The Weighted Majority Algorithm

Nick Littlestone [*]
Aiken Computation Laboratory
Harvard Univ.

Manfred K. Warmuth [†]
Dept. of Computer Sci.
U. C. Santa Cruz

Abstract

We study the construction of prediction algorithms in a situation in which a learner faces a sequence of trials, with a prediction to be made in each, and the goal of the learner is to make few mistakes. We are interested in the case that the learner has reason to believe that one of some pool of known algorithms will perform well, but the learner does not know which one. A simple and effective method, based on weighted voting, is introduced for constructing a compound algorithm in such a circumstance. We call this method the Weighted Majority Algorithm. We show that this algorithm is robust w.r.t. errors in the data. We discuss various versions of the Weighted Majority Algorithm and prove mistake bounds for them that are closely related to the mistake bounds of the best algorithms of the pool. For example, given a sequence of trials, if there is an algorithm in the pool $\mathcal{A}$ that makes at most m mistakes then the Weighted Majority Algorithm will make at most $c(\log|\mathcal{A}| + m)$ mistakes on that sequence, where c is fixed constant.

[*]Supported by ONR grant N00014-85-K-0445. Part of this research was done while this author was at the University of Calif. at Santa Cruz with support from ONR grant N00014-86-K-0454

[†]Supported by ONR grant N00014-86-K-0454. Part of this research was done while this author was on sabbatical at Aiken Computation Laboratory, Harvard, with partial support from the ONR grants N00014-85-K-0445 and N00014-86-K-0454

1 Introduction

We study on-line prediction algorithms that learn according to the following protocol. Learning proceeds in a sequence of trials. In each trial the algorithm receives an *instance* from some fixed *domain* and is to produce a binary *prediction*. At the end of the trial the algorithm receives a binary *reinforcement*, which can be viewed as the correct prediction for the instance. We evaluate such algorithms according to how many mistakes they make as in [Lit88,Lit89]. (A mistake occurs if the prediction and the reinforcement disagree.)

In this paper we investigate the situation where we are given a pool of prediction algorithms that make varying numbers of mistakes. We aim to design a *master algorithm* that uses the predictions of the pool to make its own prediction. Ideally the master algorithm should make not many more mistakes than the best algorithm of the pool, even though it does not have any *a priori* knowledge as to which of the algorithms of the pool make few mistakes for a given sequence of trials.

The overall protocol proceeds as follows in each trial: The same instance is fed to all algorithms of the pool. Each algorithm makes a prediction and these predictions are grouped together to form the instance that is fed to the master algorithm. The master algorithm then makes its prediction and receives a reinforcement, which it passes to the whole pool. We make no probabilistic assumptions about the choice of the instances and the reinforcements.

A special case considered in the paper occurs when each algorithm of the pool behaves like a function over the domain: its prediction on each point of the domain does not change over time. In this case the Halving Algorithm [Lit88] can be interpreted as a master algorithm. For each instance this algorithm predicts according to the majority of all *consistent* functions of the pool. (A function is consistent if its values agree with the reinforcements on all instances seen in the previous trials.) Note that the functions that are not consistent have been eliminated from the decision process. Each time the master algorithm makes a mistake a majority of the consistent functions are eliminated. If the sequence of trials is such that there is a consistent function in the pool F then the Halving Algorithm makes at most $\log_2 |F|$ mistakes. This type of bound goes back to [BF72].

We would like an algorithm that will still function well for sequences with which no function of the pool is consistent. This may happen, for example, if errors occur in some of the trials. We refer to the least number of inconsistent trials of any function in the pool as the number of *anomalies* of the sequence of trials with respect to that pool. If the number of anomalies is non-zero, the Halving Algorithm will eventually eliminate all functions of the pool, and there will be no functions left to base the future predictions on. The new master algorithm we develop here, called the *Weighted Majority Algorithm* (*WM*) is more robust.

Weighted Majority Algorithm (*WM*): *A non-negative weight is associated with each algorithm (function) of the pool. (All weights are initially one unless specified otherwise.) Algorithm WM forms its prediction by comparing the total weight q_0 of the algorithms of the pool that predict 0 to the total weight q_1 of the algorithms predicting 1. WM predicts according to the larger total (arbitrarily in case of a tie). When WM makes a mistake*[1]*, then the weights of those algorithms of the pool that agreed with the master algorithm are each multiplied by a fixed β such that* $0 \leq \beta < 1$.

In the case that $\beta = 0$ and the initial weights are equal, *WM* is identical to the Halving Algorithm. If $\beta > 0$, then *WM* gradually decreases the influence of functions that make a large number of mistakes and gives the functions that make few mistakes high relative weights. No single inconsistency can eliminate a function. Suppose that *WM* is applied to a pool F of functions and that the sequence of trials has m anomalies with respect to F. In this case *WM* makes no more than a constant times $\log |F| + m$ mistakes, where the constant depends on the fixed parameter β. In the case that the Vapnik-Chervonenkis dimension [VC71,BEHW88] of F is $\Omega(\log n)$ our lower bounds imply that *WM* is optimal (except for a multiplicative constant).

For the general case where *WM* is applied to a pool $\mathcal{A}$ of algorithms we show the following upper bounds on the number of mistakes made in a given sequence of trials:

1. $O(\log |\mathcal{A}| + m)$, if one algorithm of $\mathcal{A}$ makes at most m mistakes.
2. $O(\log \frac{|\mathcal{A}|}{k} + m)$, if each of a subpool of k algorithms of $\mathcal{A}$ makes at most m mistakes.
3. $O(\log \frac{|\mathcal{A}|}{k} + \frac{m}{k})$, if the total number of mistakes of a subpool of k algorithms of $\mathcal{A}$ is at most m.

[1]The mistake bounds that we prove in this paper will actually hold for two versions of *WM*, one that modifies weights only when a mistake is made (this version is given here) and one that modifies the weights at every trial by the multiplicative changes described.

In some sense these bounds show that *WM* selects the predictions of algorithms that are performing well on a given sequence.

Note that if the subpool size k is $\Omega(|\mathcal{A}|)$ then the bounds for cases 2 and 3 are $O(m)$ and $O(\frac{m}{k})$, respectively. We give an example of how Case 1 can be applied. Suppose that the instance domain is $\{0,1\}^n$. Boolean functions of the following form are called *r-of-k threshold functions*: $f(x_1,\ldots,x_n) = 1$ iff $\sum_{j=1}^{k} x_{i_j} \geq r$, where k, r and the distinct i_j are integers in the range from 1 to n. Assume we are given a sequence of trials that is consistent with such a threshold function. The goal is to design a prediction algorithm that makes a small number of mistakes for such sequences. The algorithm Winnow [Lit88] (the Fixed Threshold algorithm of [Lit89]) can be used to predict such sequences with a bound of $O(kn \log n)$ mistakes. If we know an upper bound on r, the mistake bound of Winnow can be improved to $b_r = O(kr \log n)$, which grows only logarithmically in the domain dimension n when k remains small. The improvement of the mistake bound to b_r requires choosing parameters for Winnow that depend on r (Let A_r denote Winnow when tuned to r). If A_r receives a sequence that is not consistent with any r'-of-k threshold function such that $r' \leq r$, then the number of mistakes that it makes might greatly exceed b_r.

We can overcome not knowing r by applying *WM* to the pool $\{A_{2^i}\}_{0 \leq i \leq \lceil \log_2 n \rceil}$. From the results presented in this paper, it follows that the number of mistakes made in this manner for a sequence consistent with an r-of-k threshold function will be bounded by a constant times $\log\log n + b_r$, which is $O(kr \log n)$.

The applications of the Weighted Majority Algorithm that we consider fall into two categories. The first category is illustrated by the previous example. Examples such as this one show that the Weighted Majority Algorithm is a powerful tool for constructing new efficient learning algorithms. It can be used in cases where there are several types of prediction algorithms available, or there is a choice of parameters for a learning algorithm, and the learner is unsure as to which choice is the best. The second category of use involves applying the Weighted Majority Algorithm to pools of functions. This use gives mistake bounds that grow at close to the optimal rate as the number of anomalies grows and establishes the best that can be achieved in this respect for deterministic algorithms. Many function classes of interest will, however, be too large to make this use of *WM* computationally practical.

In the next section, we prove mistake bounds for the Weighted Majority algorithm *WM*. We conclude the present section by outlining additional results presented in the full paper [LW89]. There, in addition to the basic algorithm, we discuss two variants. For one variant, which we call *WML*, we modify *WM* so that it never decreases its weights below a certain lower threshold. Algorithm *WML* has even stronger selective capabilities than *WM*. Suppose that we are given a sequence of trials such that there is one algorithm in the pool that makes few mistakes (say m_1) for an initial segment of the sequence and a second algorithm that makes m_2 mistakes for a second segment of sequence, and so forth. Assume the original sequence is partitioned into s segments. *WML* has no *a priori* knowledge as to how many segments there are, when the different segments begin, and which algorithms perform well in each segment. We can show that the number of mistakes made by *WML* is bounded by a constant times $(s \log |\mathcal{A}| + \sum_{i=1}^{s} m_i)$, where the constant depends on the parameters of the algorithm. For example, suppose that the algorithms of the pool are functions and each segment of the sequence is consistent with some function of the pool. Intuitively this means that the sequence is labeled according to some target function of the pool but at the end of each segment the target function changes. Each time the target changes to a new function, there is a cost of $O(\log|\mathcal{A}|)$ mistakes in the mistake bound for *WML*.

We investigate a second variant of *WM* which deals with countably infinite pools of algorithms, indexed by the positive integers. Barzdin and Freivald [BF72], considering pools of (recursive) functions, show that there is an algorithm that makes at most $\log_2 n + o(\log n)$ mistakes when given any sequence of trials consistent with the n-th function. We give a similar result that applies to pools of algorithms even in the case that no algorithm in the pool is consistent with the sequence of trials (i.e. every algorithm makes mistakes). We describe a variant *WMI* of the Weighted Majority Algorithm that has the property that for any countably infinite pool, any sequence of trials, and every index i, the number of mistakes it makes is bounded by a constant times $(\log i + m_i)$, where m_i is the number of mistakes made by the i-th algorithm of the pool on the given sequence of trials. More generally, the algorithms of the pool can be assigned any sequence of initial weights $w_1, w_2, \ldots$ with a finite sum, and the number of mistakes will be bounded by a constant times $log(1/w_i)+m_i$ for every i. The algorithm works by ignoring all but a finite active subpool of the algorithms, consisting of algorithms $A_1, \ldots, A_l$ of the pool, for some cutoff l. The active pool gradually grows. The cutoff l is chosen at each trial so that the sum of the weights of the inactive algorithms is small compared to the sum of the current weights of the active algorithms.

In the case where m_i grows rapidly with i, the active pool size can be kept small at the cost of at most a constant factor increase in the mistake bound that we obtain. One chooses the initial weights w_i so that $log(1/w_i)$ is comparable to m_i. In the full version of the paper, we discuss initial weight sequences $w_i = 1/(i(i+1))$, $w_i = 1/2^i$, and $w_i = 1/2^{2^{i-1}}$. The active pool size is smaller when the weights decrease more rapidly with i.

A portion of the full paper is devoted to the special case in which the basic algorithm *WM* is applied to pools of functions. We first assume that the pool contains a function consistent with all of the trials. Let m_i be a bound on the number of mistakes made by *WM* if the ith function in the pool is consistent. Changing the initial weights can be used to decrease some of the m_i at the expense of increasing others. For certain classes of functions we characterize what sets of m_i are possible mistake bounds for the Weighted Majority Algorithm, and present results from [Lit89] showing that for these classes of functions no other algorithm can do better. We then consider the case in which no function in the pool is consistent with all of the trials, and prove a lower bound on the rate at which the mistake bounds must grow as the number of anomalous trials grows.

We have begun exploring the properties of a randomized version of the Weighted Majority Algorithm. In the full paper, we give an expected mistake bound for the randomized algorithm. As described above, the deterministic Weighted Majority Algorithm predicts 1 if $\frac{q_1}{q_0+q_1} \geq \frac{1}{2}$. The randomized algorithm *WMR* instead predicts 1 with probability $\frac{q_1}{q_0+q_1}$. The randomized version of the algorithm has the property that the weights can be updated so that the rate at which *WMR* is expected to make mistakes in the long run can be made arbitrarily close to the rate at which the best prediction algorithm in the underlying pool makes mistakes. This represents an improvement by a factor of two over the limiting bound we give on the learning rate for the deterministic algorithm.

An alternate way to obtain similar performance bounds to those obtainable by the randomized algorithm is to use a deterministic algorithm that is allowed to hedge in its predictions. For this case, we allow the predictions of the Weighted Majority Algorithm and of the algorithms in the pool to be chosen from the closed interval $[0,1]$. We derive comparable bounds for this modification of the Weighted Majority Algorithm.

2 Proving Mistake Bounds for the Weighted Majority Algorithm

In this section we prove the bounds on the number of mistakes for the basic Weighted Majority Algorithm *WM*. Recall the description of *WM* given in the introduction. For a given trial, we use q_0 and q_1 to denote the total weight of the algorithms in the pool that predict 0 and 1, respectively. The parameter β is the factor by which weights are multiplied in case of a mistake and is always in the range $0 \leq \beta < 1$. Suppose that we run *WM* with a pool $\mathcal{A}$ of prediction algorithms, indexed with the integers from 1 through $|\mathcal{A}|$. We denote the initial non-negative weight corresponding to the i-th algorithm of the pool $(1 \leq i \leq |\mathcal{A}|)$ by w_i.

All proofs are surprisingly simple. We show that after each trial in which a mistake occurs the sum of the weights is at most u times the sum of the weights before the trial, for some $u < 1$. If the total initial weight is w_{init} and w_{fin} is a lower bound for the total final weight, then $w_{init}u^m \geq w_{fin}$ must hold, where m is the number of mistakes. This implies that m is at most $\frac{\log \frac{w_{init}}{w_{fin}}}{\log \frac{1}{u}}$.

Theorem 2.1 *Let $\mathcal{S}$ be any sequence of instances and reinforcements and let m_i be the number of mistakes made by the i-th algorithm of a pool $\mathcal{A}$ on the sequence $\mathcal{S}$. Let m be the number of mistakes made by WM on the sequence $\mathcal{S}$ when applied to the pool $\mathcal{A}$. Then $m \leq \frac{\log \frac{w_{init}}{w_{fin}}}{\log \frac{2}{1+\beta}}$, where $w_{fin} = \sum_{i=1}^{n} w_i \beta^{m_i}$.*

Proof. The weight of the i-th algorithm of the pool can only be updated (multiplied by β) during a trial if the algorithm makes a mistake in that trial. Thus after WM has processed the whole sequence $\mathcal{S}$, the weight of the i-th algorithm is at least $w_i\beta^{m_i}$ and the total current weight is at least w_{fin}.

During each trial $q_0 + q_1$ is the current total weight. Consider a trial in which a mistake is made. Suppose, without loss of generality, that in this trial the learner's prediction was 0, and thus $q_0 \geq q_1$. In this case the total weight after this trial will be $\beta q_0 + q_1 \leq \beta q_0 + q_1 + \frac{1-\beta}{2}(q_0 - q_1) = \frac{1+\beta}{2}(q_0 + q_1)$. Thus the ratio of the total weight after the trial to the total weight before the trial will be at most $\frac{1+\beta}{2} < 1$. The bound on m follows from the discussion preceding the theorem. □

As discussed in the introduction, if $\beta = 0$ and the initial weights are equal, then *WM* is the Halving algorithm. In that case, if all m_i are positive then $w_{fin} = 0$ and the bound of the theorem becomes vacuous.

For the following corollaries we assume that all initial weights are one. Otherwise we assume the same notation as in the theorem.

Corollary 2.2 *Assume that $\mathcal{A}$ is a pool of n prediction algorithms and that there is a subpool of $\mathcal{A}$ of size k such that each of the algorithms of the subpool makes at most m mistakes on $\mathcal{S}$. Then WM when applied to pool $\mathcal{A}$ makes at most $\frac{\log \frac{n}{k} + m \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ mistakes on the sequence $\mathcal{S}$.*

Proof. This follows from the above theorem and the fact that $w_{init} = n$ and $w_{fin} \geq k\beta^m$. □

Corollary 2.3 *Assume that $\mathcal{A}$ is a pool of n prediction algorithms and that there is a subpool of $\mathcal{A}$ of size k such that all algorithms of the subpool together make at most m mistakes in total on $\mathcal{S}$. Then WM when applied to pool $\mathcal{A}$ makes at most $\frac{\log \frac{n}{k} + \frac{m}{k} \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ mistakes on the sequence $\mathcal{S}$.*

Proof. Without loss of generality, let the first k algorithms of $\mathcal{A}$ be the subpool. Thus $\sum_{l=1}^{k} m_l \leq m$ and $w_{fin} \geq \sum_{l=1}^{k} \beta^{m_l}$. Since the latter sum is at least $k\beta^{\frac{m}{k}}$ it easy to derive the bound of the corollary using the previous theorem. $\square$

3 Acknowledgements

We thank David Helmbold for valuable discussions.

References

[BEHW88] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. *Learnability and the Vapnik-Chervonenkis Dimension.* Technical Report UCSC-CRL-87-20, University of California Computer Research Laboratory, Santa Cruz, CA, 1987, revised 1988. To appear in *JACM*.

[BF72] J. M. Barzdin and R. V. Freivald. On the prediction of general recursive functions. *Sov. Math. Dokl.*, 13:1224–1228, 1972.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[Lit89] Nick Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms.* PhD thesis, University of Calif., Santa Cruz, 1989.

[LW89] Nick Littlestone and Manfred K. Warmuth. *The Weighted Majority Algorithm.* Technical Report UCSC-CRL-89-16, Univ. of Calif., Santa Cruz, CA 95064, 1989.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Th. Prob. and its Appl.*, 16(2):264–80, 1971.