Privacy in Statistics and Machine Learning — Spring 2021
Lecture 14 & 15: Online Decision-making and Multiplicative Weights

Adam Smith and Jonathan Ullman

1 From Query Release and Synthetic Data to Two-party Games and Online Learning

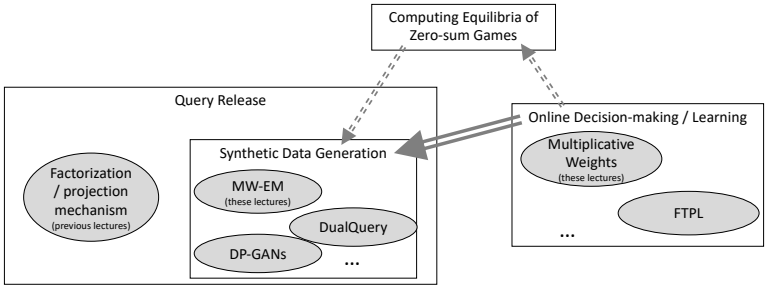
Ove the last two lectures, we saw a general framework for answering a set of queries by adding noise to some linearly-transformed version of the query answers. Over the next few lectures, we'll see a general framework for performing query release via *synthetic data generation*, and a framework for synthetic data generation via *online learning*.

The connection between synthetic data generation and query release is fairly straightforward. Given a set F of queries we want to release, the idea is to use a differentially private algorithm to generate a "fake" data set $\tilde{\mathbf{x}}$ that is similar to the original data set $\mathbf{x}$ with respect to the frequencies of all the queries in F. Anyone who wants to know the answer to a query f on $\mathbf{x}$ can instead run the query on $\tilde{\mathbf{x}}$ and use that answer instead.

The factorization/projection algorithm we saw in the last two lectures does not automatically generate synthetic data consistent with its query answers.

We will see algorithms that do generate this type of data and that come with clean guarantees on the error with which they preserve query answers. The first is *multiplicative-weights-using-the-exponential-mechanism* (abbreviated MW-EM and sometimes pronounced "em-wem"). To understand how it works, though, we will first need to understand the task of *online learning*, also called *online decision-making* (Section 2) along with a famous algorithm for online learning called *multiplicative weights* (MW) in Section 3.

Armed with the MW algorithm, we will see a first connection between online learning and synthetic data generation. That connection will lead us to MW-EM. We will then see a slightly more general version of the connection in terms of algorithms that seek to solve a particular two-player zero-sum game. The following figure gives a high-level overview of the big pieces that we will fill in.



Let's start with online learning.

2 Online Decision-Making: Definitions and Warm-Up

Online learning is a stylized model of decision-making that has provided powerful tools in optimization, game theory and other areas of computer science.

As a simple example, suppose you have a small amount of money to invest in the stock market and you start looking into some particular stock, say something stable like GameStop (GME). Should you buy it? Fortunately, advice from "experts" on the Internet abounds. You decide that each day you will consult k different newsletters, each of which offers a recommendation to buy or sell. Each day you will follow the advice of one of the k newsletters. At the end of the day, you'll know if you should have bought or sold, and thus which of the k newsletters gave good advice that day. You could assign to each of them a "cost" for the day—say 0 for a good recommendation, and 1 for a bad one. You'd like an algorithm to help you decide whose advice to follow so that, after T days, you're reasonably happy with your choices. Specifically, you'd like to look back at the costs of the k newsletters over those T days and know that you made the right decision at least (roughly) as many times as the *best newsletter in hindsight*.

As a somewhat more complex example, suppose that each day you have the choice of k different stocks to invest in. You've given up on the newsletters—even the best one wasn't that great. Instead, you'd like to base your decisions on the stocks' past performance. Each day, you decide how to spread a \$1 investment among the k stocks. Your strategy for day t is described by a portfolio vector $\mathbf{p}^t$ that lists what fraction of your dollar went into each of the k stocks. Notice that we can think of $\mathbf{p}^t$ as a probability distribution over stocks. At the end of the day, each stock a will go up or down, leading to a cost c_a^t for investing that stock (if the stock goes up, your cost will be negative). Your overall cost will be the weighted sum of the stocks' costs, which we can write as an inner product $\left\langle \mathbf{c}^t, \mathbf{p}^t \right\rangle = \sum_a c_a^t p_a^t$.

The following abstraction turns out to be tremendously useful, and captures both settings above. There is a game between two players: a decision maker $\mathcal{D}$ and an adversary $\mathcal{A}$. In each round, the decision maker selects among a set of k actions $A = \{1, ..., k\}$ (written $[k]$) and the adversary picks a cost in $[0, 1]$ for each action. $\mathcal{D}$ doesn't know the costs ahead of time, and the adversary doesn't necessarily know exactly which action $\mathcal{D}$ will choose. However, we want to model as rich a set of situations as possible, so *we will assume that the adversary knows $\mathcal{D}$'s strategy.* One way to think of this is that at each round, $\mathcal{D}$ selects a distribution over actions $\mathbf{p}^t$, and the adversary selects the costs knowing $\mathbf{p}^t$. We get the following game:

Online Learning Game

For $t = 1, 2, ..., T$, do:

- $\mathcal{D}$ selects distribution $\mathbf{p}^t$ over actions $A = [k]$
  (so $\mathbf{p}^t \in [0, 1]^k$ with $\sum_a p_a^t = 1$)
- Adversary $\mathcal{A}$ sees $\mathbf{p}^t$ and picks cost vector $\mathbf{c}^t \in [0, 1]^k$.
- An action a^t is chosen according to $\mathbf{p}^t$ and $\mathcal{D}$ pays cost $c_{a^t}^t$.
  We generally focus on the *expected* cost $\mathop{\mathbb{E}}_{a \sim \mathbf{p}^t}\left(c_a^t\right) = \left\langle \mathbf{c}^t, \mathbf{p}^t \right\rangle$.
- $\mathcal{D}$ learns the entire cost vector $\mathbf{c}^t$

Measuring Performance Via "Regret" $\mathcal{D}$'s goal is to keep her cost as low as possible. How can we understand how well she is doing? Since the adversary is picking the costs, it can always create a situation where every action has high cost. So $\mathcal{D}$ cannot hope to have low cost in general.

Perhaps, instead, we could get some kind of relative guarantee: we could compare the cost that $\mathcal{D}$ pays in some execution of the algorithm to a baseline that depends on the costs that the adversary actually chose.

One idea is to ask $\mathcal{D}$ to match the cost of the best *sequence* of actions $a^1, a^2, ..., a^T$ for the sequence of cost vectors that actually arose. But that is also an impossibly high standard. For example, the adversary could choose, at each round, one action at random to have cost 0 and set the costs of all other actions to 1. No matter how $\mathcal{D}$ makes her choices, she will have expected cost close to T even though in hindsight there will be a sequence of actions with cost zero!

The general idea is not hopeless though. Instead of competing with the best possible *sequence* in hindsight, we can ask $\mathcal{D}$ to do well with respect to the *best single action in hindsight*.

Definition 2.1. (Regret) In a particular execution of the game described by cost vectors $\mathbf{c}^1, ..., \mathbf{c}^t$ and actions $a^1, ..., a^t$, the decision-maker $\mathcal{D}$'s *regret* is the difference between her realized costs and the best single action in hindsight. We normalize by averaging over the steps of the algorithm:

$$\text{Regret} = \underbrace{\frac{1}{T}\sum_{t=1}^{T} c^t_{a^t}}_{\substack{\text{realized cost}}} - \min_{a^* \in [k]} \underbrace{\frac{1}{T}\sum_{t=1}^{T} c^t_{a^*}}_{\substack{\text{cost of } a^* \\ \text{in hindsight}}} . \tag{1}$$

This looks a lot like our definition of excess risk in optimization problems, and indeed they are connected. But the online setting comes with an important subtlety: the cost vectors are not defined until after the game has been played. The most interesting applications of online learning algorithms are exactly those where this cost vector reflects some hard-to-predict process.

The regret can be at most 1, since we've normalized the costs to lie in $[0, 1]$, and since we are looking at the average cost over all T days. So a regret close to 0 would be a useful guarantee. Surprisingly, there are algorithms that ensure that the regret is very small indeed. The main result we'll see is an algorithm for online learning, dubbed *multiplicative weights*, whose expected regret is $O\left(\sqrt{\frac{\ln(k)}{T}}\right)$ for any adversary. Algorithms whose regret guarantee goes to 0 with T are called "no-regret" algorithms, and have applications in many areas of computer science from linear program solvers to improved stochastic gradient descent to auction design.

An Algorithm That Doesn't Work: Follow The Leader To get a sense of why online learning isn't easy, let's look at an algorithm that doesn't work, dubbed "Follow the Leader". At each stage, the decision maker selects the action that has had the lowest cost so far, breaking ties arbitrarily:

Algorithm 1: Follow the Leader

Select action $a^1 = 1$;
for $t = 2, 3, .., T$ **do**
  Receive cost vector $\mathbf{c}_{t-1}$;
  Select action $a^t = \arg\min_{a \in [k]} \sum_{i=1}^{t-1} c^i_a$;
  (That is, pick action whose overall cost was lowest up until now. In case of a tie, choose the lowest-numbered action that minimizes the cost so far.)

How well does this strategy do in general? Pretty terribly, it turns out. Notice that the regret can be at most T. The "follow-the-leader" strategy can have regret pretty close to that. Even with just two decisions, the algorithm can have regret $T/2$.

Here is one way that can happen. The adversary will simply alternate between the cost vectors $\mathbf{c}^1 = (1, 0)$ and $\mathbf{c}^2 = (0, 1)$. The adversary picks $\mathbf{c}^1 = (1, 0)$ as the first cost vector, so $\mathcal{D}$ pays a cost of 1 for its first action $a^1 = 1$. In the second round, $\mathcal{D}$ plays action 2 (since it had cost 0 in round 1) but the adversary selects $\mathbf{c}^2 = (0, 1)$; again, $\mathcal{D}$ pays a cost of 1. In the third round, $\mathcal{D}$ plays action 1 (since the two actions are tied based on the costs so far), and the adversary plays $\mathbf{c}_3 = (0, 1)$. In every single round, the decision maker ends up making exactly the wrong decision! Over T rounds, the decision maker will have cost T, but the best action in hindsight will have cost at most $T/2$.

What's going on is that the deterministic follow-the-leader algorithm is thrashing and always making the worst choice in a given round. How can we avoid that sort of behavior? The answer is to *randomize* our strategy.

There are actually a few ways to do add this randomization. We'll see a classic method, called *multiplicative weights*.

Exercise 2.2 (Online Learning Requires Randomization)**.** Show that every method that plays deterministic actions (where $\mathbf{p}^t$ puts probability 1 on a single action) there is an adversary for which the regret is $\Omega(T)$.

Exercise 2.3 ($\sqrt{\ln(k)/T}$ lower bound)**.** Show that for any method (randomized or not), if the adversary picks cost vectors unfiormly at random in $\{0, 1\}^k$, the expected regret will be $\Theta\left(\sqrt{\log(k)/T}\right)$. (*Hint:* The expected cost paid the algorithm is exactly $T/2$. Show that in hindsight, with probability at least $1/2$, one of the choices will have cost less than $\frac{T}{2} - \Omega(\sqrt{T \ln(k)})$.

3 Multiplicative Weights

Even though the Follow-the-Leader algorithm did badly, it felt like it had almost the right idea. It was just too extreme in its decisions. We'd like to modify it to follow two general principles:

> We should base our decision on the past performance of the various actions, with actions that did well so far getting higher probability than actions that had higher costs. Actions that did not have the best loss so far should still get nonzero probability. (Exercise 2.2 shows that we need a randomized algorithm to get any reasonable regret bound.)

At any given time step t, let $c_a^{<t}$ denote the sum of the costs experienced so far by a given action a. We would like to define probabilities p_a^t that decrease with this sum, but don't overfit too heavily. Inspired by the exponential mechanism, we can select a given action with probability *exponentially small in the loss experienced so far*. Let $\eta < 1$ be a parameter (called the learning rate, and playing a similar role to the η in gradient descent):

$$p_a^t \propto (1 - \eta)^{c_a^{<t}} = (1 - \eta)^{(\text{total cost of } a \text{ so far})} \tag{2}$$

We obtain the following algorithm[1]:

[1]There are a few essentially equivalent versions of the update algorithm. One common form of the algorithm updates weights as $w_a^{t+1} = w_a^t(1 - \eta c_a^t)$. The analysis is essentially the same as the variant presented here.

Algorithm 2: Multiplicative Weights MW(η)

Initialize all weights to 1: w_a^1 for $a = 1, 2, ..., k$. ;
for $t = 1, 2, ..., T$ **do**
  Compute $Z^t = \sum_{a=1}^{k} w_a^t$;
  Use the distribution $\mathbf{p}^t = \mathbf{w}^t / Z^t$;
  /* Adversary picks cost vector $\mathbf{c}^t$ based on $\mathbf{p}^t$. */
  /* $\mathcal{D}$'s expected loss is $\langle \mathbf{c}^t, \mathbf{p}^t \rangle$. */
  Receive cost vector $\mathbf{c}^t \in [0,1]^k$;
  Update weights $\mathbf{w}^{t+1}$ with the formula $w_a^{t+1} = w_a^t \cdot (1-\eta)^{c_a^t}$;
  /* Observe that $w_a^{t+1} = (1-\eta)^{(\sum_{i=1}^{t} c_a^i)} = (1-\eta)^{\text{(total cost of } a \text{ so far)}}$ */

Theorem 3.1. *MW(η) has expected regret at most $2\sqrt{\ln(k)/T}$ for every adversary when $\eta = \sqrt{\log(k)/T}$.*

Proof. Let's assume that T is big enough so that $\eta \leq 1/2$ (otherwise the bound is vacuous anyway).

The idea of the proof is to look at how Z_t evolves over the execution of the algorithm. Notice that Z_t cannot increase as t grows since the weights get reduced. .

The idea is to show to two basic facts:

1. Whenever $\mathcal{D}$ has high expected cost, Z_t decreases a lot.
2. If there is a good action a^* in hindsight, then the final weight Z_{T+1} is large.

Putting these together, we'll conclude that if there is a good action in hindsight, then $\mathcal{D}$'s average expected loss will be small!

Claim 1. For each t, let v_t be the expected cost paid by $\mathcal{D}$ at time t, that is $v^t \stackrel{\text{def}}{=} \langle \mathbf{c}^t, \mathbf{p}^t \rangle$. Then

$$Z_{t+1} \leq e^{-\eta v^t} \cdot Z_t \,.$$

To prove the claim, let's try to get an upper bound on Z_{t+1}:

$$\begin{aligned}
Z_{t+1} &= \sum_a w_a^{t+1} = \sum_a w_a^t \cdot (1-\eta)^{c_a^t} && (3)\\
&\leq \sum_a w_a^t (1 - \eta c_a^t) \qquad \text{(Using the fact that } (1-\eta)^x \leq 1 - \eta x \text{ for } \eta \in [0, \tfrac{1}{2}] \text{ and } x \in [0,1]) && (4)\\
&= \left\langle \mathbf{w}^t, \vec{1} - \eta \mathbf{c}^t \right\rangle && (5)\\
&= Z_t \cdot \left\langle \mathbf{p}^t, \vec{1} - \eta \mathbf{c}^t \right\rangle && (6)\\
&= Z_t (1 - \eta v^t) && (7)\\
&\leq Z_t \cdot e^{-\eta v^t} && (8)
\end{aligned}$$

We know that $Z_1 = k$ since all the weights are initialized to 1. By induction, we can bound Z_{T+1} as a function of $\mathcal{D}$'s total expected loss:

$$Z_{T+1} \leq d \cdot \prod_t e^{-\eta v^t} = d \cdot e^{-\eta \sum_t v^t} \,. \tag{9}$$

Claim 2. For every a^*, $Z_{T+1} \geq (1-\eta)^{(\text{total cost of } a^*)} \geq e^{(-\eta - \eta^2)(\text{total cost of } a^*)}$.

This claim is simpler: Z_{T+1} is the sum of the weights, and so $Z_{T+1} \geq w_{a^*}^T = (1-\eta)^{(\text{total cost of } a^*)}$. Using the fact that $(1-x) \leq e^{-x-x^2}$ for $x \in [0,1]$ gives us the bound we want.

Combining the two claims: Let a^* be the best action in hindsight for a particular execution, and let $OPT = \sum_t c_{a^*}^t$ be its total cost. We can apply Claims 1 and 2 to relate OPT to the total expected loss:

$$e^{(-\eta-\eta^2)OPT} \leq Z_{T+1} \leq d \cdot e^{-\eta \sum_t v^t}.$$

Taking logs on both sides, and flipping the sign:

$$(\eta+\eta^2)OPT \geq \eta \sum_t v^t - \ln(d)$$

We can rearrange the terms above, and use the fact that OPT is at most T:

$$\underbrace{\left(\frac{1}{T}\sum_t v_t\right) - \frac{OPT}{T}}_{\text{expected regret}} \leq \frac{\ln(k)}{\eta T} + \eta \cdot \frac{OPT}{T} \leq \frac{\ln(k)}{\eta T} + \eta. \tag{10}$$

The left-hand side above is exactly the expected regret: each v^t measures the expected regret at time t conditioned on the previous outcomes, and OPT/T is the average cost of the best choice in hindsight. Setting $\eta = \sqrt{\ln(k)/T}$, we get a bound of $2\sqrt{\ln(k)/T}$ on the expected regret. □

3.1 Variations on the analysis of MW

Let's take some time to think about how we can modify the analysis above.

Exercise 3.2. Suppose we know ahead of time that there is a perfect choice a^* that always has cost 0. Show that we can set η so that the algorithm achieves total cost at most $2\ln(k)$. (Be careful: our proof required η to be at most $1/2$.)

Exercise 3.3. Show that if we know OPT ahead of time, we can set η to get an average cost of at most $\frac{OPT}{T} + \frac{2}{T}\sqrt{\ln(k) \cdot \max(OPT, \ln(k))}$.

Exercise 3.4. Theorem 3.1 requires us to know the number of steps T ahead of time. Show that one can modify the algorithm to adapt automatically to the length of the process. Specifically, there is a standard trick known as "repeated doubling": we start the algorithm assuming we will run for $T_0 = 4\ln(k)$ steps. If the number of steps exceeds T_0, we restart the algorithm assuming a length of $T_1 = 2T_0$. If the number of steps exceeds $T_0 + T_1$, we expand our time horizon to $T_2 = 2T_1$, and so on. Show that this variation achieves average regret $O\left(\sqrt{\ln(k)/T}\right)$, regardless of T.

3.2 From distributions over actions to vectors

Instead of thinking of the decision maker $\mathcal{D}$ as taking a single action distributed according to $\mathbf{p}^t$ at each round, we will sometimes want to think of it as playing a *portfolio* of actions described by $\mathbf{p}^t$. If costs add up across actions and are proportional to the weight placed on each action, then the realized cost is $\left\langle \mathbf{c}^t, \mathbf{p}^t \right\rangle$, the same as the expected cost was when choosing a random action according to $\mathbf{p}^t$.

For instance, in the example of stock investment that we started with, there's no reason to choose a single stock randomly—an investor can instead spread their money across many stocks, proportionally to $\mathbf{p}_t$. Their realized loss or gain is then the sum across stocks of the change in value.

Our online algorithm for choosing actions thus becomes a powerful tool for optimization, when the space of feasible solutions is the space of distributions over $[k]$, called *the probability simplex* $\Delta([k])$. When the output is viewed as a portfolio for each round, the MW algorithm is deterministic, so we can get a bound on the regret that holds with probability 1 (and not just a bound on expected regret). It also makes sense to state the regret as a comparison to the best *distribution* in hindsight, rather than the best fixed action. (The best distribution can always be taken to be a fixed action, but having a more general statement is useful in several applications.)

Theorem 3.5 (MW Updates as Distributions)**.** *For every adversary in the online learning game: If* $\mathbf{p}_1, \mathbf{c}_1, \mathbf{p}_2, \mathbf{c}_2, ..., \mathbf{p}_T, \mathbf{c}_T$ *is the sequence of distributions in* $\Delta([k])$ *and cost vectors in* $[0,1]^k$ *that arise in an execution of* $MW(\eta)$*, then for every distribution* $\mathbf{p}^* \in \Delta([k])$*, the regret with respect to* $\mathbf{p}^*$ *is bounded by* $2\sqrt{\ln(k)/T}$*, that is,*

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T}\left\langle \mathbf{c}^t, \mathbf{p}^t \right\rangle}_{\text{realized cost}} \leq \underbrace{\frac{1}{T}\sum_{t=1}^{T}\left\langle \mathbf{c}^t, \mathbf{p}^* \right\rangle}_{\text{cost of } \mathbf{p}^* \text{ in hindsight}} + 2\sqrt{\frac{\ln(k)}{T}}\,.$$

We'll see the power of this alternate view when we come to apply it to synthetic data release.

Additional Reading and Watching

- The multiplicative weights algorithm is often introduced via a slightly different setting, in which the k experts are offering binary recommendations (e.g., buy or sell). In that setting, multiplicative weights can be thought of as taking a weighted majority vote (with the same weights as we have here).
- The survey of Arora, Hazan and Kale [**?**] covers the multiplicative weights alorithm and a number of its applications.
- The algorithm is covered in many courses. Some easy-to-read lecture notes include those of Roughgarden (Winter 2016), Ene (Spring 2020) and Roth (Spring 2021). The book of Cesa-Bianchi and Lugosi provides a comprehensive but denser treatment [CBL06].
- The connection between MW and two-party games was laid out in the paper of Freund and Schapire [FS96]. The connection is also covered by Blum and Mansour in [NRTV07, Chapter 4].
- The idea of using synthetic data release for DP query release originated in [BLR08]. The connection between online decision-making and DP query release emerged in a series of papers (e.g., [DRV10, RR10, HR14, HLM12]) and lies at the heart of many recent advances in synthetic data generation.

References

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *Annual ACM Symposium on the Theory of Computing*, STOC '08, 2008.

[CBL06] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games.* Cambridge University Press, 2006.

[DRV10] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *FOCS*. IEEE, 2010.

[FS96] Yoav Freund and Robert E Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the 9th annual Conference on Computational Learning Theory*, 1996.

[HLM12] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012.

[HR14] Moritz Hardt and Guy Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *IEEE Symposium on Foundations of Computer Science*, FOCS '10, 2014.

[NRTV07] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[RR10] Aaron Roth and Tim Roughgarden. Interactive privacy via the median mechanism. In *STOC*. ACM, June 5–8 2010.