Lecture 13: Multiplicative Weights Update

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

June 22, 2021

Overview

- Multiplicative Weights Update
- Solving Linear Programs
- Conclusion
- Acknowledgements

Learning from Experts

- **Setup:** investing your co-op money on stock markets (or gambling).
- **Objective:** to get rich, but we don't know much about stock markets
- Have access to n experts (news programs, newspapers, social media)

Learning from Experts

- **Setup:** investing your co-op money on stock markets (or gambling).
- **Objective:** to get rich, but we don't know much about stock markets
- Have access to n experts (news programs, newspapers, social media)
  - Each morning, before market opens, experts predict whether the price of a stock will go up or down
  - By the time market closes, we can check the outcome, who was right or wrong.

Learning from Experts

- **Setup:** investing your co-op money on stock markets (or gambling).
- **Objective:** to get rich, but we don't know much about stock markets
- Have access to n experts (news programs, newspapers, social media)
  - Each morning, before market opens, experts predict whether the price of a stock will go up or down
  - By the time market closes, we can check the outcome, who was right or wrong.
    - Experts who were right earn one dollar
    - Experts who were wrong lose one dollar

Learning from Experts

- **Setup:** investing your co-op money on stock markets (or gambling).
- **Objective:** to get rich, but we don't know much about stock markets
- Have access to n experts (news programs, newspapers, social media)
  - Each morning, before market opens, experts predict whether the price of a stock will go up or down
  - By the time market closes, we can check the outcome, who was right or wrong.
    - Experts who were right earn one dollar
    - Experts who were wrong lose one dollar
- Some expert did really well, and if we followed their advice we would have made a lot of money...

Learning from Experts

- **Setup:** investing your co-op money on stock markets (or gambling).
- **Objective:** to get rich, but we don't know much about stock markets
- Have access to n experts (news programs, newspapers, social media)
  - Each morning, before market opens, experts predict whether the price of a stock will go up or down
  - By the time market closes, we can check the outcome, who was right or wrong.
    - Experts who were right earn one dollar
    - Experts who were wrong lose one dollar
- Some expert did really well, and if we followed their advice we would have made a lot of money...
- Hindsight is 20/20 though. To make money, need to make a decision on what & how to trade every day.

Learning from Experts

- **Setup:** investing your co-op money on stock markets (or gambling).
- **Objective:** to get rich, but we don't know much about stock markets
- Have access to n experts (news programs, newspapers, social media)
  - Each morning, before market opens, experts predict whether the price of a stock will go up or down
  - By the time market closes, we can check the outcome, who was right or wrong.
    - Experts who were right earn one dollar
    - Experts who were wrong lose one dollar
- Some expert did really well, and if we followed their advice we would have made a lot of money...
- Hindsight is 20/20 though. To make money, need to make a decision on what & how to trade every day.
- Can we hope to do *as well as the best expert* in hindsight?

Other Applications

- Online Learning

Other Applications

- Online Learning
  1. Experts are weak classifiers, want to choose hypothesis based on these experts

Boosting (in learning theory)

Other Applications

- Online Learning
  1. Experts are weak classifiers, want to choose hypothesis based on these experts

     Boosting (in learning theory)
- Solving linear programs! (today)

Other Applications

- Online Learning
  1. Experts are weak classifiers, want to choose hypothesis based on these experts

     Boosting (in learning theory)
- Solving linear programs! (today)
- Game Theory
- many more

Why this Benchmark?

Say we are trading for T days.

Why not want our algorithm to make as much money as a function of T as we can?

Why this Benchmark?

Say we are trading for T days.

Why not want our algorithm to make as much money as a function of T as we can?

- With the benchmark above, guessing correctly (in expectation) $T/2$ times is *trivial* (pick your trades at random)

Why this Benchmark?

Say we are trading for T days.

Why not want our algorithm to make as much money as a function of T as we can?

- With the benchmark above, guessing correctly (in expectation) $T/2$ times is *trivial* (pick your trades at random)
- It turns out, $T/2$ correct guesses (in expectation) is also *optimal*

Why this Benchmark?

Say we are trading for T days.

Why not want our algorithm to make as much money as a function of T as we can?

- With the benchmark above, guessing correctly (in expectation) $T/2$ times is *trivial* (pick your trades at random)
- It turns out, $T/2$ correct guesses (in expectation) is also *optimal*
  - Worst-case analysis.

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*.
What should we do?

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*.
What should we do?

- At each trading day, take *majority vote* of the opinions of the experts.

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*.
What should we do?

- At each trading day, take *majority vote* of the opinions of the experts.
- If we made the right trade, do nothing.

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*.
What should we do?

- At each trading day, take *majority vote* of the opinions of the experts.
- If we made the right trade, do nothing.
- If our trade was bad, at the end of the trading day *discard* all experts that *made a mistake that day*.

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*. What should we do?

- At each trading day, take *majority vote* of the opinions of the experts.
- If we made the right trade, do nothing.
- If our trade was bad, at the end of the trading day *discard* all experts that *made a mistake that day*.

1. Every time we made a bad trade, we discard half of the experts.
    - We took majority vote, so at least half the experts also made bad trades

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*. What should we do?

- At each trading day, take *majority vote* of the opinions of the experts.
- If we made the right trade, do nothing.
- If our trade was bad, at the end of the trading day *discard* all experts that *made a mistake that day*.

1. Every time we made a bad trade, we discard half of the experts.
    - We took majority vote, so at least half the experts also made bad trades
2. After $\log n$ bad trades, only the expert who is always right will remain! From then on, we will always be right!

Warm-up Idea

Say we knew that there was *one expert* which will be *right every time*. What should we do?

- At each trading day, take *majority vote* of the opinions of the experts.
- If we made the right trade, do nothing.
- If our trade was bad, at the end of the trading day *discard* all experts that *made a mistake that day*.

1. Every time we made a bad trade, we discard half of the experts.
    - We took majority vote, so at least half the experts also made bad trades
2. After $\log n$ bad trades, only the expert who is always right will remain! From then on, we will always be right!

Total money we made: $\geq T - \log n$

Total money best expert made: T

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.
- Algorithm from previous slide *not robust* to having "almost perfect" experts (say experts that make one mistake)

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.
- Algorithm from previous slide *not robust* to having "almost perfect" experts (say experts that make one mistake)
- There is a way of making previous idea robust:

  Whenever an expert makes a mistake, "consider their opinions with less importance."

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.
- Algorithm from previous slide *not robust* to having "almost perfect" experts (say experts that make one mistake)
- There is a way of making previous idea robust:

  Whenever an expert makes a mistake, "consider their opinions with less importance."

1. Let $w_t : [n] \to \mathbb{R}_+$ be a function from each expert to the non-negative reals, and $0 < \varepsilon < 1$
   - $w_t(i)$ is the *weight* of expert i at time t

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.
- Algorithm from previous slide *not robust* to having "almost perfect" experts (say experts that make one mistake)
- There is a way of making previous idea robust:

  Whenever an expert makes a mistake, "consider their opinions with less importance."

1. Let $w_t : [n] \to \mathbb{R}_+$ be a function from each expert to the non-negative reals, and $0 < \varepsilon < 1$
   - $w_t(i)$ is the *weight* of expert i at time t
2. In beginning every expert has weight $w_1(i) = 1$

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.
- Algorithm from previous slide *not robust* to having "almost perfect" experts (say experts that make one mistake)
- There is a way of making previous idea robust:

  Whenever an expert makes a mistake, "consider their opinions with less importance."

1. Let $w_t : [n] \to \mathbb{R}_+$ be a function from each expert to the non-negative reals, and $0 < \varepsilon < 1$
   - $w_t(i)$ is the *weight* of expert i at time t
2. In beginning every expert has weight $w_1(i) = 1$
3. If an expert makes a mistake at day t, make $w_{t+1}(i) = w_t(i) \cdot (1 - \varepsilon)$

decreasing weight

Multiplicative Weights Update Algorithm

- In general do not have a single expert who is right all the time.
- Algorithm from previous slide *not robust* to having "almost perfect" experts (say experts that make one mistake)
- There is a way of making previous idea robust:

  Whenever an expert makes a mistake, "consider their opinions with less importance."

1. Let $w_t : [n] \to \mathbb{R}_+$ be a function from each expert to the non-negative reals, and $0 < \varepsilon < 1$
   - $w_t(i)$ is the *weight* of expert i at time t
2. In beginning every expert has weight $w_1(i) = 1$
3. If an expert makes a mistake at day t, make $w_{t+1}(i) = w_t(i) \cdot (1 - \varepsilon)$
4. Each trading day, choose to trade based on *weighted majority* of the decisions of the experts

Multiplicative Weights Update Algorithm

Algorithm:

Multiplicative Weights Update Algorithm

Algorithm:

1. **Setup:** we have a binary decision to make (i.e., $\{-1, +1\}$) and we have access to n experts, indexed by the set $[n]$.

$= \{1, 2, \ldots, n\}$

Multiplicative Weights Update Algorithm

Algorithm:

1. **Setup:** we have a binary decision to make (i.e., $\{-1, +1\}$) and we have access to n experts, indexed by the set $[n]$.
   - At each time step t, expert takes decision $d_t(i) \in \{-1, +1\}$
   - Parameter $0 < \varepsilon < 1/2$

Multiplicative Weights Update Algorithm

Algorithm:

1. **Setup:** we have a binary decision to make (i.e., $\{-1, +1\}$) and we have access to n experts, indexed by the set $[n]$.
   - At each time step t, expert takes decision $d_t(i) \in \{-1, +1\}$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $w_t : [n] \to \mathbb{R}_+$ weight function
   - $w_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $w_1(i) = 1$

Multiplicative Weights Update Algorithm

Algorithm:

1. **Setup:** we have a binary decision to make (i.e., $\{-1, +1\}$) and we have access to n experts, indexed by the set $[n]$.
   - At each time step t, expert takes decision $d_t(i) \in \{-1, +1\}$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $w_t : [n] \to \mathbb{R}_+$ weight function
   - $w_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $w_1(i) = 1$
3. At each time step (i.e. for $t = 1, \ldots, T$):

Multiplicative Weights Update Algorithm

Algorithm:

1. **Setup:** we have a binary decision to make (i.e., $\{-1, +1\}$) and we have access to n experts, indexed by the set $[n]$.
   - At each time step t, expert takes decision $d_t(i) \in \{-1, +1\}$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $w_t : [n] \to \mathbb{R}_+$ weight function
   - $w_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $w_1(i) = 1$
3. At each time step (i.e. for $t = 1, \ldots, T$):
   1. Make your decision based on *weighted majority*:

$$\begin{cases} +1, & \text{if } \sum_{i=1}^{n} w_t(i) \cdot d_t(i) \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

$$\sum_{i=1}^{n} w_t(i) \cdot d_t(i) = \sum_{i\text{'s that say } +1} w_t(i) - \sum_{i\text{'s that say } -1} w_t(i)$$

Multiplicative Weights Update Algorithm

Algorithm:

1. **Setup:** we have a binary decision to make (i.e., $\{-1, +1\}$) and we have access to n experts, indexed by the set $[n]$.
   - At each time step t, expert takes decision $d_t(i) \in \{-1, +1\}$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $w_t : [n] \to \mathbb{R}_+$ weight function
   - $w_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $w_1(i) = 1$
3. At each time step (i.e. for $t = 1, \ldots, T$):
   1. Make your decision based on *weighted majority*:

$$\begin{cases} +1, & \text{if } \sum_{i=1}^{n} w_t(i) \cdot d_t(i) \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

   2. If an expert makes a mistake at time t, make

$$w_{t+1}(i) = w_t(i) \cdot (1 - \varepsilon)$$

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1+\varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

the number of mistakes of MWU algorithm $\leq 2(1+\epsilon)$ (min # mistakes one expert makes) $+ \frac{2 \log n}{\epsilon}$

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1 + \varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function

$$\Phi_t = \sum_{i=1}^{n} w_t(i)$$

potential function at time t

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1+\varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function

$$\Phi_t = \sum_{i=1}^{n} w_t(i)$$

- **Intuition:**

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1 + \varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function

$$\Phi_t = \sum_{i=1}^{n} w_t(i)$$

- **Intuition:**
  - If we make mistake, Φ_{t+1} decreases by a multiplicative factor w.r.t. Φ_t

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1+\varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function

$$\Phi_t = \sum_{i=1}^{n} w_t(i)$$

- **Intuition:**
  - If we make mistake, Φ_{t+1} decreases by a multiplicative factor w.r.t. Φ_t
  - Φ_t is monotonically decreasing (so if we get it right, potential does not increase either)

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1+\varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function

$$\Phi_t = \sum_{i=1}^{n} w_t(i)$$

- **Intuition:**
  - If we make mistake, Φ_{t+1} decreases by a multiplicative factor w.r.t. Φ_t
  - Φ_t is monotonically decreasing (so if we get it right, potential does not increase either)
  - Initially $\Phi_1 = n$
  - $\Phi_t \geq 0$ for all t

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1 + \varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1+\varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function $\Phi_t = \sum_{i=1}^{n} w_t(i)$

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1 + \varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function $\Phi_t = \sum_{i=1}^{n} w_t(i)$
- If we made a mistake, at least half the weight was on the wrong answer. Thus

$$\Phi_{t+1} = \sum_{i \text{ right}} w_t(i) + (1 - \varepsilon) \cdot \sum_{j \text{ wrong}} w_t(j) \leq \left(1 - \frac{\varepsilon}{2}\right) \cdot \Phi_t$$

$= \Phi_t - \epsilon \sum_{j \text{ wrong}} w_t(j) \leq \Phi_t - \frac{\epsilon}{2} \Phi_t$

$\sum_{i=1}^{n} w_{t+1}(i)$

$w_{t+1}(i)$

$\sum_{j \text{ wrong}} w_t(j) \geq \sum_{j \text{ right}} w_t(j) \Rightarrow 2 \cdot \sum_{j \text{ wrong}} w_t(j) \geq \sum_{\text{wrong}} + \sum_{\text{right}} = \Phi_t$

Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $M_t(i)$ be the number of mistakes that expert i made until time t. Then, for any expert $i \in [n]$, we have:

$$M_t \leq 2 \cdot (1+\varepsilon) M_t(i) + \frac{2 \log n}{\varepsilon}$$

- Potential function $\Phi_t = \sum_{i=1}^n w_t(i)$
- If we made a mistake, at least half the weight was on the wrong answer. Thus

$$\Phi_{t+1} = \sum_{i \text{ right}} w_t(i) + (1-\varepsilon) \cdot \sum_{j \text{ wrong}} w_t(j) \leq \left(1 - \frac{\varepsilon}{2}\right) \cdot \Phi_t$$

- Thus,

$$\Phi_t \leq \Phi_1 \cdot \left(1 - \frac{\varepsilon}{2}\right)^{M_t} = n \cdot \left(1 - \frac{\varepsilon}{2}\right)^{M_t}$$

Analysis

1. We have

$$\Phi_t \leq n \cdot \left(1 - \frac{\varepsilon}{2}\right)^{M_t}$$

Analysis

1. We have
$$\Phi_t \leq n \cdot \left(1 - \frac{\varepsilon}{2}\right)^{M_t}$$
2. On the other hand, have:
$$\Phi_t = \sum_{j=1}^{n} w_t(j) > w_t(i) = (1 - \varepsilon)^{M_t(i)}$$

definition

by algorithm

Analysis

1. We have

$$\Phi_t \leq n \cdot \left(1 - \frac{\varepsilon}{2}\right)^{M_t}$$

2. On the other hand, have:

$$\Phi_t = \sum_{j=1}^{n} w_t(j) > w_t(i) = (1 - \varepsilon)^{M_t(i)}$$

3. Putting (1) and (2) together

$$n \cdot \left(1 - \frac{\varepsilon}{2}\right)^{M_t} > (1-\varepsilon)^{M_t(i)} \Rightarrow \log(1-\varepsilon/2) \cdot M_t + \log n > M_t(i) \cdot \log(1-\varepsilon)$$

$\geq \Phi_t > w_t(i)$

Analysis

$$\frac{1}{\log\left(\frac{1}{1-\varepsilon/2}\right)}\left(M_t(i)\cdot\log\left(\frac{1}{1-\varepsilon}\right)+\log n\right) > M_t$$

↑

3. Putting (1) and (2) together

$$n\cdot\left(1-\frac{\varepsilon}{2}\right)^{M_t} > (1-\varepsilon)^{M_t(i)} \Rightarrow \log(1-\varepsilon/2)\cdot M_t+\log n > M_t(i)\cdot\log(1-\varepsilon)$$

4. Using inequality $-x-x^2 < \log(1-x) < -x$ for $x \in (0, 1/2)$, we get:

$$-\varepsilon/2\cdot M_t + \log n > M_t(i)\cdot(-\varepsilon-\varepsilon^2)$$

Multiplicative Weights Update - General

The same algorithm and argument above, applied to the setting:

Multiplicative Weights Update - General

The same algorithm and argument above, applied to the setting:

1. **Setup:** have access to n experts, indexed by the set $[n]$.

Multiplicative Weights Update - General

The same algorithm and argument above, applied to the setting:

1. **Setup:** have access to n experts, indexed by the set $[n]$.
    - At each time step t, each expert will guess a value $m_t(i) \in [-1, +1]$.
    - Cost of i^{th} expert answer at time t is $m_t(i)$
    - Parameter $0 < \varepsilon < 1/2$

Multiplicative Weights Update - General

The same algorithm and argument above, applied to the setting:

1. **Setup:** have access to n experts, indexed by the set $[n]$.
   - At each time step t, each expert will guess a value $m_t(i) \in [-1, +1]$.
   - Cost of i^{th} expert answer at time t is $m_t(i)$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $p_t : [n] \to \mathbb{R}_+$ weight function (normalized to sum to 1)
   - $p_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $p_1(i) = 1/n$
   - Our total cost is $\sum_t p_t \cdot m_t$

Multiplicative Weights Update - General

The same algorithm and argument above, applied to the setting:

1. **Setup:** have access to n experts, indexed by the set $[n]$.
   - At each time step t, each expert will guess a value $m_t(i) \in [-1, +1]$.
   - Cost of i^{th} expert answer at time t is $m_t(i)$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $p_t : [n] \to \mathbb{R}_+$ weight function (normalized to sum to 1)
   - $p_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $p_1(i) = 1/n$
   - Our total cost is $\sum_t p_t \cdot m_t$
3. Our goal is to minimize our total cost: $\sum_{t=1}^{T} p_t \cdot m_t$

Multiplicative Weights Update - General

The same algorithm and argument above, applied to the setting:

1. **Setup:** have access to n experts, indexed by the set $[n]$.
   - At each time step t, each expert will guess a value $m_t(i) \in [-1, +1]$.
   - Cost of i^{th} expert answer at time t is $m_t(i)$
   - Parameter $0 < \varepsilon < 1/2$
2. Let $p_t : [n] \to \mathbb{R}_+$ weight function (normalized to sum to 1)
   - $p_t(i)$ is the *weight* of expert i at time t
   - In beginning every expert has weight $p_1(i) = 1/n$
   - Our total cost is $\sum_t p_t \cdot m_t$
3. Our goal is to minimize our total cost: $\sum_{t=1}^{T} p_t \cdot m_t$

Theorem (Multiplicative Weights Update)

With the setup above, after t rounds, for any expert $i \in [n]$, we have:

our cost — cost of expert i

$$\sum_{t=1}^{T} p_t \cdot m_t \leq \sum_{t=1}^{T} m_t(i) + \varepsilon \cdot \sum_{t=1}^{T} |m_t(i)| + \frac{\ln n}{\varepsilon}$$

- Multiplicative Weights Update
- Solving Linear Programs
- Conclusion
- Acknowledgements

Solving Linear Programs

Assume we are given LP in feasibility version:

$$\begin{aligned} Ax &\geq b \\ x &\geq 0 \end{aligned}$$

Solving Linear Programs

Assume we are given LP in feasibility version:

$$Ax \geq b$$
$$x \geq 0$$

- Optimization version reduces to feasibility version by binary search.

min $c^T x$

guess $c^T x \geq \alpha$

$$\begin{pmatrix} A \\ c^T \end{pmatrix} x \geq \begin{pmatrix} b \\ \alpha \end{pmatrix}$$

Solving Linear Programs

Assume we are given LP in feasibility version:

$$Ax \geq b$$
$$x \geq 0$$

- Optimization version reduces to feasibility version by binary search.
- Think of $x \geq 0$ being the easy constraints to satisfy, whereas $Ax \geq b$ are the hard ones

Solving Linear Programs

Assume we are given LP in feasibility version:

$$Ax \geq b$$
$$x \geq 0$$

- Optimization version reduces to feasibility version by binary search.
- Think of $x \geq 0$ being the easy constraints to satisfy, whereas $Ax \geq b$ are the hard ones

Idea:

1. Think of each inequality $A_i x \geq b_i$ as an *expert* (A_i is i^{th} row of A)

$$A = \begin{pmatrix} - A_1 - \\ - A_2 - \\ \vdots \\ - A_n - \end{pmatrix}$$

Solving Linear Programs

Assume we are given LP in feasibility version:

$$Ax \geq b$$
$$x \geq 0$$

- Optimization version reduces to feasibility version by binary search.
- Think of $x \geq 0$ being the easy constraints to satisfy, whereas $Ax \geq b$ are the hard ones

Idea:

1. Think of each inequality $A_i x \geq b_i$ as an *expert* (A_i is i^{th} row of A)
2. Each constraint would like to be *the hardest constraint*, i.e. the one that is violated the most by the current proposed solution $x^{(t)}$

Solving Linear Programs

Assume we are given LP in feasibility version:

$$Ax \geq b$$
$$x \geq 0$$

- Optimization version reduces to feasibility version by binary search.
- Think of $x \geq 0$ being the easy constraints to satisfy, whereas $Ax \geq b$ are the hard ones

Idea:

1. Think of each inequality $A_i x \geq b_i$ as an *expert* (A_i is i^{th} row of A)
2. Each constraint would like to be *the hardest constraint*, i.e. the one that is violated the most by the current proposed solution $x^{(t)}$
3. More precisely: cost of i^{th} constraint

$$A_i x - b_i$$

Solving Linear Programs

Assume we are given LP in feasibility version:

$$Ax \geq b$$
$$x \geq 0$$

- Optimization version reduces to feasibility version by binary search.
- Think of $x \geq 0$ being the easy constraints to satisfy, whereas $Ax \geq b$ are the hard ones

Idea:

1. Think of each inequality $A_i x \geq b_i$ as an *expert* (A_i is i^{th} row of A)
2. Each constraint would like to be *the hardest constraint*, i.e. the one that is violated the most by the current proposed solution $x^{(t)}$
3. More precisely: cost of i^{th} constraint

$$A_i x - b_i$$

4. We would like to propose feasible solution (i.e. lower cost of *all constraints*). Hard to deal with all constraints at the same time.

Solving Linear Programs

- Would like to minimize

$$\min_{1 \le i \le m} A_i x - b_i$$

- Multiplicative Weights Update (MWU) provides way of combining *all constraints* into *one constraint*!

Solving Linear Programs

- Would like to minimize

$$\min_{1 \le i \le m} A_i x - b_i$$

- Multiplicative Weights Update (MWU) provides way of combining *all constraints* into *one constraint*!
- MWU finds probability distribution over experts (normalized weights), which in our case are the inequalities. constraints

Solving Linear Programs

- Would like to minimize

$$\min_{1 \leq i \leq m} A_i x - b_i$$

- Multiplicative Weights Update (MWU) provides way of combining *all constraints* into *one constraint*!
- MWU finds probability distribution over experts (normalized weights), which in our case are the inequalities.
- Thus, we have to deal with only the constraint $p^{(t)} A x \geq p^{(t)} b$, where

$$p^{(t)} = \frac{1}{\sum_i w_t(i)} \cdot (w_t(1), \ldots, w_t(n))$$

$x \geq 0$ (easy constraints)

normalized weights

$p^{(t)} \geq 0$, $Ax \geq b$ $\Rightarrow$ $p^{(t)} A x \geq p^{(t)} b$

Solving Linear Programs

- Would like to minimize

$$\min_{1 \leq i \leq m} A_i x - b_i$$

- Multiplicative Weights Update (MWU) provides way of combining *all constraints* into *one constraint*!
- MWU finds probability distribution over experts (normalized weights), which in our case are the inequalities.
- Thus, we have to deal with only the constraint $p^{(t)} A x \geq p^{(t)} b$, where

$$p^{(t)} = \frac{1}{\sum_i w_t(i)} \cdot (w_t(1), \ldots, w_t(n))$$

- MWU shows that over the long run:

The *total violation* of our *weighted constraints* will be close to the *total violation* of the *worst violated constraint*!

Solving Linear Programs

- Would like to minimize

$$\min_{1 \le i \le m} \underbrace{A_i x - b_i}_{\text{cost of each constraint}}$$

Solving Linear Programs

- Would like to minimize

$$\min_{1 \leq i \leq m} A_i x - b_i$$

- MWU shows that over the long run, for any inequality $i \in [m]$:

$$\sum_{t=1}^{T} p^{(t)} \cdot (Ax^{(t)} - b) < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} (A_i x^{(t)} - b_i) + \varepsilon \cdot \sum_{t=1}^{T} |A_i x^{(t)} - b_i|$$

cost of i^{th} expert

Solving Linear Programs

- Would like to minimize

$$\min_{1 \le i \le m} \underbrace{A_i x - b_i}_{m_t(i)}$$

- MWU shows that over the long run, for any inequality $i \in [m]$:

$$\sum_{t=1}^{T} p^{(t)} \cdot (A\underline{x}^{(t)} - b) < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} (A_i \underline{x}^{(t)} - b_i) + \varepsilon \cdot \sum_{t=1}^{T} |A_i x^{(t)} - b_i|$$

- But our theorem required $m_t(i) \in [-1, +1]$... How can we fix this?

later slides

Q: what solution should I return?

so far what we did was: at every step t picked a possible solution

Solving Linear Programs

- Would like to minimize

$$\min_{1 \leq i \leq m} A_i x - b_i$$

- MWU shows that over the long run, for any inequality $i \in [m]$:

$$\sum_{t=1}^{T} p^{(t)} \cdot (Ax^{(t)} - b) < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} (A_i x^{(t)} - b_i) + \varepsilon \cdot \sum_{t=1}^{T} |A_i x^{(t)} - b_i|$$

- But our theorem required $m_t(i) \in [-1, +1]$... How can we fix this?
- Return solution

$$x = \frac{1}{T} \cdot \sum_{t=1}^{T} x^{(t)}$$

average of all proposed solutions

Solving Linear Programs

- Would like to minimize

$$\min_{1 \le i \le m} A_i x - b_i$$

- MWU shows that over the long run, for any inequality $i \in [m]$:

$$\sum_{t=1}^{T} p^{(t)} \cdot (Ax^{(t)} - b) < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} (A_i x^{(t)} - b_i) + \varepsilon \cdot \sum_{t=1}^{T} |A_i x^{(t)} - b_i|$$

- But our theorem required $m_t(i) \in [-1, +1]$... How can we fix this?
- Return solution

$$x = \frac{1}{T} \cdot \sum_{t=1}^{T} x^{(t)}$$

- What if there is no $x \ge 0$ such that $p^{(t)} A x \ge p^{(t)} b$?
  - Farkas' lemma $\Rightarrow$ the system is *infeasible*, and we are done!

Algorithm:

initialize $p^{(t)}(i) = \frac{1}{m} \quad \forall \ i \in [m].$

for $t = 1, 2, \ldots, T$:

find $x^{(t)} \geq 0$ s.t.

$$p^{(t)} A x^{(t)} \geq p^{(t)} b$$

done by our Oracle

only one inequality to take care of

if at any pt there is no solution return infeasible

for $i \in [m]$:

if $A_i x^{(t)} \geq b_i$ then decrease weight of $p^{(t)}(i)$

if constraint i is satisfied decrease weight

return $x = \frac{1}{T} \sum_{t=1}^{T} x^{(t)}$

proposed solution to the LP.

Solving Linear Programs

- Would like to minimize

$$\min_{1 \le i \le m} A_i x - b_i$$

- MWU shows that over the long run, for any inequality $i \in [m]$:

$$\sum_{t=1}^{T} p^{(t)} \cdot (Ax^{(t)} - b) < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} (A_i x^{(t)} - b_i) + \varepsilon \cdot \sum_{t=1}^{T} |A_i x^{(t)} - b_i|$$

- But our theorem required $m_t(i) \in [-1, +1]$... How can we fix this?
- Return solution

$$x = \frac{1}{T} \cdot \sum_{t=1}^{T} x^{(t)}$$

- What if there is no $x \ge 0$ such that $p^{(t)} A x \ge p^{(t)} b$?
  - Farkas' lemma $\Rightarrow$ the system is *infeasible*, and we are done!
  - Thus, we will assume that above never happens.

Theorem

Definition (Oracle)

Let $A \in \mathbb{R}^{m\times n}$. We say that $\mathcal{O}$ is an oracle of *width w* for A if given a linear constraint

$$pAx \geq pb, \;\; x \geq 0$$

$\mathcal{O}(p)$ will return $y \geq 0$ such that

$$|A_i y - b_i| \leq w \quad \forall i \in [m]$$

and

$pAy \geq pb$ (y is a solution to our linear constraint)

the violation of a constraint by y is bounded by w as well as how close it is from being satisfied

Theorem

Definition (Oracle)

Let $A \in \mathbb{R}^{m \times n}$. We say that $\mathcal{O}$ is an oracle of *width w* for A if given a linear constraint

$$pAx \geq pb, \;\; x \geq 0$$

$\mathcal{O}(p)$ will return $y \geq 0$ such that

$$|A_i y - b_i| \leq w \quad \forall i \in [m]$$

Theorem (Multiplicative Weights Update)

Let $\delta > 0$ and suppose we are given an oracle with width w for A. The MWU algorithm either finds a solution $y \geq 0$ such that

$$A_i y \geq b_i - \delta \quad \forall i \in [m]$$

approximate solution

or concludes that the system is infeasible (and outputs a dual solution). Our algorithm makes $O(w^2 \log(m)/\delta^2)$ oracle calls.

Analysis

- As we said before, if oracle fails to find a solution, we found a separating hyperplane and we are done.

Analysis

- As we said before, if oracle fails to find a solution, we found a separating hyperplane and we are done.
- Otherwise, we have that MWU algorithm with costs $m_t(i) = \frac{A_i x^{(t)} - b_i}{w}$ gives us that after T steps

Now can use general thm

$$0 \leq \sum_{t=1}^{T} p^{(t)} \cdot \frac{Ax^{(t)} - b}{w} < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{w} + \varepsilon \cdot \sum_{t=1}^{T} \frac{|A_i x^{(t)} - b_i|}{w}$$

value of our solution at time t: $\frac{p^{(t)}(Ax^{(t)} - b)}{w} \geq 0$ for every t

cost of i^{th} expert

≤ 1 because our oracle has width w

$m_t(i) = \frac{\text{value of constraint } i}{\text{width}}$

Analysis

- As we said before, if oracle fails to find a solution, we found a separating hyperplane and we are done.
- Otherwise, we have that MWU algorithm with costs $m_t(i) = \dfrac{A_i x^{(t)} - b_i}{w}$ gives us that after T steps

$$0 \leq \sum_{t=1}^{T} p^{(t)} \cdot \frac{A_i x^{(t)} - b_i}{w} < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{w} + \varepsilon \cdot \sum_{t=1}^{T} \underbrace{\frac{|A_i x^{(t)} - b_i|}{w}}_{\leq 1}$$

- Thus, we have

$$\sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{T} \geq -\frac{w \log m}{T \cdot \varepsilon} - \varepsilon \cdot w$$

$$0 \leq \frac{\log m}{\epsilon} + \frac{1}{w} \sum_{t=1}^{T} (A_i x^{(t)} - b_i) + \epsilon \cdot T \qquad \times \frac{w}{T}$$

Analysis

- As we said before, if oracle fails to find a solution, we found a separating hyperplane and we are done.
- Otherwise, we have that MWU algorithm with costs $m_t(i) = \dfrac{A_i x^{(t)} - b_i}{w}$ gives us that after T steps

$$\sum_{t=1}^{T} p^{(t)} \cdot \frac{A_i x^{(t)} - b_i}{w} < \frac{\log m}{\varepsilon} + \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{w} + \varepsilon \cdot \sum_{t=1}^{T} \frac{|A_i x^{(t)} - b_i|}{w}$$

- Thus, we have

$$\sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{T} \geq - \frac{w \log m}{T \cdot \varepsilon} - \varepsilon \cdot w$$

- Setting $\varepsilon = \delta / 2w$ and $T = \dfrac{4 \cdot w^2 \cdot \log m}{\delta^2}$ we get

$$\underbrace{A_i \cdot \frac{1}{T} \sum_{t=1}^{T} x^{(t)}}_{x} - \underbrace{\sum_{t=1}^{T} \frac{b_i}{T}}_{b_i} = \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{T} \geq -\delta$$

$$A_i x - b_i \geq -\delta$$

Conclusion

- Online Learning
  1. Experts are weak classifiers, want to choose hypothesis based on these experts
  2. Boosting (in learning theory)
- Solving linear programs! (today)
- Convex Optimization
- Computational Geometry
- many more

Acknowledgement

- Lecture based largely on:
  - Lap Chi's notes
  - Yaron Singer's notes
  - Elad Hazan's survey on online optimization
- See Lap Chi's notes at https://cs.uwaterloo.ca/~lapchi/cs466/notes/L21.pdf
- See Yaron's notes https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture11.pdf
- See Elad's survey at https://arxiv.org/pdf/1909.05207.pdf
- See great survey on MWU ar https://www.cs.princeton.edu/~arora/pubs/MWsurvey.pdf