Probability Estimation in Tree-Based Linear Methods for Extreme Multi-Label Learning

Sheng-Wei Chen[1], Kuan-Ting Chen[1], Guan-Ting Chen[1], Yaxu Liu[1,2], and Chih-Jen Lin[1,2]

[1] National Taiwan University
d09944003@ntu.edu.tw, andyngewcz2d@gmail.com, r11922155@csie.ntu.edu.tw, d08944012@ntu.edu.tw, cjlin@csie.ntu.edu.tw
[2] Mohamed bin Zayed University of Artificial Intelligence
yaxu.liu@mbzuai.ac.ae, chihjen.lin@mbzuai.ac.ae

Abstract. Extreme multi-label learning aims to identify relevant labels from a large number of labels. Among various approaches, tree-based linear methods stand out for their simplicity and efficiency, and thus serve as an effective baseline. Tree-based linear methods estimate label relevance based on probability estimates at each node, where each probability is transformed from the decision value of a binary model. While most existing methods adopt a simple function for this transformation, we show that such an estimation is inappropriate and can degrade model performance. Through detailed investigation, we explain why prior works relied on this problematic function without considering well-developed probabilistic outputs for binary linear classification. Our study leads to a more suitable probability estimation for tree-based linear methods. Experiments confirm that our proposed method fixes the problem introduced by existing settings.

Keywords: Extreme multi-label classification · Tree-based linear methods · Probability estimation · Support vector machine

1 Introduction

In modern machine learning applications, a common task is identifying relevant categories from a large set of candidates for a given input. For example, in the medical domain, text from clinical records can be used to predict relevant diagnosis and procedure codes. In many cases, the candidate label set can be extremely large, ranging from thousands to millions. This task is known as extreme multi-label learning (XML).

Many studies have proposed methods to address XML. For instance, in text-based XML applications such as product categorization, document tagging, and medical coding, some works [28,16,12] propose neural network-based approaches, while others [11,24,15,29] focus on linear methods. Although neural networks are a popular choice for understanding text semantics [1], linear methods require much less computational resources. Moreover, several studies [25,19,9,6] have further demonstrated that linear models can serve as strong baselines in text classification. This study focuses on probability estimation in linear methods for XML, a critical aspect that has received little attention.

Within linear methods for XML, one-versus-rest (OVR) offers a simple way to train one binary classifier per label, but faces high training and storage costs with large label sets. To overcome this limitation, several works such as [26,11,24,15,29] have proposed tree-based linear methods, which organize labels hierarchically and decompose the XML task into smaller multi-label classification problems. These methods significantly reduce time and space complexity, as analyzed in [24] and [17].

Tree-based linear methods make predictions using a probabilistic label tree [11]. Each label corresponds to a fixed path that involves a sequence of binary classifiers, and each classifier produces a probability estimate based on the given instance $\boldsymbol{x}$:

$$P(\text{binary label} = 1 \mid \boldsymbol{x}; \boldsymbol{w}), \tag{1}$$

where $\boldsymbol{w}$ is the weight of the binary classifier. To predict the relevance of any label j, the model utilizes the probabilities along the corresponding path:

$$P(\boldsymbol{x} \text{ has label } j) = \prod_{d \in \text{path to } j} P(\text{binary label} = 1 \mid \boldsymbol{x}; \boldsymbol{w}_d),$$

where d refers to an edge between two adjacent nodes along the path to label j, and $\boldsymbol{w}_d$ is the weight of the associated binary classifier. Since the final decision is a product of all probabilities, bias in unreliable estimates can accumulate and degrade the overall prediction. Therefore, each binary classifier must provide accurate probability estimates to ensure reliable predictions. This requirement is further supported by prior work on XML [27].

Among binary classifiers, logistic regression (LR) naturally estimates (1), as the model weights are learned via maximum likelihood estimation. In contrast, support vector machines (SVMs) optimize the margin between classes and do not directly estimate probabilities (1). To bridge this gap, some works [23,31,8] have proposed effective approaches for converting SVM decision values into probabilities. Surprisingly, existing linear methods seldom use these techniques, even though many adopt SVMs as base classifiers, as shown in [24,15,29]. Instead, these models adopt simple transformations on decision values to estimate probabilities. As a result, the effectiveness of these probability estimates remains unclear, and their potential impact on prediction quality deserves further examination.

Through our investigation, we point out that the probability estimation functions used in current tree-based methods [24,15,29] lack strict monotonicity, a key property for producing meaningful probabilities from SVM outputs. We further identify possible difficulties with these methods when adopting Platt scaling [23], the most popular setting for generating SVM probabilistic outputs. Finally, we propose a probability estimation framework that addresses the problematic setting in [24,15,29] and avoids the difficulties encountered in applying Platt scaling in linear methods for XML. This framework improves predictive performance and helps the tree-based linear method become a stronger baseline for XML.

The rest of the paper is organized as follows. Section 2 introduces two linear approaches widely used in XML: one-versus-rest and tree-based methods. Section 3 points out an erroneous setting in the probability estimation of most existing SVM tree-based models. In Section 4, we discuss the difficulties in adopting the approach proposed in

Platt scaling [23], a common setting for generating SVM probabilistic outputs. Section 5 proposes a framework that helps improve estimation reliability. Section 6 demonstrates the effectiveness of the proposed framework through experiments. Finally, Section 7 summarizes our findings. Our approach has been integrated into the multi-label library LibMultiLabel.[3] This work is an extension of the third author's master thesis [3].

2 Linear Methods for XML

To tackle XML with linear models, two methods are widely applied: the one-vs-rest (OVR) method and the tree-based method. We introduce them in this section.

Assume a training set $\{(\boldsymbol{y}_i, \boldsymbol{x}_i)\}_{i=1}^{l}$ with m possible labels, where each feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and each label vector $\boldsymbol{y}_i \in \{-1, 1\}^m$. For any instance–label pair, we define $y_{ij} = 1$ if label j is relevant to instance $\boldsymbol{x}_i$, and $y_{ij} = -1$ otherwise.

2.1 One-vs-Rest (OVR) Method

OVR trains an independent binary classifier for each label $j \in \{1, \ldots, m\}$. The weight vector $\boldsymbol{w}_j \in \mathbb{R}^d$ is learned by solving

$$\min_{\boldsymbol{w}_j} \ \frac{\lambda}{2} \boldsymbol{w}_j^T \boldsymbol{w}_j + \sum_{i=1}^{l} \xi\big(y_{ij}\, \boldsymbol{w}_j^T \boldsymbol{x}_i\big), \tag{2}$$

where $\lambda > 0$ is a hyper-parameter for the regularizer $\boldsymbol{w}_j^T \boldsymbol{w}_j$, and $\xi(\cdot)$ is a loss function. The typical choices for $\xi(\cdot)$ are listed below.

- Logistic loss (LR):
$$\xi_{\text{LR}}(a) = \log\left(1 + \exp(-a)\right); \tag{3}$$
- Hinge loss (SVM-ξ_{L1}):
$$\xi_{\text{L1}}(a) = \max(0,\, 1 - a); \tag{4}$$
- Squared hinge loss (SVM-ξ_{L2}):
$$\xi_{\text{L2}}(a) = \max(0,\, 1 - a)^2. \tag{5}$$

Because (2) must be solved once per label, we end up with m binary optimization problems. After training, we can obtain the prediction scores $[\boldsymbol{w}_1^T \boldsymbol{x}, \cdots, \boldsymbol{w}_m^T \boldsymbol{x}]$ for any given $\boldsymbol{x}$. In the XML setting, where the label space is extremely large, it is common to select labels with the highest scores as the predicted relevant labels for an instance.

2.2 Tree-Based Methods

Since OVR needs to train m classifiers, both time and space complexity of OVR grow linearly in the number of labels, a situation making OVR an inefficient method when m is large. To mitigate the linear growth of the OVR training cost, recent works such

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

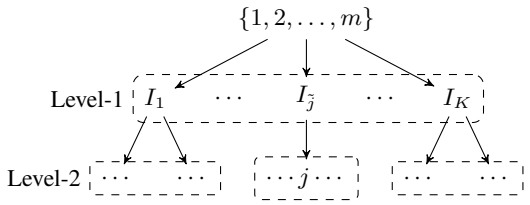


Fig. 1: A two-level tree structure, adapted from [4]. Each dashed rectangular box corresponds to a multi-label classification problem.

as [24,15,29] apply the tree-based method based on a divide-and-conquer idea. This method recursively decomposes the OVR problem into smaller sub-problems by organizing the m labels into a hierarchical tree structure.

In Figure 1, we provide a two-level tree as an example.[4] At level-1, the m labels in the root node are divided into K partitions $I_1, \ldots, I_K$, $K < m$.[5] For easy discussion, we denote these partitions as nodes and tag a meta-label on each of them, leading to a multi-label classification problem with only K meta-labels. Following the OVR setting, we address the multi-label classification problem by training a separate binary classifier for each meta-label using all instances, like (2). As a result, we obtain models $\tilde{\boldsymbol{w}}_1, \ldots \tilde{\boldsymbol{w}}_K$.

Next, we extend the tree from level-1 to level-2. Unlike the multi-label classification problem at level-1, which uses the entire training set, each problem from level-2 onward is trained only on the instances and labels associated with the corresponding parent node at the upper level. In this way, the overall task can be recursively decomposed into smaller multi-label classification problems, and we apply the OVR method to solve each of them at every level, similar to the level-1 setting.

Let us consider the multi-label classification problem for the meta-label $\tilde{j}$ with the label set $I_{\tilde{j}}$ in Figure 1 as an example. To facilitate discussion, we partition the label set so that each level-2 node contains only a single label. Thus, we can define the training set as

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

where $\phi_i : I_{\tilde{j}} \to \{-1, 1\}^{|I_{\tilde{j}}|}$ is a mapping function so that $\phi_i(j) = 1$ if label j is relevant to $\boldsymbol{x}_i$, and $\phi_i(j) = -1$ otherwise. Then, for label $j \in I_{\tilde{j}}$, the corresponding binary classifier with weight vector $\boldsymbol{w}_j$ is learned by solving

$$\min_{\boldsymbol{w}_j} \ \frac{\lambda}{2} \boldsymbol{w}_j^T \boldsymbol{w}_j + \sum_{(\phi_i, \boldsymbol{x}_i) \in D_{\tilde{j}}} \xi\Big(\phi_i(j)\, \boldsymbol{w}_j^T \boldsymbol{x}_i\Big). \tag{6}$$

After training, we estimate the probability of label j for any given instance $\boldsymbol{x}$ by a path from the root to its corresponding leaf node:

$$\text{root} \xrightarrow{\tilde{\boldsymbol{w}}_{\tilde{j}}} \text{ node } I_{\tilde{j}} \xrightarrow{\boldsymbol{w}_j} \text{leaf node (label } j), \tag{7}$$

[4] The mechanism extends to deeper levels, though here we assume two levels.

[5] To partition labels, we can either use information from the training set or additional label knowledge.

where $\tilde{\boldsymbol{w}}_{\tilde{j}}$ is the weight vector corresponding to meta-label $\tilde{j}$. We then apply the idea of probabilistic classifier chains [5] and multiply the probabilities along the path (7):

$$\begin{aligned} & P(\boldsymbol{x} \text{ has label } j \mid \boldsymbol{x}; [\tilde{\boldsymbol{w}}_{\tilde{j}}, \boldsymbol{w}_j]) \\ =& P(\boldsymbol{x} \text{ has label } j \mid \boldsymbol{x}, \boldsymbol{x} \text{ has meta-label } \tilde{j}; \boldsymbol{w}_j) \times P(\boldsymbol{x} \text{ has meta-label } \tilde{j} \mid \boldsymbol{x}; \tilde{\boldsymbol{w}}_{\tilde{j}}). \end{aligned} \quad (8)$$

Using the estimated probabilities for all labels, we can then select labels with the highest probabilities as the predicted relevant labels for instance $\boldsymbol{x}$.

2.3 Probability Estimation for Tree-Based Methods

From Section 2.2, once we train a binary classifier $\boldsymbol{w}$, we need a probabilistic model based on $\boldsymbol{w}$ to estimate the probability for a given instance $\boldsymbol{x}$:

$$P(\text{binary label} = 1 \mid \boldsymbol{x}; \boldsymbol{w}). \quad (9)$$

To facilitate the discussion, we consider any binary set as follows:

$$\tilde{D} = \{(z_i, \boldsymbol{x}_i) \mid z_i = 1 \text{ or } -1\}. \quad (10)$$

In logistic regression, we use LR loss (3) to learn the binary classifier $\boldsymbol{w}$, and the corresponding objective (6) can be rewritten as:

$$\begin{aligned} & \min_{\boldsymbol{w}} \ \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{(z_i,\boldsymbol{x}_i)\in\tilde{D}} \log\left(1 + e^{-z_i\boldsymbol{w}^T\boldsymbol{x}_i}\right) \\ \equiv & \max_{\boldsymbol{w}} \ -\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} + \log \prod_{(z_i,\boldsymbol{x}_i)\in\tilde{D}} \frac{1}{1 + e^{-z_i\boldsymbol{w}^T\boldsymbol{x}_i}}. \end{aligned} \quad (11)$$

In this case, if we do not consider the regularization term, the optimization problem becomes a maximum likelihood estimation by interpreting the term

$$\frac{1}{1 + \exp\left(-z_i\boldsymbol{w}^T\boldsymbol{x}_i\right)} \quad (12)$$

as the estimated probability for the true label z_i. Therefore, when we want to estimate the probability (9), i.e., $z_i = 1$, we can express (12) as an exponential mapping of the logistic regression loss:

$$\frac{1}{1 + \exp\left(-\boldsymbol{w}^T\boldsymbol{x}\right)} = \exp\left(-\xi_{\text{LR}}(\boldsymbol{w}^T\boldsymbol{x})\right). \quad (13)$$

On the other hand, for SVMs, we cannot directly rewrite the optimization objective into a probabilistic form, as logistic regression does in (11). As a result, many studies [23,31,8] have proposed approaches to convert SVM decision values into probabilities in binary classification, and these approaches have been well established for decades. Surprisingly, to the best of our knowledge, when SVM is used as the base binary classifier, none of the existing tree-based linear methods adopt these probability

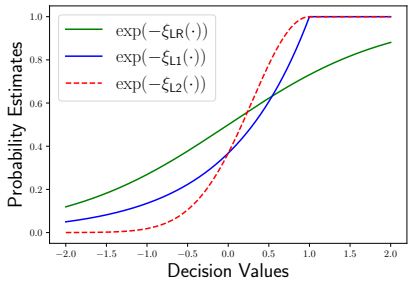


Fig. 2: Probability curves from three estimations.

conversion approaches. Instead, they apply a conversion function similar to (13) used in logistic regression. For example, the works [29,24,15] consider the same exponential form as in (13), but replace the LR loss with SVM loss functions. In other words, past works use a general form:

$$\exp\left(-\xi(\boldsymbol{w}^T\boldsymbol{x})\right), \quad \xi \in \{\xi_{\text{LR}}, \xi_{\text{L1}}, \xi_{\text{L2}}\}, \tag{14}$$

to estimate probabilities for the loss functions.[6]

3 The Issue with Existing Probability Estimation Methods in Tree-Based Models

As introduced in Section 2.3, most existing tree-based linear models apply the approximation $\exp(-\xi(\cdot))$ to convert decision values into probabilities, regardless of which loss function is used. Figure 2 illustrates the probability estimation functions used in [29,24,15] for LR and SVMs. While $\exp(-\xi_{\text{LR}}(\cdot))$ provides a smooth and strictly monotonic mapping, both $\exp(-\xi_{\text{L1}}(\cdot))$ and $\exp(-\xi_{\text{L2}}(\cdot))$ assign a probability of exactly one to any decision value greater than one, because, according to (4) and (5),

$$\max(0, 1-a) = 0 \text{ as } a \geq 1.$$

This behavior leads to a significant impact: the relative order among top-scoring labels is lost after the conversion to probabilities.

To examine the impact of the ranking distortion caused by the probability transformation, we design an experiment based on the simple OVR setting. In addition to using decision values $\boldsymbol{w}^T\boldsymbol{x}$ to rank label relevance, we evaluate rankings based on probabilities computed from (14). Table 1 compares the test performance of these two ranking approaches using precision at top-ranked labels (P@1, P@3, and P@5)[7] on four XML

[6] The works [24,15] used (14) in their code to estimate (9) though they did not explicitly state this in their papers.

[7] Section 6.1 defines the metrics P@1, P@3, and P@5.

	P@1			P@3			P@5		
EUR-Lex-4K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach $\boldsymbol{w}^T\boldsymbol{x}$	81.71	80.05	81.14	68.33	67.90	68.49	57.11	56.98	57.16
The approach (14)	0.00	−1.08	−0.52	0.00	−0.04	−0.06	0.00	0.00	0.00
Mimic-iii-8K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach $\boldsymbol{w}^T\boldsymbol{x}$	85.11	85.53	86.15	77.44	79.11	79.12	70.66	72.86	72.57
The approach (14)	0.00	−0.98	−0.15	0.00	−0.12	−0.04	0.00	−0.02	−0.01
Wiki10-31K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach $\boldsymbol{w}^T\boldsymbol{x}$	84.95	84.92	84.63	72.62	76.56	74.67	63.26	68.06	65.83
The approach (14)	0.00	−0.62	−0.47	0.00	−0.04	−0.02	0.00	0.00	0.00
AmazonCat-13K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach $\boldsymbol{w}^T\boldsymbol{x}$	92.95	93.23	93.19	78.55	78.78	78.90	63.84	63.52	64.14
The approach (14)	0.00	−1.52	−0.56	0.00	−0.14	−0.05	0.00	−0.02	0.00

Table 1: Comparison of P@1, P@3, and P@5 for predictions based on decision values $\boldsymbol{w}^T\boldsymbol{x}$ and probability estimates (14), with three loss functions (ξ_{LR}, ξ_{L1}, ξ_{L2}) in the OVR setting. For (14), we report relative differences compared to the approach of using decision values $\boldsymbol{w}^T\boldsymbol{x}$.

data sets.[8] Our goal is to examine the impact of using decision values versus probability estimates (row 1 versus row 2). Thus, we simply set $\lambda = 1$ for all loss functions. Since hyper-parameters are not tuned in this setting, it is not suitable to compare the results across different loss functions (i.e., across columns). Instead, a more rigorous comparison of these loss functions will be presented in Section 6.

For logistic regression, $\exp\left(-\xi_{\text{LR}}(\cdot)\right)$ preserves the label order, so the approaches based on decision values and probability estimates yield identical performance. In contrast, when SVM loss functions ξ_{L1} and ξ_{L2} are used, ranking labels based on probability estimates leads to performance drops, with a more pronounced decrease under ξ_{L1}.

Recall from Figure 2 that (14) does not preserve the order among labels with decision values above one, so the model cannot prioritize the most likely labels. Since P@1 considers only the top label, this ambiguity causes a significant drop in performance when probability estimates are used. As we move to P@3 and P@5, more labels are included in the evaluation, and the gap between decision values and probability estimates gradually narrows.

Although Table 1 focuses solely on the simple OVR setting, we already observe clear performance differences. Tree-based models represent a more complex setting, as they rely on probabilistic classifier chains like (8) to make predictions. In such cases, estimation errors may propagate through probability multiplication and lead to a more severe impact. In fact, prior work [27] has highlighted the importance of accurate

[8] Since training OVR models on XML data sets is expensive, we select four smaller ones. Details of these data sets are provided in Appendix A.1.

probability estimation in XML problems. However, when SVMs are used in tree-based linear methods, existing implementations still adopt a simple probability conversion as in (14), which introduces non-negligible estimation errors and must be properly addressed.

4 Why Most Existing SVM Tree-Based Methods Do Not Consider Platt Scaling for Probability Estimation

In Section 3, we pointed out that the probability estimation (14) used in most existing SVM tree-based methods [24,15,29] is problematic. However, many studies [23,31,8] have developed ways to produce probabilistic outputs from SVMs. Among them, the highly cited approach, Platt scaling [23], implemented in LIBSVM [2], is commonly used in tasks that need posterior class probabilities. They propose a parametric model that converts the decision value $\boldsymbol{w}^T\boldsymbol{x}$ into the probability[9] of the true label $z = 1$ (with $z \in \{1, -1\}$):

$$P(z = 1 \mid \boldsymbol{w}^T\boldsymbol{x}; A, B) = \frac{1}{1 + e^{A\boldsymbol{w}^T\boldsymbol{x}+B}}, \tag{15}$$

where A and B are learnable parameters. Surprisingly, the works [24,15,29] do not adopt this well-known method, even though tree-based methods rely heavily on probability estimates. Instead, they use a simple approximation (14) that has obvious problems. Given this, readers of past tree-based XML works must ask why these studies did not apply well-established binary probabilistic outputs such as Platt scaling. Therefore, we conducted a detailed investigation and found that adopting Platt scaling in the tree-based setting presents several difficulties. Specifically, we identify two main challenges:

- Estimating parameters in Platt scaling makes training too expensive for XML tasks.
- Estimating probabilities for each node independently within the tree can be suboptimal.

We elaborate on these challenges in Section 4.1 and Section 4.2, respectively.

4.1 Expensive Training Cost When Applying Platt Scaling in XML

Given any binary data set $\tilde{D}$ defined in (10), Platt scaling estimates the parameters A and B in (15) by maximizing the log-likelihood:[10]

$$\max_{A,B} \sum_{i=1}^{|\tilde{D}|} \log P(z_i \mid \boldsymbol{w}^T\boldsymbol{x}_i; A, B). \tag{16}$$

At first glance, solving (16) may seem simple and inexpensive, as it only requires optimizing two parameters (A, B), which is far cheaper than training a binary classifier. However, to avoid overfitting, an important setting in Platt scaling is to generate decision

[9] Note that $P(z = -1 \mid \boldsymbol{w}^T\boldsymbol{x}; A, B) = 1 - P(z = 1 \mid \boldsymbol{w}^T\boldsymbol{x}; A, B)$.

[10] Platt scaling [23] adopts a slightly modified formulation of (16) to prevent overfitting.

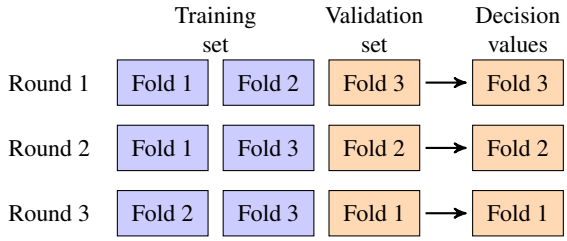


Fig. 3: An illustration of how Platt scaling collects decision values in a three-fold cross-validation setting. In each round, the purple folds are used to train the SVM model, and the orange fold is used to generate decision values. Finally, we collect decision values from all three folds to solve the problem (16).

values through cross-validation. For easier discussion, Figure 3 illustrates the three-fold cross-validation process, where the training data is split into three folds. In each round, we hold out one fold as the validation set and train an SVM model using the remaining two folds. After three rounds, we collect the decision values from all validation folds and use them to estimate the parameters A and B in (15) by solving (16).

The procedure of using F-fold cross-validation is widely accepted when applying Platt scaling to binary SVM classification, even though the training time increases by nearly F times. While such an increase in training time is generally tolerable for small or medium-sized data sets in binary classification, XML models already require a long training time due to the large number of labels. Consequently, a further increase by F times becomes unaffordable, making this strategy less practical for XML.

4.2 Limitations of Estimating Probabilities Independently

Another challenge lies in the fact that Platt scaling is originally designed to provide a reasonable probabilistic estimate for a single binary classification task, whereas XML requires aggregating the outputs of multiple binary classifiers to generate the final prediction. As a result, directly applying Platt scaling to each individual classifier can be problematic. In the following, we use (8), the product of probabilities along the tree path, to illustrate the potential issues.

To determine whether an instance $\boldsymbol{x}$ has label j, our goal in (8) is to accurately estimate

$$P(\boldsymbol{x} \text{ has label } j \mid \boldsymbol{x}; [\tilde{\boldsymbol{w}}_{\tilde{j}}, \boldsymbol{w}_j]). \tag{17}$$

Notably, tree-based methods do not estimate (17) directly. Instead, we train separate binary classifiers (i.e., $\boldsymbol{w}_j$ and $\tilde{\boldsymbol{w}}_{\tilde{j}}$) to estimate the two conditional probabilities

$$P(\boldsymbol{x} \text{ has label } j \mid \boldsymbol{x}, \boldsymbol{x} \text{ has meta-label } \tilde{j}; \boldsymbol{w}_j), \tag{18}$$

and

$$P(\boldsymbol{x} \text{ has meta-label } \tilde{j} \mid \boldsymbol{x}; \tilde{\boldsymbol{w}}_{\tilde{j}}). \tag{19}$$

At first glance, Platt scaling appears to be a natural choice for estimating the probabilities (18) and (19). However, directly applying Platt scaling may raise two concerns. First, estimating (18) and (19) separately does not necessarily yield a good estimate of (17), as the quality of (17) depends on their combination rather than on either component alone. Therefore, an estimation strategy that considers (18) and (19) jointly is more desirable. Second, as discussed in Section 2.2, the training set associated with each node becomes smaller as the tree grows deeper. Since (18) is estimated at a leaf node, the available training data can be extremely limited. Consequently, applying Platt scaling to (18) may produce less reliable probability estimates and be more prone to overfitting.

5 A Suitable Framework for Probability Estimation in Tree-Based Methods

Based on the discussions in Sections 3 and 4, we need a probability estimation framework that fixes the error in Section 3 and avoids the issues discussed in Section 4. To this end, we expect the framework to satisfy the following properties:

(a) The probability function should be strictly increasing to preserve the ranking of decision values, a property that both $\exp\left(-\xi_{\text{L1}}\left(\cdot\right)\right)$ and $\exp\left(-\xi_{\text{L2}}\left(\cdot\right)\right)$ used in existing works violate.
(b) The approach should maintain a computational cost comparable to existing works described in Section 3. For example, we should avoid the cross-validation process needed in Platt scaling.
(c) The approach should optimize global performance rather than focusing on the probabilistic estimation of individual binary problems.

Properties (a) and (c) focus on model performance, while Property (b) concerns training cost. We address these aspects in the following subsections.

5.1 Ensuring Strict Monotonicity and Optimizing Global Performance

We first consider Properties (a) and (c). Note that the probabilistic model (15) used by Platt scaling already satisfies Property (a), as this function is strictly increasing. However, Platt scaling estimates (A, B) for each binary problem without considering global performance, so the approach does not meet Property (c). To address this issue, we novelly treat each pair (A, B) as hyper-parameters instead of trainable parameters used in fitting (16). In other words, suppose the tree-based linear model involves t binary models, we can construct the following hyper-parameter set:

$$(A_1, B_1), (A_2, B_2), \cdots, (A_t, B_t). \tag{20}$$

This novel aspect of using (20) offers two advantages. First, we can select the hyper-parameters (A_j, B_j) for all j to optimize global performance. Second, during hyper-parameter search, we can evaluate the validation performance for any given (A_j, B_j) across all j without training additional models. We now illustrate how these advantages work in practice.

Let us consider tuning the regularization parameter λ as an example. In this setting, we aim to optimize the global performance on the validation set with respect to both λ and (A_j, B_j) for all j. Since (A_j, B_j) are not involved in training the weights $\boldsymbol{w}$, we only need to train the model once for a given value of λ. After training and calculating the decision values $\boldsymbol{w}^T\boldsymbol{x}$, we can directly compute the probabilistic outputs by applying any configuration of (A_j, B_j) in (15). In this way, we can evaluate the validation performance across all possible configurations without any extra model training.

5.2 Reducing Training Cost

However, even though we do not need to train additional models for any (A_j, B_j), we still need to compute (15) to obtain probabilistic outputs. In XML tasks with thousands to millions of labels, the search space formed by (A_j, B_j) across all j becomes extremely large, making it impractical to evaluate (15) for all possible configurations and perform hyper-parameter search efficiently. As a result, we still lack a method satisfying Property (b) and Property (c) simultaneously.

To reduce the computational cost while optimizing global performance, we novelly propose a simple yet effective design: we remove all hyper-parameter $B_j, \forall j$ and let all binary probabilistic models share the same hyper-parameter A. This leads to the following form:

$$\frac{1}{1 + \exp(A\boldsymbol{w}^T\boldsymbol{x})}. \tag{21}$$

Although we could similarly replace all $B_j, \forall j$ with a single shared hyper-parameter B, we choose not to do so in order to avoid performing hyper-parameter search over all combinations of (A, B).

Importantly, our novel design (21) preserves the core advantage of the setting (20), as the global performance can still be directly optimized, thereby satisfying Property (c). At the same time, our approach has significantly reduced the hyper-parameter search space, which makes the search process more efficient and meets the requirement of Property (b). To support this claim, we further demonstrate in Appendix A.5 that tuning over (λ, A) requires almost the same computational cost as tuning λ alone. Note that while the idea in Section 5.1 similarly avoids model training by evaluating different (A_j, B_j) configurations on cached decision values, the search space grows dramatically with the number of binary classifiers. By using a single shared hyper-parameter A, (21) keeps the search space small while retaining the ability to optimize global performance. As a result, our approach (21) successfully fulfills all three desired properties: (a), (b), and (c).

6 Experiments

This section compares our approach (21) with the approach using (14). We first introduce the evaluation metrics and experimental settings in Section 6.1. Then, we present the experimental results and our analysis in Section 6.2. Finally, we demonstrate in Section 6.3 that our probability estimation (21) enables tree-based linear methods to serve as stronger baselines for XML problems.

Data Set	# instances l	# features d	# labels m
EUR-Lex-4K	15,449	186,104	3,956
Mimic-iii-8K	51,095	140,767	8,685
Wiki10-31K	14,146	104,374	30,938
AmazonCat-13K	1,186,239	203,882	13,330
Amazon-670K	490,449	135,909	670,091

Table 2: Statistics of training data sets.

6.1 Evaluation and Experimental Setups

We consider XML on document data. Table 2 summarizes the statistics of the experimental data sets, with additional details provided in Appendix A.1. For handling such data with linear models, highly sparse bag-of-words (BOW) are commonly used to generate feature representations for both training and evaluation. We adopt a popular BOW variant, uni-gram TF-IDF features [20,14], to construct the feature representation for each instance $\boldsymbol{x}$.

For XML tasks, we typically use ranking-based metrics to evaluate model performance. Following previous XML works [24,28,15,29], we adopt precision at k (P@k) for $k = 1, 3$, and 5, as these metrics are among the most commonly used evaluation metrics. During evaluation, we first calculate the proportion of correct labels among the top-k predictions for each instance $\boldsymbol{x}$. We then average these proportions across all test instances to obtain the corresponding P@k.

Across all experiments, when our approach (21) is used, we tune the hyper-parameter A and the regularization parameter λ; otherwise, we tune λ only. The details are in Appendix A.2. In addition, Appendices A.3 and A.4 examine the sensitivity of A and λ to model performance, respectively. Besides, although our approach (21) introduces one more hyper-parameter, searching over (λ, A) requires almost the same computational cost as tuning λ alone, as discussed in Section 5 and Appendix A.5.

6.2 Comparison of Probability Estimations in SVMs

We propose the approach (21) to address the issues of applying Platt scaling [23] in tree-based methods. To evaluate the effectiveness of our approach, we consider LR, SVM-ξ_{L1}, and SVM-ξ_{L2}, and provide the test results in Table 3. The key findings from these results are summarized below.

First, our approach (21) improves SVM results compared to using (14) in almost all cases. This implies that our approach consistently corrects the probability estimation errors introduced by using (14). In particular, when SVM-ξ_{L1} is used, applying (14) leads to much worse performance compared to LR and SVM-ξ_{L2}. However, when our approach (21) is applied, SVM-ξ_{L1} shows a surprisingly large improvement, and the performance of SVM-ξ_{L1} becomes comparable to LR and SVM-ξ_{L2}. Therefore, our approach effectively corrects the misuse of (14) found in existing works.

Secondly, the results of LR are competitive with those of SVM-ξ_{L2} under (14). This finding differs from the conclusion of [24], as they did not perform hyper-parameter

	P@1			P@3			P@5		
EUR-Lex-4K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach (14)	81.37	81.58	82.35	68.37	68.56	**69.26**	57.37	56.53	58.01
Our approach (21)	Un-nec.	**82.54**	**82.43**	Un-nec.	**69.26**	69.20	Un-nec.	**57.80**	**58.12**
Wiki10-31K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach (14)	84.28	84.40	84.43	73.14	73.28	73.52	64.15	63.36	64.74
Our approach (21)	Un-nec.	**85.57**	**85.47**	Un-nec.	**75.20**	**74.19**	Un-nec.	**66.32**	**65.32**
Mimic-iii-8K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach (14)	84.88	83.39	84.13	77.19	77.28	77.63	71.03	71.56	71.53
Our approach (21)	Un-nec.	**85.17**	**85.32**	Un-nec.	**77.70**	**77.68**	Un-nec.	**72.07**	**71.58**
AmazonCat-13K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach (14)	94.11	91.67	93.69	79.52	79.00	79.71	64.51	63.98	64.73
Our approach (21)	Un-nec.	**93.16**	**94.01**	Un-nec.	**79.36**	**79.77**	Un-nec.	**64.23**	**64.74**
Amazon-670K									
	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}	ξ_{LR}	ξ_{L1}	ξ_{L2}
The approach (14)	43.53	39.49	43.67	38.54	34.73	38.48	34.92	31.32	34.78
Our approach (21)	Un-nec.	**41.49**	**43.91**	Un-nec.	**36.85**	**38.79**	Un-nec.	**33.47**	**35.09**

Table 3: Test results of tree-based methods. Both the approach (14) and our approach (21) consider three linear classifiers: logistic regression (LR), SVM-ξ_{L1}, and SVM-ξ_{L2}, where the loss functions ξ_{LR}, ξ_{L1}, and ξ_{L2} are used, respectively. Moreover, we tune the hyper-parameters with cross-validation. All the values are multiplied by 100. The better results of SVM-ξ_{L1} and SVM-ξ_{L2} under the two estimation approaches are denoted with bold text. The word "Un-nec." represents unnecessary.

search and used a fixed $\lambda = 0.1$ for LR and $\lambda = 1$ for SVM-ξ_{L2}. This highlights the importance of λ selection; see Section A.4 for a more detailed discussion.

Due to the competitive performance of LR, naturally we ask if our work provides no additional benefit, and people only need to use LR. The answer is no. This is because training an SVM is generally faster than LR [7], partly due to the fact that LR requires many expensive exponential and logarithmic operations [30]. In addition, recent work [18] has shown that when using sparse features such as TF-IDF in linear methods, logistic regression produces fully dense models. In contrast, SVMs can maintain similar performance with much smaller model sizes by benefiting from feature sparsity. Our improvement on probability estimation further makes SVM-based tree models a valuable choice for large-scale applications.

6.3 Our Approach Makes the Tree-Based Linear Method a Stronger Baseline

It is worth emphasizing that the tree-based linear method is highly efficient in both training and inference, making it an important baseline for large-scale applications. Our approach in (21) maintains the efficiency of tree-based linear methods while signifi-

	P@1		P@3		P@5	
Mimic-iii-8K						
AttentionXML	86.80		80.03		73.71	
	ξ_{L1}	ξ_{L2}	ξ_{L1}	ξ_{L2}	ξ_{L1}	ξ_{L2}
The approach (14)	−3.41	−2.67	−2.75	−2.40	−2.15	−2.18
Our approach (21)	**−1.63**	**−1.48**	**−2.33**	**−2.35**	**−1.64**	**−2.13**
Amazon-670K						
AttentionXML	44.36		39.58		35.94	
	ξ_{L1}	ξ_{L2}	ξ_{L1}	ξ_{L2}	ξ_{L1}	ξ_{L2}
The approach (14)	−4.87	−0.69	−4.85	−1.10	−4.62	−1.16
Our approach (21)	**−2.87**	**−0.45**	**−2.73**	**−0.79**	**−2.47**	**−0.85**

Table 4: Relative P@1, P@3, and P@5 between AttentionXML and tree-based linear methods on two data sets. That is, the results of linear methods are subtracted from those of AttentionXML. The closest results of SVM-ξ_{L1} and SVM-ξ_{L2} to AttentionXML under two estimation approaches are denoted with bold text.

cantly closing their gap to powerful nonlinear methods. Table 4 presents the relative performance gaps of the existing approach (14) and our approach (21) with respect to AttentionXML [28], which is a well-known nonlinear method. By examining the results, we observe that our enhanced approach (21) consistently reduces the gap between the tree-based linear method and AttentionXML across all metrics. These results suggest that the commonly used tree-based linear baseline has been underestimated due to the poor probability estimation with (14). With our enhanced estimation, the tree-based linear method serves as a much stronger baseline that future methods should be compared against.

7 Conclusions

In this paper, we identify the damage from the unsuitable SVM probability estimation (14) in most works of tree-based linear methods for XML. We further clarify the critical issue of why those works do not consider the well-studied SVM probabilistic outputs in (15). Subsequently, we list the properties for a suitable estimation and propose the approach (21). Experiments show that our proposed probability estimation consistently improves the performance of tree-based linear methods, which, as a result, become stronger baselines for XML classification.

Acknowledgments. We would like to thank Hung-Chih Chiang for his assistance in integrating our proposed approach into the XML library LibMultiLabel. This work was supported in part by National Science and Technology Council of Taiwan grants NSTC-113-2222-E-002-005MY3 and NSTC-114-2634-F-002-007.

References

1. Chalkidis, I., Jana, A., Hartung, D., Bommarito, M., Androutsopoulos, I., Katz, D., Aletras, N.: LexGLUE: A benchmark dataset for legal language understanding in English. In: Proceedings

of the 60th Annual Meeting of the Association for Computational Linguistics. pp. 4310–4330 (2022)
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**(3), 27:1–27:27 (2011), software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
3. Chen, G.T.: Probability Estimation in Tree-Based Linear Methods for Extreme Multi-Label Learning. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University (2026)
4. Chen, S.W., Lin, C.J.: Random label forests: An ensemble method with label subsampling for extreme multi-label problems. In: Findings of the Association for Computational Linguistics: EMNLP. pp. 14107–14119 (2024), https://www.csie.ntu.edu.tw/~cjlin/papers/random_label_forests/main.pdf
5. Dembczyński, K., Cheng, W., Hüllermeier, E.: Bayes optimal multilabel classification via probabilistic classifier chains. In: Proceedings of the 27th International Conference on Machine Learning. pp. 279–286 (2010)
6. Engel, E.: Balancing Performance and Usage Cost: A Comparative Study of Language Models for Scientific Text Classification. Master's thesis, KTH, Mathematical Statistics (2023)
7. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: a library for large linear classification. Journal of Machine Learning Research **9**, 1871–1874 (2008), http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf
8. Franc, V., Zien, A., Schölkopf, B.: Support vector machines as probabilistic models. In: Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML). pp. 665–672 (2011)
9. Galke, L., Diera, A., Lin, B.X., Khera, B., Meuser, T., Singhal, T., Karl, F., Scherp, A.: Are we really making much progress in text classification? a comparative review. arXiv preprint arXiv:2204.03954 (2022)
10. Hsieh, C.J., Chang, K.W., Lin, C.J., Keerthi, S.S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML) (2008), http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf
11. Jasinska, K., Dembczyński, K., Busa-Fekete, R., Pfannschmidt, K., Klerx, T., Hüllermeier, E.: Extreme F-measure maximization using sparse probability estimates. In: Proceedings of The 33rd International Conference on Machine Learning (ICML). pp. 1435–1444 (2016)
12. Jiang, T., Wang, D., Sun, L., Yang, H., Zhao, Z., Zhuang, F.: LightXML: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 7987–7994 (2021)
13. Johnson, A.E., Pollard, T.J., Shen, L., Lehman, L.W.H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L.A., Mark, R.G.: MIMIC-III, a freely accessible critical care database. Scientific Data **3**(1), 1–9 (2016). https://doi.org/10.1038/sdata.2016.35
14. Jones, K.S.: A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation **28**(1), 11–20 (1972)
15. Khandagale, S., Xiao, H., Babbar, R.: Bonsai: diverse and shallow trees for extreme multi-label classification. Machine Learning **109**, 2099–2119 (2020). https://doi.org/10.1007/s10994-020-05888-2
16. Kharbanda, S., Banerjee, A., Schultheis, E., Babbar, R.: CascadeXML: Rethinking transformers for end-to-end multi-resolution training in extreme multi-label classification. In: Advances in Neural Information Processing Systems (Neurips). vol. 35, pp. 2074–2087 (2022)
17. Lin, H.Z., Liu, C.H., Lin, C.J.: Exploring space efficiency in a tree-based linear model for extreme multi-label classification. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 16245–16260 (2024),

https://www.csie.ntu.edu.tw/~cjlin/papers/multilabel_tree_model_size/multilabel_tree_model_size.pdf
18. Lin, H.Z., Lu, Z.B., Chen, S.W., Liu, C.H., Lin, C.J.: On the weight density of l2-regularized linear classification and regression. In: Proceedings of the Twenty-Ninth Annual Conference on Artificial Intelligence and Statistics (AISTATS) (2026), https://www.csie.ntu.edu.tw/~cjlin/papers/dual-space/dual-space.pdf
19. Lin, Y.C., Chen, S.A., Liu, J.J., Lin, C.J.: Linear classifier: An often-forgotten baseline for text classification. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL). pp. 1876–1888 (2023), https://www.csie.ntu.edu.tw/~cjlin/papers/text_classification_baseline/text_classification_baseline.pdf, short paper
20. Luhn, H.P.: The automatic creation of literature abstracts. IBM Journal of Research and Development **2**(2), 159–165 (1958)
21. Mullenbach, J., Wiegreffe, S., Duke, J., Sun, J., Eisenstein, J.: Explainable prediction of medical codes from clinical text. In: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL). pp. 1101–1111 (2018). https://doi.org/10.18653/v1/N18-1100
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
23. Platt, J.C.: Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In: Smola, A., Bartlett, P., Schölkopf, B., Schuurmans, D. (eds.) Advances in Large Margin Classifiers. MIT Press, Cambridge, MA (2000)
24. Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., Varma, M.: Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In: Proceedings of the 2018 World Wide Web Conference (WWW). pp. 993–1002 (2018)
25. Qaraei, M., Khandagale, S., Babbar, R.: Why state-of-the-art deep learning barely works as good as a linear classifier in extreme multi-label text classification. In: Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN). pp. 223–228 (2020)
26. Tsoumakas, G., Katakis, I., Vlahavas, I.: Effective and efficient multilabel classification in domains with large number of labels. In: Proceedings of ECML/PKDD 2008 Workshop on Mining Multidimensional Data (2008)
27. Wydmuch, M., Jasinska, K., Kuznetsov, M., Busa-Fekete, R., Dembczynski, K.: A no-regret generalization of hierarchical softmax to extreme multi-label classification. In: Advances in Neural Information Processing Systems. vol. 31 (2018)
28. You, R., Zhang, Z., Wang, Z., Dai, S., Mamitsuka, H., Zhu, S.: AttentionXML: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In: Advances in Neural Information Processing Systems. vol. 32 (2019)
29. Yu, H.F., Zhong, K., Zhang, J., Chang, W.C., Dhillon, I.S.: PECOS: Prediction for enormous and correlated output spaces. Journal of Machine Learning Research **23**(98), 1–32 (2022)
30. Yuan, G.X., Ho, C.H., Lin, C.J.: Recent advances of large-scale linear classification. Proceedings of the IEEE **100**(9), 2584–2603 (2012), http://www.csie.ntu.edu.tw/~cjlin/papers/survey-linear.pdf
31. Zadrozny, B., Elkan, C.: Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (2002)

A Supplementary Materials

A.1 Data Sets

Our experiments consider five real-world XML data sets: EUR-Lex-4K, Mimic-iii-8K, AmazonCat-13K, Wiki10-31K, and Amazon-670K. Except for Mimic-iii-8K, these datasets are obtained from the LIBSVM Data website.[11] We preprocess all data sets into sparse TF-IDF representations for document classification. The data sources are listed below.

- EUR-Lex-4K: training set[12] and test set.[13]
- The raw data of Mimic-iii-8K is provided from [13]. We follow [21] to generate the training and test sets, which are then transformed into TF-IDF format using the scikit-learn [22] function with the default settings:
  - **lowercase:** True.
  - **analyzer:** "word."
  - **stop_words:** None.
  - **token_pattern:** r"(?u)|b|w|w+|b".
  - **ngram_range:** (1, 1).
  - **max_df:** 1.0.
  - **min_df:** 1.
  - **max_features:** None.
  - **vocabulary:** None.
  - **binary:** False.
  - **norm:** "l2."
  - **use_idf:** True.
  - **smooth_idf:** True.
  - **sublinear_tf:** False.
- AmazonCat-13K: training set[14] and test set.[15]
- Wiki10-31K: training set[16] and test set.[17]
- Amazon-670K: training set[18] and test set.[19]

[11] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

[12] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/eurlex_tfidf_train.svm.bz2

[13] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/eurlex_tfidf_test.svm.bz2

[14] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/AmazonCat-13K_tfidf_train_ver1.svm.bz2

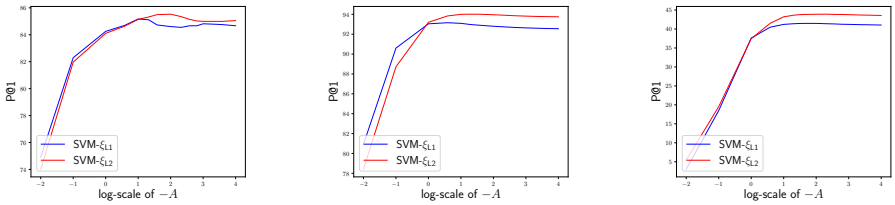
[15] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/AmazonCat-13K_tfidf_test_ver1.svm.bz2

[16] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/wiki10_31k_tfidf_train.svm.bz2

[17] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/wiki10_31k_tfidf_test.svm.bz2

[18] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/Amazon-670K_tfidf_train_ver1.svm.bz2

[19] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/Amazon-670K_tfidf_test_ver1.svm.bz2



(a) Mimic-iii-8K. (b) AmazonCat-13K. (c) Amazon-670K.

Fig. 4: Sensitivity analysis of A over three data sets on P@1. Note that the x-axis shows $\log_2(-A)$ rather than $\log_2(A)$, since $A < 0$.

A.2 Hyper-Parameter Search

It is well known that hyper-parameters are important in machine learning applications. For tree-based methods, common hyper-parameters include K for label clustering and the depth of the tree. In our experiments, we use the default settings provided by the XML library LibMultiLabel[20] for these hyper-parameters, except for the regularization parameter λ. The reason is that we found λ plays a crucial role in model performance. We will discuss this in more detail in Section A.4.

To perform hyper-parameter search in our experiments, we consider five-fold cross-validation to optimize hyper-parameters with respect to each evaluation metric. After determining the optimal hyper-parameters, we retrain the tree model on the full training set and then evaluate its performance on the test set.

For our approach using (21) and the approach using (14), we select λ from the candidates:

$$\left\{2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}\right\}.$$

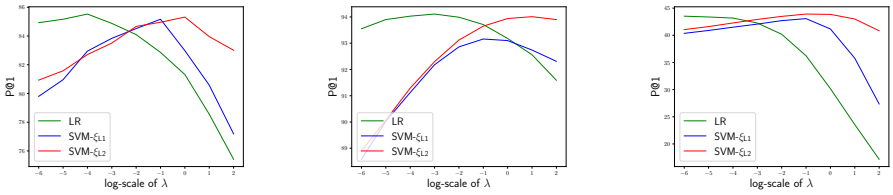
As our approach (21) introduces an additional hyper-parameter A, we consider the candidate set:

$$\{-16, -12, -10, -8, -7, -6, -5, -4, -3, -2.5, -2, -1.5, -1, -0.5, -0.25\}.$$

A.3 Sensitivity of the Hyper-Parameter A

In this work, our approach (21) introduces a hyper-parameter A to estimate the probability of SVM outputs. This subsection investigates the sensitivity of model performance to the hyper-parameter A. Since the loss function ξ_{LR} does not require A for probability estimation, we focus on ξ_{L1} and ξ_{L2}. For each value of A, we select the λ that gives the best test performance and record the results across different A. Figure 4 presents the P@1 curves on three data sets to illustrate this comparison. For ease of presentation, we plot the results against $\log_2(-A)$. We first note that the performance is low when $-A$ is very small. As $-A$ increases, the performance improves rapidly. However, when

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



(a) Mimic-iii-8K. (b) AmazonCat-13K. (c) Amazon-670K.

Fig. 5: Sensitivity analysis of λ over three data sets on P@1. Note that the x-axis shows $\log_2(\lambda)$.

$-A$ becomes sufficiently large, the changes vary only slightly and the curves remain stable. This indicates that the model is less sensitive to large values of $-A$. Therefore, in practice, we do not need to consider many possible values of A. Instead, testing only a few sufficiently large values of $-A$ is enough, or equivalently, a few sufficiently small values of A. For example, from Figure 4, we observe that setting $-A \geq 2$ (i.e., $A \leq -2$) already yields competitive results across datasets.

A.4 Sensitivity of the Hyper-Parameter λ

In Section 6.2, we mentioned that LR can achieve comparable performance to SVMs when hyper-parameter search is performed, an observation different from that in [24]. To better understand the impact of λ, Figure 5 shows the relationship between P@1 and the regularization parameter λ for ξ_{LR}, ξ_{L1}, and ξ_{L2}. For ξ_{L1} and ξ_{L2}, we use the A that gives the best test P@1 and then evaluate across different λ, while ξ_{LR} does not involve A and is reported directly over λ. The P@1 curves clearly indicate that model performance is highly sensitive to the choice of λ. Take Amazon-670K as an example. The difference in P@1 between the best and worst λ values for ξ_{LR} reaches up to 25%. Our experiments may be the first to systematically check the sensitivity of λ on tree-based linear methods, a step that has been largely neglected in prior works. Building on the importance of tuning λ and the demonstrated benefits of incorporating hyper-parameter A in our approach, we can jointly optimize A and λ to further enhance model performance with comparable computational cost.

A.5 Why the Cost of Tuning (λ, A) Can Be Similar to Tuning λ Alone

In Appendix A.4, we have shown that the regularization parameter λ strongly affects performance. Therefore, we should always consider tuning λ in tree-based linear methods. In a typical hyper-parameter search, we train a model on the training set and evaluate the model on a validation set. Importantly, whether we tune (λ, A) jointly or λ alone, only λ is used during training. The additional hyper-parameter A in (21) acts only on decision values $\boldsymbol{w}^T\boldsymbol{x}$ and does not affect the learned weights $\boldsymbol{w}$ at any node in the tree-based linear model.

To clarify why the cost remains comparable, we now detail the cross-validation procedure for tuning (λ, A). For each λ, we perform the following steps on every training/validation split in cross-validation:

(a) **Model training.** Train the tree-based linear model with the given λ. For each binary linear classifier involved in a tree-based linear model, as noted in [10], the training cost is approximately

$$O\left(\mathrm{nnz}(\{\boldsymbol{x} \mid \boldsymbol{x} \in D_{\text{tr}}\}) \cdot \#\text{ training epochs}\right), \tag{22}$$

where D_{tr} denotes the training set and nnz represents "number of non-zeros."

(b) **Decision value caching.** Compute and cache the decision values $\boldsymbol{w}^T\boldsymbol{x}$ on the validation set D_{val} across all nodes (i.e., classifiers). The cost of computing the sparse dot products for one binary classifier is

$$O\left(\mathrm{nnz}\left(\{\boldsymbol{x} \mid \boldsymbol{x} \in D_{\text{val}}\}\right)\right).$$

(c) **Probability estimation.** For each candidate A, transform the cached decision values into probabilities using (21), and then aggregate these probabilities to produce predictions. Since $\boldsymbol{w}^T\boldsymbol{x}$ is already computed in Step (b), the total computational cost for each binary classifier is only

$$O(|D_{\text{val}}|). \tag{23}$$

Notably, Step (c) involves only post-processing and does not require retraining the model. We simply reuse the cached scores obtained in Step (b) and apply lightweight computations to evaluate different values of A. As shown in (22) and (23), the cost of sweeping over A is significantly smaller than the cost of model training. Therefore, tuning (λ, A) avoids the computational bottleneck in Step (a), and the total cost remains similar to tuning λ alone.