Appearing in the proceedings of the
Second European Conference on Computational Learning Theory, March, 1995.

A decision-theoretic generalization of on-line learning and an application to boosting

Yoav Freund Robert E. Schapire

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974-0636
{yoav, schapire}@research.att.com

Abstract. We consider the problem of dynamically apportioning resources among a set of options in a worst-case on-line framework. The model we study can be interpreted as a broad, abstract extension of the well-studied on-line prediction model to a general decision-theoretic setting. We show that the multiplicative weight-update rule of Littlestone and Warmuth [10] can be adapted to this model yielding bounds that are slightly weaker in some cases, but applicable to a considerably more general class of learning problems. We show how the resulting learning algorithm can be applied to a variety of problems, including gambling, multiple-outcome prediction, repeated games and prediction of points in $\mathbb{R}^n$. We also show how the weight-update rule can be used to derive a new boosting algorithm which does not require prior knowledge about the performance of the weak learning algorithm.

1 Introduction

A gambler, frustrated by persistent horse-racing losses and envious of his friends' winnings, decides to allow a group of his fellow gamblers to make bets on his behalf. He decides he will wager a fixed sum of money in every race, but that he will apportion his money among his friends based on how well they are doing. Certainly, if he knew psychically ahead of time which of his friends would win the most, he would naturally have that friend handle all his wagers. Lacking such clairvoyance, however, he attempts to allocate each race's wager in such a way that his total winnings for the season will be reasonably close to what he would have won had he bet everything with the luckiest of his friends.

In this paper, we describe a simple algorithm for solving such dynamic allocation problems, and we show that our solution can be applied to a great assortment of learning problems. Perhaps the most surprising of these applications is the derivation of a new algorithm for "boosting," i.e., for converting a "weak" PAC learning algorithm that performs just slightly better than random guessing into one with arbitrarily high accuracy.

We formalize our *on-line allocation model* as follows. The allocation agent A has N options or *strategies* to choose from; we number these using the integers $1, \ldots, N$. At each time step $t = 1, 2, \ldots, T$, the allocator A decides on a distribution $\mathbf{p}^t$ over the strategies; that is $p_i^t \geq 0$ is the amount allocated to strategy i, and $\sum_{i=1}^N p_i^t = 1$. Each strategy i then suffers some *loss* ℓ_i^t which is determined by the (possibly adversarial) "environment." The loss suffered by A is then $\sum_{i=1}^N p_i^t \ell_i^t = \mathbf{p}^t \cdot \boldsymbol{\ell}^t$, i.e., the average loss of the strategies with respect to A's chosen allocation rule. We call this loss function the *mixture loss*.

In this paper, we always assume that the loss suffered by any strategy is bounded so that, without loss of generality, $\ell_i^t \in [0,1]$. Besides this condition, we make no assumptions about the form of the loss vectors $\boldsymbol{\ell}^t$, or about the manner in which they are generated; indeed, the adversary's choice for $\boldsymbol{\ell}^t$ may even depend on the allocator's chosen mixture $\mathbf{p}^t$.

The goal of the algorithm A is to minimize its cumulative loss relative to the loss suffered by the best strategy. That is, A attempts to minimize its *net loss*

$$L_A - \min_i L_i$$

where

$$L_A = \sum_{t=1}^{T} \mathbf{p}^t \cdot \boldsymbol{\ell}^t$$

is the total cumulative loss suffered by algorithm A on the first T trials, and

$$L_i = \sum_{t=1}^{T} \ell_i^t$$

is strategy i's cumulative loss.

In Section 2, we show that Littlestone and Warmuth's [10] "weighted majority" algorithm can be generalized to handle this problem, and we prove a number of bounds on the net loss. For instance, one of our results shows that the net loss of our algorithm can be bounded by $O\left(\sqrt{T \ln N}\right)$ or, put another way, that the average per trial net loss is decreasing at the rate $O\left(\sqrt{(\ln N)/T}\right)$. Thus, as T increases, this difference decreases to zero.

Our results for the on-line allocation model can be applied to a wide variety of learning problems, as we describe in Section 3. In particular, we generalize the results of Littlestone and Warmuth [10] and Cesa-Bianchi et al. [1] for the problem of predicting a binary sequence using the advice of a team of "experts." Whereas these authors proved worst-case bounds for making on-line randomized decisions over a binary decision and outcome space with a $\{0,1\}$-valued discrete loss, we prove (slightly weaker) bounds that are applicable to any bounded loss function over any decision and outcome spaces. Our bounds express explicitly the rate at which the loss of the learning algorithm approaches that of the best expert.

Related generalizations of the expert prediction model were studied by Vovk [12], Kivinen and Warmuth [9], and Haussler, Kivinen and Warmuth [8]. Like us, these authors focused primarily on multiplicative weight-update algorithms. Chung [2] also presented a generalization, giving the problem a game-theoretic treatment.

Finally, in Section 4, we show how a similar algorithm can be used for boosting, i.e., for converting any weak PAC learning algorithm into a strong PAC learning algorithm. Unlike the previous boosting algorithms of Freund [5, 6] and Schapire [11], the new algorithm needs no prior knowledge of the accuracy of the hypotheses of the weak learning algorithm. Rather, it adapts to the accuracies of the generated hypotheses and generates a weighted majority hypothesis in which the weight of each weak hypothesis is a function of its accuracy. The accuracy of this final hypothesis improves when *any* of the weak hypotheses is improved. This is in contrast with previous boosting algorithms whose performance bound depended only on the accuracy of the least accurate weak hypothesis. At the same time, if the weak hypotheses all have the same accuracy, the performance of the new algorithm is very close to that achieved by the best of the known boosting algorithms.

2 The algorithm and its analysis

In this section, we present our algorithm, called **Hedge**(β), for the on-line allocation problem. The algorithm and its analysis are direct generalizations of Littlestone and Warmuth's weighted majority algorithm [10].

The algorithm maintains a weight vector whose value at time t is denoted $\mathbf{w}^t = \langle w_1^t, \ldots, w_N^t \rangle$. At all times, all weights will be nonnegative. All of the weights of the initial weight vector $\mathbf{w}^1$ must be nonnegative and sum to one, so that $\sum_{i=1}^N w_i^1 = 1$. Besides these conditions, the initial weight vector may be arbitrary, and may be viewed as a "prior" over the set of strategies. Since our bounds are strongest for those strategies receiving the greatest initial weight, we will want to choose the initial weights so as to give the most weight to those strategies which we expect are most likely to perform the best. Naturally, if we have no reason to favor any of the strategies, we can set all of the initial weights equally so that $w_i^1 = 1/N$. Note that the weights on future trials need not sum to one.

Our algorithm allocates among the strategies using the current weight vector, after normalizing. That is, at time t, **Hedge**(β) chooses the distribution vector

$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}. \tag{1}$$

After the loss vector $\boldsymbol{\ell}^t$ has been received, the weight vector $\mathbf{w}^t$ is updated using the multiplicative rule

$$w_i^{t+1} = w_i^t \cdot U_\beta(\ell_i^t). \tag{2}$$

Here, $U_\beta : [0,1] \to [0,1]$ is any function, parameterized by $\beta \in [0,1]$, which satisfies

$$\beta^r \le U_\beta(r) \le 1 - (1-\beta)r. \tag{3}$$

(It can be shown that such a value $U_\beta(r)$ always exists for any β and r in the stated ranges [10].)

2.1 Analysis

The analysis of **Hedge**(β) mimics directly that given by Littlestone and Warmuth. The main idea is to derive upper and lower bounds on $\sum_{i=1}^N w_i^{T+1}$ which, together, imply an upper bound on the loss of the algorithm. We begin with an upper bound.

Lemma 1. *For any sequence of loss vectors $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^T$, we have*

$$\ln\left(\sum_{i=1}^{N} w_i^{T+1}\right) \leq -(1-\beta) L_{\mathbf{Hedge}(\beta)}.$$

Proof. For $t = 1, \ldots, T$, we have from Equations (1), (2) and (3) that

$$\sum_{i=1}^{N} w_i^{t+1} = \sum_{i=1}^{N} w_i^t U_\beta(\ell_i^t) \leq \sum_{i=1}^{N} w_i^t (1-(1-\beta)\ell_i^t) = \left(\sum_{i=1}^{N} w_i^t\right)(1-(1-\beta)\mathbf{p}^t \cdot \boldsymbol{\ell}^t). \tag{4}$$

Therefore,

$$\ln\left(\sum_{i=1}^{N} w_i^{t+1}\right) \leq \ln\left(\sum_{i=1}^{N} w_i^t\right) + \ln(1-(1-\beta)\mathbf{p}^t \cdot \boldsymbol{\ell}^t) \leq \ln\left(\sum_{i=1}^{N} w_i^t\right) - (1-\beta)\mathbf{p}^t \cdot \boldsymbol{\ell}^t. \tag{5}$$

It follows that

$$\ln\left(\sum_{i=1}^{N} w_i^{T+1}\right) \leq \ln\left(\sum_{i=1}^{N} w_i^1\right) - (1-\beta)\sum_{t=1}^{T} \mathbf{p}^t \cdot \boldsymbol{\ell}^t = -(1-\beta) L_{\mathbf{Hedge}(\beta)}.$$

□

Thus,

$$L_{\mathbf{Hedge}(\beta)} \leq \frac{-\ln\left(\sum_{i=1}^{N} w_i^{T+1}\right)}{1-\beta}. \tag{6}$$

Note that, from Equations (2) and (3),

$$w_i^{T+1} = w_i^1 \prod_{t=1}^{T} U_\beta(\ell_i^t) \geq w_i^1 \beta^{\sum_{t=1}^{T} \ell_i^t} = w_i^1 \beta^{L_i}. \tag{7}$$

This is all that is needed to complete our analysis.

Theorem 2. *For any sequence of loss vectors $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^T$, and for any $i \in \{1, \ldots, N\}$, we have*

$$L_{\mathbf{Hedge}(\beta)} \leq \frac{-\ln(w_i^1) - L_i \ln\beta}{1-\beta}. \tag{8}$$

More generally, for any nonempty set $S \subseteq \{1, \ldots, N\}$, we have

$$L_{\mathbf{Hedge}(\beta)} \leq \frac{-\ln(\sum_{i\in S} w_i^1) - (\ln\beta)\max_{i\in S} L_i}{1-\beta}. \tag{9}$$

Proof. We prove the more general statement (9) since Equation (8) follows in the special case that $S = \{i\}$.

From Equation (7),

$$\sum_{i=1}^{N} w_i^{T+1} \geq \sum_{i\in S} w_i^{T+1} \geq \sum_{i\in S} w_i^1 \beta^{L_i} \geq \beta^{\max_{i\in S} L_i} \sum_{i\in S} w_i^1.$$

The theorem now follows immediately from Equation (6). □

The simpler bound (8) states that **Hedge**(β) does not perform "too much worse" than the best strategy i for the sequence. The difference in loss depends on our choice of β and on the initial weight w_i^1 of each strategy. If each weight is set equally so that $w_i^1 = 1/N$, then this bound becomes

$$L_{\textbf{Hedge}(\beta)} \leq \frac{\min_i L_i \ln(1/\beta) + \ln N}{1-\beta}. \tag{10}$$

Since it depends only logarithmically on N, this bound is reasonable even for a very large number of strategies.

The more complicated bound (9) is a generalization of the simpler bound that is especially applicable when the number of strategies is infinite. Naturally, for uncountable collections of strategies, the sum appearing in Equation (9) can be replaced by an integral, and the maximum by a supremum.

2.2 How to choose β

So far, we have analyzed **Hedge**(β) for a given choice of β, and we have proved reasonable bounds for any choice of β. In practice, we will often want to choose β so as to maximally exploit any prior knowledge we may have about the specific problem at hand.

The following lemma will be helpful for choosing β using the bounds derived above.

Lemma 3. *Suppose* $0 \leq L \leq \tilde{L}$ *and* $0 < R \leq \tilde{R}$. *Let* $\beta = g(\tilde{L}/\tilde{R})$ *where* $g(z) = 1/(1+\sqrt{2/z})$. *Then*

$$\frac{-L \ln \beta + R}{1-\beta} \leq L + \sqrt{2\tilde{L}\tilde{R}} + R.$$

Proof. (Sketch) It can be shown that $-\ln \beta \leq (1-\beta^2)/(2\beta)$ for $\beta \in (0,1]$. Applying this approximation and the given choice of β yields the result. □

Lemma 3 can be applied to any of the bounds above since all of these bounds have the form given in the lemma. For example, suppose we have N strategies, and we also know a prior bound $\tilde{L}$ on the loss of the best strategy. Then, combining Equation (10) and Lemma 3, we have

$$L_{\textbf{Hedge}(\beta)} \leq \min_i L_i + \sqrt{2\tilde{L} \ln N} + \ln N \tag{11}$$

for $\beta = g(\tilde{L}/\ln N)$. In general, if we know ahead of time the number of trials T, then we can use $\tilde{L} = T$ as an upper bound on the cumulative loss of each strategy i.

Dividing both sides of Equation (11) by T, we obtain an explicit bound on the rate at which the average per-trial loss of **Hedge**(β) approaches the average loss for the best strategy:

$$\frac{L_{\textbf{Hedge}(\beta)}}{T} \leq \min_i \frac{L_i}{T} + \frac{\sqrt{2\tilde{L} \ln N}}{T} + \frac{\ln N}{T}. \tag{12}$$

Since $\tilde{L} \leq T$, this gives a worst case rate of convergence of $O\left(\sqrt{(\ln N)/T}\right)$. However, if $\tilde{L}$ is close to zero, then the rate of convergence will be much faster, roughly, $O((\ln N)/T)$.

Lemma 3 can also be applied to the other bounds given in Theorem 2 to obtain analogous results.

3 Applications

The framework described up to this point is quite general and can be applied in a wide variety of learning problems.

Consider the following set-up used by Chung [2]. We are given a decision space $\mathcal{D}$, a space of outcomes $\mathcal{Y}$, and a bounded loss function $\lambda : \mathcal{D} \times \mathcal{Y} \to [0,1]$. (Actually, our results require only that λ be bounded, but, by rescaling, we can assume that its range is $[0,1]$.) At every time step t, the learning algorithm selects a decision $d^t \in \mathcal{D}$, receives an outcome $y^t \in \mathcal{Y}$, and suffers loss $\lambda(d^t, y^t)$. More generally, we may allow the learner to select a distribution D^t over the space of decisions, in which case it suffers the expected loss of a decision randomly selected according to D^t; that is, its expected loss is $\Lambda(D^t, y^t)$ where

$$\Lambda(D, y) = E_{d\sim D}[\lambda(d, y)].$$

To decide on distribution D^t, we assume that the learner has access to a set of N *experts*. At every time step t, expert i produces its own distribution $\mathcal{E}_i^t$ on $\mathcal{D}$, and suffers loss $\Lambda(\mathcal{E}_i^t, y^t)$.

The goal of the learner is to combine the distributions produced by the experts so as to suffer expected loss "not much worse" than that of the best expert.

The results of Section 2 provide a method for solving this problem. Specifically, we run algorithm **Hedge**(β), treating each expert as a strategy. At every time step, **Hedge**(β) produces a distribution $\mathbf{p}^t$ on the set of experts which is used to construct the mixture distribution

$$D^t = \sum_{i=1}^{N} p_i^t \mathcal{E}_i^t.$$

For any outcome y^t, the loss suffered by **Hedge**(β) will then be

$$\Lambda(D^t, y^t) = \sum_{i=1}^{N} p_i^t \Lambda(\mathcal{E}_i^t, y^t).$$

Thus, if we define $\ell_i^t = \Lambda(\mathcal{E}_i^t, y^t)$ then the loss suffered by the learner is $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$, i.e., exactly the mixture loss that was analyzed in Section 2.

Hence, the bounds of Section 2 can be applied to our current framework. For instance, applying Equation (11), we obtain the following:

Theorem 4. *For any loss function λ, for any set of experts, and for any sequence of outcomes, the expected loss of* **Hedge**(β) *if used as described above is at most*

$$\sum_{t=1}^{T} \Lambda(D^t, y^t) \leq \min_i \sum_{t=1}^{T} \Lambda(\mathcal{E}_i^t, y^t) + \sqrt{2\tilde{L}\ln N} + \ln N$$

where $\tilde{L} \leq T$ is an assumed bound on the expected loss of the best expert, and $\beta = g(\tilde{L}/\ln N)$.

Example 1. In the k-ary prediction problem, $\mathcal{D} = \mathcal{Y} = \{1, 2, \ldots, k\}$, and $\lambda(d, y)$ is 1 if $d \neq y$ and 0 otherwise. In other words, the problem is to predict a sequence of letters over an alphabet of size k. The loss function λ is 1 if a mistake was made, and 0 otherwise. Thus, $\Lambda(D, y)$ is the probability (with respect to D) of a prediction that disagrees with y. The cumulative loss of the learner, or of any expert, is therefore the expected number of mistakes on the entire sequence. So, in this case, Theorem 2 states that the expected number of mistakes of the learning algorithm will exceed the expected number of mistakes of the best expert by at most $O\left(\sqrt{T \ln N}\right)$, or possibly much less if the loss of the best expert can be bounded ahead of time.

Bounds of this type were previously proved in the binary case ($k = 2$) by Littlestone and Warmuth [10] using the same algorithm. Their algorithm was later improved by Vovk [12] and Cesa-Bianchi et al. [1]. The main result of this section is a proof that such bounds can be shown to hold for any bounded loss function.

Example 2. The loss function λ may represent an arbitrary matrix game, such as "rock, paper, scissors." Here, $\mathcal{D} = \mathcal{Y} = \{\mathtt{R}, \mathtt{P}, \mathtt{S}\}$, and the loss function is defined by the matrix:

$$\begin{array}{ccccc} & & & \text{y} & \\ & & \mathtt{R} & \mathtt{P} & \mathtt{S} \\ & \mathtt{R} & \frac{1}{2} & 1 & 0 \\ \text{d} & \mathtt{P} & 0 & \frac{1}{2} & 1 \\ & \mathtt{S} & 1 & 0 & \frac{1}{2} \end{array}$$

The decision d represents the learner's play, and the outcome y is the adversary's play; then $\lambda(d, y)$, the learner's loss, is 1 if the learner loses the round, 0 if it wins the round, and $1/2$ if the round is tied. (For instance, $\lambda(\mathtt{S}, \mathtt{P}) = 0$ since "scissors cut paper.") So the cumulative loss of the learner (or an expert) is the expected number of losses in a series of rounds of game play (counting ties as half a loss). Our results show then that, in repeated play, the expected number of rounds lost by our algorithm will converge quickly to the expected number that would have been lost by the best of the experts (for the particular sequence of moves that were actually played by the adversary).

Example 3. Suppose that $\mathcal{D}$ and $\mathcal{Y}$ are finite, and that λ represents a game matrix as in the last example. Suppose further that we create one expert for each decision $d \in \mathcal{D}$ (i.e., that always recommends playing d). In this case, Theorem 2 implies that the learner's average per-round loss on a sequence of repeated plays of the game will converge, at worst, to the value of the game, i.e., to the loss that would have been suffered had the learner used the minimax "optimal" strategy for the game. Moreover, this holds true even if the learner knows nothing at all about the game that is being played (so that λ is unknown to the learner), and even if the adversarial opponent has complete knowledge both of the game that is being played and the algorithm that is being used by the learner. (See the related work of Hannan [7].)

Example 4. Suppose that $\mathcal{D} = \mathcal{Y}$ is the unit ball in $\mathbb{R}^n$, and that $\lambda(d, y) = ||d - y||$. Thus, the problem here is to predict the location of a point y, and the loss suffered is the Euclidean distance between the predicted point d and the actual outcome y. Theorem 2 can be applied if probabilistic predictions are allowed. However, in this setting it is more natural to require that the learner and each expert predict a single point (rather than a measure on the space of possible points). Essentially, this is the problem of "tracking" a sequence of points $y^1, \ldots, y^T$ where the loss function measures the distance to the predicted point.

To see how to handle the problem of finding deterministic predictions, notice that the loss function $\lambda(d, y)$ is convex with respect to d:

$$||(ad_1 + (1-a)d_2) - y|| \leq a||d_1 - y|| + (1-a)||d_2 - y|| \tag{13}$$

for any $a \in [0, 1]$ and any $y \in \mathcal{Y}$. Thus we can do as follows. At time t, the learner predicts with the weighted average of the experts' predictions: $d^t = \sum_{i=1}^N p_i^t e_i^t$ where $e_i^t \in \mathbb{R}^n$ is the prediction of the ith expert at time t. Regardless of the outcome y^t, Equation (13) implies that

$$||d^t - y^t|| \leq \sum_{i=1}^N p_i^t ||e_i^t - y^t|| \ .$$

Since Theorem 2 provides an upper bound on the right hand side of this inequality, we also obtain upper bounds for the left hand side. Thus, our results in this case give explicit bounds on the total error (i.e., distance between predicted and observed points) for the learner relative to the best of a team of experts.

In the one-dimensional case ($n = 1$), this case was previously analyzed by Littlestone and Warmuth [10], and later improved upon by Kivinen and Warmuth [9].

This result depends only on the convexity and the bounded range of the loss function $\lambda(d, y)$ with respect to d. Thus, it can also be applied, for example, to the squared-distance loss function $\lambda(d, y) = ||d - y||^2$, as well as the log loss function $\lambda(d, y) = -\ln(d \cdot y)$ used by Cover [3] for the design of "universal" investment portfolios. (In this last case, $\mathcal{D}$ is the set of probability vectors on n points, and $\mathcal{Y} = [1/B, B]^n$ for some constant $B > 0$.)

In many of the cases listed above, superior algorithms or analyses are known. Although weaker in specific cases, it should be emphasized that our results are far more general, and can be applied in settings that exhibit considerably less structure, such as the horse-racing example described in the introduction.

4 Boosting

In this section we show how the algorithm presented in Section 2 for the on-line allocation problem can be modified to boost the performance of weak learning algorithms.

We very briefly review the PAC learning model. Let X be a set called the *domain*. A *concept* is a Boolean function $c : X \to \{0, 1\}$. A *concept class* $\mathcal{C}$ is a collection of concepts. The learner has access to an oracle which provides labeled examples of the form $(x, c(x))$ where x is chosen randomly according to some fixed but unknown and arbitrary distribution D on the domain X, and $c \in \mathcal{C}$ is the *target concept*. After some amount of time, the learner must output a hypothesis $h : X \rightarrow [0, 1]$.[1] The *error* of the hypothesis h is the expected value $E_{x \sim D}(|h(x) - c(x)|)$ where x is chosen according to D.

A *strong PAC-learning algorithm* is an algorithm that, given $\epsilon, \delta > 0$ and access to random examples, outputs with probability $1 - \delta$ a hypothesis with error at most ϵ. Further, the running time must be polynomial in $1/\epsilon$, $1/\delta$ and other relevant parameters (namely, the "size" of the examples received, and the "size" or "complexity" of the target concept). A *weak PAC-learning algorithm* satisfies the same conditions but only for $\epsilon \geq 1/2 - \gamma$ where $\gamma > 0$ is either a constant, or decreases as $1/p$ where p is a polynomial in the relevant parameters.

Schapire [11] showed that any weak learning algorithm can be efficiently transformed or "boosted" into a strong learning algorithm. Later, Freund [5, 6] presented the "boost-by-majority" algorithm that is considerably more efficient than Schapire's algorithm. Both algorithms work by calling a given weak learning algorithm **WeakLearn** multiple times, each time presenting it with a different distribution over the domain X, and finally combining all of the generated hypotheses into a single hypothesis. The intuitive idea is to alter the distribution over the domain X in a way that increases the probability of the "harder" parts of the space.

A deficiency of the boost-by-majority algorithm is the requirement that the bias γ of the weak learning algorithm **WeakLearn** be known ahead of time. Not only is this worst-case bias usually unknown in practice, but the bias that can be achieved by **WeakLearn** will typically vary considerably from one distribution to the next. Unfortunately, the boost-by-majority algorithm cannot take advantage of hypotheses computed by **WeakLearn** with error significantly smaller than the presumed worst-case bias of $1/2 - \gamma$.

In this section, we present a new boosting algorithm which was derived from the on-line allocation algorithm of Section 2. This new algorithm is very nearly as efficient as boost-by-majority. However, unlike boost-by-majority, the accuracy of the final hypothesis produced by the new algorithm depends on the accuracy of *all* the hypotheses returned by **WeakLearn**, and so is able to more fully exploit the power of the weak learning algorithm.

Also, this new algorithm gives a clean method for handling real-valued hypotheses which often are produced by neural networks and other learning algorithms.

For the sake of simplicity, we assume that the domain X is the set $\{1, \ldots, N\}$. We present the new boosting algorithm using a simplified framework in which the distributions over the domain X are assumed to be directly accessible. In other words, the boosting algorithm is given the distribution D and can present the weak learning algorithm with any distribution vector it chooses. The running time of the boosting algorithm is linear in N.

This framework may seem unrealistic since the domain X is typically very large or

[1] The value $h(x)$ can be interpreted as a stochastic prediction of the label $c(x)$. Although we assume here that we have direct access to the bias of this prediction, our results can be extended to the case that h is instead a random mapping into $\{0, 1\}$.

Algorithm AdaBoost
Input: set of N labeled examples $\{(1, c(1)), \ldots, (N, c(N))\}$
distribution D over the examples
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize the weight vector: $w_i^1 = D(i)$ for $i = 1, \ldots, N$
Do for $t = 1, 2, \ldots, T$

1. Set
$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$$
2. Call **WeakLearn**, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis h_t.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i=1}^N p_i^t |h_t(i) - c(i)|$.
4. Set $\beta_t = \epsilon_t/(1 - \epsilon_t)$.
5. Set the new weights vector to be
$$w_i^{t+1} = w_i^t \beta_t^{1-|h_t(i)-c(i)|}$$

Output the hypothesis
$$h_f(i) = \begin{cases} 1, & \sum_{t=1}^T \left(\log \frac{1}{\beta_t}\right) h_t(i) \geq \frac{1}{2} \sum_{t=1}^T \log \frac{1}{\beta_t} \\ 0, & \text{otherwise} \end{cases} .$$

Fig. 1. The adaptive boosting algorithm.

even infinite. However, we can reduce the general problem to that of boosting over a small domain by drawing a small sample from the entire domain and using the uniform distribution over this sample as an approximation of the complete distribution. This method, called "boosting by sampling," is described in detail by Freund [6, Section 3.2].

The new boosting algorithm is described in Figure 1; we call the algorithm **AdaBoost** because it adjusts adaptively to the errors of the weak hypotheses returned by **WeakLearn**. The algorithm has a main loop which is iterated T times. Each time it defines a distribution $\mathbf{p}^t$ over X according to the current weight vector $\mathbf{w}^t$. It then feeds this distribution to **WeakLearn** and gets back a hypothesis h_t whose error is ϵ_t. If **WeakLearn** is a weak learning algorithm in the sense defined above, then $\epsilon_t \leq 1/2 - \gamma$ for all t; however, such a bound on the error need not be known ahead of time, and indeed, our results hold for any $\epsilon_t \in (0, 1/2]$.

The parameter β_t is chosen as a function of ϵ_t and is used for updating the weight vector. The update rule reduces the probability assigned to those examples on which the hypothesis makes a good prediction and increases the probability of the examples on which the prediction is poor.[2] The final hypothesis generated by the algorithm is a

[2] Furthermore, if h_t is Boolean (with range $\{0, 1\}$), then it can be shown that this update rule exactly removes the advantage of the last hypothesis. That is, the error of h_t on distribution $\mathbf{p}^{t+1}$ is exactly $1/2$.

weighted average of the T hypotheses generated by **WeakLearn**.

This algorithm is very similar to the algorithm **Hedge**(β) defined in Section 2. The "strategies" described in Section 2 are now replaced by points of the domain X, and the loss ℓ_i^t for the ith strategy at time t is defined here to be $1 - |h_t(i) - c(i)|$. The main difference between the two algorithms is that β is no longer fixed ahead of time but rather changes at each iteration according to ϵ_t.

If we are given ahead of time an upper bound $1/2 - \gamma$ on the errors $\epsilon_1, \ldots, \epsilon_T$, then we can directly apply algorithm **Hedge**(β) and its analysis. Briefly, we fix β to be $1 - \gamma$, and set $\ell_i^t = 1 - |h_t(i) - c(i)|$, and h_f as in **AdaBoost**, but with equal weight assigned to all T hypotheses. Then $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$ is exactly the accuracy of h_t on distribution $\mathbf{p}^t$, which, by assumption, is at least $1/2 + \gamma$. Also, letting $S = \{i : h_f(i) \neq c(i)\}$, it is straightforward to show that if $i \in S$ then

$$\frac{L_i}{T} = \frac{1}{T}\sum_{t=1}^{T} \ell_i^t = 1 - \frac{1}{T}\sum_{t=1}^{T} |c(i) - h_t(i)| = 1 - \left| c(i) - \frac{1}{T}\sum_{t=1}^{T} h_t(i) \right| \leq 1/2$$

by h_f's definition, and since $c(i) \in \{0, 1\}$. Thus, by Theorem 2,

$$T \cdot (1/2 + \gamma) \leq \sum_{t=1}^{T} \mathbf{p}^t \cdot \boldsymbol{\ell}^t \leq \frac{-\ln(\sum_{i \in S} D(i)) + (\gamma + \gamma^2)(T/2)}{\gamma}$$

since $-\ln(\beta) = -\ln(1 - \gamma) \leq \gamma + \gamma^2$ for $\gamma \in [0, 1/2]$. This implies that the error $\epsilon = \sum_{i \in S} D(i)$ of h_f is at most $e^{-T\gamma^2/2}$.

The boosting algorithm **AdaBoost** has two advantages over this direct application of **Hedge**(β). First, by giving a more refined analysis and choice of β, we obtain a significantly superior bound on the error ϵ. Second, the algorithm does not require prior knowledge of the accuracy of the hypotheses that **WeakLearn** will generate. Instead, it measures the accuracy of h_t at each iteration and sets its parameters accordingly. The update factor β_t decreases with ϵ_t which causes the difference between the distributions $\mathbf{p}^t$ and $\mathbf{p}^{t+1}$ to increase. Decreasing β_t also increases the weight $\ln(1/\beta_t)$ which is associated with h_t in the final hypothesis. This makes intuitive sense: more accurate hypotheses cause larger changes in the generated distributions and have more influence on the outcome of the final hypothesis.

We now give our analysis of the performance of **AdaBoost**.

Theorem 5. *Suppose the weak learning algorithm* **WeakLearn**, *when called by* **AdaBoost**, *generates hypotheses with errors* $\epsilon_1, \ldots, \epsilon_T$. *Then the error* ϵ *of the final hypothesis* h_f *output by* **AdaBoost** *is bounded above by*

$$\epsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)}. \tag{14}$$

Proof. We adapt the main arguments from Lemma 1 and Theorem 2. We use $\mathbf{p}^t$ and $\mathbf{w}^t$ as they are defined in Figure 1. We define the cost vector $\boldsymbol{\ell}^t$ as $\ell_i^t = 1 - |h_t(i) - c(i)|$ for all $1 \leq i \leq N$. Using this notation we can write $1 - \epsilon_t = \mathbf{p}^t \cdot \boldsymbol{\ell}^t$.

Substituting this equality into the first inequality in Equation (4) we get that

$$\left(\sum_{i=1}^{N} w_i^{t+1}\right) \leq \left(\sum_{i=1}^{N} w_i^{t}\right)(1-(1-\epsilon_t)(1-\beta_t)) \ . \tag{15}$$

Combining this inequality over $t = 1, \ldots, T$, we get that

$$\sum_{i=1}^{N} w_i^{T+1} \leq \prod_{t=1}^{T}(1-(1-\epsilon_t)(1-\beta_t)) \, . \tag{16}$$

The final hypothesis h_f, as defined in Figure 1, makes a mistake on instance i only if

$$\prod_{t=1}^{T} \beta_t^{-|h_t(i)-c(i)|} \geq \left(\prod_{t=1}^{T} \beta_t\right)^{-1/2} \tag{17}$$

(since $c(i) \in \{0,1\}$). The final weight of any instance i is

$$w_i^{T+1} = D(i) \prod_{t=1}^{T} \beta_t^{1-|h_t(i)-c(i)|}. \tag{18}$$

Combining Equations (17) and (18) we can lower bound the sum of the final weights by the sum of the final weights of the examples on which h_f is incorrect:

$$\sum_{i=1}^{N} w_i^{T+1} \geq \sum_{i:h_f(i)\neq c(i)} w_i^{T+1} \geq \left(\sum_{i:h_f(i)\neq c(i)} D(i)\right)\left(\prod_{t=1}^{T}\beta_t\right)^{1/2} = \epsilon \cdot \left(\prod_{t=1}^{T}\beta_t\right)^{1/2} \tag{19}$$

where ϵ is the error of h_f. Combining Equations (16) and (19), we get that

$$\epsilon \leq \prod_{t=1}^{T} \frac{1-(1-\epsilon_t)(1-\beta_t)}{\sqrt{\beta_t}}. \tag{20}$$

As all the factors in the product are positive, we can minimize the right hand side by minimizing each factor separately. Setting the derivative of the tth factor to zero, we find that the choice of β_t which minimizes the right hand side is $\beta_t = \epsilon_t/(1-\epsilon_t)$. Plugging this choice of β_t into Equation (20) we get Equation (14), completing the proof. □

The bound on the error given in Theorem 5, can also be written in the form

$$\epsilon \leq \exp\left(-\sum_{t=1}^{T} D_{\mathrm{kl}}(1/2||\epsilon_t)\right) \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right) \tag{21}$$

where $D_{\mathrm{kl}}(a||b) = a\ln(a/b) + (1-a)\ln((1-a)/(1-b))$ is the Kullback-Leibler divergence, and where $\epsilon_t = 1/2 - \gamma_t$. In the case where the errors of all the hypotheses are equal to $1/2-\gamma$, Equation (21) simplifies to

$$\epsilon \leq \exp\left(-T \cdot D_{\mathrm{kl}}(1/2||1/2-\gamma)\right) \ . \tag{22}$$

This is a form of the Chernoff bound for the probability that less than $T/2$ coin flips turn out "heads" in T tosses of a random coin whose probability for "heads" is $1/2-\gamma$. This is exactly the same closed-form bound that is given for the boost-by-majority algorithm [6]. From Equation (22) we get that the number of iterations of the boosting algorithm that is sufficient to achieve error ϵ of h_f is

$$T = \frac{1}{D_{\mathrm{kl}}\big(1/2 || 1/2-\gamma\big)} \ln \frac{1}{\epsilon} \leq \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon} \,. \qquad (23)$$

Note, however, that when the errors of the hypotheses generated by **WeakLearn** are not uniform, Theorem 5 implies that the final error depends on the error of all of the weak hypotheses. Previous bounds on the errors of boosting algorithms depended only on the maximal error of the weakest hypothesis and ignored the advantage that can be gained from the hypotheses whose errors are smaller. This advantage seems to be very relevant to practical applications of boosting, because there one expects the error of the learning algorithm to increase as the distributions fed to **WeakLearn** shift more and more away from the target distribution.

As an aside, it is interesting to consider the relation between the final hypothesis that is generated by **AdaBoost** and the one which is suggested by a Bayesian analysis. Suppose we are given a set of $\{0,1\}$-valued hypotheses $h_1, \ldots, h_T$ and that our goal is to combine the prediction of these hypotheses in the optimal way. This is especially easy if we assume that the errors of the different hypotheses are independent of each other and of the target concept, that is, if we assume that $P(h_t \neq c) = \epsilon_t$ independently of the values of the other hypotheses and of the actual label. In this case, the optimal Bayes decision rule can be written in a particularly simple form: For a given example x, we denote the sets of indices of hypotheses that predict 0 and 1 by H_0 and H_1 respectively. Then the Bayes decision rule is equivalent to predicting that $c(x) = 1$ if

$$P(c=0) \prod_{t \in H_0} (1-\epsilon_t) \prod_{t \in H_1} \epsilon_t < P(c=1) \prod_{t \in H_0} \epsilon_t \prod_{t \in H_1} (1-\epsilon_t),$$

and 0 otherwise. We add to the set of hypotheses the trivial hypothesis h_0 which always predicts the value 1. After doing this, we can replace $P(c=0)$ by ϵ_0. Taking the logarithm of both sides in this inequality and rearranging the terms, we find that the Bayes decision rule is identical to the combination rule that is generated by **AdaBoost**.

If the errors of the different hypotheses are dependent, then the Bayes optimal decision rule becomes much more complicated. However, in practice, it is common to use the simple rule described above even when there is no justification for assuming independence. An interesting alternative to this practice would be to use the algorithm **AdaBoost** to find a combination rule which, by Theorem 5, has a guaranteed non-trivial accuracy.

Finally, note that **AdaBoost**, unlike boost-by-majority, combines the weak hypotheses by summing their probabilistic predictions. Drucker, Schapire and Simard [4], in experiments they performed using boosting to improve the performance of a real-valued neural network, observed that summing the outcomes of the networks and then selecting the best prediction performs better than selecting the best prediction of each network and then combining them with a majority rule. It is interesting that the new boosting algorithm's final hypothesis uses the same combination rule that was observed to be better in practice, but which previously lacked theoretical justification. Experiments are needed to measure whether the new algorithm has an advantage in real world applications.

4.1 Improving the error bound

We show in this section how the bound given in Theorem 5 can be improved by a factor of two. The main idea of this improvement is to replace the "hard" $\{0, 1\}$-valued decision used by h_f by a "soft" threshold.

To be more precise, let

$$r(i) = \frac{\sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) h_t(i)}{\sum_{t=1}^{T} \log \frac{1}{\beta_t}}$$

be a weighted average of the weak hypotheses h_t. We will here consider final hypotheses of the form $h_f(i) = F(r(i))$ where $F : [0, 1] \to [0, 1]$. For the version of **AdaBoost** given in Figure 1, $F(r)$ is the hard threshold that equals 1 if $r \geq 1/2$ and 0 otherwise. In this section, we will instead use soft threshold functions that take values in $[0, 1]$. As mentioned above, when $h_f(i) \in [0, 1]$, we can interpret h_f as a randomized hypothesis and $h_f(i)$ as the probability of predicting 1. Then the error $E_{i \sim D}[|h_f(i) - c(i)|]$ is simply the probability of an incorrect prediction.

Theorem 6. *Let $\epsilon_1, \ldots, \epsilon_T$ be as in Theorem 5, and let $r(i)$ be as defined above. Let the modified final hypothesis be defined by $h_f = F(r(i))$ where F satisfies the following for $r \in [0, 1]$:*

$$F(1 - r) = 1 - F(r); \quad \textit{and} \quad F(r) \leq \frac{1}{2} \left(\prod_{t=1}^{T} \beta_t \right)^{1/2 - r}.$$

Then the error ϵ of h_f is bounded above by

$$\epsilon \leq 2^{T-1} \prod_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)}.$$

For instance, it can be shown that the sigmoid function $F(r) = \left(1 + \prod_{t=1}^{T} \beta_t^{2r-1}\right)^{-1}$ satisfies the conditions of the theorem.

Proof. By our assumptions on F, the error of h_f is

$$\begin{aligned}\epsilon = \sum_{i=1}^{N} D(i) \cdot |F(r(i)) - c(i)| &= \sum_{i=1}^{N} D(i) F(|r(i) - c(i)|) \\ &\leq \frac{1}{2} \sum_{i=1}^{N} \left(D(i) \prod_{t=1}^{T} \beta_t^{1/2 - |r(i) - c(i)|} \right).\end{aligned}$$

Since $c(i) \in \{0,1\}$ and by definition of $r(i)$, this implies that

$$\begin{aligned}\epsilon &\leq \frac{1}{2}\sum_{i=1}^{N}\left(D(i)\prod_{t=1}^{T}\beta_t^{1/2-|h_t(i)-c(i)|}\right)\\ &= \frac{1}{2}\left(\sum_{i=1}^{N} w_i^{T+1}\right)\prod_{t=1}^{T}\beta_t^{-1/2} \leq \frac{1}{2}\prod_{t=1}^{T}\Big((1-(1-\epsilon_t)(1-\beta_t))\beta_t^{-1/2}\Big).\end{aligned}$$

The last two steps follow from Equations (18) and (16), respectively. The theorem now follows from our choice of β_t. □

Acknowledgments

Thanks to David Helmbold, Keith Messer and Manfred Warmuth for helpful discussions.

References

1. 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, May 1993.
2. Thomas H. Chung. Approximate methods for sequential decision making using expert advice. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 183–189, 1994.
3. Thomas M. Cover. Universal portfolios. *Mathematics of Finance*, 1(1):1–29, January 1991.
4. Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, 1993.
5. Yoav Freund. *Data Filtering and Distribution Modeling Algorithms for Machine Learning*. PhD thesis, University of California at Santa Cruz, 1993. Retrievable from: ftp.cse.ucsc.edu/pub/tr/ucsc-crl-93-37.ps.Z.
6. Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, To appear.
7. James Hannan. Approximation to Bayes risk in repeated play. In M. Dresher, A. W. Tucker, and P. Wolfe, editors, *Contributions to the Theory of Games*, volume III, pages 97–139. Princeton University Press, 1957.
8. David Haussler, Jyrki Kivinen, and Manfred K. Warmuth. Tight worst-case loss bounds for predicting with expert advice. In *Proceedings of the Second European Conference on Computational Learning Theory*, 1995.
9. Jyrki Kivinen and Manfred K. Warmuth. Using experts for predicting continuous outcomes. In *Computational Learning Theory: EuroCOLT '93*, pages 109–120, 1994.
10. Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
11. Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
12. Volodimir G. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.