15-859(B) Machine Learning Theory

Avrim Blum
01/15/14

Lecture 2: Online learning I

Mistake-bound model:
- Basic results, halving and StdOpt algorithms
- Connections to information theory

Combining "expert advice":
- (Randomized) Weighted Majority algorithm
- Regret-bounds and connections to game-theory

Recap from last time

- Last time: PAC model and Occam's razor.
  - If data set has $m \geq (1/\varepsilon)[s \ln(2) + \ln(1/\delta)]$ examples, then whp any consistent hypothesis with size(h) < s has err(h) < ε.
  - Equivalently, suffices to have $s \leq (\varepsilon m - \ln(1/\delta))/\ln(2)$
  - "compression $\Rightarrow$ learning"
- [KV] book has esp. good coverage of this and related topics.
- Occam bounds $\Rightarrow$ any class is learnable if computation time is no object.

Recap: open problems

Can one efficiently PAC-learn…

- C={fns with only O(log n) relevant variables}? (or even O(loglog n) or $\omega(1)$ relevant variables)? This is a special case of DTs, DNFs.
- Monotone DNF over uniform D?
- Weak agnostic learning of monomials.

Online learning

- What if we don't want to make assumption that data is coming from some fixed distribution? Or any assumptions at all?
- Can no longer talk about past performance predicting future results.
- Can we hope to say anything interesting??

Idea: mistake bounds & regret bounds.

Mistake-bound model

- View learning as a sequence of stages.
- In each stage, algorithm is given *x*, asked to predict *f(x)*, and then is told correct value.
- Make no assumptions about order of examples.
- Goal is to bound total number of mistakes.

Alg A learns class C with mistake bound M if A makes $\leq$ M mistakes on any sequence of examples consistent with some f $\in$ C.

Mistake-bound model

Alg A learns class C with mistake bound M if A makes $\leq$ M mistakes on any sequence of examples consistent with some f $\in$C.

- Note: can no longer talk about "how much data do I need to converge?" Maybe see same examples over again and learn nothing new. But that's OK if don't make mistakes either…
- Want mistake bound poly(n, s), where n is size of example and s is size of smallest consistent f $\in$ C.
- C is **learnable** in MB model if exists alg with mistake bound and running time per stage poly(n,s).

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?
- Sure.
  - Start with $h(x) = x_1 \vee x_2 \vee ... \vee x_n$
  - 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

- Algorithm makes at most n mistakes.
- No deterministic alg can do better:

1 0 0 0 0 0 0 $+$ or $-$?
0 1 0 0 0 0 0 $+$ or $-$?
0 0 1 0 0 0 0 $+$ or $-$?
0 0 0 1 0 0 0 $+$ or $-$?
...

MB model properties

An alg A is "conservative" if it only changes its state when it makes a mistake.

Claim: if C is learnable with mistake-bound M, then it is learnable by a conservative alg.

Why?

- Take generic alg A. Create new conservative A' by running A, but rewinding state if no mistake is made.
- Still $\leq$ M mistakes because A still sees a legal sequence of examples.

MB learnable ⇒ PAC learnable

Say alg A learns C with mistake-bound M.

Transformation 1:

- Run (conservative) A until it produces a hyp h that survives $\geq (1/\varepsilon)\ln(M/\delta)$ examples.
- Pr(fooled by any given h) $\leq \delta/M$.
- Pr(fooled ever) $\leq \delta$.

Uses at most $(M/\varepsilon)\ln(M/\delta)$ examples total.

- Fancier method gets $O(\varepsilon^{-1}[M + \ln(1/\delta)])$

One more example...

- Say we view each example as an integer between 0 and 2^n-1.
- $C = \{[0,a] : a < 2^n\}$. (device fails if it gets too hot)
- In PAC model we could just pick any consistent hypothesis. Does this work in MB model?
- What would work?

What can we do with unbounded computation time?

- "Halving algorithm": take majority vote over all consistent $h \in C$. Makes at most $\lg(|C|)$ mistakes.
- What if C has functions of different sizes?
- For any (prefix-free) representation, can make at most 1 mistake per bit of target.
  - give each h a weight of $(\frac{1}{2})^{\text{size}(h)}$
  - Total sum of weights ≤ 1.
  - Take weighted vote. Each mistake removes at least $\frac{1}{2}$ of total weight left.

What can we do with unbounded computation time?

- "Halving algorithm": take majority vote over all consistent $h \in C$. Makes at most $\lg(|C|)$ mistakes.
- What if we had a "prior" p over fns in C?
  - Weight the vote according to p. Make at most $\lg(1/p_f)$ mistakes, where f is target fn.
- What if f was really chosen according to p?
  - Expected number of mistakes $\leq \sum_h [p_h \lg(1/p_h)]$ = entropy of distribution p.

Is halving alg optimal?

- Not necessarily (see hwk TBA).
- Can think of MB model as 2-player game between alg and adversary.
  - Adversary picks x to split C into $C_-(x)$ and $C_+(x)$. [fns that label x as – or + respectively]
  - Alg gets to pick one to throw out.
  - Game ends when all fns left are equivalent.
  - Adversary wants to make game last as long as possible.
- OPT(C) = MB when both play optimally.

Is halving alg optimal?

- Halving algorithm: throw out larger set.
- Optimal algorithm: throw out set with larger mistake bound.
- You'll think about this more on the hwk…

What if there is no perfect function?

Think of as $h \in C$ as "experts" giving advice to you. Want to do nearly as well as best of them in hindsight.

These are called "regret bounds".
➢Show that our algorithm does nearly as well as best predictor in some class.

We'll look at a strategy whose running time is $O(|C|)$. So, only computationally efficient when C is small.

Using "expert" advice

Say we want to predict the stock market.

- We solicit n "experts" for their advice. (Will the market go up or down?)
- We then want to use their advice somehow to make our prediction. E.g.,

Expt 1	Expt 2	Expt 3	neighbor's dog	truth
down	up	up	up	up
down	up	up	down	down
…	…	…	…	…

Can we do nearly as well as best in hindsight?

["expert" ≡ someone with an opinion. Not necessarily someone who knows anything.]
[note: would be trivial in PAC (i.i.d.) setting]

Using "expert" advice

If one expert is perfect, can get $\leq \lg(n)$ mistakes with halving alg.

But what if none is perfect? Can we do nearly as well as the best one in hindsight?

Strategy #1:

- Iterated halving algorithm. Same as before, but once we've crossed off all the experts, restart from the beginning.
- Makes at most $\lg(n)[OPT+1]$ mistakes, where OPT is #mistakes of the best expert in hindsight.

Seems wasteful. Constantly forgetting what we've "learned". Can we do better?

Weighted Majority Algorithm

Intuition: Making a mistake doesn't completely disqualify an expert. So, instead of crossing off, just lower its weight.

Weighted Majority Alg:

- Start with all experts having weight 1.
- Predict based on weighted majority vote.
- Penalize mistakes by cutting weight in half.

Weights:	1	1	1	1
Predictions:	U	U	U	D
Weights:	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1

We predict: U Truth: D

Analysis: do nearly as well as best expert in hindsight

- M = # mistakes we've made so far.
- m = # mistakes best expert has made so far.
- W = total weight (starts at n).
- After each mistake, W drops by at least 25%. So, after M mistakes, W is at most $n(3/4)^M$.
- Weight of best expert is $(1/2)^m$. So,

$$(1/2)^m \leq n(3/4)^M$$
$$(4/3)^M \leq n2^m$$
$$M \leq 2.4(m + \lg n)$$

constant ratio

Randomized Weighted Majority

2.4(m + lg n) not so good if the best expert makes a mistake 20% of the time. Can we do better? Yes.

- Instead of taking majority vote, use weights as probabilities. (e.g., if 70% on up, 30% on down, then pick 70:30) Idea: smooth out the worst case.
- Also, generalize $\frac{1}{2}$ to 1- ε.

Solves to: $M \leq \frac{-m\ln(1-\varepsilon) + \ln(n)}{\varepsilon} \approx (1+\varepsilon/2)m + \frac{1}{\varepsilon}\ln(n)$

M = expected #mistakes

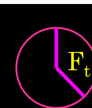
$M \leq 1.39m + 2\ln n \quad \leftarrow \varepsilon = 1/2$

$M \leq 1.15m + 4\ln n \quad \leftarrow \varepsilon = 1/4$

$M \leq 1.07m + 8\ln n \quad \leftarrow \varepsilon = 1/8$

unlike most worst-case bounds, numbers are pretty good.

Analysis



- Say at time t we have fraction F_t of weight on experts that made mistake.
- So, we have probability F_t of making a mistake, and we remove an εF_t fraction of the total weight.
  - $W_{final} = n(1-\varepsilon F_1)(1 - \varepsilon F_2)\ldots$
  - $\ln(W_{final}) = \ln(n) + \sum_t [\ln(1 - \varepsilon F_t)] \leq \ln(n) - \varepsilon \sum_t F_t$ (using ln(1-x) < -x)

    $= \ln(n) - \varepsilon M.$ ($\sum F_t$ = E[# mistakes])
- If best expert makes m mistakes, then $\ln(W_{final}) > \ln((1-\varepsilon)^m)$.
- Now solve: $\ln(n) - \varepsilon M > m \ln(1-\varepsilon)$.

$$M \leq \frac{-m\ln(1-\varepsilon) + \ln(n)}{\varepsilon} \approx (1+\varepsilon/2)m + \frac{1}{\varepsilon}\log(n)$$

Summarizing

- E[# mistakes] $\leq (1+\varepsilon)$OPT + $\varepsilon^{-1}\log(n)$ = OPT + (εOPT + $\varepsilon^{-1}\log(n)$)
- If set $\varepsilon = (\log(n)/OPT)^{1/2}$ to balance the two terms out (or use guess-and-double), get bound of $M \leq OPT + 2(OPT \cdot \log n)^{1/2} \leq OPT + 2(T \log n)^{1/2}$
- Define average regret in T time steps as:

  (avg per-day cost of alg) – (avg per-day cost of best fixed expert in hindsight).

  Goes to 0 or better as T→ ∞ = "no-regret" algorithm].

Extensions

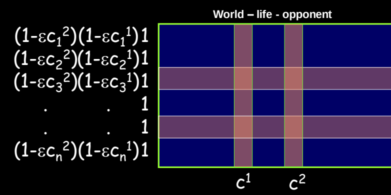
- What if experts are actions? (rows in a matrix game, ways to drive to work,...)
- At each time t, each has a loss (cost) in {0,1}.
- Can still run the algorithm
  - Rather than viewing as "pick a prediction with prob proportional to its weight" ,
  - View as "pick an expert with probability proportional to its weight"
  - Alg pays expected cost $\vec{p_t} \cdot \vec{c_t} = F_t$.
- Same analysis applies.

Do nearly as well as best action in hindsight!

Extensions

- What if losses (costs) in [0,1]?
- Just modify alg update rule: $w_i \leftarrow w_i(1 - \epsilon c_i)$.
- Fraction of wt removed from system is:
  $(\sum_i w_i \epsilon c_i)/(\sum_i w_i) = \epsilon \sum_i p_i c_i = \epsilon[our\ expected\ cost]$
- Analysis very similar to case of {0,1}.

RWM (multiplicative weights alg)

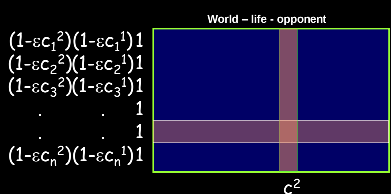


scaling so costs in [0,1]

Guarantee: do nearly as well as fixed row in hindsight

Which implies doing nearly as well (or better) than minimax optimal

Connections to minimax optimality

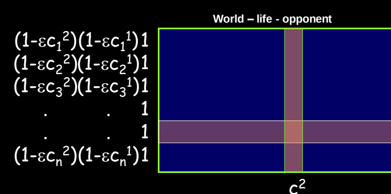


scaling so costs in [0,1]

If play RWM against a best-response oracle, $\vec{p}$ will approach minimax optimality (most $\vec{p}$ will be close).

(If if didn't, wouldn't be getting promised guarantee)

Connections to minimax optimality



scaling so costs in [0,1]

If play two RWM against each other, then empirical distributions must be near-minimax-optimal.

(Else, one or the other could & would take advantage)

A natural generalization

- A natural generalization of our regret goal (thinking of driving) 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)] · (1+ε)$cost_i$(i) + O(ε^{-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 · 11% on them.

"Specialists" or "sleeping experts" problem.

- Assume we have N rules.
- For all i, want E[$cost_i$(alg)] · (1+ε)$cost_i$(i) + O(ε^{-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 / 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 Ã <- $w_i(1+\varepsilon)^{r_i}$
  - 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 · $\varepsilon^{-1}\log N$.
- So, $E[cost_i(alg)]$ · $(1+\varepsilon)cost_i(i) + O(\varepsilon^{-1}\log N)$.

Application: 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).