On-line Algorithms in Machine Learning

On-line Algorithms

- Algorithms for settings where inputs/data arriving over time.

- Need to make decisions based on info so far, without knowing what will happen in the future.

  Rent-or-buy, elevator problem, paging, adaptive data structures (MTF, union-find, splay trees), investing.

  (As opposed to standard problems like sorting where you have all inputs at the start.)

- Can't hope to do as well as if you knew the future, but can hope to bound your "competitive ratio", or "regret".

Machine Learning

Machine Learning: improve performance at a task based on experience.

- E.g., prediction, games, etc.
- Can do in batch or on-line.
- We'll be talking about on-line learning.

Plan for today:

- Introduce "Picking a winner" ("combining expert advice") problem.
- Exponential weights (weighted majority) algorithm
- Relations to game theory, min-max theorem.

How to choose a job and decide when to switch?

- Say you are considering joining one of n startup companies.

- You watch their performance for a while and then join up with one.

- Of course, you only make money based on performance *while you're there*. And, past performance is no guarantee of future results.

- If your company is doing badly, you can switch to another. Of course, past performance is no guarantee of future results.

- What's a good strategy?

An idealized model

- n buckets.
- You are standing in one of them.
- At each time step, a ball falls into one of the buckets. If it's your bucket, you get \$1.
- Then you can choose to move if you want.
- Game ends when fullest bucket has d balls.

What can we say:

→ If we knew what was going to happen in advance, we could make \$$d$. (or more)

→ Bad news: For any deterministic strategy, and adversary can cause us to make \$0.

→ Question: How about a randomized strategy (adversary knows our strategy but can't see where we are)?

Formally: can we pick a randomized strategy such that for any sequence of events, our expected gain is high?

Strategies that don't work

- Pick a random company and stay there.

- Go to first company that gets $$d/2$.

- Go to company (bucket) that has done best so far. Break ties at random.

But, it turns out there's a simple strategy with the following amazing guarantee:

- For any sequence of events σ, your expected gain is at least $d - \frac{3}{2}\sqrt{d \log n}$.

  (Nearly as good as best possible for someone who doesn't switch jobs)

  (like "static optimality")

Exponential-Weights Algorithm

Give each bucket a "weight" starting at 1. Then repeat:

- Go to bucket i with probability w_i/W, where $W = \sum_i w_i$.

- When ball falls into some bucket, multiply its weight by $1 + \epsilon$.

E.g., what happens if all fall into one bin?

Analysis (tricky, tricky)

- Let W_t be total weight at time t.
- Let F_t be fraction of weight on bucket that ball falls in at time t.
- So, $W_{t+1} = W_t + \epsilon W_t F_t = W_t(1 + \epsilon F_t)$.
- $W_{\mathsf{Final}} = n \prod_t (1 + \epsilon F_t)$. This is $\geq (1 + \epsilon)^d$.
- Our expected gain is $\sum_t F_t$.

I have a product of numbers that's at least blah. What's the smallest their sum can be? Ans: when they're all equal.

Technically, this is easiest to calculate by taking logs...

Analysis, contd.

- Our expected gain is $\sum_t F_t$.
- $(1+\epsilon)^d \leq W_{\text{Final}} = n \prod_t (1 + \epsilon F_t)$.

Take logs:

- $d \ln(1+\epsilon) \leq \ln n + \sum_t \ln(1 + \epsilon F_t)$.
- We know $\ln(1+x) \leq x$, so:

$$\begin{aligned} d \ln(1+\epsilon) &\leq \ln n + \sum_t \epsilon F_t \\ &= \ln n + \epsilon[\text{Our Expected Gain}]. \end{aligned}$$

- Using $\ln(1+\epsilon) \geq \epsilon - \epsilon^2/2$, we get:

$$\text{Our expected gain} \geq d\left(1 - \frac{\epsilon}{2}\right) - \frac{\ln n}{\epsilon}$$

Now, plug in $\epsilon = \sqrt{(\ln n)/d}$ to get

$$d - \frac{3}{2}\sqrt{d \ln n}$$

So, this is pretty neat.

In fact, we can use this to prove the min-max theorem of 2-player zero-sum games (matrix games).

But first, a slight generalization.

Generalizations

Suppose each step we get a gain vector $[g_1, \ldots, g_n]$ where $0 \leq g_i \leq 1$.

- Alg is to multiply weight i by $(1+\epsilon)^{g_i}$.
- Can show that same result holds. For any ϵ,

$$\mathbf{E}[\text{Our gain}] \geq \left(1 - \frac{\epsilon}{2}\right) [\text{Best bucket}] - \frac{\ln n}{\epsilon}$$

- Details:
  - If weights were $[w_1, \ldots, w_n]$, then
  - $F = \mathbf{E}[\text{Our gain}] = \frac{1}{W}(w_1 g_1 + \ldots + w_n g_n)$.
  - New weights: $[w_1(1+\epsilon)^{g_1}, \ldots, w_n(1+\epsilon)^{g_n}]$.

    $\leq [w_1(1+\epsilon g_1), \ldots, w_n(1+\epsilon g_n)]$ since $g_i \leq 1$.
  - so, new sum of weights $\leq W(1+\epsilon F)$.

In Machine Learning, often thought of as combining predictions of multiple "experts". Want to do nearly as well as best expert in hindsight. Usually think of losses rather than gains.

The min-max theorem

Consider a 2-player zero-sum game:

Payoff to row player:

	N	D
N	-5	5
D	10	-10

Suppose that for any (randomized) strategy of your opponent, there exists a deterministic counter-strategy for you that guarantees you an expected gain $\geq V$.

Then, there exists a randomized strategy for you such that for any counter-strategy of the opponent, you get an expected gain $\geq V$.

We will prove min-max for matrices where values are all in range $[0, 1]$. (can always scale)

Proof of min-max theorem

Here is a randomized alg for repeated play of matrix game against an opponent: (say we are the row player)

- Create one bucket for each row.
- Pick bucket/row using our online algorithm.
- Opponent's play gives us the gain vector.

What would it mean for min-max to be false?

$\rightarrow$ If we know opponents randomized strategy, we can get expected gain $\geq V$, but if we have to choose our randomized strategy first, then opponent can force us to get $\leq V - \delta$.

$\rightarrow$ So, opponent should be able to force us to get $\leq V - \delta$ expected gain each time.

This contradicts our bound if we use $\epsilon = \delta$. Our gain per game is approaching $\mathrm{OPT}(1 - \epsilon/2)$.

In other words

If there was a gap (V versus $V - \delta$), then for *any* randomized strategy we choose, opponent who knows our strategy should be able to force us to get no more than $V - \delta$ on average per play.

- But, in T games, OPT row in hindsight gets total of at least VT
- We get at least $VT(1 - \epsilon/2) - \frac{\ln n}{\epsilon}$.
- Per play, we are getting $V(1 - \epsilon/2) - o(1)$.

Extensions

- Shifting bounds: compete not only against best single bucket/row/expert but against best strategy in hindsight that was allowed to switch a few times.

- What if there is a cost penalty for moving?

- bandit versions.