Logarithmic Time Online Multiclass prediction

Anna Choromanska*
Courant Institute of Mathematical Sciences
New York University
aec2163@columbia.edu

John Langford
Microsoft Research
New York, NY, USA
jcl@microsoft.com

Abstract

We study the problem of multiclass classification with an extremely large number of classes, with the goal of obtaining train and test time complexity logarithmic in the number of classes. We develop top-down tree construction approaches for constructing logarithmic depth trees. On the theoretical front, we formulate a new objective function, which is optimized at each node of the tree and creates dynamic partitions of the data which are both pure (in terms of class labels) and balanced. We demonstrate that under favorable conditions, we can construct logarithmic depth trees that have leaves with low label entropy. However, the objective function at the nodes is challenging to optimize computationally. We address the empirical problem with a new online decision tree construction procedure. Experiments demonstrate that this online algorithm quickly achieves small error rates relative to more common $O(k)$ approaches and simultaneously achieves significant improvement in test error compared to other logarithmic training time approaches.

1 Introduction

The central problem of this paper is computational complexity in a setting where the number of classes k for multiclass prediction is very large. Such problems occur in natural language (Which translation is best?), search (What result is best?), and detection (Who is that?) tasks. Almost all machine learning algorithms (with the notable exception of decision trees) have running times for multiclass classification which are $\mathcal{O}(k)$ with a canonical example being one-against-all classifiers [1]. In this setting, the most efficient imaginable approach has a running time of $\mathcal{O}(\log k)$ per example for both training and testing, while effectively using online learning algorithms to minimize passes over the data. Our central goal is improving the quality of most-efficient solutions so as to grow the frontier of tasks solvable via machine learning.

The goal of logarithmic (in k) complexity naturally motivates approaches that construct a logarithmic depth hierarchy over the labels, with one label per leaf. While this hierarchy is sometimes available through prior knowledge, in many scenarios it needs to be learned as well. This naturally leads to a *partition* problem which arises at each node in the hierarchy. The partition problem is finding a classifier: $c : X \to \{-1, 1\}$ which divides examples into two subsets with a purer set of labels than the original set. Definitions of purity vary, but canonical examples are the number of labels remaining in each subset, or softer notions such as the average Shannon entropy of the class labels. Despite resulting in a classifier, this problem is fundamentally different from standard binary classification. To see this, note that replacing $c(x)$ with $-c(x)$ is very bad for binary classification, but has no impact on the quality of a partition[1]. The partition problem (just like most natural clustering objectives) is fundamentally non-convex for symmetric classes since the average $\frac{c(x)-c(x)}{2}$ of $c(x)$ and $-c(x)$ creates a very poor partition.

*A part of this work was done when A.C. was at Microsoft Research New York.

[1]The problem bears parallels to clustering in this regard.

The choice of partition matters in problem dependent ways. For example, consider examples on a line with label i at position i and threshold classifiers. In this case, trying to partition class labels $\{1, 3\}$ from class label 2 results in poor performance.

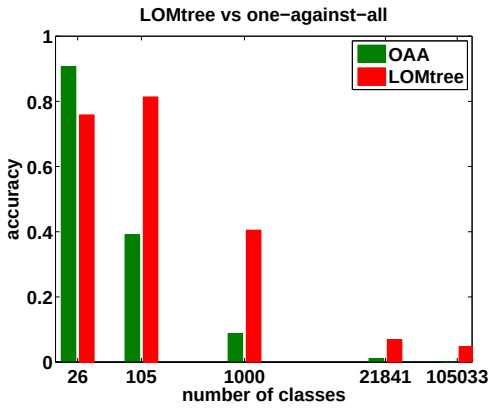

Figure 1: A comparison of One-Against-All and the Logarithmic Online Multiclass Tree (LOMtree) with one-against-all constrained to use the same training time as the LOMtree by dataset truncation and LOMtree constrained to use the same representation complexity as one-against-all. As the number of class labels grows, the problem becomes harder and the LOMtree becomes more dominant.

The partition problem is typically solved for decision tree learning via an enumerate-and-test approach amongst a small set of possible classifiers. In the multiclass setting, it is desirable to achieve substantial error reduction for each node in the tree which motivates using a richer set of classifiers in the nodes to minimize the number of nodes, and thereby decrease the computational complexity. The main theoretical contribution of this work is to establish a boosting algorithm for learning trees with $O(k)$ nodes and $O(\log k)$ depth, thereby addressing the goal of logarithmic time train and test complexity. In Section 2.3 this result generalizes a binary boosting-by-decision-tree theorem [2] to multiclass boosting. As in all boosting results, performance is critically dependent on the quality of the *weak learner*, supporting intuition that we need sufficiently rich partitioners at nodes. The approach uses a new objective for decision tree learning, which we optimize at each node of the tree. The objective and its theoretical properties are presented in Section 2.

A complete system with multiple partitions could be constructed top down (as the boosting theorem) or bottom up (as Filter Tree [3]). A bottom up partition process appears impossible with representational constraints as shown in appendix section 6 so we focus on top-down tree creation.

Whenever there are representational constraints on partitions (such as linear classifiers), finding a strong partition function requires an efficient search over this set of classifiers. Efficient searches over large function classes are routinely performed via gradient descent techniques for supervised learning, so they seem like a natural candidate. In existing literature, examples for doing this exist when the problem is indeed binary, or when there is a pre-specified hierarchy over the labels and we just need to find partitioners aligned with that hierarchy. Neither of these cases applies—we have multiple labels and want to dynamically create the choice of partition, rather than assuming that one was handed to us. Does there exist a purity criterion amenable to a gradient descent approach? The precise objective studied in theory fails this test due to its discrete nature, and even natural approximations are challenging to tractably optimize under computational constraints. As a result, we use the theoretical objective as a motivation and construct a new Logarithmic Online Multiclass Tree (LOMTree) algorithm for empirical evaluation.

Creating a tree in an online fashion creates a new class of problems. What if some node is initially created but eventually proves useless because no examples go to it? At best this results in a wasteful solution, while in practice it starves other parts of the tree which need representational complexity. To deal with this, we design an efficient process for recycling orphan nodes into locations where they are needed, and prove that the number of times a node is recycled is at most logarithmic in the number of examples. The algorithm is described in Section 3 and we analyze the swapping bound in Section 3.1.

And is it effective? Given the inherent non-convexity of the partition problem this is unavoidably an empirical question which we answer on a range of datasets varying from 26 to 105K classes in Section 4. We find that under constrained training times, this approach is quite effective compared to all baselines while dominating other $O(\log k)$ train time approaches.

What's new? To the best of our knowledge, the splitting criterion, the boosting statement, the LOMTree algorithm, the swapping guarantee, and the experimental results are all new here.

1.1 Prior Work

Only a few authors have previously addressed logarithmic time training. The Filter Tree [3] addresses consistent (and robust) multiclass classification, showing that it is possible in the statistical limit. The Filter Tree does not address the partition problem as we do here. In our experiments we compare with the Filter Tree empirically and find that addressing the partition problem is often helpful. The partition finding problem is addressed in the conditional probability tree [4], but that paper addresses conditional probability estimation. Conditional probability estimation can be converted into multiclass prediction, but doing so is not a logarithmic time operation.

Quite a few authors have addressed logarithmic testing time while allowing training time to be $O(k)$ or worse. While these approaches are intractable on our larger scale problems, we describe them here for context. The partition problem can be addressed by recursively applying spectral clustering on a confusion graph [5]. Empirically, this approach has been found to sometimes lead to badly imbalanced splits [6]. In the context of ranking, another approach uses k-means hierarchical clustering to recover the label sets for a given partition [7].

The more recent work [8] on the multiclass classification problem addresses it via sparse output coding by tuning high-cardinality multiclass categorization into a bit-by-bit decoding problem, where for the sake of scalability to large-scale problems, the authors decouple the learning processes of coding matrix and bit predictors and use probabilistic decoding to decode the optimal class label. The variant of the popular error correcting output code scheme for solving multi-label prediction problems with large output spaces under the assumption of output sparsity was also considered in previous work [9]. In general however their approach requires $O(k)$ running time to decode.

Decision trees are naturally structured to allow logarithmic time prediction. Traditional decision trees often have difficulties with a large number of classes because their splitting criteria are not well-suited to the large class setting. However, newer approaches [10] have addressed this effectively at significant scales in the context of multilabel classification (multilabel learning, with missing labels, is also addressed in [11] where the authors focus on the empirical risk minimization framework with a low-rank constraint). Additionally, a well-known problem with hierarchical classification is that the performance significantly deteriorates lower in the hierarchy [12] which some authors solve by biasing the training distribution to reduce error propagation while simultaneously combining bottom-up and top-down approaches during training [13].

2 Framework and theoretical analysis

In this section we describe the essential elements of the approach, and outline the theoretical properties of the resulting framework. We begin by presenting the high-level ideas.

2.1 Setting

We employ a hierarchical approach for learning a multiclass decision tree structure, training this structure in a *top-down* fashion. We assume that we receive examples $x \in \mathcal{X} \subseteq \mathbb{R}^d$, with labels $y \in \{1, 2, \ldots, k\}$. We also assume access to a hypothesis class $\mathcal{H}$ where each $h \in \mathcal{H}$ is a binary classifier, $h : \mathcal{X} \mapsto \{-1, 1\}$. The overall objective is to learn a tree of depth $O(\log k)$, where each node in the tree consists of a classifier $h \in \mathcal{H}$. The classifiers are trained in such a way that $h_n(x) = 1$ means that the example x is sent to the right subtree of node n, while $h(x) = -1$ sends x to the left subtree. When we reach a leaf node, we just predict according to the label with the highest frequency amongst the examples reaching that leaf.

In the interest of computational complexity, we want to encourage the number of examples going to the left and right to be *fairly balanced*. For good statistical accuracy, we want to send examples of class i almost exclusively to either the left or the right subtree, thereby refining the *purity* of the class distributions at subsequent levels in the tree. *Purity* of the tree node is therefore the measure of whether the examples of each class reaching the node are sent almost exclusively to its one child node (pure split) or to both children (impure split). The formal definitions of *balancedness* and *purity* will be introduced in Section 2.2. An objective balancing these competing criteria, and resulting theoretical properties is illustrated in the following sections. A key consideration in picking this objective is that we want to effectively optimize it over hypotheses $h \in \mathcal{H}$, while streaming over

examples in an online fashion. This seems unsuitable with some of the more standard decision tree objectives such as Shannon or Gini entropy, which leads us to design a new objective. At the same time, we show in Section 2.3 that under suitable assumptions, optimizing the objective also leads to effective reduction of the average Shannon entropy over the entire tree.

2.2 An objective and analysis of resulting partitions

We now define a criterion to measure the quality of a hypothesis $h \in \mathcal{H}$ in creating partitions at a fixed node n in the tree. Let π_i denote the proportion of label i amongst the examples reaching this node. Let $P(h(x) > 0)$ and $P(h(x) > 0|i)$ denote the fraction of examples reaching n for which $h(x) > 0$, marginally and conditional on class i respectively. Then we define the objective:

$$J(h) = 2 \sum_{i=1}^{k} \pi_i \left| P(h(x) > 0) - P(h(x) > 0|i) \right|, \tag{1}$$

where we aim to *maximize the objective* $J(h)$ to obtain high quality partitions. Intuitively, the objective encourages the fraction of examples going to the left from class i to be substantially different from the background fraction for each class i. As a concrete simple scenario, if $P(h(x) > 0) = 0.5$ for some hypothesis h, then the objective prefers $P(h(x) > 0|i)$ to be as close to 0 or 1 as possible for each class i, leading to pure partitions. We now make these intuitions more formal.

Definition 1 (Purity)**.** *The hypothesis $h \in \mathcal{H}$ induces a pure split if*

$$\alpha := \sum_{i=1}^{k} \pi_i \min(Pr(h(x) > 0|i), Pr(h(x) < 0|i)) \leq \delta,$$

where $\delta \in [0, 0.5)$, and α is called the purity factor.

In particular, a partition is called *maximally pure* if $\alpha = 0$, meaning that each class is sent exclusively to the left or the right. It is easily seen that for a hypothesis h which induces maximally pure splits, $J(h) = 2P(h(x) > 0)$. We now define a similar definition for the balancedness of a split.

Definition 2 (Balancedness)**.** *The hypothesis $h \in \mathcal{H}$ induces a balanced split if*

$$c \leq \underbrace{Pr(h(x) > 0)}_{=\beta} \leq 1 - c,$$

where $c \in [0, 0.5)$, and β is called the balancing factor.

A partition is called *maximally balanced* if $\beta = 0.5$, meaning that an equal number of examples are sent to the left and right children of the partition. The balancing factor and the purity factor are related as shown in Lemma 1 (most of the proofs are deferred to the Supplementary material).

Lemma 1. *For any hypothesis h, and any distribution over examples (x, y), the purity factor α and the balancing factor β satisfy*

$$\alpha \leq \min \left\{ \frac{2 - J(h)}{4\beta} - \beta, 0.5 \right\}. \tag{2}$$

A partition is called *maximally pure and balanced* if it satisfies both $\alpha = 0$ and $\beta = 0.5$. We see that $J(h) = 1$ for a hypothesis h inducing a maximally pure and balanced partition as captured in the next lemma. Of course we do not expect to have hypotheses producing maximally pure and balanced splits in practice.

Lemma 2. *For any hypothesis $h : \mathcal{X} \mapsto \{-1, 1\}$, the objective $J(h)$ satisfies $J(h) \in [0, 1]$. Furthermore, if h induces a maximally pure and balanced partition then $J(h) = 1$.*

2.3 Quality of the entire tree

The above section helps us understand the quality of an individual split produced by effectively maximizing $J(h)$. However, in a hierarchical setting, we must reason about the quality of the entire tree as we add more and more nodes. Does each new node ensure sufficient improvement in the

statistical quality? Can we obtain high quality predictions with shallow trees? While this may not hold in general, we adopt a boosting-style approach to this question here. We measure the quality of trees using the average entropy over all the leaves in the tree, and track the decrease of this entropy as a function of the number of nodes. In doing so, we borrow from the theoretical analysis of decision tree algorithms in Kearns and Mansour [2] originally developed to show the boosting properties of the decision trees for binary classification problems. We generalize their theoretical analysis to the multiclass classification setting.

Given a tree T, we consider the entropy function G as the measure of the quality of tree:

$$G(T) = \sum_{n \in \mathcal{L}} w_n \sum_{i=1}^{k} -\pi_{ni} \ln(\pi_{ni})$$

where π_{ni}'s are the probabilities that a randomly chosen x drawn from the underlying target distribution $\mathcal{P}$ has label i given that x reaches leaf n, $\mathcal{L}$ denotes the set of all tree leaves and w_n is the weight of leaf n defined as the probability of randomly chosen x drawn from $\mathcal{P}$ to reach leaf n (note that $\sum_{n \in \mathcal{L}} w_n = 1$).

The analysis studies a tree construction algorithm where we recursively find the leaf node with the highest weight, and choose to split it into two children. Consider the tree T constructed over t steps. Since each step takes one leaf node and splits it into two, we have a total of t leaf nodes after t steps. Let n be the heaviest node at time t and its weight w_n be denoted by w for brevity. Consider splitting this leaf to two children n_0 and n_1. For the ease of notation let $w_0 = w_{n_0}$ and $w_1 = w_{n_1}$. Also for the ease of notation let $\beta = P(h_n(x) > 0)$ and $P_i = P(h_n(x) > 0|i)$. Let π_i be the shorthand for π_{ni}. Recall that $\beta = \sum_{i=1}^{k} \pi_i P_i$ and $\sum_{i=1}^{k} \pi_i = 1$. Also notice that $w_0 = w(1-\beta)$ and $w_1 = w\beta$. Let $\boldsymbol{\pi}$ be the k-element vector with i^{th} entrance equal to π_i. Furthermore let $G(\boldsymbol{\pi}) = \sum_{i=1}^{k} -\pi_i \ln(\pi_i)$.

Let G_t be a shorthand for $G(T)$, the entropy of tree T with t leaves. Before the split the contribution of node n to G_t was $wG(\pi_1, \pi_2, \ldots, \pi_k)$. Let $\pi_i(n_0) = \frac{\pi_i(1-P_i)}{1-\beta}$ and $\pi_i(n_1) = \frac{\pi_i P_i}{\beta}$ be the probabilities that a randomly chosen x drawn from $\mathcal{P}$ has label i given that x reaches nodes n_0 and n_1 respectively. Furthermore let $\boldsymbol{\pi}(n_0)$ be the k-element vector with i^{th} entry equal to $\pi_i(n_0)$ and let $\boldsymbol{\pi}(n_1)$ be the k-element vector with i^{th} entry equal to $\pi_i(n_1)$. Notice that $\boldsymbol{\pi} = (1-\beta)\boldsymbol{\pi}(n_0) + \beta\boldsymbol{\pi}(n_1)$. After the split the contribution of the same, now internal, node n changes to $w((1-\beta)G(\pi(n_0)) + \beta G(\pi(n_1))$. We denote the difference between them as Δ_t and thus

$$\Delta_t := G_{t+1} - G_t = w\left[G(\boldsymbol{\pi}) - (1-\beta)G(\boldsymbol{\pi}(n_0)) - \beta G(\boldsymbol{\pi}(n_1))\right]. \tag{3}$$

We aim to lower-bound Δ_t. The entropy reduction of Equation 3 [2] corresponds to a gap in the Jensen's inequality applied to the concave function $G(\boldsymbol{\pi})$. One can show that $\Delta_t \geq \frac{J^2 G_t}{8\beta(1-\beta)t \ln k}$, thus the larger the objective $J(h)$ is at time t, the larger the entropy reduction ends up being, which further reinforces intuitions to maximize J. In general, it might not be possible to find any hypothesis with a large enough objective $J(h)$ to guarantee sufficient progress at this point so we appeal to a *weak learning assumption* described next.

The assumption posits that each non-leaf node of the tree T has a hypothesis h in its hypothesis class $\mathcal{H}$ which guarantees a "weak" purity of the split on any distribution $\mathcal{P}$ over $\mathcal{X}$. Under this assumption, one can use the new decision tree approach to drive the error below any threshold.

Definition 3 (Weak Hypothesis Assumption)**.** *Let $\gamma \in (0, \min(\beta_n, 1-\beta_n)]$. Let for any distribution $\mathcal{P}$ over $\mathcal{X}$ at each non-leaf node n of the tree T there exists a hypothesis $h \in \mathcal{H}$ such that $J(h)/2 = \sum_{i=1}^{k} \pi_i |P_{ni} - \beta_n| \geq \gamma$.*

The *weak hypothesis assumption* can be used to further lower-bound Δ_t as $\Delta_t > \frac{\eta^2 G_t}{16t}$ which directly leads to Theorem 1.

Theorem 1. *Under the Weak Hypothesis Assumption, for any $\alpha \in [0, 1]$, to obtain $G_t \leq \alpha$ it suffices to make*

$$t \geq \left(\frac{1}{\alpha}\right)^{\frac{4(1-\gamma)^2 \ln k}{\gamma^2}} \qquad \textit{splits.}$$

3 The LOMTree Algorithm

The objective function of Section 2 has another convenient equivalent form which yields a simple online algorithm for tree construction and training. Notice that the objective function defined in Equation 1 can be equivalently written as

$$J(h) = 2|\mathbb{E}_x[\mathbb{1}(h(x) > 0)] - \mathbb{E}_{x,i}[\mathbb{1}(h(x) > 0|i)]|$$

Maximizing this objective is a discrete optimization problem that can be relaxed, by dropping the indicator operator, to the following form

$$J(h) = 2|\mathbb{E}_x[h(x)] - \mathbb{E}_{x,i}[h(x)|i]|,$$

where $E_{x,i}[h(x)|i]$ is the expected signed margin of class i.

The empirical estimates of the expectations can be easily stored and updated online in every node of the tree. The decision whether to send an example x reaching a node to its left or right child node is based on the sign of the difference between the two expectations: $\mathbb{E}_x[h(x)]$ and $\mathbb{E}_{x,y}[h(x)|y]$ where y is a label of the the data point x. During training, the algorithm assigns a unique label to each node of the tree which is currently a leaf. This is the label with the highest frequency amongst the examples reaching that leaf. While testing, a test example is pushed down the tree along the path starting from the root and ending at the leaf, where in each non-leaf node of the path its regressor directs the example either to the left or right child node. The test example is then labeled with the label assigned to the leaf that this example descended to.

Algorithm 1 Online tree training

Input: regression algorithm R, max number of tree non-leaf nodes T, swap resistance R_S

Subroutine **SetupNode** (v)

$\boldsymbol{m}^v = \text{zeros}(k, 1)$	(sum of the margins per class)
$\boldsymbol{l}^v = \text{zeros}(k, 1)$	(# of data points per class reaching v)
$\boldsymbol{n}^v = \text{zeros}(k, 1)$	(# of data points which are used to train regressor in v)
$\boldsymbol{e}^v = \text{zeros}(k, 1)$	(expected margin per class)
$C^v = 0$	(the size of the smallest leaf in the subtree with root v)

Subroutine **UpdateC** (v)

While ($v \neq r$ AND $C^{\text{PARENT}(v)} \neq C^v$) $\{v = \text{PARENT}(v);\ C^v = \min(C^{\text{LEFT}(v)}, C^{\text{RIGHT}(v)})\}$

Subroutine **Swap** (v)

Find a leaf s for which $(C^s = C^r)$
$s_{\text{PA}} = \text{PARENT}(s)$, $s_{\text{GPA}} = \text{GRANDPARENT(s)}$, $s_{\text{SIB}} = \text{SIBLING}(s)$
If ($s_{\text{PA}} = \text{LEFT}(s_{\text{GPA}})$) $\{\text{LEFT}(s_{\text{GPA}}) = s_{\text{SIB}}\}$ **Else** $\{\text{RIGHT}(s_{\text{GPA}}) = s_{\text{SIB}}\}$
UpdateC (s_{SIB}); **SetupNode** (s); $\text{LEFT}(v) = s$; **SetupNode** (s_{PA}); $\text{RIGHT}(v) = s_{\text{PA}}$

Create the root node $r = 0$: **SetupNode** (r); $N = 1$
For each example $(\boldsymbol{x}, y)$ **do** {
  Set $j = r$
  While j is not a leaf **do** {
    If $\left(\frac{\sum_{i=1}^{k} \boldsymbol{m}^v(y)}{\sum_{i=1}^{k} \boldsymbol{n}^v(y)} > \boldsymbol{e}^j(y)\right)$ $\{c = -1\}$ **Else** $\{c = 1\}$
    Train f^j with example $(\boldsymbol{x}, c)$
    $\boldsymbol{l}^j(y)$++; $\boldsymbol{m}_c^j(y)$ += $f^j(\boldsymbol{x})$; $\boldsymbol{n}_c^j(y)$ ++; $\boldsymbol{e}_c^j(y) = \boldsymbol{m}_c^j(y)/\boldsymbol{n}_c^j(y)$
    Set j to the child of j corresponding to f^j
    If (j is a leaf) {
      $\boldsymbol{l}^j(y)$++
      If ($\boldsymbol{l}^j$ has at least two non-zero entries
        AND ($N < T$ OR $C^j - \max_{i=1,2,\ldots,k} \boldsymbol{l}^j(i) > R_S(C^r + 1)$) {
        If ($N < T$) {**SetupNode** (LEFT(j)); **SetupNode** (RIGHT(j)); N++}
        Else {**Swap**(j)}
        $C^{\text{LEFT}(j)} = \lfloor C^j/2 \rfloor$; $C^{\text{RIGHT}(j)} = C^j - C^{\text{LEFT}(j)}$
        UpdateC (LEFT(j)) } }
    C^j++

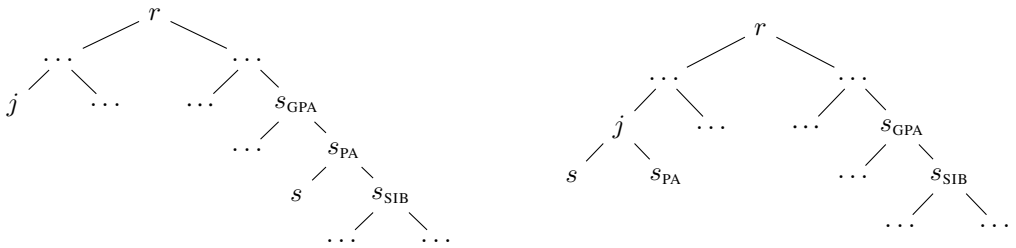

Figure 2: Illustration of the swapping procedure. **Left:** before the swap, **right:** after the swap.

The training phase, while the tree is being constructed and trained, is captured in Algorithm 1. The stopping criterion for expanding the tree is when the number of non-leaf nodes of the tree reaches the pre-specified threshold T.

3.1 Swapping

Consider a scenario where the current training example descends to leaf j. The leaf can split (create two children) if the examples that reached it in the past were coming from at least two different classes. However, if the number of non-leaf nodes of the tree reaches threshold T, no more nodes can be expanded and thus j cannot create children. Since the tree construction is done online, some nodes created at early stages of training may end up useless because no examples reach them later on. This prevents potentially useful splits such as at leaf j. This problem can be solved by recycling orphan nodes (subroutine **Swap** in Algorithm 1). The general idea behind node recycling is to allow nodes to split if a certain condition is met. In particular, node j splits if the following holds:

$$C^j - \max_{i=1,2,\ldots,k} \boldsymbol{l}^j(i) > R_S(C^r + 1), \tag{4}$$

where C^j is the size of the smallest leaf in the subtree with root j (r denotes the root of the entire tree), where the smallest leaf is the one with the smallest total number of data points that reached it in the past, $\boldsymbol{l}^j$ is a k-dimensional vector of non-negative integers where the i^{th} entrance is the count of the number of data points reaching leaf j in the past, and finally R_S is the pre-specified input parameter that we call 'swap resistance'. R_S should be set to at least 4. The subtraction of $\max_{i=1,2,\ldots,k} \boldsymbol{l}^j(i)$ in the above condition ensures that a pure node will not be recycled. It turns out that the condition captured in Inequality 4 allows us to prove (in Section 3.1) that the number of times a node is recycled is upper-bounded by the logarithm of the number of examples. If the condition is satisfied, the swap of the nodes is performed where an orphan leaf s, which was reached by the smallest number of examples in the past, and its parent s_{PA} are detached from the tree and become children of node j whereas the old sibling s_{SIB} of an orphan node s becomes a direct child of the old grandparent s_{GPA} of s. The swapping procedure is illustrated in Figure 2.

Lemma 3. *The number of times Algorithm 1 recycles a node is upper-bounded by the logarithm of the number of data points.*

Here we bound the number of swaps that any node makes. Consider $R_S = 4$. As before, let j be the node that is about to split and s be the orphan node that will be recycled (thus $C^r = C^s$). The condition in Equation 4 implies that the swap is done if $C^j > 4(C^r + 1) = 4(C^s + 1)$. Algorithm 1 makes s a child of j during the swap and sets its counter to $C^s_{new} = \left\lfloor C^j/2 \right\rfloor \geq 2(C^r + 1) = 2(C^s + 1)$. The algorithm then updates C^r. Since the value of C^s_{new} at least doubles after a swap and the maximum value of any counter is at most the number of examples, the node can be involved in at most $\log_2 n$ swaps where n is the number of examples.

4 Experiments

We conducted experiments on a variety of benchmark multiclass datasets: *Isolet*, *Sector*, *Aloi*, *ImageNet* (*ImNet*) and *ODP*[2]. The details of the datasets are provided in Table 1. The datasets were divided into training (90% and testing 10%). Furthermore, 10% of the training dataset was used as a validation set. We compared *LOMtree* with a balanced random tree of logarithmic depth (*Rtree*), *Filter tree* [14] and a one-against-all classifier (*OAA*). All methods were implemented in the open source

[2]The details of the source of each dataset are provided in the Supplementary material.

learning system Vowpal Wabbit [15] and have similar levels of optimization. The regressors in the tree nodes for *LOMtree*, *Rtree* and *Filter tree* as well as the *OAA* regressors were trained by online gradient descent for which we explored step sizes chosen from the set $\{0.25, 0.5, 0.75, 1, 2, 4, 8\}$. We used linear regressors. For each method we investigated training with up to 20 passes through the data and we selected the best setting of the parameters (step size and number of passes) as the one minimizing the holdout error. Additionally, for the *LOMtree* we investigated different settings of the stopping criterion for the tree expansion: $T = \{k-1, 2k-1, 4k-1, 8k-1, 16k-1, 32k-1, 64k-1\}$, and swap resistance $R_S = \{4, 8, 16, 32, 64, 128, 256\}$.

Table 1: Dataset sizes.

	Isolet	Sector	Aloi	ImNet	ODP
size	52.3MB	19MB	17.7MB	104GB[a]	3GB
# features	617	54K	128	6144	0.5M
# examples	7797	9619	108K	14.2M	1577418
# classes	26	105	1000	∼22K	∼105K

[a] compressed

Table 2: Test error (%) on multiclass classification problems.

	Isolet	Sector	Aloi	ImNet	ODP
LOMtree	**6.36**	16.19	**16.50**	**90.17**	**93.46**
Rtree	16.92	**15.77**	83.74	96.99	98.85
Filter tree	15.10	17.70	80.50	**92.12**	**93.76**
OAA	**3.56**	**9.17**	**13.78**	NA	NA

Table 3: Training time on multiclass classification problems.

	Isolet	Sector	Aloi
LOMtree	**16.27s**	**12.77s**	**51.86s**
OAA	19.58s	18.37s	11m2.43s

Table 4: Per-example test time on multiclass classification problems.

	Isolet	Sector	Aloi	ImNet	ODP
LOMtree	**0.14ms**	**0.13ms**	**0.06ms**	**0.52ms**	**0.26ms**
OAA	0.16 ms	0.24ms	0.33ms	0.21s	1.05s

We report test error (Table 2; top two best performers are in bold), train time (Table 3) and per-example test time (Table 4; the best performer is in bold in Table 3 and 4). Training time and test error are not reported for OAA on *ImageNet* and *ODP* due to the intractability of this method on such extreme multiclass classification problems. Note that, as expected, time-wise *LOMtree* significantly outperforms *OAA* due to building only close-to logarithmic depth trees. The improvement in the training time increases with the number of classes in the classification problem. For instance on *Aloi* training with *LOMtree* is 12.8 times faster than with OAA. The same can be said about the test time, where the per-example test time for *Aloi*, *ImageNet* and *ODP* are respectively 5.5, 403.8 and **4038.5** times faster for *LOMtree* than for *OAA*.

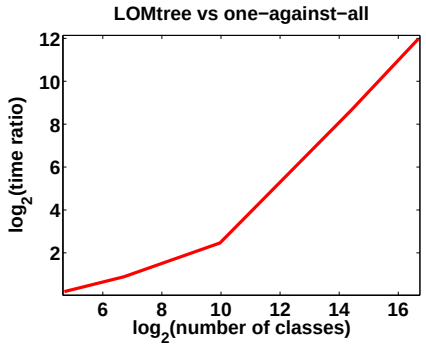

Figure 3: Logarithm of the ratio of per-example test time of *OAA* and *LOMtree* on multiclass classification problems.

Since the *Rtree* imposes a random label partition, the resulting error it obtains is generally worse than the test error obtained by the competitor methods and in particular *LOMtree* which instead learns the label partitioning directly from the data. At the same time *LOMtree* beats *Filter tree* on every dataset, though for *ImageNet* and *ODP* (both have a high level of noise) the advantage of *LOMtree* is not as significant as on the other datasets.

The time advantage of *LOMtree* comes with loss of statistical accuracy with respect to *OAA* where *OAA* is tractable. LOMTree significantly closes the gap between other logarithmic time methods and OAA, making log-time approaches attractive in large-k applications.

5 Conlusion and Future Work

The LOMTree algorithm reduces the multiclass problem to a set of binary problems organized in a tree structure where recovering the split in every tree node is done by optimizing a carefully crafted partition criterion. The criterion guarantees pure (inducing small error) and balanced splits leading to logarithmic training and testing time for the tree classifier. We provide theoretical justification via a boosting statement as well as an empirical evaluation on multiple multiclass datasets. Empirically, we find that this is the best available logarithmic time approach with performance approaching one-against all on smaller datasets where that is a viable baseline.

Acknowledgments

We would like to thank Alekh Agarwal, Dean Foster, Robert Schapire and Matus Telgarsky for valuable discussions.

References

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

[2] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 459–468. ACM Press, 1995.

[3] A. Beygelzimer, J. Langford, and P. D. Ravikumar. Error-correcting tournaments. In *ALT*, 2009.

[4] A. Beygelzimer, J. Langford, Y. Lifshits, G. B. Sorkin, and A. L. Strehl. Conditional probability tree estimation analysis and algorithms. In *UAI*, 2009.

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

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

[7] J. Weston, A. Makadia, and H. Yee. Label partitioning for sublinear ranking. In *ICML*, 2013.

[8] B. Zhao and E. P. Xing. Sparse output coding for large-scale visual recognition. In *CVPR*, 2013.

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

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

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

[12] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, and W.-Y. Ma. Support vector machines classification with a very large-scale taxonomy. In *SIGKDD Explorations*, 2005.

[13] P. N. Bennett and N. Nguyen. Refined experts: improving classification in large taxonomies. In *SIGIR*, 2009.

[14] A. Beygelzimer, J. Langford, and P. D. Ravikumar. Error-correcting tournaments. In *ALT*, 2009.

[15] J. Langford, L. Li, and A. Strehl. `http://hunch.net/~vw`, 2007.

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

[17] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer, 2004.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Logarithmic Time Online Multiclass prediction (Supplementary Material)

6 Bottom-up partitions do not work

Here we show that the most natural bottom-up construction for creating partitions is not viable with an example.

Bottom-up construction techniques start by pairing labels, either randomly or arbitrarily, and then building a predictor of whether the class label is left or right conditioned on the class label being one of the paired labels. In order to construct a full tree, this operation must compose, pairing trees with size 2 to create trees of size 4. Here, we show that the straightforward approach to composition fails.

Suppose we have a one dimensional feature space with examples of class label i having feature value i and we work with threshold predictors. Suppose we have 4 classes $1, 2, 3, 4$, and we happen to pair $(1, 3)$ and $(2, 4)$. It is easy to build a linear predictor for each of these splits. The next step is building a predictor for $(1, 3)$ vs $(2, 4)$ which is impossible because all thresholds in $(-\infty, 1)$, $(2, 3)$, and $(4, \infty)$ err on two labels while thresholds on $(1, 2)$ and $(3, 4)$ err on one label.

7 Proof of Lemma 1

We start from deriving an upper-bound on $J(h)$. Again for the ease of notation let $P_i = Pr(h(x) > 0|i)$. Thus

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h(x) > 0|i) - P(h(x) > 0)\right| = 2\sum_{i=1}^{k} \pi_i \left|P_i - \sum_{j=1}^{k} \pi_j P_j\right|,$$

where $\forall_{i=\{1,2,\ldots,k\}} 0 \leq P_i \leq 1$. Let $\alpha_i = \min(P_i, 1 - P_i)$ and recall the purity factor $\alpha = \sum_{i=1}^{k} \pi_i \alpha_i$ and the balancing factor $\beta = P(h(x) > 0)$. Without loss of generality let $\beta \leq \frac{1}{2}$. Furthermore, let

$$L_1 = \{i : i \in \{1, 2, \ldots, k\}, P_i \geq \frac{1}{2}\}, \quad L_2 = \{i : i \in \{1, 2, \ldots, k\}, P_i \in [\beta, \frac{1}{2})\}$$
$$\text{and} \quad L_3 = \{i : i \in \{1, 2, \ldots, k\}, P_i < \beta\}.$$

First notice that

$$\beta = \sum_{i=1}^{k} \pi_i P_i = \sum_{i \in L_1} \pi_i (1 - \alpha_i) + \sum_{i \in L_2 \cup L_3} \pi_i \alpha_i = \sum_{i \in L_1} \pi_i - 2\sum_{i \in L_1} \pi_i \alpha_i + \alpha \tag{5}$$

Therefore

$$\begin{aligned}
\frac{J(h)}{2} &= \sum_{i=1}^{k} \pi_i \left|P_i - \beta\right| = \sum_{i \in L_1} \pi_i (1 - \alpha_i - \beta) + \sum_{i \in L_2} \pi_i (\alpha_i - \beta) + \sum_{i \in L_3} \pi_i (\beta - \alpha_i) \\
&= \sum_{i \in L_1} \pi_i (1 - \beta) - \sum_{i \in L_1} \pi_i \alpha_i + \sum_{i \in L_2} \pi_i \alpha_i - \sum_{i \in L_2} \pi_i \beta + \sum_{i \in L_3} \pi_i \beta - \sum_{i \in L_3} \pi_i \alpha_i
\end{aligned}$$

Note that $\sum_{i \in L_3} \pi_i = 1 - \sum_{i \in L_1} \pi_i - \sum_{i \in L_2} \pi_i$ and therefore

$$\begin{aligned}
\frac{J(h)}{2} &= \sum_{i \in L_1} \pi_i (1 - \beta) - \sum_{i \in L_1} \pi_i \alpha_i + \sum_{i \in L_2} \pi_i \alpha_i - \sum_{i \in L_2} \pi_i \beta + \beta(1 - \sum_{i \in L_1} \pi_i - \sum_{i \in L_2} \pi_i) - \sum_{i \in L_3} \pi_i \alpha_i \\
&= \sum_{i \in L_1} \pi_i (1 - 2\beta) - \sum_{i \in L_1} \pi_i \alpha_i + \sum_{i \in L_2} \pi_i \alpha_i + \beta(1 - 2\sum_{i \in L_2} \pi_i) - \sum_{i \in L_3} \pi_i \alpha_i
\end{aligned}$$

Furthermore, since $-\sum_{i\in L_1}\pi_i\alpha_i+\sum_{i\in L_2}\pi_i\alpha_i-\sum_{i\in L_3}\pi_i\alpha_i=-\alpha+2\sum_{i\in L_2}\pi_i\alpha_i$ we further write that

$$\frac{J(h)}{2} = \sum_{i\in L_1}\pi_i(1-2\beta)+\beta(1-2\sum_{i\in L_2}\pi_i)-\alpha+2\sum_{i\in L_2}\pi_i\alpha_i$$

By Equation 5, it can be further rewritten as

$$\begin{aligned}\frac{J(h)}{2} &= (1-2\beta)(\beta+2\sum_{i\in L_1}\pi_i\alpha_i-\alpha)+\beta(1-2\sum_{i\in L_2}\pi_i)-\alpha+2\sum_{i\in L_2}\pi_i\alpha_i \\ &= 2(1-\beta)(\beta-\alpha)+2(1-2\beta)\sum_{i\in L_1}\pi_i\alpha_i+2\sum_{i\in L_2}\pi_i(\alpha_i-\beta)\end{aligned}$$

Since α_i's are bounded by 0.5 we obtain

$$\begin{aligned}\frac{J(h)}{2} &\leq 2(1-\beta)(\beta-\alpha)+2(1-2\beta)\sum_{i\in L_1}\pi_i\alpha_i+2\sum_{i\in L_2}\pi_i(\frac{1}{2}-\beta) \\ &\leq 2(1-\beta)(\beta-\alpha)+2(1-2\beta)\alpha+1-2\beta \\ &= 2\beta(1-\beta)-2\alpha(1-\beta)+2\alpha(1-2\beta)+1-2\beta \\ &= 1-2\beta^2-2\beta\alpha\end{aligned}$$

Thus:

$$\alpha\leq\frac{2-J(h)}{4\beta}-\beta.$$

8 Proof of Lemma 2

Proof. We first show that $J(h)\in[0,1]$. We start from deriving an upper-bound on $J(h)$, where $h\in\mathcal{H}$ is some hypothesis in the hypothesis class. For the ease of notation let $P_i=Pr(h(x)>0|i)$. Thus

$$\begin{aligned}J(h) &= 2\sum_{i=1}^{k}\pi_i\left|P(h(x)>0|i)-P(h(x)>0)\right| \\ &= 2\sum_{i=1}^{k}\pi_i\left|P_i-\sum_{j=1}^{k}\pi_jP_j\right|,\end{aligned} \tag{6}$$

where $\forall_{i=\{1,2,\ldots,k\}}0\leq P_i\leq 1$. The objective $J(h)$ is certainly maximized on the extremes of the $[0,1]$ interval. The upper-bound on $J(h)$ can be thus obtained by setting some of the P_i's to 1's and remaining ones to 0's. To be more precise, let

$$L_1=\{i:i\in\{1,2,\ldots,k\},P_i=1\} \quad \text{and} \quad L_2=\{i:i\in\{1,2,\ldots,k\},P_i=0\}.$$

Therefore it follows that

$$\begin{aligned}J(h) &\leq 2\left[\sum_{i\in L_1}\pi_i(1-\sum_{j\in L_1}\pi_j)+\sum_{i\in L_2}\pi_i\sum_{j\in L_1}\pi_j\right] \\ &= 2\left[\sum_{i\in L_1}\pi_i-(\sum_{i\in L_1}\pi_i)^2+(1-\sum_{i\in L_1}\pi_i)\sum_{i\in L_1}\pi_i\right] \\ &= 4\left[\sum_{i\in L_1}\pi_i-(\sum_{i\in L_1}\pi_i)^2\right]\end{aligned}$$

Let $b=\sum_{i\in L_1}\pi_i$ thus

$$J(h)\leq 4b(1-b)=-4b^2+4b \tag{7}$$

Since $b \in [0,1]$, it is straightforward that $-4b^2 + 4b \in [0,1]$ and thus $J(h) \in [0,1]$.

We now proceed to prove the main statement of Lemma 2, if h induces a maximally pure and balanced partition then $J(h) = 1$. Since h is maximally balanced, $P(h(x) > 0) = 0.5$. Simultaneously, since h is maximally pure $\forall_{i=\{1,2,\ldots,k\}}(P(h(x) > 0|i) = 0 \text{ or } P(h(x) > 0|i) = 1)$. Substituting that into Equation 7 yields that $J(h) = 1$. □

9 Proof of the lower-bound $\Delta_{\mathbf{t}} \geq \frac{\mathbf{J^2 G_t}}{\mathbf{8\beta(1-\beta)t \ln k}}$

Proof. First, without loss of generality assume that $P_1 \leq P_2 \leq \cdots \leq P_k$. For the ease of notation let $J = J(h_n)$. As observed in Kearns and Mansour [2], the entropy reduction of Equation 3 corresponds to a gap in the Jensen's inequality, applied to the concave function $G(\boldsymbol{\pi})$. Recall that Shannon entropy is strongly concave with respect to ℓ_1-norm (see e.g., Example 2.5 in Shalev-Shwartz [16]). As a specific consequence (see e.g. Theorem 2.1.9 in Nesterov [17]) we obtain

$$\begin{aligned}
\Delta_t &\geq w\beta(1-\beta)\|\boldsymbol{\pi}(n_0) - \boldsymbol{\pi}(n_1)\|_1^2 = w\beta(1-\beta)\left(\sum_{i=1}^{k}|\pi_i(n_0) - \pi_i(n_1)|\right)^2 \\
&= w\beta(1-\beta)\left(\sum_{i=1}^{k}\pi_i\left|\frac{P_i}{\beta} - \frac{1-P_i}{1-\beta}\right|\right)^2 = w\beta(1-\beta)\left(\sum_{i=1}^{k}\pi_i\left|\frac{P_i-\beta}{\beta(1-\beta)}\right|\right)^2 \\
&= \frac{w}{\beta(1-\beta)}\left(\sum_{i=1}^{k}|\pi_i(P_i-\beta)|\right)^2 = \frac{wJ^2}{4\beta(1-\beta)},
\end{aligned}$$

where the last equality uses the definition of $J(h) = 2\sum_{i=1}^{k}\pi_i|P_i - \beta|$.

Following the argument of Kearns and Mansour [2], notice that at round t there must be a leaf n such that $w_n \geq \frac{G_t}{2t\ln k}$ and this leaf is selected to the currently considered split (we split the leaf node with the highest weight), where $G_t = \sum_{n\in\mathcal{L}} w_n \sum_{i=1}^{k} -\pi_{ni}\ln(\pi_{ni})$. That is because

$$G_t = \sum_{n\in\mathcal{L}} w_n \sum_{i=1}^{k} -\pi_{ni}\ln(\pi_{ni}) \leq \sum_{n\in\mathcal{L}} w_n \ln k \leq 2tw_{max}\ln k,$$

where $w_{max} = \max_n w_n$. Thus $w_{max} \geq \frac{G_t}{2t\ln k}$. Thus

$$\Delta_t \geq \frac{J^2 G_t}{8\beta(1-\beta)t\ln k}.$$

□

10 Proof of Theorem 1

Proof. Note that the condition $\gamma \in (0, \min(\beta_n, 1-\beta_n)]$ implies that $\gamma \leq \frac{1}{2}$. From the *weak hypothesis assumption* it follows that for any n, β_n cannot be too near 0 or 1 since $1-\gamma \geq \beta_n \geq \gamma$. In Section 9 we showed that

$$\Delta_t \geq \frac{J^2 G_t}{8\beta(1-\beta)t\ln k}.$$

We now proceed to further lower-bound Δ_t. Note that the *weak hypothesis assumption* guarantees $J(h) \geq 2\gamma$, which further yields

$$\Delta_t \geq \frac{\gamma^2 G_t}{2(1-\gamma)^2 t\ln k}.$$

Let $\eta = \sqrt{\frac{8}{(1-\gamma)^2\ln k}}\gamma$. Then

$$\Delta_t > \frac{\eta^2 G_t}{16t}.$$

Thus we obtain the recurrence inequality

$$G_{t+1} \leq G_t - \Delta_t < G_t - \frac{\eta^2 G_t}{16t} = G_t \left[1 - \frac{\eta^2}{16t}\right]$$

One can now compute the minimum number of splits required to reduce G_t below α, where $\alpha \in [0, 1]$. We use the result from [2] (see the proof of Theorem 10) and obtain the final statement of the theorem. □

11 Experiments - dataset details

Below we provide the details of the datasets that we were using for the experiments in Section 4:

- *Isolet*: downloaded from `http://www.cs.huji.ac.il/~shais/datasets/ClassificationDatasets.html`
- *Sector* and *Aloi*: downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`
- *ImageNet* [18]: features extracted according to `http://www.di.ens.fr/willow/research/cnn/` from the authors.
- *ODP* [13]: obtained from Paul Bennett. Our version has signficantly more classes than reported in that paper because we use the entire dataaset.