Advanced Algorithms

Lecture 13: Algorithms with Predictions

Lecturer: Nathan Klein

1 Algorithms with Predictions

When solving a problem, you often might have some kind of *prediction* about the input. A natural setting where this arises is you may have some prediction about the future input in an online problem. In applications, often this comes from a machine learning model. For example, ride hailing companies (our stated application for online bipartite matching) probably have a good guess about how many requests will come from each area on a particular day and how many drivers will be available.

Given such a model, one option is to just blindly trust its prediction and execute the online algorithm based on the assumption there are no prediction errors (effectively converting it to an offline problem). I expect some companies do something like this. However this is usually a bad idea, because your model will sometimes be very, very wrong. Imagine a ride hailing app planning their whole day beforehand, and suddenly a storm breaks out, causing a massive change in demand pattern. And in these scenarios, you want your online matching algorithm to still have reasonable guarantees. Your algorithm should be robust to prediction error.

And so the natural idea is to somehow combine worst case guarantees with predictions. This is an emerging field called *algorithm with predictions* [MV22]. We will talk about using predictions in two simple settings. The first is binary search, and the second is the "rent or buy" problem.

We will keep things quite light today and leave some time at the end for questions about the upcoming midterm. If you enjoy this topic, I encourage you to check out this website and [MV22] (which is a nicely written survey that begins with the two examples we'll talk about today) for links to recent papers in this area and more in-depth examples.

1.1 Binary Search

Say you're in a library with 50 shelves in a row. The books are alphabetized by the author's last name, and there are $n = 2^{12}$ books. How many books will you have to look at to find your book of interest?

Most of us go into autopilot in this point and say look, this is binary search, and you need $\log(n) + 1 = 13$ queries. Problem solved.

That's true, but let's think a bit harder. Let's say the author's last name begins with "B". Which shelf do you think you'll start with? Probably something like the 3rd or 4th shelf, right? You have a *prediction* based on the location of "B" in the alphabet: it will likely be in one of the first few shelves.

So: how do we combine our prediction with the binary search algorithm? Matching what I expect would people do in a library, given that you predict your input is at index i, you start at index i. If you learn it's to the right of i, you search at index $i + 2$, then $i + 4$, then $i + 8$, etc., until you see an element j larger than your input. Finally, you do binary search in $[i + 1, j - 1]$. If it's to the left of i, you do the same but on the left.

Lemma 1.1. *Let h be a predictor so that $h(q)$ outputs a predicted index. Then, where $t(q)$ is the true position of q, the number of queries of this algorithm is at most $2\log(\eta)+2$, where $\eta = |h(q)-t(q)|$ is the error of the predictor.*

Proof. To stop looking in your first pass to the left or right, you need at most $\log(\eta)$ queries. This region has length at most 2η, requiring binary search at most $\log(2\eta)+1$ queries for a total of at most $2\log(\eta)+2$. □

This is a simple idea, but potentially quite impactful, since binary search is so commonly used. For example:

Lemma 1.2. *Let h be a predictor that with probability $1-\frac{1}{\log n}$ outputs a prediction with error at most $\log n$, and otherwise outputs something with arbitrarily bad error. Then, the expected runtime of binary search is at most $2\log\log(n)+O(1)$.*

Proof. With probability $1-\frac{1}{\log n}$, our runtime is at most $2\log(\log n)+2$. Otherwise, our runtime is at most $2\log(n)+2$, since the error is almost at most n. So, the expected runtime is at most:

$$(1-\frac{1}{\log n})(2\log\log(n)+2)+\frac{1}{\log n}(2\log(n)+2) \leq 2\log\log(n)+O(1)$$ □

This would correspond to an *exponential* speedup in expectation, a big deal for one of the most commonly used algorithms in existence. Of course, you would have to make the predictor very fast as well, but it could be as simple as looking at the first few bits.

We can prove something a bit more general as well:

Lemma 1.3. *Given a distribution over queries, the expected runtime is at most $2\log(\mathbb{E}[\eta])+2$ where $\mathbb{E}[\eta]$ is the expected error of the predictor.*

Proof. The runtime is:

$$2\mathbb{E}[\log(e)]+2 \leq 2\log(\mathbb{E}[e])+2$$

using Jensen's inequality: that for any convex function f, $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$. Here log is concave, so $-\log$ is convex, giving us $-\mathbb{E}[\log(e)] \leq -\log(\mathbb{E}[X])$, which implies the used inequality. □

So, even mildly good predictors can lead to good expected speedups while maintaining a worst case of about $2\log(n)$.

1.2 Rent or Buy

Say you like to go on trips in the Boston area. Every time you go, though, you need to rent a car for two days at the price of $\$r$. You could also just *buy* a car for $\$b$. (And let's say we ignore all the extra costs associated with car ownership, or bundle them into that price $\$b$). Now you want to find an algorithm for deciding whether you rent or buy that has a good worst-case competitive ratio.

The natural idea (or maybe not so natural – when I tell people this they usually say it's a terrible strategy) is this. Keep track of the amount you have spent renting. If that amount would exceed $\$b$, buy a car instead of renting that time.

Fact 1.4. *This strategy has a competitive ratio of 2.*

Proof. Consider the amount spent by the offline optimum. If it is less than $\$b$, then offline OPT does not buy and neither will we. So in this case we match the offline optimum.

Otherwise, it spends at least $\$b$. But our algorithm always spends at most $\$2b$. □

This is nice in theory, but a car is usually tens of thousands of dollars. So a competitive ratio of 2 might not sound very good – that's a lot of money to lose. A natural question to ask is: can we incorporate a *prediction* into the algorithm? Like in the binary search example, our goal would be to design an algorithm where:

1. If the prediction is bad, we don't get a competitive ratio much worse than 2. If the ratio is never worse than β, **we say the algorithm is β robust**.
2. If the prediction is close to correct, we get a competitive ratio better than 2: what we could have gotten without a prediction. If this ratio tends to α when the prediction error goes to 0, **we say the algorithm is α consistent**.

Let's try to come up with a consistent and robust algorithm together. Let p be the predicted number of days we will need the car. If $p \cdot r \gg b$, should we just buy the car right away?

No! This would completely ruin guarantee (1). We might end up spending $\$b$ when we could have just spent $\$r$, which is much much worse than 2 in the car example. Similarly, if $p \ll r$, should we decide to never buy, and rent forever? Again, no, this will lead to an arbitrarily bad ratio in the worst case.

Instead, let's be a bit more conservative and not blindly trust our prediction. We will introduce a value $\lambda \in (0, 1]$, where the smaller the λ the more you trust your prediction. To make things a bit easier on the eyes, let's make $r = 1$ (which we can do by renormalizing). Now the algorithm (first looked at and analyzed in [KPS18]) is, using our parameter $\lambda \in (0, 1]$:

1. If the prediction p is larger than b (i.e. it predicts you should buy), then buy when the amount you have spent renting would exceed $\$\lambda b$.
2. Otherwise, if it predicts you should rent, buy when the amount you have spent renting would exceed $\$\frac{b}{\lambda}$.

Let's prove now that this algorithm is both robust and consistent.

Lemma 1.5. *For any $\lambda \in (0, 1]$, the algorithm is $1 + \frac{1}{\lambda}$ robust: its competitive ratio never exceeds this value.*

Proof. First suppose $p > b$, so we predict that we should buy. If $OPT < \lambda b$, then it did not buy and neither will we so we get a competitive ratio of 1. Otherwise, $OPT \geq \lambda b$, but we never pay more than $(1 + \lambda)b$ giving us $1 + \frac{1}{\lambda}$.

Otherwise, $p < b$. If $OPT < b$ then it did not buy and neither will we, so we again get a competitive ratio of 1. Otherwise, we pay at most $\frac{b}{\lambda} + b$ whereas OPT pays at least b for a ratio of

$$\frac{b(1 + \frac{1}{\lambda})}{b} = 1 + \frac{1}{\lambda}$$

□

Lemma 1.6. *For any $\lambda \in (0, 1]$, the algorithm is $1 + \lambda$ consistent. In particular, it never exceeds a competitive ratio of:*

$$1 + \lambda + \frac{\eta}{(1 - \lambda)OPT}$$

where $\eta = |p - t|$ is the error of the prediction compared to the true number of days t.

Proof. First suppose $p > b$. If $t \leq \lambda b$, we get a competitive ratio of 1 as before. If $t > b$, then we buy and pay at most $(1+\lambda)b$ whereas OPT pays at least b, for a competitive ratio of $1+\lambda$, which is better than our guarantee. So the interesting case is when $OPT = t$ and $\lambda b < OPT < b$. Then, $\eta \geq b - OPT$ (since we predicted $p > b$), so $b \leq OPT + \eta$. Therefore:

$$\frac{(1+\lambda)b}{OPT} \leq \frac{(1+\lambda)(OPT+\eta)}{OPT} = 1 + \lambda + \frac{(1+\lambda)\eta}{OPT} \leq 1 + \lambda + \frac{\eta}{(1-\lambda)OPT}$$

where we used that for $\lambda \in (0, 1]$ we have $1 + \lambda = \frac{(1+\lambda)(1-\lambda)}{1-\lambda} = \frac{1-\lambda^2}{1-\lambda} \leq \frac{1}{1-\lambda}$.

We leave the case for when $p \leq b$ as an exercise, as it's quite similar in flavor and doesn't give much additional intuition. □

References

[KPS18] Ravi Kumar, Manish Purohit, and Zoya Svitkina. "Improving online algorithms via ML predictions". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, 9684–9693 (cit. on p. 3).

[MV22] Michael Mitzenmacher and Sergei Vassilvitskii. "Algorithms with predictions". In: *Commun. ACM* 65.7 (June 2022), 33–35. ISSN: 0001-0782. DOI: 10.1145/3528087 (cit. on p. 1).