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

Anonymous ACL submission

Abstract

Text classification is one of the essential topics in natural language process fields, and the prediction in text classification can usually be multiple labels. Thus, a text classification problem can be a particular problem in machine learning: a multi-label classification problem. 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 more and more memory space in many existing multi-label learning methods. Hence, utilizing distributed system to share those large memory requirement is a reasonable solution. We propose "random label forests", a distributed ensemble method with label subsampling, for handling extremely large-scale labels by label subsampling and parallel computing. Random label forests can reduce the memory usage per computer while keeping competitive performances over six real-world data sets.

1 Introduction

Text classification is one of the essential topics in natural language process 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 multiple labels, e.g., a document can be tagged as label-A and label-C. Hence, a text classification problem can be a particular problem in machine learning: a multi-label classification problem, which is to find the relevant labels of a data instance. For example, we can set the document ID, the document contents, and the taggings in the document tagging problem as the instances, features, and labels in a multi-label problem. Thus, we can utilize a multi-label classification method to handle a text classification problem.

Recently, the number of labels has become larger and larger, especially in the applications of e-commerce. Thus, a particular topic has been discussed recently: extreme multi-label learning (XML), which focuses on large-scale candidate labels, input instances, and input features. Because of these three large-scale things, an XML method should further consider the model training time and memory usage, besides the performance.

In the earlier years, a multi-label classification problem could be handled by the one-versus-rest (OVR) method with linear models, but the time complexity and model size directly depended on the number of labels and features. Although the OVR method is not a good solution for a millions-label XML problem, there are two branches to use OVR on the XML problem:

- DiSMEC (Babbar and Schölkopf, 2017) separates the labels into many machines and utilizes the distributed system to reduce the training time and model size per machine.
- 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 conquer the smaller problem on the clusters. The details are discussed 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), but some works (Khandagale et al., 2020; Babbar and Schölkopf, 2017) report that the long-tail distribution of positive instances over labels, which is an essential phenomenon in most XML data sets, causes 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. Furthermore, the linear models are easy to understand and more explainable, so we focus on linear models in this work.

Let us back to discuss the linear methods. Although DiSMEC and tree-based linear methods are good solutions for XML problems, each method still has disadvantages. DiSMEC can only handle a small number of labels, but not large-scale labels in each machine, and tree-based linear methods require huge memory space during the training. Hence, we hope a method can take advantage of DiSMEC and tree-based methods without taking their disadvantages.

Label subsampling is another way to divide an XML problem into many small-scale subproblems. RAKEL (Tsoumakas and Vlahavas, 2007) is a pioneer in using label subsampling with the ensemble method. After the label subsampling, RAKEL converts the smaller multi-label problem 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 extremely large-scale labels by the powerset method is almost impossible. The difference between our method and RAKEL is discussed in Appendix A.

In this work, we utilize label subsampling and the distributed system to reduce the impact of the large-scale labels. For example, using many smaller XML subproblems with subsampling in labels can recover an extremely large-scale XML problem, and 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. Moreover, 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 in the following:

- We propose a native paralleled framework, random label forests, an ensemble method with label subsampling for the XML problem.
- We explain the importance of negative instances in random label forests.
- Our experiments show that random label forests are competitive with the tree-based methods with all labels in many XML data sets.
- 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 much faster than the tree-based methods with all labels if computers are enough.

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 how to distribute a tree-based model and then presents random label forests, including discussions on the data processing issue, time complexity, and model size. We show the comparisons between several linear methods in Section 4 for the experiments. We further give the training time and model size tables for a tree-based method with all labels and random label forests.

2 Multi-Label Problems

Generally, 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 and 1 to represent the relation between an instance and a label, where $0/1$ means 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 with $\boldsymbol{y}$ as possible. By the aforementioned definition, those 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 one-versus-rest (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 data 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, these subproblems in (1) can be easily handled by some mature binary classification libraries such as LIBLINEAR (Fan et al., 2008). After the training, with a given instance $\boldsymbol{x}$, we can give $\boldsymbol{x}$ the scores

$$f^{\text{OVR}}(\boldsymbol{x}; \boldsymbol{w}_1 \cdots \boldsymbol{w}_m) = \begin{bmatrix} \boldsymbol{w}_1^T \boldsymbol{x} & \cdots & \boldsymbol{w}_m^T \boldsymbol{x} \end{bmatrix}$$

to each label by the OVR model. Moreover, we can use a 0-1 function δ to map the scores to a label vector $\left[\delta(\boldsymbol{w}_1^T \boldsymbol{x}) \cdots \delta(\boldsymbol{w}_m^T \boldsymbol{x})\right] \in \{0,1\}^m$ as the prediction.

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

are directly increasing as m becomes larger and larger. When we consider some XML problems, the issues above become important.

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, which is recursively based on some clustering methods such as K-means method with some label information. However, the label information may be missing in some datasets, so several methods are proposed in (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022) to construct the label representations without knowing any information about labels, e.g., collecting all positive instances of label-j as the representation

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

After learning about the label information issue, let us show an example with a two-level tree. The K-means divides the index set labels $\{1, \ldots, m\}$ into K partitions $I_1, \ldots, I_K$. Then, we can train a much smaller OVR linear model

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

for $\tilde{j} = 1, \ldots, K$, to determine whether $\boldsymbol{x}$ 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

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

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

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

if ξ is square-hinge loss, but 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

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

so we can rely on a data subset

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

to train another OVR model

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

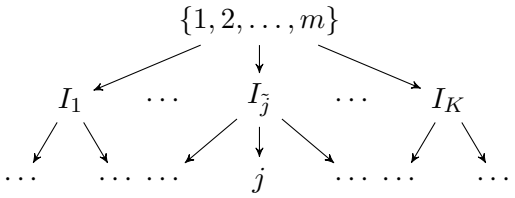


Figure 1: A two-level tree-based model.

for estimating

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

for all $j \in I_{\tilde{j}}$. The estimation of (9) corresponds to the training process at the node $I_{\tilde{j}}$ in Figure 1. Specifically, at the 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 overall 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) estimates

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

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

For the training time of linear models, a famous method (Hsieh et al., 2008) utilized several iterations for solving a binary problem and each iteration costs $O(\dot{n}l)$ if $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l$ is sparse vector, where $\dot{n}$ is the average of non-zeros over all instances. Fortunately, we usually pre-process the instances $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l$ to sparse feature representations when using linear models, so training a binary problem with the full data set D can roughly cost $O(\dot{n}l)$. Therefore, training a standard OVR model costs $O(m\dot{n}l)$. For a two-level tree-based model, we derive the training time as

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

in Appendix B, which including the cost of K-means, where $c \geq 1$ is bounded by the maximal label number of an instance, $\tilde{c} > 1$ is the average increased rate of non-zeros of label representations, and R is the maximum iteration of K-means. For data with many instances, we generally have

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

so the second part in (11) is the dommant term. If we compare with the $O(m\dot{n}l)$ cost of OVR, when m is extremely large, 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 before training any binary problem, zero features have been removed. Due to the use of l_2 regularization (i.e., $\boldsymbol{w}_j^T \boldsymbol{w}_j$ in (1)), the resulting $\boldsymbol{w}_j$ generally has non-zero components. That is, $\boldsymbol{w}_j$ is a dense vector. Now for the OVR method, we need $O(mn)$ space to store the 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, there may exist some features that are not used by any instance of a training data subset, so that we can remove some dimensions of the original feature space $\mathbb{R}^n$ in each subset. In particular, the features of the instances $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l$ are sparse in the data set. 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 $O(Kn + m\bar{n})$, which can be less than OVR's model size $O(mn)$ if $\bar{n}$ is small enough.

3 Distributed Setting for Memory Issue

Using multiple computers to contain a model can reduce the memory usage in each computer. In the past, DiSMEC was a successful work that separated the OVR model. However, training a DiSMEC model requires more and more computers as the label number m becomes larger and larger. Hence, separating a tree-based model into many computers is a practical solution for saving resources because the training speed of a tree-based model is faster than that of an OVR model. Therefore, we discuss distributing a tree-based model in the following subsections.

3.1 Distributing a Tree-Based Model

We discuss that a tree-based model separates the labels to the partitions $I_1, \ldots, I_K$ in Section 2.2, so distributing the sub-tree construction of these partitions is a possible way. Assume we have K machines. For the machine-$\tilde{j}$, to simulate the task

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 $K = 100$.

at node $I_{\tilde{j}}$ in Figure 1, we must train a binary classifier on

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

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 there is a partition from $I_1, \ldots, I_{100}$ that includes over 10% of the labels in each 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 and uniformly split labels to 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. We know that for instances with similar or even identical feature values, their label sets should be similar. For a random split of labels, these similar instances may end up with 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 to address the issue.

- In each machine, we consider a random label subset, which 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, so K-means is applied 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 how to let each machine procedure suitable probability estimate and how to 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

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

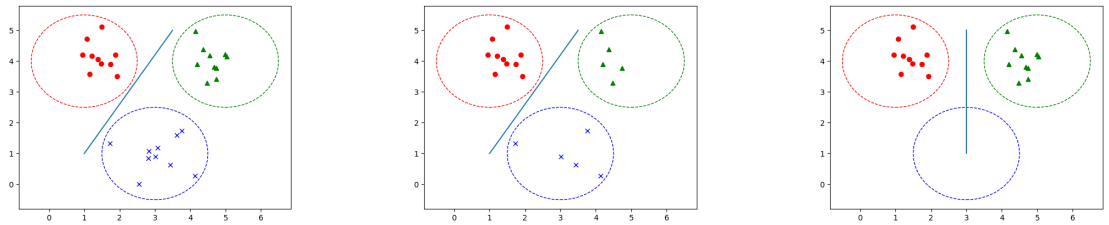
in the computer-1. We mention that the smaller sample rate r we set, the more instances have empty-labeled in $\hat{D}_1$, i.e.,

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

Thus, there is a choice of whether those empty-labeled instances should be removed.

The training time of the models can be reduced if we remove the empty-labeled instances. However, the probability estimation (4) can be inaccurate when using a tree-based model. Let us explain this issue in the following example.

- Consider a multi-class problem, which is a special 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, if we sample the negative instances by the labels in Figure 2c, the linear model can be



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

Figure 2: An example of different negative samplings.

impacted in the wrong place. 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 key 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 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

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

Clearly, 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 like that we do a non-uniform negative sampling as in Figure 2c. The model trained by (13) may not estimate the probability (4) well. 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 full 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

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

the several probability estimations.

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

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 shown in Section 4.2 seems to support this point, though other ways of label subsampling can be a future issue for investigation.

3.3 The Benefits of Random Label Forests

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

- Space complexity. Since we can 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 only contains a tree-based submodel. 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

We firstly analyze the impacts of different distributed settings in Section 3.2 over four smaller XML data sets, and discuss which setting is the better choice in Section 4.1. After deciding the distributed setting, our experiment compares three linear models: OVR, a tree-based model with all labels, random label forests on six XML data sets in Section 4.2, where the label number m is shown after the name in each data set.

We leave the details of data sets in Appendix C and discuss the hyper-parameters of the models in Appendix D. 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.

In the final, we discuss the model size in Section 4.3. Moreover, the training time comparisons are presented in Appendix E.

4.1 Analysis of Different Distributed Settings

In Section 3.2, we discuss that using random partitions without overlaps is a possible solution for avoiding the disadvantages of K-means under the distributed setting, and now we have to check the performance between these two partition methods. Besides, we propose random label forests that bypass the level-1 probability estimate of the tree-based method with random partitions. To compare those three methods reasonably, we should consider 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 with all labels that using random partitions $I_1, \ldots, I_{100}$ instead of K-means in level-1.
- "Random 10 partitions." To compare random partitions with random label forests, we should consider the exact size of the partitions. As the label sample rate of random label forests is 0.1, random partitions should only contain 10 partitions that split whole labels.
- "Random label forests-A." Similarly, we must consider the sampling type in random label forests in addition to the subset size. Thus, we uniformly split the labels to 10 partitions for producing ten submodels and re-do ten times that the number of submodels is up to 100. The differences between random label forests-A and random 10 partitions are the ensemble setting and bypassing the level-1 probability estimate.
- "Random label forests-B." Conversely, we consider the label subsets setting instead of the label partitions, so we use uniform subsampling with a rate 0.1 for producing 100 submodels. Subsampling methods usually use the uniform setting, such as feature subsampling in random forests.

We have the following observations in Table 2.

- Using K-means for the label partitions is better than using random partitions.
- A larger partition size may have stable results in random partitions.
- Random label forests-A is better than random 10 partitions, so the ensemble setting and bypassing the level-1 probability estimate are the critical points in our method.
- The performances of random label forests-B are slightly better than or similar to random label forests-A. We think that the label overlaps in the label subsampling method seem to be the connections between the submodels so that the predictions of the submodels can be more meaningful for ranking the labels.

In the final, we decided to use random label forests-B as our distributed solution, and this section shows that distributing a tree-based method is not trivial.

4.2 Comparison in Performance

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 "amazon-670k." For the data sets "amazon-3m" and "wiki-500k" because the training time is too long, we only execute the procedures once. Table 3 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 consis-

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-A	81.22	68.15	57.29	74.13	65.36	93.08	79.59	64.78	45.23	40.28	36.70
Random label forests-B	83.08	69.90	58.69	74.35	65.70	94.11	80.17	65.18	45.19	40.24	36.65

Table 2: 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, for the space limitation, we do not show the results of precision at 1 in wiki10-31k.

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 ± 0.30	69.35 ± 0.09	57.91 ± 0.12	44.58 ± 0.07	39.44 ± 0.04	35.64 ± 0.03
Random label forests	83.08 ± 0.15	69.90 ± 0.08	58.69 ± 0.06	45.19 ± 0.05	40.24 ± 0.03	36.65 ± 0.01
	wiki10-31k			amazon-3m		
One-versus-rest	85.23	75.80	67.11	-	-	-
Tree with all labels	84.72 ± 0.08	74.72 ± 0.17	65.86 ± 0.09	47.48	44.74	42.63
Random label forests	84.78 ± 0.14	74.35 ± 0.17	65.70 ± 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 ± 0.03	79.55 ± 0.04	64.61 ± 0.03	68.39	48.90	38.00
Random label forests	94.11 ± 0.04	80.17 ± 0.02	65.18 ± 0.02	64.37	45.83	36.09

Table 3: 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.

Data set	Tree with all labels	Rand. label forests
eur-lex-4k	561.02 MB	49.25 MB
wiki10-31k	6.38 GB	749.07 MB
amazoncat-13k	1.72 GB	193.13 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 4: The model size comparison between a tree with all labels and a tree of random label forests. Note that the sample rate is 0.1 in random label forests.

tently 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" is an example of the limitations in our method, 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 4 compares the model sizes between a tree-based method with all labels and a tree in random label forests after training on the data sets. We can see that the reduced ratios between full and subsampled models are indeed 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 with the distributed system deployment are a practical solution for handling an XML problem with billions of labels or more.

5 Conclusion

In this work, we propose random label forests, a distributed ensemble method with label subsampling, and tree-based linear models as the backbone, giving competitive performances using much less training time and memory usage in a machine. Hence, handling an XML problem with billions of labels or further more labels becomes practical. We discuss that removing some the negative instances will hurt the accurate of probability estimate if label subsampling is used. Furthermore, we analyze time complexity and model size in random label forests. In the future, we plan to observe other cases of the distributed ensemble method with label subsampling, such as neural network methods, so that this method can be a general solution in other XML applications, such as image classification and other natural language process problems.

6 Limitations

6.1 Breaking Label Relationships

In Section 4.2, the performance of random label forests are gaping from 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:

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

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 Appendix E, 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.

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*.

A Difference between Random Label Forests and RAKEL

Let us discuss the differences between random label forests and the pioneer label subsampling method RAKEL. The sharpest contrast is that random label forests are designed for XML problems, but RAKEL is not. The reason is that the label powerset method in RAKEL is only efficient on multi-label problems with small labels, so using that method with label subsampling in XML problems may require extremely huge number of sampled label subsets to cover whole labels. Hence, applying other XML methods such as tree-based methods on each large-scale label subset can increase the sample rate in label subsampling, and the number of label subsets can then be reduced. Furthermore, we consider to use distributed setting such that each computer only handles a smaller XML problem with a label subset.

B Time Complexity of Two-Level Tree-Based Model

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[1]. 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 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 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

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

C Data Sets

We show the statistics of data sets in Table 5. The sets "eur-lex-4k," "wiki10-31k," and "amazoncat-13k" are downloaded from "LIBSVM Data: Multi-label Classification[2]". The sets "amazon-670k," "amazon-3m," and "wiki-500k" are downloaded from the link that is supported by Yu et al. (2022). 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.

D 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 not suitable. 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 credible.

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

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

Data Set	l # of training instances	# of test instances	n # features	m # labels
eur-lex-4k	15,449	3,865	186,104	3,956
wiki10-31k	14,146	6,616	104,374	30,938
amazoncat-13k	1,186,239	306,782	203,882	13,330
amazon-670k	490,449	153,025	135,909	670,091
amazon-3m	1,717,899	742,507	337,067	2,812,281
wiki-500k	1,779,881	769,421	2,381,304	501,070

Table 5: Statistics of data sets

In our experiments, we utilize the library LibMultiLabel[3] 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 (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022):

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

In the implementations of (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022), they decided to early stop the training process in each linear model training, but 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[4] in Khandagale et al. (2020), and we consider the following hyper-parameters

- K for K-means: 100,
- max level of the tree: 10.

For the random label forests, we consider uniform sampling with a sample rate of 0.1 on the labels because reducing 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.

Data set	Tree with all labels	Rand. 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
amazon-3m	503575.79 s	23023.76 s
wiki-500k	202261.66 s	22987.20 s

Table 6: Training time comparison between a tree with all labels and a tree of random label forests. Note that the sample rate is 0.1 in random label forests.

E Comparison in Training Time

Section 3.3 further discussed the time complexity of random forests that uses two-level tree-based submodels. Table 6 shows the training time comparison between a tree-based model with all labels and a tree of random label forests over different data sets. We can see that the training time is reduced a lot in random label forests if each machine only contains one tree-based submodel. The reduction becomes more apparent, especially when the number of labels is larger.

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

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