CS-GY 6763: Lecture 8
Online Gradient Decent, Online Learning

NYU Tandon School of Engineering, Prof. Rajesh Jayaram

ONLINE AND STOCHASTIC GRADIENT DESCENT

Second part of class:

- Basics of <u>Online Learning + Optimization</u>.
- Introduction to <u>Regret Analysis</u>.
- Application to analyzing <u>Stochastic Gradient Descent.</u>
- The Experts Problem, and Multiplicative Weights Update Method.

ONLINE LEARNING

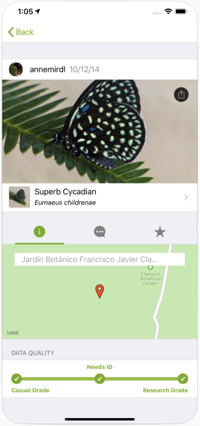
Many machine learning problems are solved in an online setting with constantly changing data.

- Spam filters are incrementally updated and adapt as they see more examples of spam over time.
- Image classification systems learn from mistakes over time (often based on user feedback).
- Content recommendation systems adapt to user behavior and clicks.

EXAMPLE

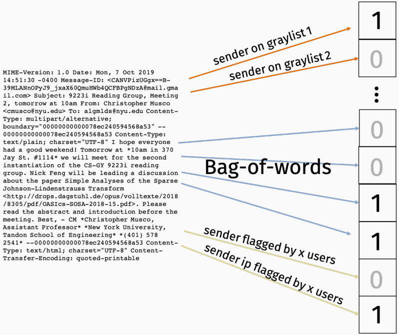
Plant identification via iNaturalist app.

(California Academy of Science + National Geographic)



- When the app fails, image is classified via crowdsourcing (backed by huge network of amateurs and experts).
- Single model that is updated constantly, not retrained in batches.

ML based email spam/scam filtering.



Markers for spam change overtime, so model might change.

EXAMPLE

ML based email spam/scam filtering.

Markers for spam change overtime, so model might change.

ONLINE LEARNING FRAMEWORK

Choose some model $M_{\mathbf{x}}$ parameterized by parameters $\mathbf{x}$ and some loss function ℓ. At time steps $1, \ldots, T$, receive data vectors $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(T)}$.

- At each time step, we pick ("play") a parameter vector $\mathbf{x}^{(i)}$.
- Make prediction $\tilde{y}^{(i)} = M_{\mathbf{x}^{(i)}}(\mathbf{a}_i)$.
- Then told true value or label $y^{(i)}$.
- Goal is to minimize cumulative loss:

$$L = \sum_{i=1}^{n} \ell(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)})$$

For example, for a regression problem we might use the ℓ_2 loss:

$$\ell(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)}) = \left|\langle \mathbf{x}^{(i)}, \mathbf{a}^{(i)} \rangle - y^{(i)}\right|^2.$$

For classification, we could use logistic/cross-entropy loss.

ONLINE OPTIMIZATION

Abstraction as optimization problem: Instead of a single objective function f, we have multiple (initially unknown) functions $f_1, \ldots, f_T : \mathbb{R}^d \to \mathbb{R}$, one for each time step.

- For time step $i \in 1, \ldots, T$, select vector $\mathbf{x}^{(i)}$.
- Observe f_i and pay cost $f_i(\mathbf{x}^{(i)})$
- Goal is to minimize $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})$.

We make no assumptions that $f_1, \ldots, f_T$ are related to each other at all!

REGRET BOUND

In offline optimization, we wanted to find $\hat{\mathbf{x}}$ satisfying $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x}} f(\mathbf{x})$. Ask for a similar thing here.

Objective: Choose $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$ so that:

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right] + \epsilon.$$

Here ϵ is called the **regret** of our solution sequence $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$.

REGRET BOUND

Regret compares to the best fixed solution in hindsight.

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right] + \epsilon.$$

It's very possible that $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) < \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right]$. Could we hope for something stronger?

Exercise: Argue that the following is impossible to achieve:

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\sum_{i=1}^{T} \min_{\mathbf{x}} f_i(\mathbf{x})\right] + \epsilon.$$

HARD EXAMPLE FOR ONLINE OPTIMIZATION

Convex functions:

$$f_1(x) = |x - h_1|$$
$$\vdots$$
$$f_n(x) = |x - h_T|$$

where $h_1, \dots, h_T$ are i.i.d. uniform $\{0, 1\}$.

REGRET BOUNDS

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right] + \epsilon.$$

Beautiful balance:

- Either $f_1, \ldots, f_T$ are similar, so we can learn predict f_i from earlier functions.
- Or $f_1, \ldots, f_T$ are very different, in which case $\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ is large, so regret bound is easy to achieve.
- Or we live somewhere in the middle.

ONLINE GRADIENT DESCENT

Online Gradient descent:

- Choose $\mathbf{x}^{(1)}$ and $\eta = \frac{R}{G\sqrt{T}}$.
- For $i = 1, \ldots, T$:
    - Play $\mathbf{x}^{(i)}$.
    - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.
    - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$

If $f_1, \ldots, f_T = f$ are all the same, this looks a lot like regular gradient descent. We update parameters using the gradient ∇f at each step.

ONLINE GRADIENT DESCENT (OGD)

$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ (the offline optimum)

Assume:

- $f_1, \ldots, f_T$ are all convex.
- Each is G-Lipschitz: for all $\mathbf{x}$, i, $\|\nabla f_i(\mathbf{x})\|_2 \leq G$.
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Online Gradient descent:

- Choose $\mathbf{x}^{(1)}$ and $\eta = \frac{R}{G\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$

ONLINE GRADIENT DESCENT ANALYSIS

Let $\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ (the offline optimum)

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Average regret overtime is bounded by $\frac{\epsilon}{T} \leq \frac{RG}{\sqrt{T}}$.

Goes $\rightarrow 0$ as $T \rightarrow \infty$.

All this with no assumptions on how $f_1, \ldots, f_T$ relate to each other! They could have even been chosen **adversarially** – e.g. with f_i depending on our choice of $\mathbf{x}_i$ and all previous choices.

ONLINE GRADIENT DESCENT ANALYSIS

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Claim 1: For all $i = 1, \ldots, T$,

$$f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(i+1)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

(Same proof as previous class. Only uses convexity of f_i.)

ONLINE GRADIENT DESCENT ANALYSIS

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Claim 1: For all $i = 1, \ldots, T$,

$$f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(i+1)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

Telescoping Sum:

$$\sum_{i=1}^{T} \left[f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*)\right] \leq \frac{\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2 - \|\mathbf{x}^{(T)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{T\eta G^2}{2}$$

$$\leq \frac{R^2}{2\eta} + \frac{T\eta G^2}{2} = RG\sqrt{T}$$

where last inequality follows from setting $\eta = \frac{R}{G\sqrt{T}}$. ■

STOCHASTIC GRADIENT DESCENT (SGD)

Efficient offline optimization method for functions f with finite sum structure:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}).$$

Goal is to find $\hat{\mathbf{x}}$ such that $f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*) + \epsilon$.

- The most widely use optimization algorithm in modern machine learning.
- Easily analyzed as a special case of online gradient descent!

STOCHASTIC GRADIENT DESCENT

Recall the machine learning setup. In empirical risk minimization, we can typically write:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$$

where f_i is the loss function for a particular data example $(\mathbf{a}^{(i)}, y^{(i)})$.

Example: least squares linear regression.

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\mathbf{x}^T \mathbf{a}^{(i)} - y^{(i)})^2$$

Note that by linearity, $\nabla f(\mathbf{x}) = \sum_{i=1}^{n} \nabla f_i(\mathbf{x})$.

Main idea: Use random approximate gradient in place of actual gradient.

Pick random $j \in 1, \ldots, n$ and update $\mathbf{x}$ using $\nabla f_j(\mathbf{x})$.

$$\mathbb{E}\left[\nabla f_j(\mathbf{x})\right] = \frac{1}{n}\nabla f(\mathbf{x}).$$

$n\nabla f_j(\mathbf{x})$ is an unbiased estimate for the true gradient $\nabla f(\mathbf{x})$, but can often be computed in a $1/n$ fraction of the time!

Trade slower convergence for cheaper iterations.

STOCHASTIC GRADIENT DESCENT

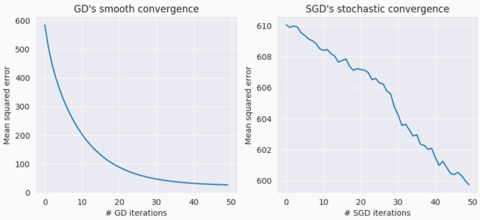
Stochastic first-order oracle for $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$.

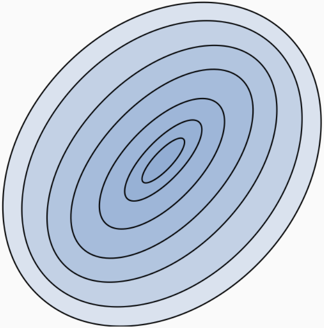
- **Function Query:** For any chosen $j, \mathbf{x}$, return $f_j(\mathbf{x})$
- **Gradient Query:** For any chosen $j, \mathbf{x}$, return $\nabla f_j(\mathbf{x})$

Stochastic Gradient descent:

- Choose starting vector $\mathbf{x}^{(1)}$, learning rate η
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, n$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_{j_i}(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^{T} \mathbf{x}^{(i)}$

VISUALIZING SGD





STOCHASTIC GRADIENT DESCENT

Assume:

- Finite sum structure: $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$, with $f_1, \ldots, f_n$ all convex.
- Lipschitz functions: for all $\mathbf{x}$, j, $\|\nabla f_j(\mathbf{x})\|_2 \leq \frac{G'}{n}$.
  - What does this imply about Lipschitz constant of f?
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Stochastic Gradient descent:

- Choose $\mathbf{x}^{(1)}$, steps T, learning rate $\eta = \frac{D}{G'\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, n$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_{j_i}(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^{T} \mathbf{x}^{(i)}$

Approach: View as online gradient descent run on function sequence $f_{j_1}, \ldots, f_{j_T}$.

Only use the fact that step equals gradient in expectation.

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ iterations:

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

Want to first show:

$$f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \frac{1}{T} \sum_{i=1}^{T} \left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*) \right]$$

STOCHASTIC GRADIENT DESCENT ANALYSIS

Jensen's Inequality: for any $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \mathbb{R}^d$, and coefficients $a_1, \ldots, a_T \geq 0$, with $\sum_i a_i = 1$, if f is convex then

$$f\left(\sum_i a_i \mathbf{x}_i\right) \leq \sum_i a_i f\left(\mathbf{x}_i\right)$$

STOCHASTIC GRADIENT DESCENT ANALYSIS

Jensen's Inequality: for any $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \mathbb{R}^d$, and coefficients $a_1, \ldots, a_T \geq 0$, with $\sum_i a_i = 1$, if f is convex then

$$f\left(\sum_i a_i \mathbf{x}_i\right) \leq \sum_i a_i f\left(\mathbf{x}_i\right)$$

Using Jensen's:

$$\begin{aligned} f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) &= f\left(\frac{1}{T}\sum_i \mathbf{x}^{(i)}\right) - \frac{1}{T}\sum_{i=1}^{T} f(\mathbf{x}^*) \\ &\leq \frac{1}{T}\sum_{i=1}^{T}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right] \end{aligned}$$

STOCHASTIC GRADIENT DESCENT ANALYSIS

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ iterations:

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$$\begin{aligned}
\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] &\leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right] \\
&= \frac{1}{T}\sum_{i=1}^{T} n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right] \\
&= \frac{n}{T} \cdot \mathbb{E}\left[\sum_{i=1}^{T} f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right] \\
&\leq \frac{n}{T} \cdot \left(R \cdot \frac{G'}{n} \cdot \sqrt{T}\right) \qquad \text{(by OGD guarantee.)}
\end{aligned}$$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

Number of iterations for error ϵ:

- **Gradient Descent**: $T = \frac{R^2 G^2}{\epsilon^2}$.
- **Stochastic Gradient Descent**: $T = \frac{R^2 G'^2}{\epsilon^2}$.

Always have $G \leq G'$. Follows by triangle inequality:

$$\max_{\mathbf{x}} \|\nabla f(\mathbf{x})\|_2 \leq \max_{\mathbf{x}} \left(\|\nabla f_1(\mathbf{x})\|_2 + \ldots + \|\nabla f_n(\mathbf{x})\|_2\right) \leq n \cdot \frac{G'}{n} = G'.$$

So GD converges strictly faster than *SGD*.

But for a fair comparison:

- SGD cost = (# of iterations) $\cdot\, O(1)$
- GD cost = (# of iterations) $\cdot\, O(n)$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

We always have $G \leq G'$. When it is much smaller then GD will perform better. When it is closer to this upper bound, SGD will perform better.

What is an extreme case where $G = G'$?

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

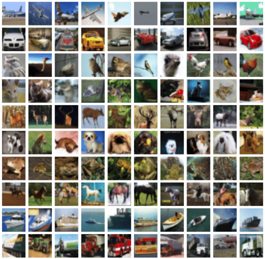
What if each gradient $\nabla f_i(\mathbf{x})$ looks like random vectors in $\mathbb{R}^d$? E.g. with $\mathcal{N}(0,1)$ entries?

$$\mathbb{E}\left[\|\nabla f_i(\mathbf{x})\|_2^2\right] =$$

$$\mathbb{E}\left[\|\nabla f(\mathbf{x})\|_2^2\right] = \mathbb{E}\left[\left\|\sum_{i=1}^{n} \nabla f_i(\mathbf{x})\right\|_2^2\right] =$$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

Takeaway: SGD performs better when there is more structure or repetition in the data set.



Online Learning: Multiplicative Weights Algorithm

LEARNING WITH EXPERTS

Imagine we are trying to pick a good time to invest in a stock S.

- Each morning, we predict if the stock will go up or down.
- To aid us, we consult a team of n experts, who each predict either "up" or "down".
- **Goal:** Make a prediction each morning, so that # mistakes is not too much worse than the best expert.

LEARNING WITH EXPERTS

Model: Days $t = 1, 2, \ldots, T$, experts $E_1, E_2, \ldots, E_n$.

- Each day t, every expert $E_i \in [n]$ makes a prediction $e_i^{(t)} \in \{0, 1\}$ for whether stock will go up or down.
- We see the advice $e_1^{(t)}, \ldots, e_n^{(t)}$, and make a prediction $x^{(t)} \in \{0, 1\}^n$.
- We then pay a cost $f_i(x^{(t)}) : \{0, 1\} \to \{0, 1\}$ where f_i is 1 if we were incorrect, and 0 if we were correct.

Goal: want to minimize **regret**:

$$\sum_{i=1}^{T} f_i(x^{(i)}) - \min_{i \in [n]} \left(\sum_{i=1}^{T} f_i(e^{(i)}) \right)$$

LEARNING WITH EXPERTS

Minimize Regret:

$$\sum_{i=1}^{T} f_i(x^{(i)}) - \min_{i \in [n]} \left(\sum_{i=1}^{T} f_i(e^{(i)}) \right)$$

- No assumptions at all on experts! The advice $e_1^{(t)}, \ldots, e_n^{(t)}$ can be arbitrarily correlated.
- Experts may or may not know what they are talking about.
- Stock movements can be arbitrary and *adversarialy*: i.e. the functions f_i can be adversarial based on $e_1^{(t)}, \ldots, e_n^{(t)}$ and all prior events!
- Still want to compete with best expert in hindsight.

First attempt, set $x^{(t)} = \texttt{Majority}(e_1^{(t)}, e_2^{(t)}, \ldots, e_n^{(t)})$.

What is wrong with this algorithm?

LEARNING WITH EXPERTS

First attempt, set $x^{(t)} = \texttt{Majority}(e_1^{(t)}, e_2^{(t)}, \ldots, e_n^{(t)})$.

What is wrong with this algorithm?

Suppose $e_i^{(t)}$ is always correct $\left(f_t(e_i^{(t)}) = 0 \text{ for each } t \in [T]\right)$, and all other experts give incorrect advice $e_j^{(t)}$. Majority is always wrong!

But we would notice that $e_i^{(t)}$ was doing well pretty quickly...

- The more an expert is wrong, the less we should trust them!

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Key idea: Give each expert E_i a weight w_i. Whenever E_i is wrong, *penalize them* by cutting their weight.

- Always choose the decision $x^{(t)} \in \{0, 1\}$ which the *weighted majority* of experts agree with.

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Key idea: Give each expert E_i a weight w_i. Whenever E_i is wrong, *penalize them* by cutting their weight.

- Always choose the decision $x^{(t)} \in \{0, 1\}$ which the *weighted majority* of experts agree with.

MPW Algorithm: Fix "learning rate" $\eta \in (0, \frac{1}{2})$, initialize weights $w_i^{(1)} = 1$ for $i \in [n]$.

1. Set $W_0^{(t)} = \sum_{i, e_i^{(t)} = 0} w_i^{(t)}$ and $W_1^{(t)} = \sum_{i, e_i^{(t)} = 1} w_i^{(t)}$.
2. If $W_0^{(t)} > W_1^{(t)}$, set $x^{(t)} = 0$ otherwise set $x^{(t)} = 1$.
3. Observe $f_t(x^{(t)})$. Then for each incorrect expert E_i, set $w_i^{(t+1)} \leftarrow (1 - \eta) w_i^{(t)}$.

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Theorem

Fix any $\eta \in (0, \frac{1}{2})$*. Let* $m_i^{(T)}$ *be the number mistakes made by expert* E_i*, and* $M^{(t)}$ *the number of mistakes the prior algorithm makes. Then for every* $i \in [n]$*, we have*

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

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Theorem

Fix any $\eta \in (0, \frac{1}{2})$. Let $m_i^{(T)}$ be the number mistakes made by expert E_i, and $M^{(t)}$ the number of mistakes the prior algorithm makes. Then for every $i \in [n]$, we have

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

Note: whenever the best expert i makes $m_i^{(T)} \gg \frac{2\ln n}{\eta}$ mistakes, our algorithm is at most $\approx 2(1+\eta)m_i^{(T)}$ mistakes – nearly a 2-approx!

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Proof: First note that $w_i^{(t+1)} = (1-\eta)^{m_i^{(t)}}$ (**why?**).

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Proof: First note that $w_i^{(t+1)} = (1-\eta)^{m_i^{(t)}}$ (**why?**).

Potential function: let $\Phi^{(t)} = \sum_{i \in [n]} w_i^{(t)} = W_0^{(t)} + W_1^{(t)}$ be the total weight at time t. Note $\Phi^{(1)} = n$.

Each time we make a mistake, it must be that the *weighted majority* of experts were incorrect.

- I.e. if the correct answer was 0 and we choose $x^{(t)} = 1$, then $W_1^{(t)} > W_0^{(t)}$, meaning $W_1^{(t)} \geq \Phi^{(t)}/2$, and then $W_1^{(t+1)} = (1-\eta)W_1^{(t)}$.

Thus, we have:

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

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Summary: each time we make a mistake, the potential decreases by a factor of $(1 - \eta/2)$, namely, after each mistake

$$\Phi^{(t+1)} \leq (1 - \frac{\eta}{2})\Phi^{(t)}$$

Since $\Phi^{(1)} = n$, we have $\Phi^{(T+1)} \leq n(1 - \frac{\eta}{2})^{M^{(T)}}$.

But we also have $\Phi^{(T)} > w_i^{(T+1)}$ for all i, so

$$n(1 - \frac{\eta}{2})^{M^{(T)}} \geq w_i^{(t+1)} = (1 - \eta)^{m_i^{(t)}}$$

Taking logarithms of both sides, and using that $-\ln(1 - \eta) \leq \eta + \eta^2$, we have the desired bound

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

THE MULTIPLICATIVE WEIGHTS ALGORITHM

Theorem

Let $m_i^{(T)}$ be the number mistakes made by expert E_i, and $M^{(t)}$ the number of mistakes the prior algorithm makes. Then for every $i \in [n]$, we have $M^{(T)} \leq 2(1+\eta)m_i^{(T)} + \frac{2 \ln n}{\eta}$

Remark: This theorem can be generalized via a *randomized* update rule: choose each expert i with probability proportion to $w_i^{(t)}$. Note that this allows for multiple possible outputs (instead of two: $\{0, 1\}$). One can show that, under this alternate rule, we have:

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

Details to be posted on course website.

Midterm

MIDTERM STATS

Overall: very good!

Midterm: Out of 55 points:
Mean: 41.16
Median: 45
Std Dev: 12
75 percentile: 49.75
25 percentile: 31
Max: 55