Machine Learning, 36, 183–199 (1999)
© 1999 Kluwer Academic Publishers. Manufactured in The Netherlands.

An Efficient Extension to Mixture Techniques for Prediction and Decision Trees

FERNANDO C. PEREIRA pereira@research.att.com
YORAM SINGER singer@research.att.com
AT&T Labs, 180 Park Avenue, Florham Park, NJ 07932

Editor: Lisa Hellerstein

Abstract. We present an efficient method for maintaining mixtures of prunings of a prediction or decision tree that extends the previous methods for "node-based" prunings (Buntine, 1990; Willems, Shtarkov, & Tjalkens, 1995; Helmbold & Schapire, 1997; Singer, 1997) to the larger class of *edge-based* prunings. The method includes an online weight-allocation algorithm that can be used for prediction, compression and classification. Although the set of edge-based prunings of a given tree is much larger than that of node-based prunings, our algorithm has similar space and time complexity to that of previous mixture algorithms for trees. Using the general online framework of Freund and Schapire (1997), we prove that our algorithm maintains correctly the mixture weights for edge-based prunings with any bounded loss function. We also give a similar algorithm for the logarithmic loss function with a corresponding weight-allocation algorithm. Finally, we describe experiments comparing node-based and edge-based mixture models for estimating the probability of the next word in English text, which show the advantages of edge-based models.

Keywords: mixture models, decision and prediction trees, on-line learning, statistical language modeling

1. Introduction

Recent work in information theory (Willems, Shtarkov, & Tjalkens, 1995) and machine learning (Helmbold & Schapire, 1997) shows that it is possible to maintain efficiently the mixture weights for certain sets of prunings of decision and prediction trees. The solution employs a recursive algorithm that updates the weights of each possible pruning, and also computes the weighted decision or prediction of the entire mixture. Those results apply to the class of *node-based* prunings, which are obtained from a *template* tree by removing entire subtrees rooted at selected nodes. Each node in the template tree has an associated *predictor*, which in the applications we consider is estimated from empirical observations. The prediction of a pruning is then an appropriate combination of the predictions of the leaf nodes of the pruning.

In applications such as statistical language modeling, in which the out-degree of nodes in the template tree may be very large, many nodes are seldom visited. Therefore, the empirical estimates of the prediction functions for those nodes may be poor. Nevertheless, the predictions of such nodes will contribute to many prunings. We propose to alleviate this problem by using the larger class of *edge-based prunings*, which are obtained by deleting some of the internal edges of a template tree, and all of their descendants. That is, while a node-based pruning deletes all the edges leaving a node and the subtrees below them, an edge-based pruning may delete only some of the edges and the subtrees below them. An edge-based pruning can thus be used to eliminate some poor-performing descendants of a node while keeping the good-performing ones, which is not possible with node-based prunings.

Our online weight-allocation algorithm maintains efficiently the correct weight for each possible edge-based pruning. The core of the algorithm is a new recurrence for calculating the mixture of all prunings rooted below a given node in the template tree. Although the set of possible edge-based prunings might be much larger than the set of the node-based prunings, we show that the space and time requirements of the new algorithm are similar to those of previous algorithms.

We also describe statistical language-modeling experiments in which edge-based mixture models are used to estimate the probability of the next word in English news text, showing that edge-based models perform better than node-based models, and that both kinds of mixture models are competitive with the back-off models (Katz, 1997) that have been a standard tool in statistical language modeling. Finally, we conclude and discuss possible extensions.

2. Preliminaries

The tasks we examine here are online classification and prediction. At each time step $t = 1, \ldots, T$ the learning algorithm receives an instance x^t and outputs a prediction $\hat{y}^t$. For example, in a context-based sequence prediction or compression algorithm, the instance $x^t = x_j x_{j+1} \cdots x_t$ is a suffix of an underlying input sequence $x_1, \ldots, x_t$ over some fixed finite input alphabet. After prediction, an outcome y^t is observed that results in a loss $l^t = L(\hat{y}^t, y^t)$. We derive a weight update rule for bounded-loss predictors using the framework introduced by Freund and Schapire (1997) which generalizes former online weight allocation algorithms (DeSantis, Markowsky, & Wegman, 1988; Vovk, 1990; Littlestone & Warmuth, 1994; Cesa-Bianchi et al., 1997) and can be applied to a wide variety of learning problems. This derivation does not depend on the precise form of the loss function, requiring only that the appropriate loss value be provided to the learning algorithm after each prediction.

More precisely, the generic learning algorithm **A** for the bounded-loss case maintains weights for a pool of N predictors. At time step t, **A** uses a normalized weight vector $\mathbf{p}^t$ such that p_i^t is the weight of predictor i and $\sum_{i=1}^{N} p_i^t = 1$. Each predictor suffers a loss l_i^t as described above, and the *mixture loss* suffered by **A** is $\mathbf{p}^t \cdot \mathbf{l}^t = \sum_{i=1}^{N} p_i^t \, l_i^t$. The goal of **A** is to minimize the cumulative mixture loss relative to the cumulative loss of the best predictor: $L_A - \min_i L_i$, where $L_A = \sum_{t=1}^{T} \mathbf{p}^t \cdot \mathbf{l}^t$ and $L_i = \sum_{t=1}^{T} l_i^t$. The set of predictors we consider in the rest of the paper have the special form of prunings of a fixed prediction tree.

In applications like compression or language modeling for speech recognition, the output of algorithm output after each input is a probability distribution over possible next symbols $\hat{y}^t = P^t$. Given an actual next symbol $y^t = x_{t+1}$, the algorithm suffers the loss $l^t = -\log P^t(x_{t+1})$. This loss function does not fit the requirements of the analysis and generic algorithm just described, but it will turn out that the same weight-update method works in that case. Therefore, we will start by extending the generic algorithm and its analysis for the bounded-loss case to edge prunings, and then show how the same weight update applies to probabilistic predictions with the logarithmic loss.

Let Σ be a finite alphabet of K symbols. A *template tree* $\mathcal{T}$ over Σ is a rooted tree in which the edges leaving each node are labeled by distinct elements of Σ (thus, the maximum node out-degree is K). Clearly, a template tree can be identified with a prefix-closed subset of Σ^*: the root node is identified with the empty string λ; if v is identified with $s \in \Sigma^*$ and there is an edge e from v to v' labeled $\sigma \in \Sigma$, v' is identified with $s\sigma$. The *depth* of a node is just the length of the corresponding string. It will also be convenient to represent the edge labeled σ from u to $u\sigma$ with the string $u\sigma$. It will be clear from the context whether a string represents a node or an edge. We write $u \sqsubset v$ to indicate that string u is a prefix of string v, or under the identification of nodes (or edges) with strings, that u is an (possibly improper) ancestor of v.

We use template trees to associate paths (strings) with instances as follows. Each node of template tree $\mathcal{T}$ is associated with a test on instances with possible outcomes in Σ. Given an instance x, tests are performed sequentially on x, starting with the test for the root node λ. If the test at node u has outcome σ and $u\sigma \in \mathcal{T}$ then the test for $u\sigma$ is performed next. Otherwise, u is the *maximal path* in $\mathcal{T}$ for x, denoted by $\mathcal{T}(x)$, and x is classified as belonging to node u. It will be convenient to abbreviate the formula $u \sqsubset \mathcal{T}(x)$, asserting that a node is on the maximal path for x, as $u \sqsubset x$.

It should be noted that our definitions of template tree and maximal path allows for incomplete template trees, in which an instance may map to a path which is a prefix of the path for another instance. This generalizes the definitions used by Willems, Shtarkov, and Tjalkens (1995) and by Helmbold and Schapire (1997), which allow only complete K-ary trees.

In sequence prediction or compression algorithms, the test for any node of depth i returns x_{t-i}, while in decision trees the sequence of tests is defined by the tree-growing algorithm that yielded $\mathcal{T}$ (Breiman et al., 1984; Quinlan, 1993). For instance, in a decision tree a test may check whether a numerical feature is greater or smaller than a threshold. Based on the outcome of the test, either the edge labeled 0 (false) or the one labeled 1 (true) is followed. For simplicity in what follows, we will concentrate on the sequence prediction case, and identify each instance (the reversal of a suffix of the input sequence) with the corresponding maximal path in $\mathcal{T}$.

To describe and analyze edge prunings, we need to associate certain quantities with each edge that could leave a node, whether the edge occurs or not in a particular pruning. A simple way to do this is to represent each potential edge leaving node u (represented by a string) by $u\sigma$ for an appropriate symbol σ, even though $u\sigma$ may not correspond to a node of the template tree. More formally, for each template tree $\mathcal{T}$ over Σ, we define the corresponding *edge extension* $\tilde{\mathcal{T}}$ as the superset of $\mathcal{T}$ defined as follows:

- If $v \in \mathcal{T}$ is a leaf node, $v\bot \in \tilde{\mathcal{T}}$ where $\bot$ is a fixed new symbol not in Σ.
- If $v \in \mathcal{T}$ is not a leaf, then $\forall\sigma \in \Sigma$: $v\sigma \in \tilde{\mathcal{T}}$.

We will call *internal* those edges that $\tilde{\mathcal{T}}$ inherits from $\mathcal{T}$, and *terminal* the additional edges in $\tilde{\mathcal{T}}$. Terminal edges serve as placeholders for the quantities associated with missing edges, as we just argued.

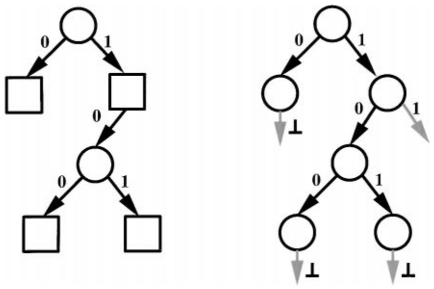

Figure 1. A template tree and its edge based extension. Boxes denote internal, incomplete, nodes that are used for prediction.

Given an instance x, we perform tests on x as before to determine a maximal path in $\tilde{\mathcal{T}}$. By construction, for a test with outcome σ performed at node s, either edge $s\sigma$ belongs to the tree, or the only outgoing edge is $s\perp$, in which case $s\perp$ is the maximal path $\tilde{\mathcal{T}}(x)$ for x.

In figure 1 we show a template tree over $\Sigma = \{0, 1\}$ and its extended version. Extended template trees are used in the analysis, but the actual algorithm does not need to represent explicitly additional terminal edges of the extended tree.

A pruning $\mathcal{P}$ of the extended tree $\tilde{\mathcal{T}}$ is an extended tree obtained by replacing zero or more of the internal edges of $\tilde{\mathcal{T}}$ with terminal edges with the same label, and deleting the subtrees reached by the affected edges. We write '$\mathcal{P}$ of $\tilde{\mathcal{T}}$' to indicate that $\mathcal{P}$ is a pruning of $\tilde{\mathcal{T}}$. Clearly, such prunings are in one-to-one correspondence with prunings of $\mathcal{T}$. The set of edges of $\mathcal{P}$ is denoted by edges $(\mathcal{P})$ and the set of all terminal edges by term $(\mathcal{P})$. When an instance x is evaluated in $\mathcal{P}$, it follows the same path that it would have followed in $\tilde{\mathcal{T}}$ until it reaches a terminal edge. Following the conventions introduced earlier, we denote by $\mathcal{P}(x)$ both that maximal path and its final (terminal) edge. The *size* of a pruning $\mathcal{P}$, written $|\mathcal{P}|$, is the sum of the number of internal edges of $\mathcal{P}$ and the number of terminal edges of $\mathcal{P}$ that are *not* terminal edges of $\tilde{\mathcal{T}}$. Thus, the size of a pruning is the number of edges at which there was the option to include or exclude the edge and its descendants in the pruning. This definition is analogous to those of of Willems, Shtarkov, and Tjalkens (1995) and Helmbold and Schapire (1997) for node-based prunings, which count the number of nodes at which there is an option to prune the tree. In both cases, the prior probability of a pruning is a simple function of size if we assume a fixed probability of pruning at any edge (node). Appendix A gives further details of the relationship between edge-based prunings and node-based prunings.

In classification or prediction, each node in the original template tree $\mathcal{T}$ has a predictor for instances terminating at that node. In terms of the extended template $\tilde{\mathcal{T}}$, each terminal edge of a pruning is associated with the predictor for its source node. That is, the predictions for observations matching nonterminal edges of a node come from the descendants of the node, while the predictions for observations matching terminal edges come from the node itself. Intuitively, the subtree rooted at an edge is pruned if the observation suffixes represented by that subtree are not significantly more informative than the shorter suffix represented by the source node of the edge.

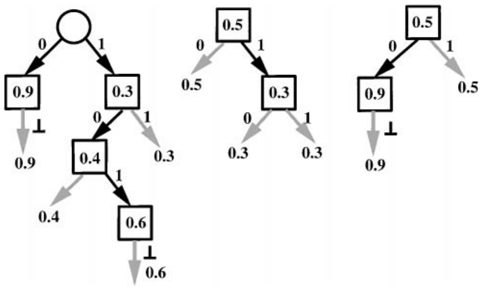

Figure 2. Three sample edge-based prunings of the template tree in figure 1.

Clearly, two prunings that share a terminal edge have agreeing predictions on any instance that reaches that terminal edge of the common subtree. For example, figure 2 shows three prunings of the template tree in figure 1. The terminal edges of each pruning are drawn in gray. Since both the leftmost and the middle tree contain the terminal edge reached on instance 11, they predict the same on this instance.

According to the foregoing discussion, the prediction at time step t of the pruning $\mathcal{P}$ is given by the prediction function for the terminal edge reached by $\mathcal{P}$ on x^t. The predictions at the nodes can depend, for instance, on the different counts of the next symbol in context-based sequence predictions or on the label associated with the instance in decision trees (Breiman et al., 1984; Quinlan, 1993).

After prediction, pruning $\mathcal{P}$ suffers loss

$$l^t_{\mathcal{P}} = l^t_{\mathcal{P}(x^t)}. \tag{1}$$

The overall loss at time t of a weight-allocation algorithm for all possible edge prunings is

$$l^t = \sum_{\mathcal{P}} w^t_{\mathcal{P}}\, l^t_{\mathcal{P}}.$$

We seek a weight-allocation algorithm **A** whose cumulative loss L_A is as close as possible to the loss $L_{\mathcal{P}}$ of the best pruning $\mathcal{P}$.

3. An efficient weight-allocation algorithm

We now describe an efficient weight-allocation algorithm for competing against the best edge-based pruning of a prediction tree, based on the *Hedge* algorithm of Freund and Schapire (1997), which maintains a non-negative weight vector over predictors. In our case, the predictors are prunings. We denote the unnormalized weight of a pruning at time t by $w^t_{\mathcal{P}}$, and its initial weight by $w^1_{\mathcal{P}}$. Initial weights can be viewed as priors over prunings. A reasonable choice of prior is $w^1_{\mathcal{P}} = 2^{-|\mathcal{P}|}$, where $|\mathcal{P}|$ is the size of the pruning as defined in Section 2. This prior is monotonically decreasing in the size of the pruning. More generally, we can define a prior over prunings from any assignment of pruning probabilities to template tree edges. The prior probability of the pruning is the probability that the pruning will be generated by the following recursive process. Starting at the root node, for each edge leaving a node we randomly decide whether to prune the subtree reached by the edge by tossing a coin with bias given by the edge's pruning probability. If the subtree is kept, the process is then recursively applied to the root of the subtree. Any such recursive prior enables efficient evaluation of the mixture weights and results in worst case loss bounds which are relatively simple to compute. Our earlier experiments with node prunings (Pereira, Singer, & Tishby, 1995) suggest that the exact value of the pruning prior is not very important in language modeling with large training sets, but a more detailed analysis of the role of the prior would be worthwhile.

At each time step t, each pruning $\mathcal{P}$ suffers a loss $l^t_{\mathcal{P}}$, and its weight is updated using the multiplicative rule

$$w^{t+1}_{\mathcal{P}} = w^t_{\mathcal{P}}\,\beta^{l^t_{\mathcal{P}}} = w^1_{\mathcal{P}} \prod_{s=1}^{t} \beta^{l^s_{\mathcal{P}}} = 2^{-|\mathcal{P}|} \prod_{s=1}^{t} \beta^{l^s_{\mathcal{P}}}, \tag{2}$$

where β is the learning rate. *Hedge* uses the normalized weight vector

$$\mathbf{p}^t = \frac{1}{\sum_{\mathcal{P}} w^t_{\mathcal{P}}} \mathbf{w}^t$$

to combine the predictions of the predictors. Our algorithm maintains efficiently both the weight vector $\mathbf{w}^t$ and the normalization $\sum_{\mathcal{P}} w^t_{\mathcal{P}}$. Freund and Schapire (1997) proved that algorithms of this form suffer a loss of at most

$$\min_{\mathcal{P}} \frac{-\ln\left(w^1_{\mathcal{P}}\right) - \ln(\beta) L_{\mathcal{P}}}{1-\beta} = \min_{\mathcal{P}} \frac{|\mathcal{P}| - \ln(\beta) L_{\mathcal{P}}}{1-\beta}$$

for bounded loss functions. Thus, the loss of the weight allocation algorithm scales linearly with the loss of the best submodel.

In a naïve implementation, the weight vector and its contribution to the normalization would be computed for each pruning separately. Such an approach is clearly infeasible given the huge number of possible prunings, especially when Σ is large. However, we can adopt the techniques of Willems et al. (1995) and Helmbold and Schapire (1997) to our case. Those techniques require $|x^t|$ time to update the weights, where $|x^t|$ is the number of edges in the path associated to x^t in $\mathcal{T}$.

The main idea of the algorithm is to extend the technique of Helmbold and Schapire (1997) for computing certain sums over all the possible node-based prunings of a template tree to similar sums ranging over all edge-based prunings of a template tree. For a fixed template tree $\mathcal{T}$ and any function $g : \text{edges}\ (\tilde{\mathcal{T}}) \rightarrow \mathbb{R}$ we are interested in computing

$$E(g) = \sum_{\mathcal{P} \text{ of } \tilde{\mathcal{T}}} 2^{-|\mathcal{P}|} \prod_{e \in \text{term}(\mathcal{P})} g(e), \tag{3}$$

where term($\mathcal{P}$) is the set of terminal edges of $\mathcal{P}$, as defined earlier.

As we show later, sums of the above form are the main tool in forming the predictions for new instances.

In the following, we fix a template tree $\mathcal{T}$ and assume that $\mathcal{T}$ is complete, that is, each node is either a leaf or has K children. We show later how to extend the analysis to incomplete template tree.

For any node u of $\tilde{\mathcal{T}}$, let $\tilde{\mathcal{T}}_u$ be the subtree of $\tilde{\mathcal{T}}$ rooted at u. Following our convention of representing nodes and edges by strings, each node or edge of $\tilde{\mathcal{T}}_u$ is identified in $\tilde{\mathcal{T}}$ with string s and in $\tilde{\mathcal{T}}_u$ with a string s' such that $s = us'$.

For any function g : edges $(\tilde{\mathcal{T}}) \rightarrow \mathbb{R}$, we define the function $\bar{g} : \tilde{\mathcal{T}} \rightarrow \mathbb{R}$ by

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

Then $E(g) = \bar{g}(\lambda)$. The following lemma gives an efficient method of computing $\bar{g}$, and thus in particular a method for computing $E(g)$. The lemma generalizes Lemma 1 of Helmbold and Schapire (1997), which in turn generalizes the more specialized results of Buntine (1990, Lemma 6.5.1) and Willems, Shtarkov, and Tjalkens (1995, Appendices III and IV).

Recall that we identify each edge with the string for the path leading to that edge, which in turn can also be identified by target node of the edge.

Lemma 1. *Let g, $\bar{g}$ be as above. Then, for any node u of a complete template tree $\tilde{\mathcal{T}}$:*
1. *If u is a leaf, then $\bar{g}(u) = g(u\bot)$;*
2. *If u is an internal node, then $\bar{g}(u) = \prod_{\sigma \in \Sigma} (\frac{1}{2} g(u\sigma) + \frac{1}{2}\bar{g}(u\sigma))$.*

Proof: Case 1 follows from the definition of $\bar{g}$ and $\tilde{\mathcal{T}}$: if u is a leaf in $\mathcal{T}$, $\mathcal{P}$ is the single node u, with edge extension $u\bot$.

For case 2 (u is an internal node), we consider first the special case of $\Sigma = \{0, 1\}$. Any pruning $\mathcal{P}$ of $\mathcal{T}_u$ falls into one of four cases: both outgoing edges from u are terminal edges of $\mathcal{P}$, only edge $u0$ is terminal in $\mathcal{P}$, only edge $u1$ is terminal in $\mathcal{P}$, or both edges are internal in $\mathcal{P}$. Denote by $\mathcal{P}_0$ ($\mathcal{P}_1$) the subtree rooted at the child $u0$ ($u1$) of u, which may be empty if the corresponding edge is terminal in $\mathcal{P}$. By the definition of $|\mathcal{P}|$, it is easy to see that in general $|\mathcal{P}| = K + \sum_{\sigma \in \Sigma} |\mathcal{P}_\sigma|$, and thus $|\mathcal{P}| = 2 + |\mathcal{P}_0| + |\mathcal{P}_1|$ for a binary alphabet. Decomposing the sum over all possible prunings into the above four cases, $\bar{g}(u)$ for a binary alphabet can be written as

$$\begin{aligned}
\bar{g}(u) &= \sum_{\mathcal{P}_0} \sum_{\mathcal{P}_1} 2^{-(2+|\mathcal{P}_0|+|\mathcal{P}_1|)} \prod_{s_0 \in \text{term}(\mathcal{P}_0)} g(u0s_0) \prod_{s_1 \in \text{term}(\mathcal{P}_1)} g(u1s_1) \\
&= \frac{1}{4} g(u0)g(u1) + \frac{1}{4} g(u1) \sum_{\mathcal{P}_0} 2^{-|\mathcal{P}_0|} \prod_{s_0} g(u0s_0) \\
&\quad + \frac{1}{4} g(u0) \sum_{\mathcal{P}_1} 2^{-|\mathcal{P}_1|} \prod_{s_1} g(u1s_1)
\end{aligned}$$

$$
\begin{aligned}
&+\frac{1}{4}\left(\sum_{\mathcal{P}_0} 2^{-|\mathcal{P}_0|}\prod_{s_0} g(u0s_0)\right)\left(\sum_{\mathcal{P}_1} 2^{-|\mathcal{P}_1|}\prod_{s_1} g(u1s_1)\right)\\
&= \frac{1}{2}g(u0)\frac{1}{2}g(u1) + \frac{1}{2}\bar{g}(u0)\frac{1}{2}g(u1) + \frac{1}{2}g(u0)\frac{1}{2}\bar{g}(u1) + \frac{1}{2}\bar{g}(u0)\frac{1}{2}\bar{g}(u1)\\
&= \left(\frac{1}{2}g(u0) + \frac{1}{2}\bar{g}(u0)\right)\left(\frac{1}{2}g(u1) + \frac{1}{2}\bar{g}(u1)\right),
\end{aligned}
$$

where $\sum_{\mathcal{P}_\sigma}$ ranges over all prunings $\mathcal{P}_\sigma$ of $\mathcal{T}_{u\sigma}$.

For the case of a general alphabet, define

$$
I_{\mathcal{P}}(u\sigma) = \begin{cases} 1 & u\sigma \text{ is an internal edge of } \mathcal{P} \\ 0 & \text{otherwise} \end{cases}
$$

and $\bar{I}_{\mathcal{P}} = \langle I_{\mathcal{P}}(u\sigma_1), I_{\mathcal{P}}(u\sigma_2), \ldots, I_{\mathcal{P}}(u\sigma_K)\rangle \in \{0,1\}^K$. Each possible pruning corresponds to a unique binary vector $\bar{I}_{\mathcal{P}}$. Thus, we can decompose $\bar{g}(u)$ into 2^K terms corresponding to the possible values of $\bar{I}_{\mathcal{P}}$, yielding for the general case

$$
\begin{aligned}
\bar{g}(u) &= \sum_{\bar{I}_{\mathcal{P}}\in\{0,1\}^K}\ \prod_{\sigma\in\Sigma, I_{\mathcal{P}}(u\sigma)=1} \frac{1}{2}\bar{g}(u\sigma) \prod_{\sigma\in\Sigma, I_{\mathcal{P}}(u\sigma)=0} \frac{1}{2}g(u\sigma)\\
&= \prod_{\sigma\in\Sigma}\left(\frac{1}{2}g(u\sigma) + \frac{1}{2}\bar{g}(u\sigma)\right),
\end{aligned}
$$

where the second equality was derived by applying $K-1$ times the identity $(a_0+b_0)(a_1+b_1) = \sum_{i=0,1,\ j=0,1} a_i b_j$. □

If the template tree $\mathcal{T}$ is incomplete, some of $\tilde{\mathcal{T}}$'s internal nodes will have at least one outgoing terminal edge. If u is such an incomplete node and $u\sigma$ a terminal edge, namely $I_{\mathcal{P}}(u\sigma) = 0$. Then, there is a single pruning $\mathcal{P}_u$ of $\tilde{\mathcal{T}}_u$ containing a single edge from u to $u\sigma$. The equation for case 2 takes then the form:

$$
\bar{g}(u) = \prod_{\sigma\in\Sigma,\ u\sigma\in\mathcal{T}}\left(\frac{1}{2}g(u\sigma) + \frac{1}{2}\bar{g}(u\sigma)\right)\prod_{\sigma\in\Sigma,\ u\sigma\notin\mathcal{T}} g(u\sigma).
$$

It is straightforward, albeit tedious, to modify Lemma 1 for the general case of incomplete template trees. Briefly, we follow the same line of inductive proof by applying the recursion for a given node only to subtrees which contain more than one edge.

Using Lemma 1 and an adaptation of the derivation of Helmbold and Schapire (1997), we now compute the losses (2) for prunings and thus their weights and the overall weight normalization. The *weight* $\mathrm{w}^t(e)$ at step t of each edge $e = u\sigma \in \tilde{\mathcal{T}}$ is defined as:

$$
\mathrm{w}^t(e) = \prod_{\substack{1\le i<t\\ e\sqsubset x^i}} \beta^{l_u^i}. \tag{4}
$$

Assume now that e is a terminal edge of pruning $\mathcal{P}$. Then the condition $e \sqsubset x^i$ is equivalent to $\mathcal{P}(x^i) = u\sigma = e$. Therefore, from the assumption and (1) we conclude

$$\mathrm{w}^t(e) = \prod_{\substack{1 \le i < t \\ \mathcal{P}(x^i)=e}} \beta^{l^i_{\mathcal{P}}}.$$

That is, if e is a terminal edge of pruning $\mathcal{P}$, $\mathrm{w}^t(e)$ is the product of the loss factors for the time steps in which the predictor associated with the source node of e was used.

Recall that $\beta^{l^i_{\mathcal{P}}}$ is the allocation algorithm's weight update factor for pruning $\mathcal{P}$ at time step i. By (2), we have

$$w^t_{\mathcal{P}} = 2^{-|\mathcal{P}|} \prod_{1 \le i < t} \beta^{l^i_{\mathcal{P}}} = 2^{-|\mathcal{P}|} \prod_{e \in \mathrm{term}(\mathcal{P})} \prod_{\substack{1 \le i < t \\ \mathcal{P}(x^i)=e}} \beta^{l^i_{\mathcal{P}}} = 2^{-|\mathcal{P}|} \prod_{e \in \mathrm{term}(\mathcal{P})} \mathrm{w}^t(e).$$

The weight normalization for prunings is then

$$\sum_{\mathcal{P}} w^t_{\mathcal{P}} = \sum_{\mathcal{P}} 2^{-|\mathcal{P}|} \prod_{e \in \mathrm{term}(\mathcal{P})} \mathrm{w}^t(e), \tag{5}$$

which has the form (3) and can therefore be computed as $\bar{\mathrm{w}}^t(\lambda)$ using Lemma 1. We use (4) to compute the weight update for edge e, although we do not need to maintain $\mathrm{w}^t(e)$ for terminal edges $e = u\bot$ in $\tilde{\mathcal{T}}$, because $\mathrm{w}^t(u\bot) = \bar{\mathrm{w}}^t(u\bot)$ by Lemma 1, case 1.

Since no loss has been incurred initially, $\mathrm{w}^1(e) = 1$ for every edge e. A straightforward induction from Lemma 1 shows that $\bar{\mathrm{w}}^1(\lambda) = 1$, and thus the priors over prunings add up correctly as required $\sum_{\mathcal{P}} w^1_{\mathcal{P}} = 1$. Finally, the normalized weights $\mathbf{p}$ can be computed directly from from $\mathrm{w}^t(u)$ and $\bar{\mathrm{w}}^t(u)$ at each node. Figure 3 shows weight-update pseudocode for complete template trees summarizing the results of our analysis.

This basic algorithm can be improved in several ways. First, in practical applications such as compression and statistical language modeling, the shape of the template tree is defined in advance (for instance, as a complete tree of given depth), but the actual tree data structure is built "on the fly" as new instances are looked up in the template. An internal node u of the tree data structure will lack the outgoing edge for e for σ when no instance reaching that edge has yet been encountered. In that case, no loss has yet been incurred by e. That is, no loss is associated to the parts of the template not yet built, and the terms associated to those losses can be ignored in the recursive calculation of the weights $\bar{\mathrm{w}}^t$.

Second, although the calculation of $\bar{\mathrm{w}}^{t+1}(u)$ in figure 3 iterates over Σ if u is an internal node, in reality only one of u's children is affected by the update, the one satisfying $u\sigma \sqsubset x^t$. Therefore we have $\mathrm{w}^{t+1}(u\sigma') = \mathrm{w}^t(u\sigma')$ and $\bar{\mathrm{w}}^{t+1}(u\sigma') = \bar{\mathrm{w}}^t(u\sigma')$ for $\sigma' \ne \sigma$, allowing the update of nodes reached at time step t to be performed in time independent of $|\Sigma|$ as follows:

$$\bar{\mathrm{w}}^{t+1}(u) = \bar{\mathrm{w}}^t(u) \frac{\frac{1}{2}\mathrm{w}^{t+1}(u\sigma) + \frac{1}{2}\bar{\mathrm{w}}^{t+1}(u\sigma)}{\frac{1}{2}\mathrm{w}^t(u\sigma) + \frac{1}{2}\bar{\mathrm{w}}^t(u\sigma)}. \tag{6}$$

```
Input: Complete template tree $\mathcal{T}$
       access to losses $l_u^t$ of node predictors
       parameter $\beta \in [0, 1]$
Initialize $\mathrm{w}^1(e) = \bar{\mathrm{w}}^1(e) = 1$ for all edges in $\tilde{\mathcal{T}}$
Do for $t = 1, 2, \ldots$

    • Update the edge weights $\mathrm{w}^t$:

      $\mathrm{w}^{t+1}(u\sigma) = \begin{cases} \mathrm{w}^t(u\sigma)\beta^{l_u^t} & \text{if } u\sigma \sqsubset x^t \ \text{(on path)} \\ \mathrm{w}^t(u\sigma) & \text{otherwise} \ \text{(off path)} \end{cases}$

    • Update the subtree weights $\bar{\mathrm{w}}^t$:

      $\bar{\mathrm{w}}^{t+1}(u) = \begin{cases} \bar{\mathrm{w}}^t(u) & \text{if } u \not\sqsubset x^t \quad \text{(off path)} \\ \bar{\mathrm{w}}^t(u)\, \beta^{l_u^t} & \text{if } u\bot = \tilde{\mathcal{T}}(x^t) \ \text{(terminal edge)} \\ \prod_{\sigma\in\Sigma} \left( \frac{1}{2}\mathrm{w}^{t+1}(u\sigma) + \frac{1}{2}\bar{\mathrm{w}}^{t+1}(u\sigma) \right) & \text{otherwise} \quad \text{(internal edge)} \end{cases}$
```

Figure 3. Pseudo-code for the weight-allocation algorithm.

Therefore, the time required to update the entire tree for an instance is at most the depth of the template tree, or if the template tree is (notionally) infinite, the length of the input instance.

For bounded loss functions, the loss bounds of the *Hedge* algorithm apply giving:

Theorem 1. *Let $\mathcal{T}$ be a template tree, let $x^1, \ldots, x^T$ be any sequence of instances, and let the losses $l_{\mathcal{P}}^t$ associated with each pruning $\mathcal{P}$ of $\mathcal{T}$ be of the form given in* (1). *Then the loss of Hedge based on the on-line weight allocation algorithm of figure* 3 *is at most*

$$\frac{-\ln\left(w_{\mathcal{P}}^1\right) - \ln(\beta)L_{\mathcal{P}}}{1-\beta} = \frac{\ln(2)|\mathcal{P}| - \ln(\beta)L_{\mathcal{P}}}{1-\beta},$$

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

The weight allocation algorithm is also used for prediction. Let P_u^t be the prediction of node u at time step t, and $\overline{\mathrm{WP}}^t(u)$ be the weighted mixture of the predictions of all prunings rooted at u at time step t. Assuming a complete template tree, a straightforward adaptation of the argument of Lemma 1 gives the following recursion for $\overline{\mathrm{WP}}^t(u)$:

$$\overline{\mathrm{WP}}^t(u) = \begin{cases} \bar{\mathrm{w}}^t(u)\, P_u^t & \text{if } u\bot = \tilde{\mathcal{T}}(x^t) \text{ (terminal edge)} \\ \prod_{\sigma\in\Sigma} \left(\frac{1}{2}\mathrm{w}^t(u\sigma)\, P_u^t + \frac{1}{2}\overline{\mathrm{WP}}^t(u\sigma) \right) & \text{otherwise (internal edge)} \end{cases} \tag{7}$$

The prediction of the entire mixture, denoted $\hat{y}^t$, is computed by normalizing $\overline{\text{WP}}^t(\lambda)$ by the total mixture weight $\bar{\text{w}}^t(\lambda)$, and applying an appropriate function to the normalized prediction:

$$\hat{y}^t = F_\beta(\overline{\text{WP}}^t(\lambda)/\bar{\text{w}}^t(\lambda)). \tag{8}$$

The function $F_\beta(y)$ depends on the learning rate β and need only to be bounded for all $0 \leq y \leq 1$, as discussed in more detail by Cesa-Bianchi et al. (1997).

In compression applications, the appropriate loss function for a pruning $\mathcal{P}$ is the negative log probability (logloss) of the input sequence, which determines the code length of the sequence as compressed by an optimal coder, for instance, an arithmetic coder governed by the next-input probabilities given by the pruning (Bell, Cleary, & Witten, 1990).

Thus, the loss incurred by $\mathcal{P}$ at time t is

$$l_{\mathcal{P}}^t = -\log P_{\mathcal{P}(x^t)}(x_{t+1}). \tag{9}$$

We can still use the weight-update algorithm of figure 3 and the prediction mixing formula (7) is we set $\beta = 1/e$ and $F_\beta(r) = r$. Then $\beta^{l_u^t} = P_u^t(x_{t+1})$, and from figure 3 and (7) it is easy to see that $\overline{\text{WP}}^t(u) = \bar{\text{w}}^{t+1}(u)$. Therefore, the prediction of the mixture is simply

$$\hat{y}^t = \bar{\text{w}}^{t+1}(\lambda)/\bar{\text{w}}^t(\lambda). \tag{10}$$

The time and the space complexity of the edge-based algorithm for the logloss remain the same. Using Lemma 1 we get the following bound on the predictions of the mixture.

Theorem 2. *Let $\mathcal{T}$ be a template tree and $x^1, \ldots, x^T$ be any sequence of instances, and $L_{\mathcal{P}} = \sum_{t=1}^T l_{\mathcal{P}}^t$ be the loss of the pruning $\mathcal{P}$ of $\mathcal{T}$ on the sequence, where $l_{\mathcal{P}}^t$ is given by* (9). *Then the logloss of the weight allocation algorithm of figure* 3 *is at most*

$$-\ln\left(w_{\mathcal{P}}^1\right) + L_{\mathcal{P}} = \ln(2)|\mathcal{P}| + L_{\mathcal{P}}.$$

Proof: The proof is a direct application of the proof technique of technique of DeSantis et al. (1988). From (10) we get that the log loss of the mixture on the sequence $x^1, \ldots, x^T$ is

$$\begin{aligned} -\sum_{t=1}^T \ln(\hat{y}^t) &= -\ln\left(\prod_{t=1}^T \hat{y}^t\right) = -\ln\left(\prod_{t=1}^T \frac{\bar{\text{w}}^{t+1}(\lambda)}{\bar{\text{w}}^t(\lambda)}\right) \\ &= -\ln(\bar{\text{w}}^{T+1}(\lambda)) - \ln(\bar{\text{w}}^1(\lambda)) = -\ln(\bar{\text{w}}^{T+1}(\lambda)), \end{aligned} \tag{11}$$

where for the last equality we used the fact that the initial weights are set such that $\bar{\text{w}}^1(\lambda) = 1$. From (5) and Lemma 1, the following holds for any pruning $\mathcal{P}$:

$$\bar{\text{w}}^{T+1}(\lambda) = \sum_{\mathcal{P}} w_{\mathcal{P}}^{T+1} \geq w_{\mathcal{P}}^{T+1}. \tag{12}$$

We now get the bound on the logloss by combining (11) and (12) while using the fact that $w^1_{\mathcal{P}} = 2^{-|\mathcal{P}|}$. □

4. Experiments

We used the algorithm just discussed to build statistical language models that estimate the probability that a word follows a sequence of other words in English text. Such models are essential in large-vocabulary speech recognition (Jelinek, 1998). In contrast to the character-based models used in text compression, the word-based models used in speech recognition have very large alphabets (the words in a dictionary). In addition, in typical applications the model is built in batch mode from a large collection of training text, and then applied to the test task without online modification. Our experiments show, however, that an online algorithm can have competitive performance with the most widely-used batch language-modeling method (Katz, 1987) even for very large vocabularies. The experiments also suggest that edge-based pruning has considerable advantages over node-based pruning for the very large alphabets required in speech recognition.

Our experiments compare edge-based and node-based mixture algorithms with the baseline *back-off* method of (Katz, 1987) on training and test material derived from the NIST-supplied North-American Business News (NAB) corpus of English news text. The text is tokenized into words. The alphabet for the algorithm consists of the K most frequent words plus an "unknown word" symbol to which all the other words are mapped; we used $K = 60{,}000$ on some experiments, but memory limits forced us to use $K = 20{,}000$ for the remaining experiments. Model performance is evaluated by the log-likelihood assigned by the model to test material. For $K = 60{,}000$, the training text consisted of around 78 million words, and, for $K = 20{,}000$, we used around 61 million words of training material (the size of the training sets was also determined by memory limits). In both cases, the training data was created by a random drawing without repetition of sentences from the whole NAB corpus. We used the log-conditional probability of the next word as prediction loss.

We store in working memory only the subtree of the template containing the prediction suffixes found in training data. When a suffix corresponding to a new branch of the template tree is observed, we allocate the corresponding nodes and initialize the counts used in forming predictions to zero. The next word probabilities are also estimated online from those counts, as we describe below.

Each node predictor must assign a conditional probability to any word even if that word has not been observed before in the context represented by the node. The are several possible methods to assign word probabilities. The modified Laplace rule (Krichevsky & Trofimov, 1981) is often used in character-based text compression, but turned out to be inadequate for large alphabets. This rule estimates the conditional probability of observing a symbol at a tree node to $\frac{n_\sigma+1/2}{n+K/2}$, where n_σ is the number of times symbol σ was observed at the node and n is the number of times the node was visited. The regret (additional loss) of this estimator scales linearly with the size of the alphabet, hence its performance is poor when only a small fraction of the entire alphabet is observed at a node. The common method for estimating probabilities of unseen words is due to Turing and Good (Good, 1953). The Good-Turing estimator is used in batch as it uses the number of words which appeared only once for estimating the conditional probability of unseen words.

Instead, we used two estimation techniques that have been shown to perform well on natural data sets and can easily modified for on-line prediction problems (Witten & Bell, 1991). The first estimates the probability of symbol σ at a node as $\frac{n_\sigma}{n+r}$ where r is the number of different symbols observed at the node. The second method, which is an approximation of the Good-Turing formula, estimates the probability of symbol σ as $\frac{(n-t_1)n_\sigma}{n^2}$, where t_1 is the number of different words that have been observed only once at the node. This scheme requires a "fall-back" estimate when $t_1 = n$, as described in more detail by Witten and Bell (1991).

We stress that the goal of the experiments was to compare the two different mixture techniques and not possible online estimation techniques for the node predictors. However, our confidence in the advantage of edge-based mixtures over node-based mixtures is increased by the fact that they can be demonstrated with both unseen event estimators, as we will now see.

For $K = 60{,}000$ we used a template tree of depth 2 with the larger training set while for $K = 20{,}000$ we used a (much larger) template tree of depth 3 with the smaller training set. Let $\hat{P}_{\text{edge}}(x_t)$ ($\hat{P}_{\text{node}}(x_t)$) be the probability of x_t as induced by the edge-based (node-based) mixture model. Figure 4 shows the differences between the average logloss of the two models every million words $\frac{1}{T_i}\sum_{t=1}^{T_i}[\log \hat{P}_{\text{edge}}(x_t) - \log \hat{P}_{\text{node}}(x_t)]$ $(T_i = 10^6 \times i)$ as the two algorithms operate in online mode on the training data . Each plot in the figure shows results for the two estimation methods. The first plot gives the results for the depth 2 template tree (on the larger data set) and the second plot gives the results for the depth 3 tree. In both cases, the initial predictions of the node-based mixture are better than those of the edge-based mixture, but eventually the edge-based mixture achieves a better performance, starting at around 10 million words for the depth 2 tree and at over 50 million words for the depth 3 tree. The results show similar trends for both estimation methods, and are consistent with the theoretical analysis. The set of edge-based prunings strictly contains the set of node-based prunings, and therefore the initial weight of a pruning that belongs to both sets is much smaller as a member of the edge-based pool of prunings. Hence, at the beginning of a run, when the number of observations is relatively small, the initial weights of the good prediction trees have a large influence on the logloss. As more data is processed, the influence of the initial weights becomes less crucial. In addition, in the larger collection of edge-based prunings there may be some that achieve lower loss than any

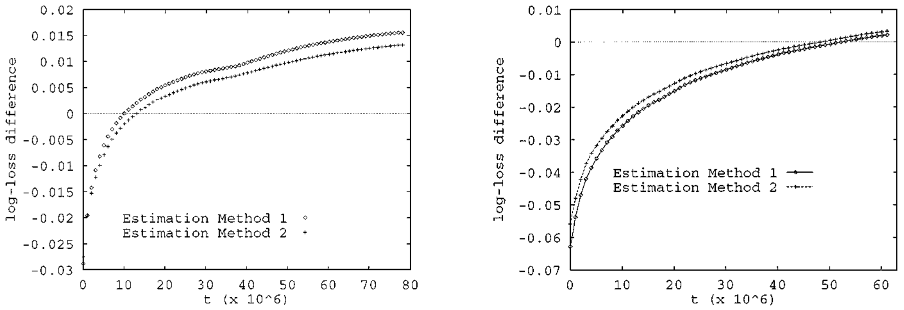

Figure 4. The difference in the log-loss of the edge-based and node-based mixtures for the NAB corpus using a depth 2 (left) and a depth 3 (right) template trees.

Table 1. Perplexity results on data the NAB corpus.

Model	Perplexity
Back-off trigram	95.2
Node-based mixture	100.5
Edge-based mixture	98.9
Node-based mixture (w/adaptation)	97.9
Edge-based mixture (w/adaptation)	96.0

node-based pruning, although it takes a lot of data to adjust the weights to favor those better prunings.

We also would like to note that the time and space needed to build an edge-based and a node-based language model is practically the same (see also Appendix A). Specifically, for the corpora described in this section it takes no more than 30 CPU minutes on an 194 MHz MIPS R10000 processor to build each model, which is much faster than standard statistical language modeling algorithms. The major drawback of the online algorithms is their memory requirements, which were over 1GB for the experiments described here. For this reason we had to limit the training set and vocabulary size in our experiments. We describe a possible extension that overcomes the memory requirements in Section 5.

Finally, we compared the performance of both mixture models with a standard back-off model (Katz, 1987). This model estimates the probability of a word n-gram seen in training from its empirical frequency in training discounted by the Good-Turing formula. The discounting makes some probability mass available for unseen n-grams. This mass is distributed among unseen n-grams according to the model probabilities of the $n-1$-grams obtained from the n-grams by dropping their first word.

For a fair comparison, we took an unseen test set of 13 million words, and measured the loss of a fixed mixture model, built from the training set, on that test set. That is, the mixture model is not allowed to adapt to the test set. Table 1 summarizes the test-set perplexity, a measure used in evaluating statistical language models which is simply the exponentiation of the average logloss on the test data. We used the second estimation method for the node predictors. We also calculated the perplexity when the models were allowed to adapt their weights to the test data. Even though the mixture models are created by an online algorithm, they achieve a performance close to the back-off model, which uses the Good-Turing estimator instead of the online approximation we used. Furthermore, the edge-based mixture model achieves a small but significant improvement in perplexity over the node-based mixture model. Given the large sizes of training and test sets, all of these perplexity differences are statistically significant.

5. Conclusions

In this paper we presented an efficient method for maintaining mixtures of prunings of a prediction or decision tree that extends the node-based prunings of earlier work to the larger class of edge-based prunings, while using the same time and space complexity of previous mixture algorithms for trees. The method could be extended in several ways. In particular, by using more sophisticated data structures, it may be possible to maintain efficiently the edge-based prunings of unbounded-depth trees in which the maximal paths are determined by the input sequence. More generally, the method might be applicable to maintain mixtures of other probabilistic models, such as product distributions represented by Bayesian networks.

Appendix: On the equivalence of edge-based and node-based prunings

We claimed that edge-based prunings generalize node-based prunings. Indeed, any node-based pruning of a complete K-ary tree can be simulated by an edge-based pruning that removes the K edges of every internal node that would have been made a leaf by the node-based pruning. Clearly, there are many more edge-based prunings than node-based prunings of any given template tree. To clarify the connections between the two types of prunings, we now calculate more precisely the numbers of possible prunings.

We denote by $N_n(d)$ the number of distinct node-based prunings of a complete K-ary tree of depth d, and by $N_e(d)$ the number of distinct edge-based prunings. Let $\mathcal{T}$ be a complete K-ary tree of depth d. There are K complete subtrees rooted at the children nodes of $\mathcal{T}$'s root, each with $N_n(d-1)$ distinct node-based prunings. A node-based pruning of $\mathcal{T}$ is either the root node alone or the root node together with K child subtrees each selected from the $N_n(d-1)$ possible ones. The total number of prunings is therefore

$$N_n(d) = N_n(d-1)^K + 1.$$

With edge-based prunings, there are 2^K possible subsets of edges that can be deleted from the root node (together with the subtrees they point to). Let $z_i \in \{0,1\}$ indicate whether the ith outgoing edge from the root has been deleted. Then

$$\begin{aligned} N_e(d) &= \sum_{z_1\in\{0,1\}} \sum_{z_2\in\{0,1\}} \cdots \sum_{z_K\in\{0,1\}} \prod_{i=1}^{K} N_e(d-1)^{z_i} \\ &= \prod_{i=1}^{K} \sum_{z_i\in\{0,1\}} N_e(d-1)^{z_i} = (N_e(d-1)+1)^K. \end{aligned}$$

Since $N_e(0) = N_n(0) = 1$, a simple induction using the recurrences for N_n and N_d shows that for $d > 2$

$$N_n(d) = N_e(d-1) + 1.$$

Therefore, to encode all edge-based prunings of a template tree as node-based prunings, we need to increase by 1 the depth of the template. Let $\mathcal{T}$ be the original template for edge-based prunings and $\mathcal{T}'$ the enlarged template for node-based prunings only. Define the prediction associated with a node in $\mathcal{T}'$ reached on instance $x_1 \cdots x_{n-1} x_n$ to be equal to

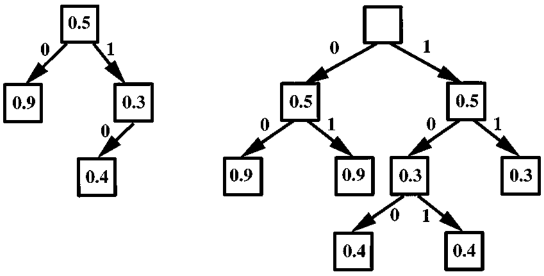

Figure 5. An illustration of the equivalence of edge-based and node based prunings: the edge-based prunings of the tree shown on the left can be encoded as node-based prunings of the tree on the right.

the prediction of the node reached on $x_1 \cdots x_{n-1}$ in $\mathcal{T}$. If we augment each instance with *any* symbol from Σ, then the set of possible edge-based prunings of $\mathcal{T}$ is equivalent to the set of possible node-based prunings excluding the root node of $\mathcal{T}'$ alone. The construction is illustrated in figure 5.

Acknowledgments

We would like to thank Dana Ron, Rob Schapire, and the anonymous reviewers for useful comments.

References

Bell, T. C., Cleary, J. G., & Witten, I. H. (1990). *Text Compression*. Englewood Cliffs, New Jersey: Prentice Hall.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression trees*. Wadsworth International Group.

Buntine, W. L. (1990). A theory of learning classification rules. Unpublished doctoral dissertation, University of Technology, Sydney.

Cesa-Bianchi, N., Freund, Y., Helmbold, D. P., Haussler, D., Schapire, R. E., & Warmuth, M. K. (1997). How to use expert advice. *Journal of the Association for Computing Machinery*, *44*(3), 427–485.

DeSantis, A., Markowsky, G., & Wegman, M. N. (1988). Learning probabilistic prediction functions. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 312–328). San Francisco, California: Morgan Kaufmann.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, *40*(3), 237–264.

Helmbold, D. P., & Schapire, R. E. (1997). Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, *27*(1), 51–68.

Jelinek, F. (1998). *Statistical methods for speech recognition*. Cambridge, Massachusetts: MIT Press.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics Specch and Signal Processing*, *35*(3), 400–401.

Krichevsky, R. E., & Trofimov, V. K. (1981). The performance of universal coding. *IEEE Transactions on Information Theory*, *27*, 199–207.

Littlestone, N., & Warmuth, M. K. (1994). The weighted majority algorithm. *Information and Computation*, *108*, 212–261.

Pereira, F. C. N., Singer, Y., & Tishby, N. (1995). Beyond word *n*-grams. In D. Yarowsky, & K. Church (Eds.), *Proceedings of the Third Workshop on Very Large Corpora* (pp. 95–106). Somerset, New Jersey: Association for Computational Linguistics.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* San Francisco, California: Morgan Kaufmann.

Rissanen, J. (1986). Complexity of strings in the class of Markov sources. *IEEE Transactions on Information Theory, 32*(4), 526–532.

Rissanen, J., & Langdon, G. G. (1981). Universal modeling and coding. *IEEE Transactions on Information Theory, IT-27*(1), 12–23.

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, *25*, 117–149.

Singer, Y. (1997). Adaptive mixtures of probabilistic transducers. *Neural Computation*, *9*(8), 1711–1733.

Vovk, V. G. (1990). Aggregating strategies. *Proceedings of the Third Annual Workshop on Computational Learning Theory* (pp. 371–383). San Francisco, California: Morgan Kaufmann.

Weinberger, M., Lempel, A., & Ziv, J. (1992). Universal coding of finite-memory sources. *IEEE Transactions on Information Theory*, *38*(3), 1002–1014.

Weinberger, M., Merhav, N., & Feder, M. (1994). Optimal sequential probability assignment for individual sequence. *IEEE Transactions on Information Theory, 40*(2), 384–396.

Weinberger, M., Rissanen, J., & Feder, M. (1995). A universal finite memory source. *IEEE Transactions on Information Theory*, *41*(3), 643–652.

Willems, F. M. J., Shtarkov, Y. M., & Tjalkens, T. J. (1995). The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, *41*(3), 653–664.

Witten, I. H., & Bell, T. C. (1991). The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, *37*(4), 1085–1094.

Received August 28, 1997
Accepted September 21, 1998
Final manuscript June 22, 1998