NIKA HAGHTALAB AND MICHAEL JORDAN

CS294: DECISIONS, LEARNING AND GAMES

Preliminaries

- **Domain (Instance space):** An arbitrary set $\mathcal{X}$ that includes all possible instances.
- **Labels:** A set $\mathcal{Y}$ that includes all possible labels or predictions for a single instance, such as $\{0,1\}$, $\{-1,1\}$, $\{false, true\}$, etc.
- **Labeled instance:** An instance-label pair $(x,y) \in \mathcal{X} \times \mathcal{Y}$ is called a labeled instance.
- **Hypothesis:** A *hypothesis* (a.k.a classifier or predictor) is a function $h : \mathcal{X} \to \mathcal{Y}$.
- **Hypothesis class:** A concept class $\mathcal{H}$ is a pre-determined set of concepts.
- **Consistency:** We say that a hypothesis $h \in \mathcal{H}$ is *consistent* with a set $\{(x_1,y_1),\ldots,(x_m,y_m)\}$, if for all $i \in [m]$, $h(x_i) = y_i$.

Lecture 1

Some Examples

In this section, we give examples of how even simple decisions have consequential implications, have to rely on learning in difficult and complex situations, and contribute to rational behavior that emerges in groups and the society. A common theme in these examples is they involve uncertainty, learning, and decentralized agency.

Example (Which route to use for your daily commute?). Everyday we need to decide on the route P^t we take to work. We want to choose a route that is the shortest (accounting for traffic), but we don't know the traffic ahead of time. In fact, in every day t, the total driving time for route P, denoted by $\text{cost}^t(P)$, may be different from what the traffic looked like in the past, e.g., due to road accidents, first day of school, and changes in the routes other drivers take. So, we want to have a strategy for picking the route P^t that leads to a short commute on average.

The average time spent on commute over T days is $\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^t)$. After T days, we can look back at the historical traffic data and compute the (fixed) route that would have had the best average commute time. The average commute time of the best fixed route in hindsight is $\min_{P*}\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^*)$. Can we have an algorithm for picking P^t that is almost as good as the best route in hindsight, i.e.,

$$\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^t) \leq \min_{P^*}\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^*) + \epsilon,$$

for an $\epsilon \to 0$?

Challenges: When the commute length (traffic) is unpredictable and possibly designed adversarially, we don't know which paths may be good in the future. It's possible that historically well-performing paths don't do well in the next round. So, we need to hedge our bets and *"explore" paths that might have not performed well historically*. We need to balance this off with "exploiting" paths that did perform well, because their average performance will remain good for some time.

Example (Route planners). You could use a route planner, like Google Maps, Apple Maps, or Waze to plan your commute. These planners run a similar "explore-exploit" algorithm in the background not only to hedge their bets but also to gain more information about the traffic as they monitor drivers in action. If this exploration suggest routes that a driver believes to be longer than other alternatives, the driver may benefit from switching his path and not following the instructions. Over the long run, this affects the quality of observations the app may receive. The apps may even lose market share if they use algorithms that explore too carelessly.

Challenges: Can we design explore-exploit algorithms that don't suggest obviously bad routes? Does the competition for market-share between several apps impact their explore-exploit tradoffs and overall learning quality?

Example (Impact of route planning at scale). The commute time and traffic is impacted by the decisions of all other drivers. Often, the more drivers on one road, the longer the commute time. So, $\text{cost}(P)$ is formed by the impact of all driver's choices. Assume that 1000s of drivers, which we refer to as one unit of traffic, want to go from A to B. And the congestion cost of an edge, denoted by $c_e(x)$, is a function of the fraction of drivers x that that take edge e.

A social planner, who tells each person which route to take, can plan an optimal routing, i.e., the least time the society spends on commute! In the above example, when half of the drivers take AVB and the other half take AWB, then commute time is just 1.5. This allocation is also individually optimal to each player. That is, no driver can switch to a different route and reduce their own commute time. Such an allocation is called an "equilibrium".

What if the city adds a bridge between V and W, with congestion cost of $c(x) = 0$? While the previously socially optimal solution is still a valid solution with commute time of 1.5, this solution is no longer an equilibrium because players benefit from deviating from their current path and taking the bridge. In fact, the only equilibrium in this case is a solution where all drivers take the bridge and pay the commute cost of 2.

Challenges: Computing socially optimal solutions and equilibria need a lot of information and coordination. Can these be achieved by simple learning algorithms? e.g., what if each diver deploys an online learning algorithm (from Example 1) or simply picks the (instantaneous) shortest route? Can learning dynamics help us arrive at good solutions or are they inherently different from the type of coordination needed for finding these solutions?

Other examples of applications our theoretical framework needs to

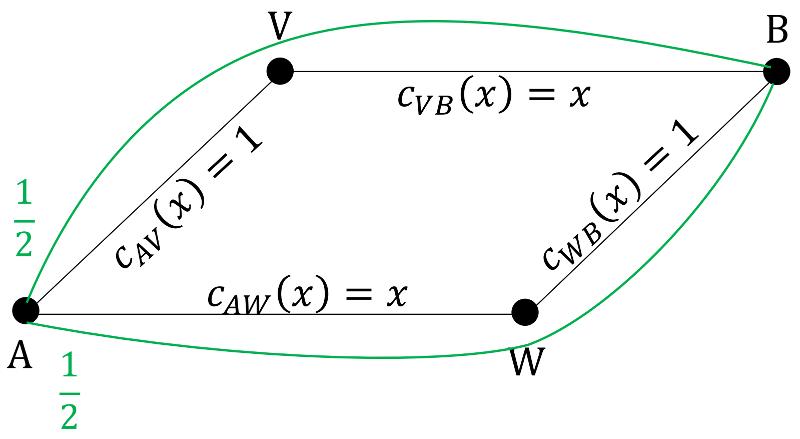

Figure 1: The green allocation is the social optima and an equilibrium. It has a cost of 1.5

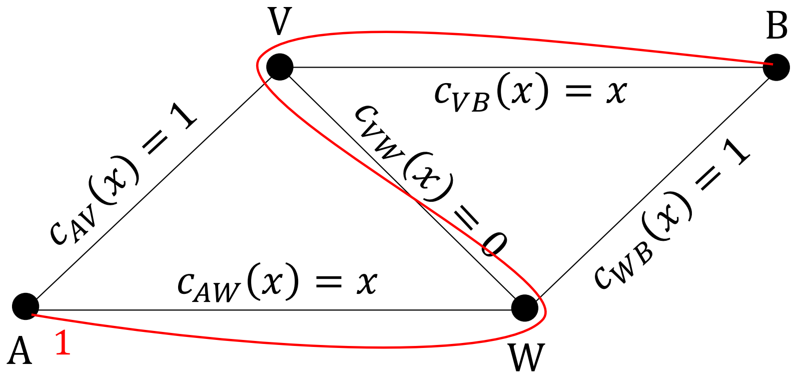

Figure 2: The red allocation is the only equilibrium and costs 2. The allocation in Fig 1 is still a valid solution with cost 1.5.

address include matching drivers and riders in rideshare programs and deciding how to split costs between friends who are sharing a ride.

Online Model

We start with a formal model of learning and learnability in presence of unpredictable, even possibly adversarial, sequences.

Consider a scenario where we are faced with a sequence of incoming instances and we need to make an irrevocable decision about the instance as soon as we see it. To provide robust learning guarantees, we assume that an all-powerful adaptive adversary is choosing the instances and therefore we do not know how well our decisions will perform a priori. An example of this scenario is deciding which route to take to arrive on campus everyday without knowing the traffic en route a priori. At the end of the day, we can look back at the traffic in the city (on all roads) and decide on the route we will be taking tomorrow. What you want is that, over a long period of time, we don't do much worse than if we took the single best route in hindsight.

In another example, consider the setting where everyday we guess whether the stock market index is going to go up or down. You may rely on some recommendation sources: Wall Street Journal, CNN, your coworker, your psychic(!), etc. At the end of the day, you observe whether you were right or wrong. Your goal is not to make many more mistakes compared to your best advisor in hindsight.

This type of decision making will be formalized here and in the next few lectures.

Mistake Bound Model

A simple model for formalizing online decisions is the Mistake Bound model. Given a class of hypotheses $\mathcal{H}$, for $t = 1, 2, 3, ...$

- Receive an $x^t \in \mathcal{X}$.
- Predict $\hat{y}^t \in \{-, +\}$.
- Receive the true label y^t. A mistake is made if $\hat{y}^t \neq y^t$

The goal is to bound the number of mistakes you make, assuming that the sequence of labeled instances (even if produced by an adversary) is consistent with respect to an unknown hypothesis $h^* \in \mathcal{H}$, i.e., for all t, $y^t = h^*(x^t)$. At every round t, the algorithm has access to the sequence of past experience, i.e., $\{(x^1, y^1), (x^2, y^2), \cdots, (x^{t-1}, y^{t-1})\}$.

Definition 1. An algorithm $\mathcal{A}$ learns the hypothesis class $\mathcal{H}$ with mistake bound M, if M is the minimum integer for which the following statement holds: For *any* online sequence $(x^1, y^1), (x^2, y^2), \ldots$ that is consistent with respect to some unknown $h^* \in \mathcal{H}$, $\mathcal{A}$ makes at most M mistakes.

Next, we provide a general-purpose algorithm for learning in the mistake bound model.

One way to look at Mistake Bound model is is that our hypotheses in $\mathcal{H}$ are like "experts". Each expert will make a prediction at every time step t. There is at least one expert who is always right (by consistency) but we don't know who it is. We can observe the history of predictions by every expert to make a decision at time t. Our goal is not to figure out who this perfect expert is, rather, to not make more than M mistakes. In fact, there is a simple and universal algorithm for learning any finite class of hypotheses in the mistake bound model.

Majority/Halving Algorithm Algorithm: At time t, consult all h_i that have yet to make a mistake. Go with their majority vote.

Theorem 1. *The Majority Algorithm has mistake bound* $M \leq \lfloor \log_2 |\mathcal{H}| \rfloor$.

Proof. Note that the mistake bound model assumes that there is some consistent $h^* \in \mathcal{H}$ such that $\forall t, h^*(x^t) = y^t$. Let $S_t = \{h_i : h_i$ has not made a mistake so far$\}$ and let $W_t = |S_t|$. If the majority make a mistake, then at least half of S_t were wrong. Thus, $W_{t+1} \leq 0.5 W_t$. Furthermore, we know $W_t \geq 1$ for all t since we assume there exists a "perfect" expert that is always right. This implies that you can halve W_t at most $\lfloor \log_2(|\mathcal{H}|) \rfloor$ times. □

The Majority algorithm relies heavily on the existence of a perfect expert, i.e., a consistent hypothesis. What if there is no consistent $h^* \in \mathcal{H}$? Can we still provide good guarantees on the mistake bound? This is what we will study in the next lecture.

Lecture 2

Lecture 3

Lecture 4

Lecture 5

Lecture 6

Lecture 7

Lecture 8

Lecture 9

Lecture 10

Lecture 11

Lecture 12

Lecture 13

Lecture 14

Lecture 15

Lecture 16

Lecture 17

Lecture 18

Lecture 19

Lecture 20

Lecture 21

Lecture 22

Lecture 23

Lecture 24

Lecture 25

Lecture 26