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

Yoav Freund Robert E. Schapire

AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
{yoav, schapire}@research.att.com

December 19, 1996

Abstract

In the first part of the paper 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 [20] 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 $\mathbb{R}^n$. In the second part of the paper we apply the multiplicative weight-update technique to derive a new boosting algorithm. This boosting algorithm does not require any prior knowledge about the performance of the weak learning algorithm. We also study generalizations of the new boosting algorithm to the problem of learning functions whose range, rather than being binary, is an arbitrary finite set or a bounded segment of the real line.

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.

*This paper appeared in *Journal of Computer and System Sciences*, 55(1):119-139, 1997. An extended abstract of this work appeared in the Proceedings of the Second European Conference on Computational Learning Theory, Barcelona, March, 1995.

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 [20] "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 [20] and Cesa-Bianchi et al. [4] 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 [25], Kivinen and Warmuth [19], and Haussler, Kivinen and Warmuth [15]. Like us, these authors focused primarily on multiplicative weight-update algorithms. Chung [5] also presented a generalization, giving the problem a game-theoretic treatment.

Boosting

Returning to the horse-racing story, suppose now that the gambler grows weary of choosing among the experts and instead wishes to create a computer program that will accurately predict the winner of a horse race based on the usual information (number of races recently won by each horse, betting odds for each horse, etc.). To create such a program, he asks his favorite expert to explain his betting strategy. Not surprisingly, the expert is unable to articulate a grand set of rules for selecting a horse. On the other hand, when presented with the data for a specific set of races, the expert has no trouble coming up with a "rule-of-thumb" for that set of races (such as, "Bet on the horse that has recently won the most races" or "Bet on the horse with the most favored odds"). Although such a rule-of-thumb, by itself, is obviously very rough and inaccurate, it is not unreasonable to expect it to provide predictions that are at least a little bit better than random guessing. Furthermore, by repeatedly asking the expert's opinion on different collections of races, the gambler is able to extract many rules-of-thumb.

In order to use these rules-of-thumb to maximum advantage, there are two problems faced by the gambler: First, how should he choose the collections of races presented to the expert so as to extract rules-of-thumb from the expert that will be the most useful? Second, once he has collected many rules-of-thumb, how can they be combined into a single, highly accurate prediction rule?

Boosting refers to this general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb. In the second part of the paper, we present and analyze a new boosting algorithm inspired by the methods we used for solving the on-line allocation problem.

Formally, boosting proceeds as follows: The booster is provided with a set of labeled training examples $(x_1, y_1), \ldots, (x_N, y_N)$, where y_i is the label associated with instance x_i; for instance, in the horse-racing example, x_i might be the observable data associated with a particular horse race, and y_i the outcome (winning horse) of that race. On each round $t = 1, \ldots, T$, the booster devises a distribution D_t over the set of examples, and requests (from an unspecified oracle) a *weak hypothesis* (or rule-of-thumb) h_t with low error ϵ_t with respect to D_t (that is, $\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$). Thus, distribution D_t specifies the relative importance of each example for the current round. After T rounds, the booster must combine the weak hypotheses into a single prediction rule.

Unlike the previous boosting algorithms of Freund [10, 11] and Schapire [22], the new algorithm needs no prior knowledge of the accuracies of the weak hypotheses. Rather, it adapts to these accuracies and generates a weighted majority hypothesis in which the weight of each weak hypothesis is a function of its accuracy. For binary prediction problems, we prove in Section 4 that the error of this final hypothesis (with respect to the given set of examples) is bounded by $\exp(-2\sum_{t=1}^{T}\gamma_t^2)$ where $\epsilon_t = 1/2 - \gamma_t$ is the error of the tth weak hypothesis. Since a hypothesis that makes entirely random guesses has error $1/2$, γ_t measures the accuracy of the tth weak hypothesis relative to random guessing. Thus, this bound shows that if we can consistently find weak hypotheses that are slightly better than random guessing, then the error of the final hypothesis drops exponentially fast.

Note that the bound on the accuracy of the 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.

In Section 5, we give two extensions of our boosting algorithm to multi-class prediction

Algorithm Hedge(β)
Parameters: $\beta \in [0,1]$
initial weight vector $\mathbf{w}^1 \in [0,1]^N$ with $\sum_{i=1}^N w_i^1 = 1$
number of trials T
Do for $t = 1, 2, \ldots, T$

1. Choose allocation
$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$$
2. Receive loss vector $\boldsymbol{\ell}^t \in [0,1]^N$ from environment.
3. Suffer loss $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$.
4. Set the new weights vector to be
$$w_i^{t+1} = w_i^t \beta^{\ell_i^t}$$

Figure 1: The on-line allocation algorithm.

problems in which each example belongs to one of several possible classes (rather than just two). We also give an extension to regression problems in which the goal is to estimate a real-valued function.

2 The on-line allocation 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 [20].

The pseudo-code for **Hedge**(β) is shown in Figure 1. 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

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

More generally, it can be shown that our analysis is applicable with only minor modification to an alternative update rule of the form

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

where $U_\beta : [0,1] \to [0,1]$ is any function, parameterized by $\beta \in [0,1]$ satisfying

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

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

2.1 Analysis

The analysis of **Hedge**(β) mimics directly that given by Littlestone and Warmuth [20]. 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 $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^T$,*

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

Proof: By a convexity argument, it can be shown that

$$\alpha^r \le 1 - (1-\alpha)r \tag{3}$$

for $\alpha \ge 0$ and $r \in [0,1]$. Combined with Equations (1) and (2), this implies

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

Applying repeatedly for $t = 1, \ldots, T$ yields

$$\begin{aligned} \sum_{i=1}^N w_i^{T+1} &\le \prod_{t=1}^T (1 - (1-\beta)\mathbf{p}^t \cdot \boldsymbol{\ell}^t) \\ &\le \exp\left(-(1-\beta)\sum_{t=1}^T \mathbf{p}^t \cdot \boldsymbol{\ell}^t\right) \end{aligned}$$

since $1 + x \le e^x$ for all x. The lemma follows immediately. ■

Thus,

$$L_{\mathbf{Hedge}(\beta)} \le \frac{-\ln\left(\sum_{i=1}^N w_i^{T+1}\right)}{1-\beta}. \tag{5}$$

Note that, from Equation (2),

$$w_i^{T+1} = w_i^1 \prod_{t=1}^T \beta^{\ell_i^t} = w_i^1 \beta^{L_i}. \tag{6}$$

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*

$$L_{\mathbf{Hedge}(\beta)} \leq \frac{-\ln(w_i^1) - L_i \ln \beta}{1-\beta}. \tag{7}$$

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

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

Proof: We prove the more general statement (8) since Equation (7) follows in the special case that $S = \{i\}$.

From Equation (6),

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

The theorem now follows immediately from Equation (5). ■

The simpler bound (7) 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

$$L_{\mathbf{Hedge}(\beta)} \leq \frac{\min_i L_i \ln(1/\beta) + \ln N}{1-\beta}. \tag{9}$$

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

The more complicated bound (8) 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 (8) can be replaced by an integral, and the maximum by a supremum.

The bound given in Equation (9) can be written as

$$L_{\mathbf{Hedge}(\beta)} \leq c \min_i L_i + a \ln N, \tag{10}$$

where $c = \ln(1/\beta)/(1-\beta)$ and $a = 1/(1-\beta)$. Vovk [24] analyzes prediction algorithms that have performance bounds of this form, and proves tight upper and lower bounds for the achievable values of c and a. Using Vovk's results, we can show that the constants a and c achieved by **Hedge**(β) are optimal.

Theorem 3 *Let B be an algorithm for the on-line allocation problem with an arbitrary number of strategies. Suppose that there exist positive real numbers a and c such that for any number of strategies N and for any sequence of loss vectors $\boldsymbol{\ell}^1, \ldots, \boldsymbol{\ell}^T$*

$$L_B \leq c \min_i L_i + a \ln N.$$

Then for all $\beta \in (0, 1)$, either

$$c \geq \frac{\ln(1/\beta)}{1-\beta} \quad \text{or} \quad a \geq \frac{1}{(1-\beta)}.$$

The proof is given in the appendix.

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 4 *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 4 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 (9) and Lemma 4, we have

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

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:

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

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

The bound given in Equation (11) can be improved in special cases in which the loss is a function of a *prediction* and an *outcome* and this function is of a special form (see example 4 below). However, for the general case, one cannot improve the square-root term $\sqrt{2\tilde{L}\ln N}$, by more than a constant factor. This is a corollary of the lower bound given by Cesa-Bianchi et al. ([4], Theorem 7) who analyze an on-line prediction problem that can be seen as a special case of the on-line allocation model.

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 [5]. We are given a decision space Δ, a space of outcomes Ω, and a bounded loss function $\lambda : \Delta \times \Omega \rightarrow [0,1]$. (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 $\delta^t \in \Delta$, receives an outcome $\omega^t \in \Omega$, and suffers loss $\lambda(\delta^t, \omega^t)$. More generally, we may allow the learner to select a distribution $\mathcal{D}^t$ over the space of decisions, in which case it suffers the expected loss of a decision randomly selected according to $\mathcal{D}^t$; that is, its expected loss is $\Lambda(\mathcal{D}^t, \omega^t)$ where

$$\Lambda(\mathcal{D}, \omega) = E_{\delta \sim \mathcal{D}}[\lambda(\delta, \omega)].$$

To decide on distribution $\mathcal{D}^t$, 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 Δ, and suffers loss $\Lambda(\mathcal{E}_i^t, \omega^t)$.

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

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

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

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

Thus, if we define $\ell_i^t = \Lambda(\mathcal{E}_i^t, \omega^t)$ 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 5 *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*

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

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, $\Delta = \Omega = \{1, 2, \ldots, k\}$, and $\lambda(\delta, \omega)$ is 1 if $\delta \neq \omega$ 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, $\Lambda(\mathcal{D}, \omega)$ is the probability (with respect to $\mathcal{D}$) of a prediction that disagrees with ω. 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 [20] using the same algorithm. Their algorithm was later improved by Vovk [25] and Cesa-Bianchi et al. [4]. 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, $\Delta = \Omega = \{\mathtt{R}, \mathtt{P}, \mathtt{S}\}$, and the loss function is defined by the matrix:

$$\begin{array}{ccccc} & & & \omega & \\ & & \mathtt{R} & \mathtt{P} & \mathtt{S} \\ & \mathtt{R} & \frac{1}{2} & 1 & 0 \\ \delta & \mathtt{P} & 0 & \frac{1}{2} & 1 \\ & \mathtt{S} & 1 & 0 & \frac{1}{2} \end{array}$$

The decision δ represents the learner's play, and the outcome ω is the adversary's play; then $\lambda(\delta, \omega)$, 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 Δ and Ω are finite, and that λ represents a game matrix as in the last example. Suppose further that we create one expert for each decision $\delta \in \Delta$ and that expert always recommends playing δ. In game-theoretic terminology such experts would be identified with *pure* strategies. von Neumann's classical min-max theorem states that for any fixed game matrix there exists a distribution over the actions, also called a *mixed* strategy which achieves the min-max optimal value of the expected loss against *any* adversarial strategy. This min-max value is also called the *value of the game*.

Suppose that we use algorithm **Hedge**(β) to choose distributions over the actions when playing a matrix game repeatedly. In this case, Theorem 2 implies that the gap between the learner's average per-round loss can never be much larger than that of the best pure strategy, and that the maximal gap decreases to zero at the rate $O(1/\sqrt{T \log |\Delta|})$. However, the expected loss of the optimal mixed strategy is a fixed convex combination of the losses of the pure strategies, thus it can never be smaller than the loss of the *best* pure strategy for a particular sequence of events. We conclude that the expected per-trial loss of **Hedge**(β) is upper bounded by the value of the game plus $O(1/\sqrt{T \log |\Delta|})$. In other words, the algorithm can never perform much worse that an algorithm that uses the optimal mixed strategy for the game, and it can be better if the adversary does not play optimally. 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. Algorithms with similar properties (but weaker convergence bounds) were first devised by Blackwell [2] and Hannan [14]. For more details see our related paper [13].

Example 4. Suppose that $\Delta = \Omega$ is the unit ball in $\mathbb{R}^n$, and that $\lambda(\delta, \omega) = ||\delta - \omega||$. Thus, the problem here is to predict the location of a point ω, and the loss suffered is the Euclidean distance between the predicted point δ and the actual outcome ω. 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 $\omega^1, \ldots, \omega^T$ 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 $\lambda(\delta, \omega)$ is convex with respect to δ:

$$||(a\delta_1 + (1-a)\delta_2) - \omega|| \leq a||\delta_1 - \omega|| + (1-a)||\delta_2 - \omega|| \tag{13}$$

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

$$||\delta^t - \omega^t|| \leq \sum_{i=1}^N p_i^t ||\varepsilon_i^t - \omega^t|| \; .$$

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 [20], and later improved upon by Kivinen and Warmuth [19].

This result depends only on the convexity and the bounded range of the loss function $\lambda(\delta, \omega)$ with respect to δ. Thus, it can also be applied, for example, to the squared-distance loss function $\lambda(\delta, \omega) = ||\delta - \omega||^2$, as well as the log loss function $\lambda(\delta, \omega) = -\ln(\delta \cdot \omega)$ used by Cover [6] for the design of "universal" investment portfolios. (In this last case, Δ is the set of probability vectors on n points, and $\Omega = [1/B, B]^n$ for some constant $B > 1$.)

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 (see, for instance, Kearns and Vazirani [18] for a more detailed description). 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 $\mathcal{D}$ on the domain X, and $c \in \mathcal{C}$ is the *target concept*. After some amount of time, the learner must output a hypothesis $h : X \to [0,1]$. The value $h(x)$ can be interpreted as a randomized prediction of the label of x that is 1 with probability $h(x)$ and 0 with probability $1 - h(x)$. (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 $\{0,1\}$.) The *error* of the hypothesis h is the expected value $E_{x \sim \mathcal{D}}(|h(x) - c(x)|)$ where x is chosen according to $\mathcal{D}$. If $h(x)$ is interpreted as a stochastic prediction, then this is simply the probability of an incorrect prediction.

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. We use **WeakLearn** to denote a generic weak learning algorithm.

Schapire [22] showed that any weak learning algorithm can be efficiently transformed or "boosted" into a strong learning algorithm. Later, Freund [10, 11] presented the "boost-by-majority" algorithm that is considerably more efficient than Schapire's. 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, thus forcing the weak learner to generate new hypotheses that make less mistakes on these parts.

An important, practical 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.

4.1 The new boosting algorithm

Although boosting has its roots in the PAC model, for the remainder of the paper, we adopt a more general learning framework in which the learner receives examples (x_i, y_i) chosen randomly according to some fixed but unknown distribution $\mathcal{P}$ on $X \times Y$, where Y is a set of possible labels. As usual, the goal is to learn to predict the label y given an instance x.

We start by describing our new boosting algorithm in the simplest case that the label set Y consists of just two possible labels, $Y = \{0, 1\}$. In later sections, we give extensions of the algorithm for more general label sets.

Freund [11] describes two frameworks in which boosting can be applied: boosting by filtering and boosting by sampling. In this paper, we use the boosting by sampling framework, which is the natural framework for analyzing "batch" learning, i.e., learning using a fixed training set which is stored in the computer's memory.

We assume that a sequence of N training examples (labeled instances) $(x_1, y_1), \ldots, (x_N, y_N)$ is drawn randomly from $X \times Y$ according to distribution $\mathcal{P}$. We use boosting to find a hypothesis h_f which is consistent with most of the sample (i.e., $h_f(x_i) = y_i$ for most $1 \leq i \leq N$). In general, a hypothesis which is accurate on the training set might not be accurate on examples outside the training set; this problem is sometimes referred to as "over-fitting." Often, however, over-fitting can be avoided by restricting the hypothesis to be simple. We will come back to this problem in Section 4.3.

The new boosting algorithm is described in Figure 2. The goal of the algorithm is to find a final hypothesis with low error relative to a given distribution D over the training examples. Unlike the distribution $\mathcal{P}$ which is over $X \times Y$ and is set by "nature," the distribution D is only over the instances in the training set and is controlled by the learner. Ordinarily, this distribution will be set to be uniform so that $D(i) = 1/N$. The algorithm maintains a set of weights $\mathbf{w}^t$ over the training examples. On iteration t a distribution $\mathbf{p}^t$ is computed by normalizing these weights. This distribution is fed to the weak learner **WeakLearn** which generates a hypothesis h_t that (we hope) has small error with respect to the distribution.[1] Using the new hypothesis h_t, the boosting algorithm generates the next weight vector $\mathbf{w}^{t+1}$, and the process repeats. After T such iterations, the final hypothesis h_f is output. The hypothesis h_f combines the outputs of the T weak hypotheses using a weighted majority vote.

We call the algorithm **AdaBoost** because, unlike previous algorithms, it adjusts adaptively to the errors of the weak hypotheses returned by **WeakLearn**. If **WeakLearn** is a PAC weak learning algorithm in the sense defined above, then $\epsilon_t \leq 1/2 - \gamma$ for all t (assuming the examples have been generated appropriately with $y_i = c(x_i)$ for some $c \in \mathcal{C}$). However, such a bound on the error need not be known ahead of time. Our results hold for any $\epsilon_t \in [0,1]$, and depend only on the performance of the weak learner on those distributions that are *actually generated* during the boosting process.

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]

Note that **AdaBoost**, unlike boost-by-majority, combines the weak hypotheses by summing their probabilistic predictions. Drucker, Schapire and Simard [9], 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.

Since it was first introduced, several successful experiments have been conducted using **AdaBoost**, including work by the authors [12], Drucker and Cortes [8], Jackson and Craven [16], Quinlan [21], and Breiman [3].

4.2 Analysis

Comparing Figures 1 and 2, there is an obvious similarity between the algorithms **Hedge**(β) and **AdaBoost**. This similarity reflects a surprising "dual" relationship between the on-line allocation model and the problem of boosting. Put another way, there is a direct mapping or reduction of the boosting problem to the on-line allocation problem. In such a reduction, one might naturally expect a correspondence relating the strategies to the weak hypotheses and the trials (and associated loss vectors) to the examples in the training set. However, the reduction we have used is reversed: the "strategies" correspond to the examples, and the trials

[1] Some learning algorithms can be generalized to use a given distribution directly. For instance, gradient based algorithms can use the probability associated with each example to scale the update step size which is based on the example. If the algorithm cannot be generalized in this way, the training sample can be re-sampled to generate a new set of training examples that is distributed according to the given distribution. The computation required to generate each re-sampled example takes $O(\log N)$ time.

[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 $1/2$.

Algorithm AdaBoost

Input: sequence of N labeled examples $\langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$
distribution D over the N 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 $h_t : X \to [0, 1]$.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|$.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Set the new weights vector to be
$$w_i^{t+1} = w_i^t \beta_t^{1 - |h_t(x_i) - y_i|}$$

Output the hypothesis

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

Figure 2: The adaptive boosting algorithm.

are associated with the weak hypotheses. Another reversal is in the definition of the loss: in **Hedge**(β) the loss ℓ_i^t is small if the ith strategy suggests a *good* action on the tth trial while in **AdaBoost** the "loss" $\ell_i^t = 1 - |h_t(x_i) - y_i|$ appearing in the weight-update rule (Step 5) is small if the tth hypothesis suggests a *bad* prediction on the ith example. The reason is that in **Hedge**(β) the weight associated with a strategy is increased if the strategy is successful while in **AdaBoost** the weight associated with an example is increased if the example is "hard."

The main technical difference between the two algorithms is that in **AdaBoost** the parameter 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 the information that $\epsilon_t \leq 1/2 - \gamma$ for some $\gamma > 0$ and for all $t = 1, \ldots, T$, then we could instead directly apply algorithm **Hedge**(β) and its analysis as follows: Fix β to be $1 - \gamma$, and set $\ell_i^t = 1 - |h_t(x_i) - y_i|$, 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 $S = \{i : h_f(x_i) \neq y_i\}$, it is straightforward to show that if $i \in S$ then

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

by h_f's definition, and since $y_i \in \{0, 1\}$. 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**. Note that this theorem applies also if, for some hypotheses, $\epsilon_t \geq 1/2$.

Theorem 6 *Suppose the weak learning algorithm* **WeakLearn**, *when called by* **AdaBoost**, *generates hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$ (as defined in Step 3 of Figure 2.) Then the error $\epsilon = \Pr_{i \sim D}[h_f(x_i) \neq y_i]$ 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 2.

Similar to Equation (4), the update rule given in Step 5 in Figure 2 implies that

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

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 2, makes a mistake on instance i only if

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

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

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

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:

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

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 6, can also be written in the form

$$\epsilon \leq \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} = \exp\left(-\sum_{t=1}^{T} \mathrm{KL}(1/2 \mid\mid 1/2-\gamma_t)\right) \leq \exp\left(-2\sum_{t=1}^{T} \gamma_t^2\right) \tag{21}$$

where $\mathrm{KL}(a \mid\mid b) = a\ln(a/b) + (1-a)\ln((1-a)/(1-b))$ is the Kullback-Leibler divergence, and where ϵ_t has been replaced by $1/2-\gamma_t$. In the case where the errors of all the hypotheses are equal to $1/2-\gamma$, Equation (21) simplifies to

$$\epsilon \leq \left(1-4\gamma^2\right)^{T/2} = \exp\left(-T \cdot \mathrm{KL}(1/2 \mid\mid 1/2-\gamma)\right) \leq \exp\left(-2T\gamma^2\right). \tag{22}$$

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 bound has the same asymptotic behavior as the bound given for the boost-by-majority algorithm [11]. 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

$$T = \left\lceil \frac{1}{\mathrm{KL}(1/2 \;||\; 1/2 - \gamma)} \ln \frac{1}{\epsilon} \right\rceil \leq \left\lceil \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon} \right\rceil . \tag{23}$$

Note, however, that when the errors of the hypotheses generated by **WeakLearn** are not uniform, Theorem 6 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.

4.3 Generalization error

We now come back to discussing the error of the final hypothesis outside the training set. Theorem 6 guarantees that the error of h_f on the sample is small; however, the quantity that interests us is the generalization error of h_f, which is the error of h_f over the whole instance space X; that is, $\epsilon_g = \Pr_{(x,y)\sim\mathcal{P}}\left[h_f(x) \neq y\right]$. In order to make ϵ_g close to the empirical error $\hat{\epsilon}$ on the training set, we have to restrict the choice of h_f in some way. One natural way of doing this in the context of boosting is to restrict the weak learner to choose its hypotheses from some simple class of functions and restrict T, the number of weak hypotheses that are combined to make h_f. The choice of the class of weak hypotheses is specific to the learning problem at hand and should reflect our knowledge about the properties of the unknown concept. As for the choice of T, various general methods can be devised. One popular method is to use an upper bound on the VC-dimension of the concept class. This method is sometimes called "structural risk minimization." See Vapnik's book [23] for an extensive discussion of the theory of structural risk minimization. For our purposes, we quote Vapnik's Theorem 6.7:

Theorem 7 (Vapnik) *Let H be a class of binary functions over some domain X. Let d be the VC-dimension of H. Let $\mathcal{P}$ be a distribution over the pairs $X \times \{0,1\}$. For $h \in H$, define the (generalization) error of h with respect to $\mathcal{P}$ to be*

$$\epsilon_g(h) \doteq \Pr_{(x,y)\sim\mathcal{P}}\left[h(x) \neq y\right].$$

Let $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ be a sample (training set) of N independent random examples drawn from $X \times \{0,1\}$ according to $\mathcal{P}$. Define the empirical error of h with respect to the sample S to be

$$\hat{\epsilon}(h) \doteq \frac{|\{i : h(x_i) \neq y_i\}|}{N} .$$

Then, for any $\delta > 0$ we have that

$$\Pr\left[\exists h \in H : |\hat{\epsilon}(h) - \epsilon_g(h)| > 2\sqrt{\frac{d\left(\ln \frac{2N}{d} + 1\right) + \ln \frac{9}{\delta}}{N}}\,\right] \leq \delta$$

where the probability is computed with respect to the random choice of the sample S.

Let $\theta : \mathbb{R} \to \{0,1\}$ be defined by

$$\theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and, for any class H of functions, let $\Theta_T(H)$ be the class of all functions defined as a linear threshold of T functions in H:

$$\Theta_T(H) = \left\{ \theta\left(\sum_{t=1}^{T} a_t h_t - b \right) : b, a_1, \ldots, a_T \in \mathbb{R}; h_1, \ldots, h_T \in H \right\}.$$

Clearly, if all hypotheses generated by **WeakLearn** belong to some class H, then the final hypothesis of **AdaBoost**, after T rounds of boosting, belongs to $\Theta_T(H)$. Thus, the next theorem provides an upper bound on the VC-dimension of the class of final hypotheses generated by **AdaBoost** in terms of the weak hypothesis class.

Theorem 8 *Let H be a class of binary functions of VC-dimension $d \geq 2$. Then the VC-dimension of $\Theta_T(H)$ is at most $2(d+1)(T+1)\log_2(e(T+1))$ (where e is the base of the natural logarithm.)*

Therefore, if the hypotheses generated by **WeakLearn** *are chosen from a class of VC-dimension $d \geq 2$, then the final hypotheses generated by* **AdaBoost** *after T iterations belong to a class of VC-dimension at most $2(d+1)(T+1)\log_2[e(T+1)]$.*

Proof: We use a result about the VC-dimension of computation networks proved by Baum and Haussler [1]. We can view the final hypothesis output by **AdaBoost** as a function that is computed by a two-layer feed-forward network where the computation units of the first layer are the weak hypotheses and the computation unit of the second layer is the linear threshold function which combines the weak hypotheses. The VC-dimension of the set of linear threshold functions over $\mathbb{R}^T$ is $T+1$ [26]. Thus the sum over all computation units of the VC-dimensions of the classes of functions associated with each unit is $Td + (T+1) < (T+1)(d+1)$. Baum and Haussler's Theorem 1 [1] implies that the number of different functions that can be realized by $h \in \Theta_T(H)$ when the domain is restricted to a set of size m is at most $((T+1)em/(T+1)(d+1))^{(T+1)(d+1)}$. If $d \geq 2$, $T \geq 1$ and we set $m = \lceil 2(T+1)(d+1)\log_2[e(T+1)] \rceil$, then the number of realizable functions is smaller than 2^m which implies that the VC-dimension of $\Theta_T(H)$ is smaller than m. ■

Following the guidelines of structural risk minimization we can do the following (assuming we know a reasonable upper bound on the VC-dimension of the class of weak hypotheses). Let h_f^T be the hypothesis generated by running **AdaBoost** for T iterations. By combining the observed empirical error of h_f^T with the bounds given in Theorems 7 and 8, we can compute an upper bound on the generalization error of h_f^T for all T. We would then select the hypothesis h_f^T that minimizes the guaranteed upper bound.

While structural risk minimization is a mathematically sound method, the upper bounds on ϵ_g that are generated in this way might be larger than the actual value and so the chosen number of iterations T might be much smaller than the optimal value, leading to inferior performance. A simple alternative is to use "cross-validation" in which a fraction of the training set is left outside the set used to generate h_f as the so-called "validation" set. The value of T is then chosen to be the one for which the error of the final hypothesis on the validation set is minimized. (For an extensive analysis of the relations between different methods for selecting model complexity in learning, see Kearns et al. [17].)

Some initial experiments using **AdaBoost** on real-world problems conducted by ourselves and Drucker and Cortes [8] indicate that **AdaBoost** tends not to over-fit; on many problems, even after hundreds of rounds of boosting, the generalization error continues to drop, or at least does not increase.

4.4 A Bayesian interpretation

The final hypothesis generated by **AdaBoost** is closely related to one suggested by a Bayesian analysis. As usual, we assume that examples (x, y) are being generated according to some distribution $\mathcal{P}$ on $X \times \{0,1\}$; all probabilities in this subsection are taken with respect to $\mathcal{P}$. Suppose we are given a set of $\{0,1\}$-valued hypotheses $h_1, \ldots, h_T$ and that our goal is to combine the predictions of these hypotheses in the optimal way. Then, given an instance x and the hypothesis predictions $h_t(x)$, the Bayes optimal decision rule says that we should predict the label with the highest likelihood, given the hypothesis values, i.e., we should predict 1 if

$$\Pr\left[y = 1 \mid h_1(x), \ldots, h_T(x)\right] > \Pr\left[y = 0 \mid h_1(x), \ldots, h_T(x)\right],$$

and otherwise we should predict 0.

This rule is especially easy to compute if we assume that the errors of the different hypotheses are independent of one another and of the target concept, that is, if we assume that the event $h_t(x) \neq y$ is conditionally independent of the actual label y and the predictions of all the other hypotheses $h_1(x), \ldots, h_{t-1}(x), h_{t+1}(x), \ldots, h_T(x)$. In this case, by applying Bayes rule, we can rewrite the Bayes optimal decision rule in a particularly simple form in which we predict 1 if

$$\Pr\left[y = 1\right] \prod_{t:h_t(x)=0} \epsilon_t \prod_{t:h_t(x)=1} (1 - \epsilon_t) > \Pr\left[y = 0\right] \prod_{t:h_t(x)=0} (1 - \epsilon_t) \prod_{t:h_t(x)=1} \epsilon_t,$$

and 0 otherwise. Here $\epsilon_t = \Pr\left[h_t(x) \neq y\right]$. We add to the set of hypotheses the trivial hypothesis h_0 which always predicts the value 1. We can then replace $\Pr\left[y = 0\right]$ by ϵ_0. Taking the logarithm of both sides in this inequality and rearranging the terms, we find that the Bayes optimal 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. (This is sometimes called "naive Bayes.") An interesting and more principled alternative to this practice would be to use the algorithm **AdaBoost** to find a combination rule which, by Theorem 6, has a guaranteed non-trivial accuracy.

4.5 Improving the error bound

We show in this section how the bound given in Theorem 6 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

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

be a weighted average of the weak hypotheses h_t. We will here consider final hypotheses of the form $h_f(x_i) = F(r(x_i))$ where $F : [0,1] \rightarrow [0,1]$. For the version of **AdaBoost** given in

Figure 2, $F(r)$ 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 $h_f(x_i) \in [0,1]$, we can interpret h_f as a randomized hypothesis and $h_f(x_i)$ as the probability of predicting 1. Then the error $E_{i\sim D}[|h_f(x_i) - y_i|]$ is simply the probability of an incorrect prediction.

Theorem 9 *Let $\epsilon_1, \ldots, \epsilon_T$ be as in Theorem 6, and let $r(x_i)$ be as defined above. Let the modified final hypothesis be defined by $h_f = F(r(x_i))$ 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

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

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

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

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

5 Boosting for multi-class and regression problems

So far, we have restricted our attention to binary classification problems in which the set of labels Y contains only two elements. In this section, we describe two possible extensions of

AdaBoost to the multi-class case in which Y is any finite set of class labels. We also give an extension for a regression problem in which Y is a real bounded interval.

We start with the multiple-label classification problem. Let $Y = \{1, 2, \ldots, k\}$ be the set of possible labels. The boosting algorithms we present output hypotheses $h_f : X \to Y$, and the error of the final hypothesis is measured in the usual way as the probability of an incorrect prediction.

The first extension of **AdaBoost**, which we call **AdaBoost.M1**, is the most direct. The weak learner generates hypotheses which assign to each instance one of the k possible labels. We require that each weak hypothesis have prediction error less than 1/2 (with respect to the distribution on which it was trained). Provided this requirement can be met, we are able prove that the error of the combined final hypothesis decreases exponentially, as in the binary case. Intuitively, however, this requirement on the performance of the weak learner is stronger than might be desired. In the binary case ($k = 2$), a random guess will be correct with probability 1/2, but when $k > 2$, the probability of a correct random prediction is only $1/k < 1/2$. Thus, our requirement that the accuracy of the weak hypothesis be greater than 1/2 is significantly stronger than simply requiring that the weak hypothesis perform better than random guessing.

In fact, when the performance of the weak learner is measured only in terms of error rate, this difficulty is unavoidable as is shown by the following informal example (also presented by Schapire [22]): Consider a learning problem where $Y = \{0, 1, 2\}$ and suppose that it is "easy" to predict whether the label is 2 but "hard" to predict whether the label is 0 or 1. Then a hypothesis which predicts correctly whenever the label is 2 and otherwise guesses randomly between 0 and 1 is guaranteed to be correct at least half of the time (significantly beating the 1/3 accuracy achieved by guessing entirely at random). On the other hand, boosting this learner to an arbitrary accuracy is infeasible since we assumed that it is hard to distinguish 0- and 1-labeled instances.

As a more natural example of this problem, consider classification of handwritten digits in an OCR application. It may be easy for the weak learner to tell that a particular image of a "7" is not a "0" but hard to tell for sure if it is a "7" or a "9". Part of the problem here is that, although the boosting algorithm can focus the attention of the weak learner on the harder examples, it has no way of forcing the weak learner to discriminate between particular labels that may be especially hard to distinguish.

In our second version of multi-class boosting, we attempt to overcome this difficulty by extending the communication between the boosting algorithm and the weak learner. First, we allow the weak learner to generate more expressive hypotheses whose output is a vector in $[0, 1]^k$, rather than a single label in Y. Intuitively, the yth component of this vector represents a "degree of belief" that the correct label is y. The components with large values (close to 1) correspond to those labels considered to be plausible. Likewise, labels considered implausible are assigned a small value (near 0), and questionable labels may be assigned a value near 1/2. If several labels are considered plausible (or implausible), then they all may be assigned large (or small) values.

While we give the weak learning algorithm more expressive power, we also place a more complex requirement on the performance of the weak hypotheses. Rather than using the usual prediction error, we ask that the weak hypotheses do well with respect to a more sophisticated error measure that we call the pseudo-loss. This pseudo-loss varies from example to example, and from one round to the next. On each iteration, the pseudo-loss function is supplied to the weak learner by the boosting algorithm, along with the distribution on the examples. By manipulating the pseudo-loss function, the boosting algorithm can focus the weak learner on the labels that are hardest to discriminate. The boosting algorithm **AdaBoost.M2**, described in Section 5.2, is based on these ideas and achieves boosting if each weak hypothesis has pseudo-loss slightly better than random guessing (with respect to the pseudo-loss measure that was supplied to the weak learner).

In addition to the two extensions described in this paper, we mention an alternative, standard approach which would be to convert the given multi-class problem into several binary problems, and then to use boosting separately on each of the binary problems. There are several standard ways of making such a conversion, one of the most successful being the error-correcting output coding approach advocated by Dietterich and Bakiri [7].

Finally, in Section 5.3 we extend **AdaBoost** to boosting regression algorithms. In this case $Y = [0, 1]$, and the error of a hypothesis is defined as $\mathrm{E}_{(x,y)\sim\mathcal{P}}\left[(h(x) - y)^2\right]$. We describe a boosting algorithm **AdaBoost.R** which, using methods similar to those used in **AdaBoost.M2**, boosts the performance of a weak regression algorithm.

5.1 First multi-class extension

In our first and most direct extension to the multi-class case, the goal of the weak learner is to generate on round t a hypothesis $h_t : X \to Y$ with low classification error $\epsilon_t \doteq \Pr_{i\sim\mathbf{p}^t}\left[h_t(x_i) \neq y_i\right]$. Our extended boosting algorithm, called **AdaBoost.M1**, is shown in Figure 3, and differs only slightly from **AdaBoost**. The main difference is in the replacement of the error $|h_t(x_i) - y_i|$ for the binary case by $[\![h_t(x_i) \neq y_i]\!]$ where, for any predicate π, we define $[\![\pi]\!]$ to be 1 if π holds and 0 otherwise. Also, the final hypothesis h_f, for a given instance x, now outputs the label y that maximizes the sum of the weights of the weak hypotheses predicting that label.

In the case of binary classification ($k = 2$), a weak hypothesis h with error significantly larger than 1/2 is of equal value to one with error significantly less than 1/2 since h can be replaced by $1 - h$. However, for $k > 2$, a hypothesis h_t with error $\epsilon_t \geq 1/2$ is useless to the boosting algorithm. If such a weak hypothesis is returned by the weak learner, our algorithm simply halts, using only the weak hypotheses that were already computed.

Theorem 10 *Suppose the weak learning algorithm* **WeakLearn**, *when called by* **AdaBoost.M1**, *generates hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$, where ϵ_t is as defined in Figure 3. Assume each $\epsilon_t \leq 1/2$. Then the error $\epsilon = \Pr_{i\sim D}\left[h_f(x_i) \neq y_i\right]$ of the final hypothesis h_f output by* **AdaBoost.M1** *is bounded above by*

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

Proof: To prove this theorem, we reduce our setup for **AdaBoost.M1** to an instantiation of **AdaBoost**, and then apply Theorem 6. For clarity, we mark with tildes variables in the reduced **AdaBoost** space. For each of the given examples (x_i, y_i), we define an **AdaBoost** example $(\tilde{x}_i, \tilde{y}_i)$ in which $\tilde{x}_i = i$ and $\tilde{y}_i = 0$. We define the **AdaBoost** distribution $\tilde{D}$ over examples to be equal to the **AdaBoost.M1** distribution D. On the tth round, we provide **AdaBoost** with a hypothesis $\tilde{h}_t$ defined by the rule

$$\tilde{h}_t(i) = [\![h_t(x_i) \neq y_i]\!]$$

in terms of the tth hypothesis h_t which was returned to **AdaBoost.M1** by **WeakLearn**.

Given this setup, it can be easily proved by induction on the number of rounds that the weight vectors, distributions and errors computed by **AdaBoost** and **AdaBoost.M1** are identical so that $\tilde{\mathbf{w}}^t = \mathbf{w}^t$, $\tilde{\mathbf{p}}^t = \mathbf{p}^t$, $\tilde{\epsilon}_t = \epsilon_t$ and $\tilde{\beta}_t = \beta_t$.

Algorithm AdaBoost.M1

Input: sequence of N examples $\langle(x_1, y_1), \ldots, (x_N, y_N)\rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$
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 $h_t : X \to Y$.

3. Calculate the error of h_t: $\epsilon_t = \sum_{i=1}^N p_i^t [\![h_t(x_i) \neq y_i]\!]$.
If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.

4. Set $\beta_t = \epsilon_t/(1 - \epsilon_t)$.

5. Set the new weights vector to be
$$w_i^{t+1} = w_i^t \beta_t^{1-[\![h_t(x_i) \neq y_i]\!]}$$

Output the hypothesis
$$h_f(x) = \arg\max_{y \in Y} \sum_{t=1}^T \left(\log \frac{1}{\beta_t}\right) [\![h_t(x) = y]\!].$$

Figure 3: A first multi-class extension of **AdaBoost**.

Suppose that **AdaBoost.M1**'s final hypothesis h_f makes a mistake on instance i so that $h_f(x_i) \neq y_i$. Then, by definition of h_f,

$$\sum_{t=1}^{T} \alpha_t [\![h_t(x_i) = y_i]\!] \leq \sum_{t=1}^{T} \alpha_t [\![h_t(x_i) = h_f(x_i)]\!]$$

where $\alpha_t = \ln(1/\beta_t)$. This implies

$$\sum_{t=1}^{T} \alpha_t [\![h_t(x_i) = y_i]\!] \leq \frac{1}{2} \sum_{t=1}^{T} \alpha_t,$$

using the fact that each $\alpha_t \geq 0$ since $\epsilon_t \leq 1/2$. By definition of $\tilde{h}_t$, this implies

$$\sum_{t=1}^{T} \alpha_t \tilde{h}_t(i) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t,$$

so $\tilde{h}_f(i) = 1$ by definition of the final **AdaBoost** hypothesis.

Therefore,

$$\Pr_{i \sim D}\left[h_f(x_i) \neq y_i\right] \leq \Pr_{i \sim D}\left[\tilde{h}_f(i) = 1\right].$$

Since each **AdaBoost** instance has a 0-label, $\Pr_{i \sim D}\left[\tilde{h}_f(i) = 1\right]$ is exactly the error of $\tilde{h}_f$. Applying Theorem 6, we can obtain a bound on this error, completing the proof. ■

It is possible, for this version of the boosting algorithm, to allow hypotheses which generate for each x, not only a predicted class label $h(x) \in Y$, but also a "confidence" $\kappa(x) \in [0, 1]$. The learner then suffers loss $1/2 - \kappa(x)/2$ if its prediction is correct and $1/2 + \kappa(x)/2$ otherwise. (Details omitted.)

5.2 Second multi-class extension

In this section we describe a second alternative extension of **AdaBoost** to the case where the label space Y is finite. This extension requires more elaborate communication between the boosting algorithm and the weak learning algorithm. The advantage of doing this is that it gives the weak learner more flexibility in making its predictions. In particular, it sometimes enables the weak learner to make useful contributions to the accuracy of the final hypothesis even when the weak hypothesis does not predict the correct label with probability greater than 1/2.

As described above, the weak learner generates hypotheses which have the form $h : X \times Y \to [0, 1]$. Roughly speaking, $h(x, y)$ measures the degree to which it is believed that y is the correct label associated with instance x. If, for a given x, $h(x, y)$ attains the same value for all y then we say that the hypothesis is *uninformative* on instance x. On the other hand, any deviation from strict equality is potentially informative, because it predicts some labels to be more plausible than others. As will be seen, any such information is potentially useful for the boosting algorithm.

Below, we formalize the goal of the weak learner by defining a pseudo-loss which measures the goodness of the weak hypotheses. To motivate our definition, we first consider the following setup. For a fixed training example (x_i, y_i), we use a given hypothesis h to answer $k - 1$ binary questions. For each of the incorrect labels $y \neq y_i$ we ask the question:

"Which is the label of x_i: y_i or y?"

In other words, we ask that the correct label y_i be discriminated from the incorrect label y.

Assume momentarily that h only takes values in $\{0,1\}$. Then if $h(x_i, y) = 0$ and $h(x_i, y_i) = 1$, we interpret h's answer to the question above to be y_i (since h deems y_i to be a plausible label for x_i, but y is considered implausible). Likewise, if $h(x_i, y) = 1$ and $h(x_i, y_i) = 0$ then the answer is y. If $h(x_i, y) = h(x_i, y_i)$, then one of the two answers is chosen uniformly at random.

In the more general case that h takes values in $[0,1]$, we interpret $h(x,y)$ as a randomized decision for the procedure above. That is, we first choose a random bit $b(x,y)$ which is 1 with probability $h(x,y)$ and 0 otherwise. We then apply the above procedure to the stochastically chosen binary function b. The probability of choosing the incorrect answer y to the question above is

$$\Pr\left[b(x_i, y_i) = 0 \wedge b(x_i, y) = 1\right] + \tfrac{1}{2}\Pr\left[b(x_i, y_i) = b(x_i, y)\right] = \tfrac{1}{2}(1 - h(x_i, y_i) + h(x_i, y)).$$

If the answers to all $k-1$ questions are considered equally important, then it is natural to define the loss of the hypothesis to be the average, over all $k-1$ questions, of the probability of an incorrect answer:

$$\frac{1}{k-1}\sum_{y \neq y_i} \tfrac{1}{2}(1 - h(x_i, y_i) + h(x_i, y)) \quad = \quad \tfrac{1}{2}\left(1 - h(x_i, y_i) + \frac{1}{k-1}\sum_{y \neq y_i} h(x_i, y)\right). \qquad (24)$$

However, as was discussed in the introduction to Section 5, different discrimination questions are likely to have different importance in different situations. For example, considering the OCR problem described earlier, it might be that at some point during the boosting process, some example of the digit "7" has been recognized as being either a "7" or a "9". At this stage the question that discriminates between "7" (the correct label) and "9" is clearly much more important than the other eight questions that discriminate "7" from the other digits.

A natural way of attaching different degrees of importance to the different questions is to assign a weight to each question. So, for each instance x_i and incorrect label $y \neq y_i$, we assign a weight $q(i,y)$ which we associate with the question that discriminates label y from the correct label y_i. We then replace the average used in Equation (24) with an average *weighted* according to $q(i,y)$; the resulting formula is called the *pseudo-loss* of h on training instance i with respect to q:

$$\mathrm{ploss}_q(h, i) \doteq \tfrac{1}{2}\left(1 - h(x_i, y_i) + \sum_{y \neq y_i} q(i,y) h(x_i, y)\right).$$

The function $q : \{1, \ldots, N\} \times Y \to [0,1]$, called the *label weighting function*, assigns to each example i in the training set a probability distribution over the $k-1$ discrimination problems defined above. So, for all i,

$$\sum_{y \neq y_i} q(i,y) = 1.$$

The weak learner's goal is to minimize the expected pseudo-loss for given distribution D and weighting function q:

$$\mathrm{ploss}_{D,q}(h) := \mathrm{E}_{i \sim D}\left[\mathrm{ploss}_q(h, i)\right].$$

As we have seen, by manipulating both the distribution on instances, and the label weighting function q, our boosting algorithm effectively forces the weak learner to focus not only on the hard instances, but also on the incorrect class labels that are hardest to eliminate. Conversely, this pseudo-loss measure may make it easier for the weak learner to get a weak advantage. For instance, if the weak learner can simply determine that a particular instance does *not* belong to a certain class (even if it has no idea which of the remaining classes is the correct one), then, depending on q, this may be enough to gain a weak advantage.

Theorem 11, the main result of this section, shows that a weak learner can be boosted if it can consistently produce weak hypotheses with pseudo-losses smaller than 1/2. Note that pseudo-loss 1/2 can be achieved trivially by any uninformative hypothesis. Furthermore, a weak hypothesis h with pseudo-loss $\epsilon > 1/2$ is also beneficial to boosting since it can be replaced by the hypothesis $1 - h$ whose pseudo-loss is $1 - \epsilon < 1/2$.

Example 5. As a simple example illustrating the use of pseudo-loss, suppose we seek an *oblivious* weak hypothesis, i.e., a weak hypothesis whose value depends only on the class label y so that $h(x, y) = h(y)$ for all x. Although oblivious hypotheses per se are generally too weak to be of interest, it may often be appropriate to find the best oblivious hypothesis on a *part* of the instance space (such as the set of instances covered by a leaf of a decision tree).

Let D be the target distribution, and q the label weighting function. For notational convenience, let us define $q(i, y_i) = -1$ for all i so that

$$\mathrm{ploss}_q(h, i) = \tfrac{1}{2}\left(1 + \sum_{y \in Y} q(i, y) h(x_i, y)\right).$$

Setting $\delta(y) = \sum_i D(i) q(i, y)$, it can be verified that for an oblivious hypothesis h,

$$\mathrm{ploss}_{D,q}(h) = \tfrac{1}{2}\left(1 + \sum_{y \in Y} h(y)\delta(y)\right),$$

which is clearly minimized by the choice

$$h(y) = \begin{cases} 1 & \text{if } \delta(y) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

Suppose now that $q(i, y) = 1/(k-1)$ for $y \neq y_i$, and let $d(y) = \Pr_{i \sim D}[y_i = y]$ be the proportion of examples with label y. Then it can be verified that h will always have pseudo-loss strictly smaller than 1/2 except in the case of a uniform distribution of labels ($d(y) = 1/k$ for all y). In contrast, when the weak learner's goal is minimization of prediction error (as in Section 5.1), it can be shown that an oblivious hypothesis with prediction error strictly less than 1/2 can only be found when one label y covers more than 1/2 the distribution ($d(y) > 1/2$). So in this case, it is much easier to find a hypothesis with small pseudo-loss rather than small prediction error.

On the other hand, if $q(i, y) = 0$ for some values of y, then the quality of prediction on these labels is of no consequence. In particular, if $q(i, y) = 0$ for all but one incorrect label for each instance i, then in order to make the pseudo-loss smaller than 1/2 the hypothesis has to predict the correct label with probability larger than 1/2, which means that in this case the pseudo-loss criterion is as stringent as the usual prediction error. However, as discussed above, this case is unavoidable because a hard binary classification problem can always be embedded in a multi-class problem.

This example suggests that it may often be significantly easier to find weak hypotheses with small pseudo-loss rather than hypotheses whose prediction error is small. On the other hand, our theoretical bound for boosting using the prediction error (Theorem 10) is stronger than the bound for ploss (Theorem 11). Empirical tests [12] have shown that pseudo-loss is generally more successful when the weak learners use very restricted hypotheses. However, for more powerful weak learners, such as decision-tree learning algorithms, there is little difference between using pseudo-loss and prediction error.

Our algorithm, called **AdaBoost.M2**, is shown in Figure 4. Here, we maintain weights $w^t_{i,y}$ for each instance i and each label $y \in Y - \{y_i\}$. The weak learner must be provided both with a distribution D_t and a label weight function q_t. Both of these are computed using the weight vector $\mathbf{w}^t$ as shown in Step 1. The weak learner's goal then is to minimize the pseudo-loss ϵ_t, as defined in Step 3. The weights are updated as shown in Step 5. The final hypothesis h_f outputs, for a given instance x, the label y that maximizes a weighted average of the weak hypothesis values $h_t(x, y)$.

Theorem 11 *Suppose the weak learning algorithm* **WeakLearn***, when called by* **AdaBoost.M2** *generates hypotheses with pseudo-losses $\epsilon_1, \ldots, \epsilon_T$, where ϵ_t is as defined in Figure 4. Then the error $\epsilon = \Pr_{i \sim D}[h_f(i) \neq y_i]$ of the final hypothesis h_f output by* **AdaBoost.M2** *is bounded above by*

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

Proof: As in the proof of Theorem 10, we reduce to an instance of **AdaBoost** and apply Theorem 6. As before, we mark **AdaBoost** variables with a tilde.

For each training instance (x_i, y_i) and for each incorrect label $y \in Y - \{y_i\}$, we define one **AdaBoost** instance $\tilde{x}_{i,y} = (i, y)$ with associated label $\tilde{y}_{i,y} = 0$. Thus, there are $\tilde{N} = N(k-1)$ **AdaBoost** instances, each indexed by a pair (i, y). The distribution over these instances is defined to be $\tilde{D}(i, y) = D(i)/(k-1)$. The tth hypothesis $\tilde{h}_t$ provided to **AdaBoost** for this reduction is defined by the rule

$$\tilde{h}_t(i, y) = \tfrac{1}{2}(1 - h_t(x_i, y_i) + h_t(x_i, y)).$$

With this setup, it can be verified that the computed distributions and errors will be identical so that $\tilde{w}^t_{i,y} = w^t_{i,y}$, $\tilde{p}^t_{i,y} = p^t_{i,y}$, $\tilde{\epsilon}_t = \epsilon_t$ and $\tilde{\beta}_t = \beta_t$.

Suppose now that $h_f(x_i) \neq y_i$ for some example i. Then, by definition of h_f,

$$\sum_{t=1}^{T} \alpha_t h_t(x_i, y_i) \leq \sum_{t=1}^{T} \alpha_t h_t(x_i, h_f(x_i)),$$

where $\alpha_t = \ln(1/\beta_t)$. This implies that

$$\sum_{t=1}^{T} \alpha_t \tilde{h}_t(i, h_f(x_i)) = \tfrac{1}{2}\sum_{t=1}^{T} \alpha_t (1 - h_t(x_i, y_i) + h_t(x_i, h_f(x_i))) \geq \tfrac{1}{2}\sum_{t=1}^{T} \alpha_t$$

so $\tilde{h}_f(i, h_f(x_i)) = 1$ by definition of $\tilde{h}_f$.

Therefore,

$$\Pr_{i \sim D}[h_f(x_i) \neq y_i] \leq \Pr_{i \sim D}\left[\exists y \neq y_i : \tilde{h}_f(i, y) = 1\right].$$

Algorithm AdaBoost.M2

Input: sequence of N examples $\langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$
distribution D over the examples
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize the weight vector: $w^1_{i,y} = D(i)/(k-1)$ for $i = 1, \ldots, N$, $y \in Y - \{y_i\}$.

Do for $t = 1, 2, \ldots, T$

1. Set $W^t_i = \sum_{y \neq y_i} w^t_{i,y}$;
$$q_t(i, y) = \frac{w^t_{i,y}}{W^t_i}$$
for $y \neq y_i$; and set
$$D_t(i) = \frac{W^t_i}{\sum_{i=1}^N W^t_i}.$$

2. Call **WeakLearn**, providing it with the distribution D_t and label weighting function q_t; get back a hypothesis $h_t : X \times Y \to [0, 1]$.

3. Calculate the pseudo-loss of h_t:
$$\epsilon_t = \tfrac{1}{2} \sum_{i=1}^N D_t(i) \left(1 - h_t(x_i, y_i) + \sum_{y \neq y_i} q_t(i, y) h_t(x_i, y) \right).$$

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.

5. Set the new weights vector to be
$$w^{t+1}_{i,y} = w^t_{i,y} \beta_t^{(1/2)(1 + h_t(x_i, y_i) - h_t(x_i, y))}$$
for $i = 1, \ldots, N$, $y \in Y - \{y_i\}$.

Output the hypothesis
$$h_f(x) = \arg\max_{y \in Y} \sum_{t=1}^T \left(\log \frac{1}{\beta_t} \right) h_t(x, y).$$

Figure 4: A second multi-class extension of **AdaBoost**.

Since all **AdaBoost** instances have a 0-label, and by definition of $\tilde{D}$, the error of $\tilde{h}_f$ is

$$\Pr_{(i,y)\sim\tilde{D}}\left[\tilde{h}_f(i,y)=1\right] \geq \frac{1}{k-1}\Pr_{i\sim D}\left[\exists y\neq y_i : \tilde{h}_f(i,y)=1\right].$$

Applying Theorem 6 to bound the error of $\tilde{h}_f$, this completes the proof. ■

Although we omit the details, the bound for **AdaBoost.M2** can be improved by a factor of two in a manner similar to that described in Section 4.5.

5.3 Boosting regression algorithms

In this section we show how boosting can be used for a regression problem. In this setting, the label space is $Y=[0,1]$. As before, the learner receives examples (x,y) chosen at random according to some distribution $\mathcal{P}$, and its goal is to find a hypothesis $h : X\to Y$ which, given some x value, predicts approximately the value y that is likely to be seen. More precisely, the learner attempts to find an h with small *mean squared error* (MSE):

$$\mathrm{E}_{(x,y)\sim\mathcal{P}}\left[(h(x)-y)^2\right]. \tag{25}$$

Our methods can be applied to any reasonable bounded error measure, but, for the sake of concreteness, we concentrate here on the squared error measure.

Following our approach for classification problems, we assume that the learner has been provided with a training set $(x_1,y_1),\ldots,(x_N,y_N)$ of examples distributed according to $\mathcal{P}$, and we focus only on the minimization of the empirical MSE:

$$\frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2.$$

Using techniques similar to those outlined in Section 4.3, the true MSE given in Equation (25) can be related to the empirical MSE.

To derive a boosting algorithm in this context, we reduce the given regression problem to a binary classification problem, and then apply **AdaBoost**. As was done for the reductions used in the proofs of Theorems 10 and 11, we mark with tildes all variables in the reduced (**AdaBoost**) space. For each example (x_i,y_i) in the training set, we define a continuum of examples indexed by pairs (i,y) for all $y\in[0,1]$: the associated instance is $\tilde{x}_{i,y}=(x_i,y)$, and the label is $\tilde{y}_{i,y}=[\![y\geq y_i]\!]$. (Recall that $[\![\pi]\!]$ is 1 if predicate π holds and 0 otherwise.) Although it is obviously infeasible to explicitly maintain an infinitely large training set, we will see later how this method can be implemented efficiently. Also, although the results of Section 4 only dealt with finitely large training sets, the extension to infinite training sets is straightforward.

Thus, informally, each instance (x_i,y_i) is mapped to an infinite set of binary questions, one for each $y\in Y$, and each of the form: "Is the correct label y_i bigger or smaller than y?"

In a similar manner, each hypothesis $h : X\to Y$ is reduced to a binary-valued hypothesis $\tilde{h} : X\times Y\to\{0,1\}$ defined by the rule

$$\tilde{h}(x,y)=[\![y\geq h(x)]\!].$$

Thus, $\tilde{h}$ attempts to answer these binary questions in a natural way using the estimated value $h(x)$.

Finally, as was done for classification problems, we assume we are given a distribution D over the training set; ordinarily, this will be uniform so that $D(i)=1/N$. In our reduction, this distribution is mapped to a density $\tilde{D}$ over pairs (i, y) in such a way that minimization of classification error in the reduced space is equivalent to minimization of MSE for the original problem. To do this, we define

$$\tilde{D}(i, y) = \frac{D(i)|y - y_i|}{Z}$$

where Z is a normalization constant:

$$Z = \sum_{i=1}^{N} D(i) \int_0^1 |y - y_i| dy.$$

It is straightforward to show that $1/4 \leq Z \leq 1/2$.

If we calculate the binary error of $\tilde{h}$ with respect to the density $\tilde{D}$, we find that, as desired, it is directly proportional to the mean squared error:

$$\begin{aligned} \sum_{i=1}^{N} \int_0^1 \left| \tilde{y}_{i,y} - \tilde{h}(\tilde{x}_{i,y}) \right| \tilde{D}(i, y) dy &= \frac{1}{Z} \sum_{i=1}^{N} D(i) \left| \int_{y_i}^{h(x_i)} |y - y_i| dy \right| \\ &= \frac{1}{2Z} \sum_{i=1}^{N} D(i)(h(x_i) - y_i)^2. \end{aligned}$$

The constant of proportionality is $1/(2Z) \in [1, 2]$.

Unraveling this reduction, we obtain the regression boosting procedure **AdaBoost.R** shown in Figure 5. As prescribed by the reduction, **AdaBoost.R** maintains a weight $w_{i,y}^t$ for each instance i and label $y \in Y$. The initial weight function $\mathbf{w}^1$ is exactly the density $\tilde{D}$ defined above. By normalizing the weights $\mathbf{w}^t$, a density $\mathbf{p}^t$ is defined at Step 1 and provided to the weak learner at Step 2. The goal of the weak learner is to find a hypothesis $h_t : X \to Y$ that minimizes the loss ϵ_t defined in Step 3. Finally, at Step 5, the weights are updated as prescribed by the reduction.

The definition of ϵ_t at Step 3 follows directly from the reduction above; it is exactly the classification error of $\tilde{h}_f$ in the reduced space. Note that, similar to **AdaBoost.M2**, **AdaBoost.R** not only varies the distribution over the examples (x_i, y_i), but also modifies from round to round the definition of the loss suffered by a hypothesis on each example. Thus, although our ultimate goal is minimization of the squared error, the weak learner must be able to handle loss functions that are more complicated than MSE.

The final hypothesis h_f also is consistent with the reduction. Each reduced weak hypothesis $\tilde{h}_t(x, y)$ is non-decreasing as a function of y. Thus, the final hypothesis $\tilde{h}_f$ generated by **AdaBoost** in the reduced space, being the threshold of a weighted sum of these hypotheses, also is non-decreasing as a function of y. As the output of $\tilde{h}_f$ is binary, this implies that for every x there is one value of y for which $\tilde{h}_f(x, y') = 0$ for all $y' < y$ and $\tilde{h}_f(x, y') = 1$ for all $y' > y$. This is exactly the value of y given by $h_f(x)$ as defined in the figure. Note that h_f is actually computing a *weighted median* of the weak hypotheses.

At first, it might seem impossible to maintain weights $w_{i,y}^t$ over an uncountable set of points. However, on closer inspection, it can be seen that, when viewed as a function of y, $w_{i,y}^t$ is a piece-wise linear function. For $t = 1$, $w_{i,y}^1$ has two linear pieces, and each update at Step 5 potentially breaks one of the pieces in two at the point $h_t(x_i)$. Initializing, storing and updating such piece-wise linear functions are all straightforward operations. Also, the integrals which appear in the figure can be evaluated explicitly since these only involve integration of piece-wise linear functions.

Algorithm AdaBoost.R

Input: sequence of N examples $\langle(x_1, y_1), \ldots, (x_N, y_N)\rangle$ with labels $y_i \in Y = [0,1]$
distribution D over the examples
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize the weight vector:

$$w_{i,y}^1 = \frac{D(i)|y - y_i|}{Z}$$

for $i = 1, \ldots, N$, $y \in Y$, where

$$Z = \sum_{i=1}^{N} D(i) \int_0^1 |y - y_i| dy.$$

Do for $t = 1, 2, \ldots, T$

1. Set
$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N \int_0^1 w_{i,y}^t dy}.$$

2. Call **WeakLearn**, providing it with the density $\mathbf{p}^t$; get back a hypothesis $h_t : X \to Y$.

3. Calculate the loss of h_t:
$$\epsilon_t = \sum_{i=1}^{N} \left| \int_{y_i}^{h_t(x_i)} p_{i,y}^t dy \right|.$$
If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.

4. Set $\beta_t = \epsilon_t/(1 - \epsilon_t)$.

5. Set the new weights vector to be
$$w_{i,y}^{t+1} = \begin{cases} w_{i,y}^t & \text{if } y_i \leq y \leq h_t(x_i) \text{ or } h_t(x_i) \leq y \leq y_i \\ w_{i,y}^t \beta_t & \text{otherwise.} \end{cases}$$
for $i = 1, \ldots, N$, $y \in Y$.

Output the hypothesis

$$h_f(x) = \inf\{y \in Y : \sum_{t: h_t(x) \leq y} \log(1/\beta_t) \geq \frac{1}{2} \sum_t \log(1/\beta_t)\}.$$

Figure 5: An extension of **AdaBoost** to regression problems.

The following theorem describes our performance guarantee for **AdaBoost.R**. The proof follows from the reduction described above coupled with a direct application of Theorem 6.

Theorem 12 *Suppose the weak learning algorithm* **WeakLearn***, when called by* **AdaBoost.R***, generates hypotheses with errors* $\epsilon_1, \ldots, \epsilon_T$*, where* ϵ_t *is as defined in Figure 5. Then the mean squared error* $\epsilon = \mathrm{E}_{i \sim D}\left[(h_f(x_i) - y_i)^2\right]$ *of the final hypothesis* h_f *output by* **AdaBoost.R** *is bounded above by*

$$\epsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}. \tag{26}$$

An unfortunate property of this setup is that there is no trivial way to generate a hypothesis whose loss is 1/2. This is a similar situation to the one we encountered with algorithm **AdaBoost.M1**. A remedy to this problem might be to allow weak hypotheses from a more general class of functions. One simple generalization is to allow for weak hypotheses that are defined by two functions: $h : X \to [0,1]$ as before, and $\kappa : X \to [0,1]$ which associates a measure of confidence to each prediction of h. The reduced hypothesis which we associate with this pair of functions is

$$\tilde{h}(x,y) = \begin{cases} (1+\kappa(x))/2 & \text{if } h(x) \geq y \\ (1-\kappa(x))/2 & \text{otherwise.} \end{cases}$$

These hypotheses are used in the same way as the ones defined before and a slight variation of algorithm **AdaBoost.R** can be used to boost the accuracy of these more general weak learners (details omitted). The advantage of this variant is that any hypothesis for which $\kappa(x)$ is identically zero has pseudo-loss exactly 1/2 and slight deviations from this hypothesis can be used to encode very weak predictions.

The method presented in this section for boosting with square loss can be used with any reasonable bounded loss function $L : Y \times Y \to [0,1]$. Here, $L(y', y)$ is a measure of the "discrepancy" between the observed label y and a predicted label y'; for instance, above we used $L(y', y) = (y' - y)^2$. The goal of learning is to find a hypothesis h with small average loss $\mathrm{E}_{(x,y)\sim\mathcal{P}}[L(h(x), y)]$. Assume, for any y, that $L(y, y) = 0$ and that $L(y', y)$ is differentiable with respect to y', non-increasing for $y' \leq y$ and non-decreasing for $y' \geq y$. Then, to modify **AdaBoost.R** to handle such a loss function, we need only replace $|y - y_i|$ in the initialization step with $|\partial L(y, y_i)/\partial y|$. The rest of the algorithm is unchanged, and the modifications needed for the analysis are straightforward.

Acknowledgments

Thanks to Corinna Cortes, Harris Drucker, David Helmbold, Keith Messer, Volodya Vovk and Manfred Warmuth for helpful discussions.

References

[1] Eric B. Baum and David Haussler. What size net gives valid generalization? In *Advances in Neural Information Processing Systems I*, pages 81–90. Morgan Kaufmann, 1989.

[2] David Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, Spring 1956.

[3] Leo Breiman. Bias, variance, and arcing classifiers. Unpublished manuscript. Available from ftp://ftp.stat.berkeley.edu/pub/users/breiman/arcall.ps.Z., 1996.

[4] Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 382–391, 1993.

[5] 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.

[6] Thomas M. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, January 1991.

[7] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[8] Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, 1996.

[9] 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.

[10] 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.

[11] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, To appear. An extended abstract appeared in *Proceedings of the Third Annual Workshop on Computational Learning Theory*, 1990.

[12] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[13] Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

[14] 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.

[15] David Haussler, Jyrki Kivinen, and Manfred K. Warmuth. Tight worst-case loss bounds for predicting with expert advice. In *Computational Learning Theory: Second European Conference, EuroCOLT '95*, pages 69–83. Springer-Verlag, 1995.

[16] Jeffrey C. Jackson and Mark W. Craven. Learning sparse perceptrons. In *Advances in Neural Information Processing Systems 8*, 1996.

[17] Michael Kearns, Yishay Mansour, Andrew Y. Ng, and Dana Ron. An experimental and theoretical comparison of model selection methods. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, 1995.

[18] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[19] Jyrki Kivinen and Manfred K. Warmuth. Using experts for predicting continuous outcomes. In *Computational Learning Theory: EuroCOLT '93*, pages 109–120. Springer-Verlag, 1994.

[20] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

[21] J. Ross Quinlan. Bagging, boosting, and C4.5. In *Proceedings, Fourteenth National Conference on Artificial Intelligence*, 1996.

[22] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[23] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

[24] V. G. Vovk. A game of prediction with expert advice. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, 1995.

[25] Volodimir G. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.

[26] R. S. Wenocur and R. M. Dudley. Some special Vapnik-Chervonenkis classes. *Discrete Mathematics*, 33:313–318, 1981.

A Proof of Theorem 3

We start with a brief review of a framework used by Vovk [24], which is very similar to the framework used in Section 3. In this framework, an on-line decision problem consists of a decision space Δ, an outcome space Ω and a loss function $\lambda : \Delta \times \Omega \to [0, \infty]$, which associates a loss to each decision and outcome pair. At each trial t the learning algorithm receives the decisions $\varepsilon_1^t, \ldots, \varepsilon_N^t \in \Delta$ of N experts, and then generates its own decision $\delta^t \in \Delta$. Upon receiving an outcome $\omega^t \in \Omega$, the learner and each expert i incur loss $\lambda(\delta^t, \omega^t)$ and $\lambda(\varepsilon_i^t, \omega^t)$, respectively. The goal of the learning algorithm is to generate decisions in such a way that its cumulative loss will not be much larger than the cumulative loss of the best expert. The following four properties are assumed to hold:

1. Δ is a compact topological space.
2. For each ω, the function $\delta \to \lambda(\delta, \omega)$ is continuous.
3. There exists δ such that, for all ω, $\lambda(\delta, \omega) < \infty$.
4. There exists no δ such that, for all ω, $\lambda(\delta, \omega) = 0$.

We now give Vovk's main result [24]. Let a decision problem defined by Ω, Δ and λ obey Assumptions 1–4. Let c and a be positive real numbers. We say that the decision problem is (c, a)-*bounded* if there exists an algorithm A such that for any finite set of experts and for any finite sequence of trials, the cumulative loss of the algorithm is bounded by

$$\sum_{t=1}^{T} \lambda(\delta^t, \omega^t) \leq c \min_i \sum_{t=1}^{T} \lambda(\varepsilon_i^t, \omega^t) + a \ln N,$$

where N is the number of experts.

We say that a distribution $\mathcal{D}$ is *simple* if it is non-zero on a finite set denoted $\mathrm{dom}(\mathcal{D})$. Let $\mathcal{S}$ be the set of simple distributions over Δ. Vovk defines the following function $c : (0,1) \to [0,\infty]$ which characterizes the hardness of any decision problem:

$$c(\beta) = \sup_{\mathcal{D}\in\mathcal{S}} \inf_{\delta\in\Delta} \sup_{\omega\in\Omega} \frac{\lambda(\delta,\omega)}{\log_\beta \sum_{\varepsilon\in\mathrm{dom}(\mathcal{D})} \beta^{\lambda(\varepsilon,\omega)}\mathcal{D}(\varepsilon)}. \tag{27}$$

He then proves the following powerful theorem:

Theorem 13 (Vovk) *A decision problem is (c, a)-bounded if and only if for all $\beta \in (0,1)$, $c \geq c(\beta)$ or $a \geq c(\beta)/\ln(1/\beta)$.*

Proof of Theorem 3: The proof consists of the following three steps: We first define a decision problem that conforms to Vovk's framework. We then show a lower bound on the function $c(\beta)$ for this problem. Finally, we show how any algorithm A, for the on-line allocation problem can be used to generate decisions in the defined problem, and so we get from Theorem 13 a lower bound on the worst case cumulative loss of A.

The decision problem is defined as follows. We fix an integer $K > 1$ and set $\Delta = S_K$ where S_K is the K dimensional simplex, i.e., $S_K = \{\mathbf{x} \in [0,1]^K : \sum_{i=1}^K x_i = 1\}$. We set Ω to be the set of unit vectors in $\mathbb{R}^K$, i.e., $\Omega = \{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$ where $\mathbf{e}_i \in \{0,1\}^K$ has a 1 in the ith component, and 0 in all other components. Finally, we define the loss function to be $\lambda(\delta, \mathbf{e}_i) \doteq \delta \cdot \mathbf{e}_i = \delta_i$. One can easily verify that these definitions conform to Assumptions 1–4.

To prove a lower bound on $c(\beta)$ for this decision problem we choose a particular simple distribution over the decision space Δ. Let $\mathcal{D}$ be the uniform distribution over the unit vectors, i.e., $\mathrm{dom}(\mathcal{D}) = \{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$. For this distribution, we can explicitly calculate

$$c(\beta) \geq \inf_{\delta\in\Delta} \sup_{\omega\in\Omega} \frac{\lambda(\delta,\omega)}{\log_\beta \sum_{\varepsilon\in\mathrm{dom}(\mathcal{D})} \beta^{\lambda(\varepsilon,\omega)}\mathcal{D}(\varepsilon)}. \tag{28}$$

First, it is easy to see that the denominator in Equation (28) is a constant:

$$\sum_{\varepsilon\in\mathrm{dom}(\mathcal{D})} \beta^{\lambda(\varepsilon,\omega)}\mathcal{D}(\varepsilon) = \frac{\beta}{K} + \frac{K-1}{K}. \tag{29}$$

For any probability vector $\delta \in \Delta$, there must exist one component i for which $\delta_i \geq 1/K$. Thus

$$\inf_{\delta\in\Delta} \sup_{\omega\in\Omega} \lambda(\delta,\omega) = 1/K. \tag{30}$$

Combining Equations (28), (29) and (30), we get that

$$c(\beta) \geq \frac{\ln(1/\beta)}{K \ln(1 - \frac{1-\beta}{K})}. \tag{31}$$

We now show how an on-line allocation algorithm A can be used as a subroutine for solving this decision problem. We match each of the N experts of the decision problem with a strategy of the allocation problem. Each iteration t of the decision problem proceeds as follows.

1. Each of the N experts generates a decision $\varepsilon_i^t \in S_K$.

2. The algorithm A generates a distribution $\mathbf{p}^t \in S_N$.

3. The learner chooses the decision $\delta^t = \sum_{i=1}^N p_i^t \varepsilon_i^t$.

4. The outcome $\omega^t \in \Omega$ is generated.

5. The learner incurs loss $\delta^t \cdot \omega^t$, and each expert suffers loss $\varepsilon_i^t \cdot \omega^t$.

6. Algorithm A receives the loss vector $\boldsymbol{\ell}^t$ where $\ell_i^t = \varepsilon_i^t \cdot \omega^t$, and incurs loss

$$\mathbf{p}^t \cdot \boldsymbol{\ell}^t = \sum_{i=1}^N p_i^t (\varepsilon_i^t \cdot \omega^t) = (\sum_{i=1}^N p_i^t \varepsilon_i^t) \cdot \omega^t = \delta^t \cdot \omega^t.$$

Observe that the loss incurred by the learner in the decision problem is equal to the loss incurred by A. Thus, if for algorithm A we have an upper bound of the form

$$L_A \leq c \min_i L_i + a \ln N,$$

then the decision problem is (c, a)-bounded. On the other hand, using the lower bound given by Theorem 13 and the lower bound on $c(\beta)$ given in Equation (31), we get that for any K and any β, either

$$c \geq \frac{\ln(1/\beta)}{K \ln(1 - \frac{1-\beta}{K})} \quad \text{or} \quad a \geq \frac{1}{K \ln(1 - \frac{1-\beta}{K})} . \tag{32}$$

As K is a free parameter we can let $K \to \infty$ and the denominators in Equation (32) become $1 - \beta$ which gives the statement of the theorem. ■