Random Label Forests: An Ensemble Method with Label Subsampling For Extreme Multi-Label Problems

Sheng-Wei Chen
National Taiwan University
d09944003@ntu.edu.tw

Chih-Jen Lin
National Taiwan University / MBZUAI
cjlin@csie.ntu.edu.tw

Abstract

Text classification is one of the essential topics in natural language processing, and each text is often associated with multiple labels. Recently, the number of labels has become larger and larger, especially in the applications of e-commerce, so handling text-related e-commerce problems further requires a large memory space in many existing multi-label learning methods. To address the space concern, utilizing a distributed system to share that large memory requirement is a possible solution. We propose "random label forests," a distributed ensemble method with label subsampling, for handling extremely large-scale labels. Random label forests can reduce memory usage per computer while keeping competitive performances over real-world data sets.

1 Introduction

Text classification is one of the essential topics in natural language processing fields. There are many valuable applications, such as product categorization for e-commerce (Shen et al., 2011; Agrawal et al., 2013; McAuley and Leskovec, 2013), coding diagnosis and procedures in medical records (Nuthakki et al., 2019), and document tagging (Zubiaga, 2009). Usually, the prediction in the text classification can be multi-labeled. Hence, a text classification problem falls into the category of multi-label classification, which is used to find the relevant labels of a data instance. For example, we can set the document content and tags in the document tagging problem as the feature and labels in a multi-label problem.

Recently, the number of labels has become larger and larger, especially in the applications of e-commerce. Thus, an emerging topic is extreme multi-label learning (XML), which focuses on large-scale candidate labels, input instances, and input features. Because of these three large-scale components, an XML method should further consider the model training time and memory usage in addition to the performance.

A simple and classic way to solve a multi-label classification problem is by the one-versus-rest (OVR) method with linear models. However, the time complexity and model size directly depend on the number of labels and features. To handle problems with many labels, two existing extensions of OVR are developed:

- DiSMEC (Babbar and Schölkopf, 2017) splits the label set into several subsets and lets each machine in a distributed system handle one subset. This way, the training time and model size per machine are reduced.
- Tree-based linear methods (Tsoumakas et al.; Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022) utilize the divide-and-conquer paradigm on labels via clustering methods such as K-means and then apply the OVR method to train the smaller problems on the clusters. We discuss the details in Section 2.2.

Besides the linear methods, we can also use low-rank embedding on features and labels to reduce the training time and memory usage, e.g., (Bhatia et al., 2015; Yu et al., 2014). However, some works (Khandagale et al., 2020; Babbar and Schölkopf, 2017) report that the distribution of positive instances over labels is long-tail in most XML data sets, so the label space cannot be well-approximated to a low-rank embedding space. For the deep-learning methods, some earlier works (Kim, 2014; Liu et al., 2017) only show competitive performance on short-text XML problems (Yu et al., 2022, Section 5.2). However, several tree-based deep-learning methods (You et al., 2019; Jiang et al., 2021; Zhang et al., 2021) can rank better on the public XML benchmark (Bhatia et al., 2016). Nevertheless, although neural network models perform better in many fields, Lin et al. (2023) point out that the linear model is still a strong baseline for certain multi-label classification data. Fur-

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 14107–14119

thermore, the linear models are easy to understand and more explainable, so we focus on linear models in this work.

Let us go back to discussing the linear methods. Although DiSMEC and tree-based linear methods are reasonable solutions for XML problems, each method still has disadvantages. DiSMEC can only handle a small number of labels, but not large-scale labels, on each machine. Tree-based linear methods require large memory space to handle all the labels during the training. Hence, we hope a method can take the advantages of DiSMEC and tree-based methods without having their disadvantages.

Label subsampling is another way to divide an XML problem into more minor subproblems. RAKEL (Tsoumakas and Vlahavas, 2007) is a pioneer in using label subsampling with the ensemble method. After the label subsampling, RAKEL converts each small-scale multi-label subproblem to a multi-class one by considering every label combination as a new class label. This setting, referred to as "label powerset" in multi-label learning, is not scalable to XML because covering the predictions of highly large-scale labels by the powerset method is almost impossible.

In this work, we utilize label subsampling and the distributed system to reduce the impact of the large-scale labels. Specifically, each computer can solve a smaller XML subproblem via some existing XML methods, such as tree-based linear methods. Hence, handling XML problems with billions of labels or more becomes practical. Since we use the label subsampling technique with the tree-based linear method, we call this method "random label forests." Let us list our contributions as follows.

- We propose a natively parallel framework, random label forests, which is an ensemble method with label subsampling for the XML problem.
- Our experiments show that random label forests are competitive with the standard tree-based methods applied to all labels.
- We analyze the model size of tree-based methods and explain why random label forests can reduce memory usage in each computer of the distributed system.
- We also analyze the time complexity of tree-based methods. The training time of random label forests is shorter than the tree-based methods with all labels.

Section 2 discusses OVR and tree-based methods for XML problems and explains why we only consider linear methods in this work. Section 3 focuses on distributing a tree-based model and then presents random label forests, including discussions on the data processing, time complexity, and model size. Section 4 shows comparison results on performance, training time, and model size.

2 Multi-Label Problems

A multi-label classification problem aims to find a function f with the parameter $\boldsymbol{\theta}$ that can predict whether a given instance $\boldsymbol{x}$, which is a feature vector in $\mathbb{R}^n$, is associated with the label-j for $j = 1, \ldots, m$, where n and m are the feature dimension and the number of labels. We use 0/1 to indicate if an instance is associated without/with a label. Hence, we can denote $\boldsymbol{y} \in \{0, 1\}^m$ as the label vector of the instance $\boldsymbol{x} \in \mathbb{R}^n$ so that the predictions $f(\boldsymbol{x}; \boldsymbol{\theta})$ can be as close to $\boldsymbol{y}$ as possible. Based on the definition above, past works (Babbar and Schölkopf, 2017; Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022) utilize linear models to handle the multi-label classification problem, and other works (Kim, 2014; Liu et al., 2017; You et al., 2019; Jiang et al., 2021; Zhang et al., 2021) use the different architectures of neural networks. Although neural network models perform better in many fields, Lin et al. (2023) point out that the linear model is still a strong baseline for certain multi-label classification data. Furthermore, the linear models are easy to understand and more explainable, so we focus on linear models in this work.

2.1 One-Versus-Rest Method

The OVR method involves training a single model per label, with the instances of that label as positives and all other instances as negatives. Thus, when training an OVR linear model on a multi-label classification problem with the training set

$$D = \{(\boldsymbol{y}_i, \boldsymbol{x}_i) \in (\{0, 1\}^m, \mathbb{R}^n) \mid i = 1, \ldots, l\},$$

where l is the number of the training instances, we solve m subproblems

$$\min_{\boldsymbol{w}_j \in \mathbb{R}^n} \frac{\lambda}{2} \boldsymbol{w}_j^T \boldsymbol{w}_j + \sum_{i=1}^{l} \xi(\boldsymbol{w}_j^T \boldsymbol{x}_i, [\boldsymbol{y}_i]_j), \quad (1)$$

for $j = 1, \ldots, m$. In each subproblem, λ is the hyper-parameter, $[\boldsymbol{a}]_j$ denotes the jth component of the vector $\boldsymbol{a}$, and ξ is the loss function for binary classification. Moreover, subproblems in (1) can be easily handled by some ma-

ture binary classification libraries such as LIBLINEAR (Fan et al., 2008). After the training procedure, we can use the OVR model to get the scores $\begin{bmatrix} \boldsymbol{w}_1^T\boldsymbol{x} & \cdots & \boldsymbol{w}_m^T\boldsymbol{x} \end{bmatrix}$ for any given instance $\boldsymbol{x}$. Moreover, a 0-1 function δ can map the scores to a label vector $\begin{bmatrix} \delta(\boldsymbol{w}_1^T\boldsymbol{x}) \cdots \delta(\boldsymbol{w}_m^T\boldsymbol{x}) \end{bmatrix} \in \{0,1\}^m$ as the prediction.

Many works (Babbar and Schölkopf, 2017; Lin et al., 2023) show that OVR linear models are useful, but

(i) the space requirement for the model parameter $\boldsymbol{\theta} = (\boldsymbol{w}_1 \cdots \boldsymbol{w}_m)$ and

(ii) the training time of (1)

increase as m becomes larger. For large XML problems, the issues above become essential.

2.2 Tree-Based Methods

To overcome the training time issue (ii), past works (e.g., Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022) utilize the tree structure to reduce the training time. The structure is constructed based on recursively clustering labels by methods such as K-means. For clustering, we need label information. If such information is not directly available, some works (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022) construct the label representations by averaging all instances of label-j:

$$\frac{\sum_i [\boldsymbol{y}_i]_j \boldsymbol{x}_i}{\| \sum_i [\boldsymbol{y}_i]_j \boldsymbol{x}_i \|_2}. \tag{2}$$

Next, we discuss a two-level tree as an example. The K-means procedure divides the index set of labels $\{1, \ldots, m\}$ into K partitions $I_1, \ldots, I_K$. Then, we can train a smaller OVR model of K weight vectors by solving

$$\min_{\tilde{\boldsymbol{w}}_{\tilde{j}} \in \mathbb{R}^n} \frac{\lambda}{2} \tilde{\boldsymbol{w}}_{\tilde{j}}^T \tilde{\boldsymbol{w}}_{\tilde{j}} + \sum_{i=1}^{l} \xi(\tilde{\boldsymbol{w}}_{\tilde{j}}^T \boldsymbol{x}_i, [\boldsymbol{z}_i]_{\tilde{j}}), \tag{3}$$

for $\tilde{j} = 1, \ldots, K$. For any $\boldsymbol{x}$, the model can determine if $\boldsymbol{x}$ has pseudo-label-$\tilde{j}$, where the pseudo-label vector $\boldsymbol{z}$ is defined by

$$[\boldsymbol{z}]_{\tilde{j}} = \begin{cases} 1 & \boldsymbol{x} \text{ has any label in the partition } I_{\tilde{j}}, \\ 0 & \text{otherwise}, \end{cases}$$

for all $\tilde{j} = 1, \ldots, K$. Prabhu et al. (2018); Yu et al. (2022) point out that the model trained by (3) can estimate the probabilities

$$P(\boldsymbol{x} \text{ has pseudo-label-}\tilde{j} \mid \boldsymbol{x}, \tilde{\boldsymbol{w}}_{\tilde{j}}), \tag{4}$$

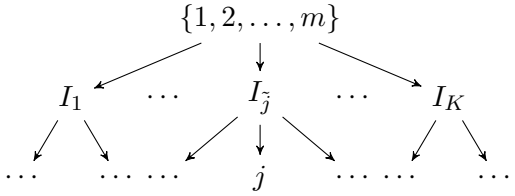


Figure 1: A two-level tree-based model.

for $\tilde{j} = 1, \ldots, K$, via the transform function

$$\sigma(\tilde{\boldsymbol{w}}_{\tilde{j}}^T \boldsymbol{x}) = \exp\left(-\max(1 - \tilde{\boldsymbol{w}}_{\tilde{j}}^T \boldsymbol{x}, 0)^2\right) \tag{5}$$

if ξ is the square-hinge loss. However, we are interested in

$$P(\boldsymbol{x} \text{ has label-}j \mid \boldsymbol{x}, \boldsymbol{\theta}), \ \forall j = 1, \ldots, m,$$

where $\boldsymbol{\theta}$ includes all the parameters (i.e., $\tilde{\boldsymbol{w}}_{\tilde{j}}$, $\boldsymbol{w}_j$, $\forall\, \tilde{j}, j$) in the model. Therefore, Prabhu et al. (2018) utilize the property that

$$\begin{aligned} & P(\boldsymbol{x} \text{ has label-}j \mid \boldsymbol{x}, \boldsymbol{\theta}) \\ = \ & P(\boldsymbol{x} \text{ has label-}j, j \in I_{\tilde{j}} \\ & \quad \mid \boldsymbol{x}, \boldsymbol{w}_j, \boldsymbol{x} \text{ has pseudo-label-}\tilde{j}\,) \cdot (4), \end{aligned} \tag{6}$$

so we can rely on a data subset

$$D_{\tilde{j}} = \{(\boldsymbol{y}_i, \boldsymbol{x}_i) \mid \boldsymbol{x}_i \text{ has any label in } I_{\tilde{j}}\} \tag{7}$$

to train another OVR model

$$\min_{\boldsymbol{w}_j \in \mathbb{R}^n} \frac{\lambda}{2} \boldsymbol{w}_j^T \boldsymbol{w}_j + \sum_{(\boldsymbol{y}_i, \boldsymbol{x}_i) \in D_{\tilde{j}}} \xi(\boldsymbol{w}_j^T \boldsymbol{x}_i, [\boldsymbol{y}_i]_j), \tag{8}$$

for estimating

$$\begin{aligned} & P(\boldsymbol{x} \text{ has label-}j \,, j \in I_{\tilde{j}} \\ & \quad \mid \boldsymbol{x}, \boldsymbol{w}_j, \boldsymbol{x} \text{ has pseudo-label-}\tilde{j}\,), \end{aligned} \tag{9}$$

for all $j \in I_{\tilde{j}}$. The estimation of (9) corresponds to the training process at node $I_{\tilde{j}}$ in Figure 1. Specifically, at node $I_{\tilde{j}}$, we have a smaller multi-label problem with labels in $I_{\tilde{j}}$ and data points in $D_{\tilde{j}}$. We still use the OVR setting for training. Thus, we have K linear models trained by the full data set D in level-1 of the tree and m linear models trained by K smaller data subsets $D_1, \ldots, D_K$ in level-2 of the tree. Furthermore, the transform function (5) and the property (6) estimate

$$\begin{aligned} & P(\boldsymbol{x} \text{ has label-}j \mid \boldsymbol{x}, \boldsymbol{\theta}) \\ \approx & \sigma(\tilde{\boldsymbol{w}}_{\tilde{j}}^T \boldsymbol{x}) \cdot \sigma(\boldsymbol{w}_j^T \boldsymbol{x}), \forall j \in I_{\tilde{j}}, \end{aligned} \tag{10}$$

for all $\tilde{j} = 1, \ldots, K$, for predicting all labels with a given $\boldsymbol{x}$. Note that for simplicity, we only show a two-level example here for describing the tree-based model.

For the training time of linear models, a popular method (Hsieh et al., 2008) costs $O(\dot{n}l)$ per iteration, where $\dot{n}$ is the average non-zeros over all instances. Because the number of iterations is usually not large, we can treat it as a constant in the time complexity analysis. Therefore, training the standard OVR that involves m binary problems costs $O(m\dot{n}l)$. For a two-level tree-based model, in Appendix C, we derive the training time as

$$O\left(Km\tilde{c}\dot{n}R + (K + \frac{cm}{K})\dot{n}l\right), \qquad (11)$$

in which the first term corresponds to the cost of K-means. In (11), $\tilde{c}\dot{n}$ with $\tilde{c} > 1$ is the average number of non-zeros in each label representation, $c \geq 1$ is a constant upper-bounded by the maximal label number of an instance, and R is the average number of K-means iterations. For data with many instances, we generally have

$$l \gg KR\tilde{c},$$

so the second part in (11) is the dominant term. If we compare it with the $O(m\dot{n}l)$ cost of OVR, when m is enormous, a tree-based model costs much less.

We have discussed a tree-based model to solve the training time issue (ii). Let us check the model size. To begin, we assume that zero features have been removed before training any binary problem. Due to the use of l_2 regularization (i.e., $\boldsymbol{w}_j^T \boldsymbol{w}_j$ in (1)), the resulting $\boldsymbol{w}_j$ is generally a dense vector, including many non-zero components. For the OVR method, we need $O(mn)$ space to store m linear models. On the other hand, the two-level tree-based model has $(K + m)$ linear models, so the model size is $O(Kn + mn)$, which is larger than the OVR model. Fortunately, while we use a training data subset (7) to train the model for estimating (8), many instances may be removed. Thus, some features may not be used by any instance of the training data subset so we can reduce the dimensionality of the feature space. Assume the reduced dimensions are $n_1, \ldots, n_K$ that correspond to the training subsets $D_1, \ldots, D_K$, and take $\bar{n} \leq n$ as the average dimension of the feature space in the level-2 models. Thus, the two-level tree-based model size becomes $O(Kn + m\bar{n})$, less than OVR's model size $O(mn)$ if $\bar{n}$ is small enough. A detailed study on the size of tree-based models is in (Lin et al., 2024).

Data set	Max. labels	Ratio of total labels
eur-lex-4k	422	0.11
wiki10-31k	3289	0.11
amazoncat-13k	2854	0.21
amazon-670k	106963	0.16
amazon-3m	352094	0.13
wiki-500k	88769	0.18

Table 1: The maximal label number of K-means partitions over six data sets as $K = 100$.

3 Distributed Settings to Address the Memory Issue

Using multiple computers to store a model can reduce the memory usage in each computer. In the past, DiSMEC distributed the training and the storage of an OVR model to multiple computers. However, the setting is still not scalable to data with extremely many labels. Hence, separating a tree-based model into several computers is a possible solution because of the faster training and smaller model size than the OVR setting. Therefore, we discuss the distributed training of a tree-based model in the following subsections.

3.1 Distributing a Tree-Based Model

We discussed in Section 2.2 that a tree-based model separates the labels to the partitions $I_1, \ldots, I_K$, so distributing the sub-tree construction of these partitions is possible. Assume we have K machines. For the machine-$\tilde{j}$, to simulate the task at node $I_{\tilde{j}}$ in Figure 1, we must train a binary classifier on

$$\text{data with labels in } I_{\tilde{j}} \text{ versus without.} \qquad (12)$$

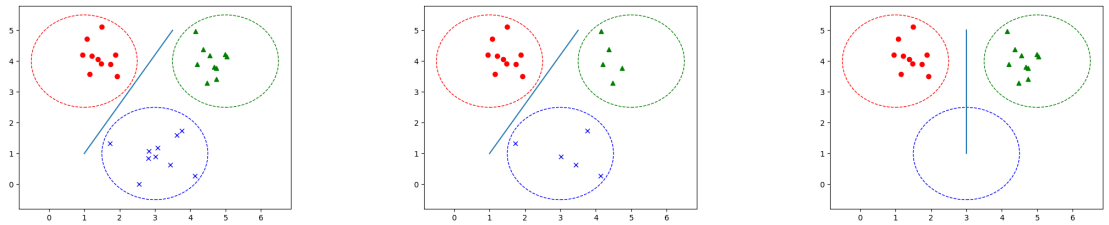
To this end, the machine-$\tilde{j}$ must have the whole training set. After that, the machine-$\tilde{j}$ continues to construct the sub-tree. Therefore, the performance will be the same as the serial setting. However, there are two issues:

(i) Unless we implement distributed K-means, the K-means procedure must be done on one computer to get the partitions.

(ii) The partitions are imbalanced. Table 1 shows that the largest partition among the 100 clusters $I_1, \ldots, I_{100}$ contains 10% or more of the whole data set. Thus, the model sizes of the partitions are hard to estimate, so the specifications of each computer are challenging to decide.

3.2 Random Label Forests

One possible solution to overcome these two issues is to omit K-means, uniformly split labels to



(a) Without negative sampling. (b) Negative sampling by instances. (c) Negative sampling by labels.

Figure 2: An example of different negative samplings.

the partitions $I_1, \ldots, I_K$ in level-1, and everything else is the same. However, our experiment in Section 4.1 shows that the performance of a tree-based model using K-means partitions is better than using random partitions. The inferior performance seems to be from the poor estimations of level-1 probabilities in (4). We know that each label subset should contain instances with similar or even identical feature values. For a random split of labels, these similar instances may end up being on both positive and negative sides of the problem (12), an ambiguous situation that may result in a poor model. In contrast, K-means helps to put these labels into the same partition, so the issue may not occur. From the discussion so far, the question becomes how to alleviate the performance issue while controlling the model size per computer. We propose the following settings.

- We let each machine handle a random label subset. This way allows us to control the model size.
- We propose bypassing the level-1 probability estimate. Instead, we run the standard tree-based method on the label subset, applying K-means on every level.

For the random label subset in each computer, instead of one partition from splitting the whole label space, we can be general so that label subsets overlap. The remaining task is to let each machine produce a suitable probability estimate and ensemble results from different machines. Because each label subset corresponds to an independent tree, our method is an ensemble method with label subsampling. We call our method "random label forests" due to the similar idea from random forests (Breiman, 2001).

Let us formally discuss random label forests in detail. Suppose we have N computers and use the sample rate r on subsampling the labels $\{1, \ldots, m\}$ to the subsets $\hat{I}_t$, for all $t = 1, \ldots, N$. Without loss of generality, let us focus on $t = 1$ as an example. Since the label part of the data set has been subsampled, the label space is modified from $\{0,1\}^m$ to $\{0,1\}^{|\hat{I}_1|}$, where we use a function ϕ_1 to describe this label mapping. Our training data subset becomes

$$\hat{D}_1 = \{(\phi_1(\boldsymbol{y}_i), \boldsymbol{x}_i) \mid \ i = 1, \ldots, l\}$$

in the computer-1. We mention that the smaller the sample rate r we set, the more instances are empty-labeled in $\hat{D}_1$, i.e.,

$$\phi_1(\boldsymbol{y}) = \mathbf{0} \in \{0,1\}^{|\hat{I}_1|}.$$

Thus, it seems that we have a choice of removing empty-labeled instances.

The training time of the models can be reduced if we remove the empty-labeled instances. However, the probability estimates from (4) may become inaccurate. Let us explain this issue by an example.

- Consider a multi-class problem, a particular case of multi-label problems, with three labels {red circle, green triangle, blue cross} in Figure 2a. If the red circle is the positive label and the others are negative labels, a linear model can be trained as the dark blue line in Figure 2a.
- If we uniformly sample the negative instances in Figure 2b, the linear model may not be affected. However, suppose we sample the negative instances by the labels in Figure 2c. In that case, the linear model can be impacted incorrectly. The reason is that we completely remove the data from some labels.
- Thus, a non-uniform negative sampling can affect (4) because the model does not estimate the probability well anymore.

The example above shows a critical point in a tree-based model training on a subset of labels. If we train a two-level tree-based model on the data subset $\hat{D}_1$, the training problem in level-1 will be

Algorithm 1 Training random label forests.

Require: Training set D, # of submodels N, sample rate r
distributed for $t = 1, \ldots, N$ **do**
  $\hat{I}_t \leftarrow$ subsample the label indices with the rate r on the full index set $\{1, \ldots, m\}$.
  $\hat{D}_t \leftarrow$ update the label part of D with $\hat{I}_t$.
  $\boldsymbol{\theta}_t \leftarrow$ train a tree-based submodel with the subset $\hat{D}_t$.
end distributed for

changed from (3) to

$$\min_{\tilde{\boldsymbol{w}}_{\tilde{j}} \in \mathbb{R}^n} \frac{\lambda}{2} \tilde{\boldsymbol{w}}_{\tilde{j}}^T \tilde{\boldsymbol{w}}_{\tilde{j}} + \sum_{i=1}^{l} \hat{\xi}(\tilde{\boldsymbol{w}}_{\tilde{j}}^T \boldsymbol{x}_i, [\hat{\boldsymbol{z}}_i]_{\tilde{j}}), \quad (13)$$

for $\tilde{j} = 1, \ldots, K$. In contrast to $\boldsymbol{z}$ in (3), a pseudo-label vector $\hat{\boldsymbol{z}}$ of the data subset $\hat{D}_1$ is defined as

$$[\hat{\boldsymbol{z}}]_{\tilde{j}} = \begin{cases} 1 & \boldsymbol{x} \text{ has any label in } I_{\tilde{j}} \text{ of } \hat{D}_1, \\ 0 & \text{otherwise.} \end{cases}$$

Many $\hat{\boldsymbol{z}}_i = \mathbf{0}$ if $\boldsymbol{x}_i$'s labels do not appear in the set $\hat{I}_1$. If these $\boldsymbol{x}_i$ are removed, it is similar to doing a non-uniform negative sampling, as in Figure 2c. The model trained by (13) may not estimate the probability (4) reasonably. Therefore, we should not remove those empty-labeled instances in $\hat{D}_1$. Besides this crucial point, all other details in the tree construction are the same as those shown in Section 2.2 for the tree using all labels.

With the label subsets $\hat{I}_1, \ldots, \hat{I}_N$ and the training subsets $\hat{D}_1, \ldots, \hat{D}_N$, we can parallelly train the submodels $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ in N computers. Algorithm 1 shows the whole training procedure.

Next, let us discuss the prediction procedure. We still assume that two-level trees are used. Since the prediction probabilities can be estimated by (10), we can estimate $P(\boldsymbol{x} \text{ has label-}j \mid \boldsymbol{x}, \boldsymbol{\theta}_t)$ for all label-j in the subsampled subset $\hat{I}_t$ by the tth tree-based submodel $\boldsymbol{\theta}_t$. Because label-j may appear in several label subsets, a natural setting is to average the several probability estimations.

$$\begin{aligned} & P(\boldsymbol{x} \text{ has label-}j \mid \boldsymbol{x}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N) \\ = & \frac{\sum_{t:j \in \hat{I}_t} P(\boldsymbol{x} \text{ has label-}j \mid \boldsymbol{x}, \boldsymbol{\theta}_t)}{|\{t \mid j \in \hat{I}_t,\ \forall t = 1, \ldots, N\}|}. \end{aligned} \quad (14)$$

Nevertheless, an issue exists because a tree-based submodel $\boldsymbol{\theta}_t$ can only predict labels in the subsampled subset $\hat{I}_t$. Hence, if some labels are never subsampled, our model can never predict those labels. This situation occurs for some rare labels, so the performance may not be affected much. Our competitive performance, as shown in Section 4.2, seems to support this point. However, other ways of label subsampling can be a future study to mitigate the issue.

3.3 The Benefits of Random Label Forests

Section 2.2 discusses the time complexity and model size of a two-level tree-based model. Now, let us check the complexities in a computer when applying random label forests with two-level tree-based submodels in the distributed system.

- Space complexity. Since we set the sample rate as r, the number of a subsampled label set becomes rm. Therefore, the model size changes from $O(Kn + m\bar{n})$ to $O(Kn + rm\bar{n})$.
- Time complexity. Similarly, the time complexity changes from (11) to

$$O\left(Krm\tilde{c}\dot{n}R + (K + \frac{crm}{K})\dot{n}l\right).$$

Therefore, we can roughly control the model size and training time by the rate r in random label forests if each computer handles a tree-based submodel. The experiments for comparing a tree-based model with all labels and a tree-based submodel of random label forests are discussed in Section 4.3 and Section 4.4. We note that Yu et al. (2022) try to parallelize the training of a single tree with all labels, but the implementation is complicated. Moreover, the time complexity of K-means cannot be reduced in that scenario.

4 Experiments

Section 3.2 discusses different distributed settings on tree-based models, so we first analyze which setting is the better choice over four smaller XML data sets in Section 4.1. After deciding on the distributed setting, we compare three linear models: OVR, a tree-based model with all labels, and random label forests on six XML data sets in Section 4.2.

Throughout this section, the label number m is shown after the name in each data set.

We leave the details of data sets in Appendix A and discuss the hyper-parameters in Appendix B. To measure the performance of an XML model, we follow the works (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022; You et al., 2019; Jiang et al., 2021) to use precision at 1, 3, and 5 as the metrics of the predictions.

We discuss the model size in Section 4.3, and the comparison of the training time is in Section 4.4.

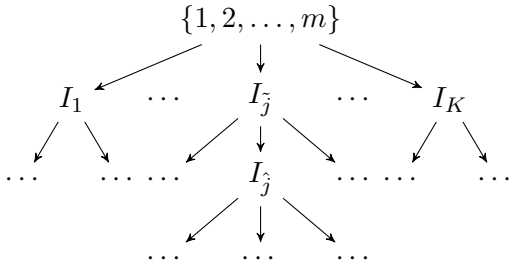
	Random partitions/selection	Random label forests
level-1	data with $I_{\tilde{j}}$ versus without	nothing
level-2	data with $I_{\hat{j}}$ versus those with $I_{\tilde{j}}$ but not $I_{\hat{j}}$	data with $I_{\hat{j}}$ versus without

Table 2: Comparison between random partitions/selection and random label forests.

4.1 Analysis of Different Distributed Settings

Section 3.2 discusses that using a uniform split of labels is a possible solution for avoiding the imbalanced K-means clusters under the distributed setting. Now, we check the performance between these two partition methods. They follow the standard setting of tree-based models but intend to continue the tree construction after level-1 in parallel. On the other hand, we have the proposed random label forests, which independently generate tree-based submodels on label subsets. Thus, we compare these three methods with the following settings.

- **Tree with all labels**. The standard tree-based method. Note that we set $K = 100$.
- **Random 100 partitions**. The tree-based method that replaces K-means in level-1 with a uniform split of labels.
- **Random 10 partitions** and **Random 10 partitions$\times$10**. In the previous setting, we consider 100 random partitions because of following the $K = 100$ in K-means. While K-means may require a careful selection of K, when using random splits, we can instead control each cluster's size according to the capacity of a machine. Thus, we try 10 partitions of the data set. Another reason for doing so is that later, for the proposed random label forests, we let each machine handle 10% of labels. Thus, we need the setting of 10 random partitions as a comparison.
  In Section 3.2, we discussed how to ensemble the prediction results of various trees. Therefore, an extension is to run the label partition several times to generate more trees. Here, we run the random 10 partitions 10 times to generate 100 label subsets. We call this setting "random 10 partitions$\times$10."
- **Random label forests-10P** and **Random label forests-10P$\times$10**. We consider label subsets generated in the previous setting to run the proposed label forests. Note that the current setting has some subtle differences from the previous one. To illustrate the differences, we extend Figure 1 to a tree with more than two levels.



Now, $I_1, \ldots, I_K$ correspond to the 10 subsets from a random partition. We can use Table 2 to show the different binary problems solved at each level. Note that "random label forests-10P" is the standard setting for training a tree model, though we run the procedure related to $I_{\tilde{j}}$ on one particular computer. In the prediction of random 10 partitions, we use the model obtained in level-1 to estimate the probability in (4). In contrast, for "random label forests-10P," we can say that there is no level-1. Using $I_{\tilde{j}}$ and all data, we construct an independent tree. It is important to note that, as explained in Section 3.2, for the binary problem involving those with $I_{\tilde{j}}$ as positive, we need all other data as negative. We cannot just consider those in $I_{\tilde{j}}$ but not $I_{\hat{j}}$; see the row "level-2" in Table 2. In prediction, each tree obtains its own probability estimation, and we calculate the average in (14).

- **Random label forests-100U** and **Random selection-100**. The discussion in Table 2 shows that for any label subset $I_{\tilde{j}}$ considered, there are two ways to continually obtain a sub-tree model. In the previous setting, $I_{\tilde{j}}$ is one partition of a random split, but we can uniformly sample labels from $\{1, \ldots, m\}$ to have label subsets. We randomly draw labels with replacements to obtain 100 label subsets, each of which has 10% of the labels. Thus, these subsets overlap with each other. Random label forests-100U is then the proposed method applied to these 100 subsets. The approach differs from random 10 partitions$\times$10 only on how label subsets are generated. Specifically, earlier, for random 10 partitions$\times$10, we run label split 10 times, each of which partitions the label set to 10 subsets. On the other hand, for the same 100 label subsets, we can apply the

Method	P@1	P@3	P@5	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
	eur-lex-4k			wiki10-31k		amazoncat-13k			amazon-670k		
Tree with all labels	82.29	69.35	57.91	74.72	65.86	93.19	79.55	64.61	44.58	39.44	35.64
Random 100 partitions	80.52	67.56	56.30	73.02	63.82	92.56	77.90	62.76	38.52	32.49	27.90
Random 10 partitions	79.82	66.60	55.84	73.61	64.55	92.36	78.06	63.11	41.76	36.43	32.30
Random label forests-10P	77.52	65.08	54.87	73.35	64.56	81.06	71.91	60.35	42.62	37.63	34.01
Random 10 partitions$\times$10	81.99	69.13	58.19	74.21	65.21	94.01	79.96	65.09	44.82	39.92	36.25
Random label forests-10P$\times$10	81.22	68.15	57.29	74.13	65.36	93.08	79.59	64.78	45.23	40.28	36.70
Random selection-100	80.51	67.93	57.37	74.22	65.65	89.41	77.82	63.97	43.76	39.35	35.86
Random label forests-100U	83.08	69.90	58.69	74.35	65.70	94.11	80.17	65.18	45.19	40.24	36.65

Table 3: Comparison of different distributed settings in precisions. For wiki10-31k, since half of the instances are associated with a unique label, precision at 1 is not a discriminable metric on this data set. Therefore, we do not show the precision at 1 results in wiki10-31k for the space limitation.

Method	P@1	P@3	P@5	P@1	P@3	P@5
	eur-lex-4k			amazon-670k		
One-versus-rest	83.47	70.62	59.05	45.41	40.41	36.97
Tree with all labels	82.29 $\pm$ 0.30	69.35 $\pm$ 0.09	57.91 $\pm$ 0.12	44.58 $\pm$ 0.07	39.44 $\pm$ 0.04	35.64 $\pm$ 0.03
Random label forests	83.08 $\pm$ 0.15	69.90 $\pm$ 0.08	58.69 $\pm$ 0.06	45.19 $\pm$ 0.05	40.24 $\pm$ 0.03	36.65 $\pm$ 0.01
	wiki10-31k			amazon-3m		
One-versus-rest	85.23	75.80	67.11	-	-	-
Tree with all labels	84.72 $\pm$ 0.08	74.72 $\pm$ 0.17	65.86 $\pm$ 0.09	47.48	44.74	42.63
Random label forests	84.78 $\pm$ 0.14	74.35 $\pm$ 0.17	65.70 $\pm$ 0.08	48.69	45.67	43.49
	amazoncat-13k			wiki-500k		
One-versus-rest	94.14	79.71	64.69	-	-	-
Tree with all labels	93.19 $\pm$ 0.03	79.55 $\pm$ 0.04	64.61 $\pm$ 0.03	68.39	48.90	38.00
Random label forests	94.11 $\pm$ 0.04	80.17 $\pm$ 0.02	65.18 $\pm$ 0.02	64.37	45.83	36.09

Table 4: Comparison of random label forests and other linear methods in precision at 1, 3, and 5 over six data sets. OVR results on "amazon-3m" and "wiki-500k" are unavailable due to lengthy running time.

same strategy as "random label forests-100" in Table 2. We call this setting as "random selection-100."

We have the following observations from Table 3.

- A comparison between "tree with all labels" and "random 100 partitions" shows that using K-means for the label partitions is better than using random partitions. We explained this observation at the beginning of Section 3.2.
- The ensemble method enhances the performance. This result can be seen by comparing
  - "random 10 partitions" and "random 10 partitions$\times$10," and
  - "random label forests-10P" and "random label forests-10P$\times$10."
- In Table 2, we illustrated an important difference between random partitions/selections and the proposed random label forests. We organize Table 3 in a way so that, except for the first two rows, every two rows serve as a comparison between the two settings. Unfortunately, results do not clearly show which way is better. We can see that "random label forests-10P" is significantly worse than "random 10 partitions." However, once we have more trees, random label forests become comparable or better.
- Regarding the generation of label subsets, we mentioned two ways: random partitions and random selections. To see which way is more effective, we should compare
  - "random 10 partitions$\times$10" and "random selection-100," and
  - "random label forests-10P$\times$10" and "random label forests-100U."

  Results seem to indicate that, if one calculates the level-1 probabilities as explained in Table 2, random partitions are better than random selection. However, opposite results occur for random label forests.

Experiments in this section fully show that distributing a tree-based method is not trivial because of different considerations and options. Because the setting "random label forests-100U" gives the best performance, we use it as our distributed solution in subsequent experiments.

4.2 Performance Comparison with Existing Multi-Label Methods

To mitigate randomization issues in tree-based methods due to K-means, we execute training and prediction procedures ten times on the data sets "eur-lex-4k," "wiki10-31k," "amazoncat-13k," and

Data set	Tree with all labels	Random label forests
eur-lex-4k	561.02 MB	49.25 MB
amazoncat-13k	1.72 GB	193.13 MB
wiki10-31k	6.38 GB	749.07 MB
amazon-670k	20.56 GB	2.92 GB
amazon-3m	135.89 GB	12.69 GB
wiki-500k	161.37 GB	13.65 GB

Table 5: The model size comparison between a tree with all labels and a tree of random label forests.

Data set	Tree with all labels	Random label forests
eur-lex-4k	224.42 s	32.18 s
wiki10-31k	4220.77 s	543.33 s
amazoncat-13k	5311.23 s	910.06 s
amazon-670k	32068.89 s	2120.25 s
wiki-500k	202261.66 s	22987.20 s
amazon-3m	503575.79 s	23023.76 s

Table 6: Training time comparison between a tree with all labels and a tree of random label forests.

"amazon-670k." We only execute the procedures once for the data sets "amazon-3m" and "wiki-500k" because the training time is too long. We compare the following methods in the rest of this section.

- OVR: The standard one-vs-rest method. This method, though not scalable even in a distributed setting, its performance on small data sets serves as the target to be achieved by tree-based methods.
- Tree with all labels: This standard tree-based method was evaluated in Section 4.1. Before our distributed extensions studied in Section 4.1, this method was mainly run on a single computer. Thus, we compare this method to see the effectiveness of our proposed distributed setting.
- Random label forests: This is the "random label forests-100U" setting in Section 4.1.

Table 4 shows the comparison results, and we have the following observations.

- OVR is slightly better than tree-based methods. This result is reasonable because tree-based methods are a kind of "hierarchical approximation" of OVR to address the scalability issue.
- The proposed random label forests are consistently better than the tree-based method with all labels except for "wiki-500k." The performance is close to that of OVR.

The worse results on "wiki-500k" are an example of our method's limitations, and we discuss this issue in Section 6.1.

4.3 Comparison in Model Size

In Section 3.3, we have analyzed the model size of random label forests that use two-level tree-based submodels. However, because

- the sparsity of data points can affect $\bar{n}$, and
- a tree-based model has more than two layers,

we conduct experiments to check the model size in practice. Table 5 compares

- the model size of a tree-based method with all labels in a single computer, and
- the model size of a tree in random label forests, which corresponds to the needed space in each computer of the distributed environment.

We can see that the ratio of space reduction is approximately close to the sample rate $r = 0.1$. Hence, random label forests can reduce the model size in each computer of the distributed system, even though the whole model may be larger than that of a tree-based model with all labels. Therefore, random label forests are effective in addressing the memory difficulty of extreme multi-label classification.

4.4 Comparison in Training Time

Table 6 shows the training time comparison between a tree-based model with all labels and a tree of random label forests. If, for the random label forests,

- each machine handles one tree-based submodel, and
- all machines used have similar configurations,

then Table 6 gives the comparison of total running time between the standard tree-based method run in one computer and the proposed random label forests run in a distributed environment. We observe significant time reduction by random label forests, especially for problems with a large number of labels.

5 Conclusion

This work proposes random label forests, a distributed ensemble method with label subsampling, and tree-based linear models as the backbone. Random label forests give competitive performances using much less training time and memory usage in a machine. Hence, handling a problem with extremely many labels becomes practical. For all methods considered and evaluated in this work, the backbone is a linear classifier. In the future, we plan to consider more sophisticated techniques, such as neural networks.

6 Limitations

6.1 Breaking Label Relationships

In Section 4.2, the performance of random label forests is worse than that of a tree-based method with all labels over "wiki-500k" data set. After the investigation, we think the hierarchy correlation between the labels in "wiki-500k" may be higher than other data sets because the labels are the tags of the documents in Wikipedia. For example, the labels of the 14th instance in the raw data of "wiki-500k" are

> 'Apollo,' 'Arts gods,' 'Deities in the Iliad,' 'Dragonslayers,' 'Health gods,' 'Knowledge gods,' 'LGBT history in Greece,' 'LGBT themes in mythology,' 'Muses,' 'Mythological Greek archers,' 'Mythological rapists,' 'Oracular gods,' 'Roman gods,' 'Solar gods,' 'Temples of Apollo,'

and there are many hierarchy relationships:

$$\text{Roman gods} \rightarrow \text{Apollo},$$
$$\text{Solar gods} \rightarrow \text{Apollo}, \ldots, \text{etc.}$$

Thus, uniform sampling in labels breaks the relations, so the performance may be hurt. If we increase the sample rate from 0.1 to 0.15, the performance of "wiki-500k" will become better to

$P@1$	$P@3$	$P@5$
65.45	46.84	36.92

However, that performance is still much lower than the tree-based model with all labels. Therefore, the label subsampling technique seems unsuitable for the data sets that include many hierarchy-correlation labels.

6.2 Requiring Distributed Resourse

In Section 4.2 and Section 4.4, we show that a tree of random label forests can use much less memory and training time than a tree-based model with all labels. However, random label forests require N tree-based submodels as an ensemble method. Hence, those benefits will be discounted if we do not have N machines.

6.3 The Applications of Distributed Ensembles Method with Label Subsampling

We only show an example, random label forests, of distributed ensemble methods with label subsampling and analyze the time complexity and model size. However, the conclusion may be different when using neural networks as the submodel. Hence, searching for more applications is vital if distributed ensemble methods with label subsampling are a general solution for reducing the memory usage of an XML method in a computer.

References

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 *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 13–24.

Rohit Babbar and Bernhard Schölkopf. 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)*, pages 721–729.

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.

Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems 28*, pages 730–738.

Leo Breiman. 2001. Random forests. *Machine Learning*, 45(1):5–32.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. 2008. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):7987–7994.

Sujay Khandagale, Han Xiao, and Rohit Babbar. 2020. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning*, 109:2099–2119.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

He-Zhe Lin, Cheng-Hung Liu, and Chih-Jen Lin. 2024. Exploring space efficiency in a tree-based linear model for extreme multi-label classification. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yu-Chen Lin, Si-An Chen, Jie-Jyun Liu, and Chih-Jen Lin. 2023. Linear classifier: An often-forgotten baseline for text classification. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1876–1888. Short paper.

Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender Systems*, pages 165–172.

Siddhartha Nuthakki, Sunil Neela, Judy W. Gichoya, and Saptarshi Purkayastha. 2019. Natural language processing of mimic-iii clinical notes for identifying diagnosis and procedures with neural networks. *Preprint*, arXiv:1912.12397.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

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 *Proceedings of the 2018 World Wide Web Conference (WWW)*, pages 993–1002.

Dan Shen, Jean David Ruvini, Manas Somaiya, and Neel Sundaresan. 2011. Item categorization in the e-commerce domain. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 1921–1924, New York, NY, USA. ACM.

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

Grigorios Tsoumakas and Ioannis Vlahavas. 2007. Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning*, pages 406–417.

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 *Advances in Neural Information Processing Systems*, volume 32.

Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit S. Dhillon. 2014. Large-scale multi-label learning with missing labels. In *Proceedings of the Thirty First International Conference on Machine Learning (ICML)*, pages 593–601.

Hsiang-Fu Yu, Kai Zhong, Jiong Zhang, Wei-Cheng Chang, and Inderjit S. Dhillon. 2022. PECOS: Prediction for enormous and correlated output spaces. *Journal of Machine Learning Research*, 23(98):1–32.

Jiong Zhang, Wei-Cheng Chang, Hsiang-Fu Yu, and Inderjit S. Dhillon. 2021. Fast multi-resolution transformer fine-tuning for extreme multi-label text classification. 34:7267–7280.

Arkaitz Zubiaga. 2009. Enhancing navigation on Wikipedia with social tags. In *Proceedings of Wikimania*.

Data Set	l instances	n features	m labels
eur-lex-4k	15,449	186,104	3,956
wiki10-31k	14,146	104,374	30,938
amazoncat-13k	1,186,239	203,882	13,330
amazon-670k	490,449	135,909	670,091
amazon-3m	1,717,899	337,067	2,812,281
wiki-500k	1,779,881	2,381,304	501,070

Table 7: Statistics of data sets

A Data Sets

We show the statistics of data sets in Table 7. The sets "eur-lex-4k," "wiki10-31k," and "amazoncat-13k" are downloaded from "LIBSVM Data: Multi-label Classification[1]". The sets "amazon-670k," "amazon-3m," and "wiki-500k" are downloaded from the link that is supported by You et al. (2019). Note that those data sets have already been preprocessed from documents to a popular sparse feature representation, "TF-IDF." Moreover, every data set has further been split into training and test parts.

B Hyper-Parameter Setting

In an XML problem, many works (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022; You et al., 2019; Jiang et al., 2021; Zhang et al., 2021) only fix a group of reasonable hyper-parameters on their models because splitting a validation set from a training set is a complex issue. The main reason comes from the long-tail distribution of data over the labels. If we uniformly split a validation set from a training set, the distribution of the validation set is usually much different from the training set in most rare labels, implying that tuning the hyper-parameters in this validation set is unsuitable. Hence, we follow those works (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022; You et al., 2019; Jiang et al., 2021; Zhang et al., 2021) to set the commonly used hyper-parameters to keep our results be credible.

In our experiments, we utilize the library LibMultiLabel[2] to handle all of the model training and evaluation. Moreover, the linear classifiers are trained by LIBLINEAR (Fan et al., 2008) in LibMultiLabel. For the detail settings in LIBLINEAR, we consider the settings from the works (Khandagale et al., 2020; Yu et al., 2022):

- using squared hinge loss (L2-SVM) and
- taking $\lambda = 1$ in the training problems.

In the implementations of (Khandagale et al., 2020; Yu et al., 2022), they decided to stop the training process early in each linear model training. However, we chose to spend more time in the model training to get a tight solution that is closer to the optimal solution of the training problem. The tree-based methods in LibMultiLabel follow the implementation[3] in Khandagale et al. (2020), and we consider the following hyper-parameters

- $K = 100$ for K-means and the max depth is 10.

For the random label forests, we consider uniform sampling with the rate of 0.1 on the labels because reducing the model size by around 90% is a practical scenario. Moreover, we consider the default setting of random forests from scikit-learn (Pedregosa et al., 2011):

training 100 tree-based submodels

for our ensemble method.

C Time Complexity of Two-Level Tree-Based Models

Because the level-1 of a two-level tree-based model is a smaller OVR model, the level-1 only costs $O(K\dot{n}l)$. For the level-2, since the instances can belong to several label partitions, we assume that the average number of instances in each of the subsets $D_1, \ldots, D_K$ is

$$\frac{cl}{K},$$

where $c \geq 1$ is a small positive number. Moreover, c is bounded by the maximal label number of an instance[4]. Hence, taking the training cost of a subset in level-2 as

$$O\left(\frac{\dot{n}cl}{K}\right)$$

is a reasonable assumption in the linear model setting, and the two-level tree-based model then costs

$$O\left((K + \frac{cm}{K})\dot{n}l\right),$$

which is different from the complexity of OVR $O(m\dot{n}l)$. Besides the training time of linear models, we must check the cost of running K-means. The process involves several iterations, in each of

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html

[2] https://www.csie.ntu.edu.tw/~cjlin/libmultilabel

[3] The work (Khandagale et al., 2020) ensembles the predictions of three tree-based models in their experiments, but LibMultiLabel only considers the single tree-based model.

[4] Prabhu et al. (2018) assume $O(\log(m))$ is the maximal label number of an instance, so c is bounded by $O(\log(m))$.

which we calculate the distance between each label representation (2) and K centers of the current clusters. If the label representations are still sparse and the average number of non-zeros is $\tilde{c}\dot{n}$, where $\tilde{c} > 1$ is a positive constant, checking the distance requires

$$O(\tilde{c}\dot{n}).$$

Thereby, one iteration of K-means requires

$$O(Km\tilde{c}\dot{n}).$$

We usually set a constant R as the maximum iteration, so the time complexity of K-means is

$$O(Km\tilde{c}\dot{n}R).$$

Hence, the total training time of a two-level tree-based model is

$$O\left(Km\tilde{c}\dot{n}R + (K + \frac{cm}{K})\dot{n}l\right).$$