Machine Learning 27(1):51-68, 1997.

Predicting nearly as well as the best pruning of a decision tree

DAVID P. HELMBOLD dph@cse.ucsc.edu
Computer and Information Sciences, University of California, Santa Cruz, CA 95064

ROBERT E. SCHAPIRE schapire@research.att.com
AT&T Labs, 600 Mountain Avenue, Murray Hill, NJ 07974

Abstract. Many algorithms for inferring a decision tree from data involve a two-phase process: First, a very large decision tree is grown which typically ends up "over-fitting" the data. To reduce over-fitting, in the second phase, the tree is pruned using one of a number of available methods. The final tree is then output and used for classification on test data.

In this paper, we suggest an alternative approach to the pruning phase. Using a given unpruned decision tree, we present a new method of making predictions on test data, and we prove that our algorithm's performance will not be "much worse" (in a precise technical sense) than the predictions made by the best reasonably small pruning of the given decision tree. Thus, our procedure is guaranteed to be competitive (in terms of the quality of its predictions) with *any* pruning algorithm. We prove that our procedure is very efficient and highly robust.

Our method can be viewed as a synthesis of two previously studied techniques. First, we apply Cesa-Bianchi et al.'s [4] results on predicting using "expert advice" (where we view each pruning as an "expert") to obtain an algorithm that has provably low prediction loss, but that is computationally infeasible. Next, we generalize and apply a method developed by Buntine [3], [2] and Willems, Shtarkov and Tjalkens [20], [21] to derive a very efficient implementation of this procedure.

1. Introduction

Many algorithms for inferring a decision tree from data, such as C4.5 [13], involve a two step process: In the first step, a very large decision tree is grown to match the data. If the training data contains noise then this large tree typically "over-fits" the data, giving quite poor performance on the test set. Therefore, in the second phase, the tree is pruned using one of a number of available methods. The final tree is then output and used for classification on test data.

In this paper, we suggest an alternative approach to the pruning phase. Using a given unpruned decision tree $\mathcal{T}$, we present a new method of making predictions on test data, and we prove that our algorithm's performance will not be "much worse" (in a precise technical sense) than the predictions made by the best reasonably small pruning of the given decision tree. More precisely, we define a value metric based on the inaccuracy and size of the tree. Our algorithm's performance is comparable to the performance of the pruning with the highest value. Thus, our procedure is guaranteed to be competitive (in terms of the quality of its predictions) with *any* pruning algorithm.

Formally, we study this problem in the on-line learning framework introduced by Littlestone [9] and extended by Littlestone and Warmuth [10] and others. In this model, at each time step $t = 1, \ldots, T$, the learner receives an instance x^t and must generate a prediction $\hat{y}^t \in [0, 1]$. After an outcome $y^t \in \{0, 1\}$ is observed (which can be thought of as the label or correct classification of the instance x^t), the learner suffers loss $|y^t - \hat{y}^t|$. Note that $\hat{y}^t$ can be interpreted as the bias of a binary prediction which is 1 with probability $\hat{y}^t$, and 0 with probability $1-\hat{y}^t$. Then the loss $|y^t - \hat{y}^t|$ is simply the probability of the learner making a mistake (i.e., a prediction differing from the outcome y^t). The tools developed for this framework make it possible to prove very strong bounds on the performance of our algorithm.

The learner computes its predictions using predictions $\xi^t_{\mathcal{P}}$ that are generated in a natural way by each pruning $\mathcal{P}$ of the given unpruned tree $\mathcal{T}$. We first show how an algorithm developed and analyzed by Cesa-Bianchi et al. [4] can be applied immediately to obtain a learning algorithm whose loss is bounded by a function that, for *any* pruning $\mathcal{P}$, is linear in the prediction loss of $\mathcal{P}$ and the size of $\mathcal{P}$ (roughly, the number of nodes in the pruning). Their algorithm is closely related to work by Vovk [16] and Littlestone and Warmuth [10]. Note that this is a "worst-case" analysis in the sense that it does not rely on statistical assumptions of any kind regarding the source of the data that is being observed. Thus, the resulting algorithm is very robust.

A naive implementation of this procedure would require computation time linear in the number of prunings of $\mathcal{T}$; obviously, this is infeasible. However, we show how techniques used by Buntine [3], [2] and Willems, Shtarkov and Tjalkens [20], [21] can be generalized and applied to our setting, yielding a very efficient implementation requiring computation time at each trial t that is linear in the length of the path defined by the instance x^t in the tree $\mathcal{T}$ (and therefore is bounded by the depth of $\mathcal{T}$).

Various authors have presented techniques for averaging a family of decision trees [6], [8], [11]. In particular, using a Bayesian formulation, Buntine [3], [2] gave a method called Bayesian smoothing for averaging the class-probability predictions of all possible prunings of a given decision tree. Although our method is very similar to Buntine's, his is designed for use on a batch of examples, while ours uses efficient incremental updates of the data structure in an on-line setting.

Willems, Shtarkov and Tjalkens [20], [21] presented their technique in a much narrower context in which the decision trees considered were assumed to have a very particular form, and the goal was data compression rather than prediction.

A primary contribution of the current paper is the distillation of key elements of these previously known methods, and synthesis with other learning-theory results leading to broader learning applications.

In independent work, Oliver and Hand [12] have experimented with averaging over different prunings of decision trees. Their results show that in some cases averaging outperforms the prunings generated by C4.5. Oliver and Hand weight the prunings by their performance on the training set, while our methods provide an efficient way to update the weights as new data is seen. In addition to prunings of the full decision tree, Oliver and Hand also included subtrees resulting from those splits that were considered but rejected when the decision tree was grown. This can be modeled in our setting by storing multiple prediction rules (one for each rejected split) at the nodes (see Section 5).

According to Breiman et al. [1], pages 87–92, predicting with the leaves of a decision tree will often have error at most twice that of the best pruning. They argue why this is likely when the structure of the decision tree is created from a training set independent of the test set (but drawn from the same distribution). In contrast, our main result is a very robust worst-case guarantee: the loss of our algorithm will be within a small constant factor of the loss of the best pruning of the decision tree on the test set, even if the training set (from which the decision tree is grown) and the test set are produced by different distributions, or the test set is chosen by an adversary.

In summary, the main result of this paper is a highly efficient and robust algorithm which provably predicts nearly as well as the best pruning of a given decision tree. We also describe how our method can be applied to the problem of predicting a sequence of symbols using variable memory length prediction models, and mention extensions to other loss functions.

2. Preliminaries

Let Σ be a finite alphabet of $|\Sigma|$ symbols. A *template tree* $\mathcal{T}$ over Σ is a rooted, $|\Sigma|$-ary tree where every internal node of $\mathcal{T}$ has one child for each symbol in Σ. Thus we can (and will) identify each node in $\mathcal{T}$ with the path (sequence of symbols in Σ) that leads from the root to that node.

We assume that there is a function which maps every instance x of the domain X to a path through the template tree $\mathcal{T}$ starting at the root and ending at a leaf. Typically, this path will be defined by a sequence of tests at the nodes of $\mathcal{T}$, each test resulting in the selection of one symbol in Σ which specifies the next child to visit. Although the template tree may be infinite, we require that each of the paths associated with an element of X be finite.

Figure 1 shows an example template tree over the alphabet $\Sigma = \{\text{T, F}\}$. The instance space X consists of all possible assignments to six boolean attributes, b_1 through b_6. Each internal node of $\mathcal{T}$ tests a single bit of the instance $x = (b_1, b_2, b_3, b_4, b_5, b_6)$ and branches left (T) if the bit is a one and branches right (F) if the bit is a zero.

For simplicity, we assume that all instances $x \in X$ are represented in a canonical form relative to the template tree. More specifically, we assume that each instance x is represented by the string in Σ^* defined by the path in $\mathcal{T}$ associated with the instance. Such a representation allows us to ignore the underlying tests at the nodes of $\mathcal{T}$. We identify instances with their canonical representations. Thus each instance x can be viewed as either an instance or a string in Σ^*.

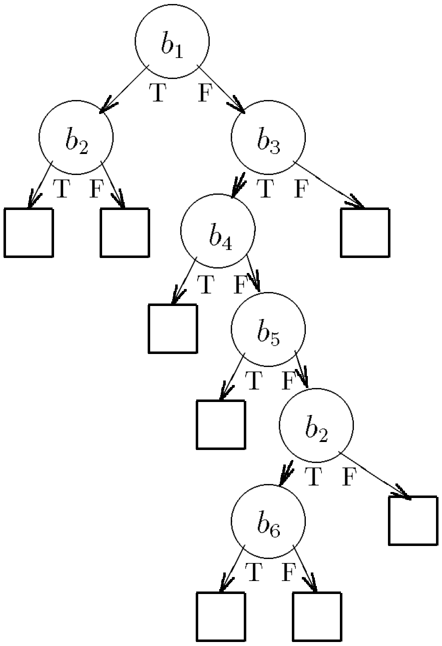

Figure 1. An example template tree.

For instance, in Figure 1, the canonical representation of the instance (0, 1, 0, 1, 0, 0) is "FF", and the canonical representation of the instance (0, 1, 1, 0, 0, 1) is "FTFFTT". Note that in both cases b_1 is 0, so b_3 is the second bit tested.

We use $|x|$ to denote the number of symbols in the canonical representation of x; thus, $|x|$ is the length of the path induced by x in $\mathcal{T}$. Also, since both instances and nodes are identified by strings in Σ^*, it is clear that a node u is a prefix of instance x (written $u \sqsubset x$) if and only if node u is on the path in $\mathcal{T}$ associated with x.

A *pruning* $\mathcal{P}$ of the tree $\mathcal{T}$ is a tree induced by replacing zero or more of the internal nodes (and associated subtrees) of $\mathcal{T}$ by leaves. Thus prunings also have the property that every node has either zero or $|\Sigma|$ children. When an instance x is evaluated in $\mathcal{P}$, it follows the same path it would have followed in $\mathcal{T}$, stopping when a leaf of $\mathcal{P}$ is reached. The leaf that is so reached is denoted by $\mathrm{leaf}_{\mathcal{P}}(x)$. The set of all leaves of $\mathcal{P}$ is written leaves($\mathcal{P}$), and the set of all nodes (including both leaves and internal nodes) is written nodes($\mathcal{P}$). (Recall that both the leaves and internal nodes of $\mathcal{P}$ are represented by strings in Σ^*.) The *size* of pruning $\mathcal{P}$, written $|\mathcal{P}|$, is the number of internal nodes and leaves in $\mathcal{P}$ *minus* the number of leaves in $\mathcal{P}$ that are also leaves of $\mathcal{T}$. Not counting the leaves of $\mathcal{T}$ in the size of $\mathcal{P}$ allows us to

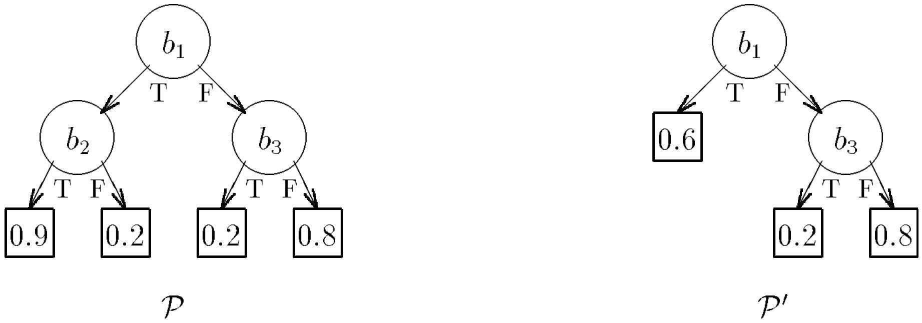

Figure 2. Two sample prunings of the template tree in Figure 1.

use the identity $\sum_{\mathcal{P}} 2^{-|\mathcal{P}|} = 1$ to simplify our later calculations. Figure 2 shows two possible prunings of the template tree shown in Figure 1, both having size 4.

In order to use these prunings for classification and prediction, we associate with each leaf of a pruning $\mathcal{P}$ a prediction rule to be used on those instances reaching that leaf in $\mathcal{P}$. We assume that the predictions made by a leaf in $\mathcal{P}$ are "inherited" from the associated node in $\mathcal{T}$. That is, we think of there being a dynamic "mini-expert" at each node of $\mathcal{T}$; the prediction of a pruning $\mathcal{P}$ is then the same as the prediction of the "mini-expert" at the leaf of $\mathcal{P}$ reached by the instance. Thus, if some node u is a prefix of a particular instance and u is a leaf in two different prunings, then both prunings will generate the same prediction for the instance.

For example, in Figure 2, we have indicated the predictions associated with the leaves of the two prunings. These predictions are real numbers in $[0, 1]$, whose interpretation is discussed further below. Note that, since both prunings contain the leaves FT and FF, both prunings give the same prediction whenever an instance reaches one of these leaves.

Although most decision-tree algorithms come up with a fixed prediction for each leaf, we allow more general mini-experts. The predictions of the mini-experts can be arbitrary functions of the current instance and/or previously seen instances. Our main results (Theorems 1 and 2) assume that each mini-expert's prediction can be computed in constant time and that all effects of a new instance on all of the various mini-experts can be recorded in $O(|x|)$ time.

The goal of our learning algorithm is to compete against the performance of the best, reasonably small such pruning by combining the predictions of all of the prunings. We study learning in the on-line prediction model used by Littlestone and Warmuth [10] and others. In this model, learning takes place in a sequence of trials $t = 1, \ldots, T$. At each time step t, an instance x^t is observed, and each pruning $\mathcal{P}$ generates a prediction $\xi^t_{\mathcal{P}} \in [0, 1]$. The master algorithm combines these predictions to produce its own prediction $\hat{y}^t \in [0,1]$. Finally, feedback $y^t \in \{0,1\}$ is observed. For example, if the path for instance x^t starts with TT then the prunings in Figure 2 make the predictions 0.9 and 0.6. In the next section we describe how the master algorithm produces its prediction $\hat{y}^t$ from these values and the predictions of the other prunings.

As discussed above, the prediction $\xi^t_{\mathcal{P}}$ of the pruning $\mathcal{P}$ is given, intuitively, by a mini-expert at the leaf reached by $\mathcal{P}$. That is, we assume formally that each[1] node u of $\mathcal{T}$ generates a prediction $\textsc{pred}^t(u) \in [0,1]$ for instance x^t, and furthermore, that

$$\xi^t_{\mathcal{P}} = \textsc{pred}^t(\mathrm{leaf}_{\mathcal{P}}(x^t)) \tag{1}$$

for all $\mathcal{P}$.

The loss of the master algorithm at time t is $|\hat{y}^t - y^t|$. We can interpret the prediction $\hat{y}^t \in [0,1]$ as the bias of a probabilistic prediction in $\{0,1\}$ which is 1 with probability $\hat{y}^t$, and 0 with probability $1-\hat{y}^t$. Then the loss suffered $|\hat{y}^t - y^t|$ is exactly the expected probability of the probabilistically predicted bit differing from the true outcome y^t.

The cumulative loss of the master algorithm is the sum of the losses incurred at all the trials:

$$L_A = \sum_{t=1}^{T} |\hat{y}^t - y^t|$$

and, analogously, the cumulative loss of each pruning $\mathcal{P}$ is

$$L_{\mathcal{P}} = \sum_{t=1}^{T} |\xi^t_{\mathcal{P}} - y^t|.$$

3. An inefficient master algorithm

In this section, we describe a master algorithm whose loss cannot be "much worse" than that of any "reasonably small" pruning. For the moment, we assume that computation time is not a consideration.

In this case, we can use the algorithm described by Cesa-Bianchi et al. [4], which is an extension of Littlestone and Warmuth's randomized weighted majority algorithm [10], and is related to Vovk's aggregating strategies [16]. This algorithm was called $P(\beta)$ in Cesa-Bianchi et al.'s notation, but we refer to it simply as the "master algorithm." The algorithm maintains a weight $w^t_{\mathcal{P}} > 0$ for each pruning $\mathcal{P}$. Thus the master algorithm of this section keeps a single explicit weight for each pruned tree. Initially,

$$\sum_{\mathcal{P}} w^1_{\mathcal{P}} = 1, \tag{2}$$

where the sum is over all possible prunings of $\mathcal{T}$.

The initial weights $w_{\mathcal{P}}^1$ can be viewed as a "prior" over the set of experts. Since our bounds are strongest for those strategies receiving the greatest initial weight, we want to choose initial weights that favor those strategies which we expect are most likely to perform the best. A reasonable choice is

$$w_{\mathcal{P}}^1 = 2^{-|\mathcal{P}|} \tag{3}$$

where $|\mathcal{P}|$ is the size measure defined in the previous section. This prior favors those prunings which are small and thus unlikely to reflect noise in the training set. (In small prunings, each leaf's prediction will tend to be based on more examples; see the discussion of bias versus variance in Breiman et al. [1], pages 87–92.) Although the master algorithm can run with any prior on the prunings, this $2^{-|\mathcal{P}|}$ prior enables us to efficiently implement the master algorithm as described in Section 4.

At each time step, the learner computes its prediction as

$$\hat{y}^t = F_\beta(r^t)$$

where $\beta \in [0,1]$ is a parameter of the algorithm, and r^t is a weighted average[2] of the predictions of the experts:

$$r^t = \frac{\sum_{\mathcal{P}} w_{\mathcal{P}}^t \xi_{\mathcal{P}}^t}{\sum_{\mathcal{P}} w_{\mathcal{P}}^t}. \tag{4}$$

The function F_β need only be bounded

$$1 + \frac{\ln((1-r)\beta + r)}{2\ln(\frac{2}{1+\beta})} \le F_\beta(r) \le \frac{-\ln(1 - r + r\beta)}{2\ln(\frac{2}{1+\beta})},$$

for all $0 \le r \le 1$. Cesa-Bianchi et al. [4] give several suitable F_β functions.

After feedback y^t is received, the weights are updated by the rule

$$w_{\mathcal{P}}^{t+1} = w_{\mathcal{P}}^t \cdot U_\beta(|\xi_{\mathcal{P}}^t - y^t|) \tag{5}$$

where U_β can be any function satisfying

$$\beta^r \le U_\beta(r) \le 1 - (1-\beta)r$$

for $r \in [0,1]$.

Cesa-Bianchi et al. show that the master algorithm suffers loss at most

$$\inf_{\mathcal{P}} \frac{L_{\mathcal{P}} \ln(1/\beta) + \ln(1/w_{\mathcal{P}}^1)}{2\ln(2/(1+\beta))}.$$

Using the choice for $w_{\mathcal{P}}^1$ given in equation (3), this bound shows that the loss of the master algorithm is linear in $L_{\mathcal{P}}$ and $|\mathcal{P}|$, for *every* pruning $\mathcal{P}$.

This bound is derived from the following two observations. First, any time the master algorithm incurs some loss ℓ, the sum of the updated weights is at most

$(\frac{1+\beta}{2})^{2\ell}$ times the sum of the weights used to predict. Thus if the master algorithm's total loss is L_A then the sum of the weights is reduced to $(\frac{1+\beta}{2})^{2L_A}$ or less. Second, if $L_{\mathcal{P}}$ is the loss of some pruning $\mathcal{P}$ then the weight of $\mathcal{P}$, and thus the sum of the weights, is always at least $w^t_{\mathcal{P}}(\beta)^{L_{\mathcal{P}}}$. Solving these constraints on the sum of the weights for L_A yields the above bound. Cesa-Bianchi et al. also discuss in detail how to choose the parameter β.

Since the preceding bound depends both on the pruning's loss and its size, the infimum might not be achieved by the pruning with the smallest loss, especially if this best pruning contains many nodes. However, the losses of the prunings are likely to grow with the number of predictions made, while the sizes of the prunings remain constant. In this case, the master algorithm's predictions will converge to those of the best pruning.

When only a few predictions are made, the loss of our algorithm is always less than (a constant times) the loss plus the size of each pruning. Thus, if there is some pruning $\mathcal{P}$ that is reasonably "small" and whose loss $L_{\mathcal{P}}$ is also reasonable then the loss of the master algorithm will also be small.

4. An efficient implementation

Unfortunately, the running time of this procedure is linear in the number of experts (i.e., prunings), which in this case is enormous (possibly even infinite). Obviously, we cannot efficiently maintain all of the weights $w^t_{\mathcal{P}}$ explicitly since there are far too many prunings to consider. Instead, we use a more subtle data structure, similar to the ones used by Buntine [3], [2] and Willems, Shtarkov and Tjalkens [20], [21], that can be used to compute the prediction $\hat{y}^t$ of the master algorithm. The size of this data structure is proportional to the number of nodes in $\mathcal{T}$ (or, more accurately, to the number of nodes that have actually been visited). Further, the time needed to compute the prediction $\hat{y}^t$ from the $\xi^t_{\mathcal{P}}$'s and to update the data structure is proportional, at each time step t, to $|x^t|$ (recall that $|x^t|$, in our canonical representation, is the length of the path defined by x^t in $\mathcal{T}$).

The basic idea is to maintain weights at the nodes that implicitly encode the weights of the various prunings. In particular, the weight of pruning $\mathcal{P}$ is represented as $2^{-|\mathcal{P}|}$ times the product of the weights stored at the leaves of $\mathcal{P}$. Initially, $\textsc{weight}^1(u) = 1$ for each node u, and these values are only changed if u is a prefix of some instance x^t. Thus even these node weights need only be stored explicitly for those nodes of $\mathcal{T}$ that have actually been visited. This allows us to apply this procedure efficiently even if $\mathcal{T}$ is extremely large, or even infinite (so long as every instance x defines a finite path through the tree).

The first main idea of this method is to show how to efficiently compute sums of the form

$$\sum_{\mathcal{P}} 2^{-|\mathcal{P}|} \prod_{s \in \text{leaves}(\mathcal{P})} g(s) \qquad (6)$$

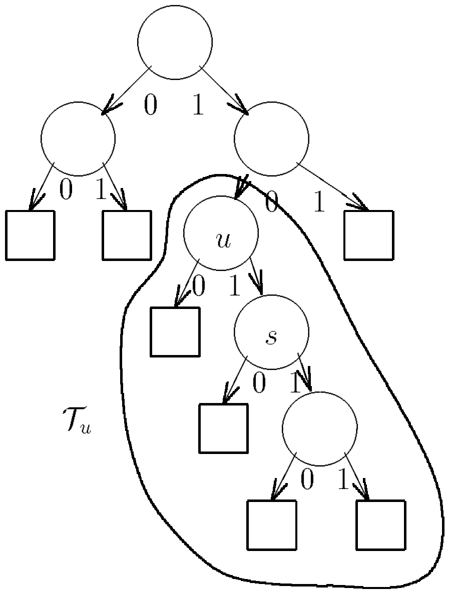

Figure 3. Subtrees and names of nodes.

where the first sum is over all prunings of $\mathcal{T}$and g is any easily computed function of the nodes. The second part of the method is to show that r^t (equation (4)) is the ratio of two sums, both of which can be written in this form.

Fix a template tree $\mathcal{T}$. For each node u in $\mathcal{T}$, let $\mathcal{T}_u$ be the subtree of $\mathcal{T}$ rooted at u. Note that each node in $\mathcal{T}_u$ is now associated with two strings: s for the path to the node in $\mathcal{T}$, and s' for the path to the node in $\mathcal{T}_u$ (beginning at the root u of the subtree). Clearly, these are related by the identity $s = us'$, the concatenation of u and s'.

For example, consider the tree in Figure 3. Node u is associated with the path 10. We use u to represent both the node in the tree and the string "10". Node s is associated with both the path 101 in the entire tree and the path $s' = 1$ in $\mathcal{T}_u$, the subtree rooted at u. Since 101 is the concatenation of 10 and 1, we have $s = us'$. This notational convenience allows us to easily express certain sums and products.

Let $g : \text{nodes}(\mathcal{T}) \to \mathbb{R}$ be any function. We define the function $\overline{g} : \text{nodes}(\mathcal{T}) \to \mathbb{R}$ as follows:

$$\overline{g}(u) = \sum_{\mathcal{P} \text{ of } \mathcal{T}_u} 2^{-|\mathcal{P}|} \prod_{s \in \text{leaves}(\mathcal{P})} g(us)$$

where we use the notation $\sum_{\mathcal{P} \text{ of } \mathcal{T}_u}$ to indicate summation over all prunings of $\mathcal{T}_u$. Note that the sum given in equation (6) is exactly equal to $\overline{g}(\lambda)$, where λ denotes the empty string. Thus, the following lemma, which gives an efficient method of computing $\overline{g}$, implies a method of computing sums of the form in equation (6). This lemma generalizes the proofs given for various special cases by Buntine [3], Lemma 6.5.1 and Willems, Shtarkov and Tjalkens [21], Appendices III and IV.

LEMMA 1 *Let g, $\overline{g}$ be as above. Then, for any node u of $\mathcal{T}$:*

1. *if u is a leaf then $\overline{g}(u) = g(u)$;*
2. *if u is an internal node, then $\overline{g}(u) = \frac{1}{2}g(u) + \frac{1}{2}\prod_{a\in\Sigma}\overline{g}(ua)$.*

Proof: Case 1 follows immediately from the definition of $\overline{g}$, keeping in mind that $|\mathcal{P}| = 0$ if $\mathcal{P}$ consists only of a leaf of $\mathcal{T}$.

For case 2, we can expand the sum recursively over the children of u. For simplicity, suppose that $\Sigma = \{0,1\}$; the similar proof for general Σ is sketched below. Note that any pruning $\mathcal{P}$ of the subtree $\mathcal{T}_u$ either contains only node u or can be decomposed into two subtrees, $\mathcal{P}_0$ and $\mathcal{P}_1$, rooted at the children $u0$ and $u1$ of u. By definition of $|\mathcal{P}|$, it can be shown that $|\mathcal{P}| = 1 + |\mathcal{P}_0| + |\mathcal{P}_1|$. Thus, separating out the case that $\mathcal{P}$ consists only of the node u, we can compute $\overline{g}(u)$ as

$$\tfrac{1}{2}g(u) + \sum_{\mathcal{P}_0}\sum_{\mathcal{P}_1} 2^{-(1+|\mathcal{P}_0|+|\mathcal{P}_1|)} \prod_{s_0} g(u0s_0) \prod_{s_1} g(u1s_1) \tag{7}$$

$$= \tfrac{1}{2}g(u) + \tfrac{1}{2}\left(\sum_{\mathcal{P}_0} 2^{-|\mathcal{P}_0|} \prod_{s_0} g(u0s_0)\right)\cdot\left(\sum_{\mathcal{P}_1} 2^{-|\mathcal{P}_1|} \prod_{s_1} g(u1s_1)\right) \tag{8}$$

$$= \tfrac{1}{2}g(u) + \tfrac{1}{2}\prod_{a\in\Sigma}\overline{g}(ua). \tag{9}$$

Here it is understood that, for $a \in \{0,1\}$, $\sum_{\mathcal{P}_a}$ denotes summation over all prunings $\mathcal{P}_a$ of $\mathcal{T}_{ua}$, and $\prod_{s_a}$ denotes product over all leaves s_a of $\mathcal{P}_a$.

In the more general case that $|\Sigma| > 2$, we repeat the sums and products in equation (7) analogously for each $a \in \Sigma$, and we use the more general identity $|\mathcal{P}| = 1 + \sum_{a\in\Sigma}|\mathcal{P}_a|$. Likewise, the factors in equation (8) are repeated for each $a \in \Sigma$, yielding equation (9) and completing the lemma. □

Thus, computing from the bottom up, the function $\overline{g}$ can be computed in time proportional to the number of nodes in $\mathcal{T}$. We will see later that, for the functions g of interest to us, a data structure can be used for even faster computation of $\overline{g}$.

We now show how Lemma 1 can be used to compute the ratio r^t of equation (4) efficiently. This will allow us to efficiently simulate the master algorithm of Cesa-Bianchi et al. [4].

For any node u, we define the "weight" of u at time step t, written $\textsc{weight}^t(u)$, as u's contribution on the first $t-1$ time steps to the weight decrease of any tree $\mathcal{P}$ which contains u as a leaf. That is, we define,

$$\textsc{weight}^t(u) = \prod_{\substack{1\le t'<t \\ u\sqsubset x^{t'}}} U_\beta(|\textsc{pred}^{t'}(u) - y^{t'}|)$$

(recall that $u \sqsubset x^{t'}$ means that u is a prefix of $x^{t'}$, and thus u is on the path described by $x^{t'}$). Clearly, by equation (1), if u is a leaf of $\mathcal{P}$, then

$$\textsc{weight}^t(u) = \prod_{\substack{1 \le t' < t \\ \mathrm{leaf}_{\mathcal{P}}(x^{t'}) = u}} U_\beta(|\xi^{t'}_{\mathcal{P}} - y^{t'}|).$$

In other words, if u is a leaf in pruning $\mathcal{P}$, then $\textsc{weight}^t(u)$ is the product of the weight update factors applied to the weight associated with $\mathcal{P}$ at those time steps when $\mathcal{P}$ predicts with the mini-expert at node u.

Recall that $U_\beta(|\xi^{t'}_{\mathcal{P}} - y^{t'}|)$ is the master algorithm's weight update function. For any pruning $\mathcal{P}$, we have by equations (3) and (5) that

$$\begin{aligned} w^t_{\mathcal{P}} &= 2^{-|\mathcal{P}|} \prod_{1 \le t' < t} U_\beta(|\xi^{t'}_{\mathcal{P}} - y^{t'}|) \\ &= 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \prod_{\substack{1 \le t' < t \\ \mathrm{leaf}_{\mathcal{P}}(x^{t'}) = u}} U_\beta(|\xi^{t'}_{\mathcal{P}} - y^{t'}|) \\ &= 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \textsc{weight}^t(u). \end{aligned}$$

Thus, the denominator of r^t is

$$\sum_{\mathcal{P}} w^t_{\mathcal{P}} = \sum_{\mathcal{P}} 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \textsc{weight}^t(u),$$

which has the form given in equation (6) and can be computed as $\overline{\textsc{weight}}^t(\lambda)$ using Lemma 1. The quantity $\overline{\textsc{weight}}^t(u)$ has an interpretation as the "weight of the subtree rooted at u." In other words $\overline{\textsc{weight}}^t(u)$ is the combined weight of all prunings of the subtree rooted at u.

Initially, each node u has weight 1 so that $\textsc{weight}^1(u) = 1$. It follows from Lemma 1 by a trivial induction argument that $\overline{\textsc{weight}}^1(u)$, the combined weight of the entire tree, is equal to 1. Thus, equation (2) is satisfied.

For the numerator of r^t, we define

$$\textsc{wpred}^t(u) = \begin{cases} \textsc{weight}^t(u)\textsc{pred}^t(u) & \text{if } u \sqsubset x^t \\ \textsc{weight}^t(u) & \text{otherwise.} \end{cases}$$

Then we have, for any pruning $\mathcal{P}$, that

$$\begin{aligned} w^t_{\mathcal{P}} \xi^t_{\mathcal{P}} &= 2^{-|\mathcal{P}|} \left(\prod_{u \in \mathrm{leaves}(\mathcal{P})} \textsc{weight}^t(u) \right) \xi^t_{\mathcal{P}} \\ &= 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \textsc{wpred}^t(u) \qquad (10) \end{aligned}$$

by equation (1). Thus,

$$\sum_{\mathcal{P}} w^t_{\mathcal{P}} \xi^t_{\mathcal{P}} = \sum_{\mathcal{P}} 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \textsc{wpred}^t(u) = \overline{\textsc{wpred}}^t(\lambda)$$

also has the form given in equation (6). As above, $\overline{\textsc{wpred}}^t(u)$ has an interpretation relating to the subtree rooted at u. The value $\overline{\textsc{wpred}}^t(u)$ is the sum over all prunings of the subtree rooted at u of the weight of the pruning times the prediction of the pruning. Thus, $\overline{\textsc{wpred}}^t(u)$ can be viewed as the "weighted prediction" of the prunings of the subtree rooted at u. The values $\overline{\textsc{wpred}}^t(u)$ are (generally) not normalized; as the total weights of the prunings decreases due to errors, so will $\overline{\textsc{wpred}}^t(u)$. Note, however, that the quotient $\overline{\textsc{wpred}}^t(u)/\overline{\textsc{weight}}^t(u)$ is the weighted average of the predictions made by the prunings of the subtree rooted at u.

We have shown then that the numerator and denominator of r^t (as expressed in equation (4)) can both be computed in time linear in the size of $\mathcal{T}$. In fact, this computation can be carried out at each time step t using time proportional to $|x^t|$ when the quantities $\textsc{weight}^t(u)$ and $\overline{\textsc{weight}}^t(u)$ are maintained at each node u. The pseudo-code for the procedure is given in Figure 4.

Initially $\textsc{weight}^1(u)$ and $\overline{\textsc{weight}}^1(u)$ are both equal to 1 for all nodes in $\mathcal{T}$. In general, after seeing x^t we must produce $r^t = \overline{\textsc{wpred}}^t(\lambda)/\overline{\textsc{weight}}^t(\lambda)$ used by the master algorithm at time t. The denominator, $\overline{\textsc{weight}}^t(\lambda)$, is immediately accessible since the $\overline{\textsc{weight}}^t(u)$ values are maintained at all of the nodes. To compute $\overline{\textsc{wpred}}^t(\lambda)$, we can apply Lemma 1 which suggests a recursive procedure taking time linear in the number of nodes of $\mathcal{T}$. Note, however, that if node u is not a prefix of x^t then $\textsc{wpred}^t(u) = \textsc{weight}^t(u)$. Furthermore, this equality also holds for all of the descendants of any u which is not a prefix of x^t so $\overline{\textsc{wpred}}^t(u) = \overline{\textsc{weight}}^t(u)$ for all u which are not prefixes of x^t. Thus $\overline{\textsc{wpred}}^t(u)$ need only be computed along the path of x^t in $\mathcal{T}$, allowing $\overline{\textsc{wpred}}^t(\lambda)$ to be computed in time linear in $|x^t|$.

Once y^t is observed, we need to update the values of $\textsc{weight}^t(u)$ and $\overline{\textsc{weight}}^t(u)$. Again, the $\textsc{weight}^t(u)$ and $\overline{\textsc{weight}}^t(u)$ values change only for those u which are prefixes of x^t. Each new $\textsc{weight}^t(u)$ value requires a single multiplication, and the new $\overline{\textsc{weight}}^t(u)$ values can be computed "bottom-up" in time proportional to $|x^t|$.

To summarize, we have thus proved the following theorem, which is the main result of this paper.

THEOREM 1 *Let $\mathcal{T}$ be a template tree, let $(x^1, y^1), \ldots, (x^T, y^T)$ be any sequence of instance-feedback pairs, and let the predictions $\xi^t_{\mathcal{P}}$ associated with each pruning $\mathcal{P}$ of $\mathcal{T}$ be of the form given in equation (1). Then the loss of the master algorithm given in Figure 4 is at most*

$$\frac{L_{\mathcal{P}} \ln(1/\beta) + |\mathcal{P}| \ln(2)}{2 \ln(2/(1+\beta))}$$

```
Input: template tree $\mathcal{T}$
       access to predictions $\textsc{pred}^t(u)$ of mini-experts
       parameter $\beta \in [0,1]$
Initialize $\textsc{weight}^1(u) = \overline{\textsc{weight}}^1(u) = 1$ for all nodes $u$ in $\mathcal{T}$
Do for $t = 1, 2, \ldots$

• Prediction:
    – Given $x^t \in \Sigma^*$
    – Compute weighted predictions $\overline{\textsc{wpred}}^t(u)$ for each subtree using the rule:
        $\overline{\textsc{wpred}}^t(u) = \begin{cases} \overline{\textsc{weight}}^t(u) & \text{if } u \not\sqsubset x^t \quad \text{(off path)} \\ \textsc{weight}^t(u)\textsc{pred}^t(u) & \text{if } u = x^t \quad \text{(path leaf)} \\ \frac{1}{2}\textsc{weight}^t(u)\textsc{pred}^t(u) + \frac{1}{2}\prod_{a\in\Sigma} \overline{\textsc{wpred}}^t(ua) & \\ & \text{otherwise} \quad \text{(path internal node)} \end{cases}$
    – Predict $\hat{y}^t = F_\beta(\overline{\textsc{wpred}}^t(\lambda)/\overline{\textsc{weight}}^t(\lambda))$

• Update:
    – Update the weight of each node, $\textsc{weight}^t$:
        $\textsc{weight}^{t+1}(u) = \begin{cases} \textsc{weight}^t(u)U_\beta(|\textsc{pred}^t(u) - y^t|) & \text{if } u \sqsubset x^t \quad \text{(on path)} \\ \textsc{weight}^t(u) & \text{otherwise} \quad \text{(off path)} \end{cases}$
    – Update the subtree weights $\overline{\textsc{weight}}^t$:
        $\overline{\textsc{weight}}^{t+1}(u) = \begin{cases} \overline{\textsc{weight}}^t(u) & \text{if } u \not\sqsubset x^t \quad \text{(off path)} \\ \textsc{weight}^{t+1}(u) & \text{if } u = x^t \quad \text{(path leaf)} \\ \frac{1}{2}\textsc{weight}^{t+1}(u) + \frac{1}{2}\prod_{a\in\Sigma} \overline{\textsc{weight}}^{t+1}(ua) & \\ & \text{otherwise} \quad \text{(path internal node)} \end{cases}$
```

Figure 4. Pseudo-code for the master algorithm.

for every pruning $\mathcal{P}$. Furthermore, the running time of this algorithm, at every time step t, is linear in $|x^t|$.

Recall that $\textsc{weight}^1(u) = \overline{\textsc{weight}}^1(u) = 1$ for any node u, and that these values are only changed if u is a prefix of some instance x^t. Thus these quantities need only be stored explicitly for the nodes of $\mathcal{T}$ that have actually been visited. This allows us to apply this procedure efficiently even if $\mathcal{T}$ is extremely large, or even infinite (so long as every instance x defines a finite path through the tree).

Finally, we remark that this $O(|x^t|)$ running time does not include the time required to update the mini-experts' predictions. However, if the template tree is produced by a batch process so that each node's predictions are fixed in advance then no updating is necessary. Furthermore, the predictions at a node will often be an easily calculated function of the instances on which that node has previously predicted. Functions such as Laplace's estimator $(\frac{hits+1}{trials+2})$ are based on the examples previously seen by that node and can be updated in constant time.

5. Multiple prediction rules at each node

In this section, we extend the preceding results to a more general setting in which there is more than one "mini-expert" or prediction rule associated with each node of the template tree. Here, our goal is to select not only the best pruning but also the best mini-expert at each leaf of this pruning.

For example, suppose we are given a template tree for routing instances but no prediction rule at the nodes. In this case, we might associate with each node two mini-experts corresponding to the deterministic boolean rules which always predict 0 or always predict 1. The goal then is to make predictions that are almost as good as the best *labeled* pruning, i.e., the best pruning whose leaves have each been labeled with the best deterministic boolean prediction rule. As before, the mini-experts need not make the same prediction every time; their predictions can depend on the current instance and past history.

More formally, let n be the number of mini-experts associated with each node of the template tree.[3] Our goal now is to compete against the predictions made by each *labeled pruning* $(\mathcal{P}, \mathcal{I})$, where $\mathcal{P}$ is a pruning and $\mathcal{I} : \text{leaves}(\mathcal{P}) \rightarrow \{1, \ldots, n\}$ assigns a mini-expert to each leaf of $\mathcal{P}$. That is, $\mathcal{P}$ tells which pruning to use, and $\mathcal{I}$ tells us which of the mini-experts to predict with for each leaf of $\mathcal{P}$. The prediction at time t of such a labeled pruning is denoted $\xi^t_{\mathcal{P},\mathcal{I}}$.

At each time step t, each node u generates a prediction $\text{PRED}^t(u,i) \in [0,1]$ for $i = 1, \ldots, n$ where i is the index of a mini-expert at node u. Analogous to Equation (1), we assume formally that

$$\xi^t_{\mathcal{P},\mathcal{I}} = \text{PRED}^t(\text{leaf}_{\mathcal{P}}(x^t), \mathcal{I}(\text{leaf}_{\mathcal{P}}(x^t))). \tag{11}$$

The cumulative loss of a labeled pruning is defined to be

$$L_{\mathcal{P},\mathcal{I}} = \sum_{t=1}^{T} |\xi^t_{\mathcal{P},\mathcal{I}} - y^t|.$$

Our goal is to come up with a master algorithm with cumulative loss close to that of the best labeled pruning.

To do so, in the obvious manner, we can replace the weights $w^t_{\mathcal{P}}$ used in Section 3 by weights $w^t_{\mathcal{P},\mathcal{I}}$ for every labeled pruning $(\mathcal{P}, \mathcal{I})$. We choose the initial weights to be

$$w^1_{\mathcal{P},\mathcal{I}} = 2^{-|\mathcal{P}|} \cdot n^{-|\mathrm{leaves}(\mathcal{P})|}.$$

As before, applying the results of Cesa-Bianchi et al. [4] immediately gives us a bound on the loss of the resulting master algorithm. To implement this algorithm efficiently, we need to be able to compute

$$r^t = \frac{\sum_{\mathcal{P},\mathcal{I}} w^t_{\mathcal{P},\mathcal{I}} \xi^t_{\mathcal{P},\mathcal{I}}}{\sum_{\mathcal{P},\mathcal{I}} w^t_{\mathcal{P},\mathcal{I}}}.$$

As before, we show how numerator and denominator can be written in the form given in Equation (6).

First, for any function $h : \mathrm{nodes}(\mathcal{T}) \times \{1, \ldots, n\} \to \mathbb{R}$, it can be verified that

$$\sum_{\mathcal{P},\mathcal{I}} 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} h(u, \mathcal{I}(u)) = \sum_{\mathcal{P}} 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \sum_{i=1}^{n} h(u, i).$$

This can be seen by "multiplying out" the product appearing on the right hand side. Therefore, any expression of the form

$$\sum_{\mathcal{P},\mathcal{I}} 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} h(u, \mathcal{I}(u)) \tag{12}$$

can be evaluated efficiently by applying Lemma 1 with $g(u)$ set to $\sum_{i=1}^{n} h(u, i)$. To compute r^t then, it suffices to write the denominator and numerator in the form given in Equation (12).

For the denominator, we define

$$\textsc{weight}^t(u, i) = \prod_{\substack{1 \le t' < t \\ u \sqsubset x^{t'}}} U_\beta(|\textsc{pred}^{t'}(u, i) - y^{t'}|).$$

Then

$$\begin{aligned} w^t_{\mathcal{P},\mathcal{I}} &= 2^{-|\mathcal{P}|} n^{-|\mathrm{leaves}(\mathcal{P})|} \prod_{1 \le t' < t} U_\beta(|\xi^{t'}_{\mathcal{P},\mathcal{I}} - y^{t'}|) \\ &= 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \frac{1}{n} \prod_{\substack{1 \le t' < t \\ \mathrm{leaf}_{\mathcal{P}}(x^{t'}) = u}} U_\beta(|\xi^{t'}_{\mathcal{P},\mathcal{I}} - y^{t'}|) \\ &= 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \frac{1}{n} \textsc{weight}^t(u, \mathcal{I}(u)). \end{aligned}$$

Thus,

$$\sum_{\mathcal{P},\mathcal{I}} w^t_{\mathcal{P},\mathcal{I}} = \sum_{\mathcal{P},\mathcal{I}} 2^{-|\mathcal{P}|} \prod_{u \in \mathrm{leaves}(\mathcal{P})} \frac{1}{n} \textsc{weight}^t(u, \mathcal{I}(u))$$

has the form given in Equation (12). Similarly, for the numerator, we define

$$\textsc{wpred}^t(u,i) = \begin{cases} \textsc{weight}^t(u,i)\textsc{pred}^t(u,i) & \text{if } u \sqsubset x^t \\ \textsc{weight}^t(u,i) & \text{otherwise.} \end{cases}$$

Then it can be shown, as in Equation (10), that

$$\sum_{\mathcal{P},\mathcal{I}} w^t_{\mathcal{P},\mathcal{I}} \xi^t_{\mathcal{P},\mathcal{I}} = \sum_{\mathcal{P},\mathcal{I}} 2^{-|\mathcal{P}|} \prod_{u \in \text{leaves}(\mathcal{P})} \frac{1}{n} \textsc{wpred}^t(u, \mathcal{I}(u))$$

which is of the desired form.

Unraveling these ideas, we obtain the algorithm shown in Figure 5. The properties of this algorithm are summarized by the following theorem:

Theorem 2 *Let $\mathcal{T}$ be a template tree, let $(x^1, y^1), \ldots, (x^T, y^T)$ be any sequence of instance-feedback pairs, and let the predictions $\xi^t_{\mathcal{P},\mathcal{I}}$ associated with each labeled pruning $(\mathcal{P}, \mathcal{I})$ be of the form given in Equation (11) where n is the number of mini-experts associated with each node. Then the loss of the master algorithm given in Figure 5 is at most*

$$\frac{L_{\mathcal{P},\mathcal{I}} \ln(1/\beta) + |\mathcal{P}| \ln(2) + |\text{leaves}(\mathcal{P})| \ln n}{2 \ln(2/(1+\beta))}$$

for every labeled pruning $(\mathcal{P}, \mathcal{I})$. Furthermore, the running time of this algorithm, at every time step t, is linear in $|x^t| n$.

6. Other applications and extensions

In a real implementation of our algorithm, the weights stored at each node may become extremely small, possibly causing a floating-point underflow. There is a simple trick for avoiding this difficulty, based on the following observation: Suppose all of the weights $\textsc{weight}^t(u)$ of the nodes u along a given root-to-leaf path are multiplied by some constant c. Then because each pruning contains exactly one leaf that is a node from the given path, this effectively causes both $\overline{\textsc{wpred}}^t(\lambda)$ and $\overline{\textsc{weight}}^t(\lambda)$ to be multiplied by c, and therefore, the ratio of these values (which is used to produce the algorithm's predictions) is unaffected. Thus, if the weights along a path in the tree seem too small, we can multiply all of these weights by a constant to prevent floating-point underflow.[4]

As a simple application of our result, we can use our method to predict a sequence of symbols, say, the next letter in a passage of English text. We might restrict our predictions to depend on the most recently observed sequence of characters. For instance, on seeing "q," we might reliably predict that the next letter is "u." Obviously, in other cases, a longer context is needed for reliable prediction. Thus, we would like to use different lengths for the different contexts. By defining a template tree in which the root node tests the last symbol, its children test the

```
Input: template tree $\mathcal{T}$
       access to predictions $\text{PRED}^t(u,i)$ of mini-experts
       parameter $\beta \in [0,1]$
Initialize $\text{WEIGHT}^1(u,i) = \overline{\text{WEIGHT}}^1(u) = 1$ for all nodes $u$ in $\mathcal{T}$, and $i = 1,\ldots,n$.
Do for $t = 1,2,\ldots$
  • Prediction:
    – Given $x^t \in \Sigma^*$
    – Compute weighted predictions $\overline{\text{WPRED}}^t(u)$ for each subtree using the rule:
      $\overline{\text{WPRED}}^t(u) = \begin{cases} \overline{\text{WEIGHT}}^t(u) & \text{if } u \not\sqsubseteq x^t \quad \text{(off path)} \\ \frac{1}{n}\sum_{i=1}^n \text{WEIGHT}^t(u,i)\text{PRED}^t(u,i) & \\ & \text{if } u = x^t \quad \text{(path leaf)} \\ \frac{1}{2n}\sum_{i=1}^n \text{WEIGHT}^t(u,i)\text{PRED}^t(u,i) + \frac{1}{2}\prod_{a\in\Sigma}\overline{\text{WPRED}}^t(ua) & \\ & \text{otherwise} \quad \text{(path internal node)} \end{cases}$
    – Predict $\hat{y}^t = F_\beta(\overline{\text{WPRED}}^t(\lambda)/\overline{\text{WEIGHT}}^t(\lambda))$
  • Update:
    – Update $\text{WEIGHT}^t$:
      $\text{WEIGHT}^{t+1}(u,i) = \begin{cases} \text{WEIGHT}^t(u,i)U_\beta(|\text{PRED}^t(u,i) - y^t|) & \text{if } u \sqsubset x^t \quad \text{(on path)} \\ \text{WEIGHT}^t(u,i) & \text{otherwise} \quad \text{(off path)} \end{cases}$
    – Update the subtree weights $\overline{\text{WEIGHT}}^t$:
      $\overline{\text{WEIGHT}}^{t+1}(u) = \begin{cases} \overline{\text{WEIGHT}}^t(u) & \text{if } u \not\sqsubseteq x^t \quad \text{(off path)} \\ \frac{1}{n}\sum_{i=1}^n \text{WEIGHT}^{t+1}(u,i) & \text{if } u = x^t \quad \text{(path leaf)} \\ \frac{1}{2n}\sum_{i=1}^n \text{WEIGHT}^{t+1}(u,i) + \frac{1}{2}\prod_{a\in\Sigma}\overline{\text{WEIGHT}}^{t+1}(ua) & \\ & \text{otherwise} \quad \text{(path internal node)} \end{cases}$
```

Figure 5. Pseudo-code for the master algorithm with multiple mini-experts.

symbol before last, and so on, we can use our method to make predictions that are competitive with the best pruning. Such a pruning, in this case, is equivalent to a rule for determining one of several variable-length contexts, which in turn can be used to predict the next symbol. Learning results on such suffix trees were presented by Ron, Singer and Tishby [15].

Similar tree machines have been used to represent finite memory sources in the information theory community, and they form the core of Rissanen's Context algorithm for universal data compression [14] (see also [17], [18], [19]). In work more closely related to the results presented here, an efficient algorithm for averaging over prunings of such trees was presented by Willems, Shtarkov and Tjalkens [20], [21]. However, these authors focus on predicting a distribution of symbols for coding purposes, rather than simply predicting what the next symbol will be.

Our method is easily extended to other loss functions provided that there exists a multiplicative weight-update algorithm of the appropriate form. For instance, such algorithms are given by Vovk [16], Kivinen and Warmuth [7], and Freund and Schapire [5].

Acknowledgments

Thanks to Jason Catlett, William Cohen, Yoav Freund, Ron Kohavi, Jonathan Oliver, Alon Orlitsky, Dana Ron, Linda Sellie, Bruce Sherrod, Yoram Singer, Manfred Warmuth, and Marcelo Weinberger for many helpful discussions. Thanks also to Meier Feder for (indirectly) bringing references [20], [21] to our attention, and to the anonymous reviewers for their careful reading and helpful comments.

Notes

1. Actually, we only use the predictions of node u when u is a prefix of x^t.
2. Although the initial weights sum to 1, this is generally not the case due to the update step of Equation (5). Therefore dividing by the sum of the weights is necessary to obtain the weighted average of the experts' predictions.
3. The generalization to the case in which the number of mini-experts varies from node to node is straightforward.
4. Note that this operation affects the data structure in other ways; for instance, all of the values $\overline{\text{WEIGHT}}^t(u)$ for nodes u along the given path must be updated.

References

1. Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.
2. Wray Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.
3. Wray Lindsay Buntine. *A Theory of Learning Classification Rules.* PhD thesis, University of Technology, Sydney, 1990.
4. Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 382–391, 1993. To appear, *Journal of the Association for Computing Machinery.*
5. Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Second European Conference, EuroCOLT '95*, pages 23–37. Springer-Verlag, 1995.
6. Trevor Hastie and Daryl Pregibon. Shrinking trees. Technical report, AT&T Bell Laboratories, 1990.
7. Jyrki Kivinen and Manfred K. Warmuth. Using experts for predicting continuous outcomes. In *Computational Learning Theory: EuroCOLT '93*, pages 109–120. Springer-Verlag, 1994.

8. Suk Wah Kwok and Chris Carter. Multiple decision trees. In Ross D. Shachter, Tod S. Levitt, Laveen N. Kanal, and John F. Lemmer, editors, *Uncertainty in Artificial Intelligence 4*, pages 327–335. North-Holland, 1990.
9. Nick Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
10. Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
11. Jonathan J. Oliver and David Hand. Averaging over decision stumps. In *Machine Learning: ECML-94*, pages 231–241. Springer-Verlag, 1994.
12. Jonathan J. Oliver and David J. Hand. On pruning and averaging decision trees. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 430–437, 1995.
13. J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
14. Jorma Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, IT-29(5):656–664, September 1983.
15. Dana Ron, Yoram Singer, and Naftali Tishby. Learning probabilistic automata with variable memory length. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 35–46, 1994.
16. Volodimir G. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.
17. M. J. Weinberger, A. Lempel, and J. Ziv. Universal coding of finite-memory sources. *IEEE Transactions on Information Theory*, 38(3):1002–1014, May 1992.
18. Marcelo J. Weinberger, Neri Merhav, and Meir Feder. Optimal sequential probability assignment for individual sequences. *IEEE Transactions on Information Theory*, 40(2):384–396, March 1994.
19. Marcelo J. Weinberger, Jorma J. Rissanen, and Meir Feder. A universal finite memory source. *IEEE Transactions on Information Theory*, 41(3):643–652, 1995.
20. F. M. J. Willems, Y. M. Shtarkov, and Tj. J. Tjalkens. Context tree weighting: a sequential universal source coding procedure for FSMX sources. In *Proceedings 1993 IEEE International Symposium on Information Theory*, page 59, 1993.
21. Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The context tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.