

Lecture 14: Decision-making under total uncertainty: the multiplicative weight algorithm

Lecturer: *Huacheng Yu*

(Today's notes below are largely lifted with minor modifications from a survey by Arora, Hazan, Kale in *Theory of Computing journal*, Volume 8 (2012), pp. 121–164.)

Today we study decision-making under total uncertainty: there is no a priori distribution on the set of possible outcomes. (This line will cause heads to explode among devout Bayesians, but it makes sense in many computer science settings. One reason is computational complexity or general lack of resources: the decision-maker usually lacks the computational power to construct the tree of all $\exp(T)$ outcomes possible in the next T steps, and the resources to do enough samples/polls/surveys to figure out their distribution. Or the algorithm designer may not be a Bayesian.)

Such decision-making (usually done with efficient algorithms) is studied in the field of *online computation*, which takes the view that the algorithm is responding to a sequence of requests that arrive one by one. The algorithm must take an action as each request arrives, and it may discover later, after seeing more requests, that its past actions were suboptimal. But past actions cannot be unchanged.

1 Motivating example: weighted majority algorithm

Now we briefly illustrate the general idea in a simple and concrete setting. This is known as the *Prediction from Expert Advice* problem.

Imagine the process of picking good times to invest in a stock. For simplicity, assume that there is a single stock of interest, and its daily price movement is modeled as a sequence of binary events: up/down. (Below, this will be generalized to allow non-binary events.) Each morning we try to predict whether the price will go up or down that day; if our prediction happens to be wrong we lose a dollar that day, and if it's correct, we lose nothing.

The stock movements can be *arbitrary* and even *adversarial*¹. To balance out this pessimistic assumption, we assume that while making our predictions, we are allowed to watch the predictions of n “experts”. These experts could be arbitrarily correlated, and they may or may not know what they are talking about.

When there is no expert at all, it is not hard to see that there is nothing nontrivial one can do: For deterministic algorithms, they can be wrong on every single day; for randomized algorithms, the best strategy is to simply randomly guess on each day.

What's the best guarantee you might hope for when there are experts? You could, for instance, ask for a prediction that is correct at every single round. But OK, this is ridiculous to ask for. Maybe all the experts are wrong, or maybe they all know nothing. The next strongest guarantee you might hope for is a prediction that is correct at every round that

¹Note that finance experts have studied stock movements for over a century and fitted all kinds of stochastic models to them. But we are doing computer science here, and we will see that this adversarial view will help us apply the same idea to a variety of other settings.

at least one expert is correct. In other words, you want to be as good as the best expert every single round. But some quick thought shows that this is again a ridiculous guarantee to hope for: maybe the experts all know nothing and just guess randomly. Then surely one of them will happen to be correct every round, but you can't possibly expect to know the right answer with no information.

Instead, what you can ask for is that *on average* over T rounds, your average performance is at least as good as the best expert's *average performance* over T rounds. That is, there may be numerous rounds where some expert is correct and you are wrong. But if there's a single expert who's correct 70% of the time, then you are also correct (almost) 70% of the time, but the rounds in which you're correct could be the same or different.

So the algorithm's goal is to limit its cumulative losses (i.e., bad predictions) to roughly the same as the *best* of these experts. At first sight this seems an impossible goal, since it is not known until the end of the sequence who the best expert was, whereas the algorithm is required to make predictions all along.

2 Warmup

As a warmup, let us first think about the special case where there is a *perfect* expert who gives the correct prediction on every single day. Given all the previous record, we know some experts have made mistakes, and they are certainly not the perfect expert. A strategy in this case is to focus on the so-far-all-correct experts, and return their majority prediction for the next day. This strategy gives $O(\log n)$ incorrect predictions overall, independent of the number of days. This is because every time we make a mistake, the majority of the so-far-all-correct expert is wrong, i.e., the number of experts who have not made any mistakes reduces by at least a half. It can only happen at most $O(\log n)$ times.

3 General best experts

The previous algorithm is too aggressive, in the sense that every time an expert makes a mistake, we remove them from consideration in the rest of the game. For general inputs, a more mild variant of this algorithm works. It maintains a *weighting* of the experts. Initially all have equal weight. As time goes on, some experts are seen as making better predictions than others, and the algorithm decreases the weight of an expert every time they make a mistake. The algorithm's prediction of up/down for each day is computed by going with the opinion of the *weighted majority* of the experts for that day.

Theorem 1. *After T steps, let $m_i^{(T)}$ be the number of mistakes of expert i and $M^{(T)}$ be the number of mistakes our algorithm has made. Then we have the following bound for every i :*

$$M^{(T)} \leq 2(1 + \eta)m_i^{(T)} + \frac{2 \ln n}{\eta}.$$

In particular, this holds for i which is the best expert, i.e. having the least $m_i^{(T)}$.

Remark: The theorem implies that roughly the algorithm's performance is not much worse than that of the best expert, except for the additive term. Unless if the best expert is

Weighted majority algorithm

Initialization: Fix an $\eta \leq \frac{1}{2}$. For each expert i , associate the weight $w_i^{(1)} := 1$.

For $t = 1, 2, \dots, T$:

1. Make the prediction that is the weighted majority of the experts' predictions based on the weights $w_1^{(t)}, \dots, w_n^{(t)}$. That is, predict “up” or “down” depending on which prediction has a higher total weight of experts advising it (breaking ties arbitrarily).
2. For every expert i who predicts wrongly, decrease his weight for the next round by multiplying it by a factor of $(1 - \eta)$:

$$w_i^{(t+1)} = (1 - \eta)w_i^{(t)} \quad (\text{update rule}). \quad (1)$$

superhuman in some way, he/she will make mistakes once in a while, and then the additive term can be ignored once the number of rounds gets large.

Proof. A simple induction shows that $w_i^{(t+1)} = (1 - \eta)^{m_i^{(t)}}$. Let $\Phi^{(t)} = \sum_i w_i^{(t)}$ (“the potential function”). Thus $\Phi^{(1)} = n$. Each time we make a mistake, the weighted majority of experts also made a mistake, so at least half the total weight decreases by a factor $1 - \eta$. Thus, the potential function decreases by a factor of at least $(1 - \eta/2)$:

$$\Phi^{(t+1)} \leq \Phi^{(t)} \left(\frac{1}{2} + \frac{1}{2}(1 - \eta) \right) = \Phi^{(t)}(1 - \eta/2).$$

Thus simple induction gives $\Phi^{(T+1)} \leq n(1 - \eta/2)^{M^{(T)}}$. Finally, since $\Phi^{(T+1)} \geq w_i^{(T+1)}$ for all i , the claimed bound follows by comparing the above two expressions

$$(1 - \eta)^{m_i^{(t)}} \leq n(1 - \eta/2)^{M^{(T)}}.$$

Upon taking logs this becomes

$$m_i^{(t)} \ln(1 - \eta) \leq M^{(T)} \ln(1 - \eta/2) + \ln n,$$

and we finish by using the fact that $\eta \leq -\ln(1 - \eta) \leq \eta + \eta^2$ when $\eta < \frac{1}{2}$. □

The beauty of this analysis is that it makes no assumption about the sequence of events: they could be arbitrarily correlated and could even depend upon our current weighting of the experts. In this sense, the algorithm delivers more than initially promised, and this lies at the root of why (after obvious generalization) it can give rise to the diverse algorithms mentioned earlier. In particular, the scenario where the events are chosen adversarially resembles a zero-sum game, which we will study in a future lecture.

3.1 Randomized version

The above algorithm is deterministic. When $m_i^{(T)} \gg \frac{2 \ln n}{\eta}$ we see from the statement of Theorem 1 that the number of mistakes made by the algorithm is bounded from above

by roughly $2(1 + \eta)m_i^{(T)}$, i.e., approximately twice the number of mistakes made by the best expert. This is tight for any deterministic algorithm (Exercise: prove this!). However, the factor of 2 can be removed by substituting the above deterministic algorithm by a randomized algorithm that predicts according to the majority opinion with probability proportional to its weight. (In other words, if the total weight of the experts saying “up” is $3/4$ then the algorithm predicts “up” with probability $3/4$ and “down” with probability $1/4$.) Then the number of mistakes after T steps is a random variable.

Theorem 2. *The expected number of mistakes made by the algorithm is the same as in Theorem 1 but without the factor 2 in both terms.*

To prove this, first note that the randomized algorithm can be restated as picking an expert i with probability proportional to its weight and using that expert’s prediction. Note that the probability of picking the expert is

$$p_i^{(t)} \stackrel{\text{def}}{=} \frac{w_i^{(t)}}{\sum_j w_j^{(t)}} = \frac{w_i^{(t)}}{\Phi^{(t)}}.$$

Now let’s slightly change notation: let $m_i^{(t)}$ be 1 if expert i makes a wrong prediction at time t and 0 else. (Thus $m_i^{(t)}$ is the *cost* incurred by this expert at that time.) Then the probability the algorithm makes a mistake at time t is simply $\sum_i p_i^{(t)} m_i^{(t)}$, which we will write as the inner product of the m and p vectors: $\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. Thus the expected number of mistakes by our algorithm at the end is

$$\sum_{t=0}^{T-1} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}.$$

Now lets compute the change in potential $\Phi^{(t)} = \sum_i w_i^{(t)}$:

$$\begin{aligned} \Phi^{(t+1)} &= \sum_i w_i^{(t+1)} \\ &= \sum_i w_i^{(t)} (1 - \eta m_i^{(t)}) \\ &= \Phi^{(t)} - \eta \Phi^{(t)} \sum_i m_i^{(t)} p_i^{(t)} \\ &= \Phi^{(t)} (1 - \eta \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \\ &\leq \Phi^{(t)} \exp(-\eta \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}). \end{aligned}$$

Note that this potential drop is not a random variable; it is a deterministic quantity that depends only on the loss vector $\mathbf{m}^{(t)}$ and the current expert weights (which in turn are determined by the loss vectors of the previous steps).

We conclude by induction that the final potential is at most

$$\prod_{t=0}^{T-1} \Phi^{(0)} \exp(-\eta \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) = \Phi^{(0)} \exp(-\eta \sum_t \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).$$

For each i this final potential is at least the final weight of the i th expert, which is

$$\prod_t (1 - \eta m_i^{(t)}) \geq (1 - \eta)^{\sum_t m_i^{(t)}}.$$

Thus taking logs and using that $\eta \leq -\ln(1 - \eta) \leq \eta(1 + \eta)$ we conclude that $\sum_{t=0}^{T-1} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$ (which is also the expected number of mistakes by our algorithm) is at most $(1 + \eta)$ times the number of mistakes by expert i , plus the same old additive factor $\ln n/\eta$.

4 The Multiplicative Weights algorithm

In the general setting, we have a choice of n *decisions* in each round, from which we are required to select one. (The precise details of the decision are not important here: think of them as just indexed from 1 to n .) In each round, each decision incurs a certain *cost*, determined by nature or an adversary. All the costs are revealed *after* we choose our decision, and we incur the cost of the decision we chose. For example, in the prediction from expert advice problem, each decision corresponds to a choice of an expert, and the cost of an expert is 1 if the expert makes a mistake, and 0 otherwise.

To motivate the Multiplicative Weights (MW) algorithm, consider the naïve strategy that, in each iteration, simply picks a decision at random. The expected penalty will be that of the “average” decision. Suppose now that a few decisions are clearly better in the long run. This is easy to spot as the costs are revealed over time, and so it is sensible to reward them by increasing their probability of being picked in the next round (hence the multiplicative weight update rule).

Intuitively, being in complete ignorance about the decisions at the outset, we select them uniformly at random. This maximum entropy starting rule reflects our ignorance. As we learn which ones are the good decisions and which ones are bad, we lower the entropy to reflect our increased knowledge. The multiplicative weight update is our means of skewing the distribution.

We now set up some notation. Let $t = 1, 2, \dots, T$ denote the current round, and let i be a generic decision. In each round t , we select a distribution $\mathbf{p}^{(t)}$ over the set of decisions, and select a decision i randomly from it. At this point, the costs of all the decisions are revealed by nature in the form of the vector $\mathbf{m}^{(t)}$ such that decision i incurs cost $m_i^{(t)}$. We assume that the costs lie in the range $[-1, 1]$. This is the only assumption we make on the costs; nature is completely free to choose the cost vector as long as these bounds are respected, even with full knowledge of the distribution that we choose our decision from.

The expected cost to the algorithm for sampling a decision i from the distribution $\mathbf{p}^{(t)}$ is

$$\mathbb{E}_{i \in \mathbf{p}^{(t)}} [m_i^{(t)}] = \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}.$$

The total expected cost over all rounds is therefore $\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. Just as before, our goal is to design an algorithm which achieves a total expected cost not too much more than the cost of the best decision in hindsight, viz. $\min_i \sum_{t=1}^T m_i^{(t)}$. Consider the following algorithm, which we call the Multiplicative Weights Algorithm. This algorithm has been studied before as the `prod` algorithm of Cesa-Bianchi, Mansour, and Stoltz.

Multiplicative Weights algorithm

Initialization: Fix an $\eta \leq \frac{1}{2}$. For each decision i , associate the weight $w_i^{(t)} := 1$.

For $t = 1, 2, \dots, T$:

1. Choose decision i with probability proportional to its weight $w_i^{(t)}$. I.e., use the distribution over decisions $\mathbf{p}^{(t)} = \{w_1^{(t)}/\Phi^{(t)}, \dots, w_n^{(t)}/\Phi^{(t)}\}$ where $\Phi^{(t)} = \sum_i w_i^{(t)}$.
2. Observe the costs of the decisions $\mathbf{m}^{(t)}$.
3. Penalize the costly decisions by updating their weights as follows: for every decision i , set

$$w_i^{(t+1)} = w_i^{(t)}(1 - \eta m_i^{(t)}) \tag{2}$$

Figure 1: The Multiplicative Weights algorithm.

The following theorem —completely analogous to Theorem 1— bounds the total expected cost of the Multiplicative Weights algorithm (given in Figure 1) in terms of the total cost of the best decision:

Theorem 3. *Assume that all costs $m_i^{(t)} \in [-1, 1]$ and $\eta \leq 1/2$. Then the Multiplicative Weights algorithm guarantees that after T rounds, for any decision i , we have*

$$\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln n}{\eta}.$$

Note that we have not addressed the optimal choice of η thus far. First, it should be small enough that all calculations in the analysis hold, say $|\eta \cdot \mathbf{m}_i^{(t)}| \leq 1/2$ for all i, t . Typically this is done by rescaling the payoffs to lie in $[-1, 1]$, which means that $\sum_{t=1}^T |m_i^{(t)}| \leq T$. Then setting $\eta \approx \sqrt{\ln n / T}$ gives the tightest upperbound on the right hand side in Theorem 3, by reducing the additive error to about $\sqrt{T \ln n}$. Of course, this is a safe choice; in practice the best η depends upon the actual sequence of events, but of course those are not known in advance.

BIBLIOGRAPHY

S. Arora, E. Hazan, S. Kale. *The multiplicative weights update method: A meta algorithm and its applications*. Theory of Computing, Volume 8 (2012), pp. 121–164.

A. Borodin and R. El Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.