Machine Learning Theory
Lecture 15

Nicholas Harvey

October 29, 2018

1 Setup

There is an agent who must make a sequence of online decisions. There are a sequence of discrete time steps; in each time step, the agent must make a decision and incur the cost of that decision. The agent has limited or no forehand knowledge of what that cost will be. Of course, the goal is to incur as little cost as possible. Usually, at the end of each time step, the agent is told the cost of each choice he could have made. (However, in "bandits" scenarios, the agent is only told the cost of the choice he actually made.)

A typical setup is that there are n experts. At each time step i, each expert gives some "advice" (which may or may not be captured mathematically). The costs at time i are specified by a vector $c_i = (c_{i,1}, ..., c_{i,n}) \in \mathbb{R}^n$, where $c_{i,j}$ is the cost of following expert j at time i. The agent must (without knowing c_i) pick an expert j whose advice to follow, then incur the cost $c_{i,j}$. The vector c_i is then revealed to the agent.

2 Finding a perfect expert: the halving algorithm

Suppose that decision at time step i the goal is for the agent to "predict" the value of a variable $v_i \in \{0, 1\}$. Each expert j provides a "prediction" $p_{i,j} \in \{0, 1\}$, which is his prediction for v_i. After the agent makes his decision, the true value of $v_i \in \{0, 1\}$ is revealed. Expert j has cost 1 if his prediction was wrong, otherwise cost 0 (i.e., $c_{i,j} = 1_{p_{i,j} \neq v_i}$).

Theorem 2.1. Suppose that there is some expert that is always correct. Then Algorithm 1 has total cost at most $\lg n$.

Note. The costs here can be adversarially chosen. The costs c_i can even depend on the expert j chosen by the algorithm in round i. In fact, since the algorithm is deterministic, the adversary can completely determine the experts chosen by the algorithm.

Proof. Every time the algorithm makes a mistake, the majority vote over S was incorrect, so the size of S shrinks by a factor of at least 2. Since we assume that some expert is always correct, the set S never becomes empty. ■

Algorithm 1 The halving algorithm.

```
1: procedure HalvingAlgorithm
2:     Let $S = [n]$ (the set of experts predicting correctly so far)
3:     for $i \leftarrow 1, 2, ...$ do
       ▷ Receive predictions $p_i$
4:         Let $m$ be the majority prediction of experts in $S$. Let $j$ be an expert that predicted $m$.
5:         Follow the advice of expert $j$, incurring cost $c_{i,j}$
           ▷ Receive cost vector $c_i$ (i.e., receive the value $v_i$)
6:         Remove all $j$ from $S$ whose predictions were incorrect
```

3 No perfect expert: the weighted majority algorithm

Now let us consider the more difficult case in which there is no guarantee of a perfect expert. So, rather than discarding experts that make a mistake, the agent will associate a "weight" with each expert. The weight serves as the "credibility" of the expert, i.e., the agent's confidence in that expert's predictions. The main idea of the multiplicative-weight update method is to decrease an agent's weight multiplicatively whenever he makes a mistake.

The weight associated with expert j at time i is $y_{i,j}$. Initially all weights are $y_{1,j} = 1$. In iteration i, if expert j is correct he incurs a cost $c_{i,j} = 0$. Otherwise, if expert j is incorrect he incurs a cost of $c_{i,j} = 1$, and his weight is multiplied by $e^{-\eta}$. His total cost incurred by the start of time t is $\sum_{i=1}^{t-1} c_{i,j}$, and his weight is $y_{t,j} = \exp(-\eta \sum_{i=1}^{t-1} c_{i,j})$.

Algorithm 2 The weighted majority algorithm.

```
1: procedure WeightedMajority
2:     Let $y_1 = (1, ..., 1)$.
3:     for $i \leftarrow 1, 2, ...$ do
       ▷ Receive predictions $p_i$
4:         If $\sum_{\text{expert } j \text{ predicting } 1} y_{i,j} \geq \frac{1}{2} \sum_j y_{i,j}$, let $m = 1$; otherwise $m = 0$.
5:         Let $j$ be an expert that predicted $m$.
6:         Follow the advice of expert $j$, incurring cost $c_{i,j} \in \{0, 1\}$
           ▷ Receive cost vector $c_i \in \{0, 1\}^n$ (i.e., receive the value $v_i$)
7:         for $j \leftarrow 1, ..., n$ do
8:             $y_{i+1,j} = \begin{cases} y_{i,j} & \text{(weight is unchanged if expert } j \text{ predicted correctly)} \\ y_{i,j} e^{-\eta} & \text{(weight is decreased if expert } j \text{ made a mistake)} \end{cases}$
```

Theorem 3.1. Let $\epsilon = 1 - e^{-\eta} \approx \eta - \eta^2/2$ (so $\eta = \ln \frac{1}{1-\epsilon} \approx \epsilon + \epsilon^2/2$). Assume that $\epsilon \leq 1/2$. Consider any time step t. Let A be the total cost (number of mistakes) of the algorithm. Let j^* be the expert with minimum total cost (number of mistakes). Then

$$A \;\leq\; 2(1+\epsilon) \sum_{i=1}^{t} c_{i,j^*} + \frac{2 \ln n}{\epsilon}.$$

References. [1, Lemma 1.3].

Note. The costs here can be completely adversarially chosen. Specifically, the costs c_i can depend on the expert j chosen by the algorithm in all rounds $1, ..., i$. (Specifically, in round i, the adversary is allowed to choose the costs c_i after the algorithm has already chosen the expert j to follow.) In fact, since the algorithm is deterministic, the adversary can completely determine the entire sequence of experts chosen by the algorithm.

Remark. Even as $\eta \to 0$, the algorithm's cost does not get arbitrarily close to the cost of the best expert: the theorem cannot prove $A < 2\sum_{i=1}^{t} c_{i,j^*}$. To trace this factor 2's origin, the relevant occurrences of 2 are highlighted in the proof. See Section 4 for further discussion.

Remark. Many expositions of the multiplicative weights algorithm scale the weights by $(1 - \epsilon c_{i,j})$ or $(1-\epsilon)^{c_{i,j}}$ instead of $\exp(-\eta c_{i,j})$. The proof can be easily modified to use those scalings instead. If the weights are scaled by $(1 - \epsilon c_{i,j})$, then note that setting $\epsilon = 1$ gives the Halving Algorithm.

Proof Idea. The proof has two main ideas.

(*a*) The cost incurred by the algorithm in iteration i can be related to the total change in the weights during iteration i.

(*b*) The total weight in iteration t provides a useful upper bound on the total cost that any expert has incurred up to iteration t.

Proof.

Step (a): We wish to analyze the cost of the algorithm in iteration i. The main fact that we know is: whenever the algorithm makes a mistake, the weighted majority vote was incorrect, i.e., $\sum_{\text{mistaken } j} y_{i,j} \geq \frac{1}{2} \sum_j y_{i,j}$. Since the mistaken experts' weights are decreased by a factor $e^{-\eta}$, the main consequence is that

$$\begin{aligned}
\sum_j y_{i+1,j} &= \sum_{\text{mistaken } j} y_{i,j} e^{-\eta} + \sum_{\text{correct } j} y_{i,j} \\
&= \sum_j y_{i,j} \; - \; (1 - e^{-\eta}) \underbrace{\sum_{\text{mistaken } j} y_{i,j}}_{\geq \sum_j y_{i,j}/2} \\
&\leq \left(1 - \frac{1 - e^{-\eta}}{2}\right) \sum_j y_{i,j} \\
&= \left(1 - \frac{\epsilon}{2}\right) \cdot \sum_j y_{i,j}.
\end{aligned}$$

Since the sum of the y values decreases by $1 - \frac{\epsilon}{2}$ every time the algorithm makes a mistake,

$$\sum_j y_{t+1,j} \;\leq\; \left(1 - \frac{\epsilon}{2}\right)^{A} \cdot \sum_j y_{1,j} \;\leq\; \exp\left(-\frac{\epsilon}{2} A\right) \cdot n. \tag{3.1}$$

That is, if the algorithm has made many mistakes, the sum of the weights must be small.

Step (b): The next idea is that the sum of the weights is actually a useful proxy for the largest weight, which corresponds to the expert with lowest cost. Indeed, it is not hard to see that $y_{t,j^*} = \max_j y_{t,j} \in [1/n, 1] \cdot \sum_j y_{t,j}$. Since the number of mistakes by expert j^* relates to the logarithm of this quantity, this multiplicative factor $1/n$ only leads to an additive error of $O(\log n)$ in estimating the cost of expert j^*. The inequality that we will use is:

$$\exp\Big(-\eta \underbrace{\sum_{i=1}^{t} c_{i,j^*}}_{\substack{\text{total cost of}\\ \text{expert } j^*}}\Big) \;=\; y_{t+1,j^*} \;\leq\; \sum_j y_{t+1,j}. \tag{3.2}$$

Combining (a) and (b): To relate the cost of expert j^* and the cost of the algorithm, we combine (3.1) and (3.2) to obtain

$$\exp\Big(-\eta\sum_{i=1}^{t} c_{i,j^*}\Big) \;\leq\; \sum_j y_{t+1,j} \;\leq\; \exp\Big(-\frac{\epsilon}{2}A\Big)\cdot n.$$

Taking the log and rearranging,

$$A \;\leq\; \frac{2\eta}{\epsilon}\sum_{i=1}^{t} c_{i,j^*} + \frac{2\ln n}{\epsilon} \;=\; \frac{2}{\epsilon}\ln\Big(\frac{1}{1-\epsilon}\Big)\sum_{i=1}^{t} c_{i,j^*} + \frac{2\ln n}{\epsilon}.$$

Applying Claim 3.2 concludes the proof. ∎

Claim 3.2.

$$\log\frac{1}{1-x} \;\leq\; x + x^2 \qquad \forall x \in [0, 1/2]$$

Remark. In this analysis, the current total weight of the experts, namely $\sum_j y_{i,j} = \|y_i\|_1$, plays an important role. One may view this quantity as a "potential function" that captures the state of the algorithm. Step (a) relates this potential to the cost of the algorithm. Step (b) relates this potential to the cost of the best expert.

4 Lower Bound

Suppose that c_i is allowed to depend on the expert j chosen by the algorithm; the algorithm may be either deterministic or randomized. (Equivalently, we may consider an ***adaptive offline adversary***. In this scenario, the cost c_i only depends on the expert j chosen by the algorithm in rounds 1 through $i-1$; however, the adversary knows the algorithm and all random bits used by the algorithm. In this case, the adversary can predict the expert j chosen by the algorithm in round i.)

Claim 4.1. Consider the case of two experts (i.e., $n = 2$). For any algorithm A, the adversary can ensure after t rounds $A \geq t$ and $\sum_{i=1}^{t} c_{i,j^*} \leq t/2$.

We may conclude that the constant factor 2 is necessary in Theorem 3.1.

Proof. The experts are defined so that expert 0 always predicts $p_{i,0} = 0$ and expert 1 always predicts $p_{i,1} = 1$. After the algorithm chooses which expert to follow, the adversary chooses the value v_i so that the algorithm is incorrect. Thus the algorithm's total cost will be t. However, at each time step, one expert always predicts correctly. Thus, the better of the two experts has total cost at most $t/2$. ■

References

[1] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3–4), 2015.