COS 511: Foundations of Machine Learning

Rob Schapire
Scribe: Jeffrey Bigham

Lecture # 19
April 15, 2003

1 Review From Last Time

During the last lecture we discussed online learning with log loss with experts advice. These experts predicted probability distributions with the goal being to come close to the best expert online. This can also be viewed from a coding theory point of view by viewing this online learning method as a way to combine compression methods to eventually find the best.

We introduced some notation:
$x_1^t = x_1, x_2, ..., x_t$
$p_{t,i}(x_t) = p_i(x_t|x_1^{t-1})$
$q_t(x_t) = q(x_t|x_1^{t-1})$
In particular, we had the following setup:
for $t = 1, ..., T$
expert $i \rightarrow p_{t,i}$
master $\rightarrow q_t$
(the master is the combiner of all the individual experts)
observe $x_t \in X$
loss of expert $i = -\ln(p_{t,i}(x_t))$
loss of master $= -\ln(q_t(x_t))$
Our goal is for the log loss of the master to not be much worse than the loss of the best expert.

Last time the algorithm was motivated by thinking of selecting the x_t's randomly, but they are actually worst case.

From this we came up with two rules:

1. an expert i^* is chosen according to $\Pr[i^* = i] = \frac{1}{N}$
2. $\Pr[x_t|i^* = i] = p_i[x_t|x_1^{t-1}]$

2 Analyzing the Loss

Now we are still left with the question of how to analyze the loss, which turns out to be fairly easy.

First we will define $p_i(x_1^t) = \Pr[x_1^t|i^* = i]$ and $q(x_1^t) = \Pr[x_1^t]$.

We then have

$$Pr[x_1^t] = \sum_i \Pr[i^*] \Pr[x_1^T|i^* = i] \tag{1}$$

$$= \sum_i \frac{1}{N} p_i(x_1^T) \tag{2}$$

Master Algorithm Loss:

$$
\begin{aligned}
-\sum_{t=1}^{T} \ln(q(x_t|x^{t-1})) &= -\ln \prod_{t=1}^{T} q(x_t|x_1^{t-1}) && (3)\\
&= -\ln q(x_1^T) && (4)\\
&= -\ln \left(\frac{1}{N} \sum_i p_i(x_1^T) \right) && (5)\\
&\le -\ln \left(\frac{1}{N} p_i(x_1^T) \right) && (6)\\
&= -\ln(p_i(x_1^T)) + \ln N && (7)\\
&= -\sum_{t=1}^{T} \ln(p_i(x_t|x_1^{t-1})) + \ln N && (8)
\end{aligned}
$$

That is to say that the cumulative loss is less than or equal to the loss of the best expert plus the log of the number of experts. The $\ln N$ is negligible intuitively because you can think of it as an amount of loss per round that is rapidly going to zero:

$$
-\frac{1}{T}\sum_{t=1}^{T} \ln(q(x_t|x_1^{t-1})) \le -\frac{1}{T}\sum_{t=1}^{T} \ln(p_i(x_t|x_1^{t-1})) + \frac{\ln N}{T} \qquad (9)
$$

3 Non-Uniform Distribution

Before we used a uniform distribution, but in general you can set any prior $Pr[i^* = i] = \pi_i$, which transforms (8) to

$$
\le \min_i \left[-\sum_{t=1}^{T} \ln(p_i(x_t|x_1^{t-1})) + \ln \frac{1}{\pi_i} \right] \qquad (10)
$$

4 An Example

$X = \{0, 1\}$

Have an expert for every number between 0 and 1.

Define expert $p \in [0, 1]$. It predicts 1 with probability p and 0 with probability $1 - p$. Working through the steps from last time we get:

$$
\begin{aligned}
q &= \frac{\int_0^1 p^{h+1}(1-p)^{t-h}dp}{\int_0^1 p^h(1-p)^{t-h}dp} && (11)\\
&= \frac{h+1}{t+2} && (12)
\end{aligned}
$$

where $h = \#$ of 1's so far and $w_{t,p} = p^h(1-p)^{t-h}$

You would expect that the intuition for this would be that if you flip a coin t times you would see h heads, but this shows that a sort of smoothing is occurring. It also means that this gives meaningful predictions even before you start.

5 Switching Experts

So far we have always talked about how to predict as well as the best expert, but this isn't always reasonable. Maybe the best expert changes between different blocks of rounds. For instance during a sequence of trials, expert i could be the best on the first n trials and expert j could be the best on the next m trials and so on. In this section we will explore how to find the best sequence of switching experts. Doing this with the ability to switch at every time step is impossible, but given a limit on the number of switchings, how can you compete against the best switching algorithm.

Simple solution:

Given k switches we can create a Meta-Expert for every possible switching sequence and run the algorithm developed in the previous section on each.

A rough estimate of the number of meta experts is $N^{k+1}\binom{T}{k}$.

The additional error in 8 becomes $(k+1)\ln N + k\ln T$. That is each time you switch experts you pay something on the order of $\ln N + \ln T$. However, if implemented naively the running time will be very great, proportional to $N^{k+1}\binom{T}{k}$ but because the experts are well-structured we can do much better.

Members of the class suggested that using some sort of dynamic programming approach might work, but instead we will look at all meta-experts but arrange the priors to put a lot of weight on experts with few switches and very little on those with a lot of switches.

meta-expert: $e_1, e_2, ..., e_T$ where $e_t = i$, meaning that the meta expert is using expert i during round t. The algorithm will pick which e_t will be used on round t.

$$\Pr[\vec{e^*} = \vec{e}] = \pi(\vec{e}) \tag{13}$$

$$\Pr[e_1^* = i] = \frac{1}{N} \tag{14}$$

$$\Pr[e_{t+1}^*|e_t^*] = \begin{cases} 1-\alpha & \text{if } e_{t+1}^* = e_t \text{ ,} \\ \frac{\alpha}{N-1} & \text{otherwise.} \end{cases} \tag{15}$$

Starting from the end, we can analyze this as follows, setting $\alpha = \frac{k}{T-1}$:

$$-\ln \pi(\vec{e}) = -\ln[\frac{1}{N}\left(\frac{\alpha}{N-1}\right)^k (1-\alpha)^{T-k-1}] \tag{16}$$

$$= \ln N + k \ln \frac{N-1}{\alpha} + (T-k-1)\ln\frac{1}{1-\alpha} \tag{17}$$

$$= \ln N + k \ln \frac{(N-1)(T-1)}{k} - (T-k-1)\ln(1-\frac{k}{T-1}) \tag{18}$$

The last term of this is $\approx k(1-\frac{k}{T-1}) \leq k$.

This is quite similar to before but the questions of how to compute q and the update remains.

First q:

$$q(x_t|x_1^{t-1}) = \Pr[x_t|x_1^{t-1}] \tag{19}$$
$$= \sum_i \Pr[x_t, e_t^* = i|x_1^{t-1}] \tag{20}$$
$$= \sum_i \underbrace{\Pr[e_t^* = i|x_1^{t-1}]}\underbrace{\Pr[x_t|x_1^{t-1}, e_t^* = i]} \tag{21}$$

The first term of the above product is $w_{t,i}$ and the second is $p_i(x_t|x_1^{t-1})$. We now set out to determine how to compute $w_{t,i}$ efficiently.

Initially:

$$w_{1,i} = \frac{1}{N}$$

During each update:

$$w_{t+1,i} = \Pr[e_{t+1}^* = i|x_1^t] \tag{22}$$
$$= \sum_j \Pr[e_{t+1}^* = i, e_t^* = j|x_1^t] \tag{23}$$
$$= \sum_j \underbrace{\Pr[e_t^* = j|x_1^t]}\underbrace{Pr[e_{t+1}^* = i|e_t^* = j, x_1^t]} \tag{24}$$

The first term of the product in the final summation is basically a measure of how good the experts are. The second term

$$= \begin{cases} 1-\alpha & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{else} \end{cases} \tag{25}$$

Bayes' Rule Refresher:

$$\Pr[B|A] = \frac{\Pr[A|B]\Pr[B]}{\Pr[A]}$$

Expanding the first term we get:

$$= \Pr[e_t^* = j|x_1^{t-1}, x_t] \tag{26}$$
$$= \frac{\Pr[x_t|e_t^* = j, x_1^{t-1}]\Pr[e_t^* = j|x_1^{t-1}]}{\Pr[x_t|x_1^{t-1}]} \tag{27}$$
$$= \frac{p_j(x_t|x_1^{t-1})w_{t,j}}{q(x_t|x_1^{t-1})} \tag{28}$$

The α term can be seen as a sort of switching rate and it can be set accordingly. In practice, however, you probably won't know the prior that favors fewer switches.

The algorithm just presented is usually called the **Weight Share** algorithm.

6 How To Make Money

In this section we switch gears a little and turn to how we can use the techniques we have recently learned for making money. Suppose you have N stocks and you want to decide how to invest your money in these stocks over a given time period. We will assume that we start with \$1 although everything we will discuss can be translated into greater amounts straightforwardly.

On each day t, every stock either goes up or down at least a little.

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

In each time period we need to decide how to allocate our money.

$$w_t(i) = \text{ fraction of wealth in stock } i \text{ at beginning of day } t$$

$$\sum_i w_t(i) = 1$$

Let S_t be the total amount of wealth at the beginning of day t. At the beginning of day t we then have $S_t w_t(i)$ in stock i and by the end of the day we have $S_t w_t(i) p_t(i)$.

$$S_{t+1} = \sum_i S_t w_t(i) p_t(i) = S_t(\vec{w_t} \cdot \vec{p_t}) \tag{29}$$

$$S_{t+1} = \prod_{t=1}^{T} (\vec{w_t} \cdot \vec{p_t}) \cdot (S_1 == 1) \tag{30}$$

Maximizing this product gives us:

$$\max \prod_t (\vec{w_t} \cdot \vec{p_t}) \equiv \max \sum_t \ln(\vec{w_t} \cdot \vec{p_t}) \tag{31}$$

$$\min \sum_t - \ln(\vec{w_t} \cdot \vec{p_t}) \leftarrow \text{just the loss function for learning} \tag{32}$$

To do almost as well as the best stock, we can simply apply Bayes' algorithm almost directly:

$$q_t(x_t | x_1^{t-1}) = \sum_i w_{t,i} p_i(x_t | x_1^{t-1}) \tag{33}$$

Our earlier algorithm gave a bound of:

$$-\sum_{t=1}^{T} \ln(q(x_t | x_1^{t-1})) \leq -\sum_{t=1}^{T} \ln(p_i(x_t | x_1^{t-1})) + \ln N \tag{34}$$

and doing a translation between the two we arrive at

$$-\sum_{t=1}^{T} \ln(\vec{w}_t \cdot \vec{p}_t) \leq -\sum_{t=1}^{T} \ln(p_t(i)) + \ln N \tag{35}$$

for our loss.

The algorithm basically just tells us to invest in each stock evenly in the beginning and then leave it there. Our wealth is as follows:

$$S_{T+1} = \text{ wealth of algorithm } \geq \frac{1}{N} \text{ wealth of best stock} \tag{36}$$

7 Foreshadowing

Next time we will look at something better $\rightarrow$ competing against a consistently balanced portfolio.