COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #21
Scribe: Xin Wan — April 22, 2019

Last time we talked about online learning of probability distributions, and showed why log loss is a convenient loss function to work with. As a reminder, the algorithm we worked with is summarized below:

Algorithm 1 online learning of probability distribution.

```
1: Initialize $w_{1,i} = \frac{1}{N}$
2: for $t = 1, \ldots, T$ do
3:     expert i predicts distribution $p_{t,i}$ on X
4:     learner/master predicts $q_t$ on X
5:     $\boxed{q_t(x) = \sum_{i=1}^{N} w_{t,i} p_{t,i}(x)}$
6:     observe $x_t \in X$
7:     loss $= -\ln q_t(x_t)$
8:     $\boxed{w_{t+1,i} = \frac{w_{t,i} p_{t,i}(x_t)}{q_t(x_t)}}$
9: end for
```

We will discuss the formula in the two boxes in this class. Last class, we also related the above algorithm to information / coding theory, where we set up the analysis as the following:

Assume Alice wants to send messages $x_1, \ldots, x_{t-1}$ to Bob, and we denote $x_1^{t-1} = \langle x_1, ... x_{t-1} \rangle$. If p_t is the estimated probability of x_t given x_1^{t-1}, then we need $-\lg p_t(x_t)$ bits to encode the message.

We will prove a bound for Algorithm 1 above that will be something like (note that here we are switching to log base-2 for comparison with the encoding analogy):

$$-\sum_{t=1}^{T} \lg q_t(x_t) \le \min_i \left[-\sum_{t=1}^{T} \lg p_{t,i}(x_t) \right] + \mathit{small}$$

Linking this to the coding theory setting, the left-hand side is equal to the total number of bits needed to encode $x_1, ..., x_T$ for the learner, whereas the first term on the right-hand side is the total number of bits needed for the best encoding method (out of the N available ones). The result is called "universal compression" in coding theory, since the result is a single coding method that, on every sequence of messages, is almost as good as the best of the N coding methods (for that individual sequence).

Now we introduce some notation which is meant to be suggestive of how the experts and learner are each trying to estimate the conditional probability of x_t given x_1^{t-1}:

$$p_{t,i}(x_t) = p_i(x_t \mid x_1^{t-1})$$

$$q_t(x_t) = q(x_t \mid x_1^{t-1})$$

Although online learning should be in an "adversarial" setting, where data might be chosen and presented by an "adversary" that tries to make learning hard, we can actually derive an algorithm here by pretending that data is generated by a random process, and the algorithm can be then shown to hold for all data, even including sequences presented by an adversary. Hence. let's for now pretend that $x_1, ..., x_T$ are generated as follows.

1. One expert i^* is chosen uniformly at random so that $Pr[i^* = i] = \frac{1}{N}$
2. The sequence $x_1, ..., x_T$ is generated by expert i^* so that

$$Pr[x_t \mid x_1^{t-1}] = p_i(x_t \mid x_1^{t-1})$$

Here, $Pr[\cdot]$ means probability with respect to this pretend random process.

Having defined this process, we next define the algorithm's prediction $q(x_t \mid x_1^{t-1})$ to be the conditional probability of x_t given x_1^{t-1} according to this random process, that is, $Pr[x_t \mid x_1^{t-}]$. Then, to compute q, we have:

$$\begin{aligned} q(x_t \mid x_1^{t-1}) &= Pr[x_t \mid x_1^{t-1}] \\ \text{Marginalizing, we have } &= \sum_{i=1}^{N} Pr[i^* = i \mid x_1^{t-1}]\, Pr[x_t \mid i^* = i, x_1^{t-1}] \\ \text{Using our notations, this is } &= \sum_{i=1}^{N} w_{t,i} p_i(x_t \mid x_1^{t-1}) \end{aligned}$$

where we define $w_{t,i}$ to be $Pr[i^* = i \mid x_1^{t-1}]$. Thus, as in Algorithm 1, we maintain one weight $w_{t,i}$ for each expert. These weights are initialized to be uniform, are used to compute the predicted distribution q as above, and are updated as below.

Now we compute $w_{t,i}$

$$w_{1,i} = Pr[i^* = i] = \frac{1}{N}$$

$$\begin{aligned} w_{t+1,i} &= Pr[i^* = i \mid x_1^t] = Pr[i^* = i \mid x_t, x_1^{t-1}] \\ &= \frac{Pr[i^* = i \mid x_1^{t-1}]\, Pr[x_t \mid i^* = i, x_1^{t-1}]}{Pr[x_t \mid x_1^{t-1}]} \text{, by Bayes Rule} \\ &= \frac{w_{t,i}\, p_i(x_t \mid x_1^{t-1})}{q(x_t \mid x_1^{t-1})} \text{, using our definition and notation above} \end{aligned}$$

The denominator in the above expression can be understood as a normalization factor. Recall that for WMA algorithm, the update was to multiply the weight of each expert by β if it made a mistake, and by 1 (i.e. unchanged) if not. Since we were there using 0-1 loss (which is 1 for mistake, 0 for no mistake), we can write the update as

$$w_{t+1,i} = \frac{w_{t,i}\, \beta^{loss}}{normalization}$$

When working with log loss, the loss of each expert is $-\ln p_{t,i}(x_t)$. If we then set $\beta = e^{-1}$, then the update rule given above can be seen to have exactly the same form.

We next prove the following regret bound for the algorithm described above.

Theorem 1 *In the setting above,*

$$-\sum_{t=1}^{T} \ln q_t(x_t) \le \min_i \left[-\sum_{t=1}^{T} \ln p_{t,i}(x_t) \right] + \ln N$$

Proof of the theorem:

1. Step 1
   First we compute the probability of any sequence x_1^T according to our pretend random process:

$$\begin{aligned} Pr[x_1^T] &= Pr[x_1, ..., x_T] \\ &= Pr[x_1]\, Pr[x_2 \mid x_1]\, Pr[x_3 \mid x_1, x_2] \cdots Pr[x_T \mid x_1^{T-1}] \text{ , by the chain rule} \\ &= \prod_{t=1}^{T} Pr[x_t \mid x_1^{t-1}] \text{ , this is just simplifying notation} \\ &= \prod_{t=1}^{T} q(x_t \mid x_1^{t-1}) \text{ , using the notation we defined previously} \end{aligned}$$

   Note that the equality above is true for all sequences of x_t's

2. Step 2
   Then we compute the probability of a given base expert predicting the sequence

$$Pr[x_1^T \mid i^* = i] = \prod_{t=1}^{T} p_i(x_t \mid x_1^{t-1}) \text{ , again, chain rule}$$

3. Step 3
   The total probability of this sequence is therefore summation (across all base experts) of the probability of choosing each base expert multiplied by the probability that it predicts the sequence.

$$\begin{aligned} Pr[x_1^T] &= \sum_{i=1}^{N} Pr[i^* = i]\, Pr[x_1^T \mid i^* = i] \\ &\ge Pr[i^* = i]\, Pr[x_1^T \mid i^* = i] \text{ for any particular } i \\ &= \frac{1}{N} Pr[x_1^T \mid i^* = i] \end{aligned}$$

Now we can put these steps together:

$$\begin{aligned}
-\sum_{t=1}^{T} \ln q_t(x_t \mid x_1^{t-1}) &= -\ln \prod_{t=1}^{T} q_t(x_t \mid x_1^{t-1}) \\
&= -\ln Pr[x_1^T], \text{ by Step 1} \\
&\le -\ln \left[\frac{1}{N} Pr[x_1^T \mid i^* = i]\right], \text{by Step 3} \\
&= -\ln \left[\prod_t p_i(x_t \mid x_1^{t-1})\right] + \ln N, \text{by Step 2} \\
&= -\sum_t \ln \left[p_i(x_t \mid x_1^{t-1})\right] + \ln N
\end{aligned}$$

Since this is true for every expert i, it is also true for the best expert, which proves the theorem.

■

There are two ways to think about this theorem: first, we can divide both sides by T. Then the theorem says that the "per time-step loss" of the learner quickly approaches that of the best expert as T increases. Alternatively, we can divide both sides by a constant and change ln to $\log_2$. This relates to coding theory: first notice that the term on the left is the number of bits that the algorithm will use to encode the sequence of messages x_1^T, and the first term on the right is the same thing for the best of N given coding methods. The bound says that the algorithm will never use more than $\lg N$ bits than the best of the coding methods, and that is true for every sequence x_1^T. There is an obvious alternative of encoding the messages, which is (for the sender) to try all N methods, pick the one that is best for the sequence x_1^T, and then send over the index of that best method together with the encoding that it gives. Since sending the index of the method requires $\lg N$ bits, this will require the same overall code length as for our algorithm. But our algorithm can be implemented online, one x_t at a time.

One way to potentially improve the performance of the algorithm is to use a non-uniform distribution when initializing the weights. That is, we can choose a "prior", which is a set of non-negative weights π_i over the experts which sums to one. We then use π_i to set the initial weights, or equivalently, to choose i^* in the pretend process for generating data.

$$w_{1,i} = Pr[i^* = i] = \pi_i, \pi_i \ge 0, \sum \pi_i = 1$$

Using this prior would lead to the following bound:

$$-\sum_{t=1}^{T} \ln q_t(x_t) \le \min_i \left[-\sum_{t=1}^{T} \ln p_{t,i}(x_t) - \ln \pi_i\right]$$

The idea is to choose a prior that gives higher probability π_i to experts that we think are more likely to be good. This bound implies that the regret relative to experts with higher prior probability π_i will be lower.

Now we consider a new scenario: so far we have only looked at an algorithm that performs almost as well as the best single expert for the entire sequence, which means we are implicitly assuming that there is one expert that will be reasonably good the entire time. But it might be the case that the nature of the data sequence changes in time so that one expert might be best for a while, but then another one might be good for a while, and so on, so that the best expert is "switching" from time to time. We now want to find an algorithm whose performance is almost as good as the best switching sequence of experts.

For the purpose of our analysis, we aim to do as well as the best switching sequence of experts with at most k switches between time 1 and T, and there are again N "base experts".

One approach to solving this problem is to create "meta-experts" by combining different base experts at different time steps. Specifically, a meta-expert is a constructed sequence of base experts. It behaves like a particular base expert at any given time, but changes from expert to expert as time changes. For example, a particular meta-expert can be "expert 3 at time 1 to 17, expert 8 at time 18 to 93, ... ". The prediction of a particular meta-expert therefore exactly matches that of a particular sequence of "switching experts" as we discussed above. Specifically, we can consider one meta-expert for every switching sequence with k switches. The theorem above applied on meta-experts as experts immediately gives us a regret bound that is log of the number of meta-experts. The loss of the resulting algorithm will be at most the loss of the best meta-expert (that is the best switching sequence), plus $\ln M$, where M is the total number of meta-experts.

By simple combinatorics, since we can choose from any of the N base experts between each switch, and we can choose any of the switches to be between $t = 1$ and $t = T$, the total number of different meta-experts is approximately:

$$M \approx N^{k+1} T^k$$

Then if we apply the theorem above, we will get a similar result except the last term changes from $\ln N$ to

$$\ln M \approx \ln N + k(\ln N + \ln T),$$

which says that each additional switch of expert in the process brings a cost of about $\ln N + \ln T$ to the bound.

There is a problem, however, with the approach above. It gives a good regret bound, but computationally, we need to maintain one weight for every meta-expert, which will be extremely expensive if the number of meta-experts is large or the sequence is long. Hence, now we will focus on an alternative method that appears expensive, but, we will see, can be implemented very efficiently.

The alternative way to construct "meta-experts" is one meta-expert for *every* sequence of base experts. So a meta-expert is a vector $\mathbf{e} = \langle e_1, ..., e_T \rangle$ where $e_t \in \{1, ..., N\}$ is a base expert that is used at time t by the meta-expert. In other words, the meta-expert's prediction at time t will be a copy of whatever base expert e_t predicts. This seems very inefficient as it creates an enormous number of meta-experts, but could have an advantage that it will make it possible to implement the algorithm very efficiently. The downside is we are including all meta-experts, including ones that have a very large number of switches. To mitigate this, we therefore choose a prior over the meta-experts in a way that gives higher weight to meta-experts with a smaller number of switches. We then apply Bayes algorithm to the meta-experts, together with the prior.

We wish to create a prior $\pi(\mathbf{e})$ over meta-experts $\mathbf{e}$. This prior is just a probability distribution. To define it, we describe a process for choosing a meta-expert $\mathbf{e}^*$ randomly (just as we earlier chose an expert i^* randomly according to π_i). Then the prior $\pi(\mathbf{e})$ will be $Pr[\mathbf{e}^* = \mathbf{e}]$. Denote $Pr[\mathbf{e}^* = \mathbf{e}] = \pi(\mathbf{e})$. Then we use the following random process to generate $\mathbf{e}^*$:

1. Pick e_1^* from a uniform distribution over base experts; $Pr[e_1^* = i] = \frac{1}{N}$
2. $Pr[e_{t+1}^* \mid e_t^*] = \begin{cases} 1-\alpha & \text{if } e_{t+1}^* = e_t^* \\ \frac{\alpha}{N-1} & \text{else} \end{cases}$

This generative process defines a probability distribution over sequences $\mathbf{e}^*$. We can compute the probability explicitly:

Assume in $\mathbf{e}$ there are $k_{\mathbf{e}}$ switches of experts. Then

$$Pr[\mathbf{e}^* = \mathbf{e}] = \frac{1}{N}(1-\alpha)^{T-k_{\mathbf{e}}-1}\left[\frac{\alpha}{N-1}\right]^{k_{\mathbf{e}}}$$

Now we plug this in the regret bound and get that the regret to any switching expert with k switches is:

$$\begin{aligned} -\ln \pi(e) &= -\ln\left[\frac{1}{N}\left[\frac{\alpha}{N-1}\right]^k (1-\alpha)^{T-k-1}\right] \\ &= \ln N + k \ln\left[\frac{N-1}{\alpha}\right] - (T-k-1)\ln(1-\alpha) \end{aligned}$$

If we set $\alpha = \frac{k}{T-1}$, then we have:

$$\begin{aligned} -\ln \pi(e) &= \ln N + k \ln\left[\frac{(N-1)(T-1)}{k}\right] - (T-k-1)\ln(1-\frac{k}{T-1}) \\ &\approx \ln N + k(\ln N + \ln T) \end{aligned}$$

This shows a similar cost of about $\ln N + \ln T$ for an additional switch of expert in the sequence.