LdSM: Logarithm-depth Streaming Multi-label Decision Trees

Maryam Majzoubi
New York University

Anna Choromanska
New York University

Abstract

We consider multi-label classification where the goal is to annotate each data point with the most relevant *subset* of labels from an extremely large label set. Efficient annotation can be achieved with balanced tree predictors, i.e. trees with logarithmic-depth in the label complexity, whose leaves correspond to labels. Designing prediction mechanism with such trees for real data applications is non-trivial as it needs to accommodate sending examples to multiple leaves while at the same time sustain high prediction accuracy. In this paper we develop the LdSM algorithm for the construction and training of multi-label decision trees, where in every node of the tree we optimize a novel objective function that favors balanced splits, maintains high class purity of children nodes, and allows sending examples to multiple directions but with a penalty that prevents tree over-growth. Each node of the tree is trained once the previous node is completed leading to a streaming approach for training. We analyze the proposed objective theoretically and show that minimizing it leads to pure and balanced data splits. Furthermore, we show a boosting theorem that captures its connection to the multi-label classification error. Experimental results on benchmark data sets demonstrate that our approach achieves high prediction accuracy and low prediction time and position LdSM as a competitive tool among existing state-of-the-art approaches.

1 INTRODUCTION

Plethora of modern machine learning approaches are concerned with performing multi-label predictions, as is the case in recommendation or ranking systems. In multi-label setting we receive examples $x \in \mathcal{X} \subseteq \mathbb{R}^d$, with labels $y \subseteq \mathcal{Y} \equiv \{1, 2, \ldots, K\}$, where each data point x is assigned a *subset* of labels y from an extremely large label set $\mathcal{Y}$. This provides a generalization of the multi-class problem [Bengio et al. (2010); Deng et al. (2011); Tianshi and Koller (2011); Choromanska and Langford (2015)], where each data point instead corresponds to a single mutually exclusive label.[1] Employing the label hierarchy, commonly represented as a tree with leaves corresponding to labels, potentially allows for faster prediction when the hierarchy is balanced and thus the tree depth is of size $\mathcal{O}(\log_M K)$ for M-ary tree, and enables overcoming the intractability problem of common baselines, such as one-against-all (OAA) [Rifkin and Klautau (2004)] that requires evaluating K classifiers per example. Tree-based predictors are therefore commonly used, but since the label hierarchy is unavailable most of the times, it has to be learned from the data.

The performance of the multi-label tree-based system heavily hinges on the structure of the tree [Mnih and Hinton (2009); Jain et al. (2016)]. Some approaches [Jasinska et al. (2016); Wydmuch et al. (2018)] assume arbitrary label hierarchy that is not learned. For example, PLT [Jasinska et al. (2016)] considers a sparse probability estimates for F-measure maximization conditioned on the label tree. Majority of techniques however carefully design a splitting criterion that is recursively applied in every node of the tree to partition the data. These criteria differ between commonly-used tree-based multi-label classification approaches. Multi-label Random Forest (MLRF) [Agrawal et al. (2013)] uses *information theoretic losses*, specifically the class entropy or the Gini index, to obtain label hierarchy. Sparse gradient boosted decision trees (GBDT-S) [Si et al. (2017)] build a regression tree that fits the residuals from the previous trees and uses the multi-label hinge or squared loss.

FastXML [Prabhu and Varma (2014)], PFastreXML [Jain et al. (2016)], and SwiftXML [Prabhu et al. (2018a)] (the last one focuses on the prediction task with partially revealed labels) constitute a family of methods that rely on *ranking losses*. FastXML learns a hierarchy over the feature

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

[1] It is non-trivial to extend multi-class trees to the multi-label setting [Prabhu et al. (2018b)] as their training and prediction mechanism is not suitable for the setting when an example is equipped with more than one label.

space, rather than the label space, relying on the intuition that in each region of the feature space only a small subset of labels is active. The node objective function there promotes generalizability via standard regression loss and rank-prioritization via normalized Discounted Cumulative Gain (nDCG) ensuring that relevant positive labels for each point are predicted with high ranks. PFastreXML improves upon FastXML by replacing the nDCG loss with its propensity scored variant (the same is used in SwiftXML) which is unbiased to the missing classes and assigns higher rewards for accurate tail label predictions. None of the above techniques use balancing term in their objective.

There also exist methods that construct tree classifiers by optimizing *clustering loss* in nodes. Hierarchical k-means underlies CRAFTML [W. Siblini and Meyer (2018)] and older approaches to multi-label classification such as LPSR [Weston et al. (2013)], and HOMER [Tsoumakas et al. (2008)].

The approach we propose in this paper belongs to the family of purely tree-based methods. It partitions tree nodes based on joint optimization in the feature and label space. The node split is based on a new objective function that explores the correlation between both spaces by conditioning the learning of feature space partitioning with data label information. The objective applies to trees of arbitrary width. It explicitly enforces class purity of children nodes (i.e. points within a partition are likely to have similar labels whereas points across partitions are likely to have different labels). Moreover, it relies on having multiple (i.e. two for binary tree) regressors at each node and thus allows sending examples to multiple children nodes. Multi-way assignment of examples is however penalized to better control tree accuracy. Finally, the objective encourages balanced partitions to ensure efficient prediction. The objective function comes with theoretical guarantees. We show that optimizing the objective improves the purity and balancedness of the data splits in isolation, i.e. when respectively the balancedness and purity is fixed. We next analyze the connection of the proposed objective with the multi-label classification error. We prove that when the objective is perfectly optimized in every tree node it leads to zero-error. We generalize this observation to a setting when the objective is gradually optimized in every tree node, but may never reach the actual optimum. We first show that minimizing the objective is causing the monotonic decrease of the error with every split. Next we prove a much stronger statement given in the form of boosting theorem that relies on weakly optimizing the objective function at each node of the tree and show that our tree algorithm boosts the weak learners at the nodes to achieve any desirable multi-label classification error in a finite number of splits. The resulting tree construction-and-training algorithm, that we call LdSM, results in **L**ogarithmic-**d**epth trees that are trained in a **s**treaming fashion, i.e. node-by-node[2], and achieve competitive performance to other state-of-the-art tree-based approaches, being accurate and efficient at prediction, on large **m**ulti-label classification problems. In summary, our proposed objective function, the resulting tree construction algorithm, and theoretical analysis are all new and constitute the contributions of our paper.

We next discuss other approaches for multi-label classification. They constitute a different family of methods than purely tree-based techniques that our method belongs to and thus are not directly relevant to our work. Those techniques include extensions of OAA [Babbar and Schölkopf (2017); Yen et al. (2016, 2017); H. Fang and Friedlander (2019); Niculescu-Mizil and Abbasnejad (2017); Babbar and Schölkopf (2019); Prabhu et al. (2018b); Khandagale et al. (2019)], deep learning methods [You et al. (2019); Liu et al. (2017); Zhang et al. (2018); Jernite et al. (2017)] and approaches for learning embeddings [Balasubramanian and Lebanon (2012); Bi and Kwok (2013); Cisse et al. (2013); Hsu et al. (2009); Tai and Lin (2012); Zhang and Schneider (2011); Chen and Lin (2012); Yu et al. (2014); Ferng and Lin (2011); Ji et al. (2008); Weston et al. (2011); Lin et al. (2014)].

The paper is organized as follows: Section 2 presents the objective function, Section 3 provides theoretical results, Section 4 shows the algorithm for tree construction and training and explains how to perform testing using the tree, Section 5 reports empirical results on benchmark multi-label data sets, and finally Section 6 concludes the paper. Supplementary material contains proofs of theorems from Section 3, additional pseudo-codes of algorithms from section 4, and additional experimental results.

2 OBJECTIVE FUNCTION

We next explain the design of the objective function for the tree of arbitrary width M, i.e. tree where each node has M children, and show a special case of a binary tree. Below we consider an arbitrary non-leaf node of the tree and thus omit node index in the notation.

In our setting, each node of the tree contains M binary classifiers h_j, where $j = 1, 2, \ldots, M$. $h_j \in \mathcal{H}$, where $\mathcal{H}$ is the hypothesis class with linear regressors. Consider an arbitrary non-leaf node and let π_i denote the normalized fraction of examples containing label i in their label set reaching that node, where the multiplicative normalizing factor is an inverse of the average number of labels per example containing label i in their label set (note that $\sum_{i=1}^{K} \pi_i = 1$). The node regressors are trained in such a way that $h_j(x) \geq 0.5$ means that the example x is sent to the j^{th} subtree of a node (thus sending example to more than

[2]When training each node we stream multiple times through the data before moving to the next node. After we move, we never go back to the previously trained ones. Thus we assume the data set is finite (but can be very large). This differs from the online setting. For distinction between streaming and online settings see Dasgupta (2008).

one child is possible). To prevent examples from stucking inside the node, in case when $h_j(x) < 0.5 \; \forall_{j=1,2,...,M}$ the example is sent to the child node corresponding to the highest margin, i.e. $(\arg\max_{j=1,2,...,M} h_j(x))^{\text{th}}$ child node. Let $P_j = P(h_j(x) > 0.5)$ be the probability that the example x reaches child $j \in \{1,\ 2, ...,\ M\}$ and let $P_j^i = P(h_j(x) > 0.5|i)$ denote the conditional probability of these event when the example belongs to class i. Note that i) $\sum_{j=1}^M P_j \geq 1$, ii) for any $i = 1, 2, \ldots, K$, $\sum_{j=1}^M P_j^i \geq 1$, and iii) $P_j = \sum_{i=1}^k \pi_i P_j^i$. The node splitting criterion is defined as follows

$$J \coloneqq \underbrace{\sum_{j=1}^{M} \sum_{l=j+1}^{M} |P_j - P_l|}_{\text{balancing term}} \underbrace{\underbrace{-\lambda_1 \sum_{i=1}^{K} \sum_{j=1}^{M} \sum_{l=j+1}^{M} \pi_i \left|P_j^i - P_l^i\right|}_{\text{class integrity term}} + \underbrace{\lambda_2 \left| \left(\sum_{j=1}^{M} P_j \right) - 1 \right|}_{\text{multi-way penalty}}}_{\text{purity term}}. \tag{1}$$

where λ_1, and λ_2 are non-negative hyper-parameters. The *balancing term* guards an even split of examples between children nodes and is minimized for the *perfectly balanced split* when $P_1 = P_2 = ... = P_M$. The *class integrity term* ensures that examples belonging to the same class are not split between children nodes. This term is maximized when $\lceil \frac{M}{2} \rceil$ or $\lfloor \frac{M}{2} \rfloor$ probabilities from among $P_1^i, P_2^i, \ldots, P_M^i$ are equal to 1 and the remaining ones are equal to 0 for any $i = 1, 2, \ldots, K$. Thus at maximum, given any class i, the examples containing this class in their label set are not split between children, but they are instead simultaneously all sent to $\lceil \frac{M}{2} \rceil$ or $\lfloor \frac{M}{2} \rfloor$ children. The third term in the objective aims at compensating this multi-way assignment of examples. The *multi-way penalty* prevents sending examples to multiple directions too often. It is maximized when $\forall_{j=1,2,...,M}\ P_j = 1$ and minimized when $\sum_{j=1}^M P_j = 1$. Thus the *purity term*, defined as the sum of the class integrity term and the multi-way penalty, is minimized for the *perfectly pure split* when no example is sent to more than one children (in other words, this is when for any $i = 1, 2, \ldots, K$, $P_j^i = 1$ for one particular setting of j and $P_j^i = 0$ for all other js).

In the binary case the objective then simplifies to the following form:

$$J \coloneqq \underbrace{|P_R - P_L|}_{\text{balancing term}} \underbrace{\underbrace{-\lambda_1 \sum_{i=1}^{k} \pi_i \left|P_R^i - P_L^i\right|}_{\text{class integrity term}} + \underbrace{\lambda_2 |P_R + P_L - 1|}_{\text{multi-way penalty}}}_{\text{purity term}}, \tag{2}$$

where P_R and P_R^i (P_L and P_L^i) denote the probabilities that the example reaches right (left) child, marginally and conditional on class i respectively.

We aim to *minimize the objective* J to obtain high quality partitions. We next show theoretical properties of the objective introduced in Equation 1.

3 THEORETICAL RESULTS

In this section we analyze the properties of the objective and its influence on the purity and balancedness of node splits. Next we show its connection to the multi-label error.

3.1 General Properties of the Objective and its Relation to Node Partitions

The two lemmas below provide the basic mathematical understanding of the objective J.

Lemma 1. *(Binary tree) For any hypotheses $h_{R/L} \in \mathcal{H}$, the objective J defined in Equation 2 satisfies $J \in [-\lambda_1,\ \lambda_2]$ and it is minimized if and only if the split is perfectly balanced and perfectly pure.*

Lemma 1 generalizes to the tree of arbitrary width as follows:

Lemma 2. *(M-ary tree) For any hypotheses $h_j \in \mathcal{H}$, where $j = 1, 2, \ldots, M$, and sufficiently large λ_2, i.e. $(M - 3 < \frac{\lambda_2}{\lambda_1})$, the objective J defined in Equation 1 satisfies $J \in [-\lambda_1(M-1),\ \lambda_2(M-1)]$ and it is minimized if and only if the split is perfectly balanced and perfectly pure.*

Let J^* denote the lowest possible value of the objective J, i.e. $J^* = -\lambda_1(M-1)$.

Next we study how the objective promotes building nodes that are as balanced and pure as possible given the data. We first introduce useful definitions.

Definition 1. *(Balancedness) The node split is β-balanced if the following holds $\max_{j=\{1,2,...,M\}} \left| P_j - \frac{\sum_{i=1}^M P_i}{M} \right| = \beta$, where $\beta \in \left[0, 1 - \frac{1}{M}\right]$ is a balancedness factor.*

Note that a split is perfectly balanced if and only if $\beta = 0$.

Definition 2. *(Purity) The node split is α-pure if the following holds*

$$\frac{1}{M} \sum_{j=1}^{M} \sum_{i=1}^{K} \pi_i \min \left(P_j^i, \sum_{l=1}^{M} P_l^i - P_j^i \right) = \alpha, \tag{3}$$

where, $\alpha \in [0, 1]$ is a purity factor.

Note that a split is perfectly pure if and only if $\alpha = 0$.
Next lemmas show that in isolation, when either the purity or balancedness of the split is fixed, decreasing the value of the objective leads to recovering more balanced or pure split, respectively.

Lemma 3. *If a node split has a fixed purity term α, with corresponding J^{α}_{purity} then $\beta \leq J - J^{\alpha}_{purity}$.*

Lemma 4. *If a node split has a fixed balanced term β, with corresponding $J^{\beta}_{balance}$ and assuming that the following condition holds: $\lambda_1(M-1) + J^{\beta}_{balance} \geq \lambda_2 \geq \lambda_1 \frac{M-1}{2}$, then*

$$\alpha \leq (J - J^{\beta}_{balance} + \lambda_2) \frac{2}{M(2\lambda_2 - \lambda_1(M-1))}. \quad (4)$$

3.2 Relation of the Objective to the Multi-label Error

We will next explore the connection of the multi-label classification error with the proposed objective. For simplicity, assume each example has r labels. Denote $t(x)$ to be the true label set of x and $y_r(x)$ to be the assigned label set of size r by the tree. Denote $e_r(\mathcal{T})$ to be the r-level error with respect to the Precision@r measure, i.e. $e_r(\mathcal{T}) = 1 - \text{Precision@}r = \frac{1}{r}\sum_{i=1}^{K} P(i \in y_r(x), i \notin t(x))$.

3.2.1 Ideal Case

Here we consider the ideal case when the objective J is perfectly minimized in every node of the tree and show that in this case the tree achieves zero multi-label classification error.

Theorem 1. *When the objective function J from Equation 1 is perfectly minimized in every node of the tree, i.e. $J = J^*$, then the resulting multi-label tree achieves zero $\hat{r}$-level multi-label error, $e_{\hat{r}}(\mathcal{T})$, for any $\hat{r} = 1, 2, \ldots, r$.*

3.2.2 Real Case: Boosting Theorem

Next we prove a bound on the classification error for the LdSM tree. In particular, we show that if the proposed objective is "weakly" optimized in each node of the tree then our algorithm will boost this weak advantage to build a tree achieving any desired level of accuracy. This weak advantage is captured in a form of the Weak Hypothesis Assumptions. We restrict ourselves to the case of binary tree. We omit the analysis for the M-ary to avoid overcomplicating the notation.

We introduce the following weak assumptions.

Assumption 3.1. *γ-Weak Hypothesis Assumption: for any distribution $\mathcal{P}$ over the data, at each node of the tree $\mathcal{T}$ there exist a partition such that $\sum_i \pi_i \left|P^i_R - P^i_L\right| \geq \gamma$, where $\gamma \in (0, 1]$.*

Remark 1. *The above definition essentially assumes that in every node of the tree we are able to recover a partition with the corresponding class integrity term (a second component of our objective) bounded away from zero. Since the value of this term ranges in $[0, 1]$, such assumption is indeed very "weak".*

Also, specifically note that it is enough that for one class i the following holds: $|P^i_R - P^i_L| \geq \gamma$ in order to satisfy the assumption.

Assumption 3.1 leads to the lemma that captures the monotonic drop of the error with each split. A similar theorem for M-ary case is provided in the Supplement.

Lemma 5. *Under the Weak Hypothesis Assumption 3.1, $e_r(\mathcal{T})$ is monotonically decreasing with every split of the tree.*

We next introduce the second weak assumption.

Assumption 3.2. *c-Weak Hypothesis Assumption: at step t of the algorithm, there exist a leaf node l^* such that its weight, $w_{l^*} \geq \frac{c}{(t+1)}$, where $c \in (0, 1]$. w_{l^*} is the probability that a randomly chosen point from distribution $\mathcal{P}$ reaches the leaf l^*.*

Remark 2. *Note that at step t of the algorithm we have $t+1$ leaves. Also note that $\sum_{\tilde{\mathcal{L}} \subset \mathcal{L}} w_{\tilde{\mathcal{L}}} = 1$, where $\tilde{\mathcal{L}}$ is a subset of the tree leaves ($\mathcal{L}$ is the set of all leaves and the sum is taken over all subsets of tree leaves) and $w_{\tilde{\mathcal{L}}}$ is the weight of this subset, or equivalently, the probability that a randomly chosen point from distribution $\mathcal{P}$ reaches all leaves in $\tilde{\mathcal{L}}$.*

Consider the case when we do not send examples to more than one direction in every node of the tree. In this case there exists a leaf with c equal to 1 and therefore $w_l \geq 1/(t+1)$ for any leaf l. When c decreases, we allow to send more examples to multiple directions in the tree. In the Assumption 3.2 we let the examples to be sent to both directions at some nodes, therefore we require that there exists a leaf with $w_l \geq c/(t+1)$, for $c < 1$. Thus Assumption 3.2 is tightly correlated with the multi-way penalty term of the objective. Note also that naturally, c has to be bounded away from zero since every leaf receives at least one example.

Then the following theorem holds.

Theorem 2. *Under the Weak Hypothesis Assumptions 3.1 and 3.2 and an additional assumption that each node produces perfectly balanced split, for any $\alpha \in [0, 1]$ to obtain $e_r(\mathcal{T}) \leq \alpha$ it suffices to have a tree with t internal nodes that satisfy*

$$(t+1) \geq \left(\frac{1}{\alpha}\right)^{\frac{16 \ln K}{c^2 \gamma^2 (1-b) \log_2(e)}}, \quad (5)$$

where $b = |P_R + P_L - 1|$.

Consider an algorithm that builds the tree in a top-down fashion so that at each step it chooses the node with the highest weight, optimizes J at that node, and splits that node to its children. The theorem does not assume that we can optimize J perfectly in the node but instead only requires weak assumptions to hold. The above theorem guarantees that we can amplify the weak gain at each node to decrease the error below any desirable threshold. In practice we expect that J can be optimized far better than what is given by the weak hypothesis assumptions, which effectively reduces the number of required splits needed to achieve given multi-label

classification error. A generalization of Theorem 2 is provided in the Supplement (note it relies on more complicated assumptions though).

4 ALGORITHM

In this section we present the algorithm for simultaneous tree construction and training. We then discuss how to assign labels to the test example. The main algorithm for tree construction and training is captured in Algorithm 1. It presents the top-level procedure for building the tree. It includes three sub-algorithms which we will explain here but their pseudo-codes are deferred to the Supplement. The tree construction is performed in a top-down node-by-node fashion. Reaching the maximum number of nodes terminates further growth of the tree. As can be seen in Algorithm 1, we select a node to be expanded into children nodes based on the priority computed as the difference of the sum and maximum value of the bins of the label histogram in the node. The priority of the node is related to the weight of the node defined in section 3. We want to split nodes that are reached by many examples but we also require them to come from different classes, where at least two classes have significant mass. High priority is attained by these nodes that were visited by many examples that correspond to many different labels. When the node is selected for expansion, we train its regressors according to the procedure **TrainRegressors** (see Algorithm 4 in the Supplement for its pseudo-code). In **TrainRegressors** we stream multiple (#epochs) times through the data reaching that node and optimize the objective function for each example according to the procedure **OptimizeObjective** (see Algorithm 3 in the Supplement). In **OptimizeObjective** we search over all possible ways of sending an example to M directions (including multi-way cases) and we choose the set of directions for which J achieves the lowest value. **TrainRegressors** uses these computed optimal direction(s) to train its regressors using cross-entropy loss. Afterwards, it updates the probabilities P_js and P_j^is in the node. Instead of taking 1-increments per example when updating probabilities, we use regressor margins (clamped to the interval $[0, 1]$).

After training the regressors, we create children for the node according to the procedure **CreateChildren** (see Algorithm 5 in the Supplement). Based on the outputs of the regressors we assign data points to its children using rule explained in Section 2 and update children's label histograms accordingly.

At testing, the prediction is formed according to Algorithm 2. Specifically, the example is sent down the tree, from the root to one or more leaves, guided by node regressors. For examples that descended to multiple leaves, we estimate the label histogram by averaging the normalized label histograms of these leaves. The normalized label histogram is computed by dividing the label histogram by the sum of its entries. Given R (the input to the Algorithm 2), we assign to the test example top R labels that correspond to the highest entries in the resulting histogram.

Algorithm 1 BuildTree

% $v.I$ denotes the list of indices of examples reaching node v
Input: · maximum # of nodes: $T_{\max}$;
· tree width: M;
· # of training epochs: E;
· training data $(x_1, y_1), \ldots, (x_N, y_N)$
%y_i: all labels of the i^{th}example

procedure UpdateHist ($LHist$**,** y**)**
for $i \in y$ **do** $LHist[i] \mathrel{+}= 1$ **end for**

$v_{root}.I \leftarrow \{1, 2, ..., N\}$; $v_{root}.Lhist \leftarrow \emptyset$
for $i \in v_{root}.I$ **do**
  % add y_i to histogram
  UpdateHist ($v_{root}.Lhist$**,** y_i**)**
end for
$t \leftarrow 1$
$Q.push(v_{root}, 0)$ % initialize priority queue Q
while $Q \neq \emptyset$ **and** $t < Tmax$ **do**
  $v \leftarrow Q.pop()$
  TrainRegressors (v**)**
  $ch \leftarrow$ **CreateChildren (**v**)**
  for $m \in ch$ **do**
    $priority \leftarrow$
      $\sum_{k \in ch[m].Lhist} ch[m].Lhist[k]$
      $- \max_{k \in ch[m].Lhist} ch[m].Lhist[k]$
    $Q.push(ch[m], priority)$
  end for
  $t \leftarrow t + M$
end while
return v_{root}

The computational complexity analysis The complexity of the **TrainRegressors** is $\mathcal{O}(M(D + K) + eM\hat{N}(\hat{d} + 2^M\hat{k}))$, where D is the feature size, K is the label size, e is the number of epochs, $\hat{N}$ is the number of examples reaching the node, $\hat{d}$ is the average number of features per point and $\hat{k}$ is the average number of labels per point. The first term, $M(D + K)$, only corresponds to the initialization of the regressors and conditional probabilities. Note that since the feature and label spaces are sparse, $\hat{d}$ and $\hat{k}$ are small numbers compared to D and K. If we expect $e\hat{N}(\hat{d} + 2^M\hat{k}) << (D + K)$, the complexity can be further reduced when using a self-balancing binary search tree to store the sparse set of weight vectors and probabilities. This would result in $\tilde{\mathcal{O}}(eM\hat{N}(\hat{d} + 2^M\hat{k}))$ computational complexity (M is usually a small number. In our experiments we used $M = 2, 4$). Let $\hat{r}$ be the average number of leaves that each example descends to. Then the overall training

complexity when building a balanced tree would become $\tilde{\mathcal{O}}(N\hat{r}eM(2^M\hat{k}+\hat{d}))$, where N is the the size of the training data. Furthermore, the testing complexity would become $\mathcal{O}(\log(K)\hat{r}M(\hat{k}+\hat{d}))$ per test data point. Note that having an explicit balancing term in our objective function encourages building trees with logarithmic depth with respect to the total number of labels, K.

5 EXPERIMENTS

We evaluated LdSM on multiple benchmark data sets (Bibtex, Mediamill, Delicious, AmazonCat-13k, Wiki10-31k, Delicious-200K, and Amazon-670k) obtained from public repository Varma (2019). The data sizes are reported in Table 2 (D is the data dimensionality). The experimental setup is described in the Supplement.

Algorithm 2 Predict (x, R)

Input: · root of the trained tree: v_{root};
· #labels to predict per example: R;
· tree width: M

procedure GetLeaves(v)
if $v.isLeaf$ **then**
  $leafList.push(v)$
else
  for $m \in 1 \dots M$ **do**
    if $v.w_m^\top x > 0.5$ **then**
      GetLeaves(v_m)
      $sent \leftarrow true$
    end if
  end for
  if not $sent$ **then**
    $m \leftarrow \arg\max_{\hat{m}\in\{1,2,\dots,M\}} v.w_{\hat{m}}^\top x$
    GetLeaves(v_m)
  end if
end if

$leafList \leftarrow \emptyset$ % list of leaves reached by example x
GetLeaves(v_{root})
$hist \leftarrow \emptyset$
for $v_l \in leafList$ **do**
  $sum \leftarrow \sum_{k\in v_l.Lhist} v_l.Lhist[k]$
  for $k \in v_l.Lhist$ **do**
    $hist[k]$ += $v_l.Lhist[k]/sum$
  end for
end for
$labels \leftarrow$ select R top entries from $hist$
return $labels$

In Table 2 we compare the Precisions $P@1$, $P@3$, and $P@5$ and nDCG scores $N@1$, $N@3$, $N@5$ (see Varma (2019) for the explanation of these evaluation metrics) obtained by LdSM and other purely tree-based competitor algorithms: LPSR, FastXML, PFastreXML, PLT, GBDT-S, and CRAFTML.The performance of the competitors were obtained from the corresponding papers introducing these techniques and multi-label repository Varma (2019). The prediction with LdSM ensemble is done by averaging the resulting histograms for each tree and then selecting R labels. At training, each tree in the ensemble differs in regressors initialization. The reported results show that LdSM either matches or, on selected problems (including large Amazon-670k data set), outperforms the existing tree-based approaches in terms of both the Precision and the nDCG score.

In Table 3 we report the Precisions of LdSM compared with Parabel, the most recent OAA approach, which is also efficient compared to the other schemes. Parabel builds a hierarchy over labels and also learns powerful 1-vs-All classifiers in the leaf nodes. It can be considered as a hybrid technique which combines 1-vs-All with label-tree approaches. It has much better prediction time compared to the other 1-vs-All approaches while achieving similar accuracies as DiSMEC/PPD-Sparse. The comparison with the rest of the techniques are deferred to the Supplement. On bigger data sets, LdSM has some loss of statistical accuracy with respect to OAA methods. However, these techniques have fundamentally different underlying mechanism from ours, which usually result in their higher complexity and longer prediction time.

We observed the largest data set (Amazon-670k) suffer from the tail label problem. For this data set we use re-ranking approach similar to Jain et al. (2016). This is applied at testing, after our tree is built and trained. Re-ranking increases the test time by $\sim 50\%$ for Amazon-670k. In Table 4 we compare the performance of our approach against Parabel on tail labels using the propensity score variant of Precision. On most of the data sets LdSM has better performance.

In Table 1 we provide per-example prediction time (training time is deferred to the Supplement) for different data sets comparing LdSM with competitor methods, as well as with Parabel. Our result demonstrates that LdSM can perform efficient multi-label prediction, with respect to the tree-based methods as well as the other techniques including OAA approaches. (Refer to the Supplement for more results.)

Figure 2 shows that the depth of trees constructed with LdSM are $\mathcal{O}(\log_M(K))$, specifically they lie in the interval $[\log_M K, 3\log_M K]$ for Mediamill, Bibtex and Delicious-200k data sets and $[\log_M K, 2\log_M K]$ for Delicious, AmazonCat-13k, Wiki10-31k and Amazon-670k data sets. Next we discuss the results captured in Figure 3. Note that additional figures related to this study can be found in the Supplement. In the top plot we report the behavior of Precision and nDCG score as the size of the LdSM ensemble grows. Clearly the most rapid improvement in Precision is achieved when increasing the ensemble size to 10 trees (across different data sets this was found to be between

5 and 10, except Bibtex (case $M = 2$), for which it was 20). After that, the increase of $P@1$, $P@3$, $P@5$, $N@1$, $N@3$, and $N@5$ saturates and we obtain less than 2% improvement when increasing the ensemble further to 50. The same can be observed for nDCG score. The bottom plot captures how the Precision and nDCG score depend on the number of nodes in the tree and the depth of the deepest tree in the ensemble. As we increase the maximum allowed number of nodes (T_{max}) in the LdSM algorithm, it recovers $\mathcal{O}(log_M(T_{max}))$-depth trees. One can observe the general tendency that increasing the number of nodes ϕ times, results in increasing the tree depth by less than $2\log_M(\phi)$. We also observed that increasing the number of nodes/tree depth for most data sets leads to the improvement in Precisions $P@1$, $P@3$, and $P@5$ and nDCG scores $N@1$, $N@3$, and $N@5$ by less than 3%, suggesting that often shallower trees already achieve acceptable performance. The plots in Figure 6 in the Supplement demonstrate that single LdSM tree outperforms single FastXML tree. The same property holds for ensembles.

In Figure 1 we show how the objective function is optimized as we move from the root deeper into the tree. Intuitively root faces the most difficult optimization task as it sees the entire data set and consequently the objective function there is optimized more weakly, i.e. to a higher level, than in case of nodes lying deeper in the tree. As we move closer to the leaves, the convergence is faster due to the "cleaner" nature of the data received by the nodes there (less label variety).

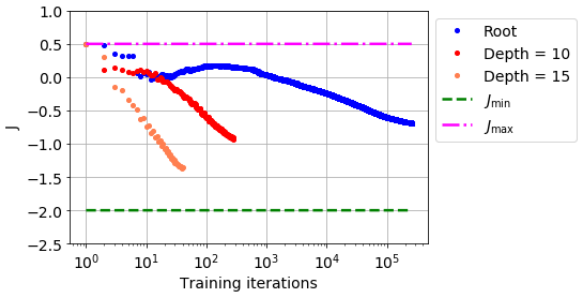

Figure 1: The behavior of the LdSM objective function J during training at different levels in the tree for an exemplary LdSM tree. Delicious data set. Tree depth is 20 and M was set to $M = 2$. J_{min} and J_{max} denote respectively the minimum and maximum value of J.

6 CONCLUSIONS

This paper develops a new decision tree algorithm, that we call LdSM, for multi-label classification problem. The technical contributions of this work include: a novel objective function and its corresponding theoretical analysis and a resulting novel algorithm for tree construction and training that we evaluate empirically. We find experimentally that LdSM is competitive to the state-of-the art multi-label approaches, performs efficient prediction, and achieves

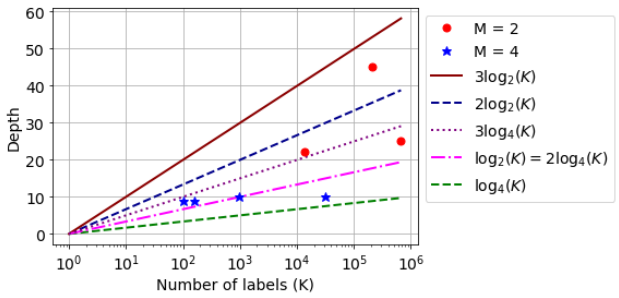

Figure 2: The depth of the deepest tree in the optimal LdSM tree ensemble (reported in Table 2) versus the number of labels in the data set (K).

Table 1: Prediction time [ms] per example for tree-based methods as well as Parabel on different data sets (LPSR and PLT are NA).

	Parabel	GBDT-S	CRAFTML	FastXML	PFastreXML	LdSM
Mediamill	NA	**0.05**	NA	0.27	0.37	**0.05**
Bibtex	NA	NA	NA	0.64	0.73	**0.013**
Delicious	NA	0.04	NA	NA	NA	**0.014**
AmazonCat-13k	NA	NA	5.12	1.21	1.34	**0.04**
Wiki10-31k	NA	0.20	NA	1.38	NA	**0.15**
Delicious-200k	NA	**0.14**	8.6	1.28	7.40	1.21
Amazon-670k	1.13	NA	5.02	1.48	1.98	**0.12**

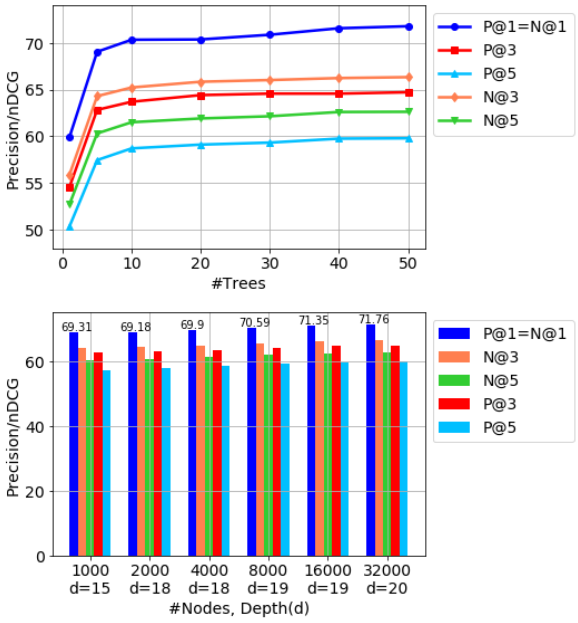

Figure 3: The behavior of precision/nDCG score as a function of the number of trees in the ensemble (**first row**) and number of nodes T_{max} (including leaves) and tree depth of the deepest tree in the ensemble (**second row**). M is set to $M = 2$. Plots were obtained for Delicious data set (Table 1(c)). The figure should be read in color.

Table 2: Precisions: $P@1$, $P@3$, and $P@5$ (%) and nDCG scores: $N@1$, $N@3$, and $N@5$ (%) obtained by different tree-based methods on common multi-label data sets.

(a) Mediamill $D = 120, K = 101$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	83.57	65.78	49.97	83.57	74.06	69.34
PLT	-	-	-	-	-	-
GBDT-S	84.23	67.85	-	-	-	-
CRAFTML	85.86	69.01	54.65	-	-	-
FastXML	84.22	67.33	53.04	84.22	75.41	72.37
PFastreXML	83.98	67.37	53.02	83.98	75.31	72.21
LdSM	**90.64**	**73.60**	**58.62**	**90.64**	**82.14**	**79.23**

(b) Bibtex $D = 1.8k, K = 159$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	62.11	36.65	26.53	62.11	56.50	58.23
PLT	-	-	-	-	-	-
GBDT-S	-	-	-	-	-	-
CRAFTML	**65.15**	**39.83**	28.99	-	-	-
FastXML	63.42	39.23	28.86	63.42	59.51	61.70
PFastreXML	63.46	39.22	29.14	63.46	59.61	62.12
LdSM	64.69	39.70	**29.25**	**64.69**	**60.37**	**62.73**

(c) Delicious $D = 500, K = 983$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	65.01	58.96	53.49	65.01	60.45	56.38
PLT	-	-	-	-	-	-
GBDT-S	69.29	63.62	-	-	-	-
CRAFTML	70.26	63.98	59.00	-	-	-
FastXML	69.61	64.12	59.27	69.61	65.47	61.90
PFastreXML	67.13	62.33	58.62	67.13	63.48	60.74
LdSM	**71.91**	**65.34**	**60.24**	**71.91**	**66.90**	**63.09**

(d) AmazonCat-13k $D = 204k, K = 13k$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	-	-	-	-	-	-
PLT	91.47	75.84	61.02	-	-	-
GBDT-S	-	-	-	-	-	-
CRAFTML	92.78	78.48	63.58	-	-	-
FastXML	93.11	**78.2**	63.41	93.11	**87.07**	**85.16**
PFastreXML	91.75	77.97	**63.68**	91.75	86.48	84.96
LdSM	**93.87**	75.41	57.86	**93.87**	85.06	80.63

(e) Wiki10-31k $D = 102k, K = 31k$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	72.72	58.51	49.50	72.72	61.71	54.63
PLT	84.34	72.34	62.72	-	-	-
GBDT-S	84.34	70.82	-	-	-	-
CRAFTML	**85.19**	**73.17**	**63.27**	-	-	-
FastXML	83.03	67.47	57.76	83.03	**75.35**	63.36
PFastreXML	83.57	68.61	59.10	83.57	72.00	64.54
LdSM	83.74	71.74	61.51	**83.74**	74.60	**66.77**

(f) Delicious-200k $D = 783k, K = 205k$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	18.59	15.43	14.07	18.59	16.17	15.13
PLT	45.37	38.94	35.88	-	-	-
GBDT-S	42.11	39.06	-	-	-	-
CRAFTML	**47.87**	**41.28**	38.01	-	-	-
FastXML	43.07	38.66	36.19	43.07	39.70	37.83
PFastreXML	41.72	37.83	35.58	41.72	38.76	37.08
LdSM	45.26	40.53	**38.23**	**45.26**	**41.66**	**39.79**

(g) Amazon-670k $D = 135k, K = 670k$						
Algorithm	P@1	P@3	P@5	N@1	N@3	N@5
LPSR	28.65	24.88	22.37	28.65	26.40	25.03
PLT	36.65	32.12	28.85	-	-	-
GBDT-S	-	-	-	-	-	-
CRAFTML	37.35	33.31	30.62	-	-	-
FastXML	36.99	33.28	30.53	36.99	35.11	33.86
PFastreXML	39.46	35.81	33.05	39.46	37.78	36.69
LdSM	**42.63**	**38.09**	**34.70**	**42.63**	**40.37**	**38.89**

Table 3: Precisions: $P@1$, $P@3$, and $P@5$ (%) obtained by LdSM and OAA approach (Parabel) on common multi-label data sets.

	OAA (Parabel)			LdSM		
	P@1	P@3	P@5	P@1	P@3	P@5
Mediamill	83.91	67.12	52.99	**90.64**	**73.60**	**58.62**
Bibtex	64.53	38.56	27.94	**64.69**	**39.70**	**29.25**
Delicious	67.44	61.83	56.75	**71.91**	**65.34**	**60.24**
AmazonCat-13k	93.03	**79.16**	**64.52**	**93.87**	75.41	57.86
Wiki10-31k	**84.31**	**72.57**	**63.39**	83.74	71.74	61.51
Delicious-200k	**46.97**	40.08	36.63	45.26	**40.53**	**38.23**
Amazon-670k	**44.89**	**39.80**	**36.00**	42.63	38.09	34.70

Table 4: Propensity Score Precisions: $PSP@1$, $PSP@3$, and $PSP@5$ (%) obtained by LdSM and OAA approach (Parabel) on common multi-label data sets.

	OAA (Parabel)			LdSM		
	PSP @1	PSP @3	PSP @5	PSP @1	PSP @3	PSP @5
Mediamill	66.51	65.21	64.30	**70.27**	**69.66**	**68.86**
Bibtex	50.88	52.42	57.36	**52.01**	**54.38**	**60.34**
Delicious	32.69	34.00	34.53	**37.27**	**38.32**	**38.46**
AmazonCat-13k	50.93	**64.00**	**72.08**	**51.06**	58.67	60.47
Wiki10-31k	11.66	**12.73**	**13.68**	**11.87**	12.35	12.89
Delicious-200k	**7.25**	7.94	8.52	7.16	**8.26**	**9.11**
Amazon-670k	25.43	29.43	32.85	**28.14**	**30.82**	**33.16**

high multi-label accuracy with logarithmic-depth trees. This new method is therefore suitable for applications involving large label spaces.

References

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. (2013). Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*.

Babbar, R. and Schölkopf, B. (2017). Dismec: Distributed sparse machines for extreme multi-label classification. In *ACM WSDM*.

Babbar, R. and Schölkopf, B. (2019). Data scarcity, robustness and extreme multi-label classification. *Machine Learning*.

Balasubramanian, K. and Lebanon, G. (2012). The landmark selection method for multiple output prediction. In *ICML*.

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

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. (2015). Sparse local embeddings for extreme multi-label classification. In *NIPS*.

Bi, W. and Kwok, J. (2013). Efficient multi-label classification with many labels. In *ICML*.

Bottou, L. (1998). Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press.

Chen, Y.-N. and Lin, H.-T. (2012). Feature-aware label space dimension reduction for multi-label classification. In *NIPS*.

Choromanska, A. and Langford, J. (2015). Logarithmic time online multiclass prediction. In *NIPS*.

Cisse, M. M., Usunier, N., Artières, T., and Gallinari, P. (2013). Robust bloom filters for large multilabel classification tasks. In *NIPS*.

Dasgupta, S. (2008). Topics in unsupervised learning. `http://cseweb.ucsd.edu/~dasgupta/291-unsup/lec6.pdf`.

Deng, J., Satheesh, S., Berg, A. C., and Fei-Fei, L. (2011). Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*.

Ferng, C.-S. and Lin, H.-T. (2011). Multi-label classification with error-correcting codes. In *ACML*.

H. Fang, M. Chengy, C.-J. H. and Friedlander, M. (2019). Fast training for large-scale one-versus-all linear classifiers using tree-structured initialization. In *SDM*.

Hsu, D. J., Kakade, S. M., Langford, J., and Zhang, T. (2009). Multi-label prediction via compressed sensing. In *NIPS*.

Jain, H., Prabhu, Y., and Varma, M. (2016). Extreme multi-label loss functions for recommendation, tagging, ranking and other missing label applications. In *ACM SIGKDD*.

Jasinska, K., Dembczynski, K., Busa-Fekete, R., Pfannschmidt, K., Klerx, T., and Hullermeier, E. (2016). Extreme f-measure maximization using sparse probability estimates. In *ICML*.

Jernite, Y., Choromanska, A., and Sontag, D. (2017). Simultaneous learning of trees and representations for extreme classification and density estimation. In *ICML*.

Ji, S., Tang, L., Yu, S., and Ye, J. (2008). Extracting shared subspace for multi-label classification. In *KDD*.

Kearns, M. and Mansour, Y. (1999). On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128.

Khandagale, S., Xiao, H., and Babbar, R. (2019). Bonsai - Diverse and Shallow Trees for Extreme Multi-label Classification. *CoRR*, abs/1904.08249.

Lin, Z., Ding, G., Hu, M., and Wang, J. (2014). Multi-label classification via feature-aware implicit label space encoding. In *ICML*.

Liu, J., Chang, W.-C., Wu, Y., and Yang, Y. (2017). Deep learning for extreme multi-label text classification. In *ACM SIGIR*.

Mnih, A. and Hinton, G. E. (2009). A scalable hierarchical distributed language model. In *NIPS*.

Nestrov, Y. (2004). *Introductory lectures on convex optimization : a basic course*. Applied optimization, Kluwer Academic Publ.

Niculescu-Mizil, A. and Abbasnejad, E. (2017). Label Filters for Large Scale Multilabel Classification. In *AISTATS*.

Prabhu, Y., Kag, A., Gopinath, S., Dahiya, K., Harsola, S., Agrawal, R., and Varma, M. (2018a). Extreme multi-label learning with label features for warm-start tagging, ranking and recommendation. In *ACM ICWSDM*.

Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., and Varma, M. (2018b). Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *WWW*.

Prabhu, Y. and Varma, M. (2014). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *ACM SIGKDD*.

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141.

Ross, S., Mineiro, P., and Langford, J. (2013). Normalized online learning. *CoRR*, abs/1305.6646.

Shalev-Shwartz, S. (2012). Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2):107–194.

Si, S., Zhang, H., Keerthi, S. S., Mahajan, D., Dhillon, I. S., and Hsieh, C.-J. (2017). Gradient boosted decision trees for high dimensional sparse output. In *ICML*.

Tai, F. and Lin, H.-T. (2012). Multilabel classification with principal label space transformation. *Neural Comput.*, 24(9):2508–2542.

Tianshi, G. and Koller, D. (2011). Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *ICCV*.

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

Varma, M. (2019). The Extreme Classification Repository. `http://manikvarma.org/downloads/XC/XMLRepository.html`.

W. Siblini, P. K. and Meyer, F. (2018). Craftml, an efficient clustering-based random forest for extreme multi-label learning. In *ICML*.

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

Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking. In *ICML*.

Wydmuch, M., Jasinska, K., Kuznetsov, M., Busa-Fekete, R., and Dembczynski, K. (2018). A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *NIPS*.

Yen, I. E., Huang, X., Dai, W., Ravikumar, P., Dhillon, I. S., and Xing, E. P. (2017). Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *SIGKDD*.

Yen, I. E., Huang, X., Ravikumar, P., Zhong, K., and Dhillon, I. S. (2016). Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. In *ICML*.

You, R., Zhang, Z., Wang, Z., Dai, S., Mamitsuka, H., and Zhu, S. (2019). Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *NIPS*.

Yu, H.-F., Jain, P., Kar, P., and Dhillon, I. (2014). Large-scale multi-label learning with missing labels. In *ICML*.

Zhang, W., Yan, J., Wang, X., and Zha, H. (2018). Deep extreme multi-label learning. In *ACM ICMR*.

Zhang, Y. and Schneider, J. (2011). Multi-label output codes using canonical correlation analysis. In *AISTATS*.