CIS 410/510 (Spring 2020): Multi-agent Systems and Game Theory

Lecture 15:
Introduction to Online Learning

Thanh H. Nguyen

Most slides are from Haifeng Xu, http://www.haifeng-xu.com/cs6501fa19/
Announcement

- Written assignment 4
  - Will be posted today
  - Deadline: May 29th, 2020
Outline

- Online Learning/Optimization
- Measure Algorithm Performance via Regret
- Warm-up: A Simple Example
- Multiplicative Weight Update
Overview of Machine Learning

- **Supervised learning**
  - Labeled training data
  - ML Algorithm
  - Classifier/Regression function

- **Unsupervised learning**
  - Unlabeled training data
  - ML Algorithm
  - Clusters/Knowledge

- **Semi-supervised learning** (a combination of the two)

What else are there?
Overview of Machine Learning

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Online learning
- Reinforcement learning
- Active learning
- . . .
Online Learning: When Data Come Online

The online learning pipeline:

- Observed one more training instance
- Update ML algorithm
- Make predictions/decisions
- Receive loss/reward
Typical Assumptions on Data

- **Statistical feedback**: instances drawn from a fixed distribution
  - Image classification, predict stock prices, choose restaurants, gambling machine (a.k.a., bandits)

- **Adversarial feedback**: instances are drawn adversarially
  - Spam detection, anomaly detection, game playing

- **Markovian feedback**: instances drawn from a distribution which is dynamically changing
  - Interventions, treatments
Online Learning for Decision Making

- Learn to commute to school
  - Bus, walking, or driving? Which route? Uncertainty on the way?
- Learn to gamble or buy stocks
Online Learning for Decision Making

- Learn to commute to school
  - Bus, walking, or driving? Which route? Uncertainty on the way?
- Learn to gamble or buy stocks
- Advertisers learn to bid for keywords
Online Learning for Decision Making

- Learn to commute to school
  - Bus, walking, or driving? Which route? Uncertainty on the way?
- Learn to gamble or buy stocks
- Advertisers learn to bid for keywords
- Recommendation systems learn to make recommendations
Online learning for Decision Making

- Learn to commute to school
  - Bus, walking, or driving? Which route? Uncertainty on the way?
- Learn to gamble or buy stocks
- Advertisers learn to bid for keywords
- Recommendation systems learn to make recommendations
- Clinical trials
- Robotics learn to react
- Learn to play games (video games and strategic games)
- Even how you learn to make decisions in your life
- ...
Adversarial Online Learning: Model Sketch

- A learner acts in an uncertain world for \( T \) time steps.
- Each step \( t = 1, \ldots, T \), learner takes an action \( i_t \in [n] = \{1, \ldots, n\} \).
- Learner observes cost vector \( c_t \) where \( c_t(i) \in [0,1] \) is the cost of action \( i \in [n] \):
  - Learner suffers cost \( c_t(i_t) \) at step \( t \).
  - Can be similarly defined as reward instead of cost, not much difference.
  - There are also “partial feedback” models (will not cover here).
- Adversarial feedbacks: \( c_t \) is chosen by an adversary:
  - The powerful adversary has access to all the history (learner actions, past costs, etc.) until \( t - 1 \), and also the learner’s algorithm.
  - There are models of stochastic feedbacks (will not cover today).
- Learner’s goal: minimize \( \sum_{t \in [T]} c_t(i_t) \).
Formal Procedure of the Model

At each time step $t = 1, \cdots, T$, the following occurs in order:

1. Learner picks a distribution $p_t$ over actions $[n]$
2. Adversary picks cost vector $c_t \in [0,1]^n$ (he knows $p_t$)
3. Action $i_t \sim p_t$ is chosen and learner incurs cost $c_t(i_t)$
4. Learner observes $c_t$ (for use in future time steps)

- Learner tries to pick distribution sequence $p_1, \cdots, p_T$ to minimize expected cost $\mathbb{E} \left[ \sum_{t \in T} c_t(i_t) \right]$
  - Expectation over randomness of action

- The adversary does not have to really exist – it is assumed mainly for the purpose of worst-case analysis
Sampling

- Sampling from given distribution
  - Step 1: Get sample \( u \) from uniform distribution over \([0, 1)\)
    - E.g. random() in python
  - Step 2: Convert this sample \( u \) into an outcome for the given distribution
    - Each target outcome is associated with a sub-interval of \([0,1)\)
    - Sub-interval size is equal to probability of the outcome.

- Example

<table>
<thead>
<tr>
<th>C</th>
<th>P(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>0.6</td>
</tr>
<tr>
<td>green</td>
<td>0.1</td>
</tr>
<tr>
<td>blue</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- If random() returns \( u = 0.83 \), then our sample is \( C = \text{blue} \)
- E.g, after sampling 8 times:
Well, Adversary Seems Too Powerful?

- Adversary can choose $c_t \equiv 1, \forall t$; learner suffers cost $T$ regardless
  - Cannot do anything non-trivial? We are done?

- If $c_t \equiv 1 \forall t$, if you look back at the end, you do not regret anything – had you known such costs *in hindsight*, you cannot do better
  - From this perspective, cost $T$ in this case is not bad

So what is a good measure for the performance of an online learning algorithm?
Outline

- Online Learning/Optimization
- Measure Algorithm Performance via Regret
- Warm-up: A Simple Example
- Multiplicative Weight Update
Regret

- Measures how much the learner regrets, had he known the cost vector $c_1, \ldots, c_T$ in hindsight

- Formally,
  \[ R_T = \mathbb{E}_{i_t \sim p_t} \sum_{t \in [T]} c_t(i_t) - \min_{i \in [n]} \sum_{t \in [T]} c_t(i) \]

- Benchmark $\min_{i \in [n]} \sum_{t} c_t(i)$ is the learner utility had he known $c_1, \ldots, c_T$ and the learner is allowed to take the best single action across all rounds.
  - There are other concepts of regret, e.g., swap regret.
  - But, $\min_{i \in [n]} \sum_{t} c_t(i)$ is mostly used.

Regret is an appropriate performance measure of online algorithms.
- It measures exactly the loss due to not knowing the data in advance.
Average Regret

\[ \bar{R}_T = \frac{R_T}{T} = \mathbb{E}_{i_t \sim p_t} \frac{1}{T} \sum_{t \in [T]} c_t(i_t) - \min_{i \in [n]} \frac{1}{T} \sum_{t \in [T]} c_t(i) \]

- When \( \bar{R}_T \to 0 \) as \( T \to \infty \), we say the algorithm has vanishing regret or no-regret; the algorithm is called a no-regret online learning algorithm
  - Equivalently, \( R_T \) is sublinear in \( T \)
  - Both are used, depending on your habits

Our goal: design no-regret algorithms by minimizing regret
A Naive Strategy: Follow the Leader (FTL)

- That is, pick the action with the smallest accumulated cost so far

What is the worst-case regret of FTL?

Answer: worst (largest) regret $T/2$

- Consider following instance with 2 actions

<table>
<thead>
<tr>
<th>$t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_t(1)$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>*</td>
</tr>
<tr>
<td>$c_t(2)$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>*</td>
</tr>
</tbody>
</table>

- FTL always pick the action with cost 1 $\rightarrow$ total cost $T$
- Best action in hindsight has cost at most $T/2$
Randomization is Necessary

In fact, any deterministic algorithm suffers (linear) regret \((n - 1)T/n\)

- Recall, adversary knows history and learner’s algorithm
  - So he can infer our \(p_t\) at time \(t\) (but do not know our sampled \(i_t \sim p_t\))
- But if \(p_t\) is deterministic, action \(i_t\) can also be inferred
- Adversary simply sets \(c_t(i_t) = 1\) and \(c_t(i) = 0\) for all \(i \neq i_t\)
- Learner suffers total cost \(T\)
- Best action in hindsight has cost at most \(T/n\)

Can randomized algorithm achieve sublinear regret?
Outline

- Online Learning/Optimization
- Measure Algorithm Performance via Regret
- Warm-up: A Simple Example
- Multiplicative Weight Update
Consider a Simpler (Special) Setting

- Only two types of costs, $c_t(i) \in \{0,1\}$
- One of the actions is perfect – it always has cost 0
  - Minimum cost in hindsight is thus 0
  - Learner does not know which action is perfect

Is it possible to achieve sublinear regret in this simpler setting?
A Natural Algorithm

Observations:
1. If an action ever had non-zero costs, it is not perfect
2. Actions with all zero costs so far, we do not really know how to distinguish them currently

These motivate to the following natural algorithm

For $t = 1, \cdots, T$
- Identify the set of actions with zero total cost so far, and pick one action from the set uniformly at random.

Note: there is always at least one action to pick since the perfect action is always a candidate
Analysis of the Algorithm

- Fix a round $t$, we examine the expected loss from this round
- Let $S_{\text{good}} = \{\text{actions with zero total cost before t}\}$ and $k = |S_{\text{good}}|$
  - So each action in $S_{\text{good}}$ is picked with probability $1/k$
- For any parameter $\epsilon \in [0,1]$, one of the following two happens
  - **Case 1:** at most $\epsilon \times k$ actions from $S_{\text{good}}$ have cost 1, in which case we suffer expected cost at most $\epsilon$
  - **Case 2:** at least $\epsilon \times k$ actions from $S_{\text{good}}$ have cost 1, in which case we suffer expected cost at most 1
- How many times can **Case 2** happen?
  - Each time it happens, size of $S_{\text{good}}$ shrinks from $k$ to at most $(1 - \epsilon)k$
  - At most $\log_{1-\epsilon} n^{-1}$ times
- The total cost of the algorithm is at most $T \times \epsilon + \log_{1-\epsilon} n^{-1} \times 1$
Analysis of the Algorithm

- The cost upper bound can be further bounded as follows

\[
\text{Total Cost} \leq T \times \epsilon + \log_{1-\epsilon} n^{-1} \times 1
\]

\[
= T \times \epsilon + \frac{\ln n}{-\ln(1 - \epsilon)}
\]

\[
\leq T \times \epsilon + \frac{\ln n}{\epsilon}
\]

Since \( \log_a b = \frac{\ln b}{\ln a} \)

Since \(-\ln(1 - \epsilon) \geq \epsilon, \forall \epsilon \in (0,1)\)

- The above upper bound holds for any \( \epsilon \), so picking \( \epsilon = \sqrt{\ln n / T} \) we have

\[
R_T = \text{Total Cost} - 0 \leq 2 \sqrt{T \ln n}
\]

Sublinear in \( T \)
Outline

- Online Learning/Optimization
- Measure Algorithm Performance via Regret
- Warm-up: A Simple Example
- Multiplicative Weight Update
What about the General Case?

- $c_t \in [0,1]^n \Rightarrow$ the weight update process is still okay
- No perfect action $\Rightarrow$ more conservative when eliminating actions
- Previous algorithm can be re-written in a more “mathematically beautiful” way, which turns out to generalize

Initialize action weight $w_1(i) = 1, \forall i = 1, \cdots, n$
For $t = 1, \cdots, T$
1. Let $W_t = \sum_{i \in [n]} w_t(i)$, pick action $i$ with probability $w_t(i)/W_t$
2. Observe cost vector $c_t \in [0,1]^n$
3. For any $i \in [n]$, update $w_{t+1}(i) = w_t(i) \cdot (1 - \epsilon \cdot c_t(i))$

Multiplicative Weight Update (MWU)
Multiplicative Weight Update

**Theorem.** Multiplicative Weight Update (MWU) achieves regret at most $O(\sqrt{T \ln n})$ for the previously described general setting.

- Proof of the theorem is omitted
- Note: we care about theoretical bound for online algorithms
  - The environment is uncertain and difficult to simulate, there is no easy way to experimentally evaluate the algorithm

Is $O(\sqrt{T \ln n})$ is best possible regret? YES!!!
Some MW description uses $w_{t+1}(i) = w_t(i) \cdot e^{-\epsilon \cdot c_t(i)}$. Analysis is similar due to the fact $e^{-\epsilon} \approx 1 - \epsilon$ for small $\epsilon \in [0,1]$

The same algorithm also works for $c_t \in [-\rho, \rho]$ (still use update rule $w_{t+1}(i) = w_t(i) \cdot (1 - \epsilon \cdot c_t(i))$). Analysis is the same.

MW update is a very powerful technique – it can also be used to solve, e.g., LP, semidefinite programs, SetCover, Boosting, etc.
- Because it works for arbitrary cost vectors
- Next, we show how it can be used to compute equilibria of games where the “cost vector” will be generated by other players