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

Prabhu & Varma
KDD 2014

February 5, 2019
Outline

1 Introduction

2 Training

3 Testing
Objective in eXtreme Multi-Label (XML) 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.
FastXML learns a hierarchy, not over the label space as is traditionally done in the multi-class setting, but rather over the feature space.

The intuition is that only a small number of labels are present, or active, in each region of feature space.

Efficient prediction can be made by determining the region in which a test point lies by traversing the learnt feature space hierarchy and then focusing exclusively on the set of labels active in the region.
FastXML Overview

- FastXML learns an ensemble of trees
- 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
- Predictions are made by returning the ranked list of most frequently occurring active labels in all the leaf nodes in the ensemble containing the test point
Outline

1. Introduction

2. Training

3. Testing
Learning to Partition a Node

- Training FastXML consists of recursively partitioning a parent's feature space between its children.
- Such node partitions should be learnt by optimizing a global measure of performance such as the ranking predictions induced by the leaf nodes.
Learning to Partition a Node

- Data \( \{(x_i, y_i)_{i=1}^N\} \) with \( D \) dimensional feature vectors \( x_i \) and \( L \) dimensional binary label vectors \( y_i \in 0, 1^L \)
- Discounted Cumulative Gain (DCG) at \( k \) of a ranked vector \( r \) given ground truth label vector \( y \) with binary levels of relevance:
  \[
  \mathcal{L}_{DCG@k}(r, y) = \sum_{l=1}^{k} \frac{y_{rl}}{\log(1 + l)}
  \]  
  \( (1) \)
- Unlike precision, DCG is sensitive to both the ranking and relevance of predictions.
FastXML partitions the current node’s feature space by learning a linear separator $w$:

$$
\min \|w\|_1 + \sum_i C_\delta(\delta_i) \log(1 + e^{-\delta_i w^T x_i})
$$

$$
- C_r \sum_i \frac{1}{2} (1 + \delta_i) \mathcal{L}_{nDCG_{@L}}(r^+, y_i)
$$

$$
- C_r \sum_i \frac{1}{2} (1 - \delta_i) \mathcal{L}_{nDCG_{@L}}(r^-, y_i)
$$

w.r.t. $w \in \mathcal{R}^D$, $\delta \in \{-1, +1\}^L$, $r^+, r^- \in \Pi(1, L)$

$i$ indexes 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 $r^+$ and $r^-$ represent the predicted label rankings for the positive and negative partition respectively.
DCG@$L$ is performed on each node, even though the ultimate leaf node rankings will be evaluated at $k << L$.

The separator function allows a label to be assigned to both partitions if 2 separate points containing the same label are split into the different feature space. This makes FastXML robust.
Learning to Partition a Node

Algorithm 1: FastXML($\{x_i, y_i\}_{i=1}^N, T$)

```
parallel-for $i = 1, \ldots, T$ do
    $n^{root} \leftarrow$ new node
    $n^{root}.Id \leftarrow \{1, \ldots, N\}$  # Root contains all instances
    GROW-NODE-RECURSIVE($n^{root}$)
    $T_i \leftarrow$ new tree
    $T_i.root \leftarrow n^{root}$
end parallel-for
return $T_1, \ldots, T_T$
```

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

procedure PROCESS-LEAF($\{x_i, y_i\}_{i=1}^N, n$)
```
$P \leftarrow \text{top-k} \left( \frac{\sum_{i \in n.Id} y_i}{|n.Id|} \right)$
return $P$  # Return scores for top $k$ labels
end procedure
Start by setting \( w = 0 \) and \( \delta_i \) to be 1 or \(+1\) uniformly at random. Each iteration, then, consists of taking three steps.

1. \( r^+ \) and \( r \) are optimized while keeping \( w \) and \( \delta \) fixed. This determines the ranked list of labels that will be predicted by the positive and negative partitions respectively.

2. \( \delta \) is optimized while keeping \( w \) and \( r \pm \) fixed. This step assigns training points in the node to the positive or negative partition.

3. Optimizing \( w \) while keeping \( \delta \) and \( r \pm \) fixed is taken only if the first two steps did not lead to a decrease in the objective function.
Algorithm 3 \textsc{Predict}(\{\mathcal{T}_1, \ldots, \mathcal{T}_T\}, x)

\begin{algorithmic}
\FOR{$i = 1, \ldots, T$}
\STATE $n \leftarrow \mathcal{T}_i.\text{root}$
\WHILE{$n$ is not a leaf}
\STATE $w \leftarrow n.w$
\IF{$w^\top x > 0$}
\STATE $n \leftarrow n.\text{left}_\text{child}$
\ELSE
\STATE $n \leftarrow n.\text{right}_\text{child}$
\ENDIF
\ENDWHILE
\STATE $P_{i}^{\text{leaf}}(x) \leftarrow n.P$
\ENDFOR
\STATE $r(x) = \text{rank}_k \left(\frac{1}{T} \sum_{i=1}^{T} P_{i}^{\text{leaf}}(x)\right)$
\RETURN $r(x)$
\end{algorithmic}
(d) RCV1-X $N = 781K$, $D = 47K$, $L = 2.5K$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>P1 (%)</th>
<th>P3 (%)</th>
<th>P5 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastXML</td>
<td>91.23 ± 0.22</td>
<td>73.51 ± 0.25</td>
<td>53.31 ± 0.65</td>
</tr>
<tr>
<td>MLRF</td>
<td>87.66 ± 0.46</td>
<td>69.89 ± 0.43</td>
<td>50.36 ± 0.74</td>
</tr>
<tr>
<td>LPSR</td>
<td>90.04 ± 0.19</td>
<td>72.27 ± 0.20</td>
<td>52.34 ± 0.61</td>
</tr>
<tr>
<td>1-vs-All</td>
<td>90.18 ± 0.18</td>
<td>72.55 ± 0.16</td>
<td>52.68 ± 0.57</td>
</tr>
</tbody>
</table>