"No-regret" algorithms for repeated decisions

General framework:
- Algorithm has N options. World chooses cost vector. Can view as matrix like this (maybe infinite # cols)
- At each time step, algorithm picks row, life picks column.
  - Alg pays cost for action chosen.
  - Alg gets column as feedback (or just its own cost in the "bandit" model).
  - Need to assume some bound on max cost. Let's say all costs between 0 and 1.

Define average regret in T time steps as:
- (avg per-day cost of Alg) - (avg per-day cost of best fixed row in hindsight).
We want this to go to 0 or better as T gets large.
[called a "no-regret" algorithm]

History and development (abridged)
- [Hannan'57, Blackwell'56]: Alg. with regret $O((N/T)^{1/2})$.
  - Re-phraseing, need only $T = O(N/v^2)$ steps to get time-average regret down to $v$. (will call this quantity $T_v$)
  - Optimal dependence on $T$ (or $v$). Game-theorists viewed #rows N as constant, not so important as T, so pretty much done.
- Learning-theory 80s-90s: "combining expert advice". Imagine large class $C$ of N prediction rules.
  - Perform (nearly) as well as best $f \in C$.
  - [Littlestone,Warmuth'89]: Weighted-majority algorithm
    - $E[\text{cost}] \leq \text{OPT}(T) + (\log N)/v$.
    - Regret $O((\log N)/T)^{1/2}$, $T_v = O((\log N)/v^2)$.
  - Optimal as fn of N too, plus lots of work on exact constants, 2nd order terms, etc. [CFHHSW93]
- Extensions to bandit model (adds extra factor of N).
Bounds have only log dependence on # choices N.
So, conceivably can do well when N is exponential in natural problem size, if only could implement efficiently.
E.g., case of paths...

This is what we discussed last time.

[Kalai-Vempala’03] and [Zinkevich’03] settings

KV setting:
- Implicit set S of feasible points in R^n (E.g., m=#edges, S={indicator vectors 011010010 for possible paths})
- Assume have oracle for offline problem: given vector c, find x ∈ S to minimize c·x (E.g., shortest path algorithm)
- Use to solve online problem: on day i, must pick x_i ∈ S before c_i is given.
  \[(c_1 x_1 + \cdots + c_T x_T)/T \rightarrow \min_{x \in S} (c_1 x_1 + \cdots + c_T x_T)/T \]

Z setting:
- Assume S is convex.
- Allow c(x) to be a convex function over S.
- Assume given any y not in S, can algorithmically find nearest x \(\in S\).

[0,1] costs vs {0,1} costs.

We analyzed Randomized Wtd Majority for case that all costs in {0,1} (correct or mistake).
Here is a simple way to extend to [0,1].
- Given cost vector c, view c_i as bias of coin. Flip to create boolean vector c’, s.t. \(E[c’_i] = c_i\). Feed c’ to alg A.
- For any sequence of vectors c’, we have:
  \(E[A[\text{cost’}(A)]] \leq \min_i \text{cost’}(i) + \text{regret term}\)
- So, \(E[A[\text{cost’}(A)]] \leq E_i[\min_i \text{cost’}(i)] + \text{regret term}\)
- LHS is \(E[A[\text{cost}(A)]]\). (since A picks weights before seeing costs)
- RHS ≤ \(\min_i E_i[\text{cost}(i)] + [r.t.] = \min_i \text{cost}(i) + [r.t.]\)

In other words, costs between 0 and 1 just make the problem easier...

Online pricing

- Say you are selling lemonade (or a cool new software tool, or bottles of water at the world expo).
- Protocol #1: for t=1,2,…T
  - Seller sets price \(p_t\)
  - Buyer arrives with valuation \(v_t\)
  - If \(v_t \geq p_t\), buyer purchases and pays \(p_t\), else doesn’t.
  - \(v_t\) revealed to algorithm.
  - Repeat
- Protocol #2: same as protocol #1 but without \(v_t\) revealed.
  - What can we do now?
  - Assume all valuations in [1,h]
  - Goal: do nearly as well as best fixed price in hindsight.
A natural generalization

(going back to full-info setting)

- A natural generalization of our regret goal is: what if we also want that on rainy days, we do nearly as well as the best route for rainy days.
- And on Mondays, do nearly as well as best route for Mondays.
- More generally, have N "rules" (on Monday, use path P).
  Goal: simultaneously for each rule i, guarantee to do nearly as well as it on the time steps in which it fires.

For all i, want E[cost(alg)] ≤ (1+ε)cost(i) + O(ε² log N).
(cost(X) = cost of X on time steps where rule i fires.)

Can we get this?

A simple algorithm and analysis (all on one slide)

- Start with all rules at weight 1.
- At each time step, of the rules i that fire, select one with probability p_i ∝ w_i.
- Update weights:
  - If didn't fire, leave weight alone.
  - If did fire, raise or lower depending on performance compared to weighted average:
    - r ∈ (Σ p_i cost(i))/(|I^+| - cost(i))
    - w_i ← w_i(1+γ)
  - So, if rule i does exactly as well as weighted average, its weight drops a little. Weight increases if does better than weighted average by more than a (1+ε) factor. This ensures sum of weights doesn't increase.
- Final w_i = (1+ε)E[cost(alg)]/(1+ε)cost(i). So, exponent ≤ ε² log N.
- So, E[cost(alg)] ≤ (1+ε)cost(i) + O(ε² log N).

Can combine with [KV],[Z] too:

- Back to driving, say we are given N "conditions" to pay attention to (is it raining, is it a Monday?...).
- Each day satisfies some and not others. Want simultaneously for each condition (incl default) to do nearly as well as best path for those days.
- To solve, create N rules: "if day satisfies condition i, then use output of KV", where KV is an instantiation of KV algorithm you run on just the days satisfying that condition.
Other uses

• What if we want to adapt to change - do nearly as well as best recent expert?
• Say we know # time steps T in advance (or guess and double). Make T copies of each expert, one who wakes up on day i for each 0 ≤ i ≤ T-1.
• Our cost in previous t days is at most (1+ε)(best expert in last t days) + O(ε^{-1} log(NT)).
• (not best possible bound since extra log(T) but not bad).