BoostXML: Gradient Boosting for Extreme Multilabel Text Classification With Tail Labels

Fengzhi Li, Yuan Zuo, Hao Lin, *Member, IEEE*, and Junjie Wu

***Abstract*— Multilabel learning involving hundreds of thousands or even millions of labels is referred to as *extreme multilabel learning* (XML), in which the labels often follow a power-law distribution with the majority occurring in very few data points as *tail labels*. Recent years have witnessed the intensive use of deep-learning methods for high-performance XML, but they are typically optimized for the head labels with abundant training instances and less consider the performance on tail labels, which, however, like the needles in haystacks, are often the focus of attention in real-life applications. In light of this, we present BoostXML, a deep learning-based XML method for extreme multilabel text classification, enhanced greatly by gradient boosting. In BoostXML, we pay more attention to tail labels in each Boosting Step by optimizing the residual mostly from unfitted training instances with tail labels. A Corrective Step is further proposed to avoid the mismatching between the text encoder and weak learners during optimization, which reduces the risk of falling into local optima and improves model performance. A Pretraining Step is also introduced in the initial stage of BoostXML to avoid exorbitant bias to tail labels. Extensive experiments on five benchmark datasets with state-of-the-art baselines demonstrate the advantage of BoostXML in tail-label prediction.**

***Index Terms*— Corrective step, deep learning, extreme multilabel learning (XML), gradient boosting, tail labels.**

I. Introduction

EXTREME Multilabel Learning (XML) aims to train a classifier that can automatically tag a new data point with the most relevant subset of labels from an extremely large label set. Due to the rapid growth of sheer data volumes and the prosperity of real-world applications, XML has attracted considerable research attention in recent years. For instance, Wikipedia nowadays has more than one million labels (categories), and a classifier is in great need to annotate a new article with the most relevant Wikipedia categories automatically. Another example is to regard billions of YouTube videos as distinct labels in a multilabel classifier and recommend a ranked list of videos to users with personalized preferences.

Given the extra-large scale of data labels, it is difficult to apply traditional multilabel classification methods for the XML task. The existing XML methods can be roughly divided into non-deep-learning methods [1], [2], [3] and deep-learning methods [4], [5], [6]. Non-deep-learning methods usually take handcraft features as input, aim primarily to reduce computational complexity, and improve training and prediction efficiency. There are mainly three major ways to achieve this goal, including projecting labels into a low-dimensional latent space [1], [7], partitioning instances or labels into smaller clusters or sets [8], [9], and leveraging sparse or sampling methods [3], [10]. Recently, to pursue higher classification precision, deep learning-based XML methods [4], [5] have been studied extensively.

Although the reported performances of deep learning XML methods seem to exceed those of non-deep learning XML methods, we argue that the performances might be biased to head labels that are associated with abundant training instances. However, the labels of real-world XML datasets usually exhibit a *power-law* distribution; that is, most labels are associated with only a few instances, which are known as *tail labels*. Tail-label prediction, e.g., for promoting enormous niche items in an electronic commerce website, in many cases is just the focal point of XML tasks. The head bias might attribute to two reasons. From the performance evaluation perspective, the widely adopted evaluation measures for existing deep learning XML methods, e.g., Precision at k, cannot well reflect the prediction performance on the tail labels. From the model structure perspective, they do not explicitly model the skewed label distribution and thus suffer from the suboptimal performance on tail labels. For example, AttentionXML [4] builds a shared classifier for all labels, which may transfer the classification information of head labels to tail labels mistakenly. These indeed motivate our study.

In this work, we propose BoostXML, a novel gradient-boosting enhanced deep learning XML model for extreme multilabel text classification,[1] which tackles the power-law distribution problem of data labels explicitly. In general, BoostXML has a similar architecture as the existing deep learning XML methods, but its classifier as well as the training

Manuscript received 21 March 2022; revised 3 January 2023 and 28 May 2023; accepted 6 June 2023. The work of Yuan Zuo was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 71901012. The work of Junjie Wu was supported in part by NSFC under Grant 72031001 and Grant 72242101. This work was supported by the High-Performance Computing (HPC) Resources at Beihang University. *(Corresponding authors: Yuan Zuo; Junjie Wu.)*

The authors are with the Department of Information Systems, Beihang University, Beijing 100191, China, and also with the Key Laboratory of Data Intelligence and Management (Beihang University), Ministry of Industry and Information Technology, Beijing 100191, China (e-mail: lifengzhi@buaa.edu.cn; zuoyuan@buaa.edu.cn; haolin@buaa.edu.cn; wujj@buaa.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3285294.

Digital Object Identifier 10.1109/TNNLS.2023.3285294

[1] As existing multilabel learning benchmark datasets are mainly text data, our proposed BoostXML is designed for multilabel text classification.

process, are designed purposefully to suit gradient boosting. Concretely, a *Pretraining Step* is firstly conducted in the training process, where a pretrained primary backbone model is utilized to initialize BoostXML. This operation can reduce the negative impact on representation learning caused by overfitting the tail labels. After the Pretraining Step, each training iteration of BoostXML is mainly composed of two steps, i.e., the *Boosting Step* and the *Corrective Step*. In the Boosting Step, weak learners are trained sequentially by fitting the residual generated by the ensemble of trained learners, most of which is thought of as coming from unfitting training instances with tail labels. Hence, by adding weak learners continuously, BoostXML pays more attention to different tail labels and attempts to correct the wrong predictions. After each Boosting Step is a Corrective Step, in which the parameters of the whole network are updated through back-propagation. This enables the text encoder and weak learners to cooperate with each other during training to avoid falling into local optima.

The main contributions of our work are summarized as follows.

1) To our best knowledge, we are among the first to tackle the widespread long-tail problem in XML tasks with a deep learning-based method.
2) To enable a neural gradient boosting procedure, a Corrective Step is introduced and executed iteratively after each Boosting Step, which avoids the mismatching between the encoder and earlier trained base learners during training.
3) To boost the performance of tail labels without seriously hurting the performance of head ones, we propose a two-step training scheme, where an extra Pretraining Step is introduced.
4) Extensive experiments are conducted on five benchmark datasets with the presence of various state-of-the-art baseline methods, and the results demonstrate the superiority of BoostXML in tail-label prediction.

The remainder of this article is organized as follows. In Section II, we introduce two types of deep XML methods and gradient boosting, and define our problem. In Section III, we propose the BoostXML method and give the training as well as predicting processes. The experimental results and related works are given in Sections IV and V, respectively. We finally conclude our work in Section VI.

II. Preliminary

A. Deep XML Methods and Tail Labels

With the advent of deep learning, neural network-based methods have become prevalent in tackling XML problems. Existing methods can be roughly divided into two categories according to the type of their classification algorithm. One is the one-versus-all classification method, which we call multiple binary classifiers, and the other is the shared binary classification method, which we call the shared binary classifier.

As illustrated in Fig. 1, multiple binary classifiers-based deep XML methods first obtain the instance representation, then compute the score of each label through its corresponding

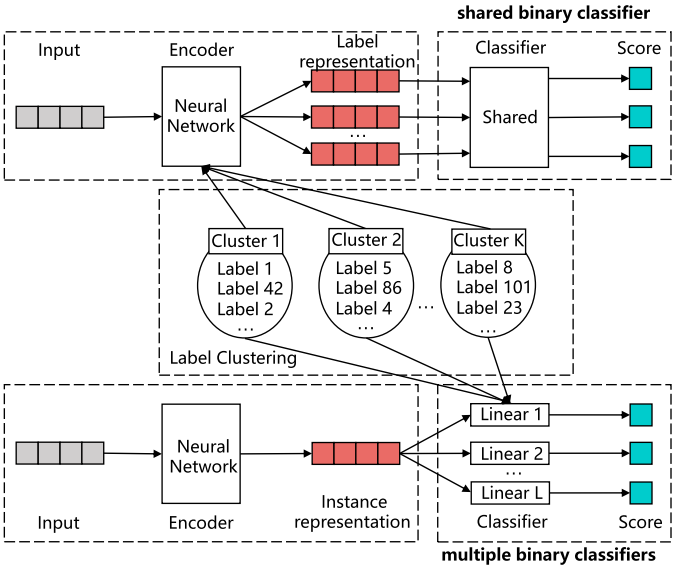


Fig. 1. Two typical types of deep XML methods.

binary classifier. On the contrary, shared binary classifier-based deep XML methods obtain label representations first, then compute the score of each label through a shared binary classifier. The computation cost of calculating the scores of all labels for each instance is intolerable and unnecessary, since extreme multilabel datasets contain hundreds of thousands to millions of instances and labels. By assigning millions of labels to multiple clusters, XML methods can first select the corresponding clusters, and then predict the scores of the candidate labels from these clusters. With label clustering, the number of labels needed to be predicted by the model is reduced significantly, thus greatly improving the efficiency of dealing with the large-scale label prediction problem.

The labels of real-world XML datasets tend to exhibit a *power-law* distribution, i.e., most labels are associated with only a few instances. These labels, also known as *tail labels*, cannot be well approximated by any linear low-dimensional basis owing to the paucity of instances. When multiple binary classifiers are adopted in a deep XML method, no explicit relations are considered between these classifiers. Therefore, classifiers of tail labels cannot be sufficiently trained because too few instances are seen during the training, which results in poor prediction performance in the testing phase. As to shared binary classifier-based XML methods, a shared classifier is trained for both tail labels and head labels, which alleviates the insufficient training of the classifier problem mentioned before. However, the shared classifier is not designed purposefully for predicting tail labels. Therefore, there is still a great need in developing a new deep-learning method that is capable of explicitly tackling XML with tail labels. In Section II-B, we briefly introduce gradient boosting and elaborate more on why it has the potential to improve the share binary classifier for tail labels.

B. Gradient Boosting

Boosting, as one of the branches of ensemble learning algorithms, is well-known for its good prediction performance.

In the family of boosting algorithms, gradient boosting is one of the most commonly used. Like other boosting algorithms, given the input $\mathbf{x}$, gradient boosting is built with an additive expansion and is formally defined as follows:

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t f_t(\mathbf{x}) \tag{1}$$

where $F_t(\mathbf{x})$ is an ensemble of t functions, and ρ_t is the weight of the tth function $f_t(\mathbf{x})$. The whole model is constructed stepwise in the sense that at each step a new model f_t is built without modifying any of the previously created models in $F_{t-1}(\mathbf{x})$. Given a training dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with N instances, the goal of tth gradient boosting step is to minimize the following objective function:

$$(\rho_t, f_t(\mathbf{x})) = \underset{\rho, f}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{t-1}(\mathbf{x}_i) + \rho f(\mathbf{x}_i)). \tag{2}$$

Instead of training on the raw D, gradient boosting's target is to fit the "residual." Specifically, each model f_t is trained on a new dataset $\hat{D} = \{(\mathbf{x}_i, r_{ti})\}_{i=1}^N$, where the pseudo-residual r_{ti} is the negative gradient at $F_{t-1}(\mathbf{x})$, and is formally defined as follows:

$$r_{ti} = -\frac{\partial L(y_i, F_{t-1}(\mathbf{x}_i))}{\partial F_{t-1}(\mathbf{x}_i)}. \tag{3}$$

Gradient boosting takes the negative gradient as the measurement of the mistakes made by $F_t(\mathbf{x})$ after the tth step, and corrects them by fitting the negative gradient with the newly added weak learner in the next round. We argue that this learning mechanism is suitable for tail labels because they are more likely to be wrongly classified than head labels. Thus, tail labels' negative gradients are usually larger and tend to receive more attention from the gradient-boosting method during training. In this regard, the gradient boosting method has the potential to be utilized in tackling XML with tail labels.

C. Problem Definition

In XML, the labels are the same as those in traditional multilabel classification, which can be written as $\mathcal{Y} = \{1, 2, \ldots, L\}$. The difference is that L is extremely large and the dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ consists of instances and labels. Each instance $\mathbf{x}$ matches a label subset of $\mathcal{Y}$, which can also be written as a binary vector $\mathbf{y} \in \{0, 1\}^L$, with the lth element $\mathbf{y}^{(l)}$ representing the presence or absence of label l. So the target of the XML task is to learn a function $F : \mathbf{x} \to \{0, 1\}^L$ that maps an instance to a subset of large-scale labels.

In this study, we focus on taking advantage of gradient boosting to help the deep XML method perform better on tail labels. Moreover, the shared binary classifier is adopted for two main reasons. First, a shared classifier has the potential to help the XML method perform better on the prediction of tail labels since it can utilize the information of head labels to help the classification of tail ones. The results of two typical methods from two types of deep XML methods confirm our above thought, which can be found in Section IV-B. Second, in practice, the base classifier used in the gradient boosting method usually needs to be a weak one due to its better diversity between base classifiers. In contrast, the multiple binary classifiers are not weak as a whole. Besides, the extra memory and computation cost brought by the multiple classifiers are unbearable given such a large number of labels.

III. BoostXML

In this section, we introduce BoostXML for XML with tail labels. The proposed framework is presented in Fig. 2.

A. Boosting Step

As a universal ensemble framework, gradient boosting can be theoretically conducted on any base learner, such as logistic regression, support vector machine (SVM), and multilayer perceptron. In practice, simple and weak base learners usually produce better performance due to the greater diversity among them. To get better prediction results on the XML task, deep-learning methods use powerful text encoders and fully connected layers for text representation and label classification, respectively. Therefore, the direct application of gradient boosting on a deep XML model will not only fail to improve its performance due to strong base learners but also greatly suffer from the calculation and storage cost of the whole model.

To avoid the above problems, BoostXML only takes the shared binary classifier (described in Section II-A) as the base learner, which is indeed a multilayer perceptron and is a natural choice of base learner for it serves as the output layer of a deep learning-based classification method. Applying MLP as the base learner makes sure our model can be trained with gradient descent algorithms in an end-to-end manner. To keep the input of each base learner consistent, BoostXML applies a shared encoder to get label representations. Besides, to carry out better feature transformation before label classification, there are usually multiple fully connected hidden layers in the shared classifier and many neurons in its hidden layers. To maintain the simplicity and weakness of the base learners, BoostXML restricts the number of hidden layers as well as the number of neurons in hidden layers of the shared classifier.[2] Meanwhile, to increase the diversity among the base learners, we randomly initialize each base learner and train them in only a few epochs to guarantee they are as weak as possible. We call the training process of each weak learner as **Boosting Step**.

In each Boosting Step, a new base learner is trained from scratch and then added to the ensemble model. The target of training the new base learner in gradient boosting is to fit the "residual" from the prediction of the current ensemble model. Given the input $\mathbf{x}_i$ of the ith instance, L label-specific representations are obtained. Each representation is a combination of hidden vectors of tokens, which is formally defined as follows:

$$\mathbf{v}_{il} = \sum_{j=1}^{J} a_l(\mathbf{x}_i)^{(j)} \phi_e(\mathbf{x}_i)^{(j)} \tag{4}$$

where $\phi_e(\mathbf{x}_i) \in \mathbb{R}^{J \times H}$ is an intermediate representation function and can be implemented as a text encoder such as

[2]More details about how to restrict an MLP's (base learner) number of hidden layers and the number of neurons in each layer can be found in the implementation details of Section IV-A.

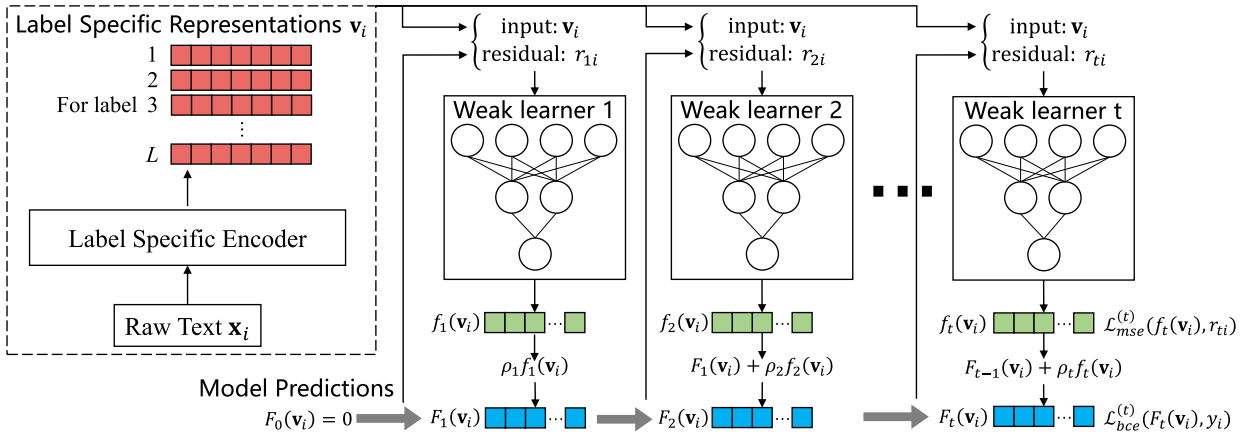


Fig. 2. Overview of the proposed BoostXML.

long short-term memory (LSTM) [11] or transformer [12], which produces J representations, one for each token x_{ij}. J is the length of text, and H is the dimension of hidden vectors. $a_l(\cdot)$ is the weight score function for label l, which meets the constraint of $\sum_{j=1}^{J} a_l(\mathbf{x}_i)^{(j)} = 1$. Specifically, $a_l(\cdot)$ in our work is an attention[3] computed between the embedding of label l and representations of J tokens.

With the label representation $\mathbf{v}_{il}$ of label l in ith instance, the objective function of tth base learner $f_t(\cdot)$ is formally defined as follows:

$$\mathcal{L}^{(t)}_{\text{mse}} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{l=1}^{L}(r_{til} - \rho_t f_t(\mathbf{v}_{il}))^2 \tag{5}$$

where $\mathcal{L}^{(t)}_{\text{mse}}$ is the mean squared error, N and L are the number of instances and labels, respectively, ρ_t is the weight (also known as the boosting rate) of base learner $f_t(\cdot)$, and r_{til} is the residual. For different learning tasks, the form of residual r_{til} will change according to the different choices of loss functions for the task. Note that, different from traditional gradient boosting, the input label representation of base learners in BoostXML is varying with different Boosting Steps since the text encoder is trained together with base learners. This will invalidate previously trained base learners. We will show how to fix this problem in Section III-B. As each label has its own representation, a shared binary classifier is adopted for all labels. To simplify the calculation, we assume the label $y_{il} \in \{-1, +1\}$, and use the logistic loss for binary classification, which is formally given as follows:

$$\mathcal{L}^{(t)}_{\log} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{l=1}^{L}\ln\left(1 + e^{-2y_{il}F_{t-1}(\mathbf{v}_{il})}\right). \tag{6}$$

Given the loss function, according to (3) and (6), the residual r_{til} can be written as follows:

$$r_{til} = -g_{il} = \frac{2y_{il}}{1 + e^{2y_{il}F_{t-1}(\mathbf{v}_{il})}} \tag{7}$$

where g_{il} is the first-order gradient of $\mathcal{L}^{(t)}_{\log}$.

Although using the first-order gradient as the measure of residual r_{til} is enough for the gradient boosting algorithm to converge, the second-order gradient is often introduced to the training process of gradient boosting due to its excellent properties. One of the most important differences between the popular XGBoost [13] and traditional gradient boosting decision trees (GBDTs) [14] is the introduction of the second-order gradient based on Taylor expansion, which helps the algorithm find the optimal or local optimal solution more quickly and accurately. Therefore, instead of just using the first-order gradient, BoostXML leverages the information of both first- and second-order gradients. The second-order gradient h_{il} and the residual r_{til} are given as follows:

$$h_{il} = \frac{4y_{il}^2 e^{2y_{il}F_{t-1}(\mathbf{v}_{il})}}{\left(1 + e^{2y_{il}F_{t-1}(\mathbf{v}_{il})}\right)^2} \tag{8}$$

$$r_{til} = -g_{il}/h_{il} = \frac{1}{2}y_{il}\left(1 + e^{-2y_{il}F_{t-1}(\mathbf{v}_{il})}\right). \tag{9}$$

Due to the very limited number of positive instances, the classification of tail labels faces a severe imbalance problem. Even the shared binary classifier tends to ignore the very few amounts of positive instances of a tail label and classifies them as negative instances. Therefore, for a wrongly classified positive instance, its predicted label and ground truth label are -1 and $+1$, respectively, then the factor $-2y_{il}F_{t-1}(\mathbf{v}_{il})$ will be a positive value. According to (9), the magnitude of the residual r_{til} is magnified exponentially with the above factor, which leads to the further amplification of the MSE loss $\mathcal{L}^{(t)}_{\text{mse}}$. Such magnified loss function can be considered as a variant of the reweighting methods, which helps the classifier pay more attention to the wrongly classified tail labels. Reweighting has proved to be effective for imbalanced classification in many methods, and extensive experiments in this study also prove that BoostXML can improve the model's classification performance on tail labels.

B. Corrective Step

In a traditional boosting framework, the inputs are usually fixed, e.g., structured data of numeric type or text data encoded by term frequency-inverse document frequency (TF-IDF). During the training of the model, each weak learner is greedily learned which means that only the parameters of tth weak learner are updated at boosting step t where all the parameters of previous $t-1$ weak learners remain

[3]We apply the same attention mechanism of AttentionXML [4] to compute label-specific representations.

unchanged. However, there are two major problems with this training paradigm in BoostXML. First, to improve the model's performance, representation learning needs to be performed by deep-learning methods on the original inputs. If we use TF-IDF or pretraining language models to obtain fixed text representations, the performance of the model will be greatly limited. Second, as the training goes on, the previously trained fixed classifier will have a deviation from the constantly updated encoder, which will reduce the overall performance of the whole model.

Therefore, we implement a Corrective Step to address the above problems. Instead of fixing the encoder and previous $t-1$ weak learners, we update the parameters of the whole network through back-propagation after the tth weak learner is trained and added to the whole network. Unlike in Boosting Step, where each weak learner has its own output and loss, the output of the Corrective Step is the ensemble output of all trained weak learners. The objective function of the Corrective Step is the Binary Cross Entropy L_{bce}, which is formally given as follows:

$$\mathcal{L}_{\text{bce}}^{(t)} = -\frac{1}{NL}\sum_{i=1}^{N}\sum_{l=1}^{L}\Big(y_{il}\ln\sigma(F_t(\mathbf{v}_{il}))+(1-y_{il})\ln(1-\sigma(F_t(\mathbf{v}_{il})))\Big) \tag{10}$$

where σ is the sigmoid function. Note that if we let the label $y_{il} \in \{0, 1\}$, the $\mathcal{L}_{\text{bce}}$ is actually equivalent to $\mathcal{L}_{\log}$ [in (6)] with label $y_{il} \in \{-1, 1\}$. We use BCE loss for the Corrective Step and use the logistic loss for the Boosting Step to keep consistent with the literature on gradient boosting and deep learning, respectively.

Corrective Step enables the encoder to update parameters to obtain better representation ability, and at the same time enables each weak classifier to adapt to the change of the encoder. Corrective Step also helps the model to avoid getting stuck in local minima, and further improve the performance of the whole model. In addition, in a traditional boosting framework, the weight ρ_t, or the so-called boosting rate, needs to be specifically updated and fixed after the boosting operation. We incorporate it into the parameters of the model and update it automatically through the Corrective Step.

C. Training and Prediction

In the training process, we alternatively carry out the Boosting Step and the Corrective Step. The Boosting Step trains a new weak learner, while the Corrective Step updates the parameters of the whole network. Successive execution of a Boosting Step and a Corrective Step can be considered a complete iteration. At the beginning of each iteration, a new weak learner is randomly initialized and updated in the Boosting Step, while the encoder and the other learners are fixed. In our implementation, to reduce the computation cost and graphics processing unit (GPU) memory consumption, we apply the same encoder as AttentionXML, which is composed of LSTM and Multilabel Attention. After the training of the weak learner, we update the entire network in the Corrective Step. The iterations continue until the number of weak learners reaches a predefined value. For large-scale datasets, following AttentionXML, BoostXML trains a single deep model for each level of a probabilistic label tree (PLT) [2] in a top-down manner. Readers interested in the implementation details of the label-specific encoder and PLT please refer to the article of AttentionXML [4].

Algorithm 1 Training Procedure of BoostXML

Input:
Training dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$;
Output:
Trained model;
▷ Pretraining Step
Pretrain the backbone network with all instances;
Initialize weights of the encoder model ϕ_e with weights of the encoder part of backbone network;
for $t = 1$ to T **do**
  ▷ Boosting Step
  Randomly initialize the weak learner f_t;
  for $b = 1$ to B_s **do**
    Draw s instances X_b and Y_b from D;
    Get label representations $\mathbf{v}$ by $\phi_e(X_b)$;
    Calculate each residual r_{til} according to Eq. (9);
    Generate label scores of the single learner by $f_t(\mathbf{v})$;
    Compute the loss according to Eq. (5) and update the parameters of f_t;
  end for
  ▷ Corrective Step
  Get the ensemble model F_t by adding f_t to F_{t-1};
  for $b = 1$ to B_s **do**
    Draw s instances X_b and Y_b from D;
    Get label representations $\mathbf{v}$ by $\phi_e(X_b)$;
    Generate label scores of the ensemble model by $F_t(\mathbf{v})$;
    Compute the loss according to Eq. (10) and update the parameters of F_t and ϕ_e;
  end for
end forreturn the trained model.

Except for the above-described Boosting Step and Corrective Step, we further introduce a *Pretraining Step* and a resulting two-stage training scheme, which is inspired by Kang et al.'s work [15]. Specifically, they demonstrate that a model trained without strategies such as class-balanced sampling or loss reweighting learns more generalized representations, for those strategies lead to a decrease in the generalization of the representations due to the close attention to tail labels, which makes the model perform less well on head labels. Therefore, they propose a two-stage training scheme for imbalanced data classification, where the representation learning stage (i.e., the first stage) performs end-to-end network training with regular cross-entropy loss, and classifier adaption stage (i.e., the second stage) is conducted with encoder fixed. We also adopt this two-stage training scheme to alleviate the problem of paying too much attention to tail labels caused by the Boosting Step. Specifically, in the first stage (i.e., Pretraining Step),

TABLE I

DETAILED DATASETS STATISTICS AND PARAMETERS SETTINGS. N_{TRAIN} AND N_{TEST} ARE THE NUMBER OF TRAINING OR TEST INSTANCES, RESPECTIVELY, D_{BOW} IS THE DIMENSION OF BOW FEATURES, L IS THE NUMBER OF LABELS, $\overline{L}$ IS THE AVERAGE NUMBER OF LABELS PER INSTANCE, $\hat{L}$ IS THE AVERAGE NUMBER OF INSTANCES PER LABEL, $\overline{W}_{\text{TRAIN}}$ AND $\overline{W}_{\text{TEST}}$ ARE THE AVERAGE NUMBER OF WORDS PER TRAINING INSTANCE AND TEST INSTANCE, RESPECTIVELY, E IS THE NUMBER OF EPOCHS, T IS THE NUMBER OF WEAK LEARNERS, AND d_h IS THE NUMBER OF NEURONS IN THE HIDDEN LAYER OF EACH WEAK LEARNER (TWO NUMBERS INDICATE TWO HIDDEN LAYERS)

Dataset	N_{train}	N_{test}	D_{bow}	L	$\overline{L}$	$\hat{L}$	$\overline{W}_{train}$	$\overline{W}_{test}$	$E=T$	d_h
EUR-Lex	15,449	3,865	186,104	3,956	5.3	20.79	1248.58	1230.4	30	24
Wiki10-31K	14,146	6,616	101,938	30,938	18.64	8.52	2484.3	2425.45	20	32
AmazonCat-13K	1,186,239	306,782	203,882	13,330	5.04	448.57	246.61	245.98	10	32,16
Wiki-500K	1,779,881	769,421	2,381,304	501,008	4.75	16.86	808.66	808.56	5	64,32
Amazon-3M	1,717,899	742,507	337,067	2,812,281	36.04	22.02	104.08	104.18	5	64,32

BoostXML conducts representation learning via the training procedure of AttentionXML, whose instance representation learning module is called the backbone network. In the second stage, the parameters of the encoder are initialized with the encoder of the model obtained in the first stage. Different from [15], in the second stage, all the weak learners and the encoder are jointly fine-tuned with the Corrective Step rather than being fixed, which helps BoostXML boost the performance of tail labels without seriously hurting the performance of head labels too much.

In the prediction process, the representation of each label is first obtained through the feedforward process of the encoder. Then the output of each learner is obtained based on the representations, and these outputs are summed up with the boosting rates as the weights to get the final output. To better illustrate the training process of BoostXML, especially the execution order of three core steps namely the Pretraining Step, Boosting Step, and Corrective Step, we summarize the whole training procedure in Algorithm 1, where T is the number of weak learners, and B_s is the number of batches sampled for training each weak learner.

IV. EXPERIMENTS

A. Experimental Setup

1) Dataset Description: We consider five popular XML benchmark datasets with raw texts, including three small-scale datasets namely *EUR-Lex*, *Wiki10-31K*, and *AmazonCat-13K*, whose number of labels ranges from thousands to tens of thousands, and two large-scale datasets namely *Wiki-500K* and *Amazon-3M*, whose number of labels ranges from hundreds of thousands to millions. We show the detailed statistics of the datasets in Table I. We keep the splitting of training and test sets the same as the repository page [16].

2) Evaluation Metrics: Precision @k denoted as P @k is widely used to evaluate the performance of XML methods. However, it fails to evaluate prediction performance on tail labels. To better validate the precision of tail labels, we choose the propensity-scored variant of P @k named PSP @k [9] as the evaluation metric, which is defined as follows:

$$\text{PSP}@k := \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{y})} \frac{y_l}{p_l} \tag{11}$$

where $\hat{y}$ is the predicted scores of labels given the instance x, $\text{rank}_k(\hat{y})$ denotes the set of top k scored labels, and y_l equals to 1 if the label l is predicted correctly or equals to 0 otherwise, p_l is the propensity score for label l, and is formally defined as follows:

$$p_l = \frac{1}{1 + Ce^{-A \log(N_l + B)}} \tag{12}$$

where N_l is the number of data points annotated with label l in the observed ground truth dataset, A and B are data specific parameters, and $C = (\log N - 1)(B+1)^A$. With this formulation, $p_l \approx 1$ for head labels and $p_l \ll 1$ for tail labels. For the five datasets used in the article, we set $A = 0.55$, $B = 1.5$ as in PfastreXML [9].

3) Baseline Methods: We compare BoostXML with eight state-of-the-art (SOTA) methods on five benchmark datasets described above. The baseline methods can be divided into non-deep-learning methods and deep-learning methods, where non-deep-learning methods can be further divided into three main categories, according to their ways of tackling the XML problem as follows.

a) Embedding-based methods: Embed labels into a lower dimensional vector space. Due to the low-rank assumption, these methods perform poorly on heavily skewed data.

1) *AnnexML [1]:* AnnexML is the SOTA embedding-based method, which considers label information in the partition stage, and improves the performance on tail-label prediction.

b) Tree-based methods: Divide the original large-scale problem into a sequence of small-scale sub-problems for more efficient training and prediction.

1) *FastXML [8]:* FastXML learns the hierarchy by optimizing the ranking loss function nDCG, which helps in predicting relevant labels.
2) *PfastreXML [9]:* PfastreXML is built upon FastXML, which can handle missing and tail labels by replacing the original loss function with the propensity-scored one.

c) One-versus-all methods: Train a binary classifier for each label independently. To overcome the problems of high-computation cost and slow prediction, these methods often rely on sparse models, label trees, or negative sampling.

1) *DiSMEC [10]:* DiSMEC presents a sparse model with a parameter threshold strategy and employs a double layer of parallelization to scale one-versus-all methods for XML.

TABLE II

PSP AT k COMPARISONS OF BOOSTXML AND OTHER COMPETING METHODS OVER FIVE BENCHMARKS. FOR NON-DEEP-LEARNING METHODS, MOST RESULTS ARE OBTAINED FROM THE EXTREME CLASSIFICATION REPOSITORY [16] OR ATTENTIONXML DIRECTLY. TO MAKE UP FOR THE MISSING RESULTS, WE RAN FASTXML ON EUR-LEX AND WIKI-500K WITH DEFAULT PARAMETERS. ALL RESULTS FOR DEEP-LEARNING METHODS ARE REEXECUTED USING THE AUTHOR'S SOURCE CODE. THE BEST RESULTS ARE IN BOLD, THE SECOND-BEST ONES ARE UNDERLINED, AND THE UNAVAILABLE ONES THAT CANNOT BE REEXECUTED ARE MARKED AS "-". THE LAST ROW IS THE RANKING RESULT OF THE FRIEDMAN TEST, AND THE PART MARKED BY "-" IS CALCULATED WITH 0

Dataset	Metric	AnnexML	FastXML	PfastreXML	Parabel	DiSMEC	Bonsai	AttentionXML	LightXML	BoostXML
EUR-Lex	PSP@1	33.88	30.09	41.68	37.20	38.45	37.33	45.33	40.49	**46.19**
	PSP@3	40.29	36.04	44.01	44.74	46.20	45.40	52.16	50.43	**53.11**
	PSP@5	43.69	39.34	45.73	49.17	50.25	49.92	55.40	54.98	**56.35**
Wiki10-31K	PSP@1	11.90	9.80	19.02	11.68	13.60	11.78	16.31	14.60	**20.29**
	PSP@3	12.76	10.17	18.34	12.73	13.10	13.27	17.67	15.53	**20.05**
	PSP@5	13.58	10.54	18.43	13.69	13.80	14.28	18.86	16.55	**20.59**
AmazonCat-13K	PSP@1	49.04	48.31	**69.52**	50.93	59.10	51.30	53.43	51.87	53.13
	PSP@3	61.13	60.26	**73.22**	64.00	67.10	64.60	68.28	67.25	68.37
	PSP@5	69.64	69.30	75.48	72.08	71.20	72.48	76.03	**76.86**	76.63
Wiki-500K	PSP@1	26.88	15.32	29.20	26.88	31.20	27.46	30.62	30.89	**31.90**
	PSP@3	30.24	18.58	27.60	31.96	33.40	32.25	38.45	**39.60**	39.35
	PSP@5	32.79	20.64	27.70	35.26	37.00	35.48	43.55	43.99	**44.16**
Amazon-3M	PSP@1	11.59	9.77	**21.38**	12.82	-	13.79	14.88	14.94	16.35
	PSP@3	14.07	11.69	**23.22**	15.61	-	16.71	17.74	17.83	19.57
	PSP@5	15.98	13.25	**24.52**	17.73	-	18.87	19.89	19.99	21.91
Friedman test	Rank	7.43	8.80	3.60	6.63	5.40	5.40	3.00	3.13	**1.60**

2) *Parabel [17]:* Parabel dramatically reduces training time by learning balanced binary label trees based on an efficient and informative label representation.
3) *Bonsai [18]:* Bonsai reduces the error propagation with a shallow k-way label tree structure.

d) Deep-learning methods: Learn distributed representations of labels or instances and project them into the high-dimensional label space by nonlinear transformation.

1) *AttentionXML [12]:* AttentionXML leverages an attention mechanism to capture the important tokens for each label. A shared classifier is applied to obtain the label ranking score. For extremely large datasets, models are trained to level by level given the PLT [2]. It is a typical example of deep XML methods with the shared binary classifier.
2) *LightXML [5]:* LightXML is the SOTA deep-learning method, which uses a transformer to encode each input text and assigns each label a classifier to get its score related to the text. For extremely large datasets, the candidate labels are sampled dynamically based on a two-level PLT. It's a typical example of deep XML methods with multiple binary classifiers.

4) Implementation Details: We use PyTorch to implement BoostXML and perform experiments on Tesla V100 GPUs. Models are trained by mini-batch gradient descent with Adam [19]. A small number of the training instances are reserved as the validation set for hyperparameter tuning. There are two key parameters for the model, i.e., the number of weak learners T and the number of neurons d_h in each hidden layer of each weak learner, which are analyzed in Section IV-D. BoostXML uses the same text encoder as AttentionXML, so the hyperparameters of BoostXML's encoder are consistent with those of AttentionXML. The batch size of our method and the baseline methods is adjusted for each dataset for faster training and prediction.

The default settings of T and d_h are presented in Table I. For different datasets, the setting of the numbers of neurons d_h in the hidden layers is related to the number of training instances N_{train} and the dimensionality of input representations H of the base learner, and the setting of the number of weak learners T is related to d_h, which indicates the weakness of a base learner. Specifically, for datasets with a smaller number of instances, such as EUR-Lex and Wiki10-31K, we set H to 256. As to datasets with a larger number of instances, we set H to 512 to ensure the expressiveness of the base learner's input representations. With a larger H and N_{train}, the base learner needs to be slightly strengthened by setting a larger d_h, due to a great number of higher dimensional inputs. For instance, we allow the base learner to have two hidden layers when $H = 512$. As to the setting of the number of weak learners T, we set a smaller T when d_h is larger. In other words, we need fewer base learners when they are not so weak.

B. Experimental Results

The results of PSP@k on five benchmark datasets are reported in Table II. The final predictions are the ensemble (or voting) of three runs, which is inconsistent with AttentionXML. Different from traditional multiclass and multilabel classification tasks, the statistical characteristics of the datasets can affect the model performance to some extent. Therefore, to facilitate the discussion of the results, we group the datasets according to the scale of the number of instances and labels. Specifically, we name three groups of datasets as follows:

1) small-scale instances and Small-scale labels (**SS datasets**): EUR-Lex and Wiki10-31K;

TABLE III
PSP AT k RESULTS OF DEEP-LEARNING METHODS ON THREE PARTS. THE BEST RESULTS ARE IN BOLD

Dataset	Part	$\hat{L}$	AttentionXML			LightXML			BoostXML		
			PSP@1	PSP@3	PSP@5	PSP@1	PSP@3	PSP@5	PSP@1	PSP@3	PSP@5
EUR-Lex	many-shot	137.30	54.01	59.52	67.20	**61.49**	**62.79**	**69.05**	52.52	59.53	68.18
	few-shot	9.77	26.33	35.19	45.43	16.64	30.64	43.47	**28.03**	**36.59**	**46.47**
	one-shot	1.00	1.11	7.53	12.90	0.00	1.08	5.38	**3.33**	**8.60**	**16.13**
Wiki10-31K	many-shot	62.85	34.37	36.28	37.34	**36.09**	**37.30**	**38.51**	34.02	34.12	34.62
	few-shot	3.03	2.90	3.32	4.23	0.08	0.22	0.56	**8.00**	**7.58**	**8.00**
	one-shot	1.00	0.04	0.14	0.33	0.00	0.00	0.00	**0.49**	**1.45**	**2.67**
AmazonCat-13K	many-shot	3953.82	75.52	79.47	82.98	**77.72**	**82.80**	**85.23**	76.10	80.07	83.33
	few-shot	66.30	**7.40**	**24.41**	48.37	2.78	16.63	47.25	6.27	23.46	**49.34**
	one-shot	1.43	0.22	1.51	5.91	0.00	0.00	0.00	**0.23**	**2.64**	**8.89**
Wiki-500K	many-shot	106.71	47.31	55.84	62.23	**49.56**	**56.40**	60.98	46.70	55.84	**62.33**
	few-shot	7.65	17.72	27.29	33.79	17.08	**28.26**	**34.51**	**19.36**	28.20	34.22
	one-shot	1.00	0.40	2.41	4.44	0.89	**5.19**	**9.02**	**0.99**	4.93	8.22
Amazon-3M	many-shot	129.18	28.44	30.28	31.65	**30.15**	**31.90**	**33.13**	26.63	28.71	30.20
	few-shot	11.25	8.73	10.94	12.65	8.26	10.50	12.22	**10.68**	**13.19**	**15.07**
	one-shot	1.00	0.19	0.43	0.73	0.30	0.66	1.08	**0.73**	**1.52**	**2.37**

2) large-scale instances and Large-scale labels (**LL datasets**): Wiki-500K and Amazon-3M; and
3) large-scale instances and Small-scale labels (**LS datasets**): AmazonCat-13K.

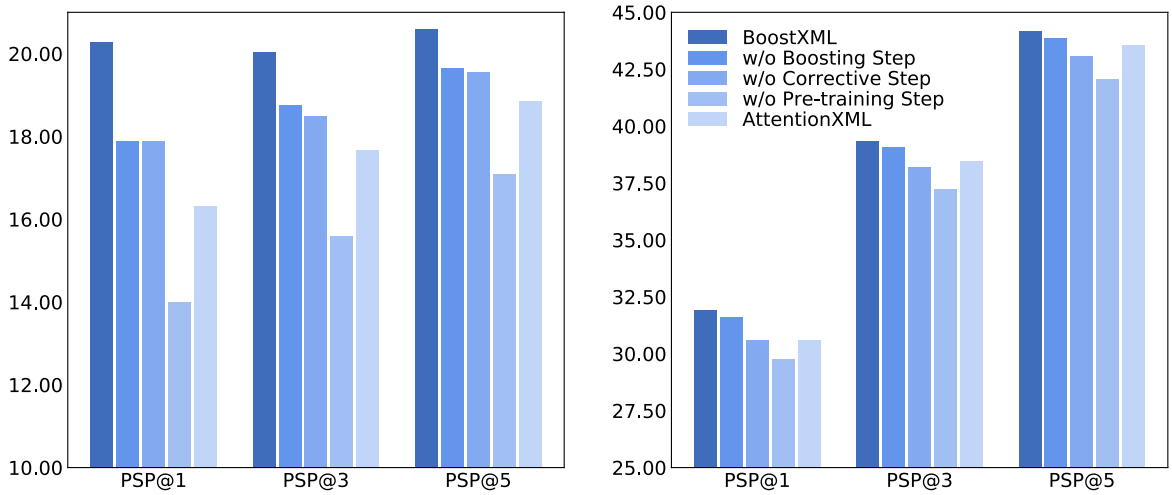
When leaving out PfastreXML, we can find that BoostXML consistently outperforms non-deep baseline methods on all five datasets, except for the PSP @1 of DiSMEC on AmazonCat-13K. The promising results indicate that our method can handle XML with skewed label distribution. Interestingly, the comparison results between deep methods and PfastreXML are counter-intuitive. Specifically, PfastreXML outperforms deep methods on two datasets with large-scale instances. It is well-known that deep methods tend to outperform non-deep methods when data is sufficient. However, these results indicate that deep methods for XML have to consider the fitting problem of tail labels, even when the number of training instances is large. Although BoostXML has narrowed the performance gap between PfastreXML on AmazonCat-13K and Amazon-3M to a certain extent, it is still worth trying to further improve the deep-learning methods against tail labels. Note that, on the remaining dataset that has large-scale instances, i.e., Wiki-500K, BoostXML outperforms PfastreXML significantly. This might be owing to the texts of Wikipedia being much longer than those of Amazon, and deep-learning methods can handle the long-range dependencies of these texts more effectively with representation learning. The above discussion illustrates the advantages and disadvantages of deep-learning methods in tackling XML with tail labels, which validates the necessity of our study. As to deep-learning methods, BoostXML outperforms AttentionXML and LightXML on all **SS** and **LL datasets**, except for PSP @3 on Wiki-500K. On **LS dataset** AmazonCat-13K, AttentionXML has a slightly better PSP @1, and LightXML has a slightly better PSP @3.

We further conduct statistical tests on the results in Table II. Specifically, we conduct the Freidman test on all methods and reject the Null hypothesis ($\mathbf{H_0}$) with a p-value of 0.00000, yielding the ranking results in the last row of Table II. It can be seen from the ranking results that BoostXML significantly outperforms the rest of the baselines. We also conduct the Wilcoxon test for pairwise comparison between BoostXML and the other baselines. We significantly reject $\mathbf{H_0}$ (p-value $<$ 0.01) when comparing BoostXML with other baselines, except for the comparison with PfastreXML where $\mathbf{H_0}$ is accepted with a p-value of 0.39425. This is because as mentioned earlier, BoostXML is inferior to PfastreXML on AmazonCat-13K and Amazon-3M. If we ignore the above-mentioned two datasets, then we can reject $\mathbf{H_0}$ with a p-value of 0.00769 when comparing with PfastreXML. Overall speaking, our method achieves significantly better results than state-of-the-art methods.

However, as described in Section IV-A2, PSP @k is not designed purposefully to evaluate prediction performance on tail labels, therefore, these close PSP @k results might not be enough to compare the deep-learning methods thoroughly. To investigate the performance of deep-learning methods on predicting tail labels, we conduct the following supplementary experiment. Specifically, we divide the labels of each dataset into three parts. Labels are sorted in reverse order according to the number of related instances, and then divided top-down according to the ratio below:

1) many-shot part: 0%–10%;
2) few-shot part: 10%–90%; and
3) one-shot part: 90%-100%.

Table III shows the average number of instances per label $\hat{L}$ and PSP @k results of deep-learning methods on the three parts. For **SS** and **LL datasets**, the average number of instances of each part has a similar distribution. For the many-shot part, $\hat{L}$ is around 100, which is sufficient for the classification task. Hence LightXML performs better on this part because of the encoder of transformer [12]. As to the few-shot part, $\hat{L}$ ranges from a few to a dozen, which is obviously inadequate. Therefore, BoostXML outperforms other methods on **SS** and **LL datasets** because of the boosting



(a) (b)

Fig. 3. PSP at k results of different ablation operations on Wiki10-31K and Wiki-500K, and w/o means "without" the corresponding component. (a) Wiki10-31K. (b) Wiki-500K.

manner, except for the Wiki-500K, where BoostXML performs comparably with LightXML. As to the **LS dataset**, the result is interesting that BoostXML is outperformed by AttentionXML on the few-shot part according to PSP @1 and PSP @3. This phenomenon might be due to the large $\hat{L}$ on the few-shot part of AmazonCat-13K. This less severe imbalance on the few-shot part might reduce the benefits of the model and make it performs slightly worse than AttentionXML. Due to the small $\hat{L}$ of the one-shot part, in which one label usually has only one instance, it is often impossible for most existing methods to learn an accurate classifier. However, BoostXML achieves significant improvement on all datasets except for Wiki-500K.

C. Ablation Study

We investigate the effectiveness of the main modules in BoostXML via this ablation study. The results of PSP @k on the small-scale dataset Wiki10-31K and the large-scale dataset Wiki-500K are illustrated in Fig. 3.

1) Effectiveness of the Boosting Step: For the operation of removing Boosting Step, if we simply remove all of the modules mentioned in Section III-A, BoostXML will degenerate into the original AttentionXML. However, what we are concerned about more is the advantages brought by the characteristic of "residual" optimization. Therefore, we only remove the residual-based optimization part but retain the operation of adding multiple weak learners one by one for training and ensemble. As we can see from the results, the optimization of residual indeed improves the model performance on tail labels.

2) Effectiveness of the Corrective Step: We fix the parameters of the encoder and previously trained weak learners. From the results, we observe a significant decline in the performance of the entire model. For Wiki-500K, the model performance is even slightly worse than that of AttentionXML. This is because the fixed text representations can no longer adapt to the continuous addition of weak learners, which limits the performance of the whole model.

3) Effectiveness of the Pretraining Step: As demonstrated in [20], although rebalancing methods improve the performance of the model for unbalanced classification, they will unexpectedly damage its ability to learn text representations. As mentioned in Section III, BoostXML is also a reweighting method, therefore, it is susceptible to the above issue. The results confirm that training without a pretrained model can greatly reduce its performance, indicating that text representation and pretraining are very important parts of deep-learning methods.

D. Parameter Sensitivity

In BoostXML, the key parameters are the number of neurons d_h in the hidden layer of each weak learner and the number of weak learners T. The parameter d_h affects the weakness of base learners explicitly, and T affects the weak learner implicitly by varying its training degree. To reduce contingency, we investigate the influence of these two parameters on the small-scale dataset Wiki10-31K and the large-scale dataset Wiki-500K.

1) Impact of the Number of Neurons d_h: As mentioned earlier, the base learners in boosting methods should be weak ones. The number of neurons d_h in the hidden layer of each weak learner explicitly controls its weakness. The smaller d_h is, the weaker the learner is. In this experiment, we vary d_h from $2^4 = 16$ to $2^7 = 128$ for Wiki10-31K and from 2^4, $2^3 = 16, 8$ to $2^7, 2^6 = 128, 64$ for Wiki-500K, while maintaining default settings for other parameters. Results are shown in Fig. 4(a) and (b). We can observe from the results that the value of d_h needs to be in an appropriate range since both too large or too small values hinder the performance of the model. A learner with a larger d_h has stronger representation ability and performs better under the same training conditions. However, this may reduce the diversity between learners, which in turn degrades the overall performance of the boosting method. In contrast, when d_h is smaller, the training of each learner is so insufficient that it becomes useless for the entire model to achieve better

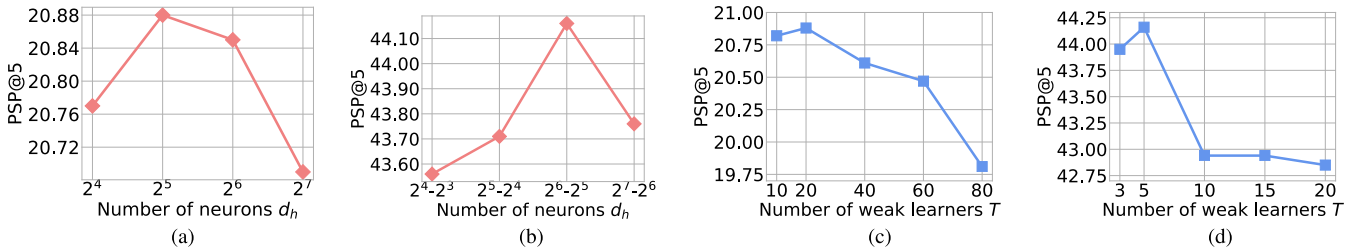


Fig. 4. PSP at 5 results with varying d_h and T on (a) and (c) Wiki10-31K and (b) and (d) Wiki-500K.

performance with the limited number of learners. Thus, we set a smaller $d_h = 2^5 = 32$ as default for Wiki10-31K with more learners and a relatively larger $d_h = 2^6, 2^5 = 64, 32$ for Wiki-500K with fewer learners.

2) Impact of the Number of Weak Learners T: The number of weak learners T is also a key parameter for BoostXML, but unlike d_h's explicit control of the weakness of the learner, T implicitly controls the weakness of the learner by influencing its training degree. To ensure the representation capability of the text encoder remains unchanged, we keep the number of training epochs of the model unchanged and only tune the number of learners, which indicates one learner might be trained for several epochs or several learners might be trained within one epoch. Results are shown in Fig. 4(c) and (d), which show that T also needs to be set in the right range given the total number of training epochs. When T is small, each learner will be trained more thoroughly and performs better, but the reduced number and diversity of learners, in turn, limit the performance of the entire model. When T is large, each learner is not trained sufficiently and is too weak for the model to achieve a satisfactory ensemble effect. Besides, too many learners will greatly increase the computation cost. According to our experience, training one extra learner per epoch is a suitable choice.

E. Case Study

In this section, we further study how BoostXML learns to classify tail labels. As mentioned in Section III-A, BoostXML utilizes "residual" optimization from boosting methods to pay more attention to tail labels that are more likely to be misclassified. To verify that BoostXML indeed focuses on tail labels during training, we record the training process of BoostXML on Wiki10-31K. At the end of each epoch, we make predictions for the training and the testing sets and save the predicted results. We apply the same partition threshold as described in Section IV-B to divide labels into many-shot, few-shot, and one-shot parts, and calculate the precision of prediction results saved in each epoch of the many-shot and few-shot part, respectively. The experimental results are shown in Fig. 5.

Let us first look into the variation of prediction precision on the training set. We can find that after the first epoch of training, the model has reached a good precision for many-shot labels because the text encoder is pretrained and only a weak learner is trained from scratch. However, the precision of the few-shot labels is unsatisfactory, for they are dominated by the many-shot labels, which is the problem that is ignored by all current deep-learning methods. As the training goes on, the model performance on the few-shot labels improves, along with degradation on the many-shot labels. This indicates that the model keeps paying more attention to tail labels, thus improving the prediction precision of tail labels. How to improve the performance of the few-shot labels without sacrificing the performance of the many-shot labels is an interesting problem, which we leave for future work. For the testing set, the variation trend of prediction precision is basically the same as that of the training set, although there is a certain degree of fluctuation.

The precision trend reflects an overall picture of BoostXML for two parts of labels, but it can not reveal the fine-grained learning manner of each label. To further study the influence of BoostXML on specific tail labels during training, we randomly sample a subset of instances from the training set and record the "residual" trends of their tail labels. We observe the residual trends of these tail labels and summarize the frequent trends of labels. The experimental results are shown in Fig. 6.

There are four types of frequent residual trends in Fig. 6, which represent different learning processes of tail labels. Fig. 6(a) and (b) show that the two tail labels catch the model's attention at several starting epochs of the training and are gradually modified in the subsequent epochs. However, as the training continues, the model could make wrong predictions again. Specifically, in Fig. 6(b), the label is predicted incorrectly due to the addition of the 15th weak learner, but it was quickly fixed. This indicates that BoostXML pays more attention to wrongly predicted tail labels and tries to correct them based on the residuals. Different from the residual trends in Fig. 6(a) and (b), the residuals in Fig. 6(c) and (d) are corrected after the first epoch. The residual trend in Fig. 6(d) is a more ideal one as compared with the trend in Fig. 6(c) since it is correctly predicted after the first epoch and will never be wrongly predicted again. This kind of label may be a relatively easy example in tail labels, which does not need much tuning. The small variations of residuals in Fig. 6(c) reflect the BoostXML's capacity of resisting disturbance, which improves our method's robustness and generalization ability.

F. Computation Time and Model Size

To make deep-learning methods suitable to the XML task, their computation cost must be controlled within an acceptable range. Therefore, in this section, we discuss the time and space complexity of BoostXML. As BoostXML and AttentionXML

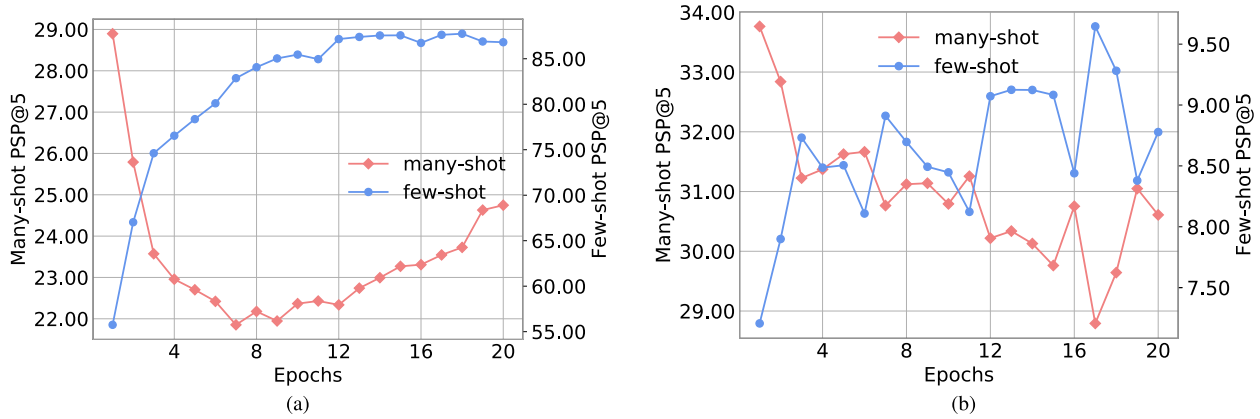


Fig. 5. PSP at 5 comparisons of many and few-shot parts on the training and testing part of Wiki10-31K. (a) Training and (b) testing dataset.

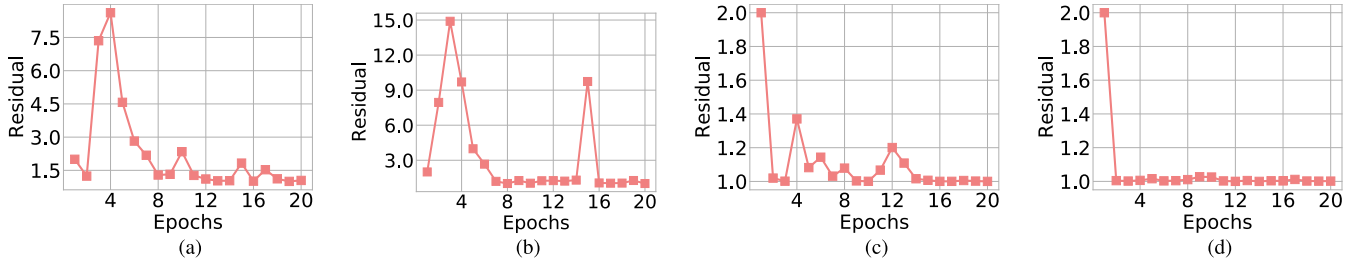


Fig. 6. Four main "residual" evolving types on tail labels recorded in the training process of Wiki10-31K. (a) Instance:10487 label:5219. (b) Instance:13577 label:26807. (c) Instance:6914 label:20319. (d) Instance:4164 label:29278.

TABLE IV

COMPUTATION TIME AND MODEL SIZE. T_{TRAIN} IS THE OVERALL TRAINING HOURS. S_{TEST} IS THE AVERAGE TIME REQUIRED TO PREDICT EACH INSTANCE. THE UNIT OF S_{TEST} IS MILLISECONDS PER INSTANCE (MS/INSTANCE). M IS THE MODEL SIZE IN GB

Dataset		AttentionXML-1	BoostXML-1
Wiki10-31K	T_{train}	1.09	2.09(+91.74%)
	S_{test}	4.49	7.63(+69.93%)
	M	0.62	0.62(+0.00%)
Wiki-500K	T_{train}	35.07	38.95(+11.06%)
	S_{test}	3.49	3.18(-8.88%)
	M	3.09	3.08(-0.32%)

use the same text encoder, we compare the time[4] and space complexity of their single model named BoostXML-1 and AttentionXML-1, respectively, i.e., we do not compare the ensemble model of three runs. The training process of BoostXML mainly differs from that of AttentionXML in the extra Boosting Step and extra weak learners in the Corrective Step. In Boosting Step, the backpropagation is only conducted on several weak learners, while the text encoder is only used in the feedforward process. Therefore, the additional computation time cost of Boosting Step is mainly the inference time of text representations plus the training time of weak learners, which is much less than the end-to-end training of both the encoder and classifiers. The additional time cost of tuning extra weak learners during the Corrective Step is negligible as compared with the overall time cost of the Corrective Step, which is nearly the same as AttentionXML's training time cost.

[4]Note that we do not include the pretrain step of BoostXML into the training time comparison, for it is practically omitted given the trained AttentionXML.

Therefore, the number of weak learners T is one of the most important factors that influence the extra training time cost of BoostXML as compared with that of AttentionXML.

To facilitate the comparison of the two models' testing time, we assume the weak learners are MLPs with only one hidden layer, and the number of neurons in its hidden layer is d_h. For a dataset with a label space size of L, the time complexity of obtaining a label's prediction score with T above MLPs is $O(Td_hL)$. In AttentionXML-1, $T = 1$ but d_h is large, while in BoostXML-1, $T > 1$ but d_h is small. Therefore, for different datasets and different parameter settings (i.e., T and d_h), BoostXML might have larger or smaller time and space complexities as compared with AttentionXML.

Apart from the above analysis, to compare the actual time and memory consumption empirically, we conduct experiments on the small-scale dataset Wiki10-31K and the large-scale dataset Wiki-500K respectively under the same hardware environment, and the results are shown in Table IV. For Wiki10-31K, BoostXML takes more time to train and predict than AttentionXML, because the number of labels ($L = 30\,938$) needs to predict on Wiki10-31K is not small due to the absence of the Label Clustering stage. Therefore the training time cost of each weak learner is not small. Besides, there are more weak learners (i.e., $T = 20$ on Wiki10-31K than on large-scale datasets. Thus, although the d_h of BoostXML has been reduced, the computation cost brought by large T and L has greatly improved its computational time. However, for large-scale datasets, such as Wiki-500K, since the size of the label space has been broken down to smaller values with PLT in the Label Clustering stage, i.e., $L_0 = 8192$ and $L_1 = 930$, and $T = 5$ is also small. Therefore, the computation cost on Wiki-500K for BoostXML-1's classifier is

smaller than for AttentionXML-1's classifier and BoostXML-1 predicts faster than AttentionXML-1. As can be seen from BoostXML's predicting time on Wiki-500K, the additional computation cost of BoostXML's Boosting Step is not significant indeed. Although BoostXML has non-negligible additional computation costs on small-scale datasets such as Wiki10-31K, this has limited practical influence for its original computation cost is not large. As to the space complexity of the model, since the MLP itself has a small number of parameters, the overall size of the model will not change much, therefore, boosting brings no additional storage burden to the model.

V. Related Work

A. Extreme Multilabel Classification

In recent years, various methods have been proposed to tackle the problem of the huge computation cost in the XML task. These methods typically use handcraft features as model inputs, and can generally be considered traditional methods. With the development of deep learning in recent years, more and more methods use more advanced encoders to perform representation learning on the raw data. These methods greatly improve the model performance on XML.

1) Traditional Methods: Usually aims to reduce the cost of learning classifiers in the extremely large label space. These methods can be roughly divided into embedding-based methods, tree-based methods, and one-versus-all methods. Most embedding-based methods reduce the effective number of labels with the low-rank label matrix assumption. Generally, they assume the label can be represented with a low-dimensional vector named embedding. In order to obtain better label and feature embeddings, methods such as sparse local embeddings for extreme classification (SLEEC) [7], AnnexML [1], adaptive extreme Feature agglomeration (DEFRAG) [21], and ExMLDS [22] are proposed. K-nearest neighbors (KNN) graph, clustering, and a few other techniques are applied to reduce the computation cost. Tree-based methods such as FastXML [8], PfastreXML [2], and SwiftXML [23] aim for faster prediction by recursively partitioning the feature space and controlling the number of active labels in each region of feature space. In addition, PLT [2] is used to partition labels for large-scale datasets in deep-learning methods due to its low computation cost and good performance. One-versus-all methods are one of the most popular strategies for multilabel classification, but they cannot be directly applied to XML due to their huge computation cost. To solve this problem, some methods like DiSMEC [10] or ProXML [24] use a sparse model or l_1-regularized Hamming loss to reduce the computation cost. Other methods like Parabel [17] and Slice [3] use additional tree structures or sampling methods to further reduce the training and prediction time of one-versus-all methods.

2) Deep-Learning Methods: Become more and more popular in recent years due to the powerful representation learning ability and the state-of-the-art performance in many real-world multilabel applications. As mentioned before, most deep-learning methods can be summarized into two types according to their classifiers. Basically, XML-CNN [25], X-transformer [26], and LightXML [5] all apply multiple binary classifiers, with differences in detail of label clustering and the encoder that obtains instance representations. As a representative method that applies the shared binary classifier, AttentionXML [4] has some implicit optimization for tail labels, due to its sharing a classifier for all labels. There are also a few exceptions. For example, ASTEC [6] uses a negative sampling method to reduce computation cost and takes BOW as input, which is different from most deep-learning methods. This method can not be classified into any one of the two paradigms.

B. Boosting Methods

The idea of boosting derives from the probably approximately correct (PAC) learning model, where Kearns and Valiant showed that the learning ability of weak learners is equivalent to that of strong learners [27]. Based on this idea, AdaBoost [28] is proposed, which takes advantage of weak learners and gets rid of the dependence on the prior knowledge of weak learners. Recent studies [29], [30] have improved Adaboost to alleviate the problem that it increases the weights of wrongly classified instances in a training process and apply Adaboost to scenarios such as transaction fraud detection. To further improve the usability of Adaboost, Gradient Boosting Machine [14] is proposed, whose idea is to optimize the cost function with the direction of negative gradient in function space. When the decision trees are used as the weak learners, the classical GBDT [14] algorithm is derived. Although GBDT enhances the performance of Adaboost, it still has several limitations. Its variants, such as XGboost [13], LightGBM [31], and CatBoost [32] are proposed to address these problems and further improve the calculation speed and accuracy. Sigrist [33] further investigates the relationship between gradient descent and second-order Newton updates, and experimentally shows that Newton boosting outperforms gradient and hybrid gradient-Newton boosting.

Although decision trees are usually used as weak learners of boosting methods, they cannot be trained in an end-to-end manner, and are not suited for visual or text data. Therefore, combining boosting methods with neural networks has attracted much attention. Schwenk and Bengio [34] are among the first to try to combine Adaboost with neural networks, which achieves better performance than Adaboost with decision trees. Recently, AdaNet [35] is proposed to build neural networks layer by layer for the image classification task. AdaBoost-CNN [36] integrates the capability of AdaBoost with CNN, which can deal with large imbalanced datasets with high accuracy. Different from them, GrowNet [37] takes neural networks as weak learners of gradient boosting and proposes a Corrective Step to improve the performance of the model. Different from GrowNet, our Corrective Step not only updates the parameters of weak learners but also the parameters of the text encoder, which greatly improves the application scope of boosting methods. Compared to emerging methods along this line [38], [39], [40], we highlight that our boosting method can help deep-learning methods achieve better performance on tail-label prediction for the XML task.

VI. Conclusion

In this article, we propose a novel deep-learning method named BoostXML to alleviate the problem of tail labels in the XML task, which combines gradient boosting and neural networks. BoostXML follows the boost-style learning scheme. During the training of each weak learner, the "residual" is optimized to help the model pay more attention to correct the prediction errors from tail labels, which improves the model performance on tail labels. BoostXML also exploits a Corrective Step to update the entire network in an end-to-end manner so that the representations of the text encoder adapt to subsequent weak learners. Experiments on five benchmark datasets demonstrate the improvements of BoostXML in predicting tail labels.

Acknowledgment

The authors thank Yunhui Liu from Tsinghua University, Beijing, China, for his valuable contributions to the experiments and language polishing in this article.

References

[1] Y. Tagami, "AnnexML: Approximate nearest neighbor search for extreme multi-label classification," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 455–464.

[2] K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier, "Extreme F-measure maximization using sparse probability estimates," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1435–1444.

[3] H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma, "Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Jan. 2019, pp. 528–536.

[4] R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu, "AttentionXML: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 5820–5830.

[5] T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang, "LightXML: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 9, pp. 7987–7994.

[6] K. Dahiya et al., "DeepXML: A deep extreme multi-label learning framework applied to short text documents," in *Proc. 14th ACM Int. Conf. Web Search Data Mining*, Mar. 2021, pp. 31–39.

[7] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain, "Sparse local embeddings for extreme multi-label classification," in *Proc. NIPS*, vol. 29, 2015, pp. 730–738.

[8] Y. Prabhu and M. Varma, "FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 263–272.

[9] H. Jain, Y. Prabhu, and M. Varma, "Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 935–944.

[10] R. Babbar and B. Schölkopf, "DiSMEC: Distributed sparse machines for extreme multi-label classification," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, Feb. 2017, pp. 721–729.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[12] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[15] B. Kang et al., "Decoupling representation and classifier for long-tailed recognition," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–16.

[16] K. Bhatia et al., "The extreme classification repository: Multi-label datasets and code," 2016. [Online]. Available: http://manikvarma.org/downloads/XC/XMLRepository.html

[17] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma, "Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising," in *Proc. World Wide Web Conf. World Wide Web*, 2018, pp. 993–1002.

[18] S. Khandagale, H. Xiao, and R. Babbar, "Bonsai: Diverse and shallow trees for extreme multi-label classification," *Mach. Learn.*, vol. 109, no. 11, pp. 2099–2119, Nov. 2020.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[20] B. Zhou, Q. Cui, X. Wei, and Z. Chen, "BBN: Bilateral-branch network with cumulative learning for long-tailed visual recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 9716–9725.

[21] A. Jalan and P. Kar, "Accelerating extreme classification via adaptive feature agglomeration," 2019, *arXiv:1905.11769*.

[22] V. Gupta, R. Wadbude, N. Natarajan, H. Karnick, P. Jain, and P. Rai, "Distributional semantics meets multi-label learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3747–3754.

[23] Y. Prabhu et al., "Extreme multi-label learning with label features for warm-start tagging, ranking & recommendation," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, Feb. 2018, pp. 441–449.

[24] R. Babbar and B. Schölkopf, "Data scarcity, robustness and extreme multi-label classification," *Mach. Learn.*, vol. 108, nos. 8–9, pp. 1329–1351, Sep. 2019.

[25] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang, "Deep learning for extreme multi-label text classification," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Aug. 2017, pp. 115–124.

[26] W.-C. Chang, H.-F. Yu, K. Zhong, Y. Yang, and I. S. Dhillon, "Taming pretrained transformers for extreme multi-label text classification," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 3163–3171.

[27] M. Kearns, "Learning Boolean formulae or finite automata is as hard as factoring," Aikem Comput. Lab., Harvard Univ., Cambridge, MA, USA, Tech. Rep. TR-14-88, 1988.

[28] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[29] L. Zheng, G. Liu, C. Yan, C. Jiang, M. Zhou, and M. Li, "Improved TrAdaBoost and its application to transaction fraud detection," *IEEE Trans. Computat. Social Syst.*, vol. 7, no. 5, pp. 1304–1316, Oct. 2020.

[30] C. Yang, G. Liu, C. Yan, and C. Jiang, "A clustering-based flexible weighting method in AdaBoost and its application to transaction fraud detection," *Sci. China Inf. Sci.*, vol. 64, no. 12, pp. 1–11, Dec. 2021.

[31] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 3146–3154.

[32] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," 2017, *arXiv:1706.09516*.

[33] F. Sigrist, "Gradient and Newton boosting for classification and regression," *Exp. Syst. Appl.*, vol. 167, Apr. 2021, Art. no. 114080.

[34] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural Comput.*, vol. 12, no. 8, pp. 1869–1887, Aug. 2000.

[35] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, "AdaNet: Adaptive structural learning of artificial neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 874–883.

[36] A. Taherkhani, G. Cosma, and T. M. McGinnity, "AdaBoost-CNN: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning," *Neurocomputing*, vol. 404, pp. 351–366, Sep. 2020.

[37] S. Badirli, X. Liu, Z. Xing, A. Bhowmik, K. Doan, and S. S. Keerthi, "Gradient boosting neural networks: GrowNet," 2020, *arXiv:2002.07971*.

[38] M. Moghimi, S. J. Belongie, M. J. Saberian, J. Yang, N. Vasconcelos, and L.-J. Li, "Boosted convolutional neural networks," in *Proc. BMVC*, vol. 5, 2016, pp. 1–13.

[39] S. Emami and G. Martínez-Muñoz, "Sequential training of neural networks with gradient boosting," 2019, *arXiv:1909.12098*.

[40] S. Ivanov and L. Prokhorenkova, "Boost then convolve: Gradient boosting meets graph neural networks," 2021, *arXiv:2101.08543*.

Fengzhi Li received the bachelor's degree from Beihang University, Beijing, China, in 2018, where he is currently pursuing the Ph.D. degree with the School of Economics and Management.

His research interests include multilabel learning and dynamic graph representation learning.

Yuan Zuo received the Ph.D. degree from Beihang University, Beijing, China, in 2017.

He is currently an Associate Professor with the Information Systems Department, Beihang University. He is also a member of the MIIT Key Laboratory of Data Intelligence and Management (DIG). His work has been published in refereed journals and conference proceedings, including IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD), and IEEE International Conference on Data Mining (ICDM). His research interests generally lie in the areas of data mining and machine learning, with special interests in learning representations, information extraction, and explainable recommendation.

Hao Lin (Member, IEEE) received the B.S. and Ph.D. degrees from Beihang University, Beijing, China, in 2013 and 2020, respectively.

He was a Post-Doctoral Researcher with the Technical University of Munich, Munich, Germany. He is currently an Assistant Professor with the School of Economics and Management, Beihang University. He is also a member of the MIIT Key Laboratory of Data Intelligence and Management (DIG). His work has been published in refereed journals and conference proceedings, including IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, *ACM Transactions on Information Systems*, and AAAI Conference on Artificial Intelligence (AAAI). His research interests generally lie in the areas of data mining and machine learning, with special interests in temporal data analysis, heterogeneous data fusion, and anomaly detection.

Junjie Wu received the Ph.D. degree in management science and engineering from Tsinghua University, Beijing, China, in 2007.

He is currently a Full Professor with the School of Economics and Management, Beihang University, Beijing. He is also the Director of the MIIT Key Laboratory of Data Intelligence and Management (DIG) and the Institute of Artificial Intelligence for Management (AIM). His general area of research is machine learning with applications in social, urban, and financial areas.

Dr. Wu was a recipient of various nationwide academic awards including NSFC Distinguished Young Scholars and the National Excellent Doctoral Dissertation.