COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Tengyu Ma

Lecture # 22
April 25, 2013

1 Recap

Last time we discussed the Bayes algorithm for learning a distribution with a log loss function. There are N experts, each of which predicts a distribution $p_{t,i}(\cdot)$ over domain X in each round. We are to predict the distribution in an online manner with as small log loss as possible. In the Bayes algorithm, we assign each expert an initial weight/prior π, and maintain a weight $w_{t,i}$ for each expert in each round. Each round the algorithm predicts the linear combination of experts' distribution according to the weights, and updates the weight $w_{t+1,i} = w_{t,i}p_{t,i}(x_t)/Z$ after obtaining the input x_t (where Z is normalization factor.)

The beauty of this algorithm is that we can interpret/analyze it as follows: We can view the initial weights π_i as priors on the experts, and pretend x_t's are generated from a certain random process defined by the priors and the predictions of the experts, and then at each round we update our weights by the posteriors.

In detail, the algorithm works as follows:

Given N, priors (distribution over $[N]$) π_i
$w_{1,i} = \pi_i$
For $t = 1$ to T
Upon seeing the experts' predictions $p_{t,i}$
Master predicts distribution $q_t(\cdot)$ over X

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

Upon observing x_t

$$w_{t+1,i} = \frac{w_{t,i}p_{t,i}(x_t)}{\text{Normalization}}$$

We proved that the log loss of q is bounded by

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

We analyzed the algorithm by pretending the data is generated by the following process,

1. $\Pr[i^* = i] = \pi_i$
2. $\Pr[x_t \mid x_1^{t-1}, i^* = i] \triangleq p_i(x_t|x_1^{t-1}) = p_{t,i}(x_t)$

If the data is generated as above, then what our algorithm does is exactly to output

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

2 Switching Experts

The Bayes algorithm above compares the loss of the algorithm with the best expert. However, it is quite likely that there are k experts (k is very small compared to N and T), each does very good prediction on a series of consecutive rounds. For example, Expert 5 does very well in the first 200 rounds, Expert 2 does well in the next 37 rounds, etc.. Then a very good strategy may be just follow the 5th expert prediction in the first 200 rounds, and then switch to the second, etc. In this section, we want to improve the Bayes algorithm so that we get as good as prediction of this kind of switching strategy.

To make this idea concrete, we refer to the real experts as "base-experts", and we define the "meta-experts" to be imaginary agents that follow the predictions of some base experts at each round, while only switching at most k times between the experts it follows.

Thus there are all together $M = N^{k+1}\binom{T-1}{k}$ meta-experts, because we can choose k switching places from the $T-1$ possible switching places, and after fixing the switching positions, in a segment of consecutive rounds, there are N choices of base-experts to follow. ($k-1$ switching places and k base-experts uniquely determine a meta-expert. For example, when $k = 3$, "switch at 6^{th} day and 8^{th} day, and use Expert 3 in the first segment, Expert 4 at the second, Expert 8 at the third" uniquely determine a meta-expert, which follows Expert 3 from Day 1 to Day 5, Expert 4 from Day 6 to Day 7, Expert 8 from Day 8 to the end.) We expect a new algorithm to do as well as the best meta-expert. A natural idea is to apply the Bayes algorithm to all of the meta-experts as follows: We choose the priors on the meta-experts to be uniform (we suppose we know T here, then we know M, and all of the meta-experts).

Thus the additional loss of our algorithm compared to the best meta-experts is

$$\ln M \approx (k+1)\ln N + k\ln(T/k)$$

which means roughly $\ln N + \ln T$ per switch.

The bound is quite good; however, the main issue is that the running time/space is at least M, which is exponential k. Next, we are going to show an alternative algorithm that can be implemented efficiently and that achieves a similar bound.

2.1 Weight share algorithm

Though the main issue of the previous algorithm is the number of meta-experts, in the weight share algorithm, we are not to reduce the number of meta-experts. Instead, we play with the priors and design a clever prior distribution under which the meta-experts can be generated efficiently. We even enlarge the class of meta-experts to be all of the combinations of experts to follow in the T rounds (instead of those with only k switches.)

A meta-experts here can be represented as $\boldsymbol{e} = (e_1, \ldots, e_T) \in \{1, \ldots, N\}^T$, where e_t is the base expert that it follows in round t.

Then we define the distribution over all the meta-experts by the following random process of choosing expert $\boldsymbol{e^*}$, and let $\pi(\boldsymbol{e})$ be the probability that e is chosen by the random process, that is, $\pi(\boldsymbol{e}) = \Pr[\boldsymbol{e^*} = \boldsymbol{e}]$

1. $\Pr[e_1^* = i] = \frac{1}{N}$
2. $\Pr[e_{t+1}^* \mid e_t^*] = \begin{cases} 1-\alpha & \text{if } e_{t+1}^* = e_t^* \\ \alpha/(N-1) & \text{otherwise} \end{cases}$

In other words, given e_t^*, e_{t+1}^* is chosen to be the same as e_t^* with probability $1-\alpha$, and chosen to be any other value with probability $\alpha/(N-1)$, where α is a constant to be determined.

Then we run the Bayes algorithm on the prior $\pi(\boldsymbol{e})$ (we will describe how to run it efficiently in a second), and get the additional loss for each $\boldsymbol{e}$ with at most k switches,

$$-\ln \pi(\boldsymbol{e}) = -\ln[\frac{1}{N} \cdot (\alpha/(N-1))^k \, (1-\alpha)^{T-k-1}] = \ln N + k \ln(\frac{N-1}{\alpha}) + (T-k-1)\ln(1-\alpha)$$

By taking $\alpha = \frac{k}{N-1}$ we have that for $\boldsymbol{e}$ with at most k switches,

$$-\ln \pi(\boldsymbol{e}) = \ln N + k \ln\left[\frac{(N-1)(T-1)}{k}\right] + (\text{some term } \leq k)$$

Thus we derived almost the same bound as in the previous section. The remaining task is to compute this efficiently.

Recall that in the Bayes algorithm, we update our weights using the posterior $\Pr[i^* = i \mid x_1^{t-1}]$, and output the distribution

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

We can compute $q_t(x_t)$ as follows:

$$\begin{aligned}
q_t(x_t) = \Pr[x_t \mid x_1^{t-1}] &= \sum_i \Pr[x_t, e_t^* = i \mid x_1^{t-1}] \\
&= \sum_i \Pr[x_t \mid e_t^* = i, x_1^{t-1}] \Pr\left[e_t^* = i \mid x_1^{t-1}\right] && \text{(Chain rule)} \\
&= \sum_i p_i(x_t \mid x_1^{t-1}) v_{t,i} && v_{t,i} \triangleq \Pr\left[e_t^* = i \mid x_1^{t-1}\right]
\end{aligned}$$

We compute $v_{t,i}$ in an inductive manner as follows. The base case is obvious,

$$v_{1,i} = \Pr[e_1^* = i] = \frac{1}{N}$$

And we can compute $v_{t_1,i}$ as follows:

$$\begin{aligned}
v_{t+1,i} &= \Pr[e_{t+1}^* = i \mid x_1^t] \\
&= \sum_j \Pr[e_{t+1}^* = i, e_t^* = j \mid x_1^t] \\
&= \sum_j \Pr[e_{t+1}^* = i \mid e_t^* = j, x_1^t] \Pr[e_t^* = j \mid x_1^t] && \text{(Chain rule)}
\end{aligned}$$

We have that

$$\begin{aligned}
\Pr[e_t^* = j \mid x_1^t] &= \frac{\Pr[e_t^* = j \mid x_1^{t-1}] \Pr[x_t \mid e_t^* = j, x_1^{t-1}]}{\Pr[x_t \mid x_1^{t-1}]} && \text{(Bayes rules)} \\
&= \frac{v_{t,j} p_j(x_t \mid x_1^{t-1})}{q_t(x_t \mid x_1^{t-1})}
\end{aligned}$$

and

$$\Pr[e^*_{t+1} = i \mid e^*_t = j, x_1^t] = \Pr[e^*_{t+1} = i \mid e^*_t = j] \qquad \text{for } e^*_{t+1}, x_1^t \text{ are independent}$$
$$= \begin{cases} (1-\alpha) & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{otherwise} \end{cases}$$

Let $c_{t,j} = \frac{v_{t,j} p_j(x_t | x_1^{t-1})}{q_t(x_t | x_1^{t-1})}$, it can be checked that $\sum_j c_{t,j} = 1$.
Putting all together, we get that

$$\begin{aligned} v_{t+1,i} &= \sum_j \Pr[e^*_{t+1} = i \mid e^*_t = j, x_1^t] \Pr[e^*_t = j \mid x_1^t] \\ &= \sum_j \frac{v_{t,j} p_j(x_t \mid x_1^{t-1})}{q_t(x_t \mid x_1^{t-1})} \cdot \begin{cases} (1-\alpha) & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{otherwise} \end{cases} \\ &= \sum_j c_{t,j} \begin{cases} (1-\alpha) & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{otherwise} \end{cases} \\ &= \frac{\alpha}{N-1} + (1 - \alpha - \frac{\alpha}{N-1}) c_{t,i} \qquad \text{since } \sum_j c_{t,j} = 1 \end{aligned}$$

Thus we can see that the running time for updating $v_{t,i}$ is $O(N)$, which is quite efficient.

3 Market Investment

In this section we try to apply the Bayes algorithm to the stock market setting. Suppose we have N stocks. On Day t, we denote by $p_t(i)$ the ratio of the price of stock i at the end of day t to its price at the beginning of Day t, and we assume that the price at the end of Day t is equal to the price at the beginning of Day $t+1$. That is,

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

Our wealth at the start of Day t is denoted by S_t, and the fraction of our wealth in stock i at the start of Day t is $w_t(i)$. (Thus $\sum_i w_t(i) = 1$.) Thus the total wealth at the start of Day t is $S_t = \sum w_t(i) S_t$ and at the end of Day t the wealth is $S_{t+1} = \sum_t w_t(i) S_t p_t(i) = S_t(\boldsymbol{w}_t \cdot \boldsymbol{p}_t)$. Note that we assume that we can update the vector $\boldsymbol{w}_t$ at the beginning of each day and it remains fixed in the same day.

Thus at the end of T days the total wealth becomes

$$S_T = \prod_{t=1}^{T} \boldsymbol{w}_t \cdot \boldsymbol{p}_t$$

(We assume without generality that we have initial wealth 1 dollar.)

This is a multiplicative term and it's natural to consider its logarithm so that multiplication changes to summation. Thus our goal is to maximize the log of S_T, which is

$$\ln S_T = \sum_t \ln(\boldsymbol{w}_t \cdot \boldsymbol{p}_t)$$

It is equivalent to minimize the negation of the above formula, which is

$$\min -\ln S_T = \min -\sum_t \ln(\boldsymbol{w}_t \cdot \boldsymbol{p}_t)$$

Observe that we have transformed the investing problem into an online learning problem. In each round, the investor chooses weight function $\boldsymbol{w}_t(\cdot)$ and the nature(the market) responds the relative increase $\boldsymbol{p}_t$. The goal is to minimize the cumulative loss function $\sum_t -\ln(\boldsymbol{w}_t \cdot \boldsymbol{p}_t)$

We apply the Bayes algorithm in the following way:

$$\begin{aligned}
&\text{Let } C = \max_{t,i} p_t(i) \\
&X = \{0, 1\} \\
&\text{Let } p_{t,i}(1) = p_t(i)/C, \text{ and } p_{t,i}(0) = 1 - p_{t,i}(1) \\
&x_t = 1 \forall t
\end{aligned}$$

Observe that

$$q_t(x_t) = \sum_i w_{t,i} p_{t,i}(1) = \sum_i w_{t,i} p_t(i)/C = \frac{\boldsymbol{w}_t \cdot \boldsymbol{p}_t}{C}$$

and then by the bound guaranteed by the Bayes algorithm, we have that

$$\sum_t -\ln \frac{\boldsymbol{w}_t \cdot \boldsymbol{p}_t}{C} = \sum_t -\ln q_t(x_t) \leq \min_i [-\sum_t p_{t,i}(x_t)] + \ln N = \min_i [-\sum_t p_t(i)/C] + \ln N$$

By canceling out the C in the two sides of the equation, basically we get that

$$-\ln(\text{wealth of the algorithm}) \leq -\ln(\text{wealth of the best single stock}) + \ln N$$

However, this seemingly good bound actually is trivial because it is eqquivalent to

$$(\text{wealth of the algorithm}) \geq \frac{1}{N}(\text{wealth of the best single stock})$$

It means that even choosing each stock with $1/N$ fraction of wealth can do as good as this bound. Also it turns out that if we run this algorithm step by step, we can see that the algorithm is equivalent to the following "buy and hold" strategy: we place $1/N$ of our wealth in each investment on Day 1, and then just leave it there, never doing any buying or selling. We will show how to improve this performance of this algorithm in the next class.