Title	Predicting like the best pruning of a decision tree based on the on-line DP (Algorithms and Theory of Computing)
Author(s)	Takimoto, Eiji; Maruoka, Akira; Vovk, Volodya
Citation	数理解析研究所講究録 (1998), 1041: 191-198
Issue Date	1998-04
URL	http://hdl.handle.net/2433/62054
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
1041 巻 1998 年 191-198

Predicting like the best pruning of a decision tree based on the on-line DP

Eiji Takimoto
瀧本 英二
Akira Maruoka
丸岡 章
Graduate School of Information Sciences
Tohoku University
{t2, maruoka}@ecei.tohoku.ac.jp

Volodya Vovk
Department of Computer Science
Royal Holloway, University of London
vovk@dcs.rhbnc.ac.uk

Abstract

Helmbold and Schapire gave an on-line prediction algorithm that, when given an unpruned decision tree, produces predictions not much worse than the predictions made by the best pruning of the given decision tree. In this paper, inspired by the idea that finding the best pruning can be efficiently solved by a dynamic programming in the "batch" setting where all the data to be predicted are given in advance, we give a new prediction algorithm. This algorithm works well for a wide class of loss functions, whereas the one given by Helmbold and Schapire works only for the absolute loss function. Moreover, our algorithm is so simple and general that it could be applied to many other on-line optimization problems solved by dynamic programming.

1 Introduction

Decision trees are widely used in the field of artificial intelligence and machine learning as natural ways of representing decision rules. Especially, in the computational learning theory communities, inferring a decision tree from given data is one of the most important problems, which has been widely investigated in various learning models. In particular, decision trees are shown to be PAC learnable with the aid of membership queries [1], while it remains open whether they are PAC learnable without membership queries. On the other hand, many experimental algorithms for inferring decision trees such as the C4.5 software package [13] have been proposed. Although super-polynomial lower bounds in the sense of PAC learning were proven for a wide class of algorithms including C4.5 [2], such algorithms are extensively used for various problems in machine learning because of their efficiency and simplicity. The performance of top-down learning algorithms for decision trees such as C4.5 was analyzed, and a theoretical explanation of empirically successful heuristics of these algorithms was given [10].

Many experimental algorithms mentioned above involve two phases. In the first phase, a decision tree that is consistent with the given data is constructed. The tree obtained here is typically too large resulting in "over-fitting" the data, and the performance of predicting the test data is not always satisfactory. Therefore, in the second phase, the tree is pruned by replacing some internal nodes (and associated subtrees) with leaves so as to reduce the over-fitting. It is often observed that the pruned tree, in spite of losing the consistency with the data, improves the quality of prediction.

Helmbold and Schapire gave in some sense a nearly optimal pruning algorithm [9]. In particular, they gave an on-line prediction algorithm using a given unpruned decision tree and showed that its performance will not be much worse than that of the best pruning of the given decision tree. Freund, Schapire, Singer and Warmuth extended the result to the problem of pruning a decision graph [7].

In this paper, we give a new algorithm for predicting nearly as well as the best pruning of a decision tree. Our algorithm is based on the observation that finding the best pruning can be efficiently solved by a dynamic programming in the "batch" setting where all the data to be predicted are given in advance. The performance of this *DP algorithm* is as good as that of Helmbold and Schapire's algorithm with respect to the loss bound for the absolute loss function; besides, the DP algorithm works for a wide class of loss functions, whereas Helmbold and Schapire's algorithm only works for the absolute loss function. The DP algorithm are based on the Aggregating Algorithm, which combines predictions made by several experts to make its own predictions so that the loss is not much larger than the loss of the best expert (see also [3]

and [17]). In particular, for efficient implementation of our idea, we extend the notion of the Aggregating Algorithm to have the Aggregating Pseudo-Algorithm (APA), which gives a "first approximation" to the Aggregating Algorithm by combining and generating not the "genuine" predictions but what we call the pseudopredictions. The DP algorithm assigns the APA to each node of the given tree $\mathcal{T}$ so that it combines the pseudopredictions coming from the child APAs and generates its own pseudoprediction sent to the parent APA. This recursive application of the APA is a quite straightforward implementation of the dynamic programming in that minimizing the loss at a node can be done by recursively minimizing the losses at the child nodes. Our technique is so simple and general that it could be applied to many other on-line optimization problems solved by dynamic programming.

2 Preliminaries

A decision tree $\mathcal{T}$ is a rooted tree where every node u is labeled with an element $V(u)$ of a set $\hat{Y}$ called the *prediction space*. Assume that there is an instance space and each instance x induces a path from the root to a leaf of $\mathcal{T}$. The path is denoted by $\mathrm{path}(x)$ and the leaf l in $\mathrm{path}(x)$ is denoted by $l = \mathrm{leaf}_{\mathcal{T}}(x)$. Then, $\mathcal{T}$ defines a decision rule that maps each instance x to the prediction $V(\mathrm{leaf}_{\mathcal{T}}(x))$. A *pruning* $\mathcal{P}$ of the decision tree $\mathcal{T}$ is a tree obtained by replacing zero or more of the internal nodes (and associated subtrees) of $\mathcal{T}$ by leaves. The pruned tree $\mathcal{P}$ induces a decision tree that for instance x makes its prediction $V(\mathrm{leaf}_{\mathcal{P}}(x))$. The set of all prunings of $\mathcal{T}$ is denoted by $\mathrm{PRUN}(\mathcal{T})$.

We study learning in the on-line prediction model, where an algorithm is required not to actually prune the tree but to make predictions for a given instance sequences based on a given decision tree $\mathcal{T}$. The goal is to make predictions that are competitive with those made by the best pruning of $\mathcal{T}$. In essence, this is the framework introduced by Littlestone and Warmuth [11, 12] and developed further by many researchers in various settings [4, 5, 6, 8, 15, 16]. Below we state the model in a general form. A prediction algorithm A is given as input a template tree $\mathcal{T}$. At each trial $t = 1, 2, \ldots$, algorithm A receives an instance x_t and generates a prediction $\hat{y}_t \in \hat{Y}$. After that, an outcome $y_t \in Y$ is observed (which can be thought of as the correct classification of x_t), where Y is a set called the *outcome space*. At this trial, the algorithm A suffers loss $\lambda(y_t, \hat{y}_t)$, where $\lambda : Y \times \hat{Y} \to [0, \infty]$ is a fixed *loss function*. We will call the triple $(Y, \hat{Y}, \lambda)$ our *game*. One of the most popular games is the absolute-loss game where $Y = \{0, 1\}$, $\hat{Y} = [0, 1]$ and $\lambda(y, \hat{y}) = |y - \hat{y}|$. Our algorithm works for a wide class of games including the absolute-loss game, whereas Helmbold and Schapire's algorithm only works for the absolute-loss game. The (cumulative) loss of A for outcome sequence $y = (y_1, \ldots, y_T) \in Y^*$, denoted $L_A(y)$, is defined as

$$L_A(y) = \sum_{t=1}^{T} \lambda(y_t, \hat{y}_t).$$

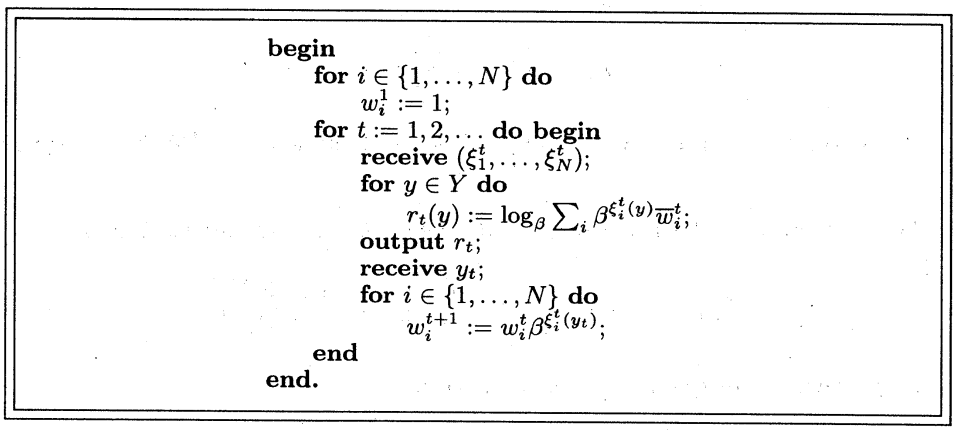
Similarly, for a pruning $\mathcal{P}$ of $\mathcal{T}$, the loss of $\mathcal{P}$ for $y \in Y^T$ is defined as

$$L_{\mathcal{P}}(y) = \sum_{t=1}^{T} \lambda(y_t, V(\mathrm{leaf}_{\mathcal{P}}(x_t))).$$

The performance of A is measured by the relative loss compared with the loss of the optimal $\mathcal{P}$ for the given instance and outcome sequences x and y.

3 Aggregating Algorithm

Helmbold and Schapire's algorithm and our algorithm are based on the *Aggregating Algorithm* (AA for short), which is a master algorithm that combines the predictions made by several experts and makes its own predictions so that the loss is not much larger than the loss of the best expert [15]. The AA can be conveniently defined in terms of the *Aggregating Pseudo-Algorithm*, or APA (cf. [17]), which is a "first approximation" to the AA generating not "genuine" predictions $\hat{y} \in \hat{Y}$ but what we call *pseudopredictions*. (Explicit consideration of the APA is essential for efficient implementation of our idea inspired by dynamic programming.) The pseudoprediction is then transformed to a genuine prediction



```
begin
    for i ∈ {1,...,N} do
        w_i^1 := 1;
    for t := 1,2,... do begin
        receive (ξ_1^t,...,ξ_N^t);
        for y ∈ Y do
            r_t(y) := log_β Σ_i β^{ξ_i^t(y)} w̄_i^t;
        output r_t;
        receive y_t;
        for i ∈ {1,...,N} do
            w_i^{t+1} := w_i^t β^{ξ_i^t(y_t)};
    end
end.
```

Figure 1: APA(β)

by the *substitution function*. In this paper, we extend the notion of the APA so that it combines not only the genuine predictions but also the pseudopredictions. This enables us to apply the APA recursively so that an APA combines the pseudopredictions generated by the child APAs and generates its own pseudoprediction that will be passed to the parent APA. In this section, we describe the AA using the notion of the APA and the substitution function.

First we extend the notion of a (genuine) prediction so that every prediction $\xi \in \hat{Y}$ is identified with the function on the outcome space Y whose value $\xi(y)$ for $y \in Y$ equals the loss $\lambda(y, \xi)$ suffered by this prediction when the true outcome is y. Here we are using ξ to denote two different objects: the prediction in $\hat{Y}$ and the function on Y describing the potential losses suffered by this prediction. More generally, we define a pseudoprediction ξ to be an arbitrary function that maps every $y \in Y$ to a non-negative real values $\xi(y)$, which is interpreted as the loss of ξ for outcome y. Note that a genuine prediction $\xi \in \hat{Y}$ is a special form of a pseudoprediction, i.e., $\xi(y) = \lambda(y, \xi)$.

Assume that there are N experts $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_N\}$. The APA combining the experts $\mathcal{E}$ behaves as follows. It maintains a weight $w_i^t \in [0, 1]$ for each expert $\mathcal{E}_i$ that reflects the actual performance of the expert $\mathcal{E}_i$ until time t. Initially (at time $t = 1$), some positive weights $w_i^t = w_i^1$ are assigned to the experts (often it is convenient not to assume that the weights are normalized). At each trial t, every expert $\mathcal{E}_i$ makes a pseudoprediction ξ_i^t for the given instance x_t, and gives it to the APA. Then, the APA outputs the pseudoprediction r_t which is a "weighted average" of ξ_i^t,

$$r_t(y) = \log_\beta \sum_{i=1}^{N} \beta^{\xi_i^t(y)} \overline{w}_i^t, \qquad y \in Y \tag{1}$$

(recall that $\xi_i^t(y) = \lambda(y, \xi_i^t)$ if ξ_i^t is a genuine prediction), where $\overline{w}_i^t$ are the normalized weights:

$$\overline{w}_i^t = \frac{w_i^t}{\sum_{i=1}^{N} w_i^t}.$$

Here, $\beta \in]0, 1[$ is the parameter of the APA, and so the APA will sometimes be written as APA(β) in order to explicitly specify β. After receiving the correct classification $y_t \in Y$, the APA updates the weights according to the rule $w_i^{t+1} = w_i^t \beta^{\xi_i^t(y_t)}$ (so the larger the expert $\mathcal{E}_i$'s loss is, the more its weight decreases). The initial weight w_i^t of the experts are given to the APA in advance and are not necessarily equal. Summarizing, we give in Figure 1 the description of APA(β) (with the equal initial weights).

The loss of $\mathcal{E}_i$ and that of APA(β) for outcome sequence $y \in Y^T$ are denoted by

$$L_i(y) = \sum_{t=1}^{T} \xi_i^t(y_t)$$

and

$$L_{\mathrm{APA}(\beta)}(y) = \sum_{t=1}^{T} r_t(y_t),$$

respectively. The following theorem gives an upper bound of the loss of APA(β) in terms of the loss of the best expert.

Theorem 1 ([17]; implicitly [15]) *Let $0 < \beta < 1$. Then, for any N experts $\mathcal{E}$ and for any $y \in Y^*$,*

$$L_{\mathrm{APA}(\beta)}(y) \le \min_{i \in \{1,\dots,N\}} \left(L_i(y) + \frac{\ln(1/\overline{w}_i^1)}{\ln(1/\beta)} \right),$$

where $\overline{w}_i^1$ are the normalized initial weights of the experts.

Of course, the above theorem would say nothing if no restrictions on pseudopredictions are imposed, because it is too easy to achieve the ideal performance for an algorithm that is allowed to always make the identical 0 as its own pseudopredictions. However, if all the experts generates genuine predictions, the APA outputs pseudopredictions that are mixtures, in some sense, of them, and the loss bound for the APA implies the loss bound for the AA given by Theorem 2 below.

Let us say that a pseudoprediction r is a *β-mixture* if

$$r(y) = \log_\beta \sum_{i=1}^{n} \beta^{\lambda(y,\xi_i)} v_i$$

for some n, some predictions $\xi_i \in \hat{Y}$, $i = 1, \dots, n$, and some normalized weights $v_i > 0$, $\sum_{i=1}^n v_i = 1$. Notice that if all experts always output genuine predictions, the APA will always output mixtures. More generally, the next lemma says that if all experts always output β-mixtures, the APA will always output β-mixtures as well.

Lemma 1 *Let $\xi_1, \dots, \xi_n$ be any β-mixtures and let $v_1, \dots, v_n$ be any non-negative normalized weights, $\sum_{i=1}^n v_i = 1$. Then, the function r defined as*

$$r(y) = \log_\beta \sum_{i=1}^{n} \beta^{\xi_i(y)} v_i$$

is a β-mixture.

Now we define the *mixability curve* $c(\beta)$ of the game $(Y, \hat{Y}, \lambda)$. For any $\beta \in]0,1[$, we put

$$c(\beta) = \sup_r \min_{\hat{y} \in \hat{Y}} \sup_{y \in Y} \frac{\lambda(y, \hat{y})}{r(y)},$$

where r ranges over all β-mixtures[1]. Generally, there is no β satisfying $c(\beta) < 1$. For the absolute-loss game, for example, the mixability curve is $c(\beta) = \ln(1/\beta)/2\ln(2/(1+\beta))$, which is greater than 1 for every β. However, many important games including the square-loss game, the log-loss game, etc., one can choose β so that $c(\beta) = 1$. A *β-substitution function* is a function Σ_β that maps every β-mixture r to a prediction $\Sigma_\beta(r) \in \hat{Y}$ which satisfies for any $y \in Y$

$$\lambda(y, \Sigma_\beta(r)) \le c(\beta) r(y). \tag{2}$$

Not that by the definition of the mixability curve, we can always find a substitution function.

Now we are ready to describe the AA (although it will not be used in this paper). The AA behaves just like the APA except that the AA outputs not the pseudoprediction r_t but the genuine prediction

[1]Strictly, we have to put some assumptions on the game $(Y, \hat{Y}, \lambda)$ in order to claim that such the value $c(\beta)$ exists. The assumptions are so natural that they are satisfied by many popular games. For further details, see [17].

$\hat{y}_t = \Sigma_\beta(r_t)$. Suppose that all experts always outputs β-mixtures. Then, Lemma 1 and inequality (2) says that the loss of the AA for $y \in Y^T$, denoted $L_{\mathrm{AA}(\beta)}(y)$, is upper bounded by

$$L_{\mathrm{AA}(\beta)}(y) = \sum_{t=1}^{T} \lambda(y, \Sigma_\beta(r_t)) \le c(\beta) \sum_{t=1}^{T} r_t(y_t) = c(\beta) L_{\mathrm{APA}(\beta)}(y).$$

Therefore, Theorem 1 implies the upper bound of the loss of the AA.

Theorem 2 ([15]) *Let $0 < \beta < 1$. Then, for any N experts $\mathcal{E}$ that generate β-mixtures and for any $y \in Y^*$,*

$$L_{\mathrm{AA}(\beta)}(y) \le c(\beta) \min_{i \in \{1,\ldots,N\}} \left(L_i(y) + \frac{\ln(1/\overline{w}_i^1)}{\ln(1/\beta)} \right). \tag{3}$$

Note that since the genuine prediction $\xi \in \hat{Y}$ is a β-mixture (indeed, we have $\xi(y) = \log_\beta \sum_{i=1}^{n} \beta^{\lambda(y,\xi_i)} v_i$ with $n = 1$, $\xi_1 = \xi$ and $v_1 = 1$), the AA combines and generates the genuine predictions so that the loss is upper bounded by inequality (3).

4 A prediction algorithm based on dynamic programming

In this section, we give a new prediction algorithm that performs nearly as well as the best pruning of a decision tree. The idea of our algorithm comes from the fact that the best pruning for a given instance sequence x and outcome sequence y can be effectively computed by a dynamic programming provided that x and y are known in advance.

For a node u, let $\mathcal{T}_u$ denote the subtree of $\mathcal{T}$ rooted at u, and let $\mathrm{child}(u)$ denote the set of child nodes of u. More precisely, $\mathcal{T}_u$ is defined inductively as follows: If u is a leaf of $\mathcal{T}$, then $\mathcal{T}_u$ is the leaf u itself, and if u is an internal node of $\mathcal{T}$, then $\mathcal{T}_u$ is the tree whose root u is connected with the subtrees $\mathcal{T}_v$ for $v \in \mathrm{child}(u)$. Let R denote the root of $\mathcal{T}$. Note that we have $\mathcal{T}_{\mathrm{R}} = \mathcal{T}$. For an outcome sequence $y \in Y^*$, the loss suffered at u is denoted by $L_u(y)$. That is,

$$L_u(y) = \sum_{t : u \in \mathrm{path}(x_t)} \lambda(y_t, V(u)).$$

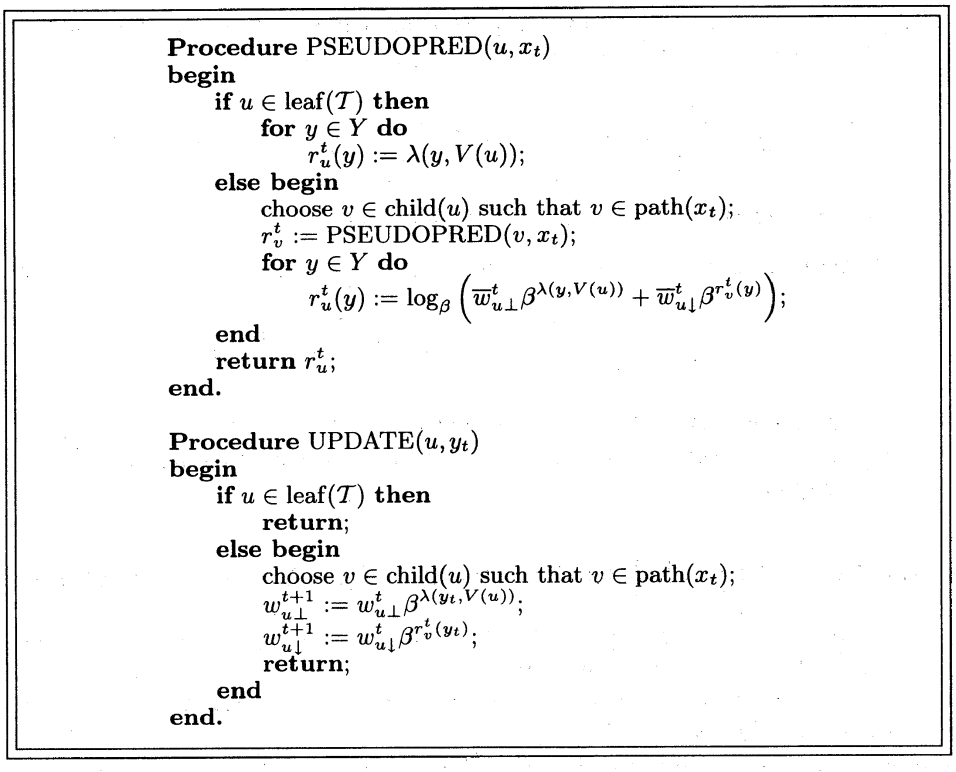
Then, for any pruning $\mathcal{P}_u$ of $\mathcal{T}_u$, the loss suffered by $\mathcal{P}_u$, denoted $L_{\mathcal{P}_u}(y)$, can be represented by the sum of $L_l(y)$ for all leaves l of $\mathcal{P}_u$. In other words, we can write $L_{\mathcal{P}_u}(y) = L_u(y)$ if $\mathcal{P}_u$ consists of a single leaf u and $L_{\mathcal{P}_u}(y) = \sum_{v \in \mathrm{child}(u)} L_{\mathcal{P}_v}(y)$ otherwise. Here, $\mathcal{P}_v$ is the subtree of $\mathcal{P}_u$ rooted at v, and it is also a pruning of $\mathcal{T}_v$. Since the losses $L_{\mathcal{P}_v}(y)$ for $v \in \mathrm{child}(u)$ are independent of each other, we can minimize the loss $L_{\mathcal{P}_u}(y)$ by minimizing each loss $L_{\mathcal{P}_v}(y)$ independently. Therefore, we have for any internal node u of $\mathcal{T}$,

$$\min_{\mathcal{P}_u \in \mathrm{PRUN}(\mathcal{T}_u)} L_{\mathcal{P}_u}(y) = \min \left\{ L_u(y), \sum_{v \in \mathrm{child}(u)} \min_{\mathcal{P}_v \in \mathrm{PRUN}(\mathcal{T}_v)} L_{\mathcal{P}_v}(y) \right\}. \tag{4}$$

Since dynamic programming can be applied to solve the minimization problem above, we can efficiently compute $\mathcal{P}_{\mathrm{R}}$ that minimizes $L_{\mathcal{P}_{\mathrm{R}}}(y)$, which is the best pruning of $\mathcal{T}$.

Now we construct an on-line version of the dynamic programming. The key idea is to associate the APA and two mini-experts $\mathcal{E}_u = \{\mathcal{E}_{u\perp}, \mathcal{E}_{u\downarrow}\}$ with each internal node u of $\mathcal{T}$, one $\mathcal{E}_{u\perp}$ generating $V(u)$ and the other $\mathcal{E}_{u\downarrow}$ generating the pseudoprediction of the subtrees below u, and to apply the APA at u, denoted $\mathrm{APA}_u(\beta)$, that combines these two pseudopredictions to obtain its own pseudoprediction r_u^t at u. Recall that the genuine prediction $V(u)$ is regarded as a pseudoprediction. More precisely, when given an instance x_t that goes through u and $v \in \mathrm{child}(u)$, the second expert $\mathcal{E}_{u\downarrow}$ generates r_v^t, i.e., the pseudoprediction made by $\mathrm{APA}_v(\beta)$, the APA at node v. Then, taking the weighted average of $V(u)$ and r_v^t according to equation (1), $\mathrm{APA}_u(\beta)$ obtains the pseudoprediction r_u^t at u. To obtain the genuine prediction $\hat{y}_t$, our algorithm applies the β-substitution function to the pseudoprediction at the root, that is, $\hat{y}_t = \Sigma_\beta(r_{\mathrm{R}}^t)$.

The algorithm using the AA instead of the APA was analyzed in the conference version [14] of this paper. However, as seen in the previous section, application of the substitution function turned out to



Procedure PSEUDOPRED(u, x_t)
begin
  if $u \in \mathrm{leaf}(\mathcal{T})$ **then**
    for $y \in Y$ **do**
      $r_u^t(y) := \lambda(y, V(u))$;
  else begin
    choose $v \in \mathrm{child}(u)$ such that $v \in \mathrm{path}(x_t)$;
    $r_v^t :=$ PSEUDOPRED(v, x_t);
    for $y \in Y$ **do**
      $r_u^t(y) := \log_\beta \left(\overline{w}_{u\perp}^t \beta^{\lambda(y,V(u))} + \overline{w}_{u\downarrow}^t \beta^{r_v^t(y)} \right)$;
  end
  return r_u^t;
end.

Procedure UPDATE(u, y_t)
begin
  if $u \in \mathrm{leaf}(\mathcal{T})$ **then**
    return;
  else begin
    choose $v \in \mathrm{child}(u)$ such that $v \in \mathrm{path}(x_t)$;
    $w_{u\perp}^{t+1} := w_{u\perp}^t \beta^{\lambda(y_t,V(u))}$;
    $w_{u\downarrow}^{t+1} := w_{u\downarrow}^t \beta^{r_v^t(y_t)}$;
    return;
  end
end.

Figure 2: $\mathrm{APA}_u(\beta)$

be the most inefficient step in the AA (it leads to multiplying the loss bound by $c(\beta)$), we perform it only once during every trial (at the very end of the trial); in the internal nodes we combine not genuine predictions but pseudopredictions using the APA.

We give the APA at node u in Figure 2 and the prediction algorithm $\mathrm{DP}(\beta)$ that controls the APAs in Figure 3.

Let the loss suffered by $\mathrm{APA}_u(\beta)$ be denoted $\hat{L}_u(y)$. That is,

$$\hat{L}_u(y) = \sum_{t:u\in\mathrm{path}(x_t)} r_u^t(y_t).$$

Since the first expert $\mathcal{E}_{u\perp}$ suffers the loss $L_u(y)$ and the second expert $\mathcal{E}_{u\downarrow}$ suffers the loss $\sum_{v\in\mathrm{child}(u)} \hat{L}_v(y)$, Theorem 1 says that for any internal node u of $\mathcal{T}$,

$$\hat{L}_u(y) \le \min\left\{ L_u(y), \sum_{v\in\mathrm{child}(u)} \hat{L}_v(y) \right\} + (\ln 2)/(\ln(1/\beta)). \tag{5}$$

By the similarity of equation (4) and inequality (5), we can roughly say that $\hat{L}_u(y)$ is not much larger than the loss of the best pruning of $\mathcal{T}_u$. More precisely, applying inequality (5) recursively, we obtain the following upper bound on the loss $\hat{L}_{\mathrm{R}}(y)$ at the root:

$$\hat{L}_{\mathrm{R}}(y) \le L_{\mathcal{P}}(y) + |\mathcal{P}|(\ln 2)/(\ln(1/\beta)),$$

for any pruning $\mathcal{P} \in \mathrm{PRUN}(\mathcal{T})$, where $|\mathcal{P}|$ denotes the number of the nodes of $\mathcal{P}$ that are not the leaf of $\mathcal{T}$. (Indeed, every node of $\mathcal{P}$ which is not $\mathcal{T}$'s leaf gives an extra loss of $(\ln 2)/(\ln(1/\beta))$.) Since every APA in the algorithm combines and generates β-mixtures, Lemma 1 says that the pseudoprediction ξ_{R}^t at the root is always a β-mixture. Therefore, inequality (2) gives the loss bound of the DP that satisfies

$$L_{\mathrm{DP}(\beta)}(y) \le c(\beta)\left(L_{\mathcal{P}}(y) + |\mathcal{P}|(\ln 2)/(\ln(1/\beta))\right),$$

```
input T;

begin
    for u ∈ nodes(T)\leaves(T) do begin
        w^t_{u⊥} := 1;
        w^t_{u↓} := 1;
    end
    for t = 1, 2, ... do begin
        receive x_t;
        r^t_R := PSEUDOPRED(R, x_t);
        ŷ_t := Σ_β(r^t_R);
        output ŷ_t;
        receive y_t;
        for u ∈ path(x_t) do
            UPDATE(u, y_t);
    end
end.
```

Figure 3: Algorithm DP(β)

for all $\mathcal{P} \in \mathrm{PRUN}$ and $y \in Y^*$.

For estimating the time spent by algorithm DP(β), we have to consider the explicit representation of the β-mixture and the computational feasibility of the β-substitution function. For this purpose, it may be convenient to identify a β-mixture $r = \log_\beta \sum_{i=1}^n \beta^{\xi_i} v_i$ as the probability distribution $((\xi_1, v_1), \ldots, (\xi_n, v_n))$ that induces r. Using this representation, we can say that Σ_β is, say, linearly computable at β if there exists an algorithm that computes each value $\Sigma_\beta((\xi_1, v_1), \ldots, (\xi_n, v_n))$ in time $O(n)$. For example, many popular games including the absolute-loss game, the square-loss game, the log-loss game, etc., their substitution functions are all linearly computable at appropriate choices of β. Now, we have our main theorem.

Theorem 3 (Main) *Let $\beta \in]0,1[$ and $c(\beta)$ be the value of the mixability curve at β for the game $(Y, \hat{Y}, \lambda)$. Then, for any $\mathcal{T}$ and $y \in Y^*$, when given $\mathcal{T}$, DP(β) makes predictions for y so that the loss is at most*

$$L_{\mathrm{DP}(\beta)}(y) \le c(\beta) \min_{\mathcal{P} \in \mathrm{PRUN}(\mathcal{T})} \left(L_{\mathcal{P}}(y) + \frac{\ln 2}{\ln(1/\beta)} |\mathcal{P}| \right). \tag{6}$$

Moreover, if Σ_β is linearly computable at β, A generates a prediction at each trial t in time $O(|x_t|)$.

References

[1] N. Bshouty, Exact learning via the monotone theory, *Inform. Computation* **123** (1995) 146–153.

[2] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour and S. Rudich, Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis, in: *Proc. 26th STOC* (1994) 253–262.

[3] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire and M. Warmuth, How to use expert advice, in: *Proc. 25th STOC* (1993) 382–391.

[4] T. Cover and E. Ordentlich, Universal portfolios with side information, *IEEE Trans. Inform. Theory* **42** (1996) 348–363.

[5] A. DeSantis, G. Markowsky, and M. N. Wegman, Learning probabilistic prediction functions, in: *Proc. 29th FOCS* (1988) 110–119.

[6] Y. Freund and R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: Lecture Notes in Computer Science, Vol. 904 (1995) 23–37. To appear in *J. Computer System Sciences.*

[7] Y. Freund, R. Schapire, Y. Singer and M. Warmuth, Using and combining predictors that specialize, in: *Proc. 29th STOC* (1997).

[8] D. Haussler, J. Kivinen and M. K. Warmuth, Tight worst-case loss bounds for predicting with expert advice, Technical Report UCSC-CRL-94-36, University of California, Santa Cruz, CA, revised December 1994. Short version in: Lecture Notes in Computer Science, Vol. 904 (1995).

[9] D. Helmbold and R. Schapire, Predicting nearly as well as the best pruning of a decision tree, in: *Proc. 8th COLT* (1995) 61–68.

[10] M. Kearns and Y. Mansour, On the boosting ability of top-down decision tree learning algorithms, in: *Proc. 28th STOC* (1996) 459–468.

[11] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, *Machine Learning* **2** (1988) 285–318.

[12] N. Littlestone and M. K. Warmuth, The weighted majority algorithm, *Inform. Computation* **108** (1994) 212–261.

[13] J. Quinlan. *C4.5: Programs for Machine Learning* (Morgan Kaufmann, 1993).

[14] E. Takimoto, K. Hirai and A. Maruoka, A simple algorithm for predicting nearly as well as the best pruning labeled with the best prediction values of a decision tree, in: Lecture Notes in Artificial Intelligence, Vol. 1316 (1997) 385–400.

[15] V. Vovk, Aggregating strategies, in: *Proc. 3rd COLT* (1990) 371–383.

[16] V. Vovk, Universal forecasting algorithms, *Inform. Computation* **96** (1992) 245–277.

[17] V. Vovk, A game of prediction with expert advice, accepted for publication in *J. Comput. Inform. Syst.* Short version in: *Proc. 8th COLT* (1995) 51–60.