CS787: Advanced Algorithms	**Scribe:** Song Zhao
Lecture 19: Multi-Armed Bandits & Streaming Algorithms	**Date:** 04/09/2019

19.1 Multi-Armed Bandits

The setup and protocol of the Multi-Armed Bandit (MAB) Problem is:

n “arms” indexed by $i \in [n]$

At each step $t \in [T]$:

Algorithm chooses an arm i_t

Adversary chooses cost vector c_t

Algorithm incurs cost vector $c_t(i_t)$.

The goal of the algorithm is to minimize:

$$\text{Regret} = \sum_{t \in [T]} c_t(i_t) - \min_{i \in [n]} \sum_{t \in [T]} c_t(i)$$

Recall from previous lecture, the regret of Hedge & FPL algorithms was $O\left(\epsilon T + \frac{\log n}{\epsilon}\right) B$ if cost are bounded by B. In the MAB problem, the only information the algorithm observes after picking an arm i_t at time t is the cost $c_t(i_t)$ of the chosen arm at that time; it does not observe the costs of other arms unlike the online prediction problems discussed previously. We wish to fill in the gap in the information the algorithm has. We can think each step of hedge & FPL as map into a number of different steps of bandit setting, let’s think those as block of steps. For each step we want to try different experts to see what their costs are. We can think about this as a trade off between exploration and exploitation.

Exploration: Trying out different experts to determine their costs.
Exploitation: Following the recommendation of Hedge in order to get good regret.

We can think of this problem as maximizing the reward or equivalently minimizing the regret. The advantage about exploration is that it gives us enough (diverse) information to run hedge or/and FPL, while the challenging thing is incurring a lot of regret. So, we don’t want to explore too often but we want to explore often enough to get sufficient information to hedge.

19.2 An Outline of Bandit Algorithm

Suppose Algorithm A is any prototypical algorithm for the standard online prediction problem. It expects a full cost vector $\widetilde{c_t}$ as input (full feedback information as opposed to bandit feedback information). Denote q_t & p_t as the probability distribution over the arms to be pulled, as generated by bandit algorithm and algorithm A respectively. $\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)$ denotes uniform distribution.

Algorithm 1 Bandit Algorithm layered over Algorithm A

```
1: procedure BANDIT($n$)
2: Run an online prediction algorithm A in background
3:     for all $t \in [T]$, do
4:         w.p. $\lambda$ explore: pick a uniform random arm to play.
5:         w.p. $1-\lambda$ exploit: follow recommendation of A.
6:         Obtain cost vector $\bar{c}_t$
7:         Send $\widetilde{c}_t$ to $A$
8:     end for
9: end procedure
```

It follows that:

$$q_t = \lambda \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right) + (1-\lambda)p_t$$
$$\bar{c}_t = (0, 0, \ldots c_{t_i}; 0 \ldots 0) \text{ w.p. } q_{t_i}$$

To maintain calibration with the prototype algorithm A, we want to define $\widetilde{c}_t$ to satisfy $\mathbb{E}[\widetilde{c}_t] = c_{t_i}$, thus with the average the random algorithm is using, Algorithm A is getting accurate information.

$$\widetilde{c}_{t_i} = \begin{cases} \frac{c_{t_i}}{q_{t_i}} & \text{if i is played} \\ 0 & \text{otherwise} \end{cases}$$

19.3 Cost of algorithm A

Recalling and rewriting the regret of Hedge and FPL cost as $O(\epsilon \min_i \sum_t c_{t_i} + \frac{\log n}{\epsilon} B)$, where B is the bound on the cost components. We understand that the cost of A is at most the cost of best expert under $\widetilde{c}$ plus the regret of A. From the definition of $\widetilde{c}_{t_i}$ and distribution q_t, the bound B placed on $\widetilde{c}$ is $\frac{n}{\lambda}$, observing that q_t is at least $\frac{\lambda}{n}$ for every coordinate.

$$\begin{aligned}
\mathbb{E}[\text{cost of A under } \widetilde{c}] &\leq \mathbb{E}[\min_i \sum_t \widetilde{c}_{t_i}](1+\epsilon) + \frac{n}{\lambda}\frac{\log n}{\epsilon} \\
&\leq \min_i [\mathbb{E}[\sum_t \widetilde{c}_{t_i}]](1+\epsilon) + \frac{n}{\lambda}\frac{\log n}{\epsilon} \\
&\leq \min_i [\sum_t c_{t_i}](1+\epsilon) + \frac{n \log n}{\lambda \epsilon}
\end{aligned}$$

Cost of Bandit algorithm:

$$\begin{aligned}
\text{Cost of Bandit} &\leq \sum_t [\lambda \cdot 1 + (1-\lambda) \cdot (\text{cost of A at t})] \\
&\leq \lambda T + \text{cost of A} \\
&\leq \min_i [\sum_t c_{t_i}](1+\epsilon) + \lambda T + \frac{n \log n}{\lambda \epsilon} \\
&\min_i [\sum_t c_{t_i}] + \lambda T + \epsilon T + \frac{n \log n}{\lambda \epsilon}
\end{aligned}$$

Regret of Bandit Algorithm is $\lambda T + \epsilon T + \frac{n \log n}{\epsilon \lambda}$. If we choose $\lambda = \epsilon$, the regret is $2\lambda\epsilon + \frac{n \log n}{\epsilon^2}$. If we choose $\epsilon = (\frac{n \log n}{T})^{\frac{1}{3}}$, we get $O(T^{\frac{2}{3}}(n \log n)^{\frac{1}{3}})$. Notice the slightly worse dependence in T and n compared to the FTPL/Hedge regret bound of $O(\sqrt{T \log n})$ in the standard online prediction scenario. It turns out that one can obtain a tighter bound of $O(\sqrt{T\sqrt{n \log n}})$ by using Hedge in the Bandit feedback case.

19.4 Streaming Algorithm

Streaming algorithms are like online algorithm in which we receive data in an online fashion. These are essentially data collection algorithm, to summarize information in a small data structure or generate summary in order to use in later analysis. In most cases while talking about streaming algorithm we only do a single pass on data, and we have a small storage space (relative to input size) to store the information. These have immense applications in applications like packet routing, web search trending analytics etc.

Proto-typical setting:
Universe U with size n,
Stream of elements $i_1, i_2, i_3, ...i_m$, with $i_j \in U \ \forall \ j \in [m]$ where m, n can be large,
f_x is the frequency of elements $x \in U = |\{t : i_t = x\}|$.

Some example quantities we want to store:
number of distinct elements,
heavy hitters,
moments of frequency $(\sum_{x \in U} f_x^p)^{\frac{1}{p}}$.

In most scenarios, the exact computation of the summary statistics for the data stream requires linear storage for exact computation. With the size of data stream or universe growing large, linear storage in potentially very expensive and sometimes strategically limited. This motivates our interest in approximations of summary statistics in a small amount of memory space since we cannot (don't) record all the information. In particular, we look for (ϵ, δ)-approximations (PAC style). ϵ refers to multiplicative approximation window tolerance, δ is the probability of failure. The intention here is with probability of $1 - \delta$ the estimator is $(1 \pm \epsilon)$ of true value.