COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Qasim Nadeem

Lecture #22
April 25, 2018

1 Introduction and Recap

In the previous lecture, we introduced an algorithm to minimize log-loss in an on-line model. We called it the Bayes algorithm because it uses Bayes rule.

Summary of the algorithm:

- # experts: N
- π is our **prior** over the experts. $\pi_i \geq 0$, $\forall i$ and $\sum_i \pi_i = 1$. If we have a clue as to which expert is going to perform the best, we can distribute the prior intelligently, else we set it uniformly.
- $w_{1,i} = \pi_i$, for all $i = 1, \ldots, N$
- For $t = 1, \ldots, T$
  - expert i predicts distribution $p_{t,i}$ over $\mathcal{X}$
  - learner then chooses the distribution q_t:

$$q_t(x) = \sum_{i=1}^{N} w_{t,i} p_{t,i}(x)$$

  - we observe $x_t \in \mathcal{X}$
  - the loss incurred in the round is $-\ln(q_t(x_t))$
  - learner updates $\forall i : w_{t+1,i} = \frac{w_{t,i} p_{t,i}(x_t)}{norm}$

We then proved the following inequality, which intuitively says that the log-loss of the Bayes algorithm is bounded by the log-loss of the best expert plus a regret term based on the prior:

$$-\sum_t \ln(q_t(x_t)) \leq \min_i \left[-\sum_t \ln(p_{t,i}(x_t) - \ln(\pi_i) \right]$$

Note that if $\pi_i = 1/N$ for all i, then we simply get a regret of $\ln(N)$.

2 Tracking the shifting best expert

Now we tackle a more challenging setting where across all the time steps $t = 1, \ldots, T$, the best expert can change over time and we want to track the best *shifting* expert. As an example, maybe for the interval $t = 1, \ldots, t_0$, expert e_0 is the best, then for the interval $t = (t_0 + 1), \ldots, t_1$, expert e_1 is the best, and so on. We let k be the the number of *switches* of best experts over the entire time sequence.

2.1 First attempt: Naively apply Bayes algorithm

We have N base experts that are given to us. We can imagine creating several *meta-experts*, each of which tracks a particular k switching of best experts in the time sequence. We can then directly apply the Bayes Algorithm using the meta-experts as our set of experts, instead of the base experts. To get a regret bound, we need to count the number of meta-experts.

A total of k switches implies $k+1$ blocks: so we have a choice of at most T^k ways to place switches. And we have a choice of N experts as the best one for each block. And so, we get the bound:

$$\#\text{ meta-experts} = M \leq N^{k+1}T^k$$

Now simply apply and analyze Bayes algorithm with a uniform prior, $\pi_i = \frac{1}{M}\ \forall i$, and the M meta-experts. And we get the regret bound:

$$\ln(M) \leq (k+1)\cdot\ln(N) + k\cdot\ln(T)$$

$$= \ln(N) + k(\ln(N) + \ln(T))$$

where $\ln(N)$ is regret with just base experts, and our regret increases at a cost of $\ln(N) + \ln(T)$ per switch.

We are happy with the regret. However, if k is large, there are exponentially many meta-experts to maintain, which is computationally infeasible. We look at an alternative technique, which has similar regret bound but is computationally efficient. Note that above we had one meta-expert for each switching sequence with at most k switches. Now instead, we will play with the prior π. Also, in this new construction, we will track one meta-expert for *every* switching sequence, not just sequences with at most k switches.

2.2 Second attempt: Make computationally efficient

Let a meta-expert be defined by a vector of T base experts:

$$\mathbf{e} = \langle e_1, e_2, ..., e_T\rangle,\ e_t \in \{1, \ldots, N\}$$

The vector indicates the base expert e_t that this meta-expert predicts with at time step t. Intuitively, we are going to give low prior to meta-experts which switch their chosen expert often, and high prior to ones that do not. So as illustration, $[1,1,1,7,7,7,7,2,2]$ will have a higher prior than $[1,2,3,4,4,5,1,7,2]$.

We will now describe a process of generating a random meta-expert. And the prior we will give each meta-expert will be the probability that our described random process generates that meta-expert. More formally:

$$\pi_i = \pi(\mathbf{e}) = Pr\left[\mathbf{e}^\star = \mathbf{e}\right]$$

Now we specify the following meta-expert generating process (denote the meta-expert being generated as $\mathbf{e}^\star$):

(i) Pick $e_1^\star$ uniformly: $Pr\left[e_1^\star = i\right] = \frac{1}{N}$

(ii) Intuitively, we want probability of switching to be low. Thus, with probability $(1-\alpha)$ we do not switch (we'll pick α later), and we distribute remaining α probability uniformly over generation of all other experts:

$$Pr\left[e^\star_{t+1}|e^\star_t\right] = \begin{cases} 1-\alpha & \text{if } e^\star_{t+1} = e^\star_t \\ \frac{\alpha}{N-1} & \text{otherwise} \end{cases}$$

What can we say about the regret of any particular meta-expert with exactly k switches generated by the above process? Well, the regret term is contributed to the bound by the prior. So, let's look at the prior we associate to such a meta-expert:

$$-\ln(\pi(\mathbf{e})) = -\ln\left[\frac{1}{N}(1-\alpha)^{T-k-1}\left(\frac{\alpha}{N-1}\right)^k\right]$$

The above directly follows from the way a meta-expert is generated. There is $\frac{1}{N}$ probability of generating e_1, $\frac{\alpha}{N-1}$ probability of each of the k switches, and $(1-\alpha)$ probability of each of the remaining $T-k-1$ non-switches. Multiplying these gives us the above.

Choosing $\alpha = \frac{k}{T-1}$, and plugging in above gives us

$$= \ln(N) + k\cdot\ln\left[\frac{(N-1)(T-1)}{k}\right] - (T-k-1)\cdot\ln\left[1-\frac{k}{T-1}\right]$$

Note that the third term above: $(T-k-1)\cdot\ln\left[1-\frac{k}{T-1}\right] \approx k\cdot(1-\frac{k}{T-1})$. Plugging this in the above expression will give us roughly the same regret bound as what we got from our earlier construction in section 2.1:

$$\approx \ln(N) + k\cdot[\ln(N)+\ln(T)]$$

2.3 Deriving the Weight-Share Algorithm

We now derive the algorithm that will allow us to achieve the above, without explicitly maintaining the exponential in k many meta-experts. We will reuse most of the notation we used in the previous lecture for the analysis of Bayes algorithm.

Some helpful notation and algebra:

- Recall that $Pr\left[\mathbf{e}^\star = \mathbf{e}\right] = \pi(\mathbf{e})$
- Then, $Pr\left[x_t \,|\, x_1^{t-1}, \mathbf{e}^\star = \mathbf{e}\right]$
  = prediction of x_t of the meta-expert $\mathbf{e}$ at time t
  = prediction of base expert e_t on x_t
- And so equivalently,
$$Pr\left[x_t \,|\, x_1^{t-1}, e^\star_t = i\right] = p_i(x_t \,|\, x_1^{t-1}) \tag{1}$$

We can now derive how the **weight share algorithm** can compute each of the requisite terms.

Recall that in the Bayes algorithm, we defined

$$q(x_t \,|\, x_1^{t-1}) = Pr\left[x_t \,|\, x_1^{t-1}\right]$$

We use marginalization on the above to get:

$$= \sum_{i=1}^{N} Pr\left[e_t^\star = i \,|\, x_1^{t-1}\right] \cdot Pr\left[x_t \,|\, e_t^\star = i, x_1^{t-1}\right] = \sum_i v_{t,i} \cdot p_i(x_t \,|\, x_1^{t-1}) \tag{2}$$

Note that we have simply renamed the first term in the sum above to $v_{t,i}$, and the second term is simplified using (1) above. The question to address now is how we should compute each of the $v_{t,i}$ terms.

At time step 1,

$$v_{1,i} = Pr\left[e_1^\star = i \,|\, \text{nothing}\right] = Pr\left[e_1^\star = i\right] = \frac{1}{N}$$

Generally,

$$v_{t+1,i} = Pr\left[e_{t+1}^\star = i \,|\, x_1^t\right] = \sum_{j=1}^{N} Pr\left[e_t^\star = j \,|\, x_1^t\right] \cdot Pr\left[e_{t+1}^\star = i \,|\, e_t^\star = j, x_1^t\right]$$

Consider the second term in the above sum: The event $e_{t+1}^\star = i$ is conditionally independent of x_1^t given $e_t^\star = j$. Thus, the second term is simply $(1-\alpha)$ if $i = j$ and $\frac{\alpha}{N-1}$ otherwise.

Consider the first term in the sum: $Pr\left[e_t^\star = j \,|\, x_1^t\right]$. This is equal to $= Pr\left[e_t^\star = j \,|\, x_t, x_1^{t-1}\right]$ Then using Bayes rule, we get:

$$= \frac{Pr\left[x_t \,|\, e_t^\star = j, x_1^{t-1}\right] \cdot Pr\left[e_t^\star = j \,|\, x_1^{t-1}\right]}{Pr\left[x_t \,|\, x_1^t\right]}$$

The denominator is exactly equal to $q(x_t \,|\, x_1^{t-1})$. The first term in the numerator is exactly $p_j(x_t \,|\, x_1^{t-1})$, and the second term in the numerator is exactly $v_{t,j}$. We can now put everything together. Our learner's prediction at time step t is computed as:

$$q(x_t \,|\, x_1^{t-1}) = \sum_{i=1}^{N} v_{t,i} \cdot p_i(x_t \,|\, x_1^{t-1}) \tag{3}$$

We get the following update rule for our learner.

$$\forall i: \; v_{t+1,i} = \sum_{j=1}^{N} \frac{v_{t,j} \cdot p_j(x_t \,|\, x_1^{t-1})}{q(x_t \,|\, x_1^{t-1})} \cdot \begin{cases} 1-\alpha & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{else} \end{cases}$$

Let c_j be the term $c_j = \frac{v_{t,j} \cdot p_j(x_t \,|\, x_1^{t-1})}{q(x_t \,|\, x_1^{t-1})}$. Then, the update rule can be rewritten and simplified as:

$$\begin{aligned} v_{t+1,i} &= \sum_{j=1}^{N} c_j \cdot \begin{cases} 1-\alpha & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{else} \end{cases} \\ &= \sum_{j=1}^{N} c_j \cdot \left[\frac{\alpha}{N-1} + \left(1 - \alpha - \frac{\alpha}{N-1}\right) \mathbb{1}\{i = j\}\right] \\ &= \frac{\alpha}{N-1} \sum_{j=1}^{N} c_j + \left(1 - \alpha - \frac{\alpha}{N-1}\right) \cdot c_i \end{aligned}$$

Observe that $\sum_{j=1}^{N} c_j = 1$; this can be seen by noticing what $q(x_t \mid x_1^{t-1})$ equates to in (3). Thus,

$$v_{t+1,i} = \frac{\alpha}{N-1} + \left(1 - \alpha - \frac{\alpha}{N-1}\right) c_i$$

Thus, we have a constant-time update rule for each $v_{t+1,i}$. So, computation time per time step to update the N weights is $O(N)$. Notice that this is exactly the same as the run time of the Bayes algorithm, and there is no dependence on k. And so, we have arrived at a computationally efficient algorithm. Note that we do assume that k and T are known to us beforehand, which allows us to set $\alpha = \frac{k}{T-1}$.

3 Setup for next lecture: Managing an investment portfolio

We will first define a simple mathematical framework modeling a market of investment options. And then we will look at some investment strategies. Let's assume the following setup. We have a total of N investment options (also called stocks below). We index them with i. We make investment decisions on a fixed regular schedule; let's say once a day. We need to strategically decide how to divide up our entire wealth between the N stocks, at the beginning of each day. Without loss of generality, assume we start with \$1 of wealth.

On day t, the price relative or price change in stock i is denoted:

$$p_t(i) = \frac{\text{price of stock } i \text{ at end of day } t}{\text{price of stock } i \text{ at beginning of day } t}$$

We also define S_t to be our total wealth at the start of day t; so $S_1 = 1$. The investment problem for us is to pick the investment allocation amongst the stocks on each day t. This allocation is denoted as the vector $\mathbf{w}_t \in \mathbb{R}^N$ where $w_t(i) \geq 0$ and $\sum_i w_t(i) = 1$. In words, $w_t(i)$ is the *fraction* of wealth in stock i at the start of day t. The constraint that the $w_t(i)$ sums to 1 is simply saying that the entire wealth is invested at all times. This is fine since we can model putting money aside with a dummy stock that does not move.

By definition then, $S_t \cdot w_t(i)$ is the total wealth in stock i at the start of day t. And furthermore, based on how stock i changes in price we get $S_t \cdot w_t(i) \cdot p_t(i)$ is the total wealth in stock i at end of day t. Then, the total wealth on day $t+1$, dependent on the allocated wealth and price changes from day t is given by

$$S_{t+1} = \sum_{i=1}^{N} S_t \cdot w_t(i) \cdot p_t(i) = S_t(\mathbf{w}_t \cdot \mathbf{p}_t)$$

. And thus, $\mathbf{w}_t \cdot \mathbf{p}_t$ is our wealth increase from time step t to $(t+1)$.

We can then simply unroll the recurrence to get: $S_{T+1} = \prod_{t=1}^{T} (\mathbf{w}_t \cdot \mathbf{p}_t)$. As an investor, we want to maximize the term S_{T+1}, our total wealth after T steps. This goal can equivalently be achieved as:

$$\max \prod_{t=1}^{T} (\mathbf{w}_t \cdot \mathbf{p}_t) \equiv \max \sum_{t=1}^{T} \ln(\mathbf{w}_t \cdot \mathbf{p}_t) \equiv \min \sum_{t=1}^{T} -\ln(\mathbf{w}_t \cdot \mathbf{p}_t) \tag{4}$$

The above is now an on-line learning problem, with a log-loss of $-\ln(\mathbf{w}_t \cdot \mathbf{p}_t)$ at each time step t, and we want to minimize the sum of log-loss over all time steps. We can write it as:

For $t = 1, \ldots, T$

– learner chooses investment allocation $\mathbf{w}_t$

– world/adversary chooses price changes of stocks $\mathbf{p}_t$

– we experience loss $= -\ln(\mathbf{w}_t \cdot \mathbf{p}_t)$

Goal: We want to analyze portfolio selection *without* statistical assumptions about the underlying market, so it is quite different from investment ideas you may have come across which rely on assumptions about the nature of the market. Next time, we shall see an algorithm, with similarities to Bayes algorithm, that has a regret style bound on how well it does compared to the **best stock**, and how that translates to wealth for the investor/learner.