15-859(B) Machine Learning Theory

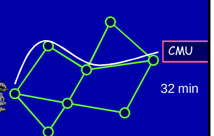
Bandit Problems and sleeping experts

Avrim Blum

Start with recap

Consider the following setting…

- Each morning, you need to pick one of N possible routes to drive to work.
- But traffic is different each day.
  - Not clear a priori which will be best.
  - When you get there you find out how long your route took. (And maybe others too or maybe not.)



- Want a strategy for picking routes so that in the long run, whatever the sequence of traffic patterns has been, you've done nearly as well as the best fixed route in hindsight. (In expectation, over internal randomness in the algorithm)

"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)

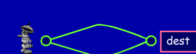
- At each time step, algorithm picks row, life picks column.
  - Alg pays cost for action chosen.
  - Alg gets column as feedback (or just its own cost in the "bandit" model).
  - Need to assume some bound on max cost. Let's say all costs between 0 and 1.

"No-regret" algorithms for repeated decisions

- 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]

To be clear…



World – life - fate (Algorithm)

1	0
0	1

- View of world/life/fate: unknown sequence LRLLRLRR...
- Goal: do well (in expectation) no matter what the sequence is.
- Note: Not trying to compete with best adaptive strategy – just best fixed choice in hindsight. Also assuming your decisions do not affect the future.
- Algorithms must be randomized or else it's hopeless.
- No-regret algorithms can potentially do better than playing minimax optimal, and never much worse.

History and development (abridged)

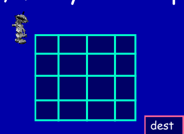
- [Hannan'57, Blackwell'56]: Alg. with regret $O((N/T)^{1/2})$.
  - Re-phrasing, need only $T = O(N/\varepsilon^2)$ steps to get time-average regret down to ε. (will call this quantity T_ε)
  - Optimal dependence on T (or ε). Game-theorists viewed #rows N as constant, not so important as T, so pretty much done.

History and development (abridged)

- [Hannan'57, Blackwell'56]: Alg. with regret $O((N/T)^{1/2})$.
  - Re-phrasing, need only $T = O(N/\varepsilon^2)$ steps to get time-average regret down to ε. (will call this quantity T_ε)
  - Optimal dependence on T (or ε). 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 $f \in C$.
  - [LittlestoneWarmuth'89]: Weighted-majority algorithm
    - $E[\text{cost}] \leq OPT(1+\varepsilon) + (\log N)/\varepsilon$.
    - Regret $O((\log N)/T)^{1/2}$. $T_\varepsilon = O((\log N)/\varepsilon^2)$.
  - 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).

Efficient implicit implementation for large 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…



- nxn grid has N = (2n choose n) possible paths.
- Recent years: series of results giving efficient implementation/alternatives in various settings, plus extensions to bandit model.

Efficient implicit implementation for large N…

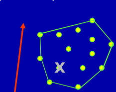
- Recent years: series of results giving efficient implementation/alternatives in various settings:
  - [HelmboldSchapire97]: best pruning of given DT.
  - [BChawlaKalai02]: list-update problem.
  - [TakimotoWarmuth02]: online shortest path in DAGs.
  - [KalaiVempala03]: elegant setting generalizing all above
    - Online linear optimization
  - [Zinkevich03]: elegant setting generalizing all above
    - Online convex optimization
  - [AwerbuchKleinberg04][McMahanB04]:[KV]→bandit model
  - [Kleinberg,FlaxmanKalaiMcMahan05]: [Z03] → bandit model
  - [DaniHayes06]: improve bandit convergence rate
  - [GolovinStreeter08]: online submodular fn maximization

  More…

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

[KV] setting:

- Implicit set S of feasible points in R^m. (E.g., m=#edges, S={indicator vectors 011010010 for possible paths})
- Assume have oracle for offline problem: given vector c, find $x \in S$ to minimize $c \cdot x$. (E.g., shortest path algorithm)
- Use to solve online problem: on day t, must pick $x_t \in S$ before c_t is given.
- $(c_1 \cdot x_1 + \ldots + c_T \cdot x_T)/T \to \min_{x \in S} x \cdot (c_1 + \ldots + c_T)/T$.



[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 $x \in S$.

Plan for today

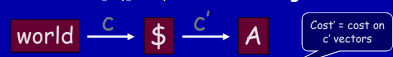
- Bandit algorithms
- Sleeping experts
- But first, a quick discussion of [0,1] vs {0,1} costs for RWM algorithm

[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:
  - $E_A[\text{cost}'(A)] \leq \min_i \text{cost}'(i)$ + [regret term]
- So, $E_\$[E_A[\text{cost}'(A)]] \leq E_\$[\min_i \text{cost}'(i)]$ + [regret term]
- LHS is $E_A[\text{cost}(A)]$.
- RHS $\leq \min_i E_\$[\text{cost}'(i)]$ + [r.t.] = $\min_i[\text{cost}(i)]$ + [r.t.]

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

Experts → Bandit setting

- In the bandit setting, only get feedback for the action we choose. Still want to compete with best action in hindsight.
- [ACFS02] give algorithm with cumulative regret $O((TN \log N)^{1/2})$. [average regret $O(((N \log N)/T)^{1/2})$.]
- Here, will give more generic, simpler approach but with worse bounds ($T^{1/2} \to T^{2/3}$).

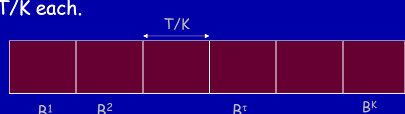
Experts → Bandit setting

Given: algorithm A for full-info setting with regret ≤ R(T).

Goal: use in black-box manner for bandit problem.

Preliminaries:

- First, suppose we break our T time steps into K blocks of size T/K each.



- Use same distrib throughout block and update based on average cost vector c^τ for block τ.
- Then, will get regret ≤ R(K) × T/K.

Because really paying T/K × c^τ per block

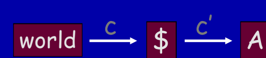
- What if we instead update on cost vector $c' \in [0,1]^N$ that's a random variable whose expectation is correct?

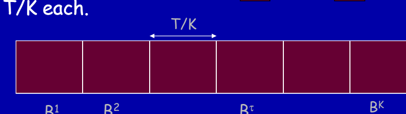
Experts → Bandit setting

Given: algorithm A for full-info setting with regret ≤ R(T).

Goal: use in black-box manner for bandit problem.

Preliminaries:

- First, suppose  blocks of size T/K each.



- Use same distrib throughout block and update based on average cost vector c^τ for block τ.
- Then, will get regret ≤ R(K) × T/K.

Because really paying T/K × c^τ per block

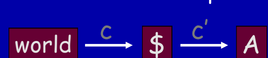
- What if we instead update on cost vector $c' \in [0,1]^N$ that's a random variable whose expectation is correct?

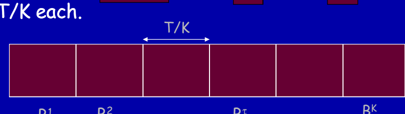
Experts → Bandit setting

Given: algorithm A for full-info setting with regret ≤ R(T).

Goal: use in black-box manner for bandit problem.

Preliminaries:

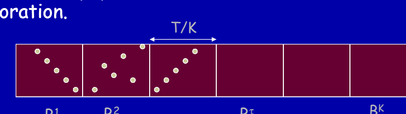
- First, suppose  blocks of size T/K each.



- Do at least as well by {0,1}→[0,1] argument. Still get regret bound R(K) × T/K.
- How does this help us for bandit problem?
- What if we instead update on cost vector $c' \in [0,1]^N$ that's a random variable whose expectation is correct?

Experts → Bandit setting

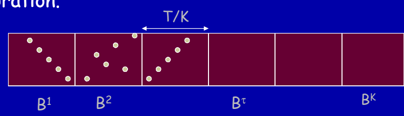
- For bandit problem, for each action, pick random time step in each block to try it as "exploration".
- Define c' only wrt these exploration steps.
- Just have to pay an extra at most NK for cost of this exploration.



- Do at least as well by {0,1}→[0,1] argument. Still get regret bound R(K) × T/K.
- How does this help us for bandit problem?
- What if we instead update on cost vector $c' \in [0,1]^N$ that's a random variable whose expectation is correct?

Experts → Bandit setting

- For bandit problem, for each action, pick random time step in each block to try it as "exploration".
- Define c' only wrt these exploration steps.
- Just have to pay an extra at most NK for cost of this exploration.



- Final bound: $R(K) \times T/K + NK$.
- Using $K = (T/N)^{2/3}$ and bound from RWM, get cumulative regret bound of $O(T^{2/3}N^{1/3} \log N)$.

A natural generalization

- 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 $E[cost_i(alg)] \leq (1+\varepsilon)cost_i(i) + O(\varepsilon^{-1}\log N)$.
  ($cost_i(X)$ = cost of X on time steps where rule i fires.)
- Can we get this? (Going back to full-info setting)

A natural generalization

- This generalization is esp natural in machine learning for combining multiple if-then rules.
- E.g., document classification. Rule: "if <word-X> appears then predict <Y>". E.g., if has football then classify as sports.
- So, if 90% of documents with football *are* about sports, we should have error $\leq$ 11% on them.

  "Specialists" or "sleeping experts" problem.

- Assume we have N rules, explicitly given.
- For all i, want $E[cost_i(alg)] \leq (1+\varepsilon)cost_i(i) + O(\varepsilon^{-1}\log N)$.
  ($cost_i(X)$ = cost of X on time steps where rule i fires.)

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 $p_i \propto w_i$.
- Update weights:
  - If didn't fire, leave weight alone.
  - If did fire, raise or lower depending on performance compared to weighted average:
    - $r_i = [\sum_j p_j\, cost(j)]/(1+\varepsilon) - cost(i)$
    - $w_i \leftarrow w_i(1+\varepsilon)^{r_i}$
  - 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 $(1+\varepsilon)$ factor. This ensures sum of weights doesn't increase.
- Final $w_i = (1+\varepsilon)^{E[cost_i(alg)]/(1+\varepsilon) - cost_i(i)}$. So, exponent $\leq \varepsilon^{-1}\log N$.
- So, $E[cost_i(alg)] \leq (1+\varepsilon)cost_i(i) + O(\varepsilon^{-1}\log N)$.

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_i", where KV_i 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 $0 \leq i \leq T-1$.
- Our cost in previous t days is at most $(1+\varepsilon)$(best expert in last t days) + $O(\varepsilon^{-1} \log(NT))$.
- (not best possible bound since extra log(T) but not bad).