# FastXML: A Fast, Accurate and Stable Tree-classifier for eXtreme Multi-label Learning

Yashoteja Prabhu
Indian Institute of Technology - Delhi
yashoteja.prabhu@gmail.com

Manik Varma
Microsoft Research
manik@microsoft.com

## 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 large label set. Extreme multi-label classification is an important research problem since not only does it enable the tackling of applications with many labels but it also allows the reformulation of ranking problems with certain advantages over existing formulations.

Our objective, in this paper, is to develop an extreme multi-label classifier that is faster to train and more accurate at prediction than the state-of-the-art Multi-label Random Forest (MLRF) algorithm [2] and the Label Partitioning for Sub-linear Ranking (LPSR) algorithm [35]. MLRF and LPSR learn a hierarchy to deal with the large number of labels but optimize task independent measures, such as the Gini index or clustering error, in order to learn the hierarchy. Our proposed FastXML algorithm achieves significantly higher accuracies by directly optimizing an nDCG based ranking loss function. We also develop an alternating minimization algorithm for efficiently optimizing the proposed formulation. Experiments reveal that FastXML can be trained on problems with more than a million labels on a standard desktop in eight hours using a single core and in an hour using multiple cores.

## Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

## General Terms

Algorithms, Performance, Machine Learning, Optimization

## Keywords

Multi-label Learning; Ranking; Extreme Classification

## 1. INTRODUCTION

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 large label set. For instance, there are more than a million categories on Wikipedia and one might wish to build a classifier that annotates a novel web page with the subset of most relevant Wikipedia categories. It should be emphasized that multi-label learning is distinct from multi-class classification which aims to predict a single mutually exclusive label.

Extreme classification is an important research problem not just because modern day applications have many categories but also because it allows the reformulation of core learning problems such as recommendation and ranking. For instance, [2] treated search engine queries as labels and built an extreme classifier which, given a novel web page, returned a ranking of queries in decreasing order of relevance. Similarly, [35] treated YouTube videos as distinct labels in an extreme classifier so as to recommend a ranked list of videos to users. Both methods provided a fresh way of thinking about ranking and recommendation problems that led to significant improvements over the state-of-the-art.

Extreme classification is also a challenging research problem as one needs to simultaneously deal with a large number of labels, dimensions and training points. An obvious baseline is provided by the 1-vs-All technique where an independent classifier is learnt per label. Such a baseline has at least two major limitations. First, training millions of high dimensional classifiers might be computationally expensive. Second, the cost of prediction might be high since all the classifiers would need to be evaluated every time a prediction needed to be made. These problems could be ameliorated if a label hierarchy was provided. Unfortunately, such a hierarchy is unavailable in many applications [2, 35].

State-of-the-art methods therefore 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.

Tree based methods can often outperform the 1-vs-All baseline in terms of prediction accuracy at a fraction of the prediction cost. However, such methods can also be expensive to train. In particular, the Label Partitioning by Sublinear Ranking (LPSR) algorithm of [35] can have


*KDD'14,* August 24–27, 2014, New York, NY, USA.

ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623651.

even higher training costs than the 1-vs-All baseline since it needs to learn the hierarchy in addition to the base classifier. Similarly, the Multi-label Random Forest (MLRF) approach of [2] required a cluster of a thousand nodes as random forest training was found to be expensive in high dimensional spaces. Such expensive approaches not only increase the cost of deploying extreme classification models and the cost of daily experimentation but also put such models beyond the reach of most practitioners.

Our objective, in this paper, is to tackle this problem and build a tree based extreme multi-label classifier, referred to as FastXML, that is faster to train as well as more accurate than the state-of-the-art MLRF and LPSR. Our key technical contributions are a novel node partitioning formulation and an algorithm for its efficient optimization. In terms of training time, FastXML can be significantly faster than MLRF since the proposed node partitioning formulation can be optimized more efficiently than the entropy or Gini index based formulations in random forests. At the same time, FastXML can be faster to train than LPSR which has to first learn computationally expensive base classifiers for accurate prediction. In terms of prediction accuracy, almost all extreme classification applications deal with scenarios where the number of relevant positive labels for a data point is orders of magnitude smaller than the number of irrelevant negative ones. Prediction accuracy is therefore not measured using traditional multi-label metrics, such as the Hamming loss, which give equal weightage to all labels whether positive or negative. Instead, most applications prefer employing ranking based measures such as precision at $k$ which focuses exclusively on the positive labels by counting the number of correct predictions in the top $k$ positive predictions. FastXML's proposed node partitioning formulation therefore directly optimizes a rank sensitive loss function which can lead to more accurate predictions over MLRF's Gini index or LPSR's clustering error.

Experiments indicated that FastXML could be significantly more accurate at prediction (by as much as 5% in some cases) on benchmark data sets with thousands of labels where accurate models for MLRF, LPSR and the 1-vs-All baseline could be learnt. Furthermore, FastXML could efficiently scale to problems involving a million labels where accurate training of MLRF, LPSR and 1-vs-ALL models would require a very large cluster. For instance, using a single core on a standard desktop, a single FastXML tree could be trained in under 10 minutes on a data set with about 4 M training points, 160 K features and 1 M labels. The entire ensemble could be trained in 8 hours using a single core and in 1 hour using multiple cores. MLRF and 1-vs-All could not be trained on such extreme multi-label data sets on a standard desktop. LPSR training could be made tractable by replacing the computationally expensive 1-vs-All base classifier by the cheaper Naïve Bayes but then its classification accuracy was found to lag behind by more than 20% on data sets such as Wikipedia.

Our contributions are: (a) We formulate a novel node partitioning objective which directly optimizes an nDCG based ranking loss and which implicitly learns balanced partitions; (b) We propose an efficient optimization algorithm for the novel formulation; and (c) we combine these in a tree algorithm which can train on problems with a million labels on a standard desktop while increasing prediction accuracy. Code for FastXML can be downloaded by clicking here.

## 2. RELATED WORK

There has been much recent progress in extreme multi-label classification [2,4,8,10,12,15,19,20,22,29,34–36,38] and most approaches are either based on trees or on embeddings.

To clarify the discussion, it is assumed that the training set can be represented as $\{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^N\}$ with $D$ dimensional real-valued feature vectors $\mathbf{x}_i \in \mathcal{R}^D$ with sparsity $O(\hat{D})$, and $L$ dimensional binary label vectors $\mathbf{y}_i \in \{0,1\}^L$ with $\mathbf{y}_{il} = 1$ if label $l$ is relevant for point $i$ and 0 otherwise. If one further assumes that the cost of training a linear binary classifier is $O(N\hat{D})$, then the training cost of the linear 1-vs-All baseline is $O(LN\hat{D})$ and the prediction cost is $O(L\hat{D})$. This is infeasible when $L$ and $N$ are in the range $10^5 - 10^6$. One can mitigate the training cost by sub-sampling the negative class for each binary classifier but the prediction cost would still be high.

Embedding methods [4, 8, 10, 12, 15, 19, 20, 22, 29, 34, 36, 38] exploit label correlations and sparsity to compress the number of labels from $L$ to $\hat{L}$. A low-dimensional embedding of the label space is found typically through a linear projection $\hat{\mathbf{y}} = \mathbf{P}\mathbf{y}$ where $\mathbf{P}$ is a $\hat{L} \times L$ projection matrix. The 1-vs-All strategy can now be applied and the cost of training and prediction in the compressed space reduces to $O(\hat{L}N\hat{D})$ and $O(\hat{L}\hat{D})$ respectively. The compressed label predictions also need to be uncompressed at an additional cost of $O(\hat{L}L)$ or higher. Methods mainly differ in the choice of compression and decompression techniques such as compressed sensing [19, 22], Bloom filters [12], SVD [29], landmark labels [4, 8], output codes [38], *etc.* An interesting variant can be obtained by compressing the features $\hat{\mathbf{x}} = \mathbf{R}\mathbf{x}$ to the same $\hat{L}$ dimensional space as the labels [34]. Predictions can then be efficiently made in the embedding space via nearest neighbour techniques and no decompression is required.

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. For instance, none of the embedding methods developed so far have been able to consistently outperform the 1-vs-All baseline when $\hat{L} \approx \log(L)$.

Tree methods that learn a hierarchy enjoy many of the same advantages as the embedding methods. In addition, they can outperform the 1-vs-All baseline even when the prediction costs are $O(\hat{D}\log L)$. Our primary comparison is therefore with tree-based methods.

The Label Partitioning by Sub-linear Ranking (LPSR) method of [35] 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. Generative classifiers such as Naïve Bayes are quick to train but have low accuracy whereas discriminative classifiers such as 1-vs-All with linear SVMs [18], deep nets or Wsabie [34] are expensive to train but have higher 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. Fi-

nally, 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 [2] 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 defined 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 [17] 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.

Finally, care should be taken to note that our objective of learning a multi-label hierarchy is very different from the objective of learning a multi-class hierarchy [5, 11, 13, 16] or exploiting a pre-existing multi-label hierarchy [7, 9, 27].

# 3. FASTXML

Our primary objective, in developing FastXML, is to enable training on a single desktop or a small cluster. At the same time, we aim to achieve greater prediction accuracy by optimizing a more suitable rank sensitive loss function as compared to MLRF's Gini index and LPSR's clustering error. We start this Section by presenting an overview of the FastXML algorithm, then go into the details of optimizing a loss function to learn a node partition and finally analyze FastXML's cost of prediction.

## 3.1 FastXML overview

FastXML learns a hierarchy, not over the label space as is traditionally done in the multi-class setting [5, 13, 16], but rather over the feature space. The intuition is similar to LPSR and MLRF's and comes from the observation that only a small number of labels are present, or active, in each region of feature space. Efficient prediction can therefore be carried out by determining the region in which a novel point lies by traversing the learnt feature space hierarchy and then focusing exclusively on the set of labels active in the region. Like MLRF, and unlike LPSR, FastXML defines the set of labels active in a region to be the union of the labels of all training points present in that region. This speeds up training as FastXML does not need to solve the label assignment integer program in each region. Furthermore, like MLRF, and unlike LPSR, FastXML learns an ensemble of trees and does not need to rely on base classifiers.

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

**parallel-for** $i = 1, .., T$ **do**
  $n^{root} \leftarrow$ new node
  $n^{root}.Id \leftarrow \{1, .., N\}$ # Root contains all instances
  GROW-NODE-RECURSIVE($n^{root}$)
  $\mathcal{T}_i \leftarrow$ new tree
  $\mathcal{T}_i$.root $\leftarrow n^{root}$
**end parallel-for**
**return** $\mathcal{T}_1, .., \mathcal{T}_T$

**procedure** GROW-NODE-RECURSIVE($n$)
  **if** $|n.Id| \leq$ MaxLeaf **then** # Make $n$ a leaf
    $n.\mathbf{P} \leftarrow$PROCESS-LEAF($\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N, n$)
  **else** # Split node and grow child nodes recursively
    $\{n.\mathbf{w}, n.\text{left\_child}, n.\text{right\_child}\}$
      $\leftarrow$ SPLIT-NODE($\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N, n$)
    GROW-NODE-RECURSIVE($n$.left_child)
    GROW-NODE-RECURSIVE($n$.right_child)
  **end if**
**end procedure**

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

Predictions are made by returning the ranked list of most frequently occurring active labels in all the leaf nodes in the ensemble containing the novel point. Algorithms 1 and 3 present pseudo-code for FastXML training and prediction respectively.

## 3.2 Learning to partition a node

Training FastXML consists of recursively partitioning a parent's feature space between its children. Such a node partition should ideally be learnt by optimizing a global measure of performance such as the ranking predictions induced by the leaf nodes, Unfortunately, optimizing global measures can be expensive as all nodes in the tree would need to be learnt jointly [21]. Existing approaches therefore optimize local measures of performance which depend solely on predictions made by the current node being partitioned. This allows the hierarchy to be learnt node by node starting from the root and going down to the leaves and is more efficient than learning all the nodes jointly.

MLRF and LPSR optimize the Gini index and clustering error as their local measure of performance. Unfortunately, neither measure is particularly well suited for ranking or extreme multi-label applications where correctly predicting the few positive relevant labels is much more important than predicting the vast number of irrelevant ones.

FastXML therefore proposes to learn the hierarchy by directly optimizing a ranking loss function. In particular, it optimizes the normalized Discounted Cumulative Gain (nDCG) [33]. This results in learning superior partitions due to two main reasons. First, nDCG is a measure which is sensitive to both ranking and relevance and therefore ensures that the relevant positive labels are predicted with ranks that are as high as possible. This cannot be guaranteed by rank insensitive measures such as the Gini index or the clustering error. Second, by being rank sensitive, nDCG

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

$Id \leftarrow n.Id$
$\delta_i[0] \sim \{-1, 1\}, \forall i \in Id$ # Random coin tosses
$\mathbf{w}[0] \leftarrow \mathbf{0}, t \leftarrow 0, t_{\mathbf{w}} \leftarrow 0, \mathcal{W}_0 \leftarrow 0$ # Various counters
**repeat**
  $\mathbf{r}^{\pm}[t+1] \leftarrow \text{rank}_L\left(\sum_{i \in Id} \frac{1}{2}(1 \pm \delta_i[t]) I_L(\mathbf{y}_i)\mathbf{y}_i\right)$
  **for** $i \in Id$ **do**
    $v^{\pm} \leftarrow C_\delta(\pm 1)\log(1 + e^{\mp \mathbf{w}[t]^\top \mathbf{x}_i}) - C_r I_L(\mathbf{y}_i) \sum_{l=1}^{L}\left(\frac{y_{ir_l^{\pm}[t+1]}}{\log(1+l)}\right)$ # Refer to (5)
    **if** $v^+ = v^-$ **then**
      $\delta_i[t+1] = \delta_i[t]$
    **else**
      $\delta_i[t+1] = \text{sign}(v^- - v^+)$
    **end if**
  **end for**
  **if** $\boldsymbol{\delta}[t+1] = \boldsymbol{\delta}[t]$ **then** # Update $\mathbf{w}$
    $\mathbf{w}[t+1] \leftarrow \underset{\mathbf{w}}{\text{argmin}} \|\mathbf{w}\|_1 + C_\delta(\delta_i[t]) \sum_{i \in Id} \log(1 + e^{-\delta_i[t]\mathbf{w}^\top \mathbf{x}_i})$
    $\mathcal{W}_{t_{\mathbf{w}}+1} \leftarrow t+1$ # Store time step of the update
    $t_{\mathbf{w}} \leftarrow t_{\mathbf{w}} + 1$ # We made one more update to $\mathbf{w}$
  **else**
    $\mathbf{w}[t+1] \leftarrow \mathbf{w}[t]$
  **end if**
  $t \leftarrow t+1$
**until** $\boldsymbol{\delta}[\mathcal{W}_{t_{\mathbf{w}}}] = \boldsymbol{\delta}[\mathcal{W}_{t_{\mathbf{w}}-1}]$ # Convergence
$n^+ \leftarrow$ new node, $n^- \leftarrow$ new node
$n^+.Id \leftarrow \{i \in Id : \mathbf{w}[t]^\top \mathbf{x}_i > 0\}$
$n^-.Id \leftarrow \{i \in Id : \mathbf{w}[t]^\top \mathbf{x}_i \leq 0\}$
**return** $\mathbf{w}[t], n^+, n^-$

can be optimized across all $L$ labels at the current node thereby ensuring that the local optimization is not myopic.

We stick to the notation introduced in the previous Section. it is assumed that the training set can be represented as $\{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^N\}$ with $D$ dimensional real-valued feature vectors $\mathbf{x}_i \in \mathcal{R}^D$ and $L$ dimensional binary label vectors $\mathbf{y}_i \in \{0,1\}^L$ with $\mathbf{y}_{il} = 1$ if label $l$ is relevant for point $i$ and 0 otherwise. Permutation indices $i_1^{\text{desc}}, \ldots, i_L^{\text{desc}}$ that sort a real-valued vector $\mathbf{y} \in \mathcal{R}^L$ in descending order are defined such that if $j > k$ then $y_{i_j^{\text{desc}}} \geq y_{i_k^{\text{desc}}}$. The $\text{rank}_k(\mathbf{y})$ operator, which returns the indices of the $k$ largest elements of $\mathbf{y}$ ranked in descending order (with ties broken randomly), can then be defined as

$$\text{rank}_k(\mathbf{y}) = [i_1^{\text{desc}}, \ldots, i_k^{\text{desc}}]^\top \tag{1}$$

Let $\Pi(1, L)$ denote the set of all permutations of $\{1, 2, \ldots, L\}$. The Discounted Cumulative Gain (DCG) at $k$ of a ranking $\mathbf{r} \in \Pi(1, L)$ given a ground truth label vector $\mathbf{y}$ with binary levels of relevance is

$$\mathcal{L}_{\text{DCG@}k}(\mathbf{r}, \mathbf{y}) = \sum_{l=1}^{k} \frac{y_{r_l}}{\log(1+l)} \tag{2}$$

Note that the $\log(1+l)$ term ensures that it is beneficial to predict the positive labels with high ranks. Thus, unlike precision (which would have been obtained had the $\log(1+l)$ term been absent), DCG is sensitive to both the ranking and the relevance of predictions. The normalized DCG, or

**Algorithm 3** PREDICT($\{\mathcal{T}_1, .. \mathcal{T}_T\}, \mathbf{x}$)

**for** $i = 1, .., T$ **do**
  $n \leftarrow \mathcal{T}_i$.root
  **while** $n$ is not a leaf **do**
    $\mathbf{w} \leftarrow n.\mathbf{w}$
    **if** $\mathbf{w}^\top \mathbf{x} > 0$ **then**
      $n \leftarrow n$.left_child
    **else**
      $n \leftarrow n$.right_child
    **end if**
  **end while**
  $\mathbf{P}_i^{\text{leaf}}(\mathbf{x}) \leftarrow n.\mathbf{P}$
**end for**
$\mathbf{r}(\mathbf{x}) = \text{rank}_k\left(\frac{1}{T}\sum_{i=1}^{T} \mathbf{P}_i^{\text{leaf}}(\mathbf{x})\right)$
**return** $\mathbf{r}(\mathbf{x})$

nDCG, is then defined as

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

$$\text{where } I_k(\mathbf{y}) = \frac{1}{\sum_{l=1}^{\min(k, \mathbf{1}^\top \mathbf{y})} \frac{1}{\log(1+l)}} \tag{4}$$

where $I_k(\mathbf{y})$ is the inverse of the DCG@$k$ of the ideal ranking for $\mathbf{y}$ obtained by predicting the ranks of all of $\mathbf{y}$'s positive labels to be higher than any of its negative ones. This normalizes $\mathcal{L}_{\text{nDCG@}k}$ to lie between 0 and 1 with larger values being better and ensures that nDCG can be used to compare rankings across label vectors with different numbers of positive labels.

FastXML partitions the current node's feature space by learning a linear separator $\mathbf{w}$ such that

$$\begin{aligned} \min \quad & \|\mathbf{w}\|_1 + \sum_i C_\delta(\delta_i)\log(1 + e^{-\delta_i \mathbf{w}^\top \mathbf{x}_i}) \\ & - C_r \sum_i \tfrac{1}{2}(1+\delta_i)\mathcal{L}_{\text{nDCG@}L}(\mathbf{r}^+, \mathbf{y}_i) \\ & - C_r \sum_i \tfrac{1}{2}(1-\delta_i)\mathcal{L}_{\text{nDCG@}L}(\mathbf{r}^-, \mathbf{y}_i) \\ \text{w.r.t.} \quad & \mathbf{w} \in \mathcal{R}^D, \boldsymbol{\delta} \in \{-1, +1\}^L, \mathbf{r}^+, \mathbf{r}^- \in \Pi(1, L) \end{aligned} \tag{5}$$

where $i$ indexes all the training points present at the node being partitioned, $\delta_i \in \{-1, +1\}$ indicates whether point $i$ was assigned to the negative or positive partition and $\mathbf{r}^+$ and $\mathbf{r}^-$ represent the predicted label rankings for the positive and negative partition respectively. $C_\delta$ and $C_r$ are user defined parameters which determine the relative importance of the three terms.

The first term in (5) is an $\ell_1$ regularizer on $\mathbf{w}$ which ensures that a sparse linear separator is used to define the partition. Given a novel point $\mathbf{x}$, the FastXML trees can therefore be traversed efficiently during prediction depending on the sign of $\mathbf{w}^\top \mathbf{x}$ at each node. The second term is the log loss of $\delta_i \mathbf{w}^\top \mathbf{x}_i$. This term couples $\boldsymbol{\delta}$ and $\mathbf{w}$ as the optimal solution of this term alone is $\delta_i^* = \text{sign}(\mathbf{w}^{*\top} \mathbf{x}_i)$. This makes it likely that points assigned to the positive (negative) partition, i.e. points for which $\delta_i = +1$ ($\delta_i = -1$), will have positive (negative) values of $\mathbf{w}^\top \mathbf{x}_i$. $C_\delta$ is relaxed to be a function of $\delta_i$ so as to allow different misclassification penalties for the positive and negative points. The third and fourth term in (5) maximize the nDCG@$L$ of the rankings

predicted for the positive and negative partitions, $\mathbf{r}^+$ and $\mathbf{r}^-$ respectively, given the ground truth label vectors $\mathbf{y}_i$ assigned to these partitions. These terms couple $\mathbf{r}^\pm$ to $\boldsymbol{\delta}$ and thus to $\mathbf{w}$. As discussed, 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.

Furthermore, it is beneficial to maximize nDCG@$L$ at each node even though the ultimate leaf node rankings will be evaluated at $k \ll L$. This leads to non-myopic decisions at the root and internal nodes. For example, optimizing nDCG at $k = 5$ at the root node of the Wikipedia data set would be equivalent to finding a separator such that all the hundreds of thousands of Wikipedia articles assigned to the positive partition could be accurately labeled with just the five most frequently occurring Wikipedia categories in the positive partition and similarly for the negative partition. Clearly this will not lead to good results at the root node and superior partitions can be learnt by considering all the Wikipedia categories rather than just the top five. Of course, as already pointed out, rank insensitive measures such as precision cannot be optimized at $k = L$ as they become more and more uninformative with increasing $k$ and $\mathcal{L}_{\text{precision@}L}(\mathbf{r}^\pm, \mathbf{y}_i) = 0$ for all points irrespective of the partitioning.

It is also worth noting that (5) 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 [2, 13, 16, 35].

Finally, note that $\boldsymbol{\delta}$ and $\mathbf{r}^\pm$ were deliberately chosen to be independent variables for efficient optimization rather than functions dependent on $\mathbf{w}$. In particular, (5) could have been formulated as an optimization problem in just $\mathbf{w}$ by discarding the log loss term and defining $\delta_i(\mathbf{w}) = \text{sign}(\mathbf{w}^\top \mathbf{x}_i)$ and $\mathbf{r}^\pm(\mathbf{w}) = \text{rank}_L\left(\sum_i \frac{1}{2}(1 \pm \delta_i(\mathbf{w}))I_L(\mathbf{y}_i)\mathbf{y}_i\right)$. Such a formulation would also have been natural but intractable at scale. Direct optimization via efficient techniques such as stochastic sub-gradient descent would not be possible due to the sharp discontinuities in $\boldsymbol{\delta}(\mathbf{w})$ and $\mathbf{r}^\pm(\mathbf{w})$. Furthermore, updates to $\mathbf{w}$ would necessitate expensive updates to $\boldsymbol{\delta}$ and $\mathbf{r}^\pm$. We therefore decouple $\boldsymbol{\delta}$ and $\mathbf{r}$ from $\mathbf{w}$ by treating them as variables for efficient optimization but then couple their optimal values through the objective function. We develop the optimization algorithm for (5) in Section 4.

## 3.3 Prediction

The objective function defined in (5) can be optimized efficiently and can lead to accurate predictions. A good objective function should, in addition, also lead to balanced partitions in order to ensure efficient prediction.

Given a novel point $\mathbf{x} \in \mathcal{R}^D$, 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 $\mathbf{P}_t^{\text{leaf}}(\mathbf{x}) \propto \sum_{i \in S_t^{\text{leaf}}(\mathbf{x})}\mathbf{y}_i$ and $S_t^{\text{leaf}}(\mathbf{x})$ are the label distribution and set of points respectively of the leaf node containing $\mathbf{x}$ in tree $t$. The average cost of prediction is upper bounded by $O(TDH + T\hat{L} + \hat{L}\log\hat{L})$ where $H$ is the average length of the path traversed by $\mathbf{x}$ in order to reach the leaf nodes in the $T$ trees and $\hat{L}$ is the number of non-zero elements in the vector $\sum_{t=1}^{T}\mathbf{P}_t^{\text{leaf}}(\mathbf{x})$. The cost is dominated by $O(TDH)$ as $\hat{L} \ll DH$. If the FastXML trees are balanced then $H = \log N \approx \log L$ and the overall cost of prediction becomes $O(TD\log L)$ which is logarithmic in the total number of labels.

One might therefore be tempted to add a balancing term to (5) so as to get $H$ as close to $\log N$ as possible. However, this comes at the cost of reduced prediction accuracy as the objective function trades-off accuracy for balance. As it empirically turns out, our proposed nDCG based objective function learns highly balanced trees and a balancing term does not need to be added to (5). Thus, FastXML's predictions can be made accurately in logarithmic time.

# 4. OPTIMIZING FASTXML

It is well recognized in the learning to rank literature that nDCG is a difficult function to optimize [25,26,31,32] since it is sharply discontinuous with respect to $\mathbf{w}$ and hence standard stochastic sub-gradient descent techniques cannot be applied. FastXML therefore employs an alternate strategy and optimizes (5) using an iterative alternating minimization algorithm. The algorithm is initialized by setting $\mathbf{w} = \mathbf{0}$ and $\delta_i$ to be $-1$ or $+1$ uniformly at random. Each iteration, then, consists of taking three steps. First, $\mathbf{r}^+$ and $\mathbf{r}^-$ are optimized while keeping $\mathbf{w}$ and $\boldsymbol{\delta}$ fixed. This step determines the ranked list of labels that will be predicted by the positive and negative partitions respectively. Second, $\boldsymbol{\delta}$ is optimized while keeping $\mathbf{w}$ and $\mathbf{r}^\pm$ fixed. This step assigns training points in the node to the positive or negative partition. The third step of optimizing $\mathbf{w}$ while keeping $\boldsymbol{\delta}$ 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 $\boldsymbol{\delta}$ and $\mathbf{r}^\pm$ takes only seconds while optimizing with respect to $\mathbf{w}$ can take minutes. This is the primary reason why (5) was formulated as a function of $\mathbf{w}$, $\boldsymbol{\delta}$ and $\mathbf{r}^\pm$ rather than just $\mathbf{w}$. The algorithm terminates when $\mathbf{r}^\pm$, $\boldsymbol{\delta}$ and $\mathbf{w}$ do not change from one iteration to the next. We prove that the objective decreases strictly in each iteration and that the proposed algorithm terminates in a finite number of iterations. In practice, it was observed on all data sets that the algorithm made rapid progress and yielded state-of-the-art results as soon as a single update to $\mathbf{w}$ had been made. We now detail the steps in each iteration.

## 4.1 Optimizing with respect to $\mathbf{r}^\pm$

Given $\mathbf{w}$ and $\boldsymbol{\delta}$, the first step in each iteration is to find the optimal rankings $\mathbf{r}^+$ and $\mathbf{r}^-$ that will be predicted by the positive and negative partition respectively. Fixing $\mathbf{w}$ and $\boldsymbol{\delta}$ simplifies (5) to

$$\max_{\mathbf{r}^\pm \in \Pi(1,L)} C_r \sum_i \tfrac{1}{2}(1+\delta_i)\mathcal{L}_{\text{nDCG@}L}(\mathbf{r}^+, \mathbf{y}_i) + C_r \sum_i \tfrac{1}{2}(1-\delta_i)\mathcal{L}_{\text{nDCG@}L}(\mathbf{r}^-, \mathbf{y}_i) \tag{7}$$

which can be compactly expressed as two independent optimization problems

$$\max_{\mathbf{r}^\pm \in \Pi(1,L)} \sum_{i:\delta_i=\pm 1} I_L(\mathbf{y}_i)\sum_{l=1}^{L}\frac{y_{ir_l^\pm}}{\log(1+l)} \tag{8}$$

$$\equiv \max_{\mathbf{r}^{\pm}\in\Pi(1,L)} \sum_{l=1}^{L} \sum_{i:\delta_i=\pm 1} \frac{I_L(\mathbf{y}_i)y_{il}}{\log(1+r_l^{\pm})} \tag{9}$$

$$\equiv \max_{\mathbf{r}^{\pm}\in\Pi(1,L)} \left( \sum_{i:\delta_i=\pm 1} I_L(\mathbf{y}_i)\mathbf{y}_i \right)^{\top} \mathbf{d}^{\pm} \tag{10}$$

where $\mathbf{d}^{\pm}$ is an $L$-vector such that $d_l^{\pm} = 1/\log(1+r_l^{\pm})$. Since $\mathbf{r}^{+}$ and $\mathbf{r}^{-}$ are permutations of $1, 2, \ldots, L$ it is clear that (10) will be maximized if $r_l^{\pm}$ is chosen as the index of the $l^{\text{th}}$ largest value in the vector $\sum_{i:\delta_i=\pm 1} I_L(\mathbf{y}_i)\mathbf{y}_i$. Thus

$$\mathbf{r}^{\pm^*} = \text{rank}_L \left( \sum_{i:\delta_i=\pm 1} I_L(\mathbf{y}_i)\mathbf{y}_i \right). \tag{11}$$

Note that the optimal values of $\mathbf{r}^{\pm}$ can be computed efficiently in time $O(n \log L + \hat{L} \log \hat{L})$ where $n$ is the number of training points present in the node being partitioned, $\hat{L}$ is the sparsity of the vector $\sum_i I_L(\mathbf{y}_i)\mathbf{y}_i$ and it is assumed that $\mathbf{y}_i$ is $\log L$-sparse.

## 4.2 Optimizing with respect to $\boldsymbol{\delta}$

The next step in an iteration is to optimize (5) with respect to $\boldsymbol{\delta}$ while keeping $\mathbf{w}$ and $\mathbf{r}^{\pm}$ fixed. This reduces to

$$\begin{aligned} \min_{\boldsymbol{\delta}\in\{-1,+1\}^L} \quad & \sum_i C_\delta(\delta_i) \log(1 + e^{-\delta_i \mathbf{w}^\top \mathbf{x}_i}) \\ & - C_r \sum_i \tfrac{1}{2}(1+\delta_i)\mathcal{L}_{\text{nDCG}@L}(\mathbf{r}^{+}, \mathbf{y}_i) \\ & - C_r \sum_i \tfrac{1}{2}(1-\delta_i)\mathcal{L}_{\text{nDCG}@L}(\mathbf{r}^{-}, \mathbf{y}_i) \end{aligned} \tag{12}$$

which decomposes over $i$. Thus, each $\delta_i$ can be optimized independently by seeing whether (12) is optimized by $\delta_i^* = +1$ or $-1$. This yields

$$\delta_i^* = \text{sign}(v_i^- - v_i^+) \quad \text{where} \tag{13}$$

$$v_i^{\pm} = C_\delta(\pm 1) \log(1 + e^{\mp \mathbf{w}^\top \mathbf{x}_i}) - C_r I_L(\mathbf{y}_i) \sum_{l=1}^{L} \frac{y_{ir_l^{\pm}}}{\log(1+l)}$$

The time complexity of obtaining $\boldsymbol{\delta}^*$ is $O(n\hat{D} + n \log L)$ assuming that the feature vectors are $\hat{D}$-sparse and the label vectors are $\log L$-sparse. This reduces to $O(n \log L)$ since $\mathbf{w} = \mathbf{0}$ in the first iteration and $\mathbf{w}^\top \mathbf{x}_i$ can be cached for all points in subsequent updates of $\mathbf{w}$.

## 4.3 Optimizing with respect to w

The final step of optimizing (5) with respect to $\mathbf{w}$ while keeping $\boldsymbol{\delta}$ and $\mathbf{r}^{\pm}$ fixed is carried out only if the first two steps did not make any progress in decreasing the objective function. This can be efficiently determined in time $O(n)$ by checking that $\boldsymbol{\delta}$ has remained unchanged. Optimizing (5) with respect to $\mathbf{w}$ while keeping $\boldsymbol{\delta}$ and $\mathbf{r}^{\pm}$ fixed is equivalent to solving the standard $\ell_1$ regularized logistic regression problem with the labels given by $\boldsymbol{\delta}$

$$\min_{\mathbf{w}\in\mathcal{R}^D} \quad \|\mathbf{w}\|_1 + \sum_i C_\delta(\delta_i) \log(1 + e^{-\delta_i \mathbf{w}^\top \mathbf{x}_i}) \tag{14}$$

This problem has been extensively studied in the literature [3,24,37]. FastXML optimizes (14) using the newGLMNET algorithm [37] as implemented in the Liblinear package [14]. No tuning of the learning rate parameter is required since (14) is optimized in the dual. The algorithm is terminated after 10 passes over the training set in case it hasn't converged already. The overall time complexity of the method is $O(n\hat{D})$ where $n$ is the number of training points in the node being partitioned and $\hat{D}$ is the average number of non-zero entries in a feature vector. This is the most time consuming of the three steps.

## 4.4 Finite termination

The formulation in (5) is non-convex, non-smooth and has a mix of discrete and continuous variables. Furthermore, even the sub-problems obtained by optimizing with respect to only one block of variables might not be convex or smooth. It is well recognized that alternating minimization based techniques can fail to converge for such hard problems in general [6]. However, in our case, it is straightforward to show that FastXML's alternating minimization algorithm for optimizing (5) will not oscillate and will converge in a finite number of iterations.

THEOREM 1. *Suppose Algorithm 2 has not yet terminated and let $\mathcal{W} = \langle \mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_i, \ldots \rangle$ be the sequence of iterations at which $\mathbf{w}$ is updated. Let $\overline{\mathcal{W}_i} = \mathcal{W} - \mathcal{W}_i$ be the sequence obtained by removing $\mathcal{W}_i$ from $\mathcal{W}$. Furthermore, let $\mathcal{W}_i \leq t < \mathcal{W}_{i+1}$. Then (a) $\boldsymbol{\delta}[\mathcal{W}_i] \notin \{\boldsymbol{\delta}[\mathcal{W}_j] | \mathcal{W}_j \in \overline{\mathcal{W}_i}\}$; and (b) $\boldsymbol{\delta}[t] \notin \{\boldsymbol{\delta}[j] | \mathcal{W}_i \leq j \neq t < \mathcal{W}_{i+1}\}$.*

PROOF. Let the objective value in (5), after iteration i, be $O[i]$. The individual sub-problems (10,12,14), that comprise a single iteration of 2, are all minimization problems, which minimize (5) w.r.t a single block of variables, and hence can never increase the objective value. In addition, algorithm 2 also ensures that any change in $\boldsymbol{\delta}$ is accompanied by a non-zero decrease in the objective.

(a) Let $\mathcal{W}_i$ be the iteration at which $\mathbf{w}$ is updated for the $i^{\text{th}}$ time. The algorithm ensures that, when $\boldsymbol{\delta} = \boldsymbol{\delta}[\mathcal{W}_i]$, $\mathbf{w}[\mathcal{W}_i]$ is the minimizer of (14), and $\mathbf{r}^{\pm}[\mathcal{W}_i]$ is the minimizer of (10), which together imply that $O[\mathcal{W}_i]$ is the unique minimum value of (5) when optimized over $\mathbf{w}, \mathbf{r}^{\pm}$, while fixing $\boldsymbol{\delta} = \boldsymbol{\delta}[\mathcal{W}_i]$.
Hence, for a given $\mathcal{W}_i$ and a $\mathcal{W}_j \in \overline{\mathcal{W}_i}$

$$(\boldsymbol{\delta}[\mathcal{W}_i] = \boldsymbol{\delta}[\mathcal{W}_j]) \implies (O[\mathcal{W}_i] = O[\mathcal{W}_j]) \tag{15}$$

Without loss of generality, let $\mathcal{W}_i < \mathcal{W}_j$. Since by assumption, the algorithm has not yet terminated, $\boldsymbol{\delta}[\mathcal{W}_i] \neq \boldsymbol{\delta}[\mathcal{W}_{(i+1)}]$. But, since there has been an update to $\boldsymbol{\delta}$, $O[\mathcal{W}_i] > O[\mathcal{W}_{(i+1)}] \geq O[\mathcal{W}_j]$. This, combined with (15) gives us $\boldsymbol{\delta}[\mathcal{W}_i] \neq \boldsymbol{\delta}[\mathcal{W}_j]$.

(b) Let $u \in \{j : \mathcal{W}_i \leq j \neq t \leq \mathcal{W}_{i+1}\}$. Without loss of generality, assume $t < u$. Since we are between two $\mathbf{w}$ updates, $\mathbf{w}[t] = \mathbf{w}[u]$. If $\boldsymbol{\delta}[t] = \boldsymbol{\delta}[u]$, then using $\mathbf{w}[t] = \mathbf{w}[u]$, we essentially solve the same optimization w.r.t $\mathbf{r}^{\pm}$ and $\boldsymbol{\delta}$ in steps $t+1$ and $u+1$. Hence, $O[t+1] = O[u+1]$. But, absence of $\mathbf{w}$ updates imply that $\boldsymbol{\delta}[t+1] \neq \boldsymbol{\delta}[t+2]$, which further implies $O[t+1] > O[t+2] \geq O[u+1]$, contradicting the earlier equality. Hence, $\boldsymbol{\delta}[t] \neq \boldsymbol{\delta}[u]$. □

Theorem 1 (b) states that $\boldsymbol{\delta}$ cannot repeat between two consecutive updates to $\mathbf{w}$ (in iterations $\mathcal{W}_i$ and $\mathcal{W}_{i+1}$). Since $\boldsymbol{\delta}$ can only take a finite number of values, this implies the number of iterations between $\mathcal{W}_i$ and $\mathcal{W}_{i+1}$ is bounded. Similarly, Theorem 1(a) states that $\boldsymbol{\delta}[\mathcal{W}_i]$ can never repeat for values $\mathcal{W}_j \in \mathcal{W}$. By a similar argument as above and Theorem 1(b), we conclude that $\mathcal{W}_i$ is bounded for all $i$. Thus, the proposed alternating minimization algorithm cannot oscillate and terminates in a finite number of iterations.

Table 1: Data set statistics

| Data set | Train $N$ | Features $D$ | Labels $L$ | Test $M$ | Avg. labels per pt. | Avg. pts 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 |
| RCV1-X | 781265 | 47236 | 2456 | 23149 | 4.61 | 1510.13 |
| WikiLSHTC | 1892600 | 1617899 | 325056 | 472835 | 3.26 | 23.74 |
| Ads-430K | 1118084 | 87890 | 434594 | 502926 | 2.10 | 7.86 |
| Ads-1M | 3917928 | 164592 | 1082898 | 1563137 | 1.96 | 7.07 |

Table 2: Results on small and medium data sets. FastXML was run with default hyper-parameter settings on all data sets. FastXML-T presents results when the parameters were tuned.

(a) BibTeX $N = 4.8K, D = 1.8K, L = 159$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML-T | $\mathbf{64.53 \pm 0.72}$ | $\mathbf{40.17 \pm 0.63}$ | $\mathbf{29.27 \pm 0.53}$ |
| FastXML | $63.26 \pm 0.84$ | $39.19 \pm 0.66$ | $28.72 \pm 0.48$ |
| MLRF | $62.81 \pm 0.84$ | $38.74 \pm 0.69$ | $28.45 \pm 0.43$ |
| LPSR | $62.95 \pm 0.70$ | $39.16 \pm 0.64$ | $28.75 \pm 0.45$ |
| 1-vs-All | $63.39 \pm 0.64$ | $39.55 \pm 0.65$ | $29.13 \pm 0.45$ |

(b) Delicious $N = 13K, D = 500K, L = 983$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{69.65 \pm 0.82}$ | $\mathbf{63.93 \pm 0.50}$ | $\mathbf{59.36 \pm 0.57}$ |
| MLRF | $67.86 \pm 0.70$ | $62.02 \pm 0.55$ | $57.59 \pm 0.43$ |
| LPSR | $65.55 \pm 0.99$ | $59.39 \pm 0.48$ | $53.99 \pm 0.31$ |
| 1-vs-All | $65.42 \pm 1.05$ | $59.34 \pm 0.56$ | $53.72 \pm 0.50$ |

(c) MediaMill $N = 30K, D = 120, L = 101$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{87.35 \pm 0.27}$ | $\mathbf{72.14 \pm 0.20}$ | $\mathbf{58.15 \pm 0.15}$ |
| MLRF | $86.83 \pm 0.18$ | $71.18 \pm 0.19$ | $57.09 \pm 0.16$ |
| LPSR | $82.33 \pm 2.15$ | $66.37 \pm 0.35$ | $50.00 \pm 0.20$ |
| 1-vs-All | $82.31 \pm 2.19$ | $66.17 \pm 0.43$ | $50.32 \pm 0.56$ |

(d) RCV1-X $N = 781K, D = 47K, L = 2.5K$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{91.23 \pm 0.22}$ | $\mathbf{73.51 \pm 0.25}$ | $\mathbf{53.31 \pm 0.65}$ |
| MLRF | $87.66 \pm 0.46$ | $69.89 \pm 0.43$ | $50.36 \pm 0.74$ |
| LPSR | $90.04 \pm 0.19$ | $72.27 \pm 0.20$ | $52.34 \pm 0.61$ |
| 1-vs-All | $90.18 \pm 0.18$ | $72.55 \pm 0.16$ | $52.68 \pm 0.57$ |

Table 3: Results on large data sets comparing the performance of FastXML to LPSR trained with Naïve Bayes as the base classifier.

(a) WikiLSHTC $N = 1.78M, D = 1.62M, L = 325K$

| Algorithm | P1 (%) | P3 (%) | P5 (%) | Train Time (hr) | Test Time (min) |
|---|---|---|---|---|---|
| FastXML | **49.78** | **33.06** | **24.40** | 9.14 | 5.10 |
| LPSR-NB | 27.91 | 16.04 | 11.57 | **1.59** | **3.52** |

(b) Ads-430K $N = 1.12M, D = 88K, L = 0.43M$

| Algorithm | P1 (%) | P3 (%) | P5 (%) | Train Time (hr) | Test Time (min) |
|---|---|---|---|---|---|
| FastXML | **27.24** | **16.28** | **11.91** | 1.81 | **1.68** |
| LPSR-NB | 19.69 | 12.71 | 9.70 | **0.84** | 3.95 |

(c) Ads-1M $N = 3.91M, D = 165K, L = 1.08M$

| Algorithm | P1 (%) | P3 (%) | P5 (%) | Train Time (hr) | Test Time (min) |
|---|---|---|---|---|---|
| FastXML | **23.45** | **14.21** | **10.41** | 8.09 | **6.26** |
| LPSR-NB | 17.08 | 11.38 | 8.83 | **3.78** | 19.32 |

Table 4: FastXML's wall clock training time (in hours) vs the number of cores used on a single machine.

| Cores | WikiLSHTC (hr) | Ads-430K (hr) | Ads-1M (hr) |
|---|---|---|---|
| 1 | 16.80 (1.00×) | 2.46 (1.00×) | 12.34 (1.00×) |
| 2 | 8.62 (1.94×) | 1.24 (1.98×) | 7.09 (1.74×) |
| 4 | 4.53 (3.71×) | 0.62 (3.97×) | 3.93 (3.13×) |
| 8 | 2.21 (7.60×) | 0.33 (7.45×) | 2.09 (5.90×) |
| 16 | 1.27 (13.22×) | 0.19 (12.95×) | 1.02 (12.10×) |

Table 5: The variation in FastXML's performance with the number of training iterations. $\mathcal{W}_i$ denotes the iteration at which $\mathbf{w}$ is updated for the $i^{\text{th}}$ time at the root node on the Ads-430K data set. Precision values and training times are reported for the full ensemble.

| $i$ | $\mathcal{W}_i$ | Objective value | Train time (hr) | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|---|---|---|
| 1 | 15 | 403419.87 | 1.88 | 27.21 | 16.28 | 11.89 |
| 2 | 21 | 237151.20 | 3.79 | 27.01 | 16.39 | 12.09 |
| 3 | 24 | 183542.04 | 5.60 | 26.95 | 16.37 | 12.10 |
| 4 | 26 | 163416.65 | 7.33 | 26.98 | 16.38 | 12.11 |
| 5 | 29 | 153592.21 | 8.99 | 26.93 | 16.38 | 12.11 |

# 5. EXPERIMENTS

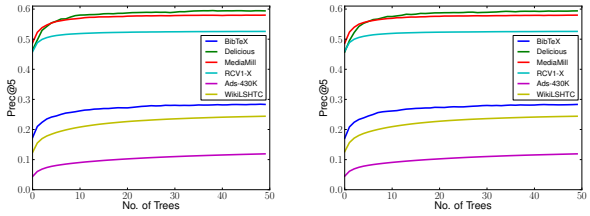
This Section compares the performance of FastXML to MLRF, LPSR and the 1-vs-All baseline on some of the largest multi-label classification data sets.

**Data sets**: 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 [23] and MediaMill [28], two medium scale data sets with thousands of labels, Delicious [30] and RCV1-X, and three large scale data sets with up to a million labels, WikiLSHTC [1], Ads430K and Ads1M. Table 1 lists the statistics of these data sets.

Features and labels are publically available for all the data sets, apart from the two proprietary Ads data sets, and were used in the experiments. WikiLSHTC is a challenge data set for which the test set has not been released and we therefore partitioned the data provided into 75% for training and 25% for testing. The RCV1-X data set has the same features as the original RCV1 data set but its label set has been expanded by forming new labels from pairs of original labels. The two proprietary Ads data sets comprise of bag of words TF-IDF features extracted from expansions of queries from the Bing query logs. Other queries similar to a given query are used as labels for that query.

Table 6: FastXML learns more stable and balanced trees than MLRF and LPSR leading to both faster training as well as faster prediction. Tree balance is measured as $H/\log(N/\text{MaxLeaf})$, where $H$ is the average length of the path traversed by a point in that tree and $\log(N/\text{MaxLeaf})$ is the average length of a path traversed in a perfectly balanced tree with at most MaxLeaf points at each leaf node. Smaller values of tree balance are better with a balance of 1 indicating a perfectly balanced tree.

| Data set | Tree Balance: $\frac{H}{log(N/MaxLeaf)}$ | | | Avg. labels per |
|---|---|---|---|---|
| | FastXML | MLRF | LPSR | leaf for FastXML |
| BibTeX | $\mathbf{1.02 \pm 0.01}$ | $3.45 \pm 0.01$ | $1.56 \pm 0.11$ | 6.15 |
| Delicious | $\mathbf{1.03 \pm 0.01}$ | $4.95 \pm 0.01$ | $3.14 \pm 0.71$ | 69.12 |
| MediaMill | $\mathbf{1.01 \pm 0.01}$ | $1.59 \pm 0.01$ | $1.06 \pm 0.01$ | 10.26 |
| RCV1-X | $\mathbf{1.02 \pm 0.01}$ | $5.62 \pm 0.15$ | $1.32 \pm 0.02$ | 18.73 |
| WikiLSHTC | $\mathbf{1.01 \pm 0.01}$ | - | $3.69 \pm 0.01$ | 13.36 |
| Ads-430K | $\mathbf{1.00 \pm 0.01}$ | - | $94.71 \pm 0.01$ | 10.97 |
| Ads-1M | $\mathbf{1.00 \pm 0.01}$ | - | $105.83 \pm 0.01$ | 11.03 |




(a) Random order (b) Forward sequential

Figure 1: The variation in FastXML's precision at 5 with the number of trees selected according to (1a) random order; and (1b) highest individual prediction accuracy on the training set. The training time can be halved on most data sets with a minimal decrease in prediction accuracy by training only 25 trees in random order.

Results are reported by averaging over ten random train and test splits for the small and medium data sets apart from RCV1-X for which only 3 splits were used. A single split was used for the large data sets.

**Baseline algorithms**: FastXML was compared to state-of-the-art tree based extreme multi-label methods such as MLRF and LPSR (see Section 2 for details) as well as the 1-vs-All baseline as implemented in M3L [18]. The 1-vs-All strategy was also used to learn the base classifier for LPSR on the small and medium data sets as it offered better performance as compared to other base classifiers such as Wsabie [34]. Unfortunately, M3L and Wsabie cannot be trained on the large data sets on a single desktop in a day. Naïve Bayes was therefore used as a base classifier for LPSR on these data sets.

Finally, note that we do not compare explicitly to embedding methods [4, 8, 10, 12, 15, 19, 20, 22, 29, 34, 36, 38] since none of these have been shown to consistently outperform the 1-vs-All base classifier.

**Parameters**: The following hyper-parameters settings were used for FastXML across all data sets: (a) Co-efficient of logistic-loss: $C_\delta = 1.0$; (b) Co-efficient of negative-nDCG loss: $C_r = 1.0$; (c) Number of trees: $T = 50$; (d) Maximum number of instances allowed in a leaf node: MaxLeaf = 10; (e) Number of labels in a leaf node whose probability scores are retained: $k = 20$; (f) Bias multiplier for Liblinear: 1.0; (g) Number of training iterations in Algorithm 2: 1; and (h) Maximum number of (outer,inner) iterations of Liblinear before termination: (10, 10). Using these default settings, it was observed that FastXML could outperform LPSR, MLRF and the 1-vs-All M3L. Tuning the hyper-parameters for each data set would lead to even superior prediction accuracies. The hyper-parameters for MLRF, LPSR and M3L were set

Table 7: Results obtained by replacing the nDCG@$L$ loss function in FastXML with others such as nDCG@5 (FastXML-nDCG5) or precision at 5 (FastXML-P5) and by replacing the Gini index in MLRF with the proposed nDCG@$L$ loss function.

(a) BibTeX $N = 4.8K, D = 1.8K, L = 159$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{63.26 \pm 0.84}$ | $\mathbf{39.19 \pm 0.66}$ | $\mathbf{28.72 \pm 0.48}$ |
| FastXML-P5 | $39.95 \pm 1.09$ | $23.01 \pm 0.46$ | $17.42 \pm 0.36$ |
| FastXML-nDCG5 | $51.75 \pm 1.28$ | $30.87 \pm 0.63$ | $22.70 \pm 0.42$ |
| MLRF-nDCG | $58.41 \pm 1.20$ | $36.45 \pm 0.65$ | $26.95 \pm 0.49$ |

(b) Delicious $N = 13K, D = 500K, L = 983$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{69.65 \pm 0.82}$ | $\mathbf{63.93 \pm 0.50}$ | $\mathbf{59.36 \pm 0.57}$ |
| FastXML-P5 | $60.11 \pm 0.91$ | $53.97 \pm 0.53$ | $49.81 \pm 0.58$ |
| FastXML-nDCG5 | $64.96 \pm 0.83$ | $59.28 \pm 0.69$ | $54.70 \pm 0.56$ |
| MLRF-nDCG | $66.70 \pm 0.75$ | $61.08 \pm 0.44$ | $56.72 \pm 0.44$ |

(c) MediaMill $N = 30K, D = 120, L = 101$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{87.35 \pm 0.27}$ | $\mathbf{72.14 \pm 0.20}$ | $\mathbf{58.15 \pm 0.15}$ |
| FastXML-P5 | $80.73 \pm 0.29$ | $67.48 \pm 0.22$ | $54.11 \pm 0.20$ |
| FastXML-nDCG5 | $86.66 \pm 0.27$ | $70.81 \pm 0.21$ | $56.51 \pm 0.20$ |
| MLRF-nDCG | $86.67 \pm 0.26$ | $71.13 \pm 0.22$ | $57.24 \pm 0.21$ |

(d) RCV1-X $N = 781K, D = 47K, L = 2.5K$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | $\mathbf{91.23 \pm 0.22}$ | $\mathbf{73.51 \pm 0.25}$ | $\mathbf{53.31 \pm 0.65}$ |
| FastXML-P5 | $66.62 \pm 0.23$ | $56.41 \pm 0.40$ | $40.83 \pm 0.14$ |
| FastXML-nDCG5 | $75.60 \pm 0.39$ | $60.85 \pm 0.43$ | $43.98 \pm 0.16$ |
| MLRF-nDCG | $87.19 \pm 0.41$ | $69.69 \pm 0.46$ | $50.25 \pm 0.56$ |

(e) WikiLSHTC $N = 1.78M, D = 1.62M, L = 325K$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | **49.78** | **33.06** | **24.40** |
| FastXML-P5 | 18.74 | 12.33 | 8.71 |
| FastXML-nDCG5 | 20.50 | 12.51 | 8.80 |

(f) Ads-430K $N = 1.12M, D = 88K, L = 0.43M$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | **27.24** | **16.28** | **11.91** |
| FastXML-P5 | 25.06 | 14.36 | 10.27 |
| FastXML-nDCG5 | 24.93 | 14.34 | 10.29 |

(g) Ads-1M $N = 3.91M, D = 165K, L = 1.08M$

| Algorithm | P1 (%) | P3 (%) | P5 (%) |
|---|---|---|---|
| FastXML | **23.45** | **14.21** | **10.41** |
| FastXML-P5 | 21.09 | 12.32 | 8.87 |
| FastXML-nDCG5 | 21.25 | 12.47 | 9.01 |

using fine grained validation on each data set so as to achieve the highest possible prediction accuracy for each method.

**Evaluation Metrics**: Extreme multi-label classification data sets exhibit positive label sparsity in that each data point has only a few positive labels associated with it. It therefore becomes important to focus on the accurate prediction of the few positive labels per data point than on the vast number of negative ones. As such, most papers [2, 19, 22, 34–36] evaluate prediction accuracy using precision at $k$ which counts the number of correct predictions in the top $k$ positive predictions. More formally, the precision at $k$ for a prediction $\hat{\mathbf{y}} \in \mathcal{R}^L$ given the ground truth label vector $\mathbf{y} \in \{0,1\}^L$ is defined as

$$\mathrm{P}k(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{k} \sum_{i \in \mathrm{rank}_k(\hat{\mathbf{y}})} \mathbf{y}_i. \tag{16}$$

All experiments were carried out on a single core of an Intel Core 2 Duo machine running at 3.3 GHz with 8 Gb of RAM. Training times were measured using the `clock` function available in C++. Parallelization experiments in Table 4 were carried out on multiple cores of Intel Xeon processors running at 2.1 GHz.

**Results on small and medium data sets**: Table 2 benchmarks the performance of FastXML on the small and medium data sets where accurate MLRF, LPSR and 1-vs-All models could be learnt. The focus is on prediction accuracy as neither training nor prediction present any challenge on these data sets – even the expensive MLRF, LPSR and 1-vs-All can be trained in seconds on BibTeX and Delicious, in minutes on MediaMill and in two hours on RCV1-X.

As compared to LPSR and 1-vs-All, FastXML could be up to 5% more accurate in terms of precision at 1 and up to 8% in terms of precision at 5. Large improvements were obtained on both Delicious and MediaMill. The smallest improvements were obtained on BibTeX which had less than five thousand training points. FastXML with default parameter settings gave more or less the same P1 as LPSR and 1-vs-All on BibTeX. A marginal improvement of 1% over the other methods could be obtained by tuning FastXML's hyper-parameters (referred to as FastXML-T in Table 2). FastXML's gains over MLRF were smaller as compared to the gains over LPSR and 1-vs-All but could still go up to 3% in terms of both precision at 1 and at 5 (on RCV1-X). All in all, these results indicate that FastXML could be significantly more accurate at prediction than highly tuned MLRF, LPSR and 1-vs-All classifiers.

**Results on large data sets**: Extreme classification is most concerned with large scale data sets having hundreds of thousands or even millions of labels. Training MLRF and 1-vs-All on such data sets was found to be infeasible without using a large cluster. LPSR training could be made tractable on a single core by replacing the 1-vs-All base classifier with Naïve Bayes. Table 3 compares the performance of FastXML to LPSR-NB. FastXML's improvements over LPSR-NB in terms of P1 ranged from 5% on Ads-1M to almost 22% on WikiLSHTC and in terms of P5 ranged from approximately 1.5% on Ads-1M to almost 13% on WikiLSHTC. FastXML could train in 1.81 hours on Ads-430K using a single core and in 8 to 9 hours on Ads-1M and WikiLSHTC with individual trees being grown in about 10 minutes. This opens up the possibility of practitioners training accurate extreme classification models on commodity hardware. Finally, FastXML could be almost 1.5 to 3 times faster at prediction than LPSR-NB which could be a critical factor in certain applications.

**Validating FastXML's hyper-parameter settings and design choices**: Table 4 lists the reduction in wall clock training time obtained by growing FastXML's trees in parallel across multiple cores on a single machine. Training could be speeded up 12 to 13 times by utilizing 16 cores demonstrating that FastXML is trivially parallelizable. The entire ensemble could be trained in approximately an hour on Ads-1M and WikiLSHTC and in 12 minutes on Ads-430K.

All the FastXML results presented so far were obtained by terminating the proposed optimization algorithm after a single update to $\mathbf{w}$. Table 5 shows the effects of allowing multiple updates to $\mathbf{w}$ while training on the Ads-430K data set. High precisions were reached after the first update to $\mathbf{w}$ which occurred after 15 updates to $\boldsymbol{\delta}$ and $\mathbf{r}$. Subsequent updates to $\mathbf{w}$, $\boldsymbol{\delta}$ and $\mathbf{r}$ yielded a significant drop in the value of the objective function but little change in prediction accuracy. As such, it would appear that training time could be significantly reduced without much loss in precision by early termination.

Table 7 reports the effects of replacing the proposed nDCG@$L$ loss function in FastXML with others such as precision at 5 (FastXML-P5) and nDCG@5 (FastXML-nDCG5). As is evident, both these loss functions were inferior to nDCG@$L$ even when performance was measured using precision at 5. This demonstrates that rank sensitive loss functions such as nDCG are better suited to extreme multi-label classification as compared to rank insensitive ones such as precision. Furthermore, nDCG should be computed non-myopically over all labels rather than just the top few.

Finally, the key difference in FastXML's formulation as compared to MLRF's was the use of the nDCG based loss function and the use of a linear separator to partition each node. Table 7 demonstrates that both these ingredients were necessary as simply replacing the Gini index or entropy in MLRF with the nDCG based loss function (MLRF-nDCG) yielded significantly poorer results as compared to FastXML.

# 6. CONCLUSIONS

This paper developed the FastXML algorithm for multi-label learning with a large number of labels. FastXML learnt an ensemble of trees with prediction costs that were logarithmic in the number of labels. The key technical contribution in FastXML was a novel node partitioning formulation which optimized an nDCG based ranking loss over all the labels. Such a loss was found to be more suitable for extreme multi-label learning than the Gini index optimized by MLRF or the clustering error optimized by LPSR. nDCG is known to be a hard loss to optimize using gradient descent based techniques. FastXML therefore developed an efficient alternating minimization algorithm for its optimization. It was proved that the proposed alternating minimization algorithm would not oscillate and would converge in a finite number of iterations. Experiments revealed that FastXML could be significantly more accurate than MLRF and LPSR while efficiently scaling to problems with more than a million labels. The FastXML code is publically available and should enable practitioners to train accurate extreme multi-label models without needing large clusters.

## Acknowledgments

We are very grateful to Purushottam Kar and Prateek Jain for helpful discussions. Yashoteja Prabhu is supported by a TCS PhD Fellowship at IIT Delhi.

## 7. REFERENCES

[1] Wikipedia dataset for the 4th large scale hierarchical text classification challenge. http://lshtc.iit.demokritos.gr/.

[2] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*, pages 13–24, 2013.

[3] G. Andrew and J. Gao. Scalable training of $L\_1$-regularized log-linear models. In *ICML*, pages 33–40, 2007.

[4] K. Balasubramanian and G. Lebanon. The landmark selection method for multiple output prediction. In *ICML*, 2012.

[5] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *NIPS*, 2010.

[6] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[7] W. Bi and J. T.-Y. Kwok. Multilabel classification on tree- and dag-structured hierarchies. In *ICML*, 2011.

[8] W. Bi and J. T.-Y. Kwok. Efficient multi-label classification with many labels. In *ICML*, pages 405–413, 2013.

[9] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Incremental algorithms for hierarchical classification. *JMLR*, 7, 2006.

[10] Y.-N. Chen and H.-T. Lin. Feature-aware label space dimension reduction for multi-label classification. In *NIPS*, pages 1538–1546, 2012.

[11] A. Choromanska and J. Langford. Logarithmic time online multiclass prediction. http://arxiv.org/abs/1406.1822, 2014.

[12] M. Cissé, N. Usunier, T. Artières, and P. Gallinari. Robust bloom filters for large multilabel classification tasks. In *NIPS*, pages 1851–1859, 2013.

[13] J. Deng, S. Satheesh, A. C. Berg, and F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, 2011.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[15] C.-S. Feng and H.-T. Lin. Multi-label classification with error-correcting codes. *JMLR*, pages 289–295, 2011.

[16] T. Gao and D. Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *ICCV*, pages 2072–2079, 2011.

[17] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *ML*, pages 3–42, 2006.

[18] B. Hariharan, S. V. N. Vishwanathan, and M. Varma. Efficient max-margin multi-label classification with applications to zero-shot learning. *ML*, 2012.

[19] D. Hsu, S. Kakade, J. Langford, and T. Zhang. Multi-label prediction via compressed sensing. In *NIPS*, 2009.

[20] S. Ji, L. Tang, S. Yu, and J. Ye. Extracting shared subspace for multi-label classification. In *KDD*, pages 381–389, 2008.

[21] C. Jose, P. Goyal, P. Aggrwal, and M. Varma. Local deep kernel learning for efficient non-linear svm prediction. In *ICML*, June 2013.

[22] A. Kapoor, R. Viswanathan, and P. Jain. Multilabel classification using bayesian compressed sensing. In *NIPS*, 2012.

[23] I. Katakis, G. Tsoumakas, and I. Vlahavas. Multilabel text classification for automated tag suggestion. In *ECML/PKDD Discovery Challenge*, 2008.

[24] K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *JMLR*, 8:1519–1555, 2007.

[25] A. Kustarev, Y. Ustinovsky, Y. Logachev, E. Grechnikov, I. Segalovich, and P. Serdyukov. Smoothing ndcg metrics using tied scores. In *CIKM*, pages 2053–2056, 2011.

[26] P. D. Ravikumar, A. Tewari, and E. Yang. On ndcg consistency of listwise ranking methods. In *AISTATS*, pages 618–626, 2011.

[27] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *JMLR*, 7, 2006.

[28] C. Snoek, M. Worring, J. van Gemert, J.-M. Geusebroek, and A. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *ACM Multimedia*, pages 421–430, 2006.

[29] F. Tai and H.-T. Lin. Multi-label classification with principal label space transformation. In *Workshop proceedings of learning from multi-label data*, 2010.

[30] G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 2008.

[31] H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to rank by optimizing ndcg measure. In *SIGIR*, pages 41–48, 2000.

[32] M. N. Volkovs and R. S. Zemel. Boltzrank: Learning to maximize expected ranking gain. In *ICML*, pages 1089–1096, 2009.

[33] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of nDCG type ranking measures. In *COLT*, pages 25–54, 2013.

[34] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

[35] J. Weston, A. Makadia, and H. Yee. Label partitioning for sublinear ranking. In *ICML*, volume 28, pages 181–189, 2013.

[36] H.-F. Yu, P. Jain, P. Kar, and I. S. Dhillon. Large-scale multi-label learning with missing labels. *ICML*, 2014.

[37] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved glmnet for l1-regularized logistic regression. *JMLR*, 13:1999–2030, 2012.

[38] Y. Zhang and J. G. Schneider. Multi-label output codes using canonical correlation analysis. In *AISTATS*, pages 873–882, 2011.