10-701, 15-781

~~15-859~~(B) Machine Learning ~~Theory~~

Avrim Blum

Lecture ~~2~~: Online learning

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

Batch learning recap

Statistical / PAC model:
- Assume training dataset S consists of iid samples from some prob distrib P.
- Algorithm works on this data, comes up with a classifier.
- Goal is for classifier to do well on new data drawn from P.
- Various bounds (Occam, VC-dimension,…) talk about how big S has to be so that optimizing over S will whp approximately optimize over P.

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…

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 $h(x) = x_1 \vee x_2 \vee \ldots \vee x_n$
  - 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

- 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 – ?
...
```

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 $\leq \sum_h [p_h \cdot \lg(1/p_h)]$ = entropy of distribution p.

Is halving alg optimal?

- Not necessarily ~~(see hwk)~~.
- 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.
  - How can you have a large set of functions with a small mistake bound?

What if there is no perfect function?

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

These are called "regret bounds".
ØShow 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.]

Using "expert" advice

If one expert is perfect, can get ≤ 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.

					prediction	correct
weights	1	1	1	1		
predictions	Y	Y	Y	N	Y	Y
weights	1	1	1	.5		
predictions	Y	N	N	Y	N	Y
weights	1	.5	.5	.5		

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,

$$\begin{aligned}(1/2)^m &\le n(3/4)^M\\ (4/3)^M &\le n2^m\\ M &\le 2.4(m + \lg n)\end{aligned}$$

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 ½ to 1- ε.

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

M = expected #mistakes

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

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

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

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 $\ln(1-x) < -x$)

    $= \ln(n) - \varepsilon M.$ ($\sum F_t = E[\#\text{ mistakes}]$)
- 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

- $E[\#\text{ mistakes}] \leq (1+\varepsilon)OPT + \varepsilon^{-1}\log(n)$.
- If set $\varepsilon=(\log(n)/OPT)^{1/2}$ to balance the two terms out (or use guess-and-double), get bound of $E[\text{mistakes}]\leq OPT+2(OPT\cdot\log n)^{1/2}\leq OPT+2(T\log n)^{1/2}$
- 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 $T\rightarrow\infty$ [= "no-regret" algorithm].

What can we use this for?

- *Can use to combine multiple algorithms to do nearly as well as best in hindsight.*
- *Can apply RWM in situations where experts are making choices that cannot be combined.*
  - Choose expert i with probability $p_i = w_i/\sum_i w_i$.
  - Experts could be different strategies for some task, or rows in a matrix game. (Alg generalizes to case where in each time step, each expert gets a cost in [0,1])

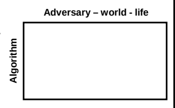
Minimax Theorem (von Neumann 1928)

- Every 2-player zero-sum game has a unique value V.
- Minimax optimal strategy for R guarantees R's expected gain at least V.
- Minimax optimal strategy for C guarantees C's expected loss at most V.

Counterintuitive: Means it doesn't hurt to publish your strategy if both players are optimal. (Borel had proved for symmetric 5x5 but thought was false for larger games)

Repeated play of matrix game

- Let's use a no-regret alg.
- Time-average performance guaranteed to approach best row in hindsight against C's empirical distribution.



- If minimax theorem were false, this would be impossible!
  - ØExistence of no-regret algs yields proof of minimax thm.

A natural generalization

- A natural generalization of this setting: say we have a list of n prediction rules, but not all rules fire on any given example.
- E.g., document classification. Rule: "if <word-X> appears then predict <Y>". E.g., if has football then classify as sports.
- E.g., path-planning: "on snowy days, use this route".
- Natural 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 $E[cost_i(alg)] \leq (1+\varepsilon)cost_i(i) + O(\varepsilon^{-1}\log n)$.
- So, if 80% of documents with football *are* about sports, we should have error $\leq$ 21% on them.

"Specialists" or "sleeping experts" problem.

A natural generalization

Generalized version of randomized WM:

- Initialize all rules to have 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:
    - $R_i = [\sum_j p_j \text{ cost}(j)]/(1+\varepsilon) - \text{cost}(i)$
    - $w_i \leftarrow w_i(1+\varepsilon)^{R_i}$
- 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+ε) factor.
- Can then prove that total sum of weights never goes up.

Why does this work?

- Update weights:
  - If didn't fire, leave weight alone.
  - If did fire, raise or lower depending on performance compared to weighted average:
    - $R_i = [\sum_j p_j \text{ cost}(j)]/(1+\varepsilon) - \text{cost}(i)$
    - $w_i \leftarrow w_i(1+\varepsilon)^{R_i}$
- Can then prove that total sum of weights never goes up.

- One way to look at weights:
  - $w_i = (1+\varepsilon)^{E[\text{cost}_i(\text{alg})]/(1+\varepsilon) - \text{cost}_i(i)}$
  - I.e., we are explicitly giving large weights to rules for which we have large regret.
  - Since sum of weights $\leq n$, exponent must be $\leq \log_{1+\varepsilon} n$
- This implies our bound: $E[\text{cost}_i(\text{alg})] \leq (1+\varepsilon)\text{cost}_i(i) + O(\varepsilon^{-1}\log n)$.