COS 340: Reasoning About Computation[*]
Online Algorithms 2

Moses Charikar

March 9, 2014

1 Introduction

In the previous lecture, we introduced the notion of online algorithms and competitive analysis. We studied the Rent or Buy problem, and then discussed the List Update problem. In this lecture, we will look at two problems: scheduling jobs on machines, and a problem about learning from expert advice.

2 Machine Scheduling

We consider a simple machine scheduling problem: Suppose we have n machines available to us. A sequence of jobs arrive online and our goal is to assign each job to a machine as soon as it arrives. Every job has a size and the *load* on a machine is the sum of sizes of jobs assigned to that machine. The goal of the algorithm is to maintain an assignment to jobs to machines that minimizes the maximum load of the n machines. For every new job, the online algorithm needs to decide which machine to assign it to, without any knowledge of future jobs. Previously assigned jobs cannot be moved to different machines. Also assume that jobs stay forever, i.e. once a job is assigned to a machine, the job contributes to the load of the machine forever. At the end, we will comment briefly on the variant of this problem when jobs complete after some time and exit the system.

We will measure the performance of this algorithm by comparing it to an offline algorithm that is allowed to use the knowledge of the entire sequence of jobs in assigning jobs to machines. Let σ denote the sequence of jobs seen so far. Let $C_A(\sigma)$ denote the maximum load for the assignment maintained by the online algorithm at the end of the request sequence σ. Let $C_{OPT}(\sigma)$ denote the maximum load for the best assignment of jobs in σ to the n machines. We say that the online algorithm is c competitive, if for all request sequences σ,

$$C_A(\sigma) \leq c \cdot C_{OPT}(\sigma).$$

[*]Princeton University, Spring 2014

We emphasize again that the online algorithm is not allowed to change its assignment of earlier jobs to machines. However, the optimal solution that we are comparing to can change dramatically as additional jobs are seen.

Consider this simple greedy algorithm for this problem: A new job is always assigned to the machine with the current minimum load (with ties broken arbitrarily). We claim that the competitive ratio of this algorithm is at most 2. Consider a request sequence σ consisting of k jobs, and say s_i is the size of the ith job. Suppose that a particular machine M has the maximum load at the end of processing these k jobs. We will obtain an upper bound on the load of machine M in terms of the jobs sizes s_i. Now suppose that job j was the last job assigned to machine M. Then before job j was assigned, machine M had the minimum load amongst the n machines. Hence the load on M before job j was assigned was at most $(\sum_{i=1}^{j-1} s_i)/n$. Thus the load on M after job j was assigned to it could be at most $s_j + (\sum_{i=1}^{j-1} s_i)/n$. The first term in the expression is $s_j \leq \max_i\{s_i\}$ (i.e. the maximum job size) and the second term is $(\sum_{i=1}^{j-1} s_i)/n \leq (\sum_{i=1}^{k} s_i)/n$ (i.e. the sum of all job sizes divided by n). Thus the maximum load for the online algorithm is

$$C_A(\sigma) \leq \max_i\{s_i\} + (\sum_{i=1}^{k} s_i)/n$$

Now, it is easy to see that $C_{OPT}(\sigma) \geq \max\limits_i\{s_i\}$ and $C_{OPT}(\sigma) \geq (\sum\limits_{i=1}^{k} s_i)/n$. Thus, we have proved that

$$C_A(\sigma) \leq 2C_{OPT}(\sigma).$$

In other words, the simple greedy algorithm is 2-competitive. We can improve the bound slightly by being more careful. In the above analysis, we can conclude that the load on machine M is at most $(1-\frac{1}{n})s_j + (\sum_{i=1}^{j} s_i)/n$. It is easy to check that this is indeed the same bound we used earlier stated a little differently. The first term is now at most $(1-\frac{1}{n})\max_i\{s_i\}$ and the second term is at most $(\sum_{i=1}^{k} s_i)/n$. Hence

$$C_A(\sigma) \leq \left(1-\frac{1}{n}\right)\max_i\{s_i\} + (\sum_{i=1}^{k} s_i)/n \leq \left(2-\frac{1}{n}\right)C_{OPT}(\sigma).$$

In fact this analysis of the greedy algorithm is tight. It is not too difficult to construct a request sequence for which the schedule constructed by the greedy algorithm has maximum load a factor $2-1/n$ times the maximum load of the optimum algorithm. The main idea is to construct a request sequence of many small jobs (which the greedy algorithm spreads evenly across all machines) followed by one big job. More specifically, consider a request sequence consisting of $n(n-1)$ jobs of size 1 followed by one job of size n. The greedy algorithm spreads the $n(n-1)$ size 1 jobs evenly across machines so that every machine has load $n-1$. Then it is forced to place the size n job on one of these machines creating a maximum load of $2n-1$. On the other hand, the optimal solution for this set of jobs has load n: we simply place the $n(n-1)$ jobs on $n-1$ machines (so that all these machines have load n), and we place the size n job on the one remaining machine.

This lower bound was for the particular greedy algorithm we considered here. Can other online algorithms for this problem have better competitive ratio? In precept, we will show lower bounds for this problem that hold for any online algorithm, not just the greedy algorithm. It turns out that the greedy algorithm is optimal for $n = 2$ and $n = 3$. Note that the competitive ratio of the greedy algorithm approaches 2 as the number of machines n gets large. It turns out that one can design algorithms whose competitive ratio is asymptotically better than 2 (the current best known competitive ratio is approximately 1.9201). However the analysis of these algorithms is beyond the scope of this course.

3 Learning from Expert Advice

We consider a simple online prediction problem that arises in machine learning. The setup is as follows: Suppose we are following a particular stock in the stock market for several days and on each day, our goal is predict whether the stock price will go up or down. As guidance, we have n experts analyzing the stock. Here is the sequence of events that happens every day: At the beginning of the day, each of the n experts announces a prediction about whether the stock will go up or down. Based on this (and based on past history), our goal is to make a decision about whether the stock will go up or down. At the end of the day, we learn what actually happened to stock. (For the purpose of this problem, we assume that the stock always goes either up or down, and does not stay at the same price). Our goal is to predict the movement of the stock accurately. In other words, we would like to make as few mistakes as possible.

Note that we make no assumptions whatsoever on the behavior of the stock price, about the way that the experts predictions are correlated to the actual behavior of the stock, etc. We would like to bound the number of mistakes that the online algorithm makes in terms of the number of mistakes made by the optimal algorithm. At this point, the problem seems completely hopeless: Clearly the “optimal” offline algorithm makes no mistakes. How could we possibly make correct predictions to compete with this ? Obviously we cannot do this. Instead we set ourselves a more modest, but still seemingly unattainable goal: can we compare the number of mistakes made by the online algorithm to the number of mistakes made by the best expert in hindsight ? If there is a least one expert that makes few mistakes, we would like to show that the online algorithm make very few mistakes. On the other hand, if every expert makes a lot of mistakes, then it is ok for the online algorithm to make a lot of mistakes as well. We will present two algorithms with precisely such a guarantee.

Possibly the first strategy that comes to mind is to predict based on the majority vote, i.e. if a majority of the experts say up, then we predict the stock will go up, otherwise we predict that the stock will go down. It is easy to see that this strategy can perform very badly compared to the best expert. It is possible that the majority vote is wrong every single time, while there is one expert who makes no mistakes at all. This suggests that we ought to look at the track record of experts in valuing their opinion. A natural idea in this direction is to associate weights with experts, update the weights to reflect the performance of the experts so far and factor these weights into our prediction. In fact this is the motivation for the following algorithm:

```
WEIGHTED MAJORITY

1. Initialize weights $w_1, \ldots, w_n$ to 1.

2. Given predictions $x_1, \ldots, x_n \in \{0, 1\}$,
   output prediction with highest total weight, i.e.
   1 if $\sum_{i:x_i=1} w_i \geq \sum_{i:x_i=0} w_i$
   0 otherwise.

3. When the correct answer $\ell \in \{0, 1\}$ is revealed, penalize each mistaken expert
   by multiplying the corresponding weight by 1/2. In other words,
   if $x_i \neq \ell$, then $w_i \to w_i/2$;
   if $x_i = \ell$, then $w_i$ is unchanged.

4. If there are more predictions, go to Step 2, else terminate.
```

Let M be the number of mistakes made by the Weighted Majority algorithm above, and let m be the number of mistakes by the best expert.

Theorem 3.1. *$M \leq 2.41(m + \log_2 n)$.*

Let W be the total weight of the experts. We will relate M and m indirectly by reasoning about W. To get some intuition, first think about whether it is possible for Weighted majority to make a lot of mistakes while the best expert makes very few. On the one hand, every time Weighted Majority makes a mistake, we will argue that W goes down. In other words, if M is large, W must be small at the end of the execution of the algorithm. We will also argue that if the best expert makes few mistakes, then the contribution to W (i.e. the weight of the best expert) must be large. Thus it not possible for Weighted Majority to make a lot of mistakes while the best expert makes very few. Quantifying this argument will give us the required bound.

Proof. Initially $W = n$. If the algorithm makes a mistake in a particular step, the total weight of experts that predict incorrectly must be at least half the total weight. Therefore, in Step 3, at least half the total weight is multiplied by $\frac{1}{2}$. This means that the total weight is reduced by a factor of at least $\frac{1}{4}$. If the algorithm makes M mistakes, we have the following upper bound on the final value of W:

$$W \leq n \left(\frac{3}{4}\right)^M \tag{1}$$

On the other hand, if the best expert makes m mistakes, then the final weight of the expert is $1/2^m$. Hence

$$W \geq \left(\frac{1}{2}\right)^m \tag{2}$$

Combining (1) and (2), we have

$$\begin{aligned}
\left(\frac{1}{2}\right)^m &\leq n\left(\frac{3}{4}\right)^M \\
\left(\frac{4}{3}\right)^M &\leq n2^m \\
M\log_2(4/3) &\leq m + \log_2 n \\
M &\leq \frac{1}{\log_2(4/3)}(m + \log_2 n) \\
&\leq 2.41(m + \log_2 n)
\end{aligned}$$

$\square$

Next, we show how we can improve the guarantee by modifying the algorithm in two ways:

1. Firstly, we use randomness. Instead of predicting the outcome with highest total weight, we view the weights as probabilities and predict each outcome with probability proportional to its weight. The advantage of the randomized approach is that it improves the worst case performance, i.e. it helps to reduce the 2.41 term in the guarantee we obtained before. Consider what happens when nearly half of the total weight predicts incorrectly. Weighted majority always predicts incorrectly, but the randomized version will make a mistake with probability close to 1/2 in this case. Using randomness has other advantages: there are settings where the "experts" may be algorithms themselves or other things that are not easily combined and here, taking weighted majority may be unnatural, while picking an "expert" with certain probability is meaningful.

2. Secondly, we multiply weights by β (instead of 1/2) for experts who predict incorrectly. The factor 1/2 we used in Weighted Majority earlier was arbitrary. Different values of the penalty β allow us to adjust the coefficients of m and $\log_2 n$ in the guarantee for $\mathbb{E}[M]$, the expected number of number of mistakes made by the algorithm.

Here is the new algorithm:

```
RANDOMIZED WEIGHTED MAJORITY

    1. Initialize weights $w_1, \dots, w_n$ to 1.

    2. Given predictions $x_1, \dots, x_n \in \{0, 1\}$,
       output $x_i$ with probability $w_i/W$ where $W = \sum_i w_i$.

    3. When the correct answer $\ell \in \{0, 1\}$ is revealed, penalize each mistaken expert
       by multiplying the corresponding weight by $\beta$. In other words,
       if $x_i \neq \ell$, then $w_i \to \beta \cdot w_i$;
       if $x_i = \ell$, then $w_i$ is unchanged.

    4. If there are more predictions, go to Step 2, else terminate.
```

Let M be the number of mistakes made by the Weighted Majority algorithm above (note that M is a random variable), and let m be the number of mistakes by the best expert.

Theorem 3.2.

$$\mathbb{E}[M] \leq \frac{m \ln(1/\beta) + \ln n}{1 - \beta} \tag{3}$$

Proof. As before, we will reason about W, the sum of weights of experts. Note that W is a random variable too. Let F_i be the fraction of the total weight of experts who have the *wrong* answer at the ith iteration. Suppose we have t iterations in all. Then the expected number of mistakes made is $\mathbb{E}[M] = \sum_{i=1}^{t} F_i$. Consider the ith iteration. A fraction F_i of the total weight was multiplied by β, hence the total weight (i.e. W) was multiplied by $(1 - (1 - \beta)F_i)$ in this iteration. Since the initial value of W was n, the final value is

$$W = n \cdot \prod_{i=1}^{t} (1 - (1 - \beta)F_i)$$

If the best expert makes m mistakes, then the weight of the best expert is β^m and the final value of W is at least β^m. This gives:

$$\beta^m \leq n \cdot \prod_{i=1}^{t} (1 - (1 - \beta)F_i)$$

Using $(1 - (1 - \beta)F_i) \leq e^{-(1-\beta)F_i}$

$$\beta^m \leq n \cdot \prod_{i=1}^{t} e^{-(1-\beta)F_i} = n \cdot e^{-(1-\beta)\sum_{i=1}^{t} F_i} = n \cdot e^{-(1-\beta)\mathbb{E}[M]}$$

Rearranging terms, we get:

$$e^{(1-\beta)\mathbb{E}[M]} \leq \left(\frac{1}{\beta}\right)^m \cdot n$$

Taking natural log of both sides:

$$(1-\beta)\mathbb{E}[M] \le m\ln(1/\beta) + \ln n$$

This gives the desired bound. □

Note that Theorem 3.2 does indeed give a stronger bound for $\mathbb{E}[M]$ for the Randomized Weighted Majority than Theorem 3.1 does for the number of mistakes M made by Weighted Majority. For instance, for $\beta = 1/2$, we get $\mathbb{E}[M] \le 1.39m + 2\ln n$, and for $\beta = 3/4$, we get $\mathbb{E}[M] \le 1.15m + 4\ln n$. By adjusting β, we can make the competitive ratio of the algorithm approach 1 at the expense of an increase in the additive constant.

Corollary 3.3. *For $\beta = 1-\epsilon$, we get the following bound:*

$$\mathbb{E}[M] \le \frac{m}{1-\epsilon} + \frac{\ln n}{\epsilon}$$

Proof. Substituting $\beta = 1-\epsilon$ in equation (3), we get

$$\mathbb{E}[M] \le m\frac{\ln(1/(1-\epsilon)}{\epsilon} + \frac{\ln n}{\epsilon} \tag{4}$$

Note that

$$\frac{1}{1-\epsilon} = 1 + \sum_{i=1}^{\infty} \epsilon^i \le e^{\sum_{i=1}^{\infty} \epsilon^i}$$

Hence

$$\ln\left(\frac{1}{1-\epsilon}\right) \le \sum_{i=1}^{\infty} \epsilon^i = \frac{\epsilon}{1-\epsilon}$$

Substituting this in (4), we get the desired inequality. □