Hierarchical Approach Multi-label Classification

Yilu Peng
Department of Electrical and Computer Engineering
New York University
Brooklyn, NY 11201
yp1232@nyu.edu

Abstract

The objective in extreme multi-label classification is to learn a classifier that can automatically tag a data point with the most relevant *subset* of labels from a extremely large label set. This paper discusses the tree structured multi-label classification algorithm FastXML[1] and the choice of loss functions in FastXML. The choice of the loss function is critical in the objective function for node partition. FastXML uses the nDCG (normalized Discounted Cumulative Gain) [2] score loss to decide the separation of left and right node. Since the nDCG loss function is a discontinuous function, it's hard to optimize. A logistic loss term is added to the objective function so that the linear separator **w** is easier to optimize. I propose that we can also use cross entropy loss instead of logistic loss when optimizing the rank sensitive loss function for FastXML. Also, the nDCG score loss can be replaced by other ranking losses such as precision score loss. Experiments are done to compare the nDCG loss and Precision loss. Results revealed that nDCG loss achives higher accuracy compared to Precision loss.

1 Introduction

1.1 Multi-Label Classification

Extreme multi-label learning is an important research problem as it has many applications in tagging, recommendation and ranking. For instance, there are more than a million labels (tags) on Wikipedia and one might wish to build an extreme multi-label classifier that tags a new article or web page with the subset of most relevant Wikipedia labels. Similarly, given a user's buying or viewing history, one might wish to build an extreme multi-label classifier that recommends the subset of millions of items that the user might wish to buy or view next. In general, one can reformulate ranking and recommendation problems as extreme classification tasks by treating each item to be ranked/recommended as a separate label, learning an extreme multi-label classifier that maps a user's feature vector to a set of labels, and then using the classifier to predict the subset of items that should be ranked/recommended to each user.

To tackle the problem of extreme multi-class learning, it is computationally expensive to train one-against-all classfiers for each possible label. A hierarchical approach is developed to reduce the multiclass problem to a set of binary classification problems.[A. Choromanska and J. Langford. Logarithmic Time Online Multiclass prediction[3]]

The objective in hierarchical multi-label learning is different from hierarchical multi-class classification which aims to predict only a single label. Multi-label classfication aims to predict a set of relevant labels for a given data point.

Tree-structured multi-label classification methods learn a hierarchy from training data as follows: The root node is initialized to contain the entire label set. A node partitioning formulation is then optimized to determine which labels should be assigned to the left child and which to the right. Nodes are recursively partitioned till each leaf contains only a small number of labels. During prediction, a novel data point is passed down the tree until it reaches a leaf node. A base multi-label classifier of choice can then be applied to the data point focusing exclusively on the leaf node label distribution. This leads to prediction costs that are sub-linear or even logarithmic if the tree is balanced.

1.2 Related Work

Embedding-based Approach

Embedding methods exploit label correlations and sparsity to compress the number of labels. The 1-vs-All strategy can then be applied and the cost of training and prediction in the compressed space are reduced respetively. Methods mainly differ in the choice of compression and decompression techniques.

Embedding methods have many advantages including simplicity, ease of implementation, strong theoretical foundations, the ability to handle label correlations, the ability to adapt to online and incremental scenarios, etc. Unfortunately, embedding methods can also pay a heavy price in terms of prediction accuracy due to the loss of information during the compression phase. None of the embedding methods developed so far has been able to consistently outperform the 1-vs-All baseline when $\hat{L}$ (number of labels after compression) $\approx \log(L)$ (L: number of labels).

Tree-Structured Approach

The Label Partitioning by Sub-linear Ranking (LPSR) method of [4] focussed on reducing the prediction time by learning a hierarchy over a base classifier or ranker. First, a base multi-label classifier was learnt for the entire label set. This step governs the overall training complexity and prediction accuracy. Next, a hierarchy was learnt in terms of a single binary tree. Nodes in the tree were grown by partitioning the node's data points into 2 clusters, corresponding to the left and the right child, using a variant of k-means over the feature vectors. The tree was grown until each leaf node had a small number of data points and corresponding labels. Finally, a relaxed integer program was optimized at each leaf node via gradient descent to activate a subset of the labels present at the node. During prediction, a novel point was passed down the tree until it reached a leaf node. Predictions were then made using the base classifier restricted to the set of active leaf node labels.

The Multi-label Random Forest (MLRF) approach of [5] did not need to learn a base classifier. Instead, an ensemble of randomized trees was learnt. Nodes were partitioned into a left and a right child by brute force optimization of a multi-label variant of the Gini index de ned over the set of positive labels in the node. Trees were grown until each leaf node had only a few labels present. During testing, a novel point was passed down each tree and predictions were made by aggregating all the leaf node label distributions.

Both LPSR and MLRF have high training costs. LPSR needs to train an accurate base multi-label classifier, perform hierarchical k-means clustering and solve a label assignment problem at each leaf node. MLRF needs to learn an ensemble of trees where the cost of growing each node is high. In particular, while training in high dimensional spaces, random forests need to sample a large number of features at each node in order to achieve a good quality, balanced split (extremely random trees [6] and other variants do not work in extreme classification scenarios as they learn imbalanced splits). Furthermore, brute force optimization of the Gini index or entropy over each feature is expensive when there are a large number of training points and labels. All in all, accurate LPSR and MLRF training can require large clusters with up to a thousand nodes in the case of MLRF.

2 FastXML

2.1 Overview

FastXML learns an ensemble of trees during training. Trees are grown by recursively partitioning nodes starting at the root until each tree is fully grown. Nodes are split by learning a separating hyperplane which partitions training points between a left and a right child. The FastXML hyperplane is learnt by optimizing nDCG such that each training point's relevant labels are ranked as highly as possible in its partition. Node partitioning terminates when a node contains less than a user specified number of points.

Leaf nodes contain a probability distribution over labels generated by normalizing the frequency counts of all the training labels reaching the node. Predictions are made in logarithmic time by passing a test point down each of the balanced trees in the ensemble. The test point is sent to an internal node's left (right) child if it lies on the negative (positive) side of the separating hyperplane at that node.

In terms of prediction accuracy, FastXML's proposed node partitioning formulation directly optimizes a rank sensitive loss function which can lead to more accurate predictions over MLRF's Gini index or LPSR's clustering error.

2.2 Build Trees (Algorithm 1)

In the procedure **parallel-for**: T is the number of trees to be grown. Id records the instances (data point) that are at a node. Root node contains all instances.

Each non-leaf node stores information: n.leftnode, n.rightnode, n.w (linear seperator at each node), n.Id (instances at this node). When the number of instances for a node is less than *MaxLeaf*, this node is processed to be a leaf. Leaf node no longer splits. Each tree is grown recursively until no node splits.

Each leaf node stores n.Id and another variable: $\mathbf{P} \leftarrow \text{top}-\text{k}\left(\frac{\sum_{i\in n.Id}\mathbf{y}_i}{|n.Id|}\right)$, scores for top k labels.

Algorithm 1 FastXML($\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, T)

```
procedure PARALLEL-FOR(i = 1, ..., $T$)
    $n^{root}$ ← new node
    $n^{root}.Id \leftarrow \{1, ..., N\}$
    GROW-NODE-RECURSIVE($n^{root}$)
    $\mathcal{T}_i$ ←new tree
    $\mathcal{T}_i.root \leftarrow n^{root}$
end procedure
return $\mathcal{T}_i, ..., \mathcal{T}_T$

procedure GROW-NODE-RECURSIVE(n)
    if $|n.Id| \leq$ MaxLeaf then
            # MaxLeaf: Maximum allowed instances in a leaf node. Larger nodes are attempted to be split, default=10
        $n.\mathbf{P} \leftarrow$ PROCESS-LEAF $\left(\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}, n\right)$
    else
        $\{n.\mathbf{w}, n.left\_child, n.right\_child\} \leftarrow$ SPLIT-NODE$\left(\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}, n\right)$
        GROW-NODE-RECURSIVE(n.left_child)
        GROW-NODE-RECURSIVE(n.right_child)
    end if
end procedure

procedure PROCESS-LEAF($\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, n)
    $\mathbf{P} \leftarrow \text{top}-\text{k}\left(\frac{\sum_{i\in n.Id}\mathbf{y}_i}{|n.Id|}\right)$
     # Return scores for top k labels (recommended labels for each node)
return P
end procedure
```

2.3 Node Partition (Algorithm 2 Split-node)

FastXML employs an alternate strategy and optimizes equation 1 using an iterative alternating minimization algorithm. First, r^+ and r^- are optimized while keeping **w** and δ fixed. This step determines the ranked list of labels that will be predicted by the positive and negative partitions respectively. (**Line 5**)

Second, δ is optimized while keeping **w** and **r** fixed. This step assigns training points in the node to the positive or negative partition. (**Line 11**)

The third step of optimizing **w** while keeping δ and $\mathbf{r}^{\pm}$ fixed is taken only if the first two steps did not lead to a decrease in the objective function. This is done to speed up training since optimizing with respect to δ and $\mathbf{r}^{\pm}$ takes only seconds while optimizing with respect to w can take minutes. This is the primary reason why equation 1 was formulated as a function of **w**, δ and $\mathbf{r}^{\pm}$ rather than just **w**. (**Line 15**)

The algorithm terminates when $\mathbf{r}^{\pm}$, δ and **w** do not change from one iteration to the next.

It is also worth noting that equation 1 allows a label to be assigned to both partitions if some of the points containing the label are assigned to the positive partition and some to the negative. This makes the FastXML trees somewhat robust as the child nodes can potentially recover from mistakes made by the parents.

Algorithm 2 SPLIT-NODE($\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, n)

```
1:  $Id \leftarrow n.Id$
2:  $\delta_i[0] \sim \{-1, 1\}, \forall i \in Id$ # *Random partitioning the instances*
3:  $\mathbf{w}[0] \leftarrow \mathbf{0}, t \leftarrow 0, t_{\mathbf{w}} \leftarrow 0, \mathcal{W}_0 \leftarrow 0$
4:  repeat
5:      $\mathbf{r}^{\pm}[t+1] \leftarrow \mathrm{rank}_L \left(\sum_{i \in Id} \frac{1}{2} (1 \pm \delta_i[t]) I_L(\mathbf{y}_i) \mathbf{y}_i\right)$
6:      for $i \in Id$ do
7:          $v^{\pm} \leftarrow C_\delta(\pm 1) \log\left(1 + e^{\mp \mathbf{w}[t]^\top \mathbf{x}_i}\right) - C_r I_L(\mathbf{y}_i) \sum_{l=1}^{L} \left(\frac{y_{i r_l^{\pm}[t+1]}}{\log(1+l)}\right)$
8:          if $v^+ = v^-$ then
9:              $\delta_i[t+1] = \delta_i[t]$
10:         else
11:             $\delta_i[t+1] = \mathrm{sign}(v^- - v^+)$
12:         end if
13:     end for
14:     if $\delta_i[t+1] = \delta_i$ then
15:         $\mathbf{w}[t+1] \leftarrow \mathrm{argmin}_{\mathbf{w}} \|\mathbf{w}\|_1 + C_\delta(\delta_i[t]) \sum_{i \in Id} \log\left(1 + e^{-\delta_i[t] \mathbf{w}^\top \mathbf{x}_i}\right)$
16:         $\mathcal{W}_{t_{\mathrm{w}}+1} \leftarrow t + 1$
17:         $t_{\mathbf{w}} \leftarrow t_{\mathbf{w}} + 1$
18:     else
19:         $\mathbf{w}[t+1] \leftarrow \mathbf{w}[t]$
20:     end if
21:     $t \leftarrow t + 1$
22: until $\delta[\mathcal{W}_{t_{\mathbf{w}}}] = \delta[\mathcal{W}_{t_{\mathbf{w}}-1}]$
23: $n^+ \leftarrow$ new node , $n^- \leftarrow$ new node
24: $n^+.Id \leftarrow \{i \in Id : \mathbf{w}[t]^\top \mathbf{x}_i > 0\}$
25: $n^-.Id \leftarrow \{i \in Id : \mathbf{w}[t]^\top \mathbf{x}_i \leq 0\}$
26: return $\mathbf{w}[t], n^+, n^-$
```

Line 5: r^+ (r^-) returns the indices of the k largest elements of y ranked in descending order in the right (left) partition

Line 7: C_δ and C_r are user defined weight parameters which determine the relative importance of the three terms, default values are 1. $v^+(v^-)$ calculates the later 3 terms in equation 1 for a single instance i, in the case of partitioning instance i to the right(left).

Line 11: δ_i = 1 if $v^- > v^+$, δ_i = -1 if $v^- < v^+$. Switch the partition of instance i if the objective function referred to equation (1) can be lowered by doing so.

Line 15: When the partition at node n no longer changes, find the **w** that minimized the first 2 terms in equation 1. This still opmitimizes the whole objective since the nDCG term, which is only affected by $\mathbf{r}^{\pm}$ and the choice of δ_i, no longer changes.

2.3.1 Objective for FastXML

$$\begin{aligned} \min \quad & \|\mathbf{w}\|_1 + \sum_i C_\delta\left(\delta_i\right)\log\left(1+e^{-\delta_i \mathbf{w}^\top \mathbf{x}_i}\right) \\ & - C_r \sum_i \frac{1}{2}\left(1+\delta_i\right)\mathcal{L}_{\mathrm{nDCG@}L}\left(\mathbf{r}^+, \mathbf{y}_i\right) \\ & - C_r \sum_i \frac{1}{2}\left(1-\delta_i\right)\mathcal{L}_{\mathrm{nDCG@}L}\left(\mathbf{r}^-, \mathbf{y}_i\right) \end{aligned} \tag{1}$$

The first term in equation 1 is an ℓ_1 regularizer on $\mathbf{w}$ which ensures that a sparse linear separator is used to de ne the partition.

The second term sums the logistic loss of $\delta_i \mathbf{w}^\top \mathbf{x}_i$, where $\delta_i^* = \operatorname{sign}\left(\mathbf{w}^{*\top}\mathbf{x}_i\right)$. The logistic loss term equals to one when $\delta_i \mathbf{w}^\top \mathbf{x}_i$ equals 0. This term decreases when $\delta_i \mathbf{w}^\top \mathbf{x}_i$ increases. This logistic loss term can be very large when $\delta_i \mathbf{w}^\top \mathbf{x}_i$ is much less than zero.

The third and fourth term in equation 1 maximize the nDCG@L of the rankings predicted for the positive and negative partitions, $\mathbf{r}^+$ and $\mathbf{r}^-$. Maximizing nDCG makes it likely that the relevant positive labels for each point are predicted with ranks as high as possible. As a result, points within a partition are likely to have similar labels whereas points across partitions are likely to have different labels.

2.3.2 nDCG score to optimize partition

Cumulative Gain (CG) is the sum of the graded relevance values of all results in a search result list. The value computed with the CG function is unaffected by changes in the ordering of search results.

The CG at a particular rank position k is defined as:

$$\mathcal{L}_{\mathrm{CG@k}}(\mathbf{r}, \mathbf{y}) = \sum_{l=1}^{k} y_{r_l} \tag{2}$$

Ranking of items matters for DCG: the log(1 + l) term ensures that it is beneficial to predict the positive labels with high ranks.

$$\mathcal{L}_{\mathrm{DCG@k}}(\mathbf{r}, \mathbf{y}) = \sum_{l=1}^{k} \frac{y_{r_l}}{\log(1+l)} \tag{3}$$

$$\mathcal{L}_{\mathrm{nDCG@}k}(\mathbf{r}, \mathbf{y}) = I_k(\mathbf{y}) \sum_{l=1}^{k} \frac{y_{r_l}}{\log(1+l)} \tag{4}$$

$$\text{where } I_k(\mathbf{y}) = \frac{1}{\sum_{l=1}^{\min(k, 1^\top \mathbf{y})} \frac{1}{\log(1+l)}} \tag{5}$$

$I_k(\mathbf{y})$ is to take inverse of the ideal-ranked DCG@k (obtained by predicting the ranks of all of y's positive labels to be higher than any of its negative ones). This normalizes DCG@k to lie between 0 and 1 and ensures that nDCG can be used to compare rankings across label vectors with different numbers of positive labels.

2.4 Label Prediction

Given a test point $\mathbf{x} \in \mathcal{R}^D$, predictions are made by these steps: First, pass a test point down each of the balanced trees in the ensemble (decided by the linear seperator $\mathbf{w}$ at each node); Second, compute the average of the label distributions ($\mathbf{P}_t^{\mathrm{leaf}}(\mathbf{x})$) for each tree node found in step 1; Third, return the ranked list of the most k frequently occurring active labels of step 2.

Thus FastXML's top ranked k predictions are given by

$$\mathbf{r}(\mathbf{x}) = \text{rank}_k \left(\frac{1}{T} \sum_{t=1}^{T} \mathbf{P}_t^{\text{leaf}}(\mathbf{x}) \right) \tag{6}$$

where T is the number of trees in the FastXML ensemble and $\mathbf{P}_t^{\text{leaf}}(\mathbf{x})$ (label distribution of the leaf node containing x in tree t) $\propto \sum_{i \in S_t^{\text{leaf}}(\mathbf{x})} \mathbf{y}_i$ and $S_t^{\text{leaf}}(\mathbf{x})$ are the label distribution and set of points respectively of the leaf node containing x in tree t.

Algorithm 3 Predict($\{\mathcal{T}_i, ..., \mathcal{T}_T\}$, $\mathbf{x}$)

```
1:  for $i = 1, \ldots, T$ do
2:      $n \leftarrow T_i$.root
3:      while n is not a leaf do
4:          $\mathbf{w} \leftarrow n.\mathbf{w}$
5:          if $\mathbf{w}^\top \mathbf{x} > 0$ then
6:              $n \leftarrow n.$ right_child
7:          else
8:              $n \leftarrow n.$ left_child
9:          end if
10:     $\mathbf{P}_i^{\text{leaf}}(\mathbf{x}) \leftarrow n.\mathbf{P}$
11:     end while
12: end for
13: $\mathbf{r}(\mathbf{x}) = \text{rank}_k \left( \frac{1}{T} \sum_{i=1}^{T} \mathbf{P}_i^{\text{leaf}}(\mathbf{x}) \right)$
14:  return $\mathbf{r}(\mathbf{x})$
```

3 Change the Objective function

I propose that we can also use cross entropy loss instead of logistic loss when optimizing the rank sensitive loss function for FastXML. Also, the nDCG score loss can be replaced by other ranking losses such as precision score loss. Experiments are done to compare the nDCG loss and Precision loss. Results revealed that nDCG loss achives higher accuracy compared to Precision loss.

3.1 linear seperator with entropy loss

Linear seperator with cross entropy loss:

$$\begin{aligned} \min \quad & \|\mathbf{w}\|_1 - C_\delta \sum_i \frac{1}{2} (1 + \delta_i) \ln(\frac{1}{1 + e^{\mathbf{w}^\top \mathbf{x}_i}}) - C_\delta \sum_i \frac{1}{2} (1 - \delta_i) \ln(1 - \frac{1}{1 + e^{\mathbf{w}^\top \mathbf{x}_i}}) \\ & - C_r \sum_i \frac{1}{2} (1 + \delta_i) \mathcal{L}_{\text{nDCG@}L} \left(\mathbf{r}^+, \mathbf{y}_i\right) - C_r \sum_i \frac{1}{2} (1 - \delta_i) \mathcal{L}_{\text{nDCG@}L} \left(\mathbf{r}^-, \mathbf{y}_i\right) \end{aligned} \tag{7}$$

The second and third terms are the cross entropy terms. The cross entropy loss is closely related to the Kullback-Leibler divergence between the empirical distribution and the predicted distribution. This function is not naturally represented as a product of the true label and the predicted value, but is convex and can be minimized using stochastic gradient descent methods.

Cross Entropy loss may have a better measure for this hyperplane partition problem.

3.2 Precision score for partition

$$\text{P@}k = \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{\mathbf{y}})} \mathbf{y}_l \tag{8}$$

Precision can be obtained from Cumulative Gain (CG) 2 by dividing k. Similar to CG, precision is only sensitive to the relevance of predictions, while DCG is also sensitive to the ranking of predictions. Results from experiments can show that precision approach has a lower accuracy than nDCG score.

4 Experiments

4.1 DataSet

Experiments were carried out on data sets with label set sizes ranging from a hundred to a million to benchmark the performance of FastXML in various regimes. The data sets include two small scale data sets with hundreds of labels, BibTeX and MediaMill, three medium scale data sets with thousands of labels, EURLex, Delicious and RCV1-X, and one large scale data sets with up to a million labels: WikiLSHTC . Table 1 lists the statistics of these data sets.

Table 1: Data set statistics

Data set	Train N	Features D	Labels L	Test M	Avg. labels per pt.	Avg. points per label
BibTex	4880	1836	159	2515	2.40	111.71
Delicious	12920	500	983	3185	19.02	311.61
MediaMill	30993	120	101	12914	4.38	1902.16
EURLex	15539	5000	3993	3809	25.73	5.31
RCV1-X	781265	47236	2456	23149	4.61	1510.13
WikiLSHTC	1892600	1617899	325056	472835	3.26	23.74

4.2 Tables of Result

Table 2: Result on small and medium datasets

(a) Bibtex

Algorithm	P1(%)	P2(%)	P3(%)	P4(%)	P5(%)
Precision	63.58	48.09	39.28	33.04	28.85
nDCG	63.58	59.54	59.65	60.67	62.15

(b) Delicious

Algorithm	P1(%)	P2(%)	P3(%)	P4(%)	P5(%)
Precision	70.02	67.11	64.22	61.88	59.74
nDCG	70.02	67.82	65.67	63.90	62.34

(c) EURLex

Algorithm	P1(%)	P2(%)	P3(%)	P4(%)	P5(%)
Precision	71.30	64.96	59.20	54.30	49.66
nDCG	71.30	66.42	62.33	59.30	57.42

(d) Mediamill

Algorithm	P1(%)	P2(%)	P3(%)	P4(%)	P5(%)
Precision	84.66	78.00	67.74	59.73	53.43
nDCG	84.66	80.84	75.87	73.56	72.77

Pn corresponds to the proportion of recommended items in the top-n set that are relevant.

Table 3: Result on large datasets

(a) RCV1X

Algorithm	P1(%)	P2(%)	P3(%)	P4(%)	P5(%)
Precision	91.15	78.29	73.35	60.73	52.71
nDCG	91.15	90.33	89.63	90.23	90.35

(b) WikiLSHTC

Algorithm	P1(%)	P2(%)	P3(%)	P4(%)	P5(%)
Precision	49.74	39.80	33.01	28.07	24.35
nDCG	49.74	46.44	45.13	44.72	44.64

Speed on small and medium datasets: The training and prediction time is within minutes on a personal computer.

Speed on larger datasets: The training and prediction time for RCV1X is about half an hour; The training and prediction time for WikiLSHTC is about one hour. However, the prediction for WikiLSHTC does not generate accurate results.

5 Conclusions

The key technical contribution in FastXML was a node partitioning formulation which optimized an nDCG based ranking loss over all the labels. Trying to improve the partitioning objective function, I sugguested that cross entropy loss may improve logistic loss, and that precision score can be compared with nDCG score. In conclusion, precision does not achieve better results than nDCG.

References

[1] Yashoteja Prabhu and Manik Varma. Fastxml: a fast, accurate and stable tree-classifier for extreme multi-label learning. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 263–272. ACM, 2014.

[2] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. A theoretical analysis of NDCG type ranking measures. *CoRR*, abs/1304.6480, 2013.

[3] Anna Choromanska and John Langford. Logarithmic time online multiclass prediction. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 55–63, 2015.

[4] Jason Weston, Ameesh Makadia, and Hector Yee. Label partitioning for sublinear ranking. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 181–189, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[5] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: recommending advertiser bid phrases for web pages. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo A. Baeza-Yates, and Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 13–24. International World Wide Web Conferences Steering Committee / ACM, 2013.

[6] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.