

The Weighted Majority Algorithm

Nick Littlestone *
Manfred K. Warmuth †

UCSC-CRL-91-28
Revised
October 26, 1992

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

*This research was primarily conducted while this author was at the University of Calif. at Santa Cruz with support from ONR grant N00014-86-K-0454, and at Harvard University, supported by ONR grant N00014-85-K-0445 and DARPA grant AFOSR-89-0506. Current address: NEC Research Institute, 4 Independence Way, Princeton, NJ 08540. E-mail address: nickl@research.nj.nec.com.

†Supported by ONR grants N00014-86-K-0454 and N00014-91-J-1162. 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. Address: Department of Computer Science, University of California at Santa Cruz. E-mail address: manfred@cs.ucsc.edu.

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 in the presence of 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.

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 *label*, which can be viewed as the correct prediction for the instance. We evaluate such algorithms according to how many mistakes they make [Ang88,BF72,Lit88,Lit89b]. (A mistake occurs if the prediction and the label disagree.) We also briefly discuss the case in which predictions and labels are chosen from the interval $[0, 1]$.

In this paper¹ 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 label, which it passes to the whole pool. We make no probabilistic assumptions about the choice of the instances and the labels; that is, the bounds we prove in this paper on the predictive performance of various algorithms all hold for a worst case sequence of instances and labels.

Our goal can be thought of as one of constructing learning algorithms that learn to take advantage of patterns in the input in order to make few mistakes. The scheme that we present for combining algorithms is general in the sense that, whatever types of patterns are handled by algorithms in the pool, these patterns are handled by the resulting combination of algorithms. We present two types of results: relative and absolute. The relative results give the number of mistakes of the master algorithm as a function of the number of mistakes of the pool members; these results do not depend on any details about the patterns handled by the algorithms. To apply the relative results we want to start with a pool of algorithms that contains one or more members capable of handling whatever input we face without many mistakes. If this pool is not too big, then the combination algorithm formed by applying the master algorithm to the pool will not make many mistakes. If, in addition, all of the algorithms in the pool are efficient and the pool is sufficiently small then the resulting algorithm will be computationally efficient.

For the absolute results we make particular assumptions about the input and look for a pool that will do well under these assumptions. We take a particular interest in patterns in the information seen by the learner in which labels (usually) depend functionally on the instances. The function that maps from the instances to the labels is called the *target function*. We say that a trial is consistent with a particular target function f if the label of the trial is given by the value of f at the instance of the trial. Given a class of potential

¹Notable differences of this paper from earlier versions of the paper [LW89a,LW89b] include more general derivations of bounds for various versions of the Weighted Majority Algorithm (Section 5) and the insertion here of some previously omitted proofs (Sections 7 and 8). In addition, we have modified algorithm *WMI* (now called *WMI*₂, Section 4), and have included some new results about infinite pools in Section 7. We have also amplified the discussion of the randomized algorithm *WMR* (Section 6).

target functions (a *target class*), we are interested in algorithms that make few mistakes when presented with any sequence of trials that is consistent with a target function chosen from the class. We sometimes also put additional structure on target classes, and ask that algorithms make fewer mistakes for some functions in a class (for example, functions that we consider simpler) than for others. In this paper we will discuss the construction of such algorithms for various target classes by applying master algorithms to pools that have been tailored for the given target classes.

We also want to be able to handle cases in which the data seen by the learner is noisy or for some other reason is not quite consistent with any function in the target class, or even any function at all. The number of *anomalies* of a sequence of trials with respect to a given target function is defined to be the number of trials that are not consistent with the function. The number of anomalies of a sequence of trials with respect to a given target class is the least number of anomalies of that sequence with respect to a member of the class. We will be interested in algorithms that are able to learn target classes in the presence of anomalies, that is, in algorithms that are able to make few mistakes on any sequence of trials that has few anomalies with respect to the target class. We will look at the rate of growth of the number of mistakes with the number of anomalies.

There is a straightforward way to construct a canonical algorithm for learning any particular finite target class of computable functions. A pool of algorithms is formed from the functions of the target class. Each function f of the target class is represented by an algorithm of the pool that computes f ; at each trial the prediction made for an instance x by that pool member is just $f(x)$ (such an algorithm pays no attention to the labels). We will refer to pools of this type interchangeably as pools of algorithms and as pools of functions. The Halving Algorithm [Ang88,BF72] (it is given this name in [Lit88]) can be interpreted as a master algorithm that learns a target class using such a pool. For each instance, the Halving Algorithm predicts according to the majority of all *consistent* functions of the pool. (A function is consistent if its values agree with the labels 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 scheme has been previously studied by Barzdin and Freivalds [BF72,BF74], who worked with a variation that applies to infinite pools of functions.

We would like an algorithm that will still function well in the presence of anomalies. If the number of anomalies of a sequence of trials with respect to a pool of functions 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 robust².

We describe the deterministic version of the Weighted Majority Algorithm as it applies to finite pools of arbitrary prediction algorithms making binary predictions. (Later we will give other versions of the algorithm in which these qualifications are varied.)

Weighted Majority Algorithm (WM): *Initially a positive 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*

²Related ideas for working with pools of learning algorithms have been developed within the framework of inductive inference in the limit [FSV89,Pit89,PS88].

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 mistake³, the weights of those algorithms of the pool that disagreed with the label are each multiplied by a fixed β such that $0 \leq \beta < 1$.

If WM is applied to a pool of functions with $\beta = 0$ and the initial weights equal, then it 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,BEHW89] of F is $\Omega(\log |F|)$ 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 .

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*: $f(x_1, \dots, x_n) = 1$ 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 . Suppose we wish to design a prediction algorithm that has a small mistake bound for the target class of all *r-of-k* threshold functions (as r and k vary in the range $1 \leq r \leq k \leq n$). The algorithm Winnow [Lit88] (the Fixed Threshold algorithm of [Lit89b]) can be used, with a bound of $O(kn \log n)$ mistakes. We can obtain this bound without a priori knowledge of r or k . If we know an upper bound r' on the value of r , the mistake bound of Winnow can be improved to $b_{r'} = O(kr' \log n)$, which grows only logarithmically in n when k and r' remain small. The improvement of the mistake bound to $b_{r'}$ is obtained by 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, where it is used to combine a small number of algorithms to produce a combination that is computationally efficient with good mistake bounds. Examples such as this one show that the Weighted Majority Algorithm is a powerful tool for constructing new efficient learning algorithms.

³The 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.

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 resulting algorithm will be efficient whenever the pool consists of a small number of efficient algorithms. 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. Note that a smaller pool size is required for efficiency than for small mistake bounds, since all of the pool algorithms need to be simulated, while the number of mistakes grows only logarithmically with the size of the pool.

In Section 2 we prove mistake bounds for the Weighted Majority Algorithm *WM* which show that in some sense it can be used to select the predictions of the right subpool of algorithms. In Section 3 we discuss a modification of *WM* that never decreases its weights below a certain lower threshold. This variant of *WM*, which we call *WML*, has even stronger selective capabilities. 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 describe the details of this modification in Section 3.

In Section 4 we investigate a second variant of *WM* which deals with countably infinite pools of algorithms, indexed by the positive integers. Barzdin and Freivalds [BF72,BF74], considering pools of (recursive) functions, show that there is an algorithm that makes at most $\log_2 i + \log_2 \log i + o(\log \log i)$ mistakes when given any sequence of trials consistent with the i -th function. We use an adaptation of their method applicable 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*₂ 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.

We give a number of upper bounds obtainable when different initial weights for the algorithms are used. The variant *WMI*₂ incorporates a method of Barzdin and Freivalds that lets one deal explicitly with computational imprecision.

An application of these techniques to deal with an unknown parameter can be found in [HSW90]. The classes *DIFF*(C, B) discussed in that paper have the property that each function in such a class has a certain “depth,” which is a non-negative integer. The basic algorithm given in that paper requires as input an estimate of the depth of the target. Mistake bounds are derived [HSW90, Figure 10] under the assumption that this estimate

is at least as large as the actual depth; to obtain a good bound it must not be too much larger than the actual depth. A version of WMI_2 was used to deal with the case where a good estimate of the depth is unavailable. Algorithm WMI_2 was applied to an infinite pool, where every algorithm in the pool was the basic algorithm, with each pool member using a different estimate of the depth. Algorithm WMI_2 was used since there was no a priori upper bound on the depth, and thus a finite pool would not suffice. This constitutes another example where the weighted majority techniques lead to an efficient algorithm for learning a parameterized function class, provided that an efficient algorithm is known when the parameter is given.

In Section 5 we generalize WM to WMG which is able to handle pools whose members produce predictions chosen from the interval $[0, 1]$. WMG uses the weighted average of the predictions of the pool members to form its own prediction: it predicts 1 if the average is larger than $\frac{1}{2}$, 0 if the average is less than $\frac{1}{2}$ and either 0 or 1 if the average is $\frac{1}{2}$. The predictions of WMG and the labels are still binary. In the same section we also prove bounds for a continuous variant WMC of WM which allows the predictions of the algorithms of the pool and the master as well as the labels to be in $[0, 1]$. WMC simply predicts with the weighted average of the predictions of the pool members. The purpose of Section 5 is a unified treatment of the proofs of all upper bounds for WMG , WMC as well as a randomized version WMR introduced in Section 6. In the concluding section, we give a table that compares all of the varieties of the Weighted Majority Algorithm that we discuss. Simple but specialized proofs for each of the upper bounds for WMG and WMR are provided in the appendix.

In Section 6 we discuss the properties of the randomized version WMR of the Weighted Majority Algorithm. The main result of this section is an expected mistake bound for WMR . The proof relies on the lemmas used to prove bounds for WMG and WMC in the previous section. Like the deterministic algorithms WMG and WMC , the randomized algorithm WMR also uses the weighted average of the predictions of the pool members for making its prediction: it predicts 1 with probability equal to the average. An alternate interpretation of the randomized algorithm involves making each prediction by choosing a member of the pool at random (with probability proportional to its current weight) and predicting 1 with probability equal to the prediction of the chosen pool member. 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. With an appropriate measure of loss the same holds for the deterministic algorithm WMC whose predictions are allowed to be in $[0, 1]$. This represents an improvement by a factor of two over the limiting mistake bound we give for the deterministic algorithm WMG whose predictions must be binary.

We consider in Sections 7 and 8 the special case in which the basic algorithm WM is applied to pools of functions. In Section 7, we 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 i th 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 show that for these classes of functions no other algorithm can do better. In Section 8 we consider the case in which no function in the pool is consistent with all of the trials; we prove a lower bound on the rate at which the mistake bounds must grow

as the number of anomalous trials grows. We compare that lower bound with the upper bounds that we obtain from the Weighted Majority Algorithm. Under certain conditions the Weighted Majority Algorithm is provably a small constant factor from optimal. We make a similar comparison for the randomized algorithm *WMR*.

The concluding section, Section 9 gives an overview of the various algorithms introduced here and mentions a number of directions for future research.

DeSantis, Markowsky and Wegman [DMW88] applied an algorithm similar to *WMC* to a countably infinite pool (as in *WMI*₂) in a completely different setting. For a countably infinite indexed pool of conditional probability distributions the goal is to iteratively construct a “master” conditional probability distribution which assigns a probability to the examples seen so far that is close to the highest probability assigned to the examples by any conditional probability distribution in the pool.

More recent work on learning classes of Boolean functions is given in [HKS91,HO91]. This work presents a Bayesian approach that considers average case upper bounds on the loss of prediction algorithms and gives upper bounds on the loss in terms of the Vapnik-Chervonenkis dimension [VC71,BEHW89]. The Bayes optimal classification algorithm that they consider is a special case of the Weighted Majority algorithm *WM*, and the randomized version *WMR* is the Gibbs algorithm of [HO91]. Furthermore our algorithm *WMR* is similar to a learning procedure studied in [LTS89]. However the analysis given there is very different from ours.

Notations and assumptions: In this paper we design various master algorithms that use the predictions of the pool of algorithms to make their own predictions. Each algorithm in the pool is given an initially positive and always non-negative weight that is updated at the end of each trial. The default values for all initial weights are 1. The total initial weight of all algorithms in the pool is denoted by w_{init} and the total final weight after all examples have been processed by w_{fin} .

For logarithms, we use “ln” to denote natural logarithms and “log₂” to denote logarithms to the base 2. Where the choice of base is not significant, such as in big-O and little-o notation, and in formulas consisting of ratios of logarithms, we omit designation of the base; for ratios we intend that the same base be chosen for numerator and denominator. Throughout the paper we shall implicitly assume that all sequences of instances and labels are finite. Note that if a fixed mistake bound holds for all finite sequences, then it must also hold for all infinite sequences.

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*. For this and the next two sections the predictions of the algorithms in the pool and the master algorithm must all be binary (that is, in $\{0, 1\}$). 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.⁴

⁴Two generalizations *WMG* and *WMC* of this deterministic master algorithm *WM* are given in Section 5. Both master algorithms allow the predictions of the algorithms in the pool to be continuous in $[0, 1]$ instead of only binary. The predictions of *WMG* must be binary and the predictions of *WMC* are allowed to be continuous in $[0, 1]$. Theorem 5.1 gives the same bounds for *WMG* as Theorem 2.1 and a slightly better bound is given for *WMC* in Theorem 5.2.

The parameter β is the factor by which weights are multiplied in case of a mistake and is always in the range $0 \leq \beta < 1$. We suppose that we run WM with a pool \mathcal{A} of prediction algorithms, indexed with the integers from 1 through $|\mathcal{A}|$. We use the notation introduced at the end of Section 1.

All proofs are surprisingly simple. We will 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 $u < 1$. In trials where no mistake occurs the total weight may only decrease. Thus $w_{init}u^m \geq w_{fin}$ must hold, where m is the number of mistakes of WM. This implies that m is at most $\frac{\log \frac{w_{init}}{w_{fin}}}{\log \frac{1}{u}}$. The proof below uses $u = \frac{1+\beta}{2}$.

Theorem 2.1: *Let \mathcal{S} be any sequence of instances and binary labels. Let m be the number of mistakes made by WM on the sequence \mathcal{S} when applied to a pool \mathcal{A} . Then $m \leq \frac{\log \frac{w_{init}}{w_{fin}}}{\log \frac{1+\beta}{2}}$.*

Proof. By the above discussion we only need to show that in trials in which WM makes a mistake the ratio of the total weight after the trial to the total weight before the trial is at most $\frac{1+\beta}{2}$. Before the trial the total weight is $q_0 + q_1$. 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 $\beta q_0 + q_1 \leq \beta q_0 + q_1 + \frac{1-\beta}{2}(q_0 - q_1) = \frac{1+\beta}{2}(q_0 + q_1)$. \square

Note that $w_{fin} \geq \sum_{i=1}^n w_i \beta^{m_i}$, where w_i denotes the initial weight of the i th algorithm in the pool, and m_i denotes the number of mistakes made by that algorithm on the sequence \mathcal{S} . When $\beta = 0$, we use the convention $0^0 = 1$. 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 $\beta > 0$, and also that all initial weights are 1. Otherwise, our assumptions and notation are as in the theorem.

Corollary 2.1: *Assume that \mathcal{A} is a pool of n prediction algorithms and that m_i is the number of mistakes made by the i -th algorithm of the pool on a sequence \mathcal{S} of instances with binary labels. Then WM when applied to pool \mathcal{A} with equal initial weights makes at most $\frac{\log n + m_i \log \frac{1}{\beta}}{\log \frac{1+\beta}{2}}$ mistakes on the sequence \mathcal{S} , for $1 \leq i \leq |\mathcal{A}|$.*

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

Note that if the initial weights were not assumed to be equal then $\log n$ in the above bound would need to be replaced by $\log \frac{w_{init}}{w_i}$, where w_i is the initial weight of the i th algorithm.

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 a sequence \mathcal{S} of instances with binary labels. Then WM when applied to pool \mathcal{A} with equal initial weights makes at most $\frac{\log \frac{n}{k} + m \log \frac{1}{\beta}}{\log \frac{1+\beta}{2}}$ mistakes on the sequence \mathcal{S} .*

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

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 \widehat{m} mistakes in total on a sequence \mathcal{S} of instances with binary labels. Then WM when applied*

to pool \mathcal{A} with equal initial weights makes at most $\frac{\log \frac{n}{k} + \frac{\widehat{m}}{k} \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ mistakes on the sequence \mathcal{S} .

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

Similar bounds to those of the above corollaries can be proven for the case when $\beta = 0$ and the number of mistakes m of the subpool is 0. In that case the upper bound on the number of mistakes made by WM becomes $\log_2 \frac{n}{k}$.

3 Shifting Target

The crux of what we have shown so far is that if $0 < \beta < 1$, then WM selects the right information from a pool of algorithms: $\beta < 1$ allows it to home in on the right information and $\beta > 0$ assures that any change that is made is gradual; any update that leads away from the goal can be reversed. There is a cost for changing the weights only gradually since the algorithm does not home in as fast.

We now modify WM so that its recovery capabilities are made even stronger. Suppose that a particular unknown subpool of a pool of algorithms has good predictive performance (as characterized in the above corollaries). However, after a number of trials a different subpool has good performance and the performance of the original subpool degrades. Then a third subpool takes over and so forth. We want to modify WM so that it keeps track of predictions of the right subpool without too many additional mistakes. In what follows we only consider subpools of size one. (Generalizations in the spirit of the above corollaries are easily obtained.) Thus the scenario is that one algorithm of the pool makes few mistakes for an initial number of trials. After that some other algorithm of the pool performs well for a number of trials and so forth. The modified Weighted Majority algorithm WML has no *a priori* knowledge as to which algorithm's prediction is currently accurate or for how many trials.

Weighted Majority Algorithm WML: *The algorithm has two fixed parameters β and γ with the ranges $0 < \beta < 1$ and $0 \leq \gamma < \frac{1}{2}$, respectively. Each algorithm of the pool \mathcal{A} receives a positive initial weight. The algorithm WML is identical to WM except that whenever the original algorithm updates a current weight (by multiplying it by β), the modified algorithm only makes the update if the weight before the update is larger than $\frac{\gamma}{|\mathcal{A}|}$ times the total weight of all algorithms at the beginning of the trial.*

Note that in the case $\gamma = 0$, WML is identical to WM.

Lemma 3.1: *Let \mathcal{S} be any sequence of instances with binary labels and let m_0 be the minimum number of mistakes made on the sequence \mathcal{S} by any one of a pool \mathcal{A} of n algorithms. If the initial weight of each algorithm is at least $\frac{\beta\gamma}{n}$ times the total initial weight, then WML when applied to pool \mathcal{A} makes at most $\frac{\log \frac{n}{\beta\gamma} + m_0 \log \frac{1}{\beta}}{\log \frac{1}{u}}$ mistakes on \mathcal{S} , where $u = \frac{1+\beta}{2} + (1-\beta)\gamma$. Furthermore, the final weight of each algorithm is at least $\frac{\beta\gamma}{n}$ times the total final weight.*

Proof. Assume without loss of generality that the prediction of *WML* was 0 in a trial in which a mistake was made. Thus weights of pool members predicting 0 are to be decreased. Let q_0 and q_1 be the total weight at the beginning of the trial of the algorithms predicting 0 and 1 respectively. Let q^* be the total weight of the algorithms of the pool that predicted 0 but whose weights are not changed during this trial because they are too small. By assumption $q^* \leq \gamma(q_0 + q_1)$. Then the total weight after the trial will be $\beta(q_0 - q^*) + q^* + q_1 \leq \frac{1+\beta}{2}(q_0 + q_1) + (1 - \beta)q^*$. The ratio of this total to the total before the trial is bounded by $\frac{1+\beta}{2} + (1 - \beta)\gamma$. Since we have assumed that $\gamma < \frac{1}{2}$, this bound is less than 1. Thus a bound for *WML* follows from the argument at the beginning of the previous section using the fact that the final sum of weights is at least $\frac{\beta\gamma}{n}w_{init}\beta^{m_0}$.

It is easy to see that since the total weight is never increased, and individual weights are not decreased if they are too small, the final weight of each algorithm will have the specified relation with the total final weight. \square

We can apply the above lemma to subsequences of a sequence of trials and get the following.

Theorem 3.1: *Let \mathcal{S} be a sequence of instances with binary labels and let $\mathcal{S}_1, \dots, \mathcal{S}_k$ be any partitioning of \mathcal{S} into k subsequences. Let l_i be the number of trials in the i -th subsequence. Let \mathcal{A} be any pool of n algorithms and let m_i be the minimum number of mistakes made by any algorithm of the pool \mathcal{A} on the subsequence \mathcal{S}_i , for $1 \leq i \leq k$. If all initial weights of the algorithms are at least $\frac{\beta\gamma}{n}$ times the total initial weight, then *WML* when applied to pool \mathcal{A} makes at most $\sum_{i=1}^k \min(\{l_i, \frac{\log \frac{n}{\beta\gamma} + m_i \log \frac{1}{\beta}}{\log \frac{1}{u}}\})$ mistakes on \mathcal{S} , where $u = \frac{1+\beta}{2} + (1 - \beta)\gamma$.*

Proof. We simply apply the previous lemma to each subsequence \mathcal{S}_i . \square

4 Selection From an Infinite Pool

In this section we assume that there is a countably infinite pool of algorithms indexed with some computable indexing (that is, we assume that there exists a simulation algorithm for the pool that can simulate any algorithm of the pool, given its index). We assume that the i -th algorithm makes at most m_i mistakes for a sequence of instances and labels \mathcal{S} . We will develop a version of the Weighted Majority Algorithm that makes at most $\inf_{i \geq 1} (c(\log_2 i + m_i))$ mistakes on \mathcal{S} , where c is a constant that depends on the parameters of the algorithm.

Assume for the moment that the original algorithm *WM* could keep track of infinitely many weights and suppose, for example, that the initial weight associated with the i th pool member is $w_i = \frac{1}{i(i+1)}$, for $i \geq 1$. Then the total initial weight is $w_{init} = \sum_{i=1}^{\infty} w_i = 1$. For each i we have that the final sum of the weights is at least $w_i \beta^{m_i} = \frac{\beta^{m_i}}{i(i+1)}$. Applying Theorem 2.1 we get that *WM* makes at most $\inf_{i \geq 1} (\frac{\log i(i+1) + m_i \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}})$ mistakes when applied to the countably infinite pool on the sequence \mathcal{S} .

The above approach is flawed because we obviously can't run *WM* on infinitely many weights. Instead, we construct an algorithm *WMI*₂ that uses an increasing initial segment

of “active” weights⁵. This algorithm and its analysis is based on techniques introduced by Barzdin and Freivalds [BF74] (they consider only the $\beta = 0$ case). These techniques, in addition to dealing with an infinite pool size, also allow one to work with finite-precision approximations to the weights in calculating the predictions of the algorithm. Though we do not consider this issue elsewhere in the paper, the analysis of this algorithm can be taken as an example of how to take approximate computations into account.⁶

Weighted Majority Algorithm WMI_2 : *The algorithm is run on a countably infinite pool of algorithms, indexed with the positive integers. It has one fixed parameter β in the range $0 \leq \beta < 1$. It is also given two computable functions, W and \widehat{W} defined on the positive integers (W needs only to be computable to arbitrary precision); the first is used to determine the initial weights, and the second is some function satisfying $\widehat{W}(i) \geq \sum_{j=i}^{\infty} W(j)$ and $\lim_{i \rightarrow \infty} \widehat{W}(i) = 0$. (Since there is flexibility in the choice of \widehat{W} we can assume for convenience that it is chosen so that it is exactly computable.) This algorithm is similar to WM. Instead of working with the weights of all of the algorithms, WMI_2 works with only the weights of an active subpool in each trial. The prediction of the algorithm is based solely on the predictions of the active subpool and only weights corresponding to algorithms of the active subpool are subject to change. The active subpool consists of pool members with indices 1 through l , for some l that is determined by the algorithm. The value of l is initialized to zero. Then initially and at the conclusion of any trial in which a mistake is made it is increased, if necessary, until the inequality $\widehat{W}(l+1) \leq \frac{u^{m+1}}{(1-\beta)(m+1)(m+2)} \widehat{W}(1)$ is satisfied, where $u = \frac{1+\beta}{2}$ and m is the number of mistakes that have been made. When l is increased the weights of the newly active algorithms are initialized. The initial weight of the i -th algorithm of the pool is set to $W(i)$. To make a prediction, WMI_2 computes the sum q_0 of the weights of the active pool members predicting 0 for the current instance, and the sum q_1 of the weights of the active pool members predicting 1. Suppose that a total of m mistakes have been made in previous trials. If $q_0 > q_1 + \frac{u^{m+1} \widehat{W}(1)}{(1-\beta)(m+1)(m+2)}$ then WMI_2 predicts 0; if $q_1 > q_0 + \frac{u^{m+1} \widehat{W}(1)}{(1-\beta)(m+1)(m+2)}$ then WMI_2 predicts 1. If neither of these inequalities holds, WMI_2 is allowed to predict 0 or 1. (This allows WMI_2 to use finite-precision approximations to q_0 and q_1 .) After the label is received in a trial in which WMI_2 has made a mistake, the weight of each active pool member that disagreed with the label is multiplied by β . The final action of each trial in which a mistake is made is to increase l as necessary, as described earlier.*

When the sums of tails of the series $\sum_{i=1}^{\infty} W(i)$ are easy to compute, it is natural to take $\widehat{W}(i) = \sum_{j=i}^{\infty} W(j)$. When this is done, $\widehat{W}(1)$ is the sum of the initial values of the full infinite sequence of weights. Note that instead of storing the current weights, an implementation of WMI_2 can store the number of mistakes made by each pool member (while active) and use this information to calculate q_0 and q_1 to the necessary precision.

Theorem 4.1: *Let \mathcal{S} be any sequence of instances with binary labels on which the i th algorithm A_i makes at most m_i mistakes (for all $i \geq 1$). Let W and \widehat{W} be computable functions satisfying the inequality given in the description of WMI_2 . After any initial sequence of trials, let ω_i be the current value of the i -th weight of WMI_2 , for $i \leq l$, and let*

⁵We use the name WMI_2 to distinguish this algorithm from the similar algorithm called WMI in earlier versions of this paper [LW89a,LW89b]. The version presented here improves the best mistake bound that can be obtained from the algorithm to match the essentially optimal bound of Barzdin and Freivalds [BF72,BF74] in the case where $\beta = 0$ and there exists a consistent pool member.

⁶The earlier version, WMI [LW89a,LW89b], does not take finite precision into account.

$\omega_i = W(i)$ for $i > l$. Let $u = \frac{1+\beta}{2}$. Then the following holds for WMI_2 when applied to the countably infinite pool \mathcal{A} on the sequence \mathcal{S} :

1. At the beginning of any trial the size of the active pool is the minimum l such that $\widehat{W}(l+1) \leq \frac{u^{m+1}}{(1-\beta)(m+1)(m+2)} \widehat{W}(1)$, where m is the number of mistakes that have been made by WMI_2 in prior trials.
2. After any initial sequence of trials in which m mistakes have been made by WMI_2

$$\sum_{i=1}^{\infty} \omega_i \leq \left(2 - \frac{1}{m+1}\right) u^m \widehat{W}(1).$$

3. If $0 < \beta < 1$, then the total number of mistakes made by WMI_2 is at most $\inf_{i \geq 1} \left(\frac{\log(\widehat{W}(1)/W(i)) + m_i \log \frac{1}{\beta} + \log 2}{\log \frac{1}{u}} \right)$. If $\beta = 0$ and for some i , $m_i = 0$, then the total number of mistakes made by WMI_2 is bounded by $1 + \log_2(\widehat{W}(1)/W(i))$.

Proof of 1. This follows immediately from the construction of WMI_2 .

Proof of 2. We prove this by induction on m . It clearly holds for $m = 0$. Suppose that the claim holds for some $m \geq 0$ and consider the trial in which the $(m+1)$ -st mistake is made. Let q_0 be the sum of the weights, at the beginning of that trial, of the active pool members predicting 0, let q_1 be the sum of the weights of the active members predicting 1, and let q^* be the sum of the (future initial) weights of the inactive members. Thus $q^* = \sum_{i=l+1}^{\infty} W(i) \leq \widehat{W}(l+1) \leq \frac{u^{m+1}}{(1-\beta)(m+1)(m+2)} \widehat{W}(1)$. Suppose that WMI_2 predicted 0. (An analogous argument applies in the case that WMI_2 predicted 1.) Thus we must have $q_1 \leq q_0 + \frac{u^{m+1} \widehat{W}(1)}{(1-\beta)(m+1)(m+2)}$. The $(m+1)$ -st mistake causes the total weight $\sum_{i=1}^{\infty} \omega_i$ to be updated to

$$\begin{aligned} \beta q_0 + q_1 + q^* &\leq \beta q_0 + q_1 + \frac{1-\beta}{2} \left(q_0 - q_1 + \frac{u^{m+1} \widehat{W}(1)}{(1-\beta)(m+1)(m+2)} \right) + q^* \\ &= \frac{1+\beta}{2} (q_0 + q_1 + q^*) + \left(\frac{1-\beta}{2} \right) \left(q^* + \frac{u^{m+1} \widehat{W}(1)}{(1-\beta)(m+1)(m+2)} \right) \\ &\leq \frac{1+\beta}{2} \left(2 - \frac{1}{m+1} \right) u^m \widehat{W}(1) + \frac{1-\beta}{2} \left(\frac{2u}{(1-\beta)(m+1)(m+2)} \right) u^m \widehat{W}(1) \\ &= \left(2 - \frac{1}{m+2} \right) u^{m+1} \widehat{W}(1) \end{aligned}$$

as desired.

Proof of 3. First we consider the case $0 < \beta < 1$. Since the i -th algorithm in the pool makes at most m_i mistakes, the final sum of the weights $\sum_{i=1}^{\infty} \omega_i$ is at least $\beta^{m_i} W(i)$. By 2, if WMI_2 makes m mistakes then the final sum of the weights is at most $2u^m \widehat{W}(1)$. Solving the resulting inequality for m gives the desired upper bound on m . When $m_i = 0$, then the final sum of the weights is at least $W(i)$. When $\beta = 0$, then $u = \frac{1}{2}$. Thus if WMI_2 makes m mistakes in this case then $W(i) \leq 2 \left(\frac{1}{2} \right)^m \widehat{W}(1)$, which yields the desired bound. \square

As we vary our choices of initial weights we encounter a trade-off between the size of the mistake bound that we obtain and the size to which the active pool may grow.⁷ We

⁷If the same weight sequence is used, the earlier algorithm WMI [LW89a,LW89b] uses a slightly smaller active pool, at the expense of a slightly larger mistake bound. See [LW89a,LW89b] for the corresponding versions of Corollaries 4.1, 4.2, and 4.3.

examine this trade-off in the following corollaries, which follow immediately. For brevity, we omit statement of the results for $\beta = 0$. Some results regarding the $\beta = 0$ case are given at the end of the next section.

Corollary 4.1: *Let S be any sequence of instances with binary labels on which the i th algorithm makes at most m_i mistakes ($i \geq 1$). Let $W(i) = \frac{1}{i(i+1)}$ and $\widehat{W}(i) = \sum_{j=i}^{\infty} W(j) = \frac{1}{i}$. Then the following holds for WMI_2 when applied to the countably infinite pool \mathcal{A} on the sequence S :*

1. After m mistakes have been made by WMI_2 the size of the active pool is $\left\lceil (1 - \beta)(m + 1)(m + 2)\left(\frac{2}{1 + \beta}\right)^{m+1} \right\rceil - 1$.
2. If $\beta > 0$ the total number m of mistakes made by WMI_2 is at most $\inf_{i \geq 1} \left(\frac{\log_2(i(i+1)) + m_i \log_2 \frac{1}{\beta} + 1}{\log_2 \frac{2}{1 + \beta}} \right)$.

□

In the introduction of this section we derived a bound for running WM on infinitely many weights. The bound for this infeasible algorithm is identical to the one given in the above corollary except for the absence of 1 in the numerator. Note that in the above corollary the pool size grows exponentially in m , which is of course an improvement over an infinite pool size. In Section 7, we will study the best that can be done when $\beta = 0$, without regard to computational complexity; we will give a weight sequence that gives a somewhat smaller mistake bound than the bound of this corollary. There we will be uninterested in the pool size. For the rest of the current section we will take the point of view of one interested in practical application of this algorithm, for whom questions of computational efficiency, and therefore of the size of the active pool, are important. We thus do not want the pool size to grow exponentially in the number of mistakes that are made (particularly in cases where the number of mistakes grows linearly with the number of trials, which can be expected, for example, when the learner faces noisy data; see Section 8).

By choosing a different weight sequence one can assure that the pool size grows only linearly in m , but this increases the rate of growth of the mistake bound with the index i : in the above corollary the dependence on the index i is logarithmic; below it is linear.

Corollary 4.2: *Let S be any sequence of instances with binary labels on which the i th algorithm makes at most m_i mistakes ($i \geq 1$). Let $W(i) = \left(\frac{1}{2}\right)^i$ and $\widehat{W}(i) = \sum_{j=i}^{\infty} W(j) = \left(\frac{1}{2}\right)^{i-1}$. Then the following holds for WMI_2 when applied to the countably infinite pool \mathcal{A} on the sequence S :*

1. After m mistakes have been made by WMI_2 the size of the active pool is $\left\lceil \log_2(1 - \beta) + \log_2((m + 1)(m + 2)) + (m + 1) \log_2 \frac{1}{u} \right\rceil$.
2. The total number m of mistakes made by WMI_2 is at most $\inf_{i \geq 1} \left(\frac{i + m_i \log_2 \frac{1}{\beta} + 1}{\log_2 \frac{2}{1 + \beta}} \right)$.

□

Observe that the weights chosen in both corollaries might be useful for particular applications. For example, if $m_i = \Omega(i)$, then choosing the weights as a sequence decreasing exponentially with i (as done in the second corollary) only increases the mistake bound by a

constant factor over the bound using the choice of weights given in the first corollary. Thus at the cost of increasing the mistake bound by a constant factor a significantly smaller pool size is obtained using the exponentially decreasing weight sequence.

In general, when given an infinite sequence of algorithms A_i with mistake bounds m_i , it is reasonable to choose the initial weights such that

1. $\sum_{j=1}^{\infty} W(j)$ is finite and
 2. the two summands $\log(\widehat{W}(1)/W(i))$ and $m_i \log \frac{1}{\beta}$ of the numerator of the bound on the total number of mistakes given in Part 3 of the above theorem are roughly equal.
- The following corollary gives a weight sequence that is useful when m_i grows exponentially with i . A very similar version has been applied in [HSW90].

Corollary 4.3: *Let S be any sequence of instances with binary labels on which the i th algorithm makes at most m_i mistakes ($i \geq 1$). Let $W(i) = \left(\frac{1}{2}\right)^{2^{i-1}}$ and $\widehat{W}(i) = 2 \left(\frac{1}{2}\right)^{2^{i-1}}$. Then the following holds for WMI_2 when applied to the countably infinite pool \mathcal{A} on the sequence S :*

1. *After m mistakes have been made by WMI_2 the size of the active pool is $\left\lceil \log_2[\log_2(1 - \beta) + \log_2((m + 1)(m + 2)) + (m + 1) \log_2 \frac{2}{1 + \beta} + 1] \right\rceil$.*
2. *The total number m of mistakes made by WMI_2 is at most $\inf_{i \geq 1} \left(\frac{2^{i-1} + m_i \log_2 \frac{1}{\beta} + 1}{\log_2 \frac{2}{1 + \beta}} \right)$.*

Proof. Given the inequality $\widehat{W}(i) \geq \sum_{j=i}^{\infty} W(j)$ then parts 1 and 2 of the corollary are a straightforward application of parts 1 and 3, respectively, of the previous theorem. To show the above inequality we bound the infinite sum from above by the geometric series with ratio $\frac{1}{2}$ and first summand $\left(\frac{1}{2}\right)^{2^{i-1}}$:

$$\sum_{j=i}^{\infty} \left(\frac{1}{2}\right)^{2^{j-1}} \leq \sum_{j=2^{i-1}}^{\infty} \left(\frac{1}{2}\right)^j = 2 \left(\frac{1}{2}\right)^{2^{i-1}}.$$

□

We give an example making use of the results of this corollary. The example is a generalization of the r -of- k threshold function example given in the introduction. We assume that we are to learn a class of target functions using an algorithm A that takes a single parameter q . Associated with each target function are two parameters $k \geq 1$ and r (these can be thought of, if one likes, as measures of the complexity of the target function). We assume that algorithm A has the property that whenever its parameter q is at least k then the number of mistakes made by A is bounded by $B(q, r) = q \cdot g(r)$, for any target function with parameters k and r . If the parameter $q < k$, then $B(q, r)$ is not assumed to be a bound on the number of mistakes (in fact, the number of mistakes the algorithm makes might be unbounded). Assume that $g(r) \geq r_0$ for all choices of r .

A bound of this form will be minimized if we choose the parameter $q = k$. (We are not claiming that the algorithm will necessarily make the fewest mistakes for a given target function when $q = k$, but that the best bound that we can obtain from the information given will be obtained when $q = k$. If our bounds accurately reflect the behavior of the algorithm then this will indeed be a good choice of q .) Our goal here is to construct an algorithm that makes a number of mistakes close to $B(k, r)$, without knowing k , for any k and r and any target function with parameters k and r chosen from the target class.

Under the assumptions that we have made, the mistake bound for $A(q)$ is at most $2B(k, r)$ for any q in the range $k \leq q \leq 2k$. Because of this, it suffices for our purposes to apply WMI_2 to the infinite pool of algorithms $A(1), A(2), A(4), A(8), \dots$. Though the optimal choice of q may not be included in this pool, an algorithm is included whose bound is at most twice the bound for the optimal choice.

The mistake bound given to us for the i -th algorithm of this pool is $m_i = B(2^{i-1}, r) = 2^{i-1}g(r)$ for any i such that $2^{i-1} \geq k$. This bound grows exponentially in i . Thus the weights given in Corollary 4.3 will serve us well.

By Corollary 4.3 we get a mistake bound of

$$\frac{2^{i-1} \left(1 + g(r) \log_2 \frac{1}{\beta} \right) + 1}{\log_2 \frac{2}{1+\beta}}$$

where i is the least positive integer such that $2^{i-1} \geq k$. By assumption $g(r) \geq r_0$. Thus the above expression is bounded by

$$\frac{B(2^{i-1}, r) \left(\frac{1}{r_0} + \log_2 \frac{1}{\beta} \right) + 1}{\log_2 \frac{2}{1+\beta}}$$

Since $B(q, r) = q \cdot g(r)$, and there exists some i such that $k \leq 2^{i-1} < 2k$, we obtain a bound

$$\frac{2B(k, r) \left(\frac{1}{r_0} + \log_2 \frac{1}{\beta} \right) + 1}{\log_2 \frac{2}{1+\beta}}$$

which is $O(B(k, r))$. The active pool size of this algorithm grows logarithmically in the number of mistakes that have been made.

5 Generalized Analysis

In this section we introduce two new master algorithms WMG and WMC . The original algorithm WM is a special case of the generalized version WMG . The generalized analysis given here for the new variants will also be used in the next section for deriving bounds for the randomized version WMR of WM . The purpose of this section is to give a unified analysis for all three versions WMG , WMC and WMR . Separate direct and simple proofs for the upper bounds of WMG and WMR are provided in the appendix.

Assumptions for WMG and WMC : Both new algorithms allow the predictions of the algorithms of the pool to be chosen from $[0, 1]$ (instead of being binary as for WM). The predictions of WMG must be binary while the predictions of WMC are allowed to be chosen from the interval $[0, 1]$. The labels associated with the instances are assumed to be binary for WMG and in $[0, 1]$ for WMC .

The *update step* of the Weighted Majority Algorithm and its variants is the step in which each weight is multiplied by some factor. In algorithm WM this step only occurs during trials in which a mistake is made. However, it is easy to see that the same mistake bound we have given will be obtained if updates are performed in every trial.

Update criteria for WMG and WMC: *WMC* updates at every trial (the same criterion is used for *WMR* introduced in the next section). For *WMG* an update step is executed either in every trial or only in trials in which a mistake occurs.⁸

We introduce notation to enable us to refer to values that occur in each trial in which an update step occurs. We use the term *update-trial* j to refer to the j th trial in which an update step occurs. We assume that there are a total of t such trials. (Thus t either denotes the total number of trials or the total number of mistakes, depending on the update criterion.) We assume that the master algorithm is applied to a pool of n algorithms, letting $x_i^{(j)}$ denote the prediction of the i th algorithm of the pool in update-trial j . Let $\lambda^{(j)}$ denote the prediction of the master algorithm in update-trial j , $\rho^{(j)}$ denote the label of update-trial j and $w_1^{(j)}, \dots, w_n^{(j)}$ denote the weights at the beginning of update-trial j . (Consequently, $w_1^{(t+1)}, \dots, w_n^{(t+1)}$ denote the weights following the final trial.) We assume that all initial weights $w_i^{(1)}$ are positive.

Let $s^{(j)} = \sum_{i=1}^n w_i^{(j)}$ and

$$\gamma^{(j)} = \frac{\sum_{i=1}^n w_i^{(j)} x_i^{(j)}}{s^{(j)}}.$$

Thus $s^{(1)} = w_{init}$ and $s^{(t+1)} = w_{fin}$.

Prediction of WMC and WMG: In the case of *WMC* the prediction $\lambda^{(j)}$ equals $\gamma^{(j)}$. For *WMG*, $\lambda^{(j)}$ is 1 when $\gamma^{(j)}$ is greater than $\frac{1}{2}$ and is 0 when $\gamma^{(j)}$ is less than $\frac{1}{2}$ (either prediction is allowed if $\gamma^{(j)} = \frac{1}{2}$).

If predictions are continuous then the notion of mistake has to be replaced by a quantity that measures how far the prediction is from the correct label. In this paper we will use the absolute loss. If an algorithm predicts x in a trial with label ρ , we say that its loss in that trial is $|x - \rho|$; this definition applies both to algorithms in the pool and to the master algorithm. We denote the total loss of the master algorithm over all trials by m , and the total loss over all trials of algorithm i of the pool by m_i . Since an update occurs in at least those trials in which the master algorithm makes a mistake, the total loss of the master algorithm is $m = \sum_{j=1}^t |\lambda^{(j)} - \rho^{(j)}|$. Algorithms in the pool may incur loss in trials without updates, so, unless updates occur in all trials, we cannot get the total loss of algorithms in the pool by summing over update trials. Instead, we have the inequality $m_i \geq \sum_{j=1}^t |x_i^{(j)} - \rho^{(j)}|$. Note that in the case where both the predictions of the pool members and the labels are binary, the losses of the pool members become numbers of mistakes, and similarly if both the predictions of the master algorithm and the labels are restricted to being binary then the loss of the master becomes a measure of its mistakes.

More than one form of the update is possible. We will specify a class of possible updates; our bounds apply to all members of the class. All updates in the class coincide with the update of *WM* given in the introduction in the special case that the pool members produce boolean predictions.

Update step for WMG and WMC: In an update step of *WMG* and *WMC* (and the randomized version *WMR* of the next section) each weight $w_i^{(j)}$ is multiplied by some

⁸More precisely, bounds for *WM* and *WMG* hold as long as an update step occurs in every trial in which a mistake occurred and possibly in some trials in which no mistake occurred; for simplicity, we restrict discussion to the extreme cases mentioned.

factor F that depends⁹ on β , $x_i^{(j)}$, and $\rho^{(j)}$:

$$w_i^{(j+1)} = F w_i^{(j)}, \text{ where } F \text{ can be any factor that satisfies} \\ \beta^{|x_i^{(j)} - \rho^{(j)}|} \leq F \leq 1 - (1 - \beta)|x_i^{(j)} - \rho^{(j)}|. \quad (5.1)$$

The following lemma implies that such a factor exists; in particular either the upper or the lower bound given for F can be chosen as the update factor. Recall that we defined 0^0 to equal 1.

Lemma 5.1: *For $\beta \geq 0$ and $0 \leq r \leq 1$, $\beta^r \leq 1 + r(\beta - 1)$.*

Proof It is easy to check the inequality for the case of $\beta = 0$. If $\beta > 0$ then the inequality follows from the convexity of β^r as a function of r for any $\beta > 0$. Convexity implies that for $0 \leq r \leq 1$, $\beta^r \leq (1 - r)\beta^0 + r\beta^1$, which is another way of writing the inequality. \square

The next lemma is the basic lemma used to derive bounds for the loss of the master algorithms *WMG* and *WMC* (and for the expected loss of *WMR* of the next section).

Lemma 5.2: *Assume that $w_i^{(1)} > 0$ for $i = 1, \dots, n$. Assume $0 \leq \beta < 1$, $0 \leq \rho^{(j)} \leq 1$, and $0 \leq x_i^{(j)} \leq 1$ for $j = 1, \dots, t$ and $i = 1, \dots, n$. Assume $w_i^{(j+1)} \leq w_i^{(j)}(1 - (1 - \beta)|x_i^{(j)} - \rho^{(j)}|)$ for $j = 1, \dots, t$ and $i = 1, \dots, n$. Then if $\beta = 0$ and $|\gamma^{(j)} - \rho^{(j)}| = 1$ for some j in $\{1, \dots, t\}$ then $w_{fin} = 0$. Otherwise*

$$\ln \frac{w_{fin}}{w_{init}} \leq \sum_{j=1}^t \ln(1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|)$$

Proof First we deal with the case where $\beta = 0$ and there is a trial j such that $|\gamma^{(j)} - \rho^{(j)}| = 1$. In this case, we have $\gamma^{(j)} = 1 - \rho^{(j)}$. For this to occur, for any i such that $w_i^{(j)} > 0$ we must have $|x_i^{(j)} - \rho^{(j)}| = 1$. This forces the use of update factors that make $w_i^{(j+1)} = 0$ for all i . Thus $w_{fin} = 0$, as desired. Where this case does not occur, we have from Inequality 5.1 that

$$s^{(j+1)} \leq \sum_{i=1}^n w_i^{(j)}(1 - (1 - \beta)|x_i^{(j)} - \rho^{(j)}|) = s^{(j)} - (1 - \beta) \sum_{i=1}^n w_i^{(j)}|x_i^{(j)} - \rho^{(j)}|.$$

By the triangle inequality, the above is bounded above by

$$s^{(j)} - (1 - \beta) \left| \sum_{i=1}^n w_i^{(j)}(x_i^{(j)} - \rho^{(j)}) \right| = s^{(j)} - (1 - \beta)|\gamma^{(j)} s^{(j)} - \rho^{(j)} s^{(j)}| \\ = s^{(j)}(1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|).$$

Thus

$$s^{(t+1)} \leq s^{(1)} \prod_{j=1}^t (1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|)$$

⁹Theorems 5.1 and 5.2 can be obtained without requiring that F satisfy the lower bound specified in Inequality 5.1 (but since the bounds of these theorems grow as w_{fin} shrinks, the bounds may become uninteresting if F is too small). Both bounds on F of Inequality 5.1 are used to obtain Corollaries 5.1 and 5.2.

Taking logarithms gives the desired result. \square

We use the above lemma to obtain the same bound for WMG as we did for WM in Theorem 2.1. Recall that for WMG, $\rho^{(j)}, \lambda^{(j)} \in \{0, 1\}$ but the $x_i^{(j)}$ are allowed to be chosen from $[0, 1]$. For the case where the $x_i^{(j)}$ are discrete as well, the algorithms WM and WMG are identical.

Theorem 5.1: *Let \mathcal{S} be any sequence of instances with binary labels. Suppose the algorithm WMG (updating in every trial or only when mistakes are made) is run with $0 \leq \beta < 1$ on the sequence \mathcal{S} and m is the total number of mistakes of WMG when applied to some pool of prediction algorithms. Then $m \leq \frac{\log \frac{w_{init}}{w_{fin}}}{\log \frac{2}{1+\beta}}$.*

Proof If $\beta = 0$ and $|\gamma^{(j)} - \rho^{(j)}| = 1$ for some j in $\{1, \dots, t\}$, then $w_{fin} = 0$ and the bound becomes vacuous. Otherwise, let $m^{(j)}$ be 1 if WMG makes a mistake in update-trial j , and 0 otherwise. The total number of mistakes made by WMG is $m = \sum_{j=1}^t m^{(j)}$. (For the version of WMG that only updates when a mistake is made, we are only paying attention to trials in which a mistake is made; thus $m^{(j)}$ is 1 for all j .) Note that if $\gamma^{(j)} < \frac{1}{2}$ then $m^{(j)} = \rho^{(j)}$, and if $\gamma^{(j)} > \frac{1}{2}$ then $m^{(j)} = 1 - \rho^{(j)}$.

Since $\log(1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|) \leq 0$, we have

$$\begin{aligned} \sum_{j=1}^t \log(1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|) &\leq \sum_{j \text{ s.t. } m^{(j)}=1} \log(1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|) \\ &\leq m \log(1 - \frac{1}{2}(1 - \beta)) = m \log(\frac{1}{2} + \frac{1}{2}\beta) \end{aligned}$$

It is easy to see that the conditions for the application of Lemma 5.2 apply and thus

$$\log \frac{w_{fin}}{w_{init}} \leq m \log \frac{1 + \beta}{2}.$$

\square

A simple proof of the above bound that does not rely on Lemma 5.2 is given in the appendix. The method there is to redefine q_0 and q_1 and essentially use the proof of Theorem 2.1.

From the definition of the total loss m_i of the i th algorithm and the definition of the update step (Inequality 5.1) we have $w_i^{(t+1)} \geq w_i^{(1)} \beta^{m_i}$ and $w_{fin} \geq \sum_{i=1}^n w_i^{(1)} \beta^{m_i}$. These inequalities become equalities if an update occurs in each trial and all update factors are made as small as allowed. Using these inequalities, corollaries in the spirit of corollaries 2.1 to 2.3 could easily be derived. We will only state the one corresponding to Corollary 2.1. For this corollary assume all initial weights are equal and $\beta > 0$.

Corollary 5.1: *Assume that \mathcal{A} is a pool of n prediction algorithms and that m_i is the total loss of the i -th algorithm of the pool on a sequence \mathcal{S} of instances with binary labels. Then WMG when applied to pool \mathcal{A} with equal initial weights makes at most $\frac{\log n + m_i \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ mistakes on the sequence \mathcal{S} , for $1 \leq i \leq |\mathcal{A}|$.*

In general, the larger the lower bound used for w_{fin} the better the upper bound provided by the above theorem. If in each update the smallest allowable factor is used (Inequality 5.1) then $w_{fin} \geq \sum_{i=1}^n w_i^{(1)} \beta^{m_i}$. However if the largest factor is used in each update we get $w_{fin} \geq \sum_{i=1}^n w_i^{(1)} \prod_{j=1}^t (1 - (1 - \beta)|x_i^{(j)} - \rho^{(j)}|)$ which may be much larger than the previous lower bound on w_{fin} when β is close to 0. The upper bounds we will derive for WMC and WMR will have the same form as the one given in the above theorem for WMG. Thus the same comments on the lower bound of w_{fin} will apply to those bounds as well.

The next lemma will be used for deriving bounds for WMC (and for WMR in the next section).

Lemma 5.3: *If the conditions of Lemma 5.2 are satisfied, then*

$$\sum_{j=1}^t |\gamma^{(j)} - \rho^{(j)}| \leq \frac{\ln \frac{w_{init}}{w_{fin}}}{1 - \beta}.$$

Proof The lemma follows from the observation that $\ln(1 - (1 - \beta)|\gamma^{(j)} - \rho^{(j)}|) \leq -(1 - \beta)|\gamma^{(j)} - \rho^{(j)}|$ and Lemma 5.2. \square

Recall that for WMC all predictions and labels are allowed to be in $[0, 1]$ and that the prediction of WMC in trial j is $\lambda^{(j)} = \gamma^{(j)}$. Lemma 5.3 gives an upper bound on the total loss m of WMC. In the next section we will see that this bound is identical to the bound on the expected total loss we obtain for WMR in Theorem 6.1. Recall that both WMC and WMR update in every trial. For WMC we assume that all instances and labels are deterministic.

Theorem 5.2: *Let \mathcal{S} be any sequence of instances and labels with labels in $[0, 1]$. Let m be the total loss of WMC on the sequence \mathcal{S} when applied to some pool of prediction algorithms.*

Then $m \leq \frac{\ln \frac{w_{init}}{w_{fin}}}{1 - \beta}$.

Proof The theorem follows immediately from Lemma 5.3. \square

It is interesting to compare the above bound for WMC (continuous prediction) to the bound obtained in Theorem 5.1 for WMG (discrete prediction). The latter bound is similar in form: $\frac{\ln \frac{w_{init}}{w_{fin}}}{\ln \frac{2}{1 + \beta}}$. The numerators of both bounds are identical. The denominators both approach zero when β approaches 1. However, the denominator of the bound for WMC is larger for β less than 1. The ratio of the denominators approaches 2 as β approaches 1, making the bound for WMC better than the bound for WMG by nearly a factor of two for β close to 1.

Corollary 5.2: *Assume that \mathcal{A} is a pool of n prediction algorithms and that m_i is the total loss of the i -th algorithm of the pool on a sequence \mathcal{S} of instances with labels in $[0, 1]$. If we apply WMC to pool \mathcal{A} with equal initial weights, then it will have a total loss of at most $\frac{\ln n + m_i \ln \frac{1}{\beta}}{1 - \beta}$ on the sequence \mathcal{S} , for $1 \leq i \leq |\mathcal{A}|$.*

6 Randomized Predictions

In this section we give a randomized version of *WM* called *WMR*.

Assumptions for *WMR*: The predictions of the pool members are in $[0, 1]$. The prediction of *WMR* is binary but probabilistic. The labels associated with the instances are binary.

We will use the notation introduced in the previous section. Recall that at trial j *WMC* predicts $\gamma^{(j)} = \frac{\sum_{i=1}^n w_i^{(j)} x_i^{(j)}}{\sum_{i=1}^n w_i^{(j)}}$ which is the weighted average prediction of the pool members.

Also *WMC* predicts 1 when $\gamma^{(j)} > \frac{1}{2}$ and 0 when $\gamma^{(j)} < \frac{1}{2}$ (and either when $\gamma^{(j)} = \frac{1}{2}$).

Prediction of *WMR*: The new randomized algorithm *WMR* simply predicts 1 with probability $\gamma^{(j)}$.

Note that if the predictions $x_i^{(j)}$ of the members of the the pool are binary then $\gamma^{(j)} = \frac{q_1}{q_0 + q_1}$ where q_0 is the total weight of all algorithms predicting 0 at trial j and q_1 is defined similarly.

Update criterion of *WMR*: Like *WMC*, *WMR* updates in every trial.

Update step of *WMR*: The update step of *WMR* is the same as the update step of *WMC* and *WMC* described in the previous section (Inequality 5.1).

Bounds on the performance of *WMR* depend on adequate independence between the randomization performed by *WMR* and the choice of the instances and labels. We will obtain bounds on the expected number of mistakes made by *WMR* under the following assumption, which we will refer to as the *weak independence condition*:

$$\mathbf{E}(\lambda^{(j)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(j)}, \rho^{(j)})) = \gamma^{(j)} \quad (\text{a.s.})$$

for $j = 1, \dots, t$. (Here we use conditional expectations with respect to random variables, as described, for example, in [Bil86, Shi84]. The notation (a.s.) stands for “almost surely,” that is, with probability 1.) The variables appearing in the condition are defined in Section 5. Note that this definition allows each of the $x_i^{(j)}$ and $\rho^{(j)}$ to be random variables. If they are deterministically chosen, then all of the weights and $\gamma^{(j)}$ are deterministic and the construction of the algorithm guarantees that the weak independence condition holds. (In that case, for the weak independence condition to hold, it suffices that $\mathbf{E}(\lambda^{(j)}) = \gamma^{(j)}$ for all j , which follows from the construction of the algorithm.)

In addition to bounds on the expected number of mistakes, we will also give a bound on the probability that the total number of mistakes exceeds the expected number by some margin. For this bound, we make the following assumption that we will refer to as the *strong independence condition*:

$$\mathbf{E}(\lambda^{(j)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) = \gamma^{(j)} \quad (\text{a.s.})$$

for $j = 1, \dots, t$. Note that in addition to conditioning on the past predictions, we are also now conditioning on the entire past, present, and future sequence of instances and labels. Satisfaction of the strong independence condition implies satisfaction of the weak one.

These independence conditions hold under a variety of assumptions about the way the randomization of *WMR* is performed and the way the instances and labels are chosen. We can assume that the instances and labels are generated either deterministically or randomly without paying any attention to the predictions of the algorithm. For the weak independence

condition, we can also let them depend arbitrarily on the predictions made by the algorithm in previous trials. In the case that the generation of the instances and labels is independent of the randomization of the algorithm, the weak independence condition will hold even if the random choices of the algorithm are not independent of each other. For example, one way to implement the randomization of *WMR* is to choose a real number r uniformly from the range $[0, 1]$ and to predict 1 if $\gamma^{(j)} \geq r$. For the purpose of obtaining expected mistake bounds, it does not matter whether the random numbers used in each trial are chosen independently or not; in fact, a single choice of r can be made initially, and this same number can be used for each trial. In this case, the randomization of the algorithm is confined to this single choice. If instead a new value for r is chosen independently in each trial (and independently of the choice of the instances and labels), then the strong independence condition holds.

There is another way to perform the randomization. One can make each prediction by choosing a member of the pool at random, with probability proportional to its current weight, and then predicting 1 if that pool member predicts 1 and 0 if it predicts 0. If the pool member's prediction is strictly between 0 and 1, then the prediction is determined by a biased coin flip with the probability of predicting 1 equal to the value of the prediction of the chosen pool member. As above, the probability of predicting 1 is $\gamma^{(j)}$. If the random choices made by this version of *WMR* are independent of each other and of the choices of the instances and labels then the strong independence condition holds.

In the special case when $\beta = 0$ and all predictions of the pool members are binary, Maass has observed in independent research [Maa91] that it suffices to make new random choices in just the trials in which a mistake has been made. If a mistake is not made in a trial, then one uses the same pool member for prediction in the next trial. Maass views the algorithm *WMR* for this special case with the “lazy” update criterion from a different perspective than ours and has obtained an elegant derivation of the $\ln n$ bound on the expected number of mistakes made by *WMR* when there is an algorithm in the pool of n algorithms (with equal initial weights) that is consistent with all examples.

In the appendix we show that if the random choices at the end of each trial with a mistake are independent then the strong independence condition holds for *WMR* with the lazy update criterion and the restrictions that $\beta = 0$ and all predictions of the pool members are binary.

In this section we only show the weaker result that the weak independence condition holds, and we further restrict the instances and predictions of the pool members to be deterministic. We assume equal initial weights. Theorem 6.1 gives the $\ln n$ bound on the expected number of mistakes if there is a pool member consistent with all examples. Since the instances and predictions of the pool members are deterministic, to show that the weak independence condition holds it suffices to show that $\mathbf{E}(\lambda^{(j)}) = \gamma^{(j)}$. Thus it is sufficient to show that at each trial the member with which we predict is equally likely to be any of the remaining consistent pool members. (The choice of pool member is not independent from one trial to the next.)

We use induction on the trial number to prove this claim. The claim clearly holds at the first trial. Note that since the sequence of instances and the algorithms of the pool are deterministic, the determination of which pool members are consistent at each trial is not a probabilistic event. The probability of choosing a particular inconsistent pool member for a response at a given trial is 0. The probability of choosing a particular consistent pool member is the sum of the probability that it was the chosen member at the previous trial,

plus the probability that a mistake was made at the previous trial times the probability of choosing the particular member at this trial. If there were n_1 consistent members before the previous trial and n_2 before the current trial, then from the induction hypothesis the probability of making a mistake at the previous trial is $\frac{n_1 - n_2}{n_1}$. Thus the probability of choosing a particular consistent member at this trial is $\frac{1}{n_1} + \frac{n_1 - n_2}{n_1} \frac{1}{n_2} = \frac{1}{n_2}$, proving the claim.

The following theorem gives a bound on the expected number of mistakes made by WMR if the weak independence condition holds.

Theorem 6.1: *Let \mathcal{S} be any sequence of instances with binary labels. Let m be the number of mistakes made by WMR on the sequence \mathcal{S} when applied to some pool of prediction algorithms.*

Then under the weak independence condition we have $\mathbf{E}(m) \leq \frac{\mathbf{E}(\ln \frac{w_{init}}{w_{fin}})}{1-\beta}$.

Proof By the weak independence condition

$$\mathbf{E}(|\lambda^{(j)} - \rho^{(j)}| \mid (x^{(1)}, \rho^{(1)}), \dots, (x^{(j)}, \rho^{(j)})) = |\gamma^{(j)} - \rho^{(j)}|.$$

Thus $\mathbf{E}(|\lambda^{(j)} - \rho^{(j)}|) = \mathbf{E}(|\gamma^{(j)} - \rho^{(j)}|)$. Thus $\mathbf{E}(m) = \mathbf{E}(\sum_{j=1}^t |\lambda^{(j)} - \rho^{(j)}|) = \mathbf{E}(\sum_{j=1}^t |\gamma^{(j)} - \rho^{(j)}|)$. The desired bound follows immediately from Lemma 5.3. \square

If the predictions of the algorithms of the pool and the labels are all chosen deterministically, then the weights are also deterministic; the bound of Theorem 6.1 becomes

$$\mathbf{E}(m) \leq \frac{\ln \frac{w_{init}}{w_{fin}}}{1-\beta}.$$

This bound is identical to the bound on the total loss proven for WMC (Theorem 5.2). The comparison made at the end of Section 5 with the bound of WMG also applies here. The bound for WMR, like the bound for WMC, is better than the bound for WMG by nearly a factor of two for β close to 1. In the case of WMC this improvement was obtained by allowing predictions from the interval $[0, 1]$; here predictions are binary (as for WMG) and the improvement comes from the randomization. Recently, Vovk [Vov90b, Vov90a] has shown that with a different rule for computing the randomized prediction one can obtain the somewhat better expected mistake bound $\frac{\ln \frac{w_{init}}{w_{fin}}}{2 \ln \frac{2}{1+\beta}}$, exactly a factor of 2 better than our deterministic bound. The bound obtained here and Vovk's bound for his algorithm approach each other as β approaches 1.

We also obtain the following corollary, analogous to Corollary 5.1 for WMG.

Corollary 6.1: *Assume that \mathcal{A} is a pool of n prediction algorithms and that m_i is the total loss of the i -th algorithm of the pool on a sequence \mathcal{S} of instances with binary labels. If WMR is applied to pool \mathcal{A} with equal initial weights, then the expected number of mistakes is at most $\frac{\log n + \mathbf{E}(m_i) \log \frac{1}{\beta}}{1-\beta}$ on the sequence \mathcal{S} , for $1 \leq i \leq |\mathcal{A}|$.*

Proof This follows immediately from Theorem 6.1 and the fact that $w_{fin} \geq \sum_{i=1}^n w_i^{(1)} \beta^{m_i}$. \square

In the case that the strong independence condition holds, Chernoff bounds can be used to obtain bounds on the probability that the actual number of mistakes is much larger than the expected number.

We will use a theorem regarding Chernoff bounds applied to supermartingales that is given in [Lit89a]. Let (X, \mathcal{A}, P) be a probability space and let $\mathcal{G}_1, \dots, \mathcal{G}_n$ be σ -algebras contained in \mathcal{A} and let S_1, \dots, S_t be a sequence of random variables on this probability space. Then the sequence $(S_1, \mathcal{G}_1), \dots, (S_n, \mathcal{G}_n)$ is a *supermartingale* if $\mathcal{G}_1 \subseteq \dots \subseteq \mathcal{G}_n$, S_i is \mathcal{G}_i -measurable for $i = 1, \dots, n$, $\mathbf{E}(|S_i|)$ is finite for $i = 1, \dots, n$, and $\mathbf{E}(S_{i+1}|\mathcal{G}_i) \leq S_i$ (a.s.) for $i = 1, \dots, n-1$.

Theorem 6.2: [Lit89a] *Let (X, \mathcal{A}, P) be a probability space and let $\mathcal{G} \subseteq \mathcal{G}_0 \subseteq \mathcal{G}_1 \subseteq \dots \subseteq \mathcal{G}_t$ be σ -algebras contained in \mathcal{A} . Let ξ_1, \dots, ξ_t be a sequence of random variables on this probability space such that $0 \leq \xi_i \leq 1$ for each i , and let c_1, \dots, c_t be \mathcal{G} -measurable random variables with $0 \leq c_i \leq 1$ for $i = 1, \dots, t$. Let*

$$S_j = \sum_{i=1}^j (\xi_i - c_i)$$

and let $S_0 = 0$. Let $\mu = \frac{1}{t} \sum_{j=1}^t c_j$. Let α be a \mathcal{G} -measurable random variable. Then if the sequence $((S_0, \mathcal{G}_0), \dots, (S_t, \mathcal{G}_t))$ is a supermartingale, then the following statement holds at almost every point in X for which $0 < \mu < 1$. If $0 \leq \alpha \leq \mu$ then

$$P(S_t \geq \alpha t | \mathcal{G}) \leq e^{-\alpha^2 t / (3\mu)}$$

We obtain the following bound.

Theorem 6.3: *Let \mathcal{S} be any sequence of instances with binary labels and let \mathcal{A} be some pool of prediction algorithms. Suppose that WMR, applied to \mathcal{A} , is run on this sequence. Let m be a random variable giving the number of mistakes made. Suppose that the strong independence condition holds. If \mathcal{S} contains t instances, and if b and a are functions of $(x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})$ such that $\mathbf{E}(m | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})) \leq b$ (a.s.) and $0 < a \leq 1$, then*

$$P(m > (1+a)b | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})) \leq e^{-a^2 b / 3} \quad (\text{a.s.})$$

Proof Suppose there are t trials. Let $\xi_j = |\lambda^{(j)} - \rho^{(j)}|$. Thus $m = \sum_{j=1}^t \xi_j$. Let $c_j = \mathbf{E}(\xi_j | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})) + \frac{1}{t}(b - \mathbf{E}(m | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})))$. Thus $\sum_{j=1}^t c_j = b$. Let $S_j = \sum_{i=1}^j (\xi_i - c_i)$ and let $S_0 = 0$. Let $\mathcal{G} = \mathcal{G}_0$ denote the σ -algebra generated by $(x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})$, and for $j = 1, \dots, t$, let \mathcal{G}_j denote the σ -algebra generated by \mathcal{G} and $\lambda_1, \dots, \lambda_j$. Note that S_j is \mathcal{G}_j -measurable and since $\gamma^{(j)}$ is a function of the instances and labels it is \mathcal{G} -measurable. Note also that $\mathbf{E}(\xi_j | \mathcal{G})$ and $\mathbf{E}(\xi_j | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}))$ are two ways of writing the same thing. Under the strong independence assumption we have for $j = 1, \dots, n-1$

$$\mathbf{E}(\xi_j - c_j | \mathcal{G}_{j-1}) = |\gamma^{(j)} - \rho^{(j)}| - c_j \quad (\text{a.s.})$$

This is \mathcal{G} -measurable, and thus equals $\mathbf{E}(\xi_j - c_j | \mathcal{G}) = \mathbf{E}(\xi_j | \mathcal{G}) - c_j \leq 0$ (a.s.). Thus

$$\begin{aligned} \mathbf{E}(S_j | \mathcal{G}_{j-1}) &= S_{j-1} + \mathbf{E}(\xi_j - c_j | \mathcal{G}_{j-1}) \\ &\leq S_{j-1} \quad (\text{a.s.}) \end{aligned}$$

Thus if we let $\alpha = ab/t$ and $\mu = b/t$ the hypotheses of Theorem 6.2 are satisfied, yielding

$$P\left(\sum_{j=1}^t (\xi_j - c_j) \geq ab|\mathcal{G}\right) \leq e^{-\alpha^2 t / (3\mu)}$$

which is one way of writing the desired inequality. \square

We will give an application of this theorem in Section 8. There the instances and labels are deterministically chosen. One use of the theorem for random instances and labels is in a case where there exists some constant b_0 such that $P(b \leq b_0) \geq 1 - \delta$ for some small δ . Then $P(m > (1+a)b \text{ and } b \leq b_0 | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)})) \leq e^{-a^2 b_0 / 3}$ (a.s.) and thus $P(m > (1+a)b) \leq e^{-a^2 b_0 / 3} + \delta$.

7 Pools of Functions

In this section we consider the case in which the Weighted Majority Algorithm is applied to a pool of functions. (All functions are from the instance domain to $\{0, 1\}$.) We can think of such a pool of functions as a pool of prediction algorithms by interpreting a function f in the pool as the prediction algorithm that predicts $f(\mathbf{x})$ for any trial with instance \mathbf{x} . The results in this section apply to the case where there exists a function in the pool that is consistent with the sequence of trials. In this section we consider only deterministic algorithms for the case when all predictions are binary; the only master algorithm we work with is *WM* (This section is independent of Sections 5 and 6). We study what can be achieved by the Weighted Majority Algorithm if one desires to obtain better mistake bounds for some functions in a pool of algorithms at the expense of worse bounds for others. We show that under certain (rather strong) restrictions on the target class, *WM* is an optimal (deterministic) algorithm in a strong sense: given any deterministic on-line prediction algorithm A , there exists a way to choose initial weights for *WM* so that for each function in the target class the mistake bound for *WM* is at least as small as the mistake bound for A . (Note that this is stronger than just saying that the worst case mistake bound for *WM* for the target class is no larger than the worst case bound for A for the target class.)

At the end of this section we turn from finite to infinite target classes, giving results regarding what sequences of mistake bounds are possible for arbitrary target classes.

Theorem 7.1: *Let $\varphi_1, \dots, \varphi_n$ be a pool of functions with range $\{0, 1\}$ and let M_1, \dots, M_n be non-negative integers such that $\sum_{i=1}^n 2^{-M_i} < 2$. If algorithm *WM* is applied to the pool with initial weights $w_j = 2^{-M_j}$ and with $\beta = 0$, and if the sequence of trials is consistent with some function φ_i of the pool, then the algorithm makes at most M_i mistakes.*

Proof From Theorem 2.1, we obtain the following upper bound on the number of mistakes of *WM*:

$$\log_2 \sum_{j=1}^n 2^{-M_j} - \log_2 2^{-M_i} < 1 + M_i.$$

Since both M_i and the number of mistakes made are integers, this gives the desired bound. \square

We next show that in one special case the Weighted Majority Algorithm is optimal. We say that a set of $\{0, 1\}$ -valued functions, $\varphi_1, \dots, \varphi_n$, each with domain X , is *shattered* by X if $(\varphi_1(x), \dots, \varphi_n(x))$ ranges over all of $\{0, 1\}^n$ as x ranges over X . This notion of shattering has been considered by Assouad [Ass83]. It is dual to the notion of shattering used to define the Vapnik-Chervonenkis dimension of a concept class [VC71, BEHW89]. If a pool of functions is shattered by its domain, then there is a lower bound matching the above upper bound for the number of mistakes made by any deterministic algorithm.

Note that the applicability of the optimality result that follows is limited. A simple counting argument shows that for a domain X to shatter a function class requires the size of the function class to be no greater than $\log_2 |X|$.

We first give two lemmas that we will need.

Lemma 7.1: *Suppose that r_1, \dots, r_n are positive real numbers such that the quotient r_i/r_{i+1} is an integer for $i = 1, \dots, n-1$. Then if $0 < s \leq \sum_{i=1}^n r_i$ and s/r_1 is an integer, there exists an $m \leq n$ such that $\sum_{i=1}^m r_i = s$.*

Proof If $r_1 = s$ then we are done. Otherwise, $r_1 < s$. Let j be the largest integer such that $\sum_{i=1}^{j-1} r_i < s$. By hypothesis, $j-1 < n$ so $j \leq n$. From the choice of j we have $\sum_{i=1}^j r_i \geq s$. It will complete the proof of the lemma to show that it is also the case that $\sum_{i=1}^j r_i \leq s$. To see this, note that there must exist integers l and l' such that $\sum_{i=1}^{j-1} r_i = lr_j$ and $s = l'r_j$. Since we are assuming that $lr_j < l'r_j$ we must have $\sum_{i=1}^j r_i = (l+1)r_j \leq l'r_j = s$, as desired. \square

Lemma 7.2: *Given $r_1, \dots, r_j > 0$, suppose that $\log_2 r_1, \dots, \log_2 r_j$ are integers (not necessarily positive) and suppose that l is an integer such that $\max_i r_i \leq 2^l \leq \sum_{i=1}^j r_i$. Then there exists a set $K \subseteq \{1, \dots, j\}$ such that $\sum_{i \in K} r_i = 2^l$.*

Proof Let r'_1, \dots, r'_j be a permutation of r_1, \dots, r_j such that $r'_1 \geq \dots \geq r'_j$. Let $k_i = \log_2 r'_i$. Then $r'_i/r'_{i+1} = 2^{k_i - k_{i+1}}$, which is an integer for $i = 1, \dots, j-1$. Similarly, $2^l/r'_1$ is an integer. Thus by Lemma 7.1 there exists some m such that $\sum_{i=1}^m r'_i = 2^l$. \square

Theorem 7.2: *Let $\{\varphi_1, \dots, \varphi_n\}$ be any collection of $\{0, 1\}$ -valued functions that is shattered by its domain (the domain can be finite or infinite), let A be any deterministic on-line learning algorithm, and for $i = 1, \dots, n$ let M_i denote the maximum number of mistakes A makes on any sequence consistent with φ_i . Assume that all of these M_i are finite. Then $\sum_{i=1}^n 2^{-M_i} < 2$.*

Proof We will first give a rough outline of the proof. An adversary constructs a sequence of trials for the learning algorithm. The adversary assigns to each function φ_i in the target class a weight that is equal to 2^{-M_i} . These weights remain fixed, except that after each trial is generated the adversary sets to zero the weights of those functions that are not consistent with the trials that have been generated. The adversary picks instances and labels so that the learner makes a mistake at each trial and so that, no matter what the learning algorithm does, the sum of the weights of the consistent functions decreases by approximately a factor of two at each trial. Eventually, since consistent functions are eliminated at each trial, there must be a consistent function φ_j whose weight is a substantial fraction of the sum of the weights of the remaining consistent functions. If we let $s = \sum_{i=1}^n 2^{-M_i}$, then the sum of the weights of the remaining consistent functions will be roughly $s2^{-t}$, where t is the number

of trials that have been generated. The weight of the consistent function φ_j will be 2^{-M_j} . Our argument will show that for this j the value of $s2^{-t}/2^{-M_j}$ is bounded by some small constant, call it c for now. The learner will have made t mistakes; since φ_j is consistent with all of the trials, we must have $t \leq M_j$. Thus we have $s/c \leq 2^{t-M_j} \leq 1$. When we fill in the details this will give the bound of the theorem.

We now give the details of the proof. The adversary maintains weights corresponding to the functions $\varphi_1, \dots, \varphi_n$ in variables u_1, \dots, u_n . Initially for each i , $u_i = 2^{-M_i}$. The trials are generated by repeating the following procedure; each iteration other than the final one generates one trial. Let r denote the number of the trial to be generated during the current iteration. For the first iteration $r = 1$ and it is incremented by 1 at each iteration. At the beginning of the r th iteration, the adversary sets $k_r = \lfloor \log_2 \sum_{i=1}^n u_i \rfloor$. Thus $2^{k_r} \leq \sum_{i=1}^n u_i < 2^{k_r+1}$. (Note that k_r is not necessarily positive.) If $\max_i u_i > 2^{k_r-1}$ then the adversary stops without generating any further trials. A total of $r-1$ trials will have been generated. Otherwise, we apply Lemma 7.2, with the r_i in the lemma corresponding to the current values of the u_i . (In iterations after the first, some of the u_i will be zero; in that case we apply the lemma to the subsequence consisting of the u_i that are non-zero.) Since $\max_i u_i \leq 2^{k_r-1} < \sum_{i=1}^n u_i$, this lemma tells us that there exists some $K \subseteq \{1, \dots, n\}$ such that $\sum_{i \in K} u_i = 2^{k_r-1}$. The adversary chooses an instance \mathbf{x} for the current trial such that $\varphi_i(\mathbf{x}) = 0$ for $i \in K$ and $\varphi_i(\mathbf{x}) = 1$ if $i \notin K$. This is possible because the target class is shattered by the domain. It chooses the label ρ so that it is not equal to the prediction of algorithm A . (We will argue later that there will be at least one function in the target class consistent with all pairs of instances and labels chosen by the adversary.) Thus the learner makes a mistake at each trial. In preparation for the next iteration, the adversary sets u_i to zero if $\varphi_i(\mathbf{x}) \neq \rho$, for each i .

The adversary continues to generate trials by repeating this strategy until it terminates as described. In a moment, we will argue that it does indeed terminate. First note that at the r th iteration, $\sum_{i=1}^n u_i \geq 2^{k_r}$. If the strategy does not terminate at the beginning of this iteration, then for the set K determined during this iteration, $\sum_{i \in K} u_i = 2^{k_r-1}$. Thus we also have $\sum_{i \notin K} u_i \geq 2^{k_r-1}$. Hence when the adversary updates the u_i at the end of the iteration, it will decrease $\sum_{i=1}^n u_i$ by at least 2^{k_r-1} . Furthermore, the new value of $\sum_{i=1}^n u_i$ will be at least 2^{k_r-1} . The strategy must terminate since at least one non-zero u_i is set to zero during each iteration.

For $r > 1$, the argument of the previous paragraph tells us that at the beginning of the r th iteration, $\sum_{i=1}^n u_i \geq 2^{k_{r-1}-1}$. Since at that time $2^{k_r+1} > \sum_{i=1}^n u_i$ we have $k_r + 1 > k_{r-1} - 1$. Therefore, $k_r \geq k_{r-1} - 1$. Hence $k_r \geq k_1 - (r - 1)$. This holds even if termination occurs at the beginning of iteration r . If a total of t trials are generated, then termination occurs at the beginning of the $(t + 1)$ st iteration, at which time we have $\max_i u_i > 2^{k_{t+1}-1} \geq 2^{k_1-(t+1)}$.

We now show how to choose a target function φ_j . Choose j such that at the beginning of iteration $t + 1$ we have $u_j = \max_i u_i$. Since $u_j > 0$ it is still at its initial value. Substituting the definitions of u_j and k_1 into the inequality at the end of the previous paragraph, we obtain

$$2^{-M_j} > 2^{\lfloor \log_2 \sum_{i=1}^n 2^{-M_i} \rfloor - (t+1)}$$

From this we get

$$t + 1 - M_j > \left\lceil \log_2 \sum_{i=1}^n 2^{-M_i} \right\rceil$$

Since u_j is non-zero at termination, the function φ_j must have been consistent with all of the labels and therefore could be the target function. Since the learner has made t mistakes, we must have $t \leq M_j$, so

$$\left\lceil \log_2 \sum_{i=1}^n 2^{-M_i} \right\rceil < 1$$

Thus

$$\log_2 \sum_{i=1}^n 2^{-M_i} < 1$$

so

$$\sum_{i=1}^n 2^{-M_i} < 2$$

as desired. \square

Suppose that for some set of integers M_1, \dots, M_n we want to construct a learning algorithm that makes no more than M_i mistakes if the target function turns out to be φ_i for $i = 1, \dots, n$. When the domain shatters the target class $\{\varphi_1, \dots, \varphi_n\}$, the following corollary tells us that if any algorithm can accomplish this, then there is a way to choose the initial weights for WM that will do so.

Corollary 7.1: *Let $\{\varphi_1, \dots, \varphi_n\}$ be a collection of functions with range $\{0, 1\}$ that is shattered by the domain X . Suppose that A is an on-line learning algorithm and that M_1, \dots, M_n are positive integers such that A makes at most M_i mistakes when given a sequence of trials consistent with φ_i , for $i = 1, \dots, n$. Suppose algorithm WM is applied to the pool $\varphi_1, \dots, \varphi_n$ with initial weights $w_j = 2^{-M_j}$ and with $\beta = 0$. If there exists an i in the range $1, \dots, n$ such that the sequence of trials is consistent with φ_i , then WM makes at most M_i mistakes.*

Proof From Theorem 7.2 we have $\sum_{i=1}^n 2^{-M_i} < 2$. Thus we obtain the desired mistake bound from Theorem 7.1. \square

We also can obtain interesting results for infinite pools, using algorithm WMI₂.

Theorem 7.3: *Let M_1, M_2, M_3, \dots be an infinite sequence of non-negative integers. Assume that the values M_i are given by a computable function of i . Then we have (a) \implies (b) \implies (c) \implies (d), where (a), (b), (c), and (d) are the following statements:*

(a) $\sum_{i=1}^{\infty} 2^{-M_i}$ is finite and its value can be computed to arbitrary precision.

(b) There exists $c \in \mathbf{R}$ such that for all domains X and computably indexed collections of total recursive functions $\varphi_1, \varphi_2, \varphi_3, \dots$ on X there exists an on-line prediction algorithm A such that for all positive integers i , for all sequences of trials consistent with φ_i , the algorithm A makes at most $M_i + c$ mistakes.

(c) For all domains X and computably indexed collections of total recursive functions $\varphi_1, \varphi_2, \varphi_3, \dots$ on X there exists $c \in \mathbf{R}$, $j > 0$, and an on-line prediction algorithm A such that for all integers $i \geq j$, for all sequences of trials consistent with φ_i , the algorithm A makes at most $M_i + c$ mistakes.

(d) $\sum_{i=1}^{\infty} 2^{-M_i}$ is finite.

It is interesting to state the contrapositive of the last implication: If $\sum_{i=1}^{\infty} 2^{-M_i}$ diverges, then there exists a domain X and a computably indexed collection of total recursive functions $\varphi_1, \varphi_2, \varphi_3, \dots$ such that for all $c \in \mathbf{R}$ and $j > 0$, and for all on-line prediction algorithms A there exists an integer $i \geq j$ for which there is a sequence of trials consistent with φ_i on which the algorithm A makes greater than $M_i + c$ mistakes. Since this holds for all positive j , in fact there exist infinitely many such i .

Proof Statement (c) is weaker than (b). Thus to prove the theorem it suffices to demonstrate that (a) implies (b) and that (c) implies (d). To see that (a) implies (b) we construct an on-line prediction algorithm A using WMI_2 . We apply WMI_2 to the pool $\varphi_1, \varphi_2, \dots$ with initial weights 2^{-M_i} . We can construct the function \widehat{W} needed by WMI_2 by using our ability to compute the sum of the initial weights to arbitrary precision. This, coupled with our ability to compute initial partial sums of the weights lets us compute upper bounds on $\sum_{j=i}^{\infty} 2^{-M_j}$ that approach 0 as i goes to infinity. We set the parameter β of WMI_2 to zero. Then if φ_i is consistent with a sequence of trials, then Theorem 4.1 gives a bound of $\log_2 \widehat{W}(1) + M_i + 1$. We can thus take $c = \log_2 \widehat{W}(1) + 1$.

(c) implies (d): We choose the domain X to consist of all finite strings of 0's and 1's. We choose the collection of functions $\varphi_1, \varphi_2, \dots$ to be the functions defined by $\varphi_i(x) = 1$ if the string x is of length at least i , and if the i th bit of x is 1; otherwise $\varphi_i(x) = 0$. Choose c and j appropriately to obtain the mistake bounds promised by the hypothesis (c). Notice that for any $n \geq j$, the collection $\varphi_j, \dots, \varphi_n$ is shattered by the domain. Thus for the algorithm A to have the given mistake bounds we must have $\sum_{i=j}^n 2^{-(M_i+c)} < 2$, by Theorem 7.2. Thus the partial sums of $\sum_{i=1}^{\infty} 2^{-M_i}$ are increasing and bounded above, so the series converges to a finite sum as desired. \square

The algorithm that we use for parts (b) and (c) is thus a version of WMI_2 . As indicated in the preceding proof, we can take the constant c of part (b) to be $\log_2 \widehat{W}(1) + 1$. The value of $\widehat{W}(1)$ is some computable upper bound on $\sum_{i=1}^{\infty} 2^{-M_i}$. By making this upper bound sufficiently precise, we can take c to be $\log_2 \sum_{i=1}^{\infty} 2^{-M_i} + 2$. (Here the additive constant 2 could be replaced by any constant greater than 1.) If we are to choose a constant c that will work regardless of the target class, then the proof indicates that we must have $\sum_{i=1}^n 2^{-(M_i+c)} < 2$ for all n . Thus we must have $\sum_{i=1}^{\infty} 2^{-(M_i+c)} \leq 2$, that is, $c \geq \log_2 \sum_{i=1}^{\infty} 2^{-M_i} - 1$. This theorem lets us rederive some of the results of Barzdin and Freivalds [BF72,BF74], given in the following corollary.

Corollary 7.2: ([BF72,BF74]) *For any algorithm A and function f , let $M(A, f)$ denote the maximum number of mistakes made by A on any sequence of trials consistent with f . Given any domain X ,*

- (a) *for every computably indexed collection of total recursive functions $\varphi_1, \varphi_2, \dots$ on X there exists an on-line prediction algorithm A such that $M(A, \varphi_1)$ is finite and $M(A, \varphi_i) = \log_2 i + \log_2 \log i + o(\log \log i)$, for $i > 1$.*
- (b) *there exists a computably indexed collection of total recursive functions $\varphi_1, \varphi_2, \dots$ on X such that for every on-line prediction algorithm A there exist infinitely many i such that $M(A, \varphi_i) > \log_2 i + \log_2 \log i$.*

Proof Part (a) follows immediately from Theorem 7.3 and the fact that $\sum_{i=2}^{\infty} \frac{1}{i \ln i (\ln \ln i)^2}$ converges. Part (b) follows from Theorem 7.3 and the fact that $\sum_{i=2}^{\infty} \frac{1}{i \ln i}$ diverges. (See

the statement of the contrapositive of the final implication of the theorem that appears immediately following the theorem.) \square

We also obtain the following result given in slightly weaker form in [BF74]. (Their lower bound is $\log_2 n - 3$.)

Theorem 7.4: *There exists a domain X and a computably indexed class of total recursive functions $\varphi_1, \varphi_2, \dots$ such that for any $n \geq 1$ and any on-line prediction algorithm A there exists a sequence of trials consistent with some function from $\{\varphi_1, \dots, \varphi_n\}$ on which A makes at least $\lfloor \log_2 n \rfloor$ mistakes.*

Proof We choose the domain and target class as in the proof of the implication (c) \implies (d) of Theorem 7.3. Thus if M_i is a mistake bound for the given algorithm for sequences consistent with φ_i we have $\sum_{i=1}^n 2^{-M_i} < 2$. Let $M = \max_{i \in \{1, \dots, n\}} M_i$. Then $n2^{-M} < 2$. Thus $M > \log_2 n - 1$. Since M is an integer, this implies that $M \geq \lfloor \log_2 n \rfloor$, as desired. \square

8 Anomalies

We continue to consider pools of functions, now without the requirement of shattering imposed in part of the previous section. In this section we consider the case where there is no function in the pool that is consistent with all of the trials. With respect to a given function, we define the number of *anomalies* in a sequence of trials to be the number of trials that are inconsistent with that function. Given a pool of functions F , we will say that a sequence has η anomalies if η is the minimum number of anomalies of the sequence with respect to any function in F . In this case we cannot necessarily say that particular trials are anomalous; for example, there may be two functions in F minimizing the number of anomalies, and they may be consistent with different trials. As in the previous section, we will assume that all functions in the pool are $\{0, 1\}$ -valued. In this section we will consider the randomized algorithm *WMR* as well as *WM*. We will assume that the instances and labels are chosen deterministically.

Note that there are two types of situations in which anomalies arise: one in which the sequence of trials is in fact consistent with some target function, but not with any function in the pool, and the other in which the same instance appears in the sequence with different labels at different appearances. The latter may occur if there are errors in the instances or the labels, or if there is insufficient information reported in each instance to uniquely determine the appropriate label. Our upper bounds apply in both types of situations. Our lower bounds are for the second type of situation.

We give a lower bound that shows that the rate at which the mistake bound for the deterministic version of *WM* grows with the number of anomalies can be made (by appropriate choice of β) arbitrarily close to the best possible for deterministic prediction algorithms. We also show that the rate at which the expected mistake bound for *WMR* grows can be made arbitrarily close to the best possible rate of growth for randomized algorithms. Given some class of functions F , let S_η be the collection of all sequences of trials \mathcal{S} that have at most η anomalies. We define $\text{opt}(F, \eta)$ to be the minimum over all deterministic algorithms A of the maximum over all sequences $\mathcal{S} \in S_\eta$ of the number of mistakes made by A on \mathcal{S} . Thus the value $\text{opt}(F, \eta)$ is thus the best bound for the class F that can be obtained (deterministically) in the presence of η anomalies. We define $\text{opt}_{\text{RAND}}(F, \eta)$ to be the minimum over all algorithms (including randomized algorithms)

A of the maximum over all sequences $\mathcal{S} \in S_\eta$ of the expected number of mistakes made by A on \mathcal{S} . The randomization used by A is assumed to be independent of the choice of the sequence.

Theorem 8.1: *For all target classes F and all $\eta \geq 0$, if $|F| > 1$ then $\text{opt}(F, \eta) \geq \text{opt}(F, 0) + 2\eta$.*

Proof Let $k = \text{opt}(F, 0)$. Since $|F| > 1$, $k \geq 1$. Saying that $\text{opt}(F, \eta) \geq k + 2\eta$ is equivalent to saying that for any deterministic learning algorithm A , there exists a function $f \in F$ and a sequence of trials having at most η anomalies with respect to f , such that A makes at least $k + 2\eta$ mistakes when presented with that sequence of trials. Given an algorithm A , we will show how an adversary can choose a function and a sequence of instances such that A makes at least $k + 2\eta$ mistakes.

We will use the notion of a mistake tree used in [Lit88,Lit89b]. A *mistake tree* for a pool of functions F over a domain X is a binary tree each of whose nodes is a non-empty subset of F and each of whose internal nodes is labeled with a point of X . Its root is F . Given any internal node F' labeled \mathbf{x} , the left child of that node, if present, must be the subset of F' consisting of all functions in F' that are 0 at \mathbf{x} . (The left child can be present only if this subset is non-empty.) The right child, if present, must be the set of functions in F' that are 1 at \mathbf{x} . (Again, this subset must be non-empty for the right child to be present.) A *complete k -mistake tree* is a mistake tree that is a complete binary tree of height k . (We define the height of a tree to be the length in edges of the longest path from the root to a leaf.) It is shown in [Lit88,Lit89b] that for any pool of functions F there exists a complete $\text{opt}(F, 0)$ -mistake tree.

The adversary's strategy is divided into two stages. For the first stage, the adversary keeps track of a current mistake tree. Initially this is a complete k -mistake tree for F . If $k = 1$, the adversary proceeds immediately to the second stage. Otherwise, the first instance chosen by the adversary is the label of the root of the tree. Whatever the algorithm predicts, the adversary tells the algorithm that its prediction is wrong. This response of the adversary eliminates some functions as possible target functions. One of the two subtrees of the root of the adversary's current mistake tree is a complete $k - 1$ mistake tree for the remaining candidate functions. The adversary sets its current mistake tree to that subtree. It chooses the next instance to be the label of the root of the new current tree. The adversary continues in this manner, forcing the algorithm to be wrong at each instance. After j mistakes, the adversary's current tree is a complete $k - j$ mistake tree for the remaining candidate target functions. As long as $j < k$, the root of the current tree has two children corresponding to non-empty subclasses of F ; thus the adversary can choose a point (the label of the root) at which it can force the algorithm to make a mistake. The adversary continues with the first stage until $k - 1$ mistakes have been made.

At the end of the first stage, there will be an instance \mathbf{x} and a pair of functions f_1 and f_2 in F both of which are consistent with the first $k - 1$ trials such that $f_1(\mathbf{x}) \neq f_2(\mathbf{x})$. (The instance \mathbf{x} is the label of the root of the final 1-mistake tree of the adversary, and the functions f_1 and f_2 can be chosen from the left and right children of the root.) The adversary now generates $2\eta + 1$ trials, each with the same instance \mathbf{x} . For each trial the adversary chooses the label to be unequal to the prediction of the algorithm. Suppose without loss of generality that for the majority of the second-stage trials the generated labels are consistent with f_1 . Then at most η of the trials are inconsistent with f_1 . Thus we have found a sequence of trials with at most η anomalies with respect to F on which the algorithm has made $2\eta + k$ mistakes, as desired. \square

In the case of η anomalies, our mistake bound for WM is $\frac{\log |F| + \eta \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ (see Corollary 2.2, $k = 1$). As β approaches 1, the coefficient of η approaches the optimal value of 2. However, at the same time the coefficient of $\log |F|$ approaches infinity. Thus some compromise is needed. For $\beta = \frac{1}{2}$, we have the bound $\frac{\log_2 |F| + \eta}{\log_2 \frac{4}{3}}$. For some concept classes $opt(F, 0) = \log_2 |F|$; for others it is strictly less, and there is an additional gap between the lower bound that we give and our bound for WM .

We can establish a lower bound for randomized algorithms by essentially the same argument. The lower bound applies even in the case that the choice of the instances and labels is made without any dependence on the predictions of the algorithm, and thus it also applies to the model in which such dependence is allowed.

Theorem 8.2: *For all target classes F and all $\eta \geq 0$, if $|F| > 1$ then $opt_{RAND}(F, \eta) \geq \frac{1}{2}opt(F, 0) + \eta$.*

Proof We describe how to adapt the proof of Theorem 8.1. We require the adversary to compute the sequence of instances and labels ahead of time, since we do not want the sequence to depend on the random choices made by the algorithm. We can, however, assume that the adversary has full knowledge of the algorithm. The adversary proceeds exactly as above, except that instead of choosing labels as above, it determines the probability that the algorithm will predict 1 at the current trial, given the preceding sequence of instances and labels chosen by the adversary. (The probability is not conditioned on previous responses of the algorithm.) The adversary chooses the label 0 if and only if this probability exceeds $\frac{1}{2}$. Thus at each trial the expected number of mistakes is at least $\frac{1}{2}$, giving a lower bound one-half the size of the lower bound we obtained for deterministic algorithms. \square

Suppose we run WMR with $0 < \beta < 1$ on a pool of size n with equal initial weights and that there are η anomalies. Let m denote the number of mistakes made by WMR . Corollary 6.1 gives an upper bound on the expected number of mistakes of

$$\mathbf{E}(m) \leq \frac{\ln n + \eta \ln \frac{1}{\beta}}{1 - \beta}$$

In this section we have shown that the best possible value of the coefficient of η in the upper bound for any algorithm is 1. Its value in the upper bound for WMR can be made arbitrarily close to 1 by choosing β sufficiently close to 1.

It is also interesting to look at the upper bound given by Theorem 6.3 in this situation. Since the instances and labels are here assumed to be chosen deterministically, by Theorem 6.3 for $0 < a \leq 1$

$$P(m > \frac{(1+a)\ln n}{1-\beta} + \frac{\eta(1+a)\ln(1/\beta)}{1-\beta}) \leq e^{\frac{-a^2(\ln n + \eta \ln(1/\beta))}{3(1-\beta)}}$$

By making β close to 1 and a close to 0, the factor $\frac{(1+a)\ln(1/\beta)}{1-\beta}$ multiplying η can be made arbitrarily close to 1. A natural assumption is that the number of anomalies grows in proportion to the number of trials. In that case, after sufficiently many trials, with β sufficiently close to 1, we will have high confidence that the ratio of the number of mistakes to the number of anomalies is not much more than 1. (By contrast, in the bound for the deterministic algorithm WM , the factor multiplying η is always greater than 2.)

Master Algorithm	Predictions of Pool Members	Predictions of Master Algorithm	Labels	Bound	Comments
<i>WM</i>	binary	binary	binary	$\frac{\ln \frac{w_{init}}{w_{fin}}}{\ln \frac{2}{1+\beta}}$	
<i>WML</i>	binary	binary	binary	(Thm 3.1)	for shifting target
<i>WMI₂</i>	binary	binary	binary	(Thm 4.1)	for infinite pools
<i>WMG</i>	[0, 1]	binary	binary	$\frac{\ln \frac{w_{init}}{w_{fin}}}{\ln \frac{2}{1+\beta}}$	
<i>WMC</i>	[0, 1]	[0, 1]	[0, 1]	$\frac{\ln \frac{w_{init}}{w_{fin}}}{1-\beta}$	
<i>WMR</i>	[0, 1]	binary	binary	$\frac{\ln \frac{w_{init}}{w_{fin}}}{1-\beta}$	randomized predictions

Table 9.1: Summary of Weighted Majority Versions

9 Conclusion

We have investigated various master prediction algorithms that use the predictions of a pool of algorithms to make their own predictions. Initially each member of the pool is given a positive weight. The weight of a pool member represents the “belief” of the master algorithm in the predictions of the member. These weights are updated depending on how good the predictions of the corresponding algorithms are. In all our algorithms the prediction of the master is based on the weighted average of the predictions of the pool members.

In the most basic master algorithm all predictions of the pool members and the master as well as the labels of the examples are required to be binary. For that algorithm the master simply predicts in the same way as the weighted majority of the pool members do. Various variants of this basic algorithm have been introduced that allow continuous predictions and labels in $[0, 1]$. We have also given a probabilistic variant. Table 9.1 gives a summary.

We have developed upper bounds for the predictive performance of the master algorithms in terms of the performance of the algorithms in the pool. We have applied *WM* in various settings (corollaries 2.1 to 2.3). Similar corollaries hold for all variants. The simplest corollary gives a bound on the performance of the master as a function of the best algorithm (specialist) in the pool (Corollary 2.1). We have given a version *WML* of *WM* that works well for cases where different pool members are specialists for various sections of the trial sequence. Probabilistic versions and versions that allow continuous predictions which also have the capability of tracking the specialist can easily be developed. The same holds for the version *WMI₂* of *WM* that allows the pool to be countably infinite. Furthermore it would be easy to combine the capabilities of tracking the specialist and handling infinite pool size into a single algorithm.

We have tried to keep the exposition simple in that each master algorithm focuses on one setting in which weighted majority techniques are useful. We have developed a unified proof method for all bounds as well as given simple direct proofs that sometimes seem unrelated.

Note that each of the master algorithms spends little computation time beyond that required for simulating the algorithms in the pool (which might be done in parallel). The number of operations required in each trial to compute the prediction of the master algorithm and to update the weights, if necessary, is linear in the size of the pool for all algorithms other than WMI_2 , where it is linear in the size of the active subpool plus the time required to compute \widehat{W} .

As discussed in the introduction our techniques can be used to find the best setting of parameters of a given algorithm and to establish loss bounds for learning classes of functions. Our methods justify the following philosophy: For a given application find a large pool of candidate prediction algorithms which might work well for the application and combine their predictions using one of the master algorithms presented. (The practical size of the pool will depend on available computational resources.) The additional loss of the master over the best pool member is essentially only logarithmic in the size of the pool and there is only minimal computational overhead aside from running all pool member in parallel. One might design schemes for removing algorithms from the pool if their weight degenerates after a reasonable number of examples have been processed and schemes for replacing the removed algorithms by new trial candidates. However if the trial sequence is non-stationary in that a previously bad algorithm may become a specialist at a later point, then algorithms should be removed only temporarily.

One might also design master algorithms for case where not all algorithms of the pool are available during the initial trials but the algorithms of the pool “wake up” and “fall asleep” at various times. Finally one could use our master algorithms for designing simple networks of algorithms. In each node the input predictions are combined into an output prediction by using one of the master algorithms. The simplest network would be a tree with the algorithms of the pool at the leaves and WM algorithms at the internal nodes. In case of a mistake the weights of the subpool of the children of the root which predicted wrong are updated and recursively, for all nodes whose weights were updated, the weights of the subpool of their children that predicted wrong are updated. Does this hierarchical application have any advantage over a single master that combines all predictions of the pool?

In all master algorithms presented the parameter β ($0 \leq \beta < 1$) measures how drastic the update is. (The smaller β the more drastic the update.) We always kept β constant for all trials. Particularly when there is statistical noise on the examples, it might be advantageous to slowly increase β with time.

10 Acknowledgments

We thank David Haussler, David Helmbold and Wolfgang Maass for valuable discussions.

References

- [Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.
- [Ass83] Patrick Assouad. Densité et dimension. *Ann. Inst. Fourier, Grenoble*, 33(3):233–282, 1983.

- [BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *JACM*, 36(4):929–965, 1989.
- [BF72] J. M. Barzdin and R. V. Freivalds. On the prediction of general recursive functions. *Sov. Math. Dokl.*, 13:1224–1228, 1972.
- [BF74] J. M. Barzdin and R. V. Freivalds. Prognozirovanie i predel’nyi sintez effektivno perechislimykh klassov funktsii (prediction and limit synthesis of effectively enumerable classes of functions). In J. M. Barzdin, editor, *Theory of Algorithms and Programs*, volume 1, pages 101–111. Latvian State University, 1974. (in Russian).
- [Bil86] Patrick Billingsley. *Probability and Measure*. Wiley, New York, 1986.
- [DMW88] Alfredo DeSantis, George Markowski, and Mark N. Wegman. Learning probabilistic prediction functions. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 312–328, Published by Morgan Kaufmann, San Mateo, CA, 1988.
- [FSV89] R. V. Freivalds, C. H. Smith, and M. Velauthapillai. Trade-off among parameters affecting inductive inference. *Information and Computation*, 82:323–349, 1989.
- [HKS91] David Haussler, Michael Kearns, and Robert Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 61–74, Published by Morgan Kaufmann, San Mateo, CA, 1991.
- [HO91] David Haussler and Manfred Opper. Calculation of the learning curve of Bayes optimal classification algorithm for learning a perceptron with noise. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 75–87, Published by Morgan Kaufmann, San Mateo, CA, 1991.
- [HSW90] David Helmbold, Robert Sloan, and Manfred K. Warmuth. Learning nested differences of intersection-closed concept classes. *Machine Learning*, 5:165–196, 1990. Special issue for the Second Annual Workshop on Computation Learning Theory, 1989, Santa Cruz, California.
- [Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
- [Lit89a] Nick Littlestone. From on-line to batch learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 269–284, Published by Morgan Kaufmann, San Mateo, CA, 1989.
- [Lit89b] Nick Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. PhD thesis, Technical Report UCSC-CRL-89-11, University of Calif., Santa Cruz, 1989.
- [LTS89] Esther Levin, Naftali Tishby, and Sarah A. Solla. A statistical approach to learning and generalization in layered neural networks. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 245–258, Published by Morgan Kaufmann, San Mateo, CA, 1989.
- [LW89a] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, pages 256–261. IEEE, 1989.

- [LW89b] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. Technical Report UCSC-CRL-89-16, University of Calif. Computer Research Laboratory, Santa Cruz, CA 95064, 1989.
- [Maa91] Wolfgang Maas. On-line learning with an oblivious environment and the power of randomization. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 167–175, Published by Morgan Kaufmann, San Mateo, CA, 1991.
- [Pit89] Lenny Pitt. Probabilistic inductive inference. *JACM*, 36(2):383–433, 1989.
- [PS88] L. Pitt and C. H. Smith. Probability and plurality of aggregations of learning machines. *Information and Computation*, 77(1):77–92, 1988.
- [Shi84] Albert Nikolaevich Shiryaev. *Probability*. Springer, New York, 1984.
- [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.
- [Vov90a] Volodimir G. Vovk. private communication, 1990.
- [Vov90b] Volodimir G. Vovk. Aggregating strategies. In *Proceedings of the Third Workshop on Computational Learning Theory*, pages 371–383, Published by Morgan Kaufmann, San Mateo, CA, 1990.

Appendix

We first give alternate proofs for theorems 5.1 and 6.1 which give upper bounds on the loss of *WMG* and *WMR*, respectively. Secondly, we show that an algorithm introduced by Maass (described in Section 6) satisfies the strong independence condition.

The alternate proofs are simple and are very similar to the proof of the upper bound on the number of mistakes of *WM* given in Theorem 2.1. Recall that in that proof q_0 is the sum of all the current weights w_i such that the i -th algorithm predicts 0 and q_1 is defined similarly. Thus the i -th algorithm either contributes all of its weight to q_0 or to q_1 . For *WMG* and *WMR* we allow the prediction x_i to be continuous in $[0, 1]$ and if $x_i^{(j)}$ is not 0 or 1, the weight w_i is split between q_0 and q_1 , leading to the following more general definition:

$$q_0 = \sum_{i=1}^n w_i(1 - x_i) \text{ and } q_1 = \sum_{i=1}^n w_i x_i.$$

Note that the total weight before the trial is again $q_0 + q_1$. Also if $x_i \in \{0, 1\}$ then the above coincides with the previous definition of q_0 and q_1 which was used for *WM*.

Alternate proof of Theorem 5.1, the upper bound on the total loss of *WMG*:
Recall that *WMG* may predict 0 if $q_0 \geq q_1$ and 1 if $q_1 \geq q_0$.

If no mistake occurs in a trial then the weight may only decrease. As in the proof of Theorem 2.1 we only have to show that if *WMG* makes a mistake then the total weight after the update is at most $\frac{1+\beta}{2}(q_0 + q_1)$.

Assume $q_1 \geq q_0$ at trial j and *WMG*'s prediction is 1 even though the correct prediction is $\rho = 0$. Then the total after the update is

$$\sum_{i=1}^n w_i(1 - (1 - \beta)x_i) = \sum_{i=1}^n w_i(1 - x_i + \beta x_i) = q_0 + \beta q_1.$$

In the other case assume $q_0 \geq q_1$ and WMG's prediction is 0 even though the correct prediction is $\rho = 1$. Then

$$\sum_{i=1}^n w_i(1 - (1 - \beta)(1 - x_i)) = \sum_{i=1}^n w_i(\beta(1 - x_i) + x_i) = \beta q_0 + q_1.$$

In both case the larger q_0 and q_1 is multiplied by β and thus as in the proof of Theorem 2.1, it is easy to show that the total after the update is at most $\frac{1+\beta}{2}(q_0 + q_1)$. \square

Alternate proof of Theorem 6.1, the upper bound on the expected total loss of WMR: Assume there are t trials. For the j -th trial, with sums of weights q_0 and q_1 as described above, and with label $\rho^{(j)}$, let $s^{(j)} = q_0 + q_1$ and let $u^{(j)} = q_b$, where b is the complement of the label $\rho^{(j)}$. Thus $s^{(1)} = w_{init}$. Let $s^{(t+1)} = w_{fin}$. In the j -th trial the probability that WMR makes a mistake is $\frac{u^{(j)}}{s^{(j)}}$. Also the total weight decreases by $(1 - \beta)u^{(j)}$. We observe that $s^{(j+1)} = s^{(j)} - (1 - \beta)u^{(j)}$ and

$$\int_{s^{(j+1)}}^{s^{(j)}} \frac{1}{x} dx \geq \int_{s^{(j+1)}}^{s^{(j)}} \frac{1}{s^{(j)}} dx = (1 - \beta) \frac{u^{(j)}}{s^{(j)}}.$$

The expected number of mistakes in the t trials is the sum of the probabilities of making a mistake in each trial, which equals

$$\sum_{j=1}^t \frac{u^{(j)}}{s^{(j)}} \leq \frac{1}{1 - \beta} \sum_{j=1}^t \int_{s^{(j+1)}}^{s^{(j)}} \frac{1}{x} dx = \frac{1}{1 - \beta} \int_{s^{(t+1)}}^{s^{(1)}} \frac{1}{x} dx = \frac{\ln \frac{s^{(1)}}{s^{(t+1)}}}{1 - \beta}$$

as desired. \square

Next we show that an appropriately constructed version of the algorithm introduced by Maass (described in Section 6) satisfies the strong independence condition. We will use the following lemma regarding conditional expectations (cf. [Bil86, exercise 34.4]):

Lemma A.1: *Let ξ be a random variable such that $\mathbf{E}(|\xi|) < \infty$. Let η be a $\{0, 1\}$ -valued random variable, let \mathcal{G} be some finite σ -algebra, and let \mathcal{A} be the σ -algebra generated by \mathcal{G} and η . Then*

$$\eta \mathbf{E}(\eta \xi | \mathcal{G}) = \mathbf{E}(\eta | \mathcal{G}) \mathbf{E}(\eta \xi | \mathcal{A}) \quad (a.s.)$$

Proof For any atom $U \in \mathcal{G}$ such that $P(U) > 0$ we have

$$\int_U \mathbf{E}(\eta \xi | \mathcal{G}) = \int_U \eta \xi = \int_U \mathbf{E}(\eta \xi | \mathcal{A}) = \int_U \eta \mathbf{E}(\xi | \mathcal{A}) = a \int_U \eta$$

where a is the unique value of $\mathbf{E}(\xi | \mathcal{A})$ on the set $U \cap \eta^{-1}(1)$. This equals

$$a \int_U \mathbf{E}(\eta | \mathcal{G}) = abP(U)$$

where b is the unique value of $\mathbf{E}(\eta | \mathcal{G})$ on the set U . Let c be the unique value of $\mathbf{E}(\eta \xi | \mathcal{G})$ on U . Then we obtain $cP(U) = abP(U)$. This gives the desired result for any point in $U \cap \eta^{-1}(1)$. The result is trivial on the rest of U , thus proving equality holds almost surely, as desired. \square

Theorem A.1: *Assume that all pool members make binary predictions. Suppose that algorithm WMR is implemented as follows: The weights are updated in each trial as described in Section 5, using $\beta = 0$. Since the pool members' predictions are binary this amounts to either leaving each weight unchanged or setting it to zero, depending on whether or not the corresponding pool member's prediction matches the label. Each prediction is made by making the same prediction that a randomly chosen pool member makes. However, a new pool member is not chosen in every trial. Initially, and at the beginning of any trial immediately following a mistake, a new choice is made. Otherwise the most recently chosen pool member is used. Suppose that the random choice is made as follows: at each trial j for which a new pool member must be chosen, the algorithm independently chooses a real number $r^{(j)}$ uniformly from $[0, 1]$. It then chooses the pool member with index k for whatever k satisfies $\frac{\sum_{i=1}^{k-1} w_i^{(j)}}{s^{(j)}} < r^{(j)} \leq \frac{\sum_{i=1}^k w_i^{(j)}}{s^{(j)}}$. (The event $r^{(j)} = 0$ has 0 probability—it can choose arbitrarily in that event; we assume that real numbers are available, and will not discuss approximations necessary for actual implementation of the algorithm.) If this algorithm is run and the choices of all of the $r^{(j)}$ are independent of each other and of the choices of all of the instances and labels, then the strong independence condition is satisfied.*

Proof Suppose there are t trials. Let $\nu^{(j)}$ denote the index of the pool member chosen in trial j . It suffices to show that for $1 \leq j \leq t$ and $1 \leq k \leq n$

$$P(\nu^{(j)} = k | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) = \frac{w_k^{(j)}}{s^{(j)}}$$

We prove this by induction. This clearly holds for the first trial. For other trials, we have

$$\begin{aligned} & P(\nu^{(j)} = k | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) \\ &= P(\nu^{(j)} = k \text{ and } \lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) \\ & \quad + P(\nu^{(j)} = k \text{ and } \lambda^{(j-1)} \neq \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) \end{aligned}$$

We look at these two terms separately:

$$\begin{aligned} & P(\nu^{(j)} = k \text{ and } \lambda^{(j-1)} \neq \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) \\ &= P\left(\frac{\sum_{i=1}^{k-1} w_i^{(j)}}{s^{(j)}} < r^{(j)} \leq \frac{\sum_{i=1}^k w_i^{(j)}}{s^{(j)}} \right. \\ & \quad \left. \text{and } \lambda^{(j-1)} \neq \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}\right) \end{aligned}$$

This is 0 if $\lambda^{(j-1)} = \rho^{(j-1)}$ and otherwise equals $\frac{w_k^{(j)}}{s^{(j)}}$. For the other term, we have wherever $\lambda^{(j-1)} = \rho^{(j-1)}$

$$\begin{aligned} & P(\nu^{(j)} = k \text{ and } \lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)}) \\ &= \frac{P(\nu^{(j)} = k \text{ and } \lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-2)})}{P(\lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-2)})} \end{aligned}$$

using Lemma A.1. Since a new random choice is not made when no mistake is made, this equals

$$\frac{P(\nu^{(j-1)} = k \text{ and } \lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-2)})}{P(\lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-2)})}$$

By the induction hypothesis

$$\begin{aligned}
 P(\nu^{(j-1)} = k \quad \text{and} \quad \lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-2)}) \\
 = \begin{cases} \frac{w_k^{(j-1)}}{s^{(j-1)}} & \text{if } x_k^{(j-1)} = \rho^{(j-1)} \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

This equals $\frac{w_k^{(j)}}{s^{(j-1)}}$. Also

$$\begin{aligned}
 P(\lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-2)}) \\
 = \sum_{k \text{ s.t. } x_k^{(j-1)} = \rho^{(j-1)}} \frac{w_k^{(j-1)}}{s^{(j-1)}} = \frac{s^{(j)}}{s^{(j-1)}}
 \end{aligned}$$

Thus we get

$$P(\nu^{(j)} = k \text{ and } \lambda^{(j-1)} = \rho^{(j-1)} | (x^{(1)}, \rho^{(1)}), \dots, (x^{(t)}, \rho^{(t)}), \lambda^{(1)}, \dots, \lambda^{(j-1)})$$

equals 0 where $\lambda^{(j-1)} \neq \rho^{(j-1)}$ and equals $\frac{w_k^{(j)}}{s^{(j)}}$ where $\lambda^{(j-1)} = \rho^{(j-1)}$. Putting the two terms that we started with together yields the desired result. \square