

Topic 18: The Weighted Majority Algorithm

*Lecturer: Sally Goldman**Scribe: Marc Wallace*

18.1 Introduction

In these notes we study the construction of a prediction algorithm 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. In particular, consider the case in which 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. This algorithm is shown to be robust with respect to errors in the data. We then discuss other situations in which such a compound algorithm can be applied. The material presented here comes from the paper “The Weighted Majority Algorithm,” by Nick Littlestone and Manfred Warmuth [30].

Before describing the weighted majority algorithm in detail we briefly discuss some learning problems in which the above scenario applies. The first case is one that we have seen before. Suppose one knows that the correct prediction comes from some target concept selected from some known concept class, then one can apply the weighted majority algorithm where each concept in the class is one of the algorithms in the pool. As we shall see for such situations, the weighted majority algorithm is just the natural generalization of the halving algorithm. Another situation in which the weighted majority algorithm can be used is the situation in which one has a set of algorithms for making predictions, one of which will work well on a given data set. However, it may be that the best prediction algorithm to use depends on the data set and thus it is not known a priori which algorithm to use. Finally, as we shall discuss in more detail below the weighted majority algorithm can often be applied to help in situations in which the prediction algorithm has a parameter that must be selected and the best choice for the parameter depends on the target. In such cases one can build the pool of algorithms by choosing various values for the parameter.

The basic idea behind the weighted majority algorithm is to give each algorithm in the pool the input instance, get all the predictions, combine these into one prediction, and then if wrong pass that data along to the algorithms in the pool. Observe that the halving algorithm could be considered as a weak master algorithm by simply taking the majority prediction and throwing out all bad predictors at each mistake. Unfortunately this algorithm is not robust against noise. If one algorithm is perfect, but that source is noisy, then the main algorithm will reject it and may never converge to one solution (it will run out of algorithms).

We define the number of *anomalies* in a sequence of trials as the least number of inconsistent trials of any algorithm in the pool. Notice if there are any anomalies, then the halving

algorithm will fail to converge by eliminating all functions.

18.2 Weighted Majority Algorithm

In this section we introduce the weighted majority algorithm.

Weighted Majority Algorithm (WM)

Initially assign non-negative weights (say 1) to each algorithm in the pool

To make a prediction for an instance:

Let q_0 be the total weight of algs that predict 0

Let q_1 be the total weight of algs that predict 1

Predict 0 iff $q_0 \geq q_1$

If a mistake is made, multiply the weights of the algorithms agreeing with the incorrect prediction by some fixed non-negative $\beta < 1$.

Notice that using $\beta = 0$ would yield the halving algorithm. However, it is more interesting to consider what happens as $\beta \rightarrow 1$. Suppose the pool is called F , and a given sequence of trials has m anomalies with respect to F . Then

$$\# \text{ mistakes} \leq c(\log |F| + m)$$

where c is some constant dependent on β .

In general the maximum number of mistakes will be:

- $O(\log |F| + m)$ if one algorithm in F makes at most m mistakes.
- $O(\log(|F|/k) + m)$ if each of a set of k algorithms in F makes at most m mistakes.
- $O(\log(|F|/k) + (m/k))$ if the total number of mistakes of a set of k algorithms in F is at most m mistakes.

As an application of WM (weighted majority), we return to the use of WINNOWER1 to learn r -of- k threshold functions. Normally the mistake bound is $O(kn \log n)$. However, if the learner has prior knowledge of a close upper bound for r , the mistake bound can be reduced to $b_r = O(kr \log n)$ by carefully selecting the parameters for WINNOWER1. Let A_r denote the algorithm WINNOWER1 with a guess of r . Now any sequence that cannot be represented by an r' -of- k function for $r' \leq r$ will fail phenomenally, and make far too many mistakes. However, we can apply WM to the set of algorithms $\{A_{2^i}\}$ for all i satisfying $2^i \leq n$. Then the number of mistakes will be $O(\log \log n + b_r) = O(kr \log n)$.

Now clearly if the size of our algorithm pool is not polynomial the weighted majority algorithm is not computationally feasible. Still, many reasonable pools of algorithms will be of polynomial size, or can be reduced to such without a dramatic increase in the average number of mistakes.

18.3 Analysis of Weighted Majority Algorithm

Let $A = \{A_1, \dots, A_{|A|}\}$ be the pool of algorithms, and w_i the weight of A_i . The basic structure of the proofs that follow will be: First, show that after each trial, if a mistake occurs then the sum of the weights is decreases by at least a factor of u for some $u < 1$. Second, let W_{init} be the total initial weight and W_{fin} be a lower bound for the total final weights; then $W_{init}u^m \geq W_{fin}$, where m is the number of mistakes. This implies that

$$m \leq \frac{\log \frac{W_{init}}{W_{fin}}}{\log \frac{1}{u}}.$$

We now prove a general theorem bounding the number of mistakes made by the weighted majority algorithm. We can then apply to obtain the three results given above.

Theorem 18.1 *Let S be a sequence of instances and m_i the number of mistakes made by A_i on S . Let m be the number of mistakes made by WM on S when applied to the pool A . Then*

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

where $W_{fin} = \sum_{i=1}^n w_i \beta^{m_i}$.

Notice that if the initial weights are all 1, we have

$$m \leq \frac{\log \frac{|A|}{\sum_{i=1}^n \beta^{m_i}}}{\log \frac{2}{1+\beta}}$$

Proof: We notice that the weights are updated only when their algorithm makes a mistake; and since A_i can make only m_i mistakes, we must have $w_{i_{final}} \geq w_{i_{init}} \beta^{m_i}$. Furthermore, at the end the total current weight must be greater than or equal to W_{fin} .

Now, during each trial, let $q_0 + q_1$ be the current total weight, where q_0 and q_1 are as defined in the definition of WM. Suppose we have a trial in which a mistake was made. Without loss of generality we assume the prediction was zero. Then the total weight after the trial will be:

$$\beta q_0 + q_1 \leq \beta q_0 + q_1 + \frac{1-\beta}{2}(q_0 - q_1) = \frac{1+\beta}{2}(q_0 + q_1)$$

If no mistake is made then the weights remain the same, of course. So we have $u = \frac{1+\beta}{2} < 1$.

Finally, since we know $W_{init}u^m \geq W_{fin}$ we take logs of both sides and get

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

■

Now we provide some (more useful) corollaries. Assume $\beta > 0$ and all initial weights are one. Let $|A| = n$.

Corollary 18.1 *Assume there is a subpool of A (of size k) such that each algorithm in it makes no more than m mistakes on a set of instances S . Then WM applied to A makes no more than*

$$\frac{\log \frac{n}{k} + m \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$$

mistakes on S .

Proof: $W_{init} = n$ and $W_{fin} \geq k\beta^m$. Apply the theorem. ■

The theorem and the corollary give us the first two relations stated above concerning the order of the mistake bounds: $O(\log |F| + m)$ if one algorithm in F makes at most m mistakes, and $O(\log(|F|/k) + m)$ if each of a set of k algorithms in F makes at most m mistakes. The next corollary will yield the third relation. However, before proving this corollary we review some basic definitions and lemmas.

Definition 18.1 *A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is concave (respectively convex) over an interval D of \mathbb{R} if for all $x \in D$, $f''(x) \geq 0$ ($f''(x) \leq 0$).*

The following to standard lemmas [20, 32] are often useful.

Lemma 18.1 *Let f be a function from \mathbb{R} to \mathbb{R} that is concave over some interval D of \mathbb{R} . Let q be a natural number, and let $x_1, x_2, \dots, x_q \in D$. Then*

$$\sum_{i=1}^q x_i \leq S \Rightarrow \sum_{i=1}^q f(x_i) \geq qf(S/q).$$

Lemma 18.2 *Let f be a function from \mathbb{R} to \mathbb{R} that is convex over some interval D of \mathbb{R} . Let q be a natural number, and let $x_1, x_2, \dots, x_q \in D$. Then*

$$\sum_{i=1}^q x_i \leq S \Rightarrow \sum_{i=1}^q f(x_i) \leq qf(S/q).$$

We are now ready to prove the corollary.

Corollary 18.2 *Assume there is a subpool of A (of size k) such that all algorithms (together) make no more than m mistakes on a set of instances S . Then WM applied to A makes no more than*

$$\frac{\log \frac{n}{k} + \frac{m}{k} \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$$

mistakes on S .

Proof: Without loss of generality we can assume the first k algorithms of A are the subpool in question. Then $\sum_{i=1}^k m_i \leq m$. Since the sum of the final weights is greater than the sum of k of the final weights (or equal), we have

$$w_{fin} \geq \sum_{i=1}^k w_{i_{fin}} \geq \sum_{i=1}^k \beta^{m_i}$$

Finally, from Lemma 18.1 it follows that

$$\sum_{i=1}^k \beta^{m_i} \geq k\beta^{\frac{m}{k}}$$

As a final note, if $\beta = 0$ and $m = 0$ the number of mistakes made will be no more than $\log_2 \frac{n}{k}$. ■

18.4 Variations on Weighted Majority

In this section we demonstrate the generality of the weighted majority algorithm by discussing some useful learning situation in which it can be applied.

18.4.1 Shifting Target

We modify WM so that it can recover more easily if the target is changed. For example, suppose a particular subset of algorithms predicts well in the beginning, but after some period of time some other set of algorithms has the best performance (and the initial set may yield terribly wrong predictions). We want the overall performance to remain good.

For simplicity of notation and mathematics, assume that the subpools which do well (either initially or after a while) are all of size one: it is not too difficult to generalize but the basic ideas can easily be lost in the dredge of notation.

Our new algorithm will be called WML. WML has two basic parameters, β and γ , with $0 < \beta < 1$ and $0 \leq \gamma < 1/2$. Each algorithm in A has a non-negative weight associated with it. Everything is basically the same as WM except for the updating: whenever WM would multiply a weight by β , WML will multiply by β if and only if the weight is larger than $\gamma/|A|$ times the total weight of all algorithms in A .

The idea, therefore, is to follow the same updating scheme, unless a weight is already extremely low. If it is then it is probably negligible, so do not bother updating it. This allows us to take a previously poorly performing algorithm (which has just started predicting extremely well) and put lots of weight on it, since its weight has not been allowed dropped too low.

Now suppose that some algorithm makes m_1 mistakes in the first segment of some sequence of instances, another makes m_2 mistakes in the next segment, and so on. Of course, WML does not know when the segments start and finish. Still, the number of mistakes will be no greater than

$$c(S \log |A| + \sum_{i=1}^s m_i)$$

where S is the number of segments there are and c is some constant.

This mistake bound makes sense, since the $S \log |A|$ part is really $\sum_{i=1}^S \log |A|$. When there is a change between segments, the WML will make some constant times $\log |A|$ mistakes

while the weights are updated so that the (currently) accurate algorithm receives the majority of the weight. The other m_i is from the accurate algorithms themselves, as according to the bounds for WM.

18.4.2 Selection from an Infinite Pool

Suppose we have a (countably) infinite pool of algorithms from which to choose. Clearly we cannot ask for input from all of them. But by using ever increasing sets of the algorithms one can approximate the standard WM model, with only an additional term of order $\log i$ where it is the i th algorithm which predicts well.

Previously (in homework 3, problem 1) we saw that one could apply the halving algorithm to such an infinite set, and achieve a mistake bound of order $\log i$. Since the WM is merely an extension of the halving algorithm, it is possible to construct a variant of the WM algorithm which increases its pool over time, in which the number of mistakes is bounded by $c(\log i + m_i)$, where the constant c depends not only on the initial parameters, but on how big the initial set of algorithms grows.

18.4.3 Pool of Functions

We now consider the application of WM to a pool of Boolean functions on some domain. This is a true generalization of the halving algorithm. Now, suppose there is a function consistent with the sequence of trials. We wish to obtain better bounds for the number of mistakes on this function, at the possible expense of greater numbers of mistakes on the other functions. Hence, the following theorem:

Theorem 18.2 *Let f_1, \dots, f_n be a pool of functions, and m_1, \dots, m_n be integers such that $\sum_{i=1}^n 2^{-m_i} < 2$. If WM is applied to the pool with initial weights $w_j = 2^{m_j}$ and $\beta = 0$, and if the sequence of trials is consistent with some f_i , then WM will make no more than m_i mistakes.*

Proof: Recall from the discussion of WM that for m the number of mistakes, we have

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

where $W_{fin} = \sum_{i=1}^n w_i \beta^{m_i}$, and m_i is the number of mistakes made by the i th algorithm on a given sequence.

For $\beta = 0$ the denominator drops out. Now, $W_{init} = \sum_{i=1}^n 2^{-m_i} < 2$. And since f_i makes no mistakes at all, $W_{fin} \geq$ initial weight of $f_i = 2^{-m_i}$. Thus the number of mistakes m is no more than:

$$m \leq \log W_{init} - \log W_{fin} \leq \log 2 - \log 2^{-m_i} = 1 + m_i$$

which yields the desired result. ■

18.4.4 Randomized Responses

Notice that we can consider WM to be a deterministic algorithm which predicts 1 whenever $\frac{q_1}{q_0+q_1} \geq \frac{1}{2}$.

Consider a randomized version in which 1 is predicted with probability $\frac{q_1}{q_0+q_1}$. This algorithm may make more initial mistakes when the probability is near 1/2, but it can be shown that we can update the weights so that the rate of mistakes (in the long run) can be made arbitrarily close to the rate of mistakes of the best prediction algorithm in the pool. This would yield an improvement of a factor of two over the limiting bounds for the learning rate of the deterministic version of WM.

