classifiers for Leaf Nodes

Each leaf node can be treated as a small-scale sub-problem with the label subset $\Pi^{h|u}$. Accordingly, we can learn the base classifier with any traditional multi-label learning methods. Taking the OvA method as an example, we train a binary classifier for each label

[2]We specially note that some of positive instances may belong to multiple meta-labels. They will be predicted once in this step.

Table 2: Summary of datasets. We refer to n_{tr}, n_{tst}, d, and l as the numbers of training instances, test instances, features, and labels, respectively; $avg.l$ and $avg.n$ as average labels per-instance and average instances per-label, respectively; l_{tail} as the number of labels appearing at most 10 times.

Dataset	n_{tr}	n_{tst}	d	l	$avg.l$	$avg.n$	$l_{tail}(\%)$	data size
EURLex-4K	15,539	3,809	5,000	3,993	5.31	25.73	2,685 (67.2)	large-scale
Wiki10-31K	14,146	6,616	101,938	30,938	18.64	8.52	27,853 (90.0)	large-scale
AmazonCat-13K	1,186,239	306,792	203,882	13,330	5.04	448.58	4,629 (34.7)	extreme-scale
WikiLSHTC-325K	1,778,351	587,084	1,617,899	325,056	3.19	17.46	230,790 (71.0)	extreme-scale
Amazon-670K	490,449	153,025	135,909	670,091	5.45	3.99	640,153 (95.5)	extreme-scale

Algorithm 1 Greedy optimization for the objective of Eq.3

Input: The label subset $\Pi^{i|j}$
Initialize the assignments $\mathbf{F}$ randomly, and then initialize the clustering centers $\boldsymbol{\mu}$ with $\mathbf{F}$
Repeat
  Fixing $\mathbf{F}$, update $\boldsymbol{\mu}$ using Eq.4
  Fixing $\boldsymbol{\mu}$, update $\mathbf{F}$ with the following greedy loop
  For each label within $\Pi^{i|j}$
    Pick one label in the descending order of $w_s \mathbf{p}_s^\top \boldsymbol{\mu}_{\hat{u}_s}$
    Assign it to a cluster using Eq.6
    Update the problem scale
  End For
Until convergence
Output: $\mathbf{F}$, *i.e.,* child label subsets $\{\Pi^{i+1|jk+u}\}_{u=1}^{u=k}$

Algorithm 2 Generative process of Cbolt

Input: An XML dataset $\Omega = \{(\mathbf{x}_g, \mathbf{y}_g)\}_{g=1}^{g=n}$
Repeat for each label subset $\Pi^{i|j}$,
  N-Step: generate new balanced subsets by **Algorithm** 1
  P-Step: train the path classifier
  O-Step: generate overlapping label subsets with Eqs.8 and 9
Until the tree depth h is reached
Output: A label tree $\mathcal{T} = \{(\Pi^{i|j}, C^{i|j})\}_{i=0,j=1}^{i=h,j=k^i}$

$s \in \Pi^{h|u}$, whose positive instances are defined as $\mathcal{X}_s^+ = \{\mathbf{x}_g | y_{gs} = 1, g \in [n]\}$. Specially considering the overlapping setting, there may be several subsets containing label s. Accordingly, its negative instances are all positive instances of those subsets plus $\mathcal{X}_s^+$, defined as $\mathcal{X}_s^- = \{\mathbf{x}_g | y_{gs} = 1 \ \& \ s, i \in \Pi^{h|u}, s \neq i, g \in [n]\}_{u=1}^{u=k^h}$. We train an unique SVMs classifier for the label over the training dataset $\{(\mathcal{X}_s^+, +1), (\mathcal{X}_s^-, -1)\}$.

3.3 Prediction with Cbolt

We focus on the prediction process of selecting the candidate leaf nodes. We employ the greedy method to achieve efficient prediction, where we find the candidate leaf nodes by greedily traversing several most probable paths at each level. We refer the readers to more details in [26, 31].

Specifically, for each test example $\mathbf{x}$, we denote its selected set of candidate leaf nodes by $\ddot{\Pi}^h$, and the path score of each candidate leaf node $\ddot{\Pi}^{h|u}$ is computed as follows:

$$\lambda_u^*(\mathbf{x}) = \prod_{(i|j) \in \mathcal{A}_u} C^{i|j}(\mathbf{x})_u, \quad \ddot{\Pi}^{h|u} \in \ddot{\Pi}^h, \tag{10}$$

where $\mathcal{A}_u$ is the node set of the corresponding path, and $C^{i|j}$ denotes the child node in this path. All labels within the candidate leaf nodes are treated as candidate labels. The final prediction is made by ranking the product of the path score and predictive score of base classifier, where the path score of any label is naturally as follows:

$$\lambda_s^*(\mathbf{x}) = \lambda_u^*(\mathbf{x}), \quad s \in \ddot{\Pi}^{h|u} \tag{11}$$

Considering each overlapping label s covered by multiple candidate leaf nodes, its path score is the maximum one among all those leaves:

$$\lambda_s^*(\mathbf{x}) = \max_u \lambda_u^*(\mathbf{x}), \quad s \in \ddot{\Pi}^{h|u} \tag{12}$$

The intuition is that the main aim of the O-Step is to preserve the marginal labels that are easily missed, while we consider the maximum path score implies the most appropriate path if multiple paths are preserved.

3.4 Time Complexity of Cbolt

For simplicity, we analyze the time complexity by using the OvA method as both path and base classifiers. The following assumptions are made: 1) each text has $\log(l)$ relevant labels averagely, 2) each text feature has $\hat{d}$ non-zero values averagely. In the N-Step, it takes $O(Tkl)$ time to cluster labels for each level, where T is the iteration number of weighted k-means clustering. In the P-Step, training the OvA path classifiers require $O(nk\hat{d}\log(l))$ time for each level. The O-Step has to predict training instances, requiring $O(nk\hat{d}\log(l))$ times for each level. In summary, the time complexity of Cbolt generation is about $O((nk\hat{d}\log(l)+Tkl\hat{d})\log(l))$ time. We will show the efficiency evaluations in the experimental part.

4 EXPERIMENT

In this section, we mainly attempt to answer the following questions:

Q1: Whether the label trees generated by Cbolt can be effective for XML tasks?
Q2: Whether Cbolt can be more efficient with balanced problem scales?
Q3: Whether applying Cbolt can achieve competitive performance with strong XML baselines?

Table 3: Comparisons among the label tree methods. The best results of all methods are presented in bold.

	EURLex-4K			Wiki10-31K			AmazonCat-13K			WikiLSHTC-325K			Amazon-670K		
	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
Parabel	82.2	68.8	57.5	84.3	72.9	63.6	**93.0**	**79.2**	**64.5**	65.1	43.2	32.0	44.9	39.8	36.0
Bonsai	82.9	69.5	58.0	84.7	73.7	64.6	92.9	79.0	64.3	66.4	44.4	32.9	45.5	40.3	36.5
CBOLT	**83.2**	**70.0**	**58.3**	**85.0**	**73.8**	**65.2**	92.9	**79.2**	**64.5**	**66.5**	**44.6**	**33.1**	**45.9**	**40.8**	**37.1**
	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5
Parabel	36.5	44.1	48.3	11.6	12.8	13.8	50.9	64.0	72.1	26.8	33.3	37.3	25.5	29.4	32.9
Bonsai	37.0	45.0	49.4	11.8	13.4	14.7	51.2	64.5	72.4	28.2	35.4	39.6	26.1	30.2	33.6
CBOLT	**37.2**	**45.4**	**49.7**	**12.0**	**13.5**	**15.0**	**51.3**	**64.8**	**72.7**	**28.3**	**35.9**	**40.6**	**26.5**	**30.9**	**34.6**

Table 4: Comparisons among the label tree methods predicted by a single tree. The top results are indicated in bold.

	EURLex-4K			Wiki10-31K			AmazonCat-13K			WikiLSHTC-325K			Amazon-670K		
	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
Parabel	81.0	67.7	56.6	84.0	72.6	63.2	92.3	78.5	63.9	62.9	41.3	30.3	43.3	38.0	34.1
Bonsai	82.3	69.0	**57.6**	84.3	**73.6**	64.5	92.6	78.7	64.0	65.2	43.2	31.9	44.4	39.2	35.4
CBOLT	**82.5**	**69.2**	**57.6**	**84.5**	**73.6**	**64.8**	**92.7**	**78.8**	**64.2**	**65.3**	**43.4**	**32.1**	**44.8**	**39.8**	**36.1**
	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5
Parabel	36.0	43.3	47.6	11.6	12.8	13.8	50.9	63.6	71.3	25.8	31.4	34.7	24.6	28.0	30.9
Bonsai	**36.7**	44.8	49.0	11.8	**13.4**	14.7	**51.2**	64.3	71.9	27.5	34.1	38.1	25.4	29.3	32.6
CBOLT	**36.7**	**44.9**	**49.1**	**11.9**	**13.4**	**14.8**	**51.2**	**64.4**	**72.2**	**27.7**	**34.7**	**39.0**	**25.7**	**30.0**	**33.5**

Table 5: Results of L-P, L-R, and L-F1. The top results are indicated in bold.

	EURLex-4K			Wiki10-31K			AmazonCat-13K			WikiLSHTC-325K			Amazon-670K		
	L-P	L-R	L-F1	L-P	L-R	L-F1	L-P	L-R	L-F1	L-P	L-R	L-F1	L-P	L-R	L-F1
Parabel	28.3	85.6	42.5	54.5	48.2	51.1	**27.6**	93.4	**42.6**	18.7	72.0	29.7	16.7	59.5	26.1
Bonsai	28.1	**91.3**	42.9	49.9	**80.0**	59.9	20.0	**97.9**	33.3	18.5	**80.2**	30.1	16.5	68.9	26.6
CBOLT	**30.9**	88.9	**45.8**	**57.4**	68.3	**62.4**	21.7	96.7	35.4	**20.7**	77.0	**32.6**	**18.4**	**69.5**	**29.1**

Datasets. We employ 5 benchmark XML datasets downloaded from the public extreme classification repository[3] [4], including 2 large-scale datasets EURLex-4K and Wiki10-31K, and 3 extreme-scale datasets AmazonCat-13K, WikiLSHTC-325K, and Amazon-670K. Their statistics are described in Table 2.

Metrics. We employ two prevalent ranking-based XML metrics Precision@K (P@K) and Propensity Scored Precision@K (S@K) to measure the performance. Following the prior studies [40, 42], in the experiment we will report the results of P@K and S@K as $K = 1, 3, 5$. The two metrics are formally defined as follows:

$$\text{P@}K = \frac{1}{K} \sum_{i \in \text{rank}_K(\hat{\mathbf{y}})} \mathbf{y}_i,$$

$$\text{PSP@}K = \frac{1}{K} \sum_{i \in \text{rank}_K(\hat{\mathbf{y}})} \frac{\mathbf{y}_i}{p_i},$$

where $\text{rank}_K(\mathbf{y})$ returns the K largest indices of $\mathbf{y}$ ranked in descending order, and p_i expresses the propensity score for the i-th label [4]. We note that the metric of PSP@K takes the occurrence number of each label into consideration.

Implementation details. We train 100-ary label trees for CBOLT, *i.e.,* $k = 100$. The balanced coefficient β is set to 2; the overlapping coefficient ρ is set to $\{0.5, 0.7\}$, and the overlapping upper bound parameter γ is set to 1.0 for all datasets. The tree depth h is fixed to 3 for WikiLSHTC-325K and Amazon-670K, and fixed 2 for the other three datasets. Following previous studies [26, 31], we report the ensemble results of 3 independent label trees.

4.1 Evaluation of CBOLT (*Q1*)

To answer ***Q1***, we compare CBOLT against two existing label tree methods Parabel [31] and Bonsai [26]. To exactly evaluate the quality of the label trees, we employ a simple shallow classifier, *i.e.,* vanilla OvA method with SVMs, as both path and base classifiers for all comparing label tree methods.

Evaluation by P@K and S@K. As shown in Table 3, we can observe that CBOLT outperforms Parabel and Bonsai, and achieves the highest scores of P@K and S@K in 29/30 cases. The results directly indicate that the label trees generated by CBOLT are superior

[3] http://manikvarma.org/downloads/XC/XMLRepository.html

Table 6: Running times (minutes) of comparing label tree methods with a single thread and 50 threads, respectively. T_t and C_t denote the running times of generating the label trees and training the base classifiers, respectively.

k-ray	Thread	Dataset	Parabel		Cbolt-O		Cbolt	
			T_t (C_t)	Total	T_t (C_t)	Total	T_t (C_t)	Total
$k = 2$	1	EURLex-4K	0.24 (0.72)	0.96	0.24 (0.69)	0.94	0.27 (1.04)	1.31
		Wiki10-31K	3.64 (6.44)	10.07	3.34 (6.42)	9.76	4.04 (7.65)	11.68
		AmazonCat-13K	6.72 (21.92)	28.64	13.25 (21.65)	35.9	17.34 (32.39)	49.73
		WikiLSHTC-325K	14.72 (132.10)	146.82	11.22 (126.20)	137.42	12.99 (172.01)	185
		Amazon-670K	9.77 (32.59)	42.36	9.72 (31.95)	41.67	12.21 (36.87)	49.08
	50	EURLex-4K	0.17 (0.28)	0.45	0.18 (0.25)	0.43	0.20 (0.38)	0.58
		Wiki10-31K	3.18 (0.30)	3.48	2.46 (0.25)	2.71	3.07 (0.36)	3.43
		AmazonCat-13K	4.26 (2.05)	6.31	7.11 (1.57)	8.68	8.36 (2.27)	10.63
		WikiLSHTC-325K	11.72 (7.59)	19.31	12.05 (6.80)	18.85	13.95 (10.52)	24.47
		Amazon-670K	8.66 (1.28)	9.94	8.53 (1.27)	9.8	9.67 (1.79)	11.46
k-ray	Thread	Dataset	Bonsai		Cbolt-O		Cbolt	
			T_t (C_t)	Total	T_t (C_t)	Total	T_t (C_t)	Total
$k = 100$	1	EURLex-4K	0.66 (0.74)	1.4	0.57 (0.60)	1.17	0.57 (0.62)	1.19
		Wiki10-31K	11.80 (12.20)	24.01	6.11 (6.39)	12.49	6.18 (9.37)	15.55
		AmazonCat-13K	16.01 (75.66)	91.67	16.76 (35.34)	52.1	22.88 (51.24)	74.12
		WikiLSHTC-325K	65.64 (182.12)	247.76	57.75 (102.42)	160.17	62.91 (177.20)	240.11
		Amazon-670K	35.77 (59.79)	95.96	30.78 (29.90)	60.68	40.90 (37.39)	78.29
	50	EURLex-4K	0.04 (0.28)	0.32	0.03 (0.20)	0.23	0.03 (0.22)	0.24
		Wiki10-31K	0.77 (1.30)	2.06	0.32 (0.80)	1.12	0.46 (1.03)	1.48
		AmazonCat-13K	0.88 (9.13)	10.01	0.85 (2.91)	3.76	0.92 (3.99)	4.92
		WikiLSHTC-325K	6.69 (9.09)	15.77	5.75 (7.22)	12.97	6.18 (7.28)	13.46
		Amazon-670K	4.03 (2.78)	6.81	3.47 (1.17)	4.64	5.00 (1.24)	6.24

Table 7: Comparing with existing XML baselines across Amazon-670K. The best results are indicated in bold.

Method	P@1	P@3	P@5	S@1	S@3	S@5
SLEEC	35.1	31.3	28.6	20.6	23.3	26.0
AnnexML	42.4	36.9	33.0	21.6	24.8	27.7
FastXML	37.0	33.3	30.5	19.4	23.3	26.9
DiSMEC	44.7	39.7	36.1	27.8	30.6	34.1
Parabel	44.9	39.8	36.0	25.4	29.4	32.9
Bonsai	45.6	40.4	36.6	27.1	30.8	34.1
ProXML	43.5	38.7	35.3	30.8	32.8	35.1
PfastreXML	39.5	35.8	33.1	29.3	30.8	32.4
XML-CNN	35.4	31.9	29.3	28.7	33.3	36.5
XT	42.5	37.9	34.4	24.8	28.2	31.2
AttentionXML	47.6	42.6	38.9	30.3	33.9	37.1
APLC-XLNet	43.5	38.8	35.3	26.1	29.7	32.8
DeepXML	47.8	42.8	39.1	32.1	35.1	37.8
LightXML	**49.1**	43.8	39.8	-	-	-
Cbolt + LightXML	49.0	**43.9**	**40.0**	**32.4**	**35.3**	**38.0**

to the ones of Parabel and Bonsai, since all comparing methods share the same path and base classifiers. The performance improvements of Cbolt can be more significant across WikiLSHTC-325K and Amazon-670K, which contain much more labels. This observation implies that Cbolt can be more effective when the label trees become larger, making Cbolt more practical in real-world applications. Additionally, we also see that the improvements of S@K are higher than those of P@K, demonstrating the superior performance of Cbolt for infrequent labels. We consider that the improvements of S@K are mainly caused by the overlapping technique, we will further discuss this phenomenon in the ablative study part.

Evaluation by a single tree. To avoid the interference of ensemble learning, we further compare Cbolt with Parabel and Bonsai by using a single tree. The results are reported in Table 4. We can observe very similar trends comparing with the ensemble results of 3 trees, where Cbolt also achieves the highest scores in all cases. Those results further indicate that Cbolt can output higher quality label trees compared with the existing baselines.

Evaluation by the accuracy of candidate leaf nodes. To examine the accuracy of the candidate leaf nodes, we suggest a novel metric for label trees named **L**eaf-**F1** (**L-F1**). Borrowing the spirit of F1 score, the L-F1 is a combination of **L**eaf-**P**recision (**L-P**) and **L**eaf-**R**ecall (**L-R**). For each test instance ($\mathbf{x}, \mathbf{y}$), the three metrics

Table 8: Ablative study of CBOLT. The top results are presented in bold.

	EURLex-4K			Wiki10-31K			AmazonCat-13K			WikiLSHTC-325K			Amazon-670K		
	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
CBOLT-W	82.9	69.5	58.0	84.7	73.7	64.6	**92.9**	79.0	64.3	66.4	44.4	32.9	45.5	40.3	36.5
CBOLT-O	83.0	69.6	58.1	84.7	73.7	65.0	**92.9**	79.1	64.5	**66.6**	44.5	33.0	45.6	40.5	36.8
CBOLT	**83.2**	**70.0**	**58.3**	**85.0**	**73.8**	**65.2**	**92.9**	**79.2**	**64.5**	66.5	**44.6**	**33.1**	**45.9**	**40.8**	**37.1**
	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5	S@1	S@3	S@5
CBOLT-W	37.0	45.0	49.4	11.8	13.4	14.7	51.2	64.5	72.4	28.2	35.4	39.6	26.1	30.2	33.6
CBOLT-O	37.1	45.0	49.5	11.9	13.4	14.9	**51.3**	64.6	72.6	**28.3**	35.5	39.9	26.1	30.3	33.9
CBOLT	**37.2**	**45.4**	**49.7**	**12.0**	**13.5**	**15.0**	**51.3**	**64.8**	**72.7**	**28.3**	**35.9**	**40.6**	**26.5**	**30.9**	**34.6**

are defined as follows:

$$\text{L-P} = \frac{\sum_{\ddot{\Pi}^{h|u} \in \ddot{\Pi}^h} \mathbb{I}(\{s|y_s = 1\} \cap \ddot{\Pi}^{h|u} \neq \emptyset)}{|\ddot{\Pi}^h|},$$
$$\text{L-R} = \frac{\sum_{\ddot{\Pi}^{h|u} \in \ddot{\Pi}^h} |\{s|y_s = 1\} \cap \ddot{\Pi}^{h|u}|}{|\{s|y_s = 1\}|},$$
$$\text{L-F1} = \frac{2 \times \text{L-P} \times \text{L-R}}{\text{L-P} + \text{L-R}}, \tag{13}$$

where $\ddot{\Pi}^{h|u}$ denotes a candidate leaf node of $\mathbf{x}$, and $\ddot{\Pi}^h$ is the set of all candidate leaf nodes. Specially, we note that the L-P measures the proportion of candidate leaf nodes that contains the ground-truth labels. It is used to handle the case of high L-R scores, however, caused by certain leaf nodes with big label subsets. By combining L-P and L-R, the L-F1 can precisely measure the accuracy of candidate leaf nodes.

We report the results of L-P, L-R, and L-F1 in Table 5. Most importantly, it can be clearly seen that the L-F1 scores of CBOLT outperform those of Parabel and Bonsai. In more detail, the results of L-F1 imply that the label trees of CBOLT can select more precise candidate labels within candidate leaf nodes during prediction. This indirectly proves that the weighted clustering technique of the N-Step alleviates the inconsistent path problem to some extent. Besides, we can see that Bonsai achieves higher L-R scores in most settings. We kindly argue that is because its leaf nodes are imbalanced and it contains a number of leaf nodes with big label subsets.

4.2 Evaluation of Efficiency (*Q2*)

To answer ***Q2***, we report the running times of comparing label tree methods with a single thread and 50 threads. In more detail, we report the running times of generating label trees (training the base classifiers) and total time, respectively. The running times without the O-Step, denoted by "-O", are also presented. All efficiency evaluations are conducted on a 32-core AMD Threadripper 3970X 3.7GHz processor with 64GB RAM.

We show the results in Table 6, and make the following findings. First, we can see that the base classifiers' training times of CBOLT-O are shorter than those of Parabel and Bonsai in all cases. The improvements strongly indicate the effectiveness of the balanced constraint of CBOLT, and especially, the improvements over Parabel demonstrate that the balanced problem scales are superior to the balanced labels to reduce the time cost. Second, the full CBOLT, even with overlapping labels, can be more efficient than Bonsai in most cases, which are mainly benefit from the improvements with faster base classifiers training. Therefore, we consider that CBOLT is a strong candidate label tree method for XML, since it is both effective and efficient compared to the existing methods. Third, CBOLT only takes a little more than CBOLT-O in most cases, further indicating the efficiency of CBOLT. Finally, although the 2-ary label trees run faster with a single thread, they become less efficient than 100-ary label trees with multiple threads. In those cases, CBOLT can achieve higher efficiency than Parabel, therefore we argue that CBOLT is a better proposal than Parabel in practical scenarios with parallelization.

4.3 Applying CBOLT to Existing XML Methods (*Q3*)

To answer ***Q3***, we apply CBOLT[4] to a recent language model-based XML method LightXML [24]. We then compare it against 14 existing XML baseline methods including SLEEC [5], AnnexML [37], FastXML [32], DiSMEC [2], Parabel [31], Bonsai [26], ProXML [3], PfastreXML [22], XML-CNN [27], XT [39], AttentionXML [43], X-Transformer [10], APLC-XLNet [41], DeepXML [13], and LightXML [24]. The results[5] on Amazon-670K are reported in Table 7. Because the datasets are completely the same as previous XML studies, for each baseline we directly show their results reported in the original papers.

Overall, we can observe that CBOLT +LightXML can achieve the highest scores among all comparing XML methods in most cases. Most importantly, CBOLT +LightXML performs better than LightXML in 2/3 settings especially on P@3 and P@5. First, the results can be integrate with existing XML methods and effectively improve the performance. As a generic step, we kindly argue that CBOLT can be applied to other strong XML methods, and achieve higher performance. Second, the improvements of P@3 and P@5 over LightXML indicate that CBOLT can improve the overall classification accuracy from the ranking perspective, which may be more significant for the scenarios with more labels per-instances. To some extent, the results implies the robustness of CBOLT.

[4]We use the default settings of LightXML.
[5]We only report the results on Amazon-670K since the original LightXML hasnot generated label trees for EURLex-4K, Wiki10-31K, and AmazonCat-13K [24], while WikiLSHTC-325K contains no raw texts.

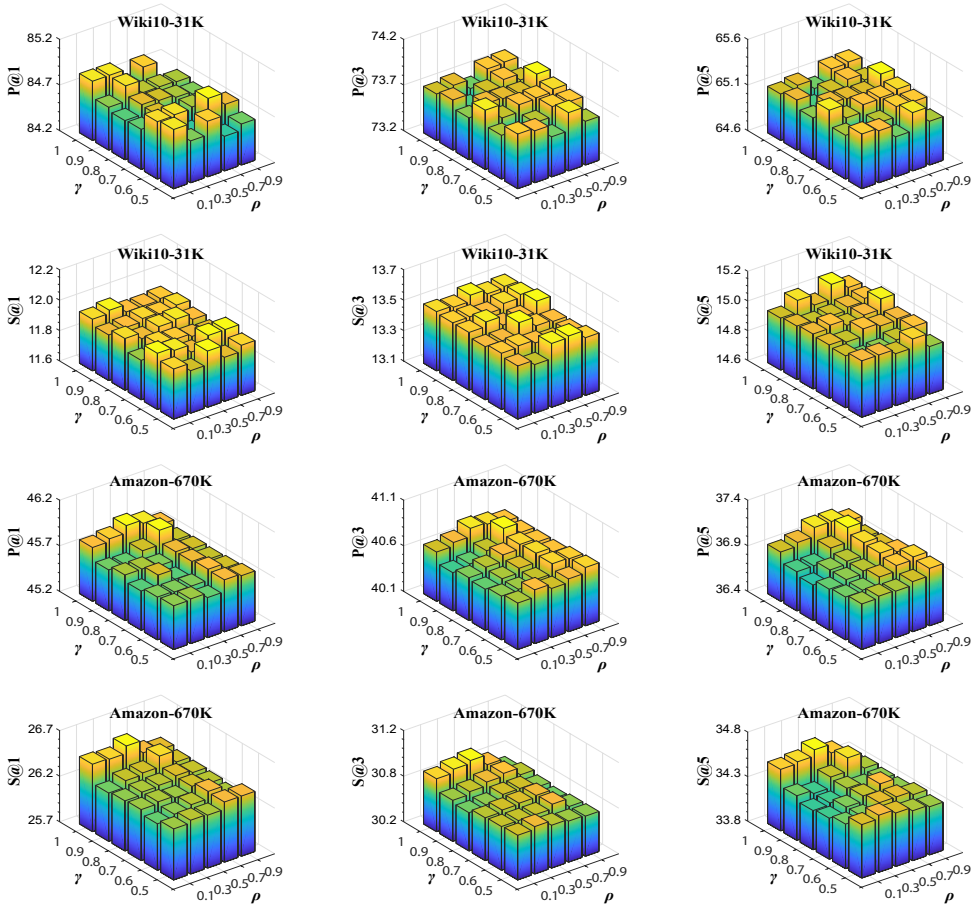


Figure 2: Sensitivity analysis of ρ and γ across Wiki10-31K and Amazon-670K.

4.4 Sensitivity Analysis

Ablative study. We conduct ablative studies to examine the effectiveness of the two key ideas of CBOLT. The notation "-W" refers to the removal of the clustering weights in Eq.3, and we re-declare that "-O" refers to the removal of the O-Step. The results of the ablative study are shown in Table 8. First, the performance gain of weighted clustering indicates that it can solve the inconsistent path problem to some extent, giving more accurate candidate leaf nodes. Second, the performance gain of the O-Step, especially the significant improvement of S@K, proves that it can effectively preserve the marginal labels, most of which are infrequent labels.

Analysis of ρ and γ. We empirically evaluate the impacts of the overlapping coefficient ρ, and the overlapping upper bound parameter γ. To this end, we report the scores of P@K and S@K across Wiki10-31K and Amazon-670K. Fig.2 shows the results with different values of ρ over $\{0.1, 0.3, \cdots, 0.9\}$ and γ over $\{0.5, 0.6, \cdots, 1\}$. We can observe that on both datasets, the best results usually appear when $\rho = \{0.5, 0.7\}$ and $\gamma = 1.0$, indicating that the overlapping labels have positive effects on the results.

5 CONCLUSION

One practical solution for XML is to build label trees to partition the large-scale XML problem into a number of small-scale subproblems. However, the existing methods of label trees suffer from the inconsistent path and disjoint label subset problems. In this paper, we aim to solve both problems and propose a novel method of label tree named CBOLT. The basic idea of CBOLT is to adopt a weighted clustering method to generate new label subsets and allow the marginal labels belonging to multiple label subsets, leading to consistent and overlapping label subsets of new nodes. Additionally, we propose a balanced regularization to make the problem scales of leaf nodes balanced, enabling to improve the efficiency. Empirical results demonstrate that CBOLT can outperform the existing label tree methods and be competitive with the strong XML baselines.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support for this project from the National Key R&D Program of China (No.2021ZD0112501, No.2021ZD0112502), and the project of Computer Education Research Association of Chinese Universities (No.CERACU2022R13).

REFERENCES

[1] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. 2013. Multi-label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages. In *International World Wide Web Conference*. 13–24.

[2] Rohit Babbar and Bernhard Schölkopf. 2017. DiSMEC: Distributed Sparse Machines for Extreme Multi-label Classification. In *ACM International Conference on Web Search and Data Mining*. 721–729.

[3] Rohit Babbar and Bernhard Schölkopf. 2019. Data Scarcity, Robustness and Extreme Multi-label Classification. *Machine Learning* 108, 8–9 (2019), 1329–1351.

[4] Kush Bhatia, Kunal Dahiya, Himanshu Jain, Purushottam Kar, Anshul Mittal, Yashoteja Prabhu, and Manik Varma. 2016. The Extreme Classification Repository: Multi-label Datasets and Code.

[5] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse Local Embeddings for Extreme Multi-label Classification. In *Neural Information Processing Systems*. 730–738.

[6] Ella Bingham and Heikki Mannila. 2001. Random Projection in Dimensionality Reduction: Applications to Image and Text Data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 245–250.

[7] Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. 2010. Random Projections for k-means Clustering. In *Neural Information Processing Systems*. 298–306.

[8] Timothy I. Cannings and Richard J. Samworth. 2017. Random-Projection Ensemble Classification. *Journal of the Royal Statistical Society Series B* 79, 4 (2017), 959–1035.

[9] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit Dhillon. 2019. X-BERT: eXtreme Multi-label Text Classification with using Bidirectional Encoder Representations from Transformers. *arXiv preprint arXiv:1905.02331* (2019).

[10] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 3163–3171.

[11] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 3163–3171.

[12] Kenneth L. Clarkson and David P. Woodruff. 2017. Low-rank Approximation and Regression in Input Sparsity Time. *J. ACM* 63, 6 (2017), 1–45.

[13] Kunal Dahiya, Deepak Saini, Anshul Mittal, Ankush Shaw, Kushal Dave, Akshay Soni, Himanshu Jain, Sumeet Agarwal, and Manik Varma. 2021. DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents. In *ACM International Conference on Web Search and Data Mining*. 31–39.

[14] Sanjoy Dasgupta and Anupam Gupta. 2003. An Elementary Proof of A Theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms* 22, 1 (2003), 60–65.

[15] Petros Drineas, Michael W Mahoney, Shan Muthukrishnan, and Tamás Sarlós. 2011. Faster Least Squares Approximation. *Numerische mathematik* 117, 2 (2011), 219–249.

[16] Robert Durrant and Ata Kabán. 2013. Sharp Generalization Error Bounds for Randomly-projected Classifiers. In *International Conference on Machine Learning*. 693–701.

[17] Robert J. Durrant and Ata Kabán. 2015. Random Projections as Regularizers: Learning A Linear Discriminant From Fewer Observations than Dimensions. *Machine Learning* 99, 2 (2015), 257–286.

[18] Xiaoli Zhang Fern and Carla E. Brodley. 2003. Random Projection for High Dimensional Data Clustering: A Cluster Ensemble Approach. In *International Conference on Machine Learning*. 186–193.

[19] Zhiqi Ge and Ximing Li. 2021. To Be or not to Be, Tail Labels in Extreme Multi-label Learning. In *ACM International Conference on Information and Knowledge Management*. 555–564.

[20] Vivek Gupta, Rahul Wadbude, Nagarajan Natarajan, Harish Karnick, Prateek Jain, and Piyush Rai. 2019. Distributional Semantics Meets Multi-label Learning. In *AAAI Conference on Artificial Intelligence*. 3747–3754.

[21] Himanshu Jain, Venkatesh Balasubramanian, Bhanu Chunduri, and Manik Varma. 2019. Slice: Scalable Linear Extreme Classifiers trained on 100 Million Labels for Related Searches. In *ACM International Conference on Web Search and Data Mining*. 528–536.

[22] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 935–944.

[23] Ankit Jalan and Purushottam Kar. 2019. Accelerating Extreme Classification via Adaptive Feature Agglomeration. In *International Joint Conference on Artificial Intelligence*. 2600–2606.

[24] Ting Jiang, Deqing Wang, Leilei Sun, Huayi Yang, Zhengyang Zhao, and Fuzhen Zhuang. 2021. LightXML: Transformer with Dynamic Negative Sampling for High-Performance Extreme Multi-label Text Classification. In *AAAI Conference on Artificial Intelligence*. 7987–7994.

[25] William B. Johnson, Joram Lindenstrauss, and Gideon Schechtman. 1984. Extensions of Lipschitz Mappings into A Hilbert Space. *Contemp. Math.* 26, 189-206 (1984), 1.

[26] Sujay Khandagale, Han Xiao, and Rohit Babbar. 2020. Bonsai: Diverse and Shallow Trees for Extreme Multi-label Classification. *Machine Learning* 109, 11 (2020), 2099–2119.

[27] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep Learning for Extreme Multi-label Text Classification. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 115–124.

[28] Weiwei Liu, Haobo Wang, Xiaobo Shen, and Ivor Tsang. 2020. The Emerging Trends of Multi-Label Learning. *arXiv preprint arXiv:2011.11197* (2020).

[29] Saurabh Paul, Christos Boutsidis, Malik Magdon-Ismail, and Petros Drineas. 2014. Random Projections for Linear Support Vector Machines. *ACM Transactions on Knowledge Discovery from Data* 8, 4 (2014), 1–25.

[30] Yashoteja Prabhu, Anil Kag, Shilpa Gopinath, Kunal Dahiya, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Extreme Multi-label Learning with Label Features for Warm-start Tagging, Ranking & Recommendation. In *International Conference on Web Search and Data Mining*. 441–449.

[31] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising. In *The Web Conference*. 993–1002.

[32] Yashoteja Prabhu and Manik Varma. 2014. FastXML: A Fast, Accurate and Stable Tree-classifier for eXtreme Multi-label Learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 263–272.

[33] Tamas Sarlos. 2006. Improved Approximation Algorithms for Large Matrices via Random Projections. In *Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 143–152.

[34] Alon Schclar and Lior Rokach. 2009. Random Projection Ensemble Classifiers. In *International Conference on Enterprise Information Systems*. Springer, 309–316.

[35] Qinfeng Shi, Chunhua Shen, Rhys Hill, and Rhys Hill. 2012. Is Margin Preserved after Random Projection?. In *International Conference on Machine Learning*. 643–650.

[36] Wissam Siblini, Pascale Kuntz, and Frank Meyer. 2018. CRAFTML, an Efficient Clustering-based Random Forest for Extreme Multi-label Learning. In *International Conference on Machine Learning*. 4671–4680.

[37] Yukihiro Tagami. 2017. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 455–464.

[38] Yashaswi Verma. 2019. An Embarrassingly Simple Baseline for eXtreme Multi-label Prediction. *arXiv preprint arXiv:1912.08140* (2019).

[39] Marek Wydmuch, Kalina Jasinska, Mikhail Kuznetsov, Róbert Busa-Fekete, and Krzysztof Dembczynskii. 2018. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *Neural Information Processing Systems*. 6358–6368.

[40] Guangxu Xun, Kishlay Jha, Jianhui Sun, and Aidong Zhang. 2020. Correlation Networks for Extreme Multi-label Text Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1074–1082.

[41] Hui Ye, Zhiyu Chen, Da-Han Wang, and Brian D. Davison. 2020. Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Clusters for Extreme Multi-label Text Classification. In *International Conference on Machine Learning*. 10809–10819.

[42] Hui Ye, Zhiyu Chen, Da-Han Wang, and Brian Davison. 2020. Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Clusters for Extreme Multi-label Text Classification. In *International Conference on Machine Learning*. 10809–10819.

[43] Ronghui You, Zihan Zhang, Ziye Wang, Suyang Dai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2019. AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification. In *Neural Information Processing Systems*. 5812–5822.

[44] Wenjie Zhang, Junchi Yan, Xiangfeng Wang, and Hongyuan Zha. 2018. Deep Extreme Multi-label Learning. In *ACM on International Conference on Multimedia Retrieval*. 100–107.