On the Influence of Hyperparameters in Tree-Based Linear Methods for Extreme Multi-Label Text Classification: Insights for Efficient and Effective Search

Kuan-Ting Chen*
National Taiwan University
Taipei, Taiwan
andyngewcz2d@gmail.com

Hung-Chih Chiang
National Taiwan University
Taipei, Taiwan
r13922189@ntu.edu.tw

Chih-Jen Lin
National Taiwan University
Taipei, Taiwan
Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, United Arab Emirates
cjlin@csie.ntu.edu.tw

Abstract

Extreme multi-label text classification (XMTC) aims to assign each document the most relevant labels from a large label set. Tree-based linear methods based on bag-of-words features have proven to be strong baselines for XMTC in practice. These methods construct a tree structure over the label space to enable fast training and prediction. However, to search for the best model, the large hyperparameter search space makes the process difficult and time-consuming. This study innovatively analyzes the hyperparameters of tree-based linear methods. Based on our findings, we suggest an efficient and effective guideline. Experiments show that our proposed guideline leads to consistent improvements across datasets, thereby strengthening tree-based linear methods as a stronger XMTC baseline. Our proposed setting has been integrated into a multi-label library to facilitate practical usage.

CCS Concepts

• **Computing methodologies** → **Supervised learning by classification**.

Keywords

Extreme multi-label classification, Tree-based linear methods, Hyperparameter search, Text classification

ACM Reference Format:
Kuan-Ting Chen, Hung-Chih Chiang, and Chih-Jen Lin. 2026. On the Influence of Hyperparameters in Tree-Based Linear Methods for Extreme Multi-Label Text Classification: Insights for Efficient and Effective Search. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM '26), November 07–11, 2026, Rome, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3799682.3840104

1 Introduction

In industrial applications, many systems need to assign multiple labels to a given document. For example, e-commerce platforms classify product descriptions to better capture user intent for search and recommendation [22], and similar tagging needs arise across various content domains [20, 30]. As these services manage massive text collections and thousands or even millions of possible labels, selecting relevant labels efficiently becomes a major challenge. This task is known as Extreme Multi-Label Text Classification (XMTC).

To tackle XMTC, researchers have explored both neural network approaches [12, 16, 27, 29] and linear methods [1, 11, 15, 24, 28]. While neural networks are effective at capturing contextual meaning [2], linear methods with simple bag-of-words features (BoW) train much faster and require fewer resources. Therefore, recent studies have shown that linear models can serve as strong baselines in text classification tasks [8, 10, 19], including XMTC [4, 25].

Among linear approaches for XMTC, the most effective are tree-based linear methods [11, 15, 24, 28]. To apply tree-based linear methods in practice, we first convert raw text into numerical BoW features. During training, the method builds a label tree and trains classifiers for the internal nodes. At inference, the model traverses the tree and selects only a few candidate labels at each level to speed up prediction.

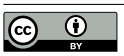
While the above tree-based linear methods are designed to be efficient, practitioners in industry still struggle to produce a simple and strong baseline due to the difficulty of finding suitable hyperparameters. Existing hyperparameter search algorithms such as [26] may not find optimal configurations efficiently. Furthermore, the importance of each hyperparameter remains unclear, so practitioners cannot focus the search on the most critical ones. Many existing methods simply adopt default settings or fix a single configuration without analyzing the impact of different hyperparameter choices.

To unlock the full potential of tree-based linear methods, this work addresses the difficulty of hyperparameter search by identifying the key hyperparameters that influence performance. Our contributions can be summarized as follows:

- We systematically study the impact of various hyperparameters on the performance of tree-based linear models.
- We explain why some hyperparameters strongly affect performance.
- We propose a guideline for hyperparameter search that focuses on three key hyperparameters, and experiments demonstrate the effectiveness of the approach.

We provide supplementary materials and experimental code at https://www.csie.ntu.edu.tw/~cjlin/papers/tree_hyperparams_search. In addition, we have integrated our implementation into the

*Part of this work was done at Mohamed bin Zayed University of Artificial Intelligence.



CIKM '26, Rome, Italy

ACM ISBN 979-8-4007-2539-5/2026/11
https://doi.org/10.1145/3799682.3840104

library LibMultiLabel,[1] and provided a user-friendly tutorial to facilitate easy adoption: https://www.csie.ntu.edu.tw/~cjlin/libmultilabel/auto_examples/plot_tree_gridsearch_tutorial.html. This work is an extension of the second author's master's thesis [7].

2 Linear Methods for Extreme Multi-label Text Classification

We start from a text dataset with N documents and L labels:

$$D = \left\{(d_i, \boldsymbol{y}_i) \mid i = 1, \ldots, N\right\}, \tag{1}$$

where each document d_i corresponds to an L-dimensional label vector $\boldsymbol{y}_i \in \{-1, 1\}^L$. If the document d_i is associated with the label $j \in \{1, 2, \ldots, L\}$, we assign $y_{ij} = 1$; otherwise, we assign $y_{ij} = -1$.

2.1 Bag-of-Words Feature Generation

Bag-of-Words (BoW) is a simple approach that converts each document into a numerical format based on word frequencies. The idea is to describe a document by recording which words appear and how often they appear. To build BoW features, a common procedure is as follows:

(1) Clean and tokenize the text into individual words, optionally capturing short phrases of n consecutive words, known as n-grams.
(2) Remove words and phrases that are overly rare or frequent to form the vocabulary.
(3) Apply TF-IDF [13, 21], a scaled version of word frequency, to convert each document d_i into a feature vector $\boldsymbol{x}_i \in \mathbb{R}^{|\text{vocabulary}|}$.

In this way, the original text dataset D becomes a numerical dataset:

$$D_{\text{BoW}} = \left\{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i = 1, \ldots, N\right\}. \tag{2}$$

2.2 One-versus-Rest Method

The simplest way to train a model for multi-label problems is to use the one-versus-rest (OVR) method. For each label $j \in \{1, 2, \ldots, L\}$, the OVR method builds a separate binary linear classifier $\boldsymbol{w}_j$ to determine whether a document is relevant to label j. Using the BoW dataset defined in (2), we train each classifier $\boldsymbol{w}_j$ by solving the following optimization problem:

$$\min_{\boldsymbol{w}_j} \ \frac{1}{2}\boldsymbol{w}_j^T\boldsymbol{w}_j + C \cdot \sum_{i=1}^{N} \xi\left(y_{ij}\boldsymbol{w}_j^T\boldsymbol{x}_i\right), \tag{3}$$

where C is the cost parameter and $\xi(\cdot)$ denotes a loss function, with common choices listed below.

- Logistic loss:

$$\xi_{\text{LR}}(a) \equiv \log\left(1 + \exp(-a)\right). \tag{4}$$

- Hinge loss (L1 loss):

$$\xi_{\text{L1}}(a) \equiv \max(0,\ 1 - a). \tag{5}$$

- Squared hinge loss (L2 loss):

$$\xi_{\text{L2}}(a) \equiv \max(0,\ 1 - a)^2. \tag{6}$$

Although the OVR method is simple and intuitive, the training time and model size grow linearly with the number of labels L, making the approach impractical for XMTC problems.

[1] https://github.com/ntumlgroup/LibMultiLabel

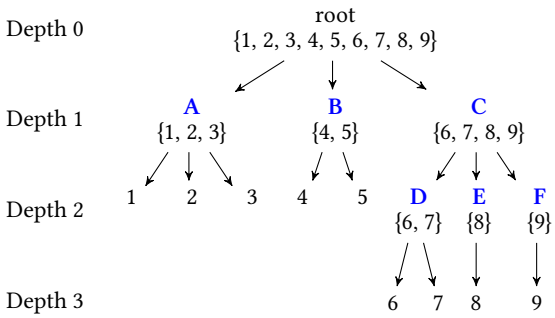


Figure 1: A label tree with nine labels $\{1, 2, \ldots, 9\}$, where the number of clusters $K = 3$. The internal nodes in blue (A, B, C, D, E, F) correspond to subsets of original labels. Adapted from [18] under CC BY 4.0.

2.3 Tree-Based Linear Methods

To efficiently handle XMTC, tree-based linear methods [11, 15, 24, 28] adopt a divide-and-conquer strategy that recursively decomposes the task into smaller multi-label problems. This design significantly reduces training and prediction costs.

2.3.1 Tree Construction. We first generate a representation for each label, typically by aggregating the feature vectors of instances associated with that label or by using additional label information. Next, we organize the labels into a tree structure based on their representations. Starting from the root, we recursively cluster the label representations into K partitions, where each partition becomes a child node containing a subset of the labels. This process continues until the tree meets certain conditions, such as the number of labels per node falling below a threshold or the overall tree depth reaching a predefined limit. When the process terminates, each label becomes a child node and also the leaf node of the whole tree. This gives us a label tree where each label corresponds to a unique leaf node. An example with nine labels is shown in Figure 1.

2.3.2 Model Training. The training process starts from the root node of the label tree. For each non-leaf node, we define a multi-label classification problem, where all child nodes serve as the labels. Similar to the OVR setting, we train a separate binary classifier for each child node using an objective analogous to (3).

However, when training the binary classifiers at each node, we do not use all instances from the training set. Instead, we only include the instances associated with the node's label set. Among these instances, we treat those associated with a child node's label set as positives, while all others are treated as negatives. For example, in Figure 1, node C involves a three-label classification problem, including meta-label D, meta-label E, and meta-label F. We use the instances associated with meta-label D (i.e., label 6 and label 7), meta-label E (i.e., label 8), or meta-label F (i.e., label 9) as the training data. For the binary classifier corresponding to child node D, we treat the instances relevant to label 6 or label 7 as positives, and those associated with label 8 or label 9 but with neither label 6 nor label 7 as negatives.

2.3.3 Label Prediction. Since the label space is extremely large in XMTC settings, we usually care only about the top few labels during prediction. To efficiently identify these top-ranked labels, tree-based methods often adopt beam search to prioritize likely label sets and reduce inference time.

Beam search operates with a predefined beam width B. At each layer, we keep only the top-B nodes based on their prediction scores and proceed to the next layer. In the final layer, we only evaluate the labels in the B nodes selected from the second-to-last layer to make predictions. We provide details in supplementary materials.

3 Difficulty of Hyperparameter Search for Tree-based Linear Methods

In Section 2, we introduced the overall workflow of tree-based linear methods. As each stage in the workflow involves multiple hyperparameters, the number of hyperparameter configurations quickly explodes. To illustrate this complexity, we discuss the commonly used hyperparameters as follows.

First, generating BoW features involves many well-studied hyperparameters in various packages. We follow scikit-learn [23], a popular machine learning library, to consider the following settings.

- **strip_accents**: Whether to remove accents and normalize characters.
- **tokenizer**: Function that splits text into words. This step can also include stemming or lemmatization. Stemming removes common suffixes to obtain a root form (e.g., "quality" to "quali"), while lemmatization applies grammatical rules to identify the correct base form (e.g., "went" to "go").
- **stop_words**: Whether to remove common words that carry little semantic meaning, such as "i," "am," and "a."
- **ngram_range**: Defines the range of n-grams to extract from each document. A setting (a, b) collects all n-grams with n between a and b (inclusive). For example, setting (1, 2) includes both uni-grams and bi-grams.
- **max_features**: Limits the vocabulary size by keeping only the most frequent terms.
- **min_df**: Removes terms that appear in fewer than a specified number of documents.

Second, to define a tree structure and train the model across nodes, we have to specify:

- $\boldsymbol{C}$: Cost parameter in Equation (3).
- $\boldsymbol{K}$: Number of clusters per node.
- $\boldsymbol{d_{\max}}$: Maximum tree depth.

Note that during training, one can also select the loss function (e.g., ξ_{LR}, ξ_{L1}, or ξ_{L2}), which adds another dimension to the search space. However, in practice, users often choose the loss function based on available implementations. Therefore, in our later analysis, we separately examine the influence of hyperparameters under each loss function.

Finally, during prediction, we need to consider:

- $\boldsymbol{B}$: Number of top-B candidates retained at each non-leaf layer.

Obviously, a full search over so many hyperparameters leads to an exponential growth in configurations. Practitioners may fail to find good configurations within a reasonable budget. Unfortunately, it remains unclear which hyperparameters truly impact performance. Most existing studies lack a detailed analysis. Instead, they adopt fixed settings or explore only a subset of hyperparameters.

For example, [15, 24, 28] do not specify their settings for generating BoW features, and [4] only mentions using n-gram features in a product search scenario. Regarding label tree construction, although [15, 24, 28] consider tree depth, they fix the number of clusters at $K = 2$, $K = 100$, and $K = 32$, respectively. In contrast, [28] explores different values of K but does not consider the size of the label set L when selecting this parameter. We will revisit this issue later. For the cost parameter C, [6] reports a substantial performance variation across different values of C. However, the underlying reasons for this observation are not examined. Finally, only [4, 24] discuss the beam width B used in the prediction stage.

Nevertheless, these works have introduced effective tree-based linear methods for XMTC and have established efficient baselines. To complete the understanding of these methods, we decide to thoroughly study the hyperparameters.

4 Hyperparameter Sensitivity in Tree-Based Methods

It is difficult to effectively study the influence of so many hyperparameters, as directly exploring all combinations is impractical. To make the analysis manageable, we design a reasonable and feasible approach. We vary one hyperparameter at a time, with the others fixed at commonly used values in popular libraries. In this work, we refer to these values as the default values. This setup allows us to isolate the effect of each hyperparameter and assess how sensitive model performance is to individual changes.

4.1 Sensitivity Analysis

We conduct our analysis on four XMTC datasets.[2] We provide the analysis setup and hyperparameter settings in supplementary materials. Note that for the number of clusters K, rather than fixing a constant value, we consider the following formulation to adapt to different label scales:

$$K = \left\lceil \alpha \cdot L^{1/d_{\max}} \right\rceil, \tag{7}$$

where α is a scaling factor that we tune in practice to control the cluster size. This adaptive setting is important for analyzing the effect of K, and we detail our novel choice in supplementary materials.

For each hyperparameter, we generate a set of configurations by varying its value over the search range, while setting the remaining hyperparameters to their default values. For each configuration, we train a model on the training set and then evaluate performance on the test set using precision at top-ranked labels (P@1, P@3, and P@5).[3] Finally, we compute the performance variance across configurations for each hyperparameter to assess sensitivity.

Table 1 shows the variance of P@1 for two representative datasets. Other datasets and metrics exhibit similar trends (see supplementary materials). The cost parameter C shows the largest variance across all datasets and loss functions, far exceeding that of other hyperparameters. The number of clusters K and the ngram_range

[2] See Section 6 for dataset details.
[3] See Section 6.1 for P@1, P@3, and P@5 definitions.

Table 1: Variance of P@1 for different hyperparameters. The highest result is bold-faced, the second highest is underlined, and the third highest is double underlined. For the calculation of the variance, we scaled the value of P@1 to $[0, 100]$.

	EUR-Lex-57K			Mimic-iii-8K		
	ξ_{L2}	ξ_{L1}	ξ_{LR}	ξ_{L2}	ξ_{L1}	ξ_{LR}
TF-IDF feature generation						
strip_accents	0.011	0.003	0.006	0.011	0.029	0.000
tokenizer	0.069	0.107	0.271	0.181	0.150	0.020
stop_words	0.000	0.298	0.478	0.142	0.602	0.971
ngram_range	0.084	0.444	0.137	0.259	0.536	0.402
max_features	0.013	0.049	0.021	0.049	0.225	0.007
min_df	0.029	0.018	0.008	0.036	0.046	0.004
Training						
C	**1.248**	**2.592**	**19.645**	**7.288**	**8.181**	**12.883**
K	0.133	0.297	0.321	0.341	3.079	0.336
$d_{\max}$	0.001	0.007	0.000	0.032	0.031	0.002
Prediction						
B	0.001	0.020	0.001	0.047	0.074	0.050

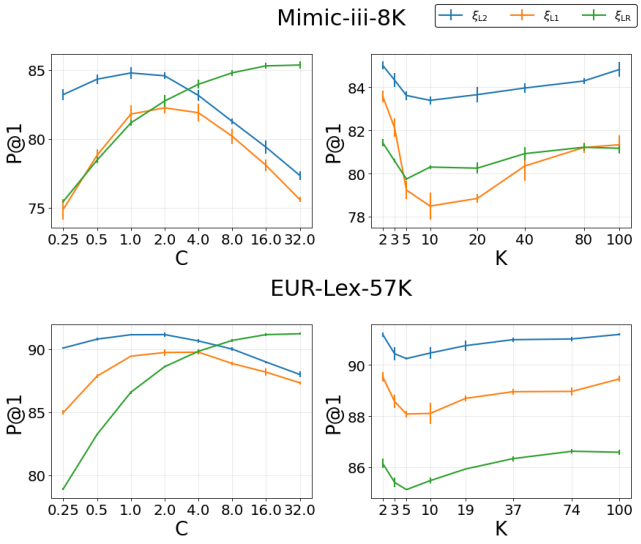


Figure 2: Sensitivity analysis of C and K over Mimic-iii-8K and EUR-Lex-57K on P@1. We average the results over three runs with different random seeds and plot the 95% confidence intervals as error bars.

also have large effects. While the stop_words affects performance, the effect is smaller and more dataset dependent. The other hyperparameters have only minor effects.

To further explore how the search space affects performance, we plot P@1 across different values of the two most influential hyperparameters: the cost parameter C and the number of clusters K. Figure 2 presents the results. Similar patterns emerge across other

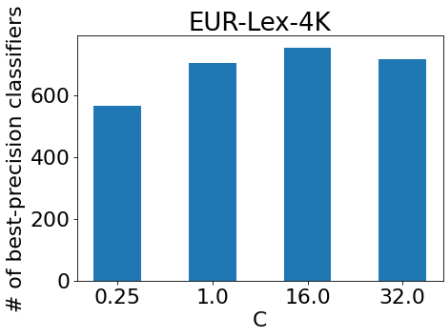


Figure 3: Performance of individual binary classifiers in the OVR setting with different C. The plot shows how many classifiers achieve their highest precision for each C.

datasets and metrics (see supplementary materials). We summarize our findings below.

First, for the most influential hyperparameter, the cost parameter C, each loss function reaches its best performance in a different range. For the loss ξ_{LR}, performance steadily increases as C grows. In contrast, ξ_{L1} performs best with smaller C, and ξ_{L2} requires even smaller values.

Second, the number of clusters K shows a U-shaped effect on performance. Results are strong when $K = 2$, then decline as K increases, and improve again at larger values. Interestingly, while [24] fixes $K = 2$ and [15, 28] suggest that a larger K leads to better performance, none of these studies discuss the gap between these extremes.

On the other hand, for ngram_range and stop_words, which show secondary but noticeable effects, we provide further analysis in supplementary materials.

4.2 Why Certain Hyperparameters Are Highly Influential: A Deep Investigation

In the previous section, we found that the cost parameter C and the number of clusters K substantially affect model performance. Therefore, we delve into these two hyperparameters to uncover the reasons why they are particularly influential.

4.2.1 The Cost Parameter C. As discussed in Section 2, both OVR and tree-based linear methods solve many binary problems, each associated with a cost parameter C. To study how C affects performance, we examine how different values of C influence individual binary classifiers. Since the number of training instances for each binary problem can vary across nodes in the tree-based settings, we first analyze the OVR setting, where each binary classifier is trained on the entire training set. For each binary classifier, we check the C value that yields the highest precision, then accumulate the counts for each C value, as shown in Figure 3. Obviously, no single C value dominates the best performance across all classifiers. In other words, each classifier prefers a different optimal C.

Besides, in the supplementary materials, we analyze the effect of C under different numbers of positives and negatives in both OVR and tree-based linear methods. We find that, since each binary problem involves varying degrees of class imbalance, the optimal C

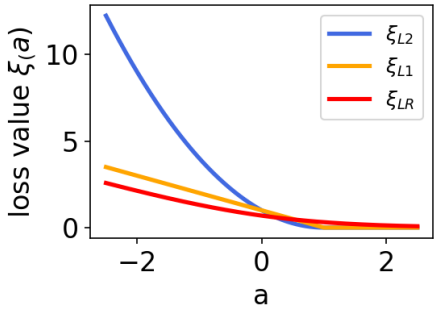


Figure 4: Comparison of ξ_{L2}, ξ_{L1}, and ξ_{LR} loss functions.

differs across classifiers. This observation explains why adjusting C can lead to significant performance changes, as shown in section 4.1.

With the above observations and analysis, naturally we should consider assigning a distinct C to each binary classifier while jointly optimizing all classifiers to maximize overall performance. However, if we consider all C hyperparameters across all classifiers and tune them together with other hyperparameters, the search space will grow exponentially and make the hyperparameter search impractical. Therefore, we only have two options:

(1) Independently optimize C for each binary classifier and collect them to form the final model. For example, for each binary classifier, we conduct a validation procedure to select the corresponding best C.
(2) Use a single shared C for all binary classifiers and optimize the overall performance.

In multi-class classification, a special case of the multi-label problem where each instance has only one label, previous studies [5, 14] show that these two options yield comparable performance. They also point out that tuning C for each classifier separately only optimizes the local binary problem and does not guarantee the best global performance. Therefore, popular multi-class implementations [3] adopt the same C for all binary classifiers, i.e., option 2.

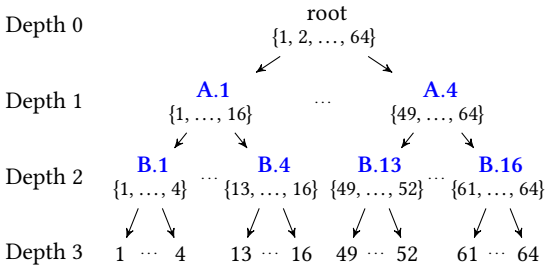
In XMTC with tree-based linear methods, we also find that using the same C across all binary classifiers can lead to good results. Our analysis in the supplementary materials shows that option 1 can even yield lower overall performance than option 2. The reason may be that, in the tree-based setting, the limitation of option 1 becomes more pronounced due to the many interconnected binary problems across nodes. Therefore, for tree-based linear methods, we recommend using a single C for all classifiers.

Moreover, as shown in Figure 2, the effective range of C values differs across loss functions, with ξ_{LR} generally requiring larger C values. To investigate this difference, we compare these loss functions in Figure 4 and observe that when the input to the loss function is less than zero (i.e., $a < 0$),[4] the following relationship holds:

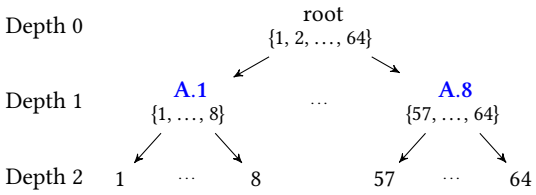
$$\xi_{\text{LR}} < \xi_{\text{L1}} < \xi_{\text{L2}}.$$

Therefore, a smaller loss value requires a larger C to impose sufficiently strong penalties in the optimization problem (3), which is why the ξ_{LR} loss needs a larger C.

[4]Recall that a is defined in (4)-(6).



(a) Tree structure for $K = 4$. The tree has four levels, and depth 1 contains four clusters. Each cluster at depth 1 splits into four clusters at depth 2, and each cluster at depth 2 further splits into four true labels at depth 3.



(b) Tree structure for $K = 8$. The tree has three levels, and depth 1 contains eight clusters. Each cluster at depth 1 splits into eight true labels at depth 2, the final level.

Figure 5: Illustration of balanced label trees with $L = 64$ for $K = 4$ and $K = 8$.

In summary, while previous work [6] has reported that model performance is sensitive to C, we explain why C plays a critical role in tree-based linear methods.

4.2.2 Number of Clusters K. Recall from Figure 2 that varying K produces a surprisingly U-shaped performance trend. This finding intrigues us to further investigate how K affects the tree structure and, in turn, the model performance. However, analyzing this behavior is challenging, and after several attempts, we find that different values of K directly affect the number of labels evaluated during prediction. When fewer labels are evaluated, some relevant labels may be pruned by the beam width B, leading to lower performance. Varying K changes how many labels are evaluated and therefore produces the U-shaped trend. To illustrate this effect, we begin with the following example.

Figure 5 compares two balanced tree structures generated with $K = 4$ and $K = 8$, respectively, under a fixed number of labels $L = 64$. In this example, we consider a beam width $B = 3$ for prediction in tree-based methods. During inference, we only retain B nodes at each level, so some labels may never be visited or evaluated once their corresponding nodes are pruned. As shown in Figure 5a, when $K = 4$, we select 3 out of 4 clusters at depth 1, and only their child nodes proceed to depth 2. This results in $3 \times 4 = 12$ clusters at depth 2. We then select 3 clusters at depth 2, with each cluster containing 4 child labels. Therefore, at the final level, we only evaluate $3 \times 4 = 12$ labels. In contrast, in Figure 5b, when $K = 8$, the tree has only two levels. At depth 1, we select 3 out of 8 clusters, and at depth 2, the last level, we evaluate $3 \times 8 = 24$ labels. Clearly, this example shows

Table 2: Search ranges of selected hyperparameters for the loss functions ξ_{LR}, ξ_{L1}, and ξ_{L2}.

Hyperparameters	ξ_{L2}	ξ_{L1}	ξ_{LR}
C	{0.5, 1, 2}	{1, 2, 4}	{8, 16, 32}
K	{Equation (7) | $\alpha = 2^{-2}, 2^{5}$}		
ngram_range	{(1,1), (1,2), (1,3)}		
stop_words	Enable stop words for data language(s)		

that, under the same beam width B, the two different K settings can affect the number of evaluated labels. In the supplementary materials, we further confirm that this relationship explains the U-shaped performance in real-world settings. Thus, we must carefully select K to avoid reducing the number of evaluated labels.

5 A Practical Guideline for Efficient and Effective Search

In Section 4, we explored the hyperparameter space of tree-based linear methods and identified the most influential ones. These insights help narrow the hyperparameter search to a small, focused space. While exhaustive search over the full hyperparameter space is computationally infeasible for XMTC, the reduced search space enables us to systematically evaluate all combinations. In this way, we make hyperparameter search capture dependencies between hyperparameters in a practical and efficient manner. We now suggest a guideline for hyperparameter search in tree-based linear methods.

5.1 Our Proposed Guideline

Since our analysis in Section 4 reveals that the cost parameter C, the number of clusters K, ngram_range, and stop_words are the most influential, we should focus on tuning these four hyperparameters. For the remaining hyperparameters, we adopt their commonly used values, as they have minimal impact on performance. In this way, we drastically reduce the number of combinations to explore. This leads to the following guideline:

(1) Apply instance-wise data normalization.
(2) Select a suitable range for the cost parameter C based on the loss function.
(3) Consider both relatively small and large values of the number of clusters K.
(4) Consider only smaller n-gram features.
(5) Always enable stop words removal.

For the first step, previous studies in text classification [9, 17] suggest normalizing each feature vector to unit length, as the data scale can influence the optimization problem (3). Normalization brings the suitable range of C in (3) into a similar numerical scale.

Since we have shown that the cost parameter C affects model performance differently across loss functions, we can assign a tailored search range to each loss function. The number of clusters K exhibits a U-shaped performance trend. Therefore, we focus on relatively small or large values while skipping the middle range. For n-grams, we consider only smaller n-gram features, as our analysis in Section 4 shows that larger n-grams often add little or even reduce accuracy. We also recommend filtering out stop words, as this strategy usually improves results.

While this general guideline does not specify concrete values for each parameter, we now suggest a realization based on our empirical observations in Section 4. This design enables industry users, who may have limited knowledge of hyperparameters, to easily apply tree-based linear methods for building a simple and strong baseline. We summarize our choices of search ranges for C, K, and n-gram features in Table 2. For each loss function, the search space is manageable, with only $3 \times 2 \times 3 = 18$ configurations. We can even run them in parallel for a more efficient search.

5.2 Challenges of Cross-Validation in XMTC

To ensure reliable evaluation for each hyperparameter configuration, we must keep the test set independent of both hyperparameter search and model training. Specifically, we should perform hyperparameter tuning solely on the training data and evaluate the final model once on the test set. Therefore, a common approach is to conduct cross-validation on the training data, where the training data is split into several folds and each fold serves once as the validation set while the others are used for training.

The most straightforward approach to create these folds is to split the data by instance at random. However, a random split can alter the label distribution in each fold so that the distribution in the fold no longer matches the distribution in the original dataset. In multi-class classification, a special case of the multi-label problem where each instance has only one label, researchers can place the instances of each label evenly into folds to ensure that each fold reflects the overall label frequency. In contrast, in multi-label settings, each instance can belong to multiple labels. When allocating instances for one label, the same instances may also belong to other labels, so the allocation for one label can disrupt the distribution of others. As a result, preserving the target distribution for all labels simultaneously becomes difficult. This issue becomes even more severe in XMTC, where many labels are rare and may be absent from an entire fold. This challenge raises the question of whether the naive approach of randomly splitting instances can still work well for hyperparameter search in XMTC. Therefore, in Section 6, we evaluate the effectiveness of this approach in producing reliable hyperparameter selection.

6 Experimental Results

In this section, we evaluate the effectiveness of our guideline on six real-world XMTC datasets. In addition to the four datasets used in Section 4 for hyperparameter analysis, we include two more datasets to examine the generality of the approach. We summarize the dataset statistics in Table 4. Data sources are provided in the supplementary materials.

6.1 Evaluation Metrics

In XMTC settings, we usually aim to identify a small number of relevant labels from an extremely large label set. Therefore, we adopt precision at k (P@k), a popular ranking-based metric, to evaluate performance. For each test instance, precision measures the fraction of relevant labels among the top-k predictions. We then average the precision values across all test instances to obtain P@k.

Table 3: P@1, P@3 and P@5 test scores for our investigation on three configurations.

	EUR-Lex-4K			EUR-Lex-57K			Mimic-iii-8K			Wiki10-31k			AmazonCat-13K			Amazon-670K		
	ξ_{L2}	ξ_{L1}	ξ_{LR}	ξ_{L2}	ξ_{L1}	ξ_{LR}	ξ_{L2}	ξ_{L1}	ξ_{LR}	ξ_{L2}	ξ_{L1}	ξ_{LR}	ξ_{L2}	ξ_{L1}	ξ_{LR}	ξ_{L2}	ξ_{L1}	ξ_{LR}
P@1																		
Default	82.08	80.15	73.77	91.16	89.46	86.59	84.79	81.80	81.16	84.80	84.02	81.43	93.61	91.68	93.32	44.20	38.62	30.25
CV-Tuned	**83.29**	**82.28**	**82.74**	**92.12**	**91.22**	**91.77**	**84.88**	**84.49**	**85.41**	**85.04**	**84.30**	**85.14**	**94.12**	**92.33**	**94.53**	**44.38**	**41.97**	**43.83**
Best-on-Test	83.34	82.48	82.74	92.17	91.65	92.17	85.68	84.99	85.68	85.22	84.67	85.22	94.12	92.33	94.53	44.38	41.99	43.83
P@3																		
Default	69.06	66.74	59.41	81.14	79.06	74.30	77.54	76.25	73.60	74.54	74.32	67.74	79.76	79.11	77.96	38.87	33.10	25.67
CV-Tuned	**71.69**	**70.19**	**70.64**	**82.73**	**82.18**	**82.22**	**79.01**	**79.25**	**78.31**	**75.14**	**74.45**	**74.89**	**80.27**	**79.61**	**80.04**	**39.09**	**36.76**	**38.74**
Best-on-Test	71.69	70.37	70.64	82.97	82.23	82.46	79.36	79.25	78.69	75.17	74.80	75.09	80.27	79.61	80.04	39.09	36.76	38.74
P@5																		
Default	57.98	54.45	48.78	68.33	65.43	61.70	71.48	70.47	67.00	65.79	64.74	59.08	64.77	63.80	62.65	34.98	28.95	22.46
CV-Tuned	**60.11**	**58.32**	**58.80**	**70.05**	**68.83**	**69.66**	**72.87**	**73.16**	**72.00**	**66.09**	**65.06**	**65.74**	**65.15**	**64.40**	**64.92**	**35.34**	**33.02**	**35.01**
Best-on-Test	60.11	58.47	58.80	70.17	69.02	69.66	73.34	73.16	72.12	66.18	65.06	65.74	65.15	64.40	64.92	35.34	33.02	35.01

Table 4: Data Statistics.

dataset	# training instances	# test instances	# labels L
EUR-Lex-4K	15,449	3,865	3,956
EUR-Lex-57K	51,000	6,000	4,271
Mimic-iii-8K	49,354	3,372	8,921
AmazonCat-13K	1,186,239	306,782	13,330
Wiki10-31K	14,146	6,616	30,938
Amazon-670K	490,449	153,025	670,091

6.2 Performance with Our Hyperparameter Search Guideline

In Section 5, we propose a guideline that substantially reduces the hyperparameter search space for tree-based linear methods. To evaluate the effectiveness of this approach, we follow prior works [15, 24, 27, 28] and report test P@k for $k = 1$, 3, and 5. Specifically, we randomly split the training data into three folds and perform cross-validation separately for P@1, P@3, and P@5. For each metric, we select the hyperparameter configuration with the highest cross-validation score. Using this configuration, we retrain the model on the entire training set and evaluate the model on the test set for that metric.

For each loss (ξ_{LR}, ξ_{L1}, and ξ_{L2}), Table 3 compares the following hyperparameter configurations:

- **Default:** Use the default settings to train the model and evaluate on the test set.
- **CV-Tuned:** Use three-fold cross-validation on the training data to select the best settings from the search ranges in Table 2. We then use the selected settings to train the model and evaluate the model on the test set.
- **Best-On-Test:** Select the best-performing setting on the test set from the same search ranges in Table 2. For this setting, we assume that the labels of the test data are known, a situation that is prohibited in practice. The results serve only as an upper bound for comparison.

Across all datasets and loss functions, CV-Tuned consistently outperforms the Default setting. These results demonstrate that our guideline for adjusting the three key hyperparameters is highly effective. For example, ξ_{LR} performs the worst among the three loss functions under the Default setting, likely because the default $C = 1$ is too small for ξ_{LR}. However, with appropriately tuned hyperparameters, ξ_{LR} becomes competitive with the other two loss functions and even surpasses them in some cases. Thus, proper hyperparameter tuning must be considered in model development. Moreover, CV-Tuned achieves performance very close to Best-on-Test, which directly selects the best configuration based on test performance. The small gap between these two settings demonstrates the effectiveness of our proposed guideline.

Besides, the results in Table 3 address the cross-validation challenges in XMTC discussed in Section 5.2, where practitioners may be unsure how to split folds for hyperparameter search. Despite using randomly generated folds, CV-Tuned identifies near-optimal or optimal settings, with results on EUR-Lex-4K, AmazonCat-13K, and Amazon-670K closely matching Best-on-Test. This suggests that simple random splits are sufficient for effective hyperparameter selection, avoiding the need to design or select sophisticated sampling strategies for XMTC.

7 Conclusions

In this paper, we systematically analyze the hyperparameters of tree-based linear methods for XMTC. We identify the key hyperparameters that have the greatest impact on model performance and explain why they play such a critical role. Based on these insights, we suggest a guideline for hyperparameter search that focuses on tuning only three hyperparameters instead of searching over all possible hyperparameter combinations. Experiments demonstrate that our proposed guideline consistently improves the performance of tree-based linear methods. Moreover, we show that when constructing folds for cross-validation, a simple random split is sufficient. As a result, we strengthen existing tree-based linear methods and help them serve as stronger baselines for XMTC.

Acknowledgments

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.

GenAI Usage Disclosure

The authors used generative AI tools only to improve writing clarity and correct grammar. All research design, experiments, analyses, and conclusions were conducted by the authors without AI assistance.

References

[1] 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)*. 721–729.

[2] 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*. 4310–4330.

[3] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2, 3 (2011), 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[4] 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)*.

[5] Pai-Hsuen Chen, Chih-Jen Lin, and Bernhard Schölkopf. 2005. A Tutorial on ν-Support Vector Machines. *Applied Stochastic Models in Business and Industry* 21 (2005), 111–136. Issue 2. http://www.csie.ntu.edu.tw/~cjlin/papers/nusvmtoturial.pdf

[6] Sheng-Wei Chen, Kuan-Ting Chen, Guan-Ting Chen, Yaxu Liu, and Chih-Jen Lin. 2026. Probability Estimation in Tree-Based Linear Methods for Extreme Multi-Label Learning. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*. https://www.csie.ntu.edu.tw/~cjlin/papers/tree_prob_issue/tree_prob_issue.pdf

[7] Hung-Chih Chiang. 2026. *On the Influence of Hyperparameters in Tree-Based Linear Methods for Extreme Multi-Label Text Classification: Insights for Efficient and Effective Search*. Master's thesis. Department of Computer Science and Information Engineering, National Taiwan University.

[8] Eva Engel. 2023. *Balancing Performance and Usage Cost: A Comparative Study of Language Models for Scientific Text Classification*. Master's thesis. KTH, Mathematical Statistics.

[9] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research* 9 (2008), 1871–1874. http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf

[10] Lukas Galke, Andor Diera, Bao Xin Lin, Bhakti Khera, Tim Meuser, Tushar Singhal, Fabian Karl, and Ansgar Scherp. 2022. Are we really making much progress in text classification? A comparative review. *arXiv preprint arXiv:2204.03954* (2022).

[11] Kalina Jasinska, Krzysztof Dembczyński, Róbert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hüllermeier. 2016. Extreme F-measure maximization using sparse probability estimates. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*. 1435–1444.

[12] Ting Jiang, Deqing Wang, Leilei Sun, Huayi Yang, Zhengyang Zhao, and Fuzhen Zhuang. 2021. LightXML: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 7987–7994.

[13] Karen S. Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28, 1 (1972), 11–20.

[14] Wei-Chun Kao, Kai-Min Chung, Chia-Liang Sun, and Chih-Jen Lin. 2004. Decomposition Methods for Linear Support Vector Machines. *Neural Computation* 16, 8 (2004), 1689–1704. http://www.csie.ntu.edu.tw/~cjlin/papers/linear.pdf

[15] Sujay Khandagale, Han Xiao, and Rohit Babbar. 2020. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning* 109 (2020), 2099–2119. doi:10.1007/s10994-020-05888-2

[16] Siddhant Kharbanda, Atmadeep Banerjee, Erik Schultheis, and Rohit Babbar. 2022. 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. 2074–2087.

[17] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research* 5 (2004), 361–397.

[18] He-Zhe Lin, Cheng-Hung Liu, and Chih-Jen Lin. 2024. Exploring Space Efficiency in a Tree-based Linear Model for Extreme Multi-label Classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 16245–16260. https://www.csie.ntu.edu.tw/~cjlin/papers/multilabel_tree_model_size/multilabel_tree_model_size.pdf

[19] Yu-Chen Lin, Si-An Chen, Jie-Jyun Liu, and Chih-Jen Lin. 2023. Linear Classifier: An Often-Forgotten Baseline for Text Classification. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*. 1876–1888. Short paper. https://www.csie.ntu.edu.tw/~cjlin/papers/text_classification_baseline/text_classification_baseline.pdf

[20] Johannes Loza Mencía, Eneldoand Fürnkranz. 2010. Efficient multilabel classification algorithms for large-scale problems in the legal domain. In *Semantic Processing of Legal Texts: Where the Language of Law Meets the Law of Language*, Enrico Francesconi, Simonetta Montemagni, Wim Peters, and Daniela Tiscornia (Eds.). Springer Berlin Heidelberg, 192–215.

[21] Hans Peter Luhn. 1958. The automatic creation of literature abstracts. *IBM Journal of Research and Development* 2, 2 (1958), 159–165.

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

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

[24] 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)*. 993–1002.

[25] Mohammadreza Qaraei, Sujay Khandagale, and Rohit Babbar. 2020. 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)*. 223–228.

[26] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25 (NIPS)*. 2951–2959. doi:10.5555/2999325.2999464

[27] 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*, Vol. 32.

[28] 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 (2022), 1–32.

[29] Jiong Zhang, Wei-Cheng Chang, Hsiang-Fu Yu, and Inderjit Dhillon. 2021. Fast Multi-Resolution Transformer Fine-tuning for Extreme Multi-label Text Classification. In *Advances in Neural Information Processing Systems*, Vol. 34. 7267–7280.

[30] Arkaitz Zubiaga. 2009. Enhancing Navigation on Wikipedia with Social Tags. In *Proceedings of Wikimania*.