Exploring Space Efficiency in a Tree-based Linear Model for Extreme Multi-label Classification

He-Zhe Lin[1,2] **Cheng-Hung Liu**[1] **Chih-Jen Lin**[1,2]

[1]National Taiwan University
[2]Mohamed bin Zayed University of Artificial Intelligence
{b07902028, d07944009, cjlin}@csie.ntu.edu.tw

Abstract

Extreme multi-label classification (XMC) aims to identify relevant subsets from numerous labels. Among the various approaches for XMC, tree-based linear models are effective due to their superior efficiency and simplicity. However, the space complexity of tree-based methods is not well-studied. Many past works assume that storing the model is not affordable and apply techniques such as pruning to save space, which may lead to performance loss. In this work, we conduct both theoretical and empirical analyses on the space to store a tree model under the assumption of sparse data, a condition frequently met in text data. We found that some features may be unused when training binary classifiers in a tree method, resulting in zero values in the weight vectors. Hence, storing only non-zero elements can greatly save space. Our experimental results indicate that tree models can require less than 10% of the size of the standard one-vs-rest method for multi-label text classification. Our research provides a simple procedure to estimate the size of a tree model before training any classifier in the tree nodes. Then, if the model size is already acceptable, this approach can help avoid modifying the model through weight pruning or other techniques.

1 Introduction

Extreme multi-label classification (XMC) focuses on tagging a given instance with a relevant subset of labels from an extremely large label set. There is a wide range of applications, from online retail search systems (Chang et al., 2021) to automatically tagging labels on a given article or web page (Jain et al., 2016). XMC problems are commonly encountered in real-world applications, especially in the area of text data. Notably, Bhatia et al. (2016) provide many text data sets for XMC.

Many methods have been proposed to solve XMC for text data. For example, neural networks, particularly pre-trained language models, are effective due to their ability to understand context (Chalkidis et al., 2022). However, linear methods with bag-of-words features remain very useful for XMC due to their simplicity and superior efficiency (Yu et al., 2022). Linear methods are also competitive in certain circumstances (Chang et al., 2021). This motivates us to investigate the time and space complexity of linear methods in XMC problems in this work.

Among linear methods for multi-label problems, the simplest one-vs-rest (OVR) setting treats a multi-label problem with L labels as L independent binary problems, learning a weight vector $\boldsymbol{w}_j$ for each label $j \in \{1, \ldots, L\}$. However, for OVR in XMC, current computing resources become unaffordable as the training time and storage for $\boldsymbol{w}_j$'s grow linearly with L.

An important line of research on reducing time and space is to construct a label tree (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022). The label tree recursively decomposes the XMC problem into smaller ones, and each node in the tree only handles a subset of labels so that the overall training time grows with respect to $O(\log^2 L)$ instead of $O(L)$ as shown in Prabhu et al. (2018). However, we explain in Section 2.2 that more classifiers are needed for a label tree compared to OVR. If weights of these classifiers are dense vectors in the same dimensionality of the input features, storing a tree-based model would need more space then OVR. To tackle this issue, nearly all past works on tree-based models such as Prabhu et al. (2018); Khandagale et al. (2020); Yu et al. (2022) apply weight pruning to change small non-zero entries to zero. However, our experiment in Section 4 shows that weight pruning via an improper threshold can result in varying degrees of performance drops. This situation motivates us to study the model size of a tree-based model in practice.

In this work, we focus on a less-studied aspect

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 16245–16260

– the needed space to store a linear tree model for sparse data. Data sparsity is ubiquitous in XMC. For example, for the bag-of-words features widely used in text classification, each instance only contains several non-zero entries. In the tree methods, each node handles a subset of labels and trains on instances corresponding to them. Since the data are sparse, the training subset for a node may have some unused features, causing a reduction on the feature space. Thus, if weight vectors are mapped back to the input feature space, many elements are zeros and we can use a sparse format to save the space. This property suggests that a tree model – despite having more classifiers – may still take less space than an OVR model. Surprisingly, none of the papers investigate the actual size of a tree model; instead, pruning is directly applied.

For sparse training data, we study this issue through theoretical analysis and experiments. Our main finding is that for sparse data, a tree-based linear model is smaller than what people thought before. Compared with the OVR approach, the model size is 10% or even less for problems with a large number of labels. Our research suggests that one should not impose the pruning procedure without checking the model size, which can be easily calculated before the real training.

The outline is as follows. Section 2 defines the XMC problem and introduces two main linear approaches: OVR and tree-based methods. Section 3 compares both the time complexity and the model size of the two approaches. We review some techniques to reduce the model size and discuss their potential issues in Section 4. We explain the space-efficiency of tree-based linear models for sparse data in Section 5. Section 6 presents experimental results and conclusions are provided in Section 7. This work is an extension of the first author's master thesis (Lin, 2024). Additional materials including programs used for experiments are available at https://www.csie.ntu.edu.tw/~cjlin/papers/multilabel_tree_model_size/.

2 Linear Methods for XMC

To address XMC, there are two main categories for linear methods: the one-vs-rest (OVR) method and tree-based methods. In this section, we will introduce how these methods work. Consider the multi-label classification problem involving L labels and ℓ training instances $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{\ell}$, where

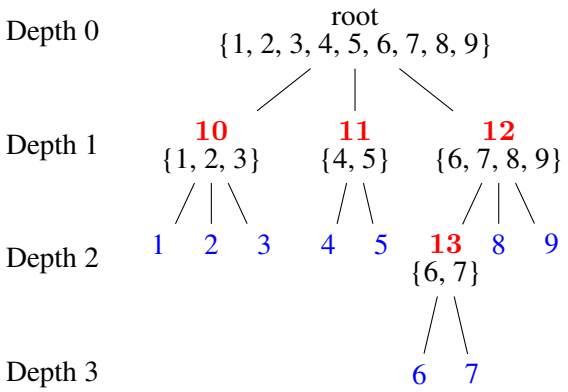


Figure 1: A label tree with nine labels. We set the number of clusters $K = 3$ at each node for the label partition. In the figure, each internal node colored red is associated with a label subset, and each leaf node colored blue corresponds to a single label.

$\boldsymbol{x}_i \in \mathbb{R}^n$ represents the feature vector with n features, and $\boldsymbol{y}_i \in \{-1, 1\}^L$ is the L-dimensional label vector. For each label vector $\boldsymbol{y}_i$, if $\boldsymbol{x}_i$ is associated with label j, then the j-th component of $\boldsymbol{y}_i$, denoted by y_{ij}, is 1; otherwise, $y_{ij} = -1$.

2.1 One-vs-rest (OVR) Method

For each label $j \in \{1, \ldots, L\}$, the linear one-vs-rest approach trains a binary linear classifier by using instances with label j as positive instances and the others as negative ones. The learned weight vector $\boldsymbol{w}_j \in \mathbb{R}^n$ is obtained by solving the minimization problem

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) = \sum_{i=1}^{\ell} \xi(y_{ij} \boldsymbol{w}^T \boldsymbol{x}_i) + R_\lambda(\boldsymbol{w}), \quad (1)$$

where $\xi(\cdot)$ is the loss function and $R_\lambda(\boldsymbol{w})$ denotes the regularization function with parameter λ. As we will discuss in Section 3, OVR is inefficient since both the time and space complexity grow linearly in the number of labels.

2.2 Tree-based Methods

To reduce the needed time and space, many works (Prabhu et al., 2018; Khandagale et al., 2020; Yu et al., 2022) focused on tree-based methods. These methods use divide-and-conquer to recursively break down a multi-label problem with L labels into several smaller sub-problems involving subsets of labels. Figure 1 is an example of a constructed tree with nine labels. A possible procedure of construction is as follows.

- For each label $j = 1, \ldots, L$, we compute the label representation $\boldsymbol{v}_j$ by summing up

	positive	negative
13	with labels 6 or 7	with labels 8 or 9 but neither with label 6 nor 7
8	with label 8	with labels 6, 7 or 9 but not with label 8
9	with label 9	with labels 6, 7 or 8 but not with label 9

Table 1: An example for training an OVR model for the node of meta-label **12** in Figure 1, where the node has 3 children including meta-label **13**, label 8, and label 9. For example, an instance having labels 6, 7 and 8 is regarded as positive in the first two binary problems since it has labels within the label subsets for meta-labels **13** and label 8.

the feature vectors of the instances associated with label j followed by normalization. That is,

$$\boldsymbol{v}'_j = \sum_{i:y_{ij}=1} \boldsymbol{x}_i \text{ and } \boldsymbol{v}_j = \frac{\boldsymbol{v}'_j}{\|\boldsymbol{v}'_j\|_2}. \quad (2)$$

- Starting from the root node, we apply a clustering method to all $\boldsymbol{v}_j$'s to partition all labels into K clusters. Each cluster corresponds to a child node and contains a subset of labels.
- Each child node is recursively partitioned into K clusters[1] until either of the following termination conditions happen.
  - The number of labels in a node is no more than K.
  - The node reaches depth-$(d_{\max} - 1)$, where $d_{\max}$ is a pre-set maximum tree depth.[2]

  If a node stops partitioning but still has multiple labels in the subset, we add a child node for each label in the subset. Therefore, every leaf node in the tree corresponds to a single label.

We denote d as the actual depth of the tree. It may be smaller than the specified depth $d_{\max}$ if before $d_{\max}$ each leaf node already has only one label.

For easy discussion, for any node which is neither the root nor a leaf node, we tag a meta-label on it; see an example in Figure 1. In the label tree, each internal (non-leaf) node with r child nodes corresponds to a multi-label classification problem of r (meta)-labels. The r children may include meta-labels (i.e., internal nodes) and the original labels (i.e, leaf nodes). Similar to the OVR setting, we train a binary problem for each of the r branches. However, instead of using the whole training set, we only use instances with labels in the node's label subset. Take the node of meta-label **12** in Figure 1 as an example. We only use instances having at least one of labels {6, 7, 8, 9} to train three binary problems, which correspond to the three children including meta-label **12**, label 8 and label 9. The positive/negative instances of each binary problem are shown in Table 1. As our focus is on the time and space complexity for constructing a tree model, we omit discussing the details of the prediction procedure. Readers can check Prabhu et al. (2018); Khandagale et al. (2020); Yu et al. (2022) for details.

3 Time and Space Analysis for Linear Methods

In this section, we compare both the training time and the model size for OVR and tree-based methods. The training-time analysis clearly demonstrates why people favor tree-based methods over OVR, while the model-size discussion highlights the expensive space cost with tree-based methods. Although these methods are well documented, our descriptions may be the first to discuss the complexity of the model size in detail.

Through this section, we follow Prabhu et al. (2018) to assume the constructed label tree of depth d is balanced, as shown in Figure 2. That is, given the number of clusters K, we assume an ideal situation so that at each node, a clustering method splits its label subset to K equally sized clusters. By this design, a specified $d_{\max}$ that is not too large to exhaust all labels results in a tree with depth $d = d_{\max}$. Thus, in our analysis of using the tree in Figure 2, only d appears in the time and space complexity.

We further assume that each training instance $\boldsymbol{x}_i$ has $\bar{n}$ non-zero elements on average.

Table 2 gives a summary for the training time complexity and the model size. We now explain each entry in detail.

[1] Each child node may have different K's, but for simplicity, we apply the same number of partitions here.

[2] We follow the setting in Khandagale et al. (2020). In contrast, Prabhu et al. (2018) set a parameter M to stop growing the tree if the number of labels in a node is less than M.

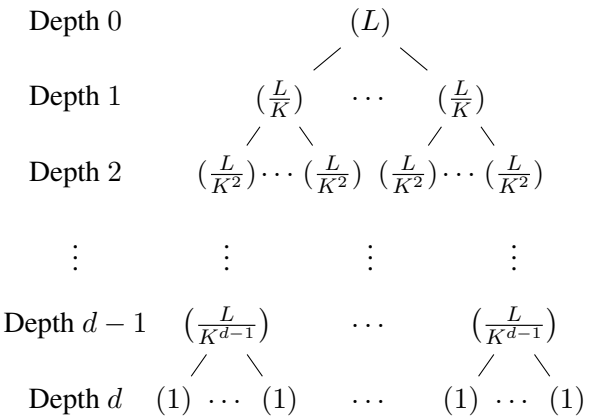


Figure 2: Illustration for a depth d balanced label tree with number of clusters K. The number within $(\cdot)$ at each node means the size of the node's label subset. Nodes from depth 0 to depth $(d-2)$ all have K children. Because the tree needs to be terminated at depth d, each node at depth $d-1$ has L/K^{d-1} children.

Model	OVR	Balanced-tree
Time	$O(L\ell\bar{n})$	Eq. (5)
Model size	Ln	$\left(L + \frac{K^d - K}{K-1}\right) n$

Table 2: A summary of the training time complexity and the model size for linear methods.

3.1 Time Analysis

For training an OVR model, we use all instances $\{\boldsymbol{x}_i\}_{i=1}^{\ell}$ to train each of the L binary problems (1). According to Hsieh et al. (2008); Galli and Lin (2022), the complexity for solving a binary problem is

$$O(\text{\# nonzero feature values in the training set}) \times \text{\# iterations}. \quad (3)$$

Because the number of iterations is usually not large in practice, we may treat it as a constant in the complexity analysis. Therefore, the time complexity for training an OVR model is

$$O(L\ell\bar{n}). \quad (4)$$

In contrast, training a tree model is very time-efficient because most binary problems only use part of the training set. Prabhu et al. (2018) give theoretical time complexity for the training procedure under reasonable assumptions. However, they do not set the maximum depth in a tree as a termination condition, and their analysis lacks detailed explanation. In Appendix A we show that the time complexity for training a tree model of tree depth d is

$$O\left(\ell\bar{n}\log L \times \left(K(d-1) + \frac{L}{K^{d-1}}\right)\right). \quad (5)$$

Based on (5), the time for training a tree of depth $d = \lceil \log_K L \rceil$ is

$$O\left(K\ell\bar{n}\log^2 L\right). \quad (6)$$

3.2 Space Analysis

To obtain the model size, we must compute the number of weight vectors in a model and discuss the needed space to store each weight vector.

We explain that in general, the solution of (1) is dense; that is, most elements of $\boldsymbol{w}_j$ are non-zeros. This property is critical for our analysis. We mentioned in Section 3.1 that problem (1) is solved by iterative optimization algorithms, where each iteration often involves using the gradient for updating $\boldsymbol{w}$. For easy discussion, let us assume that both $\xi(\cdot)$ and $R_\lambda(\boldsymbol{w})$ are differentiable (e.g., logistic loss and ℓ_2-regularization). The gradient $\nabla f(\boldsymbol{w}) \in \mathbb{R}^n$ is given by

$$\sum_{i=1}^{\ell} \xi'(y_{ij}\boldsymbol{w}^T\boldsymbol{x}_i)y_{ij}\boldsymbol{x}_i + \nabla_{\boldsymbol{w}} R_\lambda(\boldsymbol{w}). \quad (7)$$

If the derivative $\xi'(\cdot)$ is non-zero, which is always the case for logistic loss, as long as a feature occurs in some instances, then the corresponding gradient component is likely non-zero. The reason is that the sum of several non-zero values usually remains non-zero. Therefore, regardless of the sparsity of the feature vectors $\{\boldsymbol{x}_i\}_{i=1}^{\ell}$, we roughly have that

$$\text{if a feature is used in the training set} \quad (8)$$
$$\Rightarrow \text{corresponding component in } \boldsymbol{w} \text{ is non-zero.}$$

For our discussion, we assume that every feature in the training set (i.e., $\boldsymbol{x}_i$, $\forall i$) is used. This assumption is reasonable because one should remove unused features in the input data.

From the property in (8) and our assumption that the training set has no unused features, for storing an OVR model we need to save

$$Ln \quad (9)$$

weight values for all L weight vectors.

Next, we compute the number of weight vectors $\hat{L}$ in a tree model. From the discussion in

Section 2.2, each node trains the same number of weight vectors as the node's children, so we have

$$\begin{aligned}\hat{L} &= \sum_{s\in\text{nodes}} (\#\text{ child nodes of } s)\\ &= (\#\text{ nodes in the tree}) - 1\\ &= (\#\text{ leaf nodes}) + (\#\text{ internal nodes}) - 1\\ &= L + (\#\text{ meta-labels}). \end{aligned} \tag{10}$$

From (10), we see that training a tree model needs to afford additional storage for the weights of the meta-labels compared to OVR. For a balanced tree in Figure 2, the number of meta-labels is

$$\sum_{i=1}^{d-1} K^i = \frac{K^d - K}{K-1}.$$

So the total number of weight values we have to store for a balanced tree model is

$$\hat{L}n = \left(L + \frac{K^d - K}{K-1}\right) n. \tag{11}$$

4 Techniques for Reducing Model-size and Their Issues

Results in Section 3.2 indicate that the huge model size is problematic for both OVR and tree-based method in the XMC case. For example, Yu et al. (2022) mentioned that the well-known Wiki-500k (Bhatia et al., 2016) data set requires approximately 5TB space to store a linear OVR model, which is infeasible for a single computer. In this section, we briefly review some techniques to reduce the model size and discuss their issues.

For OVR method, Babbar and Schölkopf (2017) use weight pruning to change small values to zero. By storing only non-zero weights, this strategy effectively reduces the model size. However, the model may behave differently since it is fundamentally changed. Also, this strategy brings other issues such as the selection of suitable pruning thresholds. Other works (Yen et al., 2016, 2017) use L1-regularization to encourage sparse weight vectors without sacrificing performance. However, according to the results provided in Prabhu et al. (2018), the slow training and prediction time for XMC problems is still not addressed.

On the other hand, tree-based methods, according to the results in Section 3.2, have a larger model compared to OVR. Since the model size for OVR is already not affordable, past works such as Prabhu et al. (2018); Khandagale et al. (2020); Yu et al. (2022) may directly assume that reducing the model size is a must. Therefore, all of them perform weight pruning as in Babbar and Schölkopf (2017). However, in the following experiment we show that weight pruning in tree-based methods may cause a performance loss.

Pruning	P@1	P@3	P@5	P@1	P@3	P@5
	EUR-Lex			AmazonCat-13k		
No	82.12	68.90	57.72	92.96	79.21	64.43
Yes	82.08	68.83	57.50	92.95	79.19	64.40
	Wiki10-31k			Amazon-670k		
No	84.49	74.37	65.55	44.11	38.94	35.09
Yes	84.66	74.37	65.46	43.75	38.55	34.64

Table 3: Precision scores of the tree model without and with weight pruning. To construct the label tree, we use LibMultiLabel's default parameters $K = 100$ and $d_{\max} = 6$. In general, the partitioning of most nodes stops before reaching $d_{\max}$ (For example, Amazon-670k has $L = 670,091$ labels, resulting in a balanced tree with a depth of $\log_{100} L \leq 3$. So if most nodes reach a depth of 10, the tree would be extremely imbalanced). We train each binary problem (1) using squared hinge loss with ℓ_2-regularization.

We consider the four smaller data sets used in our experiments; see details in Section 6. By using the package LibMultiLabel[3] to conduct the training and prediction, we compare the test precision scores (P@$\{1, 3, 5\}$) without and with weight pruning in Table 3. The threshold for pruning is 0.1, so any weight within $[-0.1, 0.1]$ is changed to zero. From Table 3, the performance drops in all data sets, though the score differences vary across data sets. In particular, the loss is significant in Amazon-670k. Note that we choose the 0.1 threshold by following Prabhu et al. (2018) and Khandagale et al. (2020). Our results indicate the difficulty in choosing a suitable threshold. In fact, the size of the tree model, discussed in Section 5 and experimentally shown later in Figure 4, is very small and can be easily stored in one computer.

5 Inherent Pruning in Tree-based Methods for Sparse Data

In this section, we explain that for sparse data, the number of weight values needed to be stored in a tree model can be much less than not only the huge value in (11), but also nL, the number of weight values in an OVR model. We stress that the model

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

size reduction here is not achieved by applying any techniques. Instead, it can be regarded as an innate advantage of tree-based methods and we call such reduction "inherent pruning."

In the tree-based method, suppose we are training weight vectors at a tree node u. Only a subset of all training instances (specifically, instances having any label in the label subset of u) are used. Because the feature vectors are sparse, some features may have no values in the subset of training instances. We can remove the unused features before training a multi-label model for the node. Alternatively, if our optimization algorithm for each binary classification problem (1) satisfies that

- the initial $\boldsymbol{w}$ is zero, and
- for unused features (i.e., feature value are zero across all instances), the corresponding $\boldsymbol{w}$ components are never updated,

then we can conveniently feed the subset of data into the optimization algorithm and get a $\boldsymbol{w}$ vector with many zero elements. This way, we keep all weight vectors in all nodes to have the same dimension n. We can collect them as a large sparse matrix for easy use.

We use the name "inherent pruning" because the tree-based method itself "prunes" weight values for unused features by not updating the corresponding weight components during training. Our survey shows that few works, except Jasinska-Kobus et al. (2020), mentioned the identity. Even though, Jasinska-Kobus et al. (2020) only briefly said that "the weight sparsity increases with the depth of a tree ... implies a significant reduction of space" without further discussion or experiments.

5.1 Analysis on Balanced Trees

We theoretically analyze the size of a tree model for sparse data under the following assumptions.

- Our analysis follows Figure 2 to have a K-ary balanced tree of depth d.
- As the tree depth grows, the training subset becomes smaller and the number of used features also reduces. Hence, when the number of labels is divided by K, we assume that the number of remaining features is multiplied by a ratio $\alpha \in (0, 1)$.

We list the information for each depth in Table 4.

Depth	# nodes	# children / node	# features
0	K^0	K	n
1	K^1	K	αn
$\vdots$	$\vdots$	$\vdots$	$\vdots$
i	K^i	K	$\alpha^i n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$d-1$	K^{d-1}	L/K^{d-1}	$\alpha^{d-1} n$

Table 4: Depth-wise summary of a tree model with depth d.

Under the assumptions, we discuss the largest possible tree depth, denoted by D. For a tree of depth D, because nodes at depth-D cover all labels, each node at depth-$(D-1)$ must contain at least two labels; see the illustration in Figure 2. Therefore, D is the largest possible integer to satisfy

$$\frac{L}{K^{D-1}} \geq 2. \tag{12}$$

We then have

$$D = \left\lfloor 1 + \log_K \frac{L}{2} \right\rfloor. \tag{13}$$

On the other hand, the minimum depth of a label tree is $d = 2$. Otherwise, if $d = 1$, Table 4 shows that at depth-0, the root node has L children, which is simply the OVR case. Therefore, the range of the tree depth is $2 \leq d \leq D$.

We choose the OVR model with Ln weight numbers as the comparison baseline because an OVR model is more space-efficient than a tree model with a dense weight matrix, which takes a space of $\hat{L}n$ weight values in (11). Then, we compute the number of non-zero weights in a tree model

$$\begin{aligned} &\sum_{i=0}^{d-2} (K^i)(K)(\alpha^i n) + K^{d-1} \left(\frac{L}{K^{d-1}} \right) (\alpha^{d-1} n) \\ &= Kn \cdot \frac{(K\alpha)^{d-1} - 1}{K\alpha - 1} + L\alpha^{d-1} n. \end{aligned} \tag{14}$$

A minor issue is that to have (14) well defined, we need $K\alpha \neq 1$. We discuss this exceptional situation in Appendix B.2.

We compare (14) with the number of non-zeros in an OVR model by the following ratio:

$$\frac{(14)}{Ln} = \frac{K((K\alpha)^{d-1} - 1)}{L(K\alpha - 1)} + \alpha^{d-1}. \tag{15}$$

The following theorem illustrates that the tree model generally contains less non-zero weight values than the OVR model. We give the proof in Appendix B.

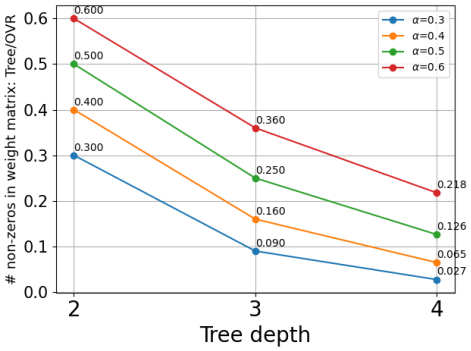


Figure 3: The ratio of number of non-zeros between a tree model and an OVR model, calculated based on (15). We show the cases for $\alpha = \{0.3, 0.4, 0.5, 0.6\}$.

Theorem 1. *Consider $2 < d \leq D$ and assume $K \geq 4$. Let α^* be the unique solution in $(0, 1)$ of the equation*

$$\alpha^{d-2}(K^{d-D} + \alpha) - 1 = 0. \tag{16}$$

If $\alpha < \max\{2/K, \alpha^*\}$, then the ratio (15) is smaller than one.

In Theorem 1, we consider $d > 2$ because for $d = 2$, the obtained bound on α is in a slightly different form; see details in Appendix B.

Although Theorem 1 imposes an upper bound on α, we show that the bound is in general close to one. For example, if $L = 2 \cdot 10^8$ and $K = 100$, we have $D = 5$ according to (13). Then we only need $\alpha < 0.999$ for a depth 2 or 3 tree, $\alpha < 0.996$ for a depth 4 tree and $\alpha < 0.819$ for a depth 5 tree. Therefore, even if the number of used features is only minorly reduced after each label division, we can significantly lower the size of a tree model from (11) to be smaller than that of OVR.

The following theorem shows that a deeper tree leads to a smaller model, with its proof given in Appendix B.

Theorem 2. *If $\alpha < 1 - 1/(2K)$, the ratio (15) is decreasing in d for $2 \leq d \leq D - 2$. Specifically, for a tree with depth d within this range, the ratio is smaller than that of a tree of depth $d + 1$.*

As an illustration of Theorem 2, we plot the ratio (15) with $L = 2 \cdot 10^8$, $K = 100$ and different α's in Figure 3. For this illustration, $D = 5$, so Theorem 2 is applicable for $2 \leq d \leq 3$. In Figure 3, we see that the ratio reduces as the tree grows from $d = 2$ to 4, regardless of the value of α.[4] Later in Section 6.3, we shall see that the experiments on real-world data align with our theoretical analysis on balanced trees.

[4]See more discussion in Appendix C.

Data set	#training data l	#features n	#labels L
EUR-Lex	15,449	186,104	3,956
AmazonCat-13k	1,186,239	203,882	13,330
Wiki10-31k	14,146	104,374	30,938
Wiki-500k	1,779,881	2,381,304	501,070
Amazon-670k	490,449	135,909	670,091
Amazon-3m	1,717,899	337,067	2,812,281

Table 5: The statistics of extreme multi-label data sets, ordered by the number of labels. Wiki-500k and Amazon-3m are downloaded from the GitHub repository provided in You et al. (2019); others are from "LIBSVM Data: Multi-label Classification."[5] More details are in Appendix D.

6 Experimental Results

In this section, we compare the model size of OVR and tree-based methods across several extreme multi-label text data sets. The statistics for these data sets are listed in Table 5.

6.1 Experimental Settings

For the tree model to be compared, the constructed tree should have high prediction performances; otherwise, claiming that a low-performing tree saves space would be meaningless. To find out settings with high performances, we must conduct a hyper-parameter search on the number of clusters K and the maximum tree depth $d_{\max}$, etc. Since developing an effective search procedure is out of the scope of this work, we instead consider tree structures that have been investigated in Khandagale et al. (2020) due to their high performances among almost all data sets. Specifically, we calculate the model size for the following cases:

- fixed-K: In Khandagale et al. (2020) they suggested wide trees ($K \geq 100$). Therefore, we fix $K = 100$ for all data sets and check the model size with $d_{\max} \in \{2, 3, 4, 5, 6\}$. This setting can be regarded as an empirical validation for the study in Section 5.1.
- varied-K: In contrast to a fixed K, we vary it according to the specified $d_{\max}$. Specifically, we set $K = \lceil L^{1/d_{\max}} \rceil$ by considering $d_{\max} \in \{2, 3, 4\}$. We do not consider larger $d_{\max}$ because under this setting of choosing

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

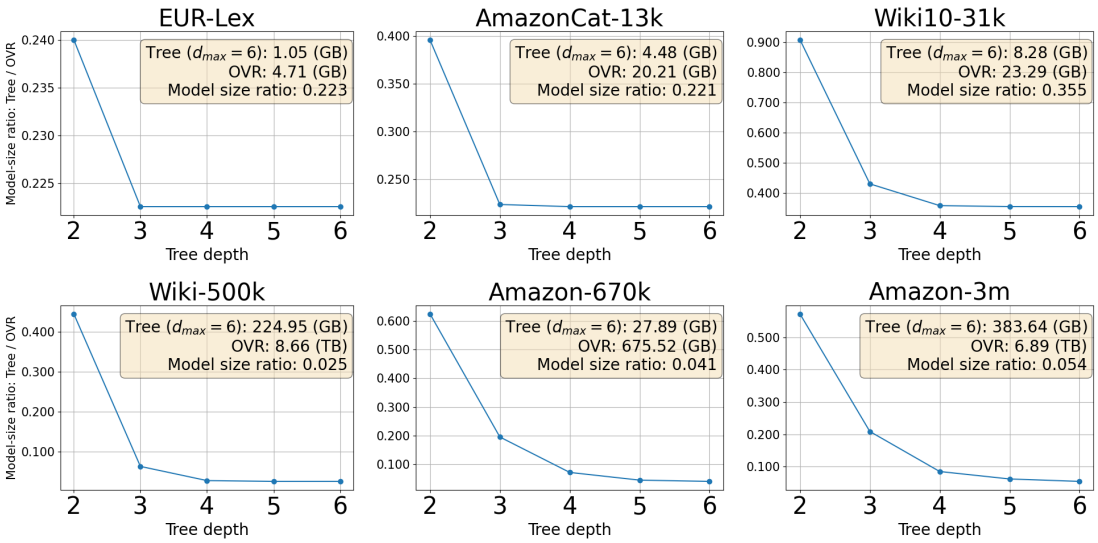


Figure 4: The ratio between the tree model size and the OVR model size with $K = 100$ and various $d_{\max}$. The box in each sub-figure shows the actual model size of tree/OVR models under $d_{\max} = 6$ and the ratio between the two.

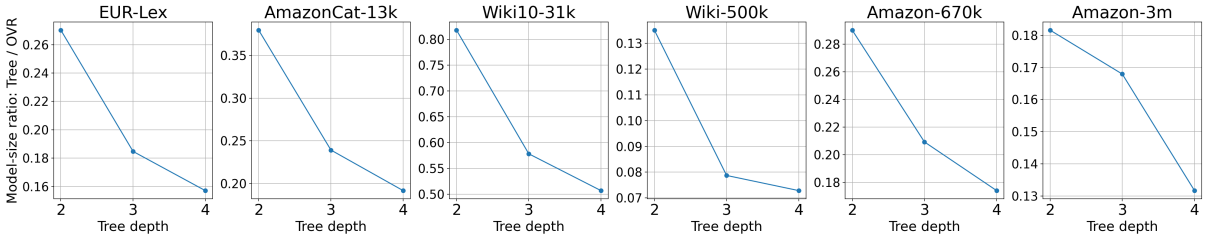


Figure 5: The ratio between the tree model size and the OVR model size with $d_{\max} = \{2, 3, 4\}$ and $K = \lceil L^{1/d_{\max}} \rceil$.

K, the performance (Khandagale et al., 2020) of deeper trees is poor.

We use the software LibMultiLabel[6] to conduct the experiment. Currently LibMultiLabel uses K-means algorithm (Elkan, 2003) implemented in the package scikit-learn[7] for partitioning. Since the K-means algorithm involves a random selection of K centroids, we conduct the experiments five times on all data sets using different seeds. More details of our experimental settings are in Appendix D.

6.2 Estimating Model Size Prior to Training

We explain that the size of a tree-based model can be estimated before training any binary classifiers. Based on (8), a tight upper bound on the true model size is by summing up each binary problem's used features. Suppose the weights are stored as double-precision floating-point numbers. For an OVR model, we have

$$nL \times 8 \text{ bytes} \tag{17}$$

as the estimation of the model size. For the tree model, the number of non-zero weights is bounded by

$$\sum_{u \in \text{nodes}} \#\text{ children of } u \cdot \#\text{ used features of } u. \tag{18}$$

However, we also need to store the index of each used feature. If we assume a four-byte integer storage for the index, then the model size for a tree model is roughly

$$(18) \times 12 \text{ bytes}, \tag{19}$$

and the ratio of a tree-based model to an OVR model is

$$\frac{(18) \times 1.5}{nL}. \tag{20}$$

6.3 Empirical Analysis on the Model Size

Figure 4 shows the relative size of a tree model compared to OVR under the fixed-K setting. Besides, in a separate box of each sub-figure, we give the actual model size of an OVR model and a tree model with $d_{\max} = 6$. We find that the memory

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

[7] https://scikit-learn.org

consumption is indeed acceptable for a single computer. For smaller data sets, the tree model size is around 20 to 40% of the OVR model, while for large data sets, the ratio is lower than 10%. Our results fully support the space efficiency of tree-based methods on sparse data. Moreover, as $d_{\max}$ grows in the early stage, the ratio (20) significantly drops. This observation is consistent with our analysis in Section 5.1. However, if we further grow $d_{\max}$, the model size may not change much. The reason is that the partitioning finishes before reaching the maximum depth, causing the actual tree depth d being smaller than $d_{\max}$.

For the varied-K setting, the relative size between the two models is presented in Figure 5. The model size still keeps decreasing as the tree becomes deeper. A comparison between Figure 4 and Figure 5 shows that, under the same $d_{\max}$, in general a larger K leads to a smaller model. For example, if $d_{\max} = 4$, the setting of $K = \lceil L^{1/d_{\max}} \rceil$ of varied-K leads to $K \leq 100$ (as our largest L is less than 10^8); i.e., smaller than fixed-K. The ratios in Figure 4, especially for the larger four sets, are clearly smaller than those in Figure 5. Although a larger K brings more binary problems to train in a tree model, it also leads to more unused features after a label division. Apparently, within the scope of our experimental settings, the increase of unused features has a higher influence on the model size than the more binary problems.

6.4 Empirical Study on the Reduction Rate α

In Section 5.1, for balanced trees, we assume that the number of used features is multiplied by $\alpha \in (0, 1)$ when the number of labels is divided by K. In Appendix E, we empirically study the reduction of used features for unbalanced label trees on real data.

7 Conclusions

In this work, we identify that the many unused features are the main reason for the space-efficiency of tree-based methods under sparse data conditions. Our findings indicate that, for large data sets, the size of a tree model can be reduced to just 10% of the size of an OVR model. In practice, one can first calculate the tree model size as soon as the label tree is constructed. By doing so, we can check whether the model size exceeds the available memory before training any binary problem. This approach avoids directly changing the trained weights such as pruning, which carries the risks of compromising the performance.

Limitations

Though we can estimate the model size for a tree-based model before training the model, it may be still time-consuming to generate the constructed label tree because K-means algorithm takes considerable time. Besides, analyzing the tree model size using a non-constant feature reduction rate (depending on the number of clusters K and the depth d) could be a possible direction.

Acknowledgments

This work was supported in part by National Science and Technology Council of Taiwan grant 110-2221-E-002-115-MY3.

References

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.

Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Katz, and Nikolaos Aletras. 2022. LexGLUE: A benchmark dataset for legal language understanding in English. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 4310–4330.

Wei-Cheng Chang, Daniel Jiang, Hsiang-Fu Yu, Choon-Hui Teo, Jiong Zhang, Kai Zhong, Kedarnath Kolluri, Qie Hu, Nikhil Shandilya, Vyacheslav Ievgrafov, Japinder Singh, and Inderjit S Dhillon. 2021. Extreme multi-label learning for semantic matching in product search. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*, pages 147–153.

Leonardo Galli and Chih-Jen Lin. 2022. A study on truncated Newton methods for linear classification. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2828–2841.

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)*.

Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 935–944.

Kalina Jasinska-Kobus, Marek Wydmuch, Krzysztof Dembczynski, Mikhail Kuznetsov, and Robert Busa-Fekete. 2020. Probabilistic label trees for extreme multi-label classification. *Preprint*, arXiv:2009.11218.

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

He-Zhe Lin. 2024. Exploring space efficiency in a tree-based linear model for extreme multi-label classification. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University.

James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

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.

Ian En Hsu Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. 2017. PPDsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 545–553.

Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit Dhillon. 2016. PD-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, pages 3069–3077.

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, 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.

A Time Analysis on Tree Models

We explain the time complexity (5) for constructing a balanced-tree model with tree depth d. As shown in Figure 2, each node from depth-0 to depth-$(d-2)$ contains K children, and each node at depth-$(d-1)$ has at most $\lceil L/K^{d-1} \rceil$ children. Our assumptions on the training data are based on Prabhu et al. (2018). We assume that

- each training instance $\boldsymbol{x}_i$ has $\bar{n}$ non-zero elements on average, and
- the average number of relevant labels for each instance is bounded by $c \log L$, where c is a constant.

As in Section 2.2, there are three parts to construct a tree model.

1. Computing the label representations as in (2) costs $O(l\bar{n} \log L)$-time.
2. To build a label tree, K-means clustering (MacQueen, 1967) is used to recursively partition the labels. The K-means algorithm has several iterations. For each iteration, one need to calculate the distance from all label representations to the center of each cluster. So learning K-means clustering from depth-1 to depth-$(d-1)$ costs

   $$O(\text{nnz}(V) \times K \times \#\text{iterations} \times d)\text{-time},$$

   where V is the label representation matrix.
3. The last part is to train classifiers for the tree nodes. According to Hsieh et al. (2008); Galli and Lin (2022), the complexity for solving a binary problem is (3). Because the number of iterations is usually not large in practice, we may treat it as a constant in the complexity analysis. Then, we compute the training complexity for each depth-0 to depth-$(d-1)$.

   - Depth-0: We have to train K classifiers. For each binary problem we use all ℓ training instances. Therefore, solving K binary problems costs

     $$O(K\ell\bar{n}). \tag{21}$$

   - Depth-1 to depth-$(d-2)$: At depth-i there are K^i nodes. The corresponding K^i label subsets of the nodes form a partition of all L labels. Since we assume that each instance $\boldsymbol{x}_i$ has less than $c \log L$ labels on average, $\boldsymbol{x}_i$ is used in no more than $c \log L$ nodes. Therefore, by summing the number of used training instances in the K^i nodes, we get a total of $\ell c \log L$. Finally, for each node we have K binary problems to train, so the time complexity for training all K^i nodes (K^{i+1} binary problems in total) is

     $$O(K\ell c(\log L)\bar{n}). \tag{22}$$

   - Depth-$(d-1)$: This is similar to the previous case. The only difference is that for each node we have $\lceil L/K^{d-1} \rceil$ problems to solve (instead of K), which leads to a complexity of

     $$O\left(\frac{L}{K^{d-1}}\ell c(\log L)\bar{n}\right). \tag{23}$$

Therefore, the time complexity for training is

$$(21) + (d-2) \cdot (22) + (23) = (5).$$

By the inequality of arithmetic and geometric means, the inner term in (5) can be written as

$$\underbrace{K + \cdots + K}_{(d-1)\text{ times}} + \frac{L}{K^{d-1}} \geq d\sqrt[d]{L}. \tag{24}$$

The equality in (24) holds when

$$K = \frac{L}{K^{d-1}},$$

which means $d = \log_K L$. Under this value of d, the complexity is as in (6).

B Proof of Theorems

B.1 Proof of Theorem 1

Here we give a full version of Theorem 1 by including the case of $d = 2$.

Theorem 1. *The ratio (15) is smaller than one if any of the following conditions holds.*

(i) $d = 2$ *and*

$$\alpha < 1 - 1/(2K^{D-2}). \tag{25}$$

(ii) $d > 2$, $K \geq 4$ *and* $\alpha < \max\{2/K, \alpha^*\}$, *where* α^* *is the unique solution in* $(0, 1)$ *of the equation*

$$\alpha^{d-2}(K^{d-D} + \alpha) - 1 = 0. \tag{26}$$

Proof of Theorem 1. If $d = 2$, we have

$$(15) = \frac{K}{L} + \alpha \leq \frac{K}{2K^{D-1}} + \alpha < 1, \tag{27}$$

where the inequalities follow from (12) and (25).

If $d > 2$, from the condition

$$\alpha < \max\{2/K, \alpha^*\}, \tag{28}$$

we consider two cases.

a. $\alpha < 2/K$: In this case, α of course satisfies (28). We see that the number of non-zeros in a tree model (14) is increasing in α. Therefore, by $\alpha < 2/K$ we have

$$\begin{aligned}(15) &< \frac{K(K(2/K))^{d-1}}{L(K(2/K) - 1)} + \left(\frac{2}{K}\right)^{d-1} \\ &\leq \frac{2^{d-2}}{K^{D-2}} + \left(\frac{2}{K}\right)^{d-1} \qquad (29) \\ &= \left(\frac{2}{K}\right)^{d-2}\left(\frac{1}{K^{D-d}} + \frac{2}{K}\right) \\ &\leq \left(\frac{2}{4}\right)^{3-2}\left(1 + \frac{2}{4}\right) < 1, \qquad (30)\end{aligned}$$

where (29) is from (12) and (30) follows from $K \geq 4$ and $2 < d \leq D$.

b. We consider α satisfying (28) but not in the previous case. We must have

$$2/K \leq \alpha < \alpha^*.$$

The condition on α implies that $K\alpha \geq 2$ and thus

$$\begin{aligned}\frac{K\alpha}{K\alpha - 1} &= 1 + \frac{1}{K\alpha - 1} \\ &\leq 1 + \frac{1}{2 - 1} = 2.\end{aligned} \tag{31}$$

Then,

$$\begin{aligned}(15) &\leq \frac{K(K\alpha)^{d-1}}{2K^{D-1}(K\alpha - 1)} + \alpha^{d-1} \qquad (32) \\ &= \frac{K\alpha}{K\alpha - 1}\frac{K(K\alpha)^{d-2}}{2K^{D-1}} + \alpha^{d-1} \\ &\leq 2 \cdot \frac{(K\alpha)^{d-2}}{2K^{D-2}} + \alpha^{d-1} \qquad (33) \\ &= \alpha^{d-2}(K^{d-D} + \alpha), \qquad (34)\end{aligned}$$

where (32) follows from (12) and (33) is from (31). Consider the function

$$f(\alpha) = \alpha^{d-2}(K^{d-D} + \alpha) - 1.$$

Clearly, $f(\alpha)$ is strictly increasing in $[0, 1]$, $f(0) = -1$, and $f(1) > 0$. Therefore, f has a unique root α^* in $(0, 1)$ and we have

$$(34) < 1 \text{ if } \alpha < \alpha^*.$$

□

B.2 Theorem 1 for the Exceptional Case of $K\alpha = 1$

First we show that the full version of Theorem 1 is still valid by slightly changing the proof. If $d = 2$, the ratio (15) is in fact

$$Kn + L\alpha n,$$

so (27) remains the same.

If $d > 2$, we only need to check the case of $\alpha < 2/K$ because $\alpha = 1/K$ falls into it. Now (14) becomes

$$(d - 1)Kn + L\alpha^{d-1}n. \tag{35}$$

Therefore, the ratio (15) becomes

$$\begin{aligned}\frac{(35)}{Ln} &= \frac{K(d-1)}{L} + \alpha^{d-1} \\ &\leq \frac{d-1}{2K^{D-2}} + \frac{1}{K^{d-1}} \qquad (36) \\ &\leq \frac{d-1}{2K^{d-2}} + \frac{1}{K^{d-1}}, \qquad (37)\end{aligned}$$

where (36) follows from (12). For the first term in (37), we see that for $K \geq 4$ and $d > 2$, the derivative of

$$\frac{d-1}{2K^{d-2}}$$

with respect to d is

$$\begin{aligned}&\frac{2K^{d-2}(1 - (\ln K)(d-1))}{4K^{2d-4}} \\ &< \frac{2K^{d-2}(1 - \ln K)}{4K^{2d-4}} < 0,\end{aligned}$$

showing that the term is decreasing in d. Because (37) is also decreasing in K, we can get an upper bound by considering $K = 4$ and $d = 3$ to have

$$(37) \leq \frac{3-1}{2 \cdot 4^{3-2}} + \frac{1}{4^{3-1}} = \frac{5}{16} < 1. \quad (38)$$

Interestingly, though in Section B.1 we do not require any condition on the value $K\alpha$, from the property of our balanced trees, we can prove $K\alpha \geq 1$ in the following theorem. Thus, the situation of $K\alpha = 1$ is in fact an extreme case.

Theorem 3. *The feature reduction ratio $\alpha \geq 1/K$ (i.e., $K\alpha \geq 1$).*

Proof of Theorem 3. Suppose n features are used at a node u. According to the definition of α, there are αn features used in each child node of u. Assume for contradiction that $\alpha < 1/K$. The total number of used features among all u's child nodes would be less than

$$K \times (\alpha n) < n,$$

which leads to a contradiction. □

B.3 Proof of Theorem 2

We prove the theorem by showing that for

$$d = 2, \ldots, D-2, \quad (39)$$

the number of non-zero weight values in a tree of depth d is more than a tree of depth $d+1$. For a tree of depth d, the number of non-zeros is

$$\sum_{i=0}^{d-2} (K^i)(K)(\alpha^i n) + L\alpha^{d-1} n, \quad (40)$$

and the number of non-zeros in a depth $(d+1)$ tree is

$$\sum_{i=0}^{d-1} (K^i)(K)(\alpha^i n) + L\alpha^d n. \quad (41)$$

Then we have

$$\begin{aligned}(40) - (41) &= L\alpha^{d-1} n - K^d \alpha^{d-1} n - L\alpha^d n \\ &= \alpha^{d-1} n (L(1-\alpha) - K^d). \quad (42)\end{aligned}$$

Since we have $L \geq 2K^{D-1}$ from (12) and assume $\alpha < 1 - 1/(2K)$,

$$\begin{aligned}L(1-\alpha) &\geq 2K^{D-1}(1-\alpha) \\ &> \frac{2K^{D-1}}{2K} \\ &= K^{D-2} \geq K^d, \quad (43)\end{aligned}$$

where the last inequality is from the range of d considered in (39). We then use (43) in (42) to show that (40) − (41) > 0.

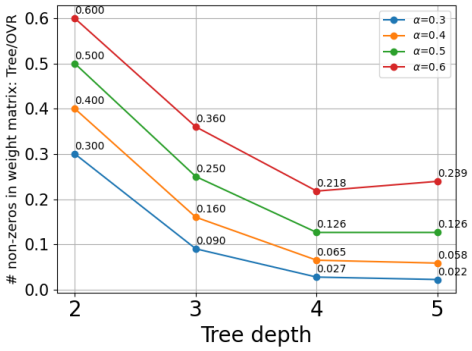


Figure 6: The ratio of number of non-zeros between a tree model and an OVR model from $d = 2$ to $d = D$.

C Comments on Theorem 2

In Theorem 2, we show the model size decreases for $2 \leq d \leq D-2$. However, the ratio for $d = D$ may be larger than the ratio for $d = D-1$. Figure 6 is the plot from $d = 2$ to $d = D$ using the same example as in Section 5.1. We see that for $\alpha = 0.6$, the ratio for $d = 5$ is slightly larger than $d = 4$. Therefore, our theorem gives the widest interval for a decreasing ratio under the given assumption $\alpha < 1 - 1/2K$.

D Details of Experimental Settings

We use LibMultiLabel[8] version 0.6.0. For data sets, the specific link of each set is as follows

- EUR-Lex:
  - Training: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/eurlex_tfidf_train.svm.bz2
  - Testing: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/eurlex_tfidf_test.svm.bz2
- Wiki10-31k:
  - Training: Training:https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/wiki10_31k_tfidf_train.svm.bz2
  - Testing: Testing:https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/wiki10_31k_tfidf_test.svm.bz2

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

- AmazonCat-13k:
  - Training: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/AmazonCat-13K_tfidf_train_ver1.svm.bz2
  - Testing: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/AmazonCat-13K_tfidf_test_ver1.svm.bz2
- Amazon-670k:
  - Training: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/Amazon-670K_tfidf_train_ver2.svm.bz2
  - Testing: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/Amazon-670K_tfidf_test_ver2.svm.bz2
- Wiki-500k and Amazon-3m are downloaded from the following GitHub repository provided in You et al. (2019)
  - https://github.com/yourh/AttentionXML

E Empirical Observations on the Feature Reduction Ratio α

Figure 7 shows the histogram of the reduction ratio α_u for each internal node u, computed by

$$\frac{\#\text{ used features of } u}{\#\text{ used features of } u\text{'s parent}}. \tag{44}$$

In the same figure we also show the weighted average of α for depth-i, denoted by $\bar{\alpha}$ and defined as

$$\bar{\alpha} = \frac{\sum_{u \in \text{depth-}i} \alpha_u \cdot \#\text{ children of } u}{\sum_{u \in \text{depth-}i} \#\text{ children of } u}. \tag{45}$$

We use Figure 8 to illustrate the reason for reporting the weighted average. When training nodes at depth-1, node B, C and D has respectively two, two and six weight vectors, so the average features used for each weight vector should be

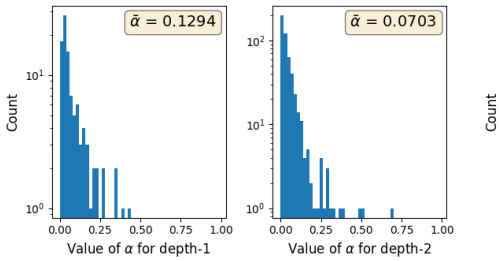
$$\bar{n} = \frac{10 \cdot 2 + 30 \cdot 2 + 80 \cdot 6}{2 + 2 + 6},$$

and the average reduction ratio is

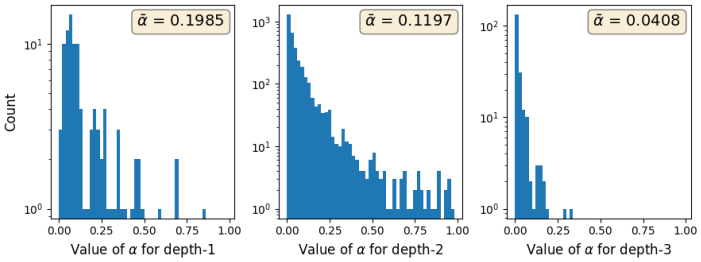
$$\frac{\bar{n}}{100} = \frac{0.1 \cdot 2 + 0.3 \cdot 2 + 0.8 \cdot 6}{2 + 2 + 6} = \frac{\alpha_B \cdot 2 + \alpha_C \cdot 2 + \alpha_D \cdot 6}{2 + 2 + 6},$$

which is equivalent to (45).

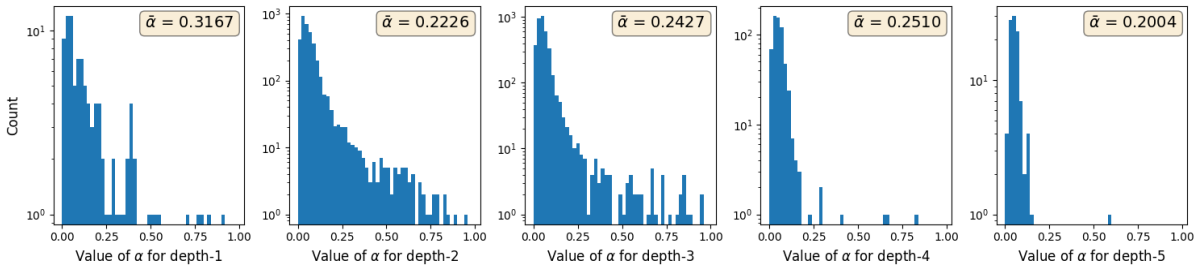
For results in Figure 7, we construct the trees by setting $K = 100$ and $d_{\max} = 6$. The depth of the resulting tree for EUR-Lex and Amazon-13k is only 3 and 4 respectively because the construction has reached the termination condition. Clearly, we see that most $\bar{\alpha}$'s are small, generally much lower than 0.5. The only exception is in the depth-5 of the Amazon-3m set. The reason of a relatively larger $\bar{\alpha}$ is that, under an unbalanced setting, some nodes at layer $d_{\max} - 1$ still contain many labels (much more than K) and need further partitioning, but the tree has reached $d_{\max}$. Therefore, these nodes have more used features than others in the same layer. Then their α values from (44) are larger than others. Together with their larger number of children, these nodes dominate the calculation in (45) and lead to a large weighted average.



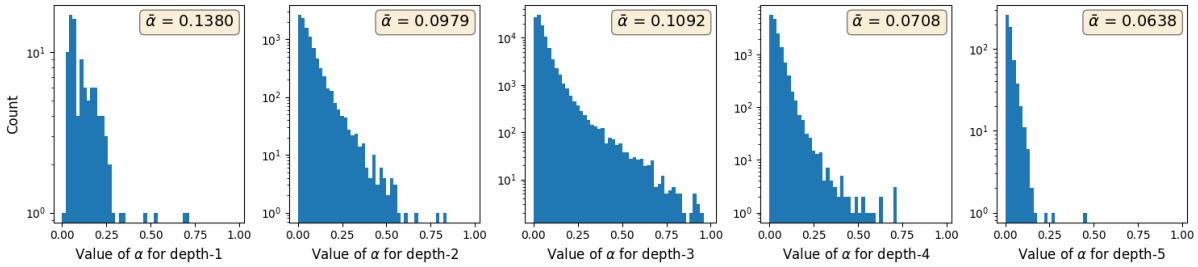
(a) EUR-Lex



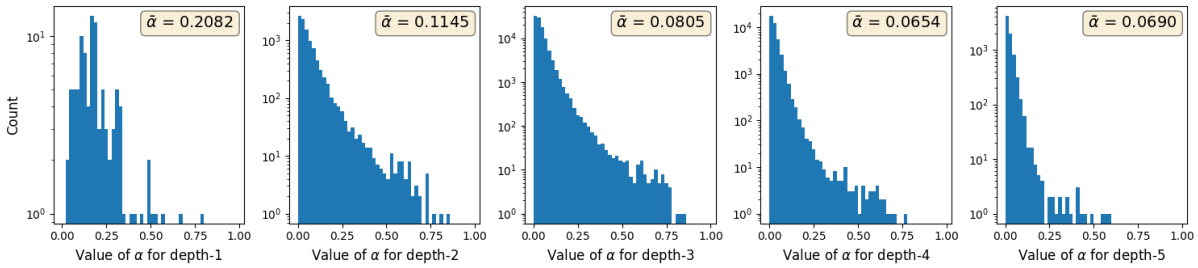
(b) Amazoncat-13k



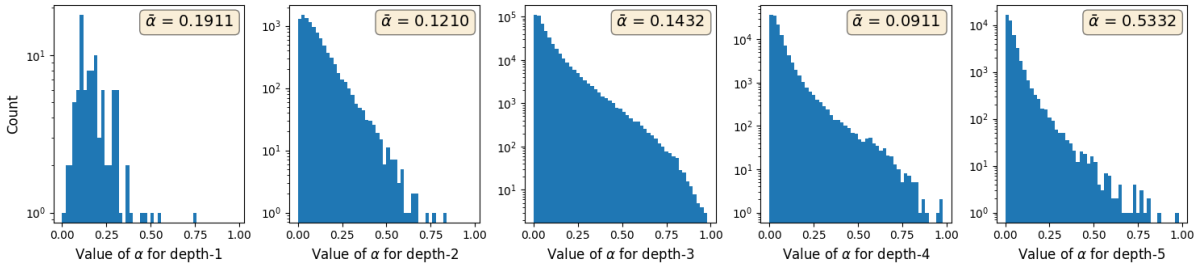
(c) Wiki10-31k



(d) Wiki-500k



(e) Amazon-670k



(f) Amazon-3m

Figure 7: The histogram of α values of nodes with the same depth. The weighted average $\bar{\alpha}$ defined in (45) is listed in the box of each subfigure.

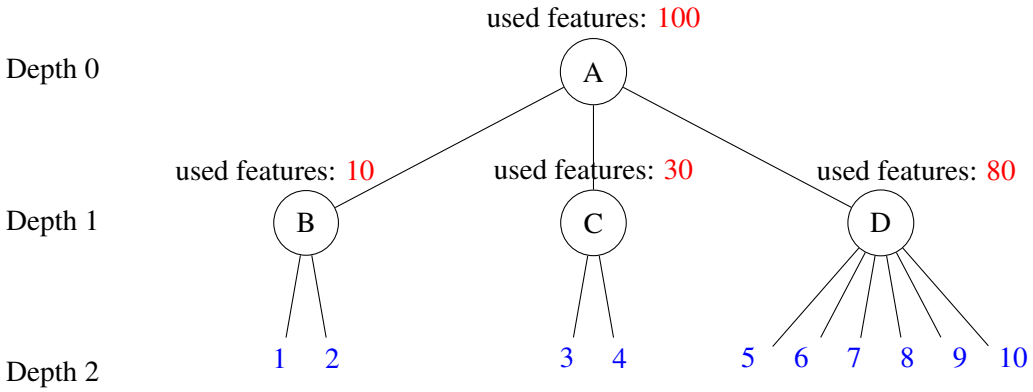


Figure 8: A label tree with ten labels. At depth 1, nodes B, C, and D respectively train two, two, and six linear classifiers. We explain in Appendix E that a weighted average of α values at nodes B, C, and D as in (45) is a reasonable setting to calculate the reduction ratio for depth-1.