CS 466/666 Algorithm Design and Analysis . Spring 2020

Lecture 21: Multiplicative weight update method

It is a useful method with various applications, including online optimization.

We will see one interesting application in solving linear programs.

Online decisions

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

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

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

Can we 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 weight update method

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

- We have a set of n decisions.
- There are many rounds $t = 1, 2, \ldots, T$.
- 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 value 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 (which is more flexible than making a decision to follow one expert, think of buying a convex combination of stock options, e.g. $\frac{1}{2}$ dollar Google, $\frac{1}{2}$ dollar Microsoft).

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

- 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 quite 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 on which expert is good, and so we just set equal weights, i.e. $w_i^{(1)} = 1 \ \forall i$.

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.

Algorithm

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

For $1 \le t \le T$ do

1). Let $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$. 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 $\text{ALG} \le (1+\varepsilon) \cdot \text{OPT} + \frac{\ln n}{\varepsilon}$.

Theorem Assume $m_i^{(t)} \in [-1, +1]$ and $\eta \le \frac{1}{2}$. After T rounds, it holds that

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

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

$$\begin{aligned}
\Phi^{(t+1)} = \sum_{i=1}^{n} w_i^{(t+1)} = \sum_{i=1}^{n} w_i^{(t)} (1 - \eta m_i^{(t)}) &= \Phi^{(t)} - \sum_{i=1}^{n} \eta w_i^{(t)} m_i^{(t)} \\
&= \Phi^{(t)} - \eta \Phi^{(t)} \sum_{i=1}^{n} p_i^{(t)} m_i^{(t)} \quad \text{// using } p_i^{(t)} = w_i^{(t)} / \Phi^{(t)} \\
&= \Phi^{(t)} (1 - \eta \langle m^{(t)}, p^{(t)} \rangle) \\
&\le \Phi^{(t)} e^{-\eta \langle m^{(t)}, p^{(t)} \rangle} \quad \text{// using } 1 - x \le e^{-x}
\end{aligned}$$

By induction, $\Phi^{(T+1)} \le \Phi^{(1)} \prod_{t=1}^{T} e^{-\eta \langle m^{(t)}, p^{(t)} \rangle} = n e^{-\eta \sum_{t=1}^{T} \langle m^{(t)}, p^{(t)} \rangle}$.

So, if we don't do well, the potential function decreases.

On the other hand, $\Phi^{(T+1)} = \sum_{i=1}^{n} w_i^{(T+1)} \ge w_i^{(T+1)} = \prod_{t=1}^{T} (1 - \eta m_i^{(t)}) \ge (1-\eta)^{\sum_{\ge 0} m_i^{(t)}} (1+\eta)^{-\sum_{<0} m_i^{(t)}}$.

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.

Combining, we get $n e^{-\eta \sum_{t=1}^{T} \langle m^{(t)}, p^{(t)} \rangle} \geq (1-\eta)^{\sum_{\geq 0} m_i^{(t)}} (1+\eta)^{-\sum_{<0} m_i^{(t)}}$.

Taking logs, we have

$$\begin{aligned} \ln n - \eta \sum_{t=1}^{T} \langle m^{(t)}, p^{(t)} \rangle &\geq \sum_{\geq 0} m_i^{(t)} \ln(1-\eta) - \sum_{<0} m_i^{(t)} \ln(1+\eta) \\ &\geq (-\eta - \eta^2) \sum_{\geq 0} m_i^{(t)} - (\eta - \eta^2) \sum_{<0} m_i^{(t)} \\ &= -\eta \sum_{t=1}^{T} m_i^{(t)} - \eta^2 \sum_{t=1}^{T} |m_i^{(t)}|, \end{aligned}$$

where the second inequality uses the facts that $\ln(1-\eta) \geq -\eta - \eta^2$ and $\ln(1+\eta) \geq \eta - \eta^2$, while both facts can be derived from the Taylor expansion $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$.

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

Therefore, $\sum_{t=1}^{T} \langle m^{(t)}, p^{(t)} \rangle \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}| + \frac{\ln n}{\eta}$. □

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 will 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 it to be as small as possible.

- We did not assume anything about the outcome, except that $m_i^{(t)} \in [-1,+1]$. The outcome could be controlled by 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.

- This multiplicative update method has applications in various areas, e.g. learning and boosting, game theory, online optimization, etc. Today, we only use it to solve linear programs, and you are referred to the survey for other applications.

Solving linear programs

A linear program is of the form $\min \langle c,x \rangle$ s.t. $Ax \geq b$, where $A \in \mathbb{R}^{m \times n}$, $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$.

This is an important problem that captures many interesting problems, and it can be solved in polynomial time, using the ellipsoid method or the interior point method.

We show how to use the multiplicative weight update method to solve linear programs.
This method is not as competitive as the other methods in general, but it offers some advantages for some combinatorial problems, and we will see one example next time.

To apply the multiplicative update method, we assume that the linear program is given in the feasibility version. $Ax \geq b$ and $x \geq 0$.
Note that the optimization version can be reduced to the feasibility version by binary search.
We think of the constraints $x \geq 0$ as easy constraints, and the constraints $Ax \geq b$ as the hard constraints.

Reduction

There is an interesting reduction from this problem to the online expert problem.

1). Think of each constraint $A_i x \geq b_i$ as an expert, where A_i denotes the i-th row of A.
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 $x^{(t)}$.
Specifically, if the proposed solution (the outcome in the t-th step) is $x^{(t)}$, then we define the cost of the i-th constraint is $A_i x^{(t)} - b$. So, the cost is smaller if it is violated more.

2). 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 much easier to deal with only one constraint (not necessarily an original constraint of the LP).

3). The multiplicative weight 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 $p^{(t)} A x \geq p^{(t)} b$, where $p^{(t)}$ is a row vector
We know that, over the long run, the total violation of these constraints will be close to the total violation of the happiest constraint (as it will put higher probability on a happier constraint).

4) So, for us, if we can always find a solution $x^{(t)}$ satisfying the constraint $p^{(t)} A x \geq p^{(t)} b$,

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 by the solution $\frac{1}{T}\sum_{t=1}^{T} x^{(t)}$, and we are happy.

Direct algorithm

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

In the beginning, we just set $p^{(1)} = \vec{1}/n$, and we find a solution that satisfies the constraint $p^{(1)}Ax \geq p^{(1)}b$.

The solution may satisfy some constraints, but may violate some constraints.

Now, we do multiplicative weight update, by putting more weight on the constraints with larger violation.

This let us focus more on the violated constraints, and then we find a solution $x^{(2)}$ that satisfies $p^{(2)}Ax \geq p^{(2)}b$, and this process is repeated T steps.

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

Feasibility

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 the original linear program is not solvable (and $p^{(t)}$ is a dual solution certifying that the system is infeasible).

So, if the original system is feasible, then such a solution must exist (as a feasible solution will always satisfy this constraint).

Oracle and width

Given a constraint $p^{(t)}Ax \geq p^{(t)}b$, we assume that there is an "oracle" that will return a solution $x^{(t)}$ such that $p^{(t)}Ax^{(t)} \geq p^{(t)}b$ and $x^{(t)} \geq 0$.

An important quantity of the oracle is the <u>width</u>.

We say an oracle is of width w if the returned solution $x^{(t)}$ always satisfies $|A_i x^{(t)} - b| \leq w$ for all $1 \leq i \leq m$ where A_i is the i-th row of A.

The runtime of the method is heavily dependent on the width, as it corresponds to the lost bound $m_i^{(t)} \in [-w, +w]$ in the online expert problem.

In the following, we assume that there is an oracle of width w to solve $p^{(t)}Ax \geq p^{(t)}b$.

Algorithm

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

For $t=1$ to T do

1) Use the oracle to find a solution $x^{(t)}$ that satisfies $p^{(t)}Ax \geq p^{(t)}b$, $x \geq 0$,

and furthermore $|A_i x^{(t)} - b_i| \leq w$ 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.

Theorem Let $\varepsilon > 0$ be a given error parameter. Suppose that there is an oracle with width w. Then, there is an algorithm which either finds a solution $\bar{x}$ such that $A_i\bar{x} \geq b_i - \varepsilon$ for all i, or 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 $O(m+n)$ per call.

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

By the theorem of the multiplicative weight 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 $m_i^{(t)} = \left(A_i x^{(t)} - b_i\right)/w$.

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

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

(That is, the multiplicative update constraint is never happy.)

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

Again, designing an oracle of small width is crucial.

It also depends on our choice of easy constraints (that the oracle needs to satisfy) and hard constraints (that the multiplicative update method combines).

For LP coming from combinatorial problems, we usually have a straightforward bound that $w = O(n)$.

and thus it is a polynomial time method provided 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, Hazan, and Kale. See it for more applications.