Logarithmic Time One-Against-Some

Hal Daumé III [*1] **Nikos Karampatziakis** [*2] **John Langford** [*2] **Paul Mineiro** [*2]

Abstract

We create a new online reduction of multiclass classification to binary classification for which training and prediction time scale logarithmically with the number of classes. We show that several simple techniques give rise to an algorithm which is superior to previous logarithmic time classification approaches while competing with one-against-all in space. The core construction is based on using a tree to select a small subset of labels with high recall, which are then scored using a one-against-some structure with high precision.

1. Introduction

Can we efficiently predict which face is in the picture amongst multiple billions of people? In a translation, can we effectively predict which word should come next amongst 10^5 possibilities? More generally can we predict one of K classes in polylogarithmic time in K? This question gives rise to the area of extreme multiclass classification (Bengio et al., 2010; Beygelzimer et al., 2009; Bhatia et al., 2015; Choromanska & Langford, 2015; Morin & Bengio, 2005; Prabhu & Varma, 2014; Weston et al., 2013), in which K is very large. If efficiency is not a concern, the most common and generally effective representation for multiclass prediction is a one-against-all (OAA) structure. Here, inference consists of computing a score for each class and returning the class with the maximum score. If efficiency is a concern, an attractive strategy for picking one of K items is to use a tree; unfortunately, this often comes at the cost of increased error.

A general replacement for the one-against-all approach must satisfy a difficult set of desiderata.

- High accuracy: The approach should provide accuracy competitive with OAA, a remarkably strong baseline (Rifkin & Klautau, 2004) which is the standard "output layer" of many learning systems such as winners of the ImageNet contest (He et al., 2015; Simonyan & Zisserman, 2014).
- High speed at training time *and* test time: A multiclass classifier must spend at least $\Omega(\log K)$ time (Choromanska & Langford, 2015)) so this is a natural benchmark to optimize against.
- Online operation: Many learning algorithms use either online updates or mini-batch updates. Approaches satisfying this constraint can be easily composed into an end-to-end learning system for solving complex problems like image recognition. For algorithms which operate in batch fashion, online components can be easily used.
- Linear space: In order to have a drop-in replacement for OAA, an approach must not take much more space than OAA. Memory is at a premium when K is very large, especially for models trained on GPUs, or deployed to small devices.

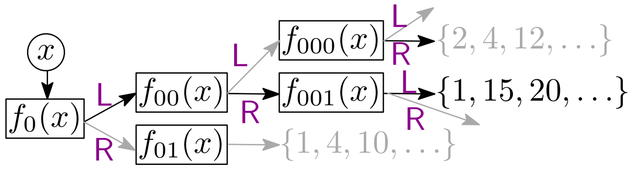


Figure 1. An example is routed through the tree to a leaf node associated with a set of eligible classes $\{1, 15, 20, \dots\}$, which will then be subsequently scored to pick a final label. The root classifier choses child L and classifier f_{00} chooses child R, etc.

We use an OAA-like structure to make a final prediction, but instead of scoring *every* class, we only score a small subset of $O(\log K)$ classes. We call this "one-against-some" (OAS). How can you efficiently determine what classes should be scored? We use a *dynamically* built tree to efficiently whittle down the set of candidate classes. The goal of the tree is to maximize the *recall* of the candidate set so we call this approach "The Recall Tree." In a traditional tree-based classifier, a traversal of the tree leads to a leaf, and a leaf corresponds to a single label, In the Recall Tree, we loosen the latter requirement and allow a leaf to corresponds to a *set* of labels of size $O(\log K)$. At test time, when a leaf is reached, scores are computed for this small subset (see Figure 1).

[*]Equal contribution [1]University of Maryland [2]Microsoft. Correspondence to: Paul Mineiro <pmineiro@microsoft.com>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

structure RecallTree		
score$_y$	$: X \to \mathbb{R}$	regressors for $y \in [K]$
tree	: BinaryTree	tree structure
structure BinaryTree		
id	$: \mathbb{N}$	unique node identifier
leaf	: bool	is this a leaf?
f	$: X \to \{\pm 1\}$	router, as binary classifier
left	: BinaryTree	left child (for non-leaves)
right	: BinaryTree	right child (for non-leaves)
hist	$: \mathbb{N}^K$	count of labels at this node
total	$: \mathbb{N}$	sum of items in hist
candidates	$\subset [K]$	top F freq items in hist

Figure 2. The Recall Tree data structure.

The Recall Tree achieves good accuracy, improving on previous online approaches (Choromanska & Langford, 2015) and sometimes surpassing the OAA baseline. The algorithm requires only $\mathrm{poly}(\log K)$ time during training and testing. In practice, the computational benefits are substantial when $K \geq 1000$.[1] The Recall Tree constructs a tree and learns parameters in a fully online manner as a reduction, allowing composition with systems trained via online updates. All of this requires only twice as much space as OAA approaches.

Our contributions are the following:

- We propose a new online tree construction algorithm which jointly optimizes the construction of the tree, the routers and the underlying OAS regressors (see section 3.1).
- We analyze elements of the algorithm, including a new boosting bound (see section 3.3) on multiclass classification performance and a representational trick which allows the algorithm to perform well if *either* a tree representation does well or an OAA representation does well as discussed in section 3.2.
- We experiment with the new algorithm, both to analyze its performance relative to baselines and understand the impact of design decisions via ablation experiments.

The net effect is a theoretically motivated algorithm which empirically performs well providing a plausible replacement for the standard one-against-all approach for large K.

2. The Recall Tree Algorithm

Here we present a concrete description of the Recall Tree and defer all theoretical results that motivate our design decisions to section 3.

[1]Our implementation of baseline approaches, including OAA, involve vectorized computations that increase throughput by a factor of 10 to 20, making them much more difficult to outpace than naïve implementations.

Algorithm 1 Predict. $n.f(x)$ evaluates the node's route, *score*$_y(x)$ evaluates a per-class regressor, $\widehat{\mathsf{recall}}(\circ)$ is an empirical bound on the recall of a node $(\circ)$ (see Eq (1)), and $x + \{(n.\mathrm{id} : 1)\}$ indicates the addition of a sparse feature with index n.id and value 1.

```
1: Input: Example $x$, Root Node $n$
2: Output: Predicted class $\hat{y}$
3: while $n$.leaf is false do
4:     $c \leftarrow n.f(x) > 0 \ ? \ n.\mathrm{left} : n.\mathrm{right}$
5:     if $\widehat{\mathsf{recall}}(n) > \widehat{\mathsf{recall}}(c)$ then
6:         break
7:     end if
8:     $n \leftarrow c$
9:     $x \leftarrow x + \{(n.\mathrm{id} : 1)\}$
10: end while
11: $\hat{y} \leftarrow \operatorname{argmax}_{y \in n.\mathrm{candidates}} score_y(x)$
```

2.1. Recall Tree at Test Time

The Recall Tree data structure (see Figure 2) consists of two components: (1) a binary tree, described below; and (2) a scoring function *score*$_y(x)$ that will evaluate the quality of a small set of candidates y to make a final prediction. Each node n in the binary tree maintains:

- a *router*, denoted f, that maps an example to either a left or right child; routers are implemented as binary classifiers;
- a histogram of the labels of all training examples that have been routed to, or through, n.

The primary purpose of the histogram is to generate a candidate set of labels to be scored, taken to be the most frequent labels in that histogram. Intuitively, the goal of the candidate set is to maintain good recall, while the goal of the *score* function is to achieve good precision. Crucially, the leaves of the tree do *not* partition the set of classes: classes can (and do) have support at multiple leaves.

At test time, an input x is provided and a recursive computation begins at the root of the tree. The tree is descended according to the binary classification decision made at each internal node. When the recursion ends (for instance, when a leaf is reached), the top F most frequent labels according to the node's label counter are used as a candidate set. When $F = O(\log K)$ this does not compromise the goal of achieving logarithmic time classification. Once this candidate set is chosen, each y in that set is scored using the *score* function, and the largest scoring y is returned.

It turns out that it is advantageous to allow the recursion to end *before* hitting a leaf, which is a consequence of how training happens on tree-structured classifiers. In particular, the number of labeled examples that the root classifier "sees" is much larger than the number of labeled examples that any leaf sees. This potentially leads to: (1) high variance toward the leaves; and (2) insufficient representation complexity toward the root. Instead of halting at a leaf, we can halt at an internal node for which the top F most frequent labels contain the true answer with a sufficiently high probability.

Algorithm 1 formalizes the test-time behavior of the Recall Tree. The primary routing occurs in the first line of the main loop, where c is the child selected by the current node's router. On the next line, the recursion considers the possibility of terminating on an internal node if the bounded recall, $\widehat{\mathsf{recall}}$, of the current node n is greater than the estimated recall of the chosen child c. If the recursion does not end, a new "path feature" is added to x at the end of the main loop, which records the path taken in the recall tree: the benefit of adding these features is that it increases the representational capacity of the recall tree to ensure competitiveness with OAA (§3.2). Whichever way the recursion ends, the final node n has a (small) set of candidate labels n.candidates $\subset Y$. Each is scored according to a one-against-some rule and the label with the largest score is returned.

A natural way to estimate recall at a node n is to consider it's empirical recall $\hat{r}_n$. This is simply the fraction of the mass consumed by the F most frequent labels in n's counter. For example, if the counter saw label 1 two times, label 4 fifty times and label 3 ten times, and if $F = 2$, then the empirical recall would be $60/62$. However, because, in general, a parent node will see more data than a child node, the quality of this estimate is likely to be much better for the parent than the child due to a missing mass problem (Good, 1953). To accomodate this, we instead use an empirical Bernstein lower bound (Maurer & Pontil, 2009), which is summarized by the following proposition.

Proposition 1. *For all multiclass classification problems defined by a distribution $\mathcal{D}$ over $X \times [K]$, and all nodes n in a fixed tree, there exists a constant $\lambda > 0$ such that with probability $1 - \delta$:*

$$\widehat{\mathsf{recall}}(n) = \hat{r}_n - \sqrt{\frac{\lambda \hat{r}_n (1 - \hat{r}_n)}{m_n}} - \frac{\lambda}{m_n} \leq r_n \quad (1)$$

where $\hat{r}_n$ is the empirical recall of node n computed over $m_n = n$.total items; and r_n is the expected value of this recall in the population limit.

Here, λ is a hyperparameter of the recall tree (in fact, it is the only additional hyperparameter), which controls how aggressively the tree branches. We show in our experiments that these various design decisions (path features, Bernstein lower bounds, and early termination) are useful in practice.

Algorithm 2 Train. An input labeled example descends the tree as in Algorithm 1. update_candidates updates the set of candidate labels at each node and update_regressors updates the one-against-some regressors; and $\widehat{\mathsf{recall}}(\circ)$ is a an empirical bound on the recall of a node $(\circ)$ (see section 3.1).

```
Input: Example $(x, y)$, Root node $n$
Output: Update tree with root at $n$
while n.leaf is false do
  update_router$(x, y, n)$
  $c \leftarrow n.f(x) > 0$ ? n.left : n.right
  update_candidates$(x, y, c)$
  if $\widehat{\mathsf{recall}}(n) > \widehat{\mathsf{recall}}(c)$ then
    break
  end if
  $n \leftarrow c$
  $x \leftarrow x + \{(n.\text{id} : 1)\}$
end while
update_regressors$(x, y, n.\text{candidates})$
```

2.2. Recall Tree at Training Time

The Recall Tree maintains one regressor for each class and a tree whose purpose is to eliminate regressor from consideration. We refer to the per-class regressor as one-against-some (OAS) regressors. The tree creates a high recall set of candidate classes and then leverages the OAS regressors to achieve precision. Algorithm 2 outlines the learning procedures, which we now describe in more detail.

Learning the regressors for each class In Algorithm 2, update_regressors updates the candidate set regressors using the standard OAA strategy restricted to the set of eligible classes. If the true label is not in the F most frequent classes at this node then no update occurs.

Learning the set of candidates in each node In Algorithm 2, update_candidates updates the count of the true label at this node. At each node, the most frequent F labels are the candidate set.

Learning the routers at each node In Algorithm 2, update_router updates the router at a node by optimizing the reduction in the entropy of the label distribution (the label entropy) due to routing, as detailed in Algorithm 3. This is in accordance with our theory (Section 3.3). The label entropy for a node is estimated using the empirical counts of each class label entering the node. These counts are reliable as update_router is only called for the root or nodes whose true recall bound is better than their children. The expected label entropy after routing is estimated by averaging the estimated label entropy of each child node, weighted by the fraction of examples routing left or right. Finally, we compute the advantage of routing left vs. right

Algorithm 3 update_router. entropy computes two values: the empirical entropy of labels incident on a node without and with (respectively) an extra label y. $\hat{H}_{|\text{left}}$ is an estimate of the average entropy if the example is routed left. $\text{Learn}_n(x, w, y)$ is an importance-weighted update to the binary classifier $f(x)$ for node n with features x, label y, and weight w.

```
**Input:** Example $(x, y)$; Node $n$
**Output:** Update node $n$
$(\hat{H}_{\text{left}}, \hat{H}'_{\text{left}}) \doteq \mathsf{entropy}(n.\text{left}, y)$
$(\hat{H}_{\text{right}}, \hat{H}'_{\text{right}}) \doteq \mathsf{entropy}(n.\text{right}, y)$
$\hat{H}_{|\text{left}} \doteq \frac{n.\text{left.total}}{n.\text{total}} \hat{H}'_{\text{left}} + \frac{n.\text{right.total}}{n.\text{total}} \hat{H}_{\text{right}}$
$\hat{H}_{|\text{right}} \doteq \frac{n.\text{left.total}}{n.\text{total}} \hat{H}_{\text{left}} + \frac{n.\text{right.total}}{n.\text{total}} \hat{H}'_{\text{right}}$
$\widehat{\Delta H}_{\text{post}} \leftarrow \hat{H}_{|\text{left}} - \hat{H}_{|\text{right}}$
$\text{Learn}_n(x, |\widehat{\Delta H}_{\text{post}}|, \text{sign}(\widehat{\Delta H}_{\text{post}}))$
```

Algorithm 4 update_regressors updates the OAS scoring functions for a single example.

```
**Input:** Example $(x, y)$; Candidate set candidates
**Output:** Update scoring functions *score*
**if** $y \in$ candidates **then**
    online update to $score_y(x)$ with label $+1$
    **for** $\hat{y} \in$ candidates $- \{y\}$ **do**
        online update to $score_{\hat{y}}(x)$ with label $-1$
    **end for**
**end if**
```

by taking the difference of the expected label entropies for routing left vs. right. The sign of this difference determines the binary label for updating the router.

Tree depth control We calculate a lower bound $\widehat{\mathsf{recall}}(n)$ on the true recall of node n (Section 3.1), halting descent as in Algorithm 2. As we descend the tree, the bound first increases (empirical recall increases) then declines (variance increases). We also limit the maximum depth d of the tree. This parameter is typically not operative but adds an additional safety check and sees some use on datasets where multipasses are employed.

3. Theoretical Motivation

Online construction of an optimal logarithmic time regressors for multiclass classification given an arbitrary fixed representation at each node appears deeply intractable. A primary difficulty is that decisions have to be *hard* since we cannot afford to maintain a distribution over all class labels. Choosing a classifier so as to minimize error rate has been considered for cryptographic primitives (Blum et al., 1993) so it is plausibly hard on average rather than merely hard in the worst case. Furthermore, the *joint* optimization of all regressors does not nicely decompose into independent problems. Solving the above problems requires an implausible break-through in complexity theory which we do not achieve here. Instead, we use learning theory to assist the design by analyzing various simplifications of the problem.

3.1. One-Against-Some Recall

For binary classification, a simple trick can (in theory) collapse the number of leaves while preserving prediction performance. In particular, branching programs (Mansour & McAllester, 2002) result in exponentially more succinct representations than decision trees (Kearns & Mansour, 1996) by joining nodes to create directed acyclic graphs. The key observation is that nodes in the same level with a similar distribution over class labels can be joined into one node, implying that the number of nodes at one level is only $\theta(1/\gamma)$ where γ is the weak learning parameter rather than exponential in the depth. This approach generally fails in the multiclass setting because covering the simplex of multiclass label distributions requires $(K-1)^{\theta(1/\gamma)}$ nodes.

One easy special case exists. When the distribution over class labels is skewed so one label is the majority class, learning an entropy minimizing binary classifier predicts whether the class is the majority or not. There are only K possible OAS regressors of this sort so maintaining one for each class label is computationally tractable.

Using OAS classifiers creates a limited branching program structure over predictions. Aside from the space savings generated, this also implies that nodes deep in the tree use many more labeled examples than are otherwise available. In finite sample regimes, which are not covered by these boosting analyses, more labeled samples induce a better predictor as per standard sample complexity analysis.

As a result, we use the empirical Bernstein lower bound on recall described in §2.1. Reducing the depth of the tree by using this lower bound and joining labeled examples from many leaves in a one-against-some approach both relieves data sparsity problems and allows greater error tolerance by the root node.

3.2. Path Features

Different multiclass classification schemes give rise to different multiclass hypothesis classes. For example, the set of multiclass decision boundaries realizable under an OAA structure over linear regressors is fundamentally different from that realizable under a tree structure over linear regressors. Are OAA types of representations inherently more or less powerful than a tree based representation? Figure 3 shows two learning problems illustrating two extremes assuming a linear representation.

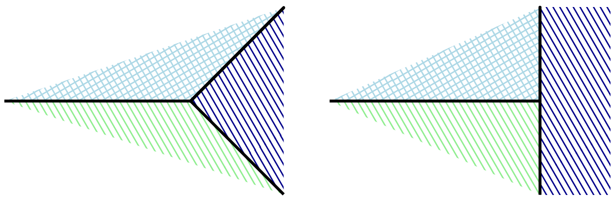


Figure 3. Two different distributions over class labels in the plane with each color/pattern representing support for a single class label. The left distribution is easily solved with an OAA classifier while the right distribution is easily solved with a decision tree.

Linear OAA: If all the class parameter vectors happen to have the same ℓ_2 norm, then OAA classification is equivalent to finding the nearest neighbor amongst a set of vectors (one per class) which partition the space into a Voronoi diagram as in 3 on the left. The general case, with unequal vectors corresponds to a weighted Voronoi diagram where the magnitude of two vectors sharing a border determines the edge of the partition. No weighted Voronoi diagram can account for the partition on the right.

Trees: If the partition of a space can be represented by a sequence of conditional splits, then a tree can represent the solution accurately as in 3 on the right. On the other hand, extra work is generally required to represent a Voronoi diagram as on the left. In general, the number of edges in a multidimensional Voronoi diagram may grow at least quadratically in the number of points implying that the number of nodes required for a tree to faithfully represent a Voronoi diagram is at least $\Theta(n^2)$.

Based on this, neither tree-based nor OAA style prediction is inherently more powerful, with the best solution being problem dependent.

Since we are interested in starting with a tree-based approach and ending with a OAS classifier, there is a simple representational trick which provides the best of both worlds. We can add features which record the path through the tree. To be precise, let T be a tree and $\text{path}_T(x)$ be a vector with one dimension per node in T which is set to 1 if x traverses the node and 0 otherwise. The following proposition holds for linear representations, which are special because they are tractably analyzed and because they are the fundamental building blocks around which many more complex representations are built.

Proposition. *For any distribution D over $X \times [K]$ for which a tree T achieves error rate ϵ, a OAA classifier over linear regressors, whose input consists of $x \in X$* and *the corresponding routing path of x in T (as indicator features) can also achieve error rate ϵ.*

Proof. A linear OAA classifier is defined by a matrix w_{iy} where i ranges over the input and y ranges over the labels. Let $w_{iy} = 0$ by default and 1 when i corresponds to a leaf for which the tree predicts y. Under this representation, the prediction of $\text{OAA}(x, \text{path}_T(x))$ is identical to $T(x)$, and hence achieves the same error rate. $\square$

3.3. Optimization Objective

The Shannon Entropy of class labels is optimized in the router of Algorithm 3. Why?

Since the Recall Tree jointly optimizes over many base learning algorithms, the systemic properties of the joint optimization are important to consider. A theory of decision tree learning as boosting (Kearns & Mansour, 1996) provides a way to understand these joint properties in a population limit (or equivalently on a training set iterated until convergence). In essence, the analysis shows each level of the tree boosts the accuracy of the resulting tree with this conclusion holding for several common objectives.

In boosting for multiclass classification (Choromanska et al., 2016; Choromanska & Langford, 2015; Takimoto & Maruoka, 2003), it is important to achieve a weak dependence on the number of class labels. Shannon Entropy is particularly well-suited to this goal, because it has only a logarithmic dependence on the number of class labels. Let $\pi_{i|n}$ be the probability that the correct label is i, conditioned on the corresponding example reaching node n. Then $H_n = \sum_{i=1}^{K} \pi_{i|n} \log_2 \frac{1}{\pi_{i|n}}$ is the Shannon entropy of class labels reaching node n.

For this section, we consider a simplified algorithm which neglects concerns of finite sample analysis, how optimization is done, and the leaf predictors. What's left is the value of optimizing the router objective. We consider an algorithm which recursively splits the leaf with the largest proportion p of all examples starting at the root and reaching the leaf. The leaf is split into two new leaves to the left l and right r. If p_l and p_r are the fraction of examples going left and right (so $p_l + p_r = 1$), the split criterion minimizes the expectation over the leaves of the average class entropy, $p_l H_l + p_r H_r$. This might be achieved by `update_router` in Algorithm 2 or by any other means. With this criterion we are in a position to directly optimize information boosting.

Definition 1. (γ-Weak Learning Assumption) For all distributions $D(x, y)$ a learning algorithm using examples $(x, y)^*$ IID from D finds a binary classifier $c : X \to \{l, r\}$ satisfying

$$p_l H_l + p_r H_r \leq H_n - \gamma \quad .$$

This approach is similar to previous (Takimoto & Maruoka, 2003) except that we boost in an *additive* rather than a *multiplicative* sense. A multiplicative approach suppresses a necessary dependence on K. In particular, for any nontrivial γ there exists a K such that with a uniform distribution U, $H_U(1 - \gamma) > 1$. As a consequence, theorems proved

Table 1. Datasets used for experimentation.

Dataset	Source	Task	Classes	Examples
ALOI	Geusebroek et al. (2005)	Visual Object Recognition	$1k$	10^5
Imagenet	Oquab et al. (2014)	Visual Object Recognition	$\approx 20k$	$\approx 10^7$
LTCB	Mahoney (2009)	Language Modeling	$\approx 80k$	$\approx 10^8$
ODP	Bennett & Nguyen (2009)	Document Classification	$\approx 100k$	$\approx 10^6$

with a multiplicative γ are necessarily vacuous for large K while additive approaches do not suffer from this issue.

As long as Weak Learning occurs, we can prove the following theorem.

Theorem 2. *If γ Weak Learning holds for every node in the tree and nodes with the largest fraction of examples are split first, then after $t > 2$ splits the multiclass error rate ϵ of the tree is bounded by:*

$$\epsilon \leq H_1 - \gamma \ln(t + 1)$$

where H_1 is the entropy of the marginal distribution of class labels.

The proof in appendix A reuses techniques from (Choromanska & Langford, 2015; Kearns & Mansour, 1996) but has a tighter result.

The most important observation from the theorem is that as t (the number of splits) increases, the error rate is increasingly bounded. This rate depends on $\ln t$ agreeing with the intuition that boosting happens level by level in the tree. The dependence on the initial entropy H_1 shows that skewed marginal class distributions are inherently easier to learn than uniform marginal class distributions, as might be expected. These results are similar to previous results (Choromanska et al., 2016; Choromanska & Langford, 2015; Kearns & Mansour, 1996; Takimoto & Maruoka, 2003) with advantages. We handle multiclass rather than binary classification (Kearns & Mansour, 1996), we bound error rates instead of entropy (Choromanska et al., 2016; Choromanska & Langford, 2015), and we use additive rather than multiplicative weak learning (Takimoto & Maruoka, 2003).

4. Empirical Results

We study several questions empirically.

1. What is the benefit of using one-against-some on a recall set?
2. What is the benefit of path features?
3. Is the online nature of the Recall Tree useful on nonstationary problems?
4. How does the Recall Tree compare to one-against-all statistically and computationally?
5. How does the Recall Tree compare to LOMTree statistically and computationally?

Throughout this section we conduct experiments using learning with a linear representation.

4.1. Datasets

Table 1 overviews the data sets used for experimentation. These include the largest datasets where published results are available for LOMTree (Aloi, Imagenet, ODP), plus an additional language modeling data set (LTCB). Implementations of the learning algorithms, and scripts to reproduce the data sets and experimental results, are available on github (Mineiro, 2017). Additional details about the datasets can be found in Appendix B.

4.2. Comparison with other Algorithms

In our first set of experiments, we compare Recall Tree with a strong computational baseline and a strong statistical baseline. The computational baseline is LOMTree, the only other online logarithmic-time multiclass algorithm of which we are aware. The statistical baseline is OAA, whose statistical performance we want to match (or even exceed), and whose linear computational dependence on the number of classes we want to avoid. Details regarding the experimental methodology are in Appendix C. Results are summarized in Figure 4.

Comparison with LOMTree The Recall Tree uses a factor of 32 less state than the LOMTree which makes a dramatic difference in feasibility for large scale applications. Given this state reduction, the default expectation is worse prediction performance by the Recall Tree. Instead, we observe superior or onpar statistical performance despite the state constraint. This typically comes with an additional computational cost since the Recall Tree evaluates a number of per-class regressors.

Comparison with OAA On one dataset (ALOI) prediction performance is superior to OAA while on the others it is somewhat worse.

Computationally OAA has favorable constant factors since it is highly amenable to vectorization. Conversely, the

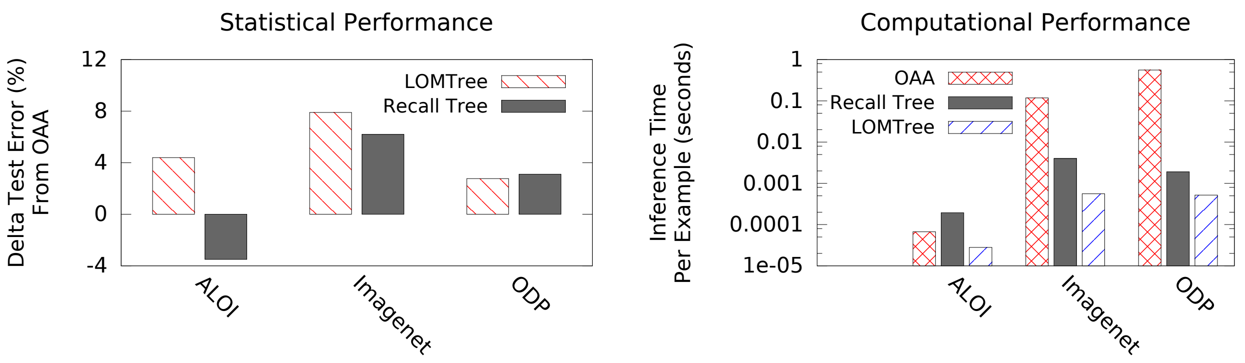


Figure 4. Empirical comparison of statistical (left) and computational (right) performance of Recall Tree against two strong competitors: OAA (statistically good) and LOMTree (computationally good). In both graphs, lower is better. Recall Tree has poly(log) dependence upon number of classes (like LOMTree) but can surpass OAA statistically.

conditional execution pattern of the Recall Tree frustrates vectorization even with example mini-batching. Thus on ALOI although Recall Tree does on average 50 hyperplane evaluations per example while OAA does 1000, OAA is actually faster: larger numbers of classes are required to experience the asymptotic benefits. For ODP with $\sim 10^5$ classes, with negative gradient subsampling and using 24 cores in parallel, OAA is about the same wall clock time to train as Recall Tree on a single core.[2] Negative gradient sampling does not improve inference times, which are ~ 300 times slower for OAA than Recall Tree on ODP.

4.3. Online Operation

In this experiment we leverage the online nature of the algorithm to exploit nonstationarity in the data to improve results. This is not something that is easily done with batch oriented algorithms, or with algorithms that post-process a trained predictor to accelerate inference.

We consider two versions of LTCB. In both versions the task is to predict the next word given the previous 6 tokens. The difference is that in one version, the Wikipedia dump is processed in the original order ("in-order"); whereas in the other version the training data is permuted prior to input to the learning algorithm ("permuted"). We assess progressive validation loss (Blum et al., 1999) on the sequence. The result in Figure 5a confirms the Recall Tree is able to take advantage of the sequentially revealed data; in particular, the far-right difference in accuracies is significant at a factor $P < 0.0001$ according to an $N-1$ Chi-squared test.

4.4. Path Features and Multiple Regressors

Two differences between Recall Tree and LOMTree are the use of multiple regressors at each tree node and the augmentation of the example with path features. In this experiment we explore the impact of these design choices using the ALOI dataset.

Figure 5b shows the effect of these two aspects on statistical performance. As the candidate set size is increased, test error decreases, but with diminishing returns. Disabling path features degrades performance, and the effect is more pronounced as the candidate set size increases. This is expected, as a larger candidate set size decreases the difficulty of obtaining good recall (i.e., a good tree) but increases the difficulty of obtaining good precision (i.e., good class regressors), and path features are only applicable to the latter. All differences here are significant at a $P < 0.0001$ according to an $N-1$ Chi-squared test, except for when the candidate set size is 2, where there is no significant difference.

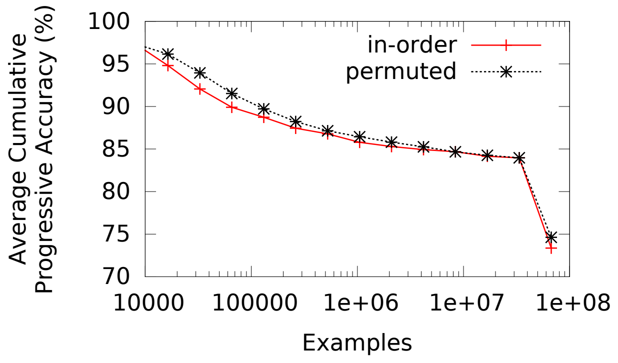
4.5. Is the empirical Bernstein bound useful?

To test this we trained on the LTCB dataset with a multiplier on the bound of either 0 (i.e. just using empirical recall directly) or 1. The results are stark: with a multiplier of 1, the test error was 78% while with a multiplier of 0 the test error was 91%. Clearly, in the few samples per class regime this form of direct regularization is very helpful.

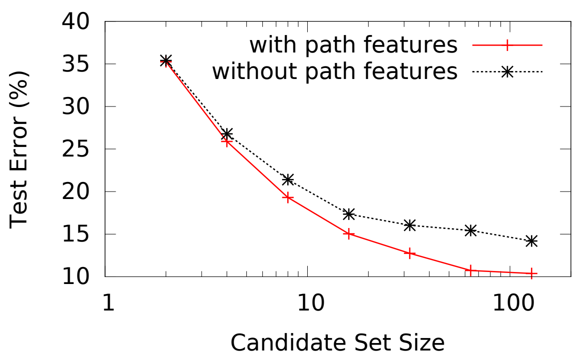
5. Related Work

The LOMTree (Choromanska et al., 2016; Choromanska & Langford, 2015) is the closest prior work. It misses on space requirements: up to a factor of 64 more space than OAA was used experimentally. Despite working with radically less space we show the Recall Tree typically provides better predictive performance. The key differences here are algorithmic: a tighter reduction at internal nodes and the one-against-some approach yields generally better performance despite much tighter resource constraints.

[2]While not yet implemented, Recall Tree can presumably also leverage multicore for acceleration.



(a) When the LTCB dataset is presented in the original order, Recall Tree is able to exploit sequential correlations for improved performance. After all examples are processed, the average progressive accuracy is 73.3% vs. 74.6%.



(b) Test error on ALOI for various candidate set sizes, with or without path features (all other parameters held fixed). Using multiple regressors per leaf and including path features improves performance.

Figure 5.

Our use of entropy optimization is closely related to the foundational work on decision tree learning (Quinlan, 1993), picking single features on which to split based on entropy. More recently, it is decision tree learning can be thought of *as* boosting (Kearns & Mansour, 1996) for multiclass learning (Takimoto & Maruoka, 2003), based on on a generalized notion of entropy, which results in low 0/1 loss. Relative to these works we show *how* to efficiently achieve weak learning by reduction to binary classification making this approach empirically practical. We also address a structural issue in the multiclass analysis (see section 3.3).

Other approaches such as hierarchical softmax (HSM) and the the Filter Tree (Beygelzimer et al., 2009) use a fixed tree structure (Morin & Bengio, 2005). In domains in which there is no prespecified tree hierarchy, using a random tree structure can lead to considerable underperformance as shown previously (Bengio et al., 2010; Choromanska & Langford, 2015).

Most other approaches in extreme classification either do not work online (Mnih & Hinton, 2009; Prabhu & Varma, 2014) or only focus on speeding up either prediction time or training time but not both. Most of the works that enjoy sublinear inference time (but (super)linear training time) are based on tree decomposition approaches. In (Mnih & Hinton, 2009) the authors try to add tree structure learning to HSM via iteratively clustering the classes. While the end result is a classifier whose inference time scales logarithmically with the number of classes, the clustering steps are batch and scale poorly with the number of classes. Similar remarks apply to (Bengio et al., 2010) where the authors propose to learn a tree by solving an eigenvalue problem after (OAA) training. The work of (Weston et al., 2013) is similar in spirit to ours, as the authors propose to learn a label filter to reduce the number of candidate classes in an OAA approach. However they learn the tree after training the underlying OAA regressors while here we learn, and more crucially use, the tree during training of the OAS regressors. Among the approaches that speed up training time we distinguish exact ones (de Brébisson & Vincent, 2015; Vincent et al., 2015) that have only been proposed for particular loss functions and approximate ones such as negative sampling as used e.g. in (Weston et al., 2011). Though these techniques do not address inference time, separate procedures for speeding up inference (given a trained model) have been proposed (Shrivastava & Li, 2014). However, such two step procedures can lead to substantially suboptimal results.

6. Conclusion

In this work we proposed the Recall Tree, a reduction of multiclass to binary classification, which operates online and scales logarithmically with the number of classes. Unlike the LOMTree (Choromanska & Langford, 2015), we share classifiers among the nodes of the tree which alleviates data sparsity at deep levels while greatly reducing the required state. We also use a tighter analysis which is more closely followed in the implementation. These features allow us to reduce the statistical gap with OAA while still operating many orders of magnitude faster for large K multiclass datasets. In the future we plan to investigate multiway splits in the tree since $O(\log K)$-way splits does not affect our $O(\text{poly} \log K)$ running time and they might reduce contention in the root and nodes high in the tree.

References

Bengio, Samy, Weston, Jason, and Grangier, David. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, pp. 163–171, 2010.

Bennett, Paul N and Nguyen, Nam. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 11–18. ACM, 2009.

Beygelzimer, Alina, Langford, John, and Ravikumar, Pradeep. Error-correcting tournaments. In *Algorithmic Learning Theory, 20th International Conference, ALT 2009, Porto, Portugal, October 3-5, 2009. Proceedings*, pp. 247–262, 2009.

Bhatia, Kush, Jain, Himanshu, Kar, Purushottam, Varma, Manik, and Jain, Prateek. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pp. 730–738, 2015.

Blum, Avrim, Furst, Merrick, Kearns, Michael, and Lipton, Richard J. Cryptographic primitives based on hard learning problems. In *Advances in cryptologyCRYPTO93*, pp. 278–291. Springer, 1993.

Blum, Avrim, Kalai, Adam, and Langford, John. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory, COLT 1999, Santa Cruz, CA, USA, July 7-9, 1999*, pp. 203–208, 1999.

Choromanska, Anna, Choromanski, Krzysztof, and Bojarski, Mariusz. On the boosting ability of top-down decision tree learning algorithm for multiclass classification. *CoRR*, abs/1605.05223, 2016. URL http://arxiv.org/abs/1605.05223.

Choromanska, Anna E and Langford, John. Logarithmic time online multiclass prediction. In *Advances in Neural Information Processing Systems*, pp. 55–63, 2015.

de Brébisson, Alexandre and Vincent, Pascal. An exploration of softmax alternatives belonging to the spherical loss family. *arXiv preprint arXiv:1511.05042*, 2015.

Geusebroek, Jan-Mark, Burghouts, Gertjan J, and Smeulders, Arnold WM. The Amsterdam library of object images. *International Journal of Computer Vision*, 61(1): 103–112, 2005.

Good, I. J. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40 (16):237–264, 1953.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

Kearns, Michael and Mansour, Yishay. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of STOC*, pp. 459–468. ACM, 1996.

Mahoney, Matt. Large text compression benchmark. *http://www.mattmahoney.net/text/text.html*, 2009.

Mansour, Yishay and McAllester, David. Boosting using branching programs. *Journal of Computer and System Sciences*, 64(1):103–112, 2002.

Maurer, Andreas and Pontil, Massimiliano. Empirical bernstein bounds and sample variance penalization. *arXiv preprint arXiv:0907.3740*, 2009.

Mineiro, Paul. Recall tree demo, 2017. URL https://github.com/JohnLangford/vowpal_wabbit/tree/master/demo/recall_tree.

Mnih, Andriy and Hinton, Geoffrey E. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pp. 1081–1088, 2009.

Morin, Frederic and Bengio, Yoshua. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pp. 246–252, 2005.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014.

Prabhu, Yashoteja and Varma, Manik. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD*, pp. 263–272. ACM, 2014.

Quinlan, J.R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

Rifkin, Ryan and Klautau, Aldebaro. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

Shrivastava, Anshumali and Li, Ping. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pp. 2321–2329, 2014.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

Takimoto, Eiji and Maruoka, Akira. Top-down decision tree learning as information based boosting. *Theor. Comput. Sci.*, 292(2):447–464, 2003. doi: 10.1016/S0304-3975(02)00181-0. URL http://dx.doi.org/10.1016/S0304-3975(02)00181-0.

Vincent, Pascal, de Brébisson, Alexandre, and Bouthillier, Xavier. Efficient exact gradient update for training deep networks with very large sparse targets. In *NIPS 28*, pp. 1108–1116, 2015.

Weston, Jason, Bengio, Samy, and Usunier, Nicolas. WSABIE: scaling up to large vocabulary image annotation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2764–2770, 2011.

Weston, Jason, Makadia, Ameesh, and Yee, Hector. Label partitioning for sublinear ranking. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 181–189, 2013.