Learning Extreme Multi-label Tree-classifier via Nearest Neighbor Graph Partitioning

Yukihiro Tagami
Yahoo Japan Corporation
Tokyo, Japan
yutagami@yahoo-corp.jp

ABSTRACT

Web scale classification problems, such as Web page tagging and E-commerce product recommendation, are typically regarded as multi-label classification with an extremely large number of labels. In this paper, we propose GPT, which is a novel tree-based approach for extreme multi-label learning. GPT recursively splits a feature space with a hyperplane at each internal node, considering approximate k-nearest neighbor graph on the label space. We learn the linear binary classifiers using a simple optimization procedure. We conducted evaluations on several large-scale real-world data sets and compared our proposed method with recent state-of-the-art methods. Experimental results demonstrate the effectiveness of our proposed method.

Keywords

Extreme multi-label classification, approximate k-nearest neighbor graph.

1. INTRODUCTION

In this paper, we address a multi-label classification problem with an extremely large label set (10^4 to 10^6). We consider a data set $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, which consists of N training points, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^M$ is the M-dimensional feature vector and $\boldsymbol{y}_i \in \mathcal{Y} \subseteq \{0,1\}^L$ is the corresponding L-dimensional label vector. $y_{ij} = 1$ if the i-th sample has the j-th label and $y_{ij} = 0$ otherwise. Multi-label learning aims to build a classifier, $f : \mathbb{R}^M \to \{0,1\}^L$, which accurately predicts the label vector for a given sample.

For example, the traditional one-versus-rest technique, which learns an independent classifier for each label, needs to train the same number of binary classifiers as labels. Furthermore, at the prediction phase of this approach, all binary classifiers are applied to each test point. Thus, as the number of labels increases, this naive approach might be computationally expensive in terms of both training and prediction steps.

To overcome the above problem, some methods have been proposed [5, 2, 3]. FastXML [5] is a tree-based extreme multi-label classifier. This method learns an ensemble of multiple trees using random initialization. At the training phase of each tree, FastXML recursively partitions the feature space corresponding to the internal node by using a linear classifier optimized for nDCG-based ranking loss. A test point traverses the tree from the root node to a leaf node, and FastXML then predicts labels using empirical label distribution of training points in the leaf node.

2. PROPOSED METHOD

As described above, FastXML makes a prediction using the labels of training points in the leaf node that a test point has reached. This prediction procedure is considered as a k-nearest neighbor algorithm using all training points in the feature subspace of the leaf node. From this point of view, our proposed method attempts to split each internal node of a tree while keeping as many "nearest neighbors" on the label space as possible. Thus, we call our proposed method a "graph partitioning tree" (GPT).

To split data points at each internal node of a tree, we first construct a k-nearest neighbor graph on the label space. Each vertex of the graph corresponds to a training point. A directed edge connects from the i-th vertex to the j-th one if the j-th point is included in the set of the "nearest neighbors" of the i-th point on the label space. In this paper, the set of "nearest neighbors" of the i-th sample is defined using the inner product between normalized label vectors $\boldsymbol{y}/|\boldsymbol{y}|$ as follows.

$$\mathcal{N}_i := \mathop{\arg\max}_{S:|S|=k, i\notin S} \sum_{j\in S} \frac{\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{y}_j}{|\boldsymbol{y}_i||\boldsymbol{y}_j|},$$

where S is the index set in which the number of elements equals k and $|\boldsymbol{y}_i| = \sum_j y_{ij}$ is the number of labels that a data point has.

Since the label vectors $\boldsymbol{y}$ are typically sparse, we can efficiently find the nearest neighbors $\mathcal{N}_i$ by using an inverted index. The estimated computational cost for all data points is $\sum_{j=1}^{L} n_j(n_j - 1)/2$, where n_j is the number of data points that have the j-th label. However, if a few n_j corresponding to "head" labels are near N, which means almost all data points have the same label, the above cost reaches $\mathcal{O}(N^2)$ at the root node. Therefore, we focus on tail labels and ignore some head labels. We only consider tail labels under the condition $n_j < n_{th}$ to find approximate nearest neighbors $\tilde{\mathcal{N}}_i$, using the threshold parameter n_{th}. Using this simple

Table 1: Statistics of datasets

Dataset	#Train N	#Test N_{test}	#Features M	#Labels L
AmazonCat-13K	1,186,239	306,782	203,882	13,330
Wiki10-31K	14,146	6,616	101,938	30,938
Delicious-200K	196,606	100,095	782,585	205,443
WikiLSHTC-325K	1,778,351	587,084	1,617,899	325,056
Amazon-670K	490,449	153,025	135,909	670,091

approximation, we only consider tail labels when the node is close to the root and has a lot of training points. In contrast, when the node is near a leaf and includes a relatively small amount of data, we construct the graph using all labels.

After an approximate k-nearest neighbor graph is constructed using $\tilde{\mathcal{N}}_i$, we want to learn a linear classifier by finding the minimum graph cut. In contrast with the common min-cut problem, we need to partition the graph with a hyperplane $\boldsymbol{w}$ on the feature space to predict unknown test points. Thus, in a similar way to stochastic k-means clustering, we sequentially maximize the following objective function for each i-th sample.

$$\sum_{j \in \tilde{\mathcal{N}}_i} \log \sigma(c_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_j) + \sum_{j \in S^-} \log \sigma(-c_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_j) - \lambda |\boldsymbol{w}|_1,$$

where $\sigma(z) = 1/(1 + \exp(-z))$ is a sigmoid function, S^- is the set of indices randomly selected from data points that the node of the tree has, and λ is a regularization parameter. c_i is an indicator variable representing which side of the partition the i-th point belongs to. $c_i = +1$ if $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i \geq 0$ and $c_i = -1$ otherwise.

We learn the linear separator $\boldsymbol{w}$ using FTRL-Proximal algorithm [4] withd AdaGrad [1] learning rate scheduling.

3. EXPERIMENT

Data sets. We evaluated our proposed method using five large scale multi-label data sets. These data sets were provided by the Extreme Classification Repository[1] and had already been pre-processed and separated into training and test sets. The statistics for the data sets are summarized in Table 1.

Baselines and evaluation setting. We compared GPT with several state-of-the-art tree-based classifiers: FastXML [5], PfastreXML [2], and PLT [3]. PfastreXML is the method that extends FastXML to improve tail label predictions.

We evaluated the performance of the methods with precision at k ($k \in \{1, 3, 5\}$), which is a widely adopted metrics for extreme multi-label classification and ranking tasks: P@k := $\frac{1}{kN_{test}} \sum_{i=1}^{N_{test}} \sum_{l=1}^{k} y_{i\pi(l)}$. Here, $\pi(k) = j$ means that the j-th label is ranked in the k-th position by the predicted score.

In all experiments, we used default hyper-parameters to train GPT; the number of learners: 50 (the same as the default value of FastXML and PfastreXML), the threshold parameter for finding approximate nearest neighbors: $n_{th} = 50$, the number of approximate nearest neighbors and randomly sampled points used in learning: $|\tilde{\mathcal{N}}_i| = |S^-| = 10$, the number of epochs for optimization $\boldsymbol{w}$: 10, the initial learning rate: $\eta_0 = 0.1$, the L_1 regularization parameter: $\lambda = 4.0$, and the maximum number of data points allowed in a leaf node: 10.

[1]`https://manikvarma.github.io/downloads/XC/XMLRepository.html`

Table 2: Experimental results

Dataset		GPT	FastXML	PfastreXML	PLT
AmazonCat-13K	P@1	0.9084	**0.9310**	0.8994	0.9147
	P@3	0.7676	**0.7818**	0.7724	0.7584
	P@5	0.6255	0.6338	**0.6353**	0.6102
Wiki10-31K	P@1	**0.8476**	0.8295	0.8263	0.8434
	P@3	**0.7322**	0.6756	0.6874	0.7234
	P@5	**0.6320**	0.5770	0.6006	0.6272
Delicious-200K	P@1	**0.4746**	0.4320	0.3762	0.4537
	P@3	**0.4165**	0.3868	0.3562	0.3894
	P@5	**0.3871**	0.3621	0.3403	0.3588
WikiLSHTC-325K	P@1	**0.6336**	0.4975	0.5810	0.4567
	P@3	**0.3997**	0.3310	0.3761	0.2913
	P@5	**0.2906**	0.2445	0.2769	0.2195
Amazon-670K	P@1	**0.4236**	0.3697	0.3919	0.3665
	P@3	**0.3725**	0.3332	0.3584	0.3212
	P@5	**0.3384**	0.3053	0.3321	0.2885

We used C++ implementations provided by the authors for FastXML and PfastreXML. In this case, the suggested hyper-parameters were adopted. Since we use ordinary (not propensity scored) precision at k as evaluation metrics, propensity scores of PfastreXML are set the same value for all labels. For PLT, we only referred to the values reported in the original paper [3].

Results. The experimental results are summarized in Table 2. The **bold** elements indicate the best performance of all methods. Our proposed GPT achieved the best performances of the four tree-based methods for almost all data sets. For example, on the WikiLSHTC-325K data set, GPT was superior to the PfastreXML, which is the second best, by approximately 5% in absolute terms of P@1. On the dataset, our C++ implementation of GPT made predictions in 0.9 milliseconds per test point on a single CPU thread.

These experimental results indicate that our proposed graph partitioning method is promising for extreme multi-label classification.

4. REFERENCES

[1] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 2011.

[2] H. Jain, Y. Prabhu, and M. Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *KDD*, 2016.

[3] K. Jasinska, K. Dembczyński, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hüllermeier. Extreme f-measure maximization using sparse probability estimates. In *ICML*, 2016.

[4] H. B. McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *AISTATS*, 2011.

[5] Y. Prabhu and M. Varma. FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *KDD*, 2014.