Online Learning

Your guide:

Avrim Blum

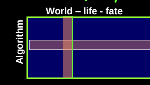
Carnegie Mellon University

[Machine Learning Summer School 2012]

Recap

"No-regret" algorithms for repeated decisions:

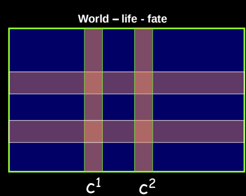
- Algorithm has N options. World chooses cost vector. Can view as matrix like this (maybe infinite # cols)



- At each time step, algorithm picks row, life picks column.
  - Alg pays cost (or gets benefit) for action chosen.
  - Alg gets column as feedback (or just its own cost/benefit in the "bandit" model).
  - Goal: do nearly as well as best fixed row in hindsight.

RWM

$(1-\varepsilon c_1^2)(1-\varepsilon c_1^1)1$
$(1-\varepsilon c_2^2)(1-\varepsilon c_2^1)1$
$(1-\varepsilon c_3^2)(1-\varepsilon c_3^1)1$
. . 1
. . 1
$(1-\varepsilon c_n^2)(1-\varepsilon c_n^1)1$



scaling so costs in [0,1]

Guarantee: $E[\text{cost}] \leq OPT + 2(OPT \cdot \log n)^{1/2}$

Since $OPT \leq T$, this is at most $OPT + 2(T\log n)^{1/2}$.

So, regret/time step $\leq 2(T\log n)^{1/2}/T \to 0$.

[ACFS02]: applying RWM to bandits

- What if only get your own cost/benefit as feedback?
- Use of RWM as subroutine to get algorithm with cumulative regret $O((TN \log N)^{1/2})$.
  [average regret $O(((N \log N)/T)^{1/2})$.]
- Will do a somewhat weaker version of their analysis (same algorithm but not as tight a bound).
- For fun, talk about it in the context of online pricing…

Online pricing

- Say you are selling lemonade (or a cool new software tool, or bottles of water at the world cup).
- 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.
  - Repeat.

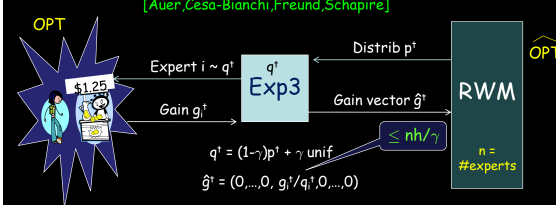
View each possible price as a different row/expert

- Assume all valuations $\leq h$.
- Goal: do nearly as well as best fixed price in hindsight.

Multi-armed bandit problem

Exponential Weights for Exploration and Exploitation ($\exp^3$)

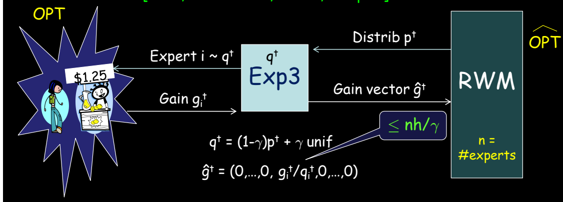
[Auer,Cesa-Bianchi,Freund,Schapire]



1. RWM believes gain is: $p^t \cdot \hat{g}^t = p_i^t(g_i^t/q_i^t) \equiv g^t_{RWM}$
2. $\sum_t g^t_{RWM} \geq \widehat{OPT}/(1+\epsilon) - O(\epsilon^{-1}\, nh/\gamma \log n)$
3. Actual gain is: $g_i^t = g^t_{RWM}(q_i^t/p_i^t) \geq g^t_{RWM}(1-\gamma)$
4. $E[\widehat{OPT}] \geq OPT$. Because $E[\hat{g}_j^t] = (1-q_j^t)0 + q_j^t(g_j^t/q_j^t) = g_j^t$,
   so $E[\max_j[\sum_t \hat{g}_j^t]] \geq \max_j[E[\sum_t \hat{g}_j^t]] = OPT$.

Multi-armed bandit problem

Exponential Weights for Exploration and Exploitation (exp³)
[Auer,Cesa-Bianchi,Freund,Schapire]



Conclusion ($\gamma = \epsilon$):
E[Exp3] $\geq$ OPT/$(1+\epsilon)^2$ - $O(\epsilon^{-2}$ nh log(n))

Balancing would give O((OPT nh log n)$^{2/3}$) in bound because of ϵ^{-2}. But can reduce to ϵ^{-1} and O((OPT nh log n)$^{1/2}$) more care in analysis.

A natural generalization

(Going back to full-info setting, thinking about paths…)

- 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?

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)] $\leq$ $(1+\varepsilon)cost_i(i)$ + $O(\varepsilon^{-1}\log N)$.

Lots of uses

- Can combine multiple if-then rules
- Can combine multiple learning algorithms:
  - Back to driving, say we are given N "conditions" to pay attention to (is it raining?, is it a Monday?, …).
  - Create N rules: "if day satisfies condition i, then use output of Alg_i", where Alg_i is an instantiation of an experts algorithm you run on just the days satisfying that condition.
  - Simultaneously, for each condition i, do nearly as well as Alg_i which itself does nearly as well as best path for condition i.

Adapting to change

- What if we want to adapt to change - do nearly as well as best recent expert?
- For each expert, instantiate copy who wakes up on day t for each $0 \leq t \leq T-1$.
- Our cost in previous t days is at most $(1+\epsilon)$(best expert in last t days) + $O(\epsilon^{-1}\log(NT))$.
- (not best possible bound since extra log(T) but not bad).

Summary

Algorithms for online decision-making with strong guarantees on performance compared to best fixed choice.

- Application: play repeated game against adversary. Perform nearly as well as fixed strategy in hindsight.

Can apply even with very limited feedback.

- Application: which way to drive to work, with only feedback about your own paths; online pricing, even if only have buy/no buy feedback.

More general forms of regret

1. "best expert" or "external" regret:
   - Given n strategies. Compete with best of them in hindsight.
2. "sleeping expert" or "regret with time-intervals":
   - Given n strategies, k properties. Let S_i be set of days satisfying property i (might overlap). Want to simultaneously achieve low regret over each S_i.
3. "internal" or "swap" regret: like (2), except that S_i = set of days in which we chose strategy i.

Internal/swap-regret

- E.g., each day we pick one stock to buy shares in.
  - Don't want to have regret of the form "every time I bought IBM, I should have bought Microsoft instead".
- Formally, regret is wrt optimal function $f:\{1,\ldots,n\}\rightarrow\{1,\ldots,n\}$ such that every time you played action j, it plays f(j).

Weird... why care?

"Correlated equilibrium"

- Distribution over entries in matrix, such that if a trusted party chooses one at random and tells you your part, you have no incentive to deviate.
- E.g., Shapley game.

	R	P	S
R	-1,-1	-1,1	1,-1
P	1,-1	-1,-1	-1,1
S	-1,1	1,-1	-1,-1

In general-sum games, if all players have low swap-regret, then empirical distribution of play is apx correlated equilibrium.

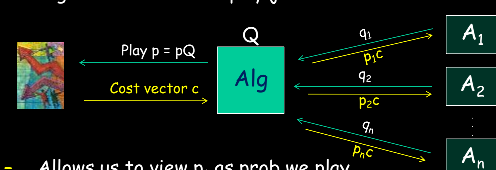
Internal/swap-regret, contd

Algorithms for achieving low regret of this form:

- Foster & Vohra, Hart & Mas-Colell, Fudenberg & Levine.
- Will present method of [BM05] showing how to convert any "best expert" algorithm into one achieving low swap regret.

Can convert any "best expert" algorithm A into one achieving low swap regret. Idea:

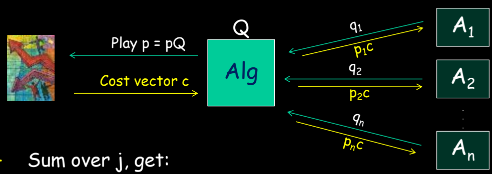
- Instantiate one copy A_j responsible for expected regret over times we play j.



- Allows us to view p_j as prob we play action j, or as prob we play alg A_j.
- Give A_j feedback of p_jc.
- A_j guarantees $\sum_t (p_j^t c^t)\cdot q_j^t \leq \min_i \sum_t p_j^t c_i^t$ + [regret term]
- Write as: $\sum_t p_j^t (q_j^t \cdot c^t) \leq \min_i \sum_t p_j^t c_i^t$ + [regret term]

Can convert any "best expert" algorithm A into one achieving low swap regret. Idea:

- Instantiate one copy A_j responsible for expected regret over times we play j.



- Sum over j, get:

$$\Sigma_t p^tQ^tc^t \leq \Sigma_j \min_i \Sigma_t p_j^tc_i^t + n[\text{regret term}]$$

Our total cost

For each j, can move our prob to its own i=f(j)

- Write as: $\Sigma_t p_j^t(q_j^t \cdot c^t) \leq \min_i \Sigma_t p_j^tc_i^t + [\text{regret term}]$

Itinerary

- Stop 1: Minimizing regret and combining advice.
  - Randomized Wtd Majority / Multiplicative Weights alg
  - Connections to game theory
- Stop 2: Extensions
  - Online learning from limited feedback (bandit algs)
  - Algorithms for large action spaces, sleeping experts
- Stop 3: Powerful online LTF algorithms
  - Winnow, Perceptron
- Stop 4: Powerful tools for using these algorithms
  - Kernels and Similarity functions
- Stop 5: Something completely different
  - Distributed machine learning

Transition…

- So far, we have been examining problems of selecting among choices/algorithms/experts given to us from outside.
- Now, turn to design of online algorithms for learning over data described by features.

A typical ML setting

- Say you want a computer program to help you decide which email messages are urgent and which can be dealt with later.
- Might represent each message by n features. (e.g., return address, keywords, header info, etc.)
- On each message received, you make a classification and then later find out if you messed up.
- Goal: if there exists a "simple" rule that works (is perfect? low error?) then our alg does well.

Simple example: disjunctions

- Suppose features are boolean: $X = \{0,1\}^n$.
- Target is an OR function, like $x_3 \vee x_9 \vee x_{12}$.
- Can we find an on-line strategy that makes at most n mistakes? (assume perfect target)
- Sure.
  - Start with $h(x) = x_1 \vee x_2 \vee \ldots \vee x_n$
  - Invariant: {vars in h} $\supseteq$ {vars in f}
  - Mistake on negative: throw out vars in h set to 1 in x. Maintains invariant and decreases $|h|$ by 1.
  - No mistakes on positives. So at most n mistakes total.

Simple example: disjunctions

- Suppose features are boolean: $X = \{0,1\}^n$.
- Target is an OR function, like $x_3 \vee x_9 \vee x_{12}$.
- Can we find an on-line strategy that makes at most n mistakes? (assume perfect target)
- Compare to "experts" setting:
  - Could define 2^n experts, one for each OR fn.
  - #mistakes $\leq$ log(# experts)
  - This way is much more efficient…
  - …but, requires some expert to be perfect.

Simple example: disjunctions

- But what if we believe only r out of the n variables are relevant?
- I.e., in principle, should be able to get only $O(\log n^r) = O(r \log n)$ mistakes.
- Can we do it efficiently?

Winnow algorithm

- Winnow algorithm for learning a disjunction of r out of n variables. eg $f(x)= x_3 \vee x_9 \vee x_{12}$
- h(x): predict pos iff $w_1x_1 + \ldots + w_nx_n \geq n$.
- Initialize $w_i = 1$ for all i.
  - Mistake on pos: $w_i \leftarrow 2w_i$ for all $x_i=1$.
  - Mistake on neg: $w_i \leftarrow 0$ for all $x_i=1$.

Winnow algorithm

- Winnow algorithm for learning a disjunction of r out of n variables. eg $f(x)= x_3 \vee x_9 \vee x_{12}$
- h(x): predict pos iff $w_1x_1 + \ldots + w_nx_n \geq n$.
- Initialize $w_i = 1$ for all i.
  - Mistake on pos: $w_i \leftarrow 2w_i$ for all $x_i=1$.
  - Mistake on neg: $w_i \leftarrow 0$ for all $x_i=1$.
- Thm: Winnow makes at most O(r log n) mistakes.

Proof:

- Each M.o.p. doubles at least one relevant weight (and note that rel wts never set to 0). At most r(1+log n) of these.
- Each M.o.p. adds < n to total weight. Each M.o.n removes at least n from total weight. So #(M.o.n) $\leq$ 1+ #(M.o.p).
- That's it!

A generalization

- Winnow algorithm for learning a linear separator with non-neg integer weights: e.g., $2x_3 + 4x_9 + x_{10} + 3x_{12} \geq 5$.
- h(x): predict pos iff $w_1x_1 + \ldots + w_nx_n \geq n$.
- Initialize $w_i = 1$ for all i.
  - Mistake on pos: $w_i \leftarrow w_i(1+\epsilon)$ for all $x_i=1$.
  - Mistake on neg: $w_i \leftarrow w_i/(1+\epsilon)$ for all $x_i=1$.
  - Use $\epsilon = O(1/W)$, W = sum of wts in target.

Thm: Winnow makes at most $O(W^2 \log n)$ mistakes.

Winnow for general LTFs

More generally, can show the following:

Suppose $\exists$ w* s.t.:

- $w^* \cdot x \geq c$ on positive x,
- $w^* \cdot x \leq c - \gamma$ on negative x.

Then mistake bound is

- $O((L_1(w^*)/\gamma)^2 \log n)$

Multiply by $L_\infty(X)$ if features not {0,1}.

Perceptron algorithm

An even older and simpler algorithm, with a bound of a different form.

Suppose $\exists$ w* s.t.:

- $w^* \cdot x \geq \gamma$ on positive x,
- $w^* \cdot x \leq -\gamma$ on negative x.

Then mistake bound is

- $O((L_2(w^*)L_2(x)/\gamma)^2)$

L_2 margin of examples

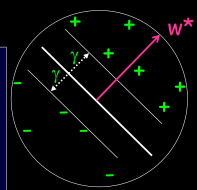
Perceptron algorithm

Thm: Suppose data is consistent with some LTF $w^* \cdot x > 0$, where we scale so $L_2(w^*)=1$, $L_2(x) \leq 1$,

$\gamma = \min_x |w^* \cdot x|$

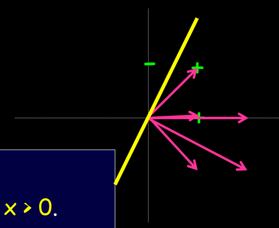
Then # mistakes $\leq 1/\gamma^2$.

Algorithm:
Initialize $w=0$. Use $w \cdot x > 0$.
- Mistake on pos: $w \leftarrow w+x$.
- Mistake on neg: $w \leftarrow w-x$.



Perceptron algorithm

Example: (0,1) –
(1,1) +
(1,0) +



Algorithm:
Initialize $w=0$. Use $w \cdot x > 0$.
- Mistake on pos: $w \leftarrow w+x$.
- Mistake on neg: $w \leftarrow w-x$.

Analysis

Thm: Suppose data is consistent with some LTF $w^* \cdot x > 0$, where $||w^*||=1$ and

$\gamma = \min_x |w^* \cdot x|$ (after scaling so all $||x|| \leq 1$)

Then # mistakes $\leq 1/\gamma^2$.

Proof: consider $|w \cdot w^*|$ and $||w||$
- Each mistake increases $|w \cdot w^*|$ by at least γ.
  $(w + x) \cdot w^* = w \cdot w^* + x \cdot w^* \geq w \cdot w^* + \gamma$.
- Each mistake increases $w \cdot w$ by at most 1.
  $(w + x) \cdot (w + x) = w \cdot w + 2(w \cdot x) + x \cdot x \leq w \cdot w + 1$.
- So, in M mistakes, $\gamma M \leq |w \cdot w^*| \leq ||w|| \leq M^{1/2}$.
- So, $M \leq 1/\gamma^2$.

What if no perfect separator?

In this case, a mistake could cause $|w \cdot w^*|$ to drop.
Impact: magnitude of $x \cdot w^*$ in units of γ.
Mistakes(perceptron) $\leq 1/\gamma^2 + O$(how much, in units of γ, you would have to move the points to all be correct by γ)

Proof: consider $|w \cdot w^*|$ and $||w||$
- Each mistake increases $|w \cdot w^*|$ by at least γ.
  $(w + x) \cdot w^* = w \cdot w^* + x \cdot w^* \geq w \cdot w^* + \gamma$.
- Each mistake increases $w \cdot w$ by at most 1.
  $(w + x) \cdot (w + x) = w \cdot w + 2(w \cdot x) + x \cdot x \leq w \cdot w + 1$.
- So, in M mistakes, $\gamma M \leq |w \cdot w^*| \leq ||w|| \leq M^{1/2}$.
- So, $M \leq 1/\gamma^2$.

What if no perfect separator?

In this case, a mistake could cause $|w \cdot w^*|$ to drop.
Impact: magnitude of $x \cdot w^*$ in units of γ.
Mistakes(perceptron) $\leq 1/\gamma^2 + O$(how much, in units of γ, you would have to move the points to all be correct by γ)

Note that γ was not part of the algorithm.

So, mistake-bound of Perceptron $\leq \min_\gamma$ (above).

Equivalently,
mistake bound $\leq \min_{w^*} ||w^*||^2 + O(\text{hinge loss}(w^*))$.