

Lecture 16: Multiplicative update method

It is a useful method with a variety of applications, in particular we will see how to apply it to solve linear programs.

Introduction

Consider the simple setting of investing in stock markets (or gambling in football matches).

There are n experts say in the newspapers.

Each morning, they will predict UP/DOWN of the stock markets (or WIN/LOSE of a game).

Each evening, we will see the outcome, who are right and who are wrong, and the right experts would gain one dollar and the wrong experts would lose one dollar.

This process will repeat for many days. In the end, some experts may do very well, while some may do poorly.

Suppose you know nothing about investment, but you saw that some expert did really well, and wonder that if you followed his/her advices then you could also earn a lot of money.

But, of course, you only know who is the best expert in hindsight.

If you would like to make money, you need to make "online" decisions every day, while in the beginning you have no idea who is good.

Can you still hope to do as well as the best expert in hindsight?

Perhaps surprisingly, the multiplicative update method provides a simple strategy that guarantees to do "almost" as well as the best experts in hindsight.

Multiplicative weights update method

Before we state the method, we first state the more general setting in which the method applies.

- We have a set of n decisions.
- There are many rounds $t = 1, 2, \dots, T$.
- After each round, we learn the outcome of each decision, which is represented as a vector $m^{(t)} \in \mathbb{R}^n$.

The assumption is that each entry of $m^{(t)}$ is in the range $[-1, +1]$ (instead of just binary valued above).

We denote the i -th entry of $m^{(t)}$ as $m_i^{(t)}$, and we think of it as the "loss" of expert i in round t .

- In each round, we need to make a decision, by specifying a probability distribution $p^{(t)}$ over the n decisions (this is more general than making a decision to follow one expert, think of buying a convex combination of stock options, e.g. 0.5 dollar Google, 0.5 dollar Microsoft).

This is the setup. Our goal is to minimize the total cost, comparing to the best decision.

- After all T rounds, the total cost of decision i (or expert i) is $\sum_{t=1}^T m_i^{(t)}$.
- In each round, the cost of our decision is $\langle m^{(t)}, p^{(t)} \rangle$, and thus the total cost is $\sum_{t=1}^T \langle m^{(t)}, p^{(t)} \rangle$.

The multiplicative weight update algorithm is natural and intuitive.

We keep a weight $w_i^{(t)}$ on each expert i , representing our trust on expert i at time t .

Initially, we have no idea which decision is good, and so we just set equal weights, i.e. $w_i^{(1)} = 1$.

Then, we observe the outcome and update the weight multiplicatively by their performance, and we put higher probability on experts with larger weights. The precise algorithm is as follows.

Fix a parameter $\eta \leq \frac{1}{2}$. Set $w_i^{(1)} = 1$ for $1 \leq i \leq n$.

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

- 1). Let $\phi^{(t)} = \sum w_i^{(t)}$. Set $p_i^{(t)} = w_i^{(t)} / \phi^{(t)}$.
- 2). Observe $m^{(t)}$.
- 3). Update $w_i^{(t+1)} := w_i^{(t)} (1 - \eta m_i^{(t)})$.

The analysis is also natural. If we didn't do well in a round, then ϕ will decrease much. Then, if there is a good decision, its weight will stand out, and we will follow that decision before long.

Informally, the following theorem says that $ALG \leq (1+\epsilon)OPT + \frac{\ln n}{\epsilon}$.

Theorem Assume $m_i^{(t)} \in [-1, 1]$ and $\eta \leq \frac{1}{2}$. After T rounds, we have

$$\sum_{t=1}^T \langle m^{(t)}, p^{(t)} \rangle \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln n}{\eta} \quad \text{for all } 1 \leq i \leq n.$$

Proof Using $\phi^{(t)} = \sum_{i=1}^n w_i^{(t)}$ as a potential function, we have

$$\begin{aligned} \phi^{(t+1)} &= \sum_{i=1}^n w_i^{(t+1)} = \sum_{i=1}^n w_i^{(t)} (1 - \eta m_i^{(t)}) = \phi^{(t)} - \sum_{i=1}^n \eta w_i^{(t)} m_i^{(t)} \\ &= \phi^{(t)} - \eta \phi^{(t)} \sum_{i=1}^n p_i^{(t)} m_i^{(t)} \quad (\text{using } p_i^{(t)} = w_i^{(t)} / \phi^{(t)}) \end{aligned}$$

$$= \phi^{(t)}(1-\eta < m^{(t)}, p^{(t)} >) \\ \leq \phi^{(t)} e^{-\eta < m^{(t)}, p^{(t)} >} \quad (\text{using } 1-x \leq e^{-x}).$$

$$\text{By induction, } \phi^{(T+1)} \leq \phi^{(1)} \prod_{t=1}^T e^{-\eta < m^{(t)}, p^{(t)} >} = n e^{-\eta \sum_{t=1}^T < m^{(t)}, p^{(t)} >}.$$

$$\text{On the other hand, } \phi^{(T+1)} = \sum_{i=1}^n w_i^{(T+1)} \geq w_i^{(T+1)} = \prod_{t=1}^T (1-\eta m_i^{(t)}) \geq (1-\eta)^{\sum_{t=1}^T m_i^{(t)}} (1+\eta)^{-\sum_{t=1}^T m_i^{(t)}}.$$

where the last inequality follows from the facts that $(1-\eta x) \geq (1-\eta)^x$ for $x \in [0, 1]$ and $(1-\eta x) \geq (1+\eta)^{-x}$ for $x \in [-1, 0]$, while these facts follow from the convexity of the exponential function, and so the line connecting the two endpoints is above the curve.

$$\text{Combining, we get } n e^{-\eta \sum_{t=1}^T < m^{(t)}, p^{(t)} >} \geq (1-\eta)^{\sum_{t=1}^T m_i^{(t)}} (1+\eta)^{-\sum_{t=1}^T m_i^{(t)}}.$$

$$\begin{aligned} \text{Taking logs, we have } \ln n - \eta \sum_{t=1}^T < m^{(t)}, p^{(t)} > &\geq \sum_{t=1}^T m_i^{(t)} \ln(1-\eta) - \sum_{t=1}^T m_i^{(t)} \ln(1+\eta) \\ &\geq (-\eta - \eta^2) \sum_{t=1}^T m_i^{(t)} - (\eta - \eta^2) \sum_{t=1}^T m_i^{(t)} \\ &= -\eta \sum_{t=1}^T m_i^{(t)} - \eta^2 \sum_{t=1}^T |m_i^{(t)}|, \end{aligned}$$

where the second inequality uses the facts that $\ln(1-\eta) \geq -\eta - \eta^2$ and $\ln(1+\eta) \leq \eta - \eta^2$.

while both facts can be derived from the Taylor expansion $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$

and the assumption $\eta \leq \frac{1}{2}$ to control the error term in the former.

$$\text{Therefore, } \sum_{t=1}^T < m^{(t)}, p^{(t)} > \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln n}{\eta}. \quad \square$$

Some remarks are in order:

- The assumption $m_i^{(t)} \in [-1, 1]$ is important (where did we use it in the proof?). Intuitively, this bounds the mistake we could make in each round. If it is unbounded, we may need many rounds to recover from a mistake we made in the beginning. If $m_i^{(t)} \in [-w, w]$, we could reduce to the case $m_i^{(t)} \in [-1, 1]$ by scaling down by a factor of w , but then the additive term would become $\frac{w \ln n}{\eta}$. This parameter w is called the width, and is crucial in the analysis of multiplicative update method, and we would like w to be as small as possible.

- A corollary of the theorem is that $\sum_{t=1}^T < m^{(t)}, p^{(t)} > \leq \sum_{t=1}^T < m^{(t)} + \eta |m^{(t)}|, p > + \frac{\ln n}{\eta}$, where p is any probability distribution over the experts. This interpretation is useful but we won't need it today.

...

- We did not assume anything about the outcome, except that $m_i^{(t)} \in [-1, 1]$. The outcome could be controlled by the nature (as in the stock market example), or even controlled by an adversary, who always chooses the worst outcome for us after seeing our probability distribution. In this case, we will perform very poorly, but what we can say is that every expert will also perform poorly. Actually, this is the approach we would take in applying multiplicative update in solving LP / SDP.
- This multiplicative weights update method has applications in various areas, e.g. learning and boosting, game theory, online optimization, etc. Today, we will only use it to solve LP, and you are referred to the survey for other applications.

Solving linear program

Consider the problem of checking a system of linear inequalities is feasible or not. Recall that the optimization version can be reduced to this feasibility version by doing binary search on the objective value: $\exists x \geq 0 \quad Ax \geq b$.

We think of the constraints $x \geq 0$ as easy constraints, and the constraints $Ax \geq b$ are hard constraints.

There is an interesting reduction from this problem to the multiplicative update framework:

- Think of each constraint $A_i x \geq b_i$ as an expert. Each constraint would like itself to be the hardest constraint to be satisfied. So, they are happier if their constraint is violated by the proposed solution $x^{(t)}$. In particular, if the proposed solution (the outcome) is $x^{(t)}$, the cost of the i -th constraint is $A_i x^{(t)} - b_i$. So, the cost is smaller if the constraint is violated more.
- We, the LP solver, would like to propose a solution to satisfy all the constraints (to make them unhappy). It is difficult to deal with all the constraints simultaneously. It would be easier to deal with one constraint at a time (not necessarily an original constraint of the LP).
- The multiplicative update method provides a way to combine the constraints into one constraint. Recall that the multiplicative method will find a probability distribution over the experts (in this case, the constraints) in each round, so that in the long run the multiplicative method would do almost as well as the best expert. In our setting, the multiplicative method will have a probability distribution $p^{(t)}$ over the constraints, and it combines them into one constraint $p^{(t)} A x \geq p^{(t)} b$ (here $p^{(t)}$ is a row vector). We know that, over the long run, the total violation of these constraints would be close to the total violation of the

happiest constraint (because it will put higher probability on a happy constraint).

- So, for us, the LP solver, if we could always find a solution $x^{(t)}$ satisfying that constraint $p^{(t)}Ax \geq p^{(t)}b$, then we make this constraint unhappy and thus every constraint not very happy, and this means that all the constraints are almost feasible in the long run (and we are happy).

This is the intuition of the reduction. If you find it not natural, we can also understand it directly.

In the beginning, we just take set $p^{(1)} = \vec{1}/n$ and we find a solution $x^{(1)}$ that satisfies the constraint $p^{(1)}Ax \geq p^{(1)}b$. This solution will satisfy some constraints, but may also violate some constraint. Now, we do multiplicative weights update, by putting more weight on the constraints with larger violation. This in effect let us focus on the violated constraints more, and then we find a solution $x^{(2)}$ that satisfies $p^{(2)}Ax \geq p^{(2)}b$, and this procedure is repeated.

In the analysis, we will show that the average solution $\frac{1}{T} \sum_{t=1}^T x^{(t)}$ will almost satisfy all constraints.

One question remains:

- How to find a solution satisfying $p^{(t)}Ax \geq p^{(t)}b$? This is very easy, because we are just solving a system with two inequalities $p^{(t)}Ax \geq p^{(t)}b$ and $x \geq 0$. Indeed, this is not satisfiable if and only if $p^{(t)}A \leq 0$ and $p^{(t)}b > 0$, and in this case $p^{(t)}$ is a dual solution certifying that the original system is infeasible. So, if the original system is feasible, then such a solution must exist (indeed a feasible solution would always satisfy this constraint).

Okay, so let us see the formal details of the algorithm and the analysis.

Initially, $p^{(1)} = \vec{1}/n$.

For $t = 1$ to T do

- 1) Find a solution $x^{(t)}$ that satisfies $p^{(t)}Ax \geq p^{(t)}b$ and $|A_i x^{(t)} - b_i| \leq w$ for all i .
- 2) Set the outcome $m_i^{(t)} = \frac{A_i x^{(t)} - b_i}{w}$ for all i .
- 3) Use $m^{(t)}$ to apply the multiplicative update method to compute $p^{(t+1)}$.

Return $\bar{x} = \frac{1}{T} \sum_{t=1}^T x^{(t)}$ as our approximate solution to the LP.

The first step is often referred to as calling an ORACLE to return a solution with width at most w . If we can design an ORACLE with smaller width, the above algorithm would converge faster to a solution.

Theorem Let $\varepsilon > 0$ be a given error parameter. Suppose there is an oracle with width w .

Then, there is an algorithm which either finds a solution \bar{x} such that $A_i x \geq b_i - \varepsilon$ for all i , or correctly concludes that the system is infeasible by showing a dual solution.

The algorithm calls the oracle at most $O(w^2 \log(m)/\varepsilon^2)$ times, with additional processing time $O(m)$ per call.

Proof As we argued before, if the oracle fails at round t , then $p^{(t)}$ is a dual solution certifying the system is infeasible. So, we focus on the case when the oracle never fails.

By the theorem of the multiplicative weights update method, we have

$$\sum_{t=1}^T \langle m^{(t)}, p^{(t)} \rangle \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln m}{\eta} \quad \text{for all } i.$$

Recall that we set $m_i^{(t)} = (A_i x^{(t)} - b_i)/w$.

Since the oracle always finds $x^{(t)}$ such that $p^{(t)} A x^{(t)} \geq p^{(t)} b$, we have

$$\langle p^{(t)}, m^{(t)} \rangle = \sum_i p_i (A_i x^{(t)} - b_i) = p^{(t)} A x^{(t)} - p^{(t)} b \geq 0, \text{ and thus } \sum_{t=1}^T \langle m^{(t)}, p^{(t)} \rangle \geq 0.$$

This implies that

$$\sum_{t=1}^T \frac{A_i x^{(t)} - b_i}{w} + \eta T + \frac{\ln m}{\eta} \geq 0$$

$$\Rightarrow \sum_{t=1}^T \frac{A_i x^{(t)} - b_i}{T} \geq -\eta w - \frac{w \ln m}{\eta T}$$

Set $\eta = \frac{\varepsilon}{2w}$ so that $-\eta w = -\frac{\varepsilon}{2}$, and then set $T = \frac{4w^2 \ln m}{\varepsilon^2}$, we get $\sum_{t=1}^T \frac{A_i x^{(t)} - b_i}{T} \geq -\varepsilon$.

Now, by setting $\bar{x} = \frac{1}{T} \sum_{t=1}^T x^{(t)}$, we have $A_i \bar{x} - b_i \geq -\varepsilon$. This holds for all i , and we're done. \square

Again, designing an oracle of small width is crucial. For LP coming from combinatorial problems, we usually have a straightforward bound that $w = O(n)$, and thus it is a polynomial time method providing that ε is not too small. Note that ε is the additive error on the objective value, and usually we just set it to be a small constant for combinatorial problems.

References

The material is from the survey "The multiplicative weight update method : a meta algorithm and applications",
by Arora, Kale and Hazen. See the survey for more applications.