1 Full versus Partial Information

In online full information setting, we are able to observe the historical costs of choosing any of the experts. More formally, at time $t$, after taking action $i \in [n]$ we know $c^i(j)$ for all actions $j \in [n]$. Such an assumption makes sense in some problems such as classification where by observing adversary’s choice of labeled instances $(x_t, y_t)$ we can calculate the cost of each classifier $h \in H$ by taking $c^i(h) = \mathbb{1}(h(x_t) \neq y_t)$. On the other hand, many examples involve situations where we only observe the cost function for some of the experts, e.g., only the expert that we followed at that time step. Examples of this include online route optimization where we may only observe the total travel time for the route we took, or observe the traffic on different parts of this route.

This is referred to as partial information in online learning. Formally, when playing action $i^t$, we may observe costs of experts $c^i(j)$ for $j \in E_i$. When $E_i = [n]$, this is just the full information setting. When $E_i = \{i^t\}$, the online learning problem is commonly referred to Adversarial Multi-Armed Bandits. In more detail, in the adversarial multi-armed bandit setting, we have $n$ experts and at each time step we choose one of the $n$ experts and we only observe the cost of that expert.

In the following sections we will see an algorithm for the bandit setting and we will derive regret bounds for the algorithm. Since, we don’t get to observe the cost of experts we didn’t play in the bandit setting, we can never rule out picking an expert completely and we need to have some probability of picking every expert at any given point. This is referred to as exploration and allows us to uncover potentially great experts that we have not played in the past. On the other hand, similar to the full information setting, we also want to take experts whose past performances have been good with higher probability. This is referred to as exploitation. Therefore any bandit algorithm needs to balance exploration and exploitation of experts.

2 An Easy Algorithm for Adversarial Multi-Armed Bandits

In this section, we show that any no-regret algorithm from the full information setting can be altered to work for the bandit setting as well, albeit with slightly worse regret guarantees. For ease of presentation, in the following we make use of the Randomized Weighted Majority (RWM) Algorithm that is no-regret in the full information setting (See Section 2.4 of Lecture 11). The high level idea for this algorithm is as follows:

At iteration $t$,
Learner vs Adversary

\[ i^t \sim \tilde{q}^t \]

Algorithm for Bandit Setting

\[ \tilde{p}^t \]

Random Weighted Majority (n experts)

Algorithm 1

- Use probabilities suggested by a no-regret algorithm in a full information setting (such as RWM). Let us denote these probabilities by vector \( \tilde{p}^t \).
- Sample an expert \( i^t \) from a distribution \( \tilde{q}^t \), where \( \tilde{q}^t = (1 - \gamma) \tilde{p}^t + \gamma (\frac{1}{n}, \ldots, \frac{1}{n}) \). This is equivalent to taking \( i^t \sim \tilde{p}^t \), \( (1 - \gamma) \) fraction of the time, and taking \( i^t \sim [n] \) uniformly the rest of the time.
- Let the full-information algorithm update the weights of the experts. To do this, the algorithm needs to see the utilities (that is either the loss or reward) of every expert. For this we create a hypothetical utility vector as follows:

\[ \hat{u}^t = \left( 0, \ldots, 0, \frac{u_{i^t}^t}{q_{i^t}^t}, 0, \ldots, 0 \right) \]

Here \( u_{i^t}^t \) is the utility of the one expert \( i^t \) that is chosen at time \( t \) and \( q_{i^t}^t \) is the probability of choosing expert \( i^t \). The vector \( \hat{u}^t \) is then fed in to the full-information algorithm (such as RWM) to update the weights of the experts accordingly.

This high level idea is summarized in Figure 1. A few remarks:

1. At a high level, for the update rule to work well for the full information algorithm, we would want \( \hat{u}^t \) to represent the full vector of utilities \( \bar{u}^t \) in expectation. That is, \( E_{i^t \sim q_i^t} [\hat{u}^t] = (u_1^t, \ldots, u_n^t) \). We prove this later in this section.

2. The version of RWM that we described in Lecture 11 assumes that the utilities are in range \([0, 1]\). On the other hand, we are creating an expert with utility \( \frac{u_{i^t}^t}{q_{i^t}^t} \in [0, n/\gamma] \). Therefore, we need to adapt the regret bound of RWM to scale with this larger range of utilities.

Algorithm 1 shows the algorithm that corresponds to using Figure 1. In this version, we consider utilities as rewards to the algorithm instead of costs. This results in \(+\epsilon\) used in update rule of Algorithm 1 in line 6.
Algorithm 1 Reduction to Full Information

1: Initialize $w^1_1, \ldots, w^n_1 = 1$
2: for time $t = 1, \ldots, T$ do
3: $\forall i \in [n], q^t_i \leftarrow (1 - \gamma) \frac{w^t_i}{\sum_{j=1}^n w^t_j} + \frac{2}{n}$
4: Pick $i^t \sim \vec{q}^t$ and receive $u^t_i \in [0, 1]$
5: Let $\hat{u}^t_i = [0, \ldots, 0, \frac{u^t_i}{q^t_i}, 0, \ldots, 0]$ \% (Approximately equal to $w^t_i \exp \left( \frac{\epsilon u^t_i}{n} \cdot \gamma \right)$
6: $\forall i \in [n], w^{t+1}_i \leftarrow w^t_i \left( 1 + \frac{\epsilon \gamma}{n} \right)$
7: end for

Theorem 2.1 (Reduction to Full Information). With $\gamma = \epsilon$, Algorithm 1 guarantees that

$$E \left[ \sum_{t=1}^T u^t_i \right] \geq E \left[ \max_j \sum_{t=1}^T u^t_j \right] (1 - \gamma)^2 - \frac{n}{\gamma^2} \ln(n)$$

It follows that $\text{REGRET} \leq T^{2/3} \left( n \ln(n) \right)^{1/3}$ if $\gamma = \left( \frac{n \ln(n)}{T} \right)^{1/3}$.

We will prove the theorem by combining the proofs of several facts

Fact 2.2. $\forall j \in [n], \hat{u}^t_j$ is an unbiased estimator of $u^t_j$.

Proof. This follows from:

$$E_{i^t \sim \vec{q}^t} \left[ \hat{u}^t_j \right] = q^t_j \cdot \frac{u^t_i}{q^t_i} + (1 - q^t_j) \cdot 0 = u^t_j \text{.} \quad \square$$

Let $\text{OPT}_{\text{RWM}} = \max_j \sum_{t=1}^T \hat{u}^t_j$ be the best utility in hindsight according to RWM and $\text{OPT} = \max_j \sum_{t=1}^T u^t_j$ be the true best in hindsight utility. Next we show that $\text{OPT}_{\text{RWM}}$ is more competitive than $\text{OPT}$.

Fact 2.3. $E_{i^t \sim \vec{q}^t} [\text{OPT}_{\text{RWM}}] \geq \text{OPT}$.

Proof. We will use Jensen’s inequality in proving this fact for asserting that expectation of a maximum is greater than maximum of an expectation. This is because the maximum accounts for the random variable that is taken expectation of, when the maximum is inside of the expectation.

$$E_{i^t \sim \vec{q}^t} [\text{OPT}_{\text{RWM}}] = E_{i^t \sim \vec{q}^t} \left[ \max_j \sum_{t=1}^T \hat{u}^t_j \right] \geq \max_j E_{i^t \sim \vec{q}^t} \left[ \sum_{t=1}^T \hat{u}^t_j \right] = \max_j \sum_{t=1}^T u^t_j = \text{OPT},$$

where the penultimate transition is by Fact 1.2. \quad \square

Fact 2.4.

$$\sum_{t=1}^T \hat{p}^t \cdot \hat{u}^t \geq (1 - \epsilon) \text{OPT}_{\text{RWM}} - O \left( \frac{1}{\epsilon} \ln(n) \frac{n}{\gamma} \right)$$
Proof. This fact follows from Theorem 1.1 of Lecture 12 but using utilities instead of costs. Note that the utilities used by RWM in the bandit algorithm are in \([0, n/\gamma]\) instead of the usual \([0, 1]\), leading to a scaling factor in the last term of the expression above.

Fact 2.5. \(u^t_i \geq (1 - \gamma) \cdot p^t_i \cdot \hat{u}^t_i, \) i.e., The true reward is a large fraction of RWM’s expected reward.

Proof. Let \(i = i^t\) for readability purposes. Then,

\[
u^t_i = \hat{u}^t_i q^t_i = p^t_i \cdot \hat{u}^t_i \cdot \frac{q^t_i}{p^t_i} \geq p^t_i \cdot \hat{u}^t_i (1 - \gamma).
\]

Here, \(p^t_i \cdot \hat{u}^t_i\) is RWM’s reward and \(\frac{q^t_i}{p^t_i}\) is a scaling factor, which is small as shown.

Proof of Theorem 2.1. Combining all facts together, the bandit algorithm’s reward is:

\[
\mathbb{E}_{i^t \sim \vec{q}^t} \left[ \sum_{t=1}^T u^t_i \right] \geq (1 - \gamma) \mathbb{E}_{i^t \sim \vec{q}^t} \sum_{t=1}^T [p^t_i \cdot \hat{u}^t_i] \quad \text{(By Fact 1.5)}
\]

\[
\geq (1 - \gamma)(1 - \epsilon) \mathbb{E}[\text{OPT}_{\text{RWM}}] - (1 - \gamma) \frac{1}{\epsilon \gamma} n \ln(n) \quad \text{(By Fact 1.4)}
\]

\[
\geq (1 - \gamma)^2 \mathbb{E}[\text{OPT}] - \frac{n}{\gamma^2} \ln(n) \quad \text{(By Fact 1.3)}.
\]

3  EXP3 Algorithm: A Better Bandit Algorithm

While Algorithm 1 is a no-regret algorithm, its regret bound dependence \(T^{2/3}\) is far from optimal. Below, we introduce EXP3 that enjoys better regret guarantees. Interestingly, EXP3 is essentially equal to Algorithm 1, with the exception that its update rule does not have an additional \(\gamma\) dependence (See line 6). This requires a much more careful analysis of EXP3 that give us the following regret bound.

**Algorithm 2** EXP3: Exponential Weights for Exploration and Exploitation

1: Initialize \(w^1_1, \ldots, w^1_n = 1\)
2: for time \(t = 1, \ldots, T\) do
3: \(\forall i \in [n], q^t_i \leftarrow (1 - \gamma) \frac{w^t_i}{\sum_{j=1}^n w^t_j} + \frac{\gamma}{n}\)
4: Pick \(i^t \sim \vec{q}^t\) and receive \(u^t_{i^t} \in [0, 1]\)
5: Let \(\hat{u}^t = \left[0, \ldots, 0, \frac{w^t_i}{q^t_i}, 0, \ldots, 0\right]\)
6: \(\forall i \in [n], w^{t+1}_i \leftarrow w^t_i \exp \left(\frac{\epsilon u^t_i}{n}\right)\)
7: end for
Theorem 3.1. (EXP3 Regret). When $\gamma = \epsilon$, EXP3 algorithm guarantees

$$E \left[ \sum_{t=1}^{T} u_{it}^{t} \right] \geq E \left[ \max_{j} \sum_{t=1}^{T} u_{jt}^{t} \right] (1 - \gamma) - \frac{n}{\gamma} \ln(n).$$

It follows that Regret (EXP3) $\leq \sqrt{Tn \ln(n)}$ if $\gamma = \sqrt{\frac{\ln(n)}{T}}$.

EXP3 requires a more careful analysis (i.e. proving Theorem 3.1) than we have done for Theorem 2.1. We will not prove EXP3’s regret bound in this lecture and instead refer the interested reader to Auer et al. [2002].

Let us remark that the regret bound of Theorem 3.1 is nearly tight in both its dependence on $T$ and its dependence on $n$. Note that in the partial information setting our regret bounds grow with $\sqrt{\ln(n)}$ while in the partial information setting they grow with $\sqrt{n}$. This polynomial dependence in the partial information setting is unavoidable because we need to explore nearly all experts to uncover potentially excellent ones that have never been played before.

4 Weak Learners

Definition 4.1 (Strong Learner in the standard realizable PAC learnability setting). A strong learner can, for any $D$, for any $\epsilon > 0$, for any $\delta > 0$, learn a classifier $h$ such that with probability $(1 - \delta)$,

$$\text{err}_D(h) \leq \epsilon.$$

Remark 4.2. Suppose learnability is defined as follows: For any $D$, for any $\epsilon$, and for $\delta = \frac{1}{2}$, a learner can learn a classifier $h$ such that with probability $\frac{1}{2}$,

$$\text{err}_D(h) \leq \epsilon.$$

We showed that this notion of learnability is equivalent to strong learnability in homework 2. This implies the exact value of $\delta$ does not matter, it can always be boosted to meet our requirements. Now we see what role $\epsilon$ plays by defining weak learners.

Definition 4.3 (Weak learner). A weak learner can for any $D$, some fixed $\gamma > 0$, for any $\delta > 0$, learn a classifier $h$ such that with probability $(1 - \delta)$,

$$\text{err}_D(h) \leq \frac{1}{2} - \gamma.$$

Note that if a learner ignores the samples and guesses $+1, -1$ uniformly at random, it has an error of exactly $1/2$. A weak learner is one that can essentially beat this random prediction by a small $\gamma > 0$.

Freund [1995] and Schapire [1990] showed that weak learnability is equivalent to strong learnability, i.e., there is a boosting algorithm that uses a weak learner on an adaptively designed short sequence of distributions and gives a strong learner.
4.1 Warmup

In this section, we see how a special class of weak learners can be boosted to strong learners. These weak learners are special, because in addition to predicting $+1$ or $-1$, they can also tell us that they don’t know the answer. To make sure that these learners do “learn”, there must be a small fraction of the data where they can make better than random predictions.

Formally consider a classifier $g(x) \in \{+1, -1, \text{not sure}\}$. For any $D$ some $\epsilon < 1/2$ and some $\epsilon'' > 0$, we say that $g(\cdot)$ can $(1 - \epsilon', \epsilon)$-learn if

1. On at most $1 - \epsilon'$ fraction of data, $g$ says \textit{not sure}.

2. Conditioned on making a prediction, $g$ has error of at most $\epsilon$.

Let us first show that $g(\cdot)$ that $(1 - \epsilon', \epsilon)$-learns is a special type of weak learner. Let $h(x) \in \{-1, +1\}$ be a classifier such that on input $x$ randomly outputs $+1$ or $-1$ with equal probability if $g(x) = \text{not sure}$. Otherwise, $h(x) = g(x)$ which is either $+1$ or $-1$. Note that

$$\text{err}_D(h) \leq \frac{1}{2}(1 - \epsilon') + \epsilon' \epsilon \leq \frac{1}{2} - \gamma,$$

when $\gamma = \epsilon'(0.5 - \epsilon) > 0$. So $h$ is a weak learner.

In the next lecture, we show how these weak learners can be boosted to make a strong learner.

References

