Topics in Machine Learning Theory

Avrim Blum

09/03/14

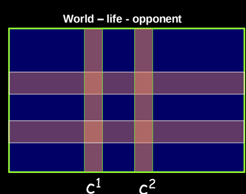
Lecture 3: Shifting/Sleeping Experts, the Winnow Algorithm, and L_1 Margin Bounds

Recap from end of last time

RWM (multiplicative weights alg)

$p_i = w_i/W$

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



scaling so costs in [0,1]

Our expected cost: $\vec{p} \cdot \vec{c}$

At each step, $W \leftarrow W - \sum_i \epsilon c_i w_i = W(1 - \epsilon \vec{c} \cdot \vec{p})$

So, $\ln(W_{final}) \leq \ln(n) - \epsilon \cdot \sum_t (\vec{c}^t \cdot \vec{p}^t)$

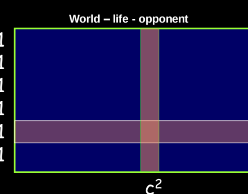
So, $\mathrm{E[our\ cost]} \leq \frac{1}{\epsilon}\ln(n) - \frac{1}{\epsilon}\ln(W_{final})$

Which then implies…

RWM (multiplicative weights alg)

$p_i = w_i/W$

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



scaling so costs in [0,1]

We pay $\vec{p} \cdot \vec{c}$

$E[cost] \leq OPT(1+\epsilon) + \frac{1}{\epsilon}\log n \leq OPT + \log n + O(\sqrt{T \cdot \log n})$

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

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)] $\leq$ $(1+\varepsilon)cost_i(i)$ + 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 $\leq$ 11% on them.

"Specialists" or "sleeping experts" problem.

- Assume we have N rules.
- For all i, want E[$cost_i$(alg)] $\leq$ $(1+\varepsilon)cost_i(i)$ + 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)

(with a *little* handwaving)

- Start with all rules at weight 1.
- At each time step, of the rules i that fire, select one with probability $p_i \propto w_i$.
- When we get our results, 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, $E[cost_i(alg)] \leq (1+\varepsilon)cost_i(i) + O(\varepsilon^{-1}\log N)$.

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).

Next topic: Winnow algorithm

Recap: 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.
  - We saw this is optimal.

Recap: disjunctions

- But what if most features are irrelevant?
- Target is an OR of r out of n.
- In principle, what kind of mistake bound could we hope to get?
- Ans: $\log(n^r) = O(r \log n)$, using halving.

Can we get this *efficiently*?

Yes – using Winnow algorithm.

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 $x_i=1$.

Theorem: Winnow makes at most $1 + 2r(1 + \lg n) = O(r \log n)$ mistakes.

Proof

Thm: Winnow makes $\leq 1 + 2r(1+\lg n)$ mistakes.

- 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.

Proof, step 1: how many mistakes on positive exs?

Ans:

- each such mistake doubles at least one relevant weight.
- Any such weight can be doubled at most $\lceil \lg n \rceil$ times.
- So, at most $r\lceil \lg n \rceil \leq r(1+\lg n)$ such mistakes.

Proof

Thm: Winnow makes $\leq 1 + 2r(1+\lg n)$ mistakes.

- 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.

Proof, step 1: at most $r(1+\lg n)$ mistakes on positives

Proof, step 2: how many mistakes on negatives?

- Total sum of weights is initially n.
- Each mistake on positives adds at most n to the total.
- Each mistake on negatives removes at least n from total.
- So, #(mistakes on negs) $\leq 1 +$ #(mistakes on positives).

Proof

Thm: Winnow makes $\leq 1 + 2r(1+\lg n)$ mistakes.

- 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.

Proof, step 1: at most $r(1+\lg n)$ mistakes on positives

Proof, step 2: at most $1 + r(1+\lg n)$ mistakes on negs

Done.

Open question: efficient alg with mistake bound poly(r, log(n)) for length-r decision lists?

Extensions

Winnow algorithm for learning a k-of-r function: e.g., $x_3 + x_9 + x_{10} + x_{12} \geq 2$.

- 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 ϵ = 1/(2k).

Thm: Winnow makes O(rk log n) mistakes.

Idea: think of alg as adding/removing chips.

Extensions

- Winnow algorithm for learning a k-of-r function: e.g., $x_3 + x_9 + x_{10} + x_{12} \geq 2$.
- 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 ϵ = 1/(2k).

Analysis:

- Each m.o.p. adds at least k relevant chips, and each m.o.n removes at most k-1 relevant chips. At most r(1/ϵ)log n relevant chips total.

Extensions

- 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 ϵ = 1/(2k).

Analysis:

- Each m.op. adds at least k relevant chips, and each m.o.n removes at most k-1 relevant chips. At most r(1/ϵ)log n relevant chips total.
- Each m.o.n. removes almost as much total weight as each m.o.p. adds. At most ϵn added in m.o.p., at least $\epsilon n/(1+\epsilon)$ removed in m.o.n. Can't be negative.

Extensions

- $k \cdot M_{pos} - (k-1) \cdot M_{neg} \le \left(\frac{r}{\epsilon}\right) \log n$.
- $n + M_{pos} \cdot \epsilon n - M_{neg} \cdot \frac{\epsilon n}{1+\epsilon} \ge 0$.
  - I.e., $\frac{1+\epsilon}{\epsilon} + (1+\epsilon) M_{pos} \ge M_{neg}$.
- Plug in to first equation and solve.

Analysis:

- Each m.op. adds at least k relevant chips, and each m.o.n removes at most k-1 relevant chips. At most r(1/ϵ)log n relevant chips total.
- Each m.o.n. removes almost as much total weight as each m.o.p. adds. At most ϵn added in m.o.p., at least $\epsilon n/(1+\epsilon)$ removed in m.o.n. Can't be negative.

Extensions

- $k \cdot M_{pos} - (k-1) \cdot M_{neg} \le \left(\frac{r}{\epsilon}\right) \log n$.
- $n + M_{pos} \cdot \epsilon n - M_{neg} \cdot \frac{\epsilon n}{1+\epsilon} \ge 0$.
  - I.e., $\frac{1+\epsilon}{\epsilon} + (1+\epsilon) M_{pos} \ge M_{neg}$.
- Plug in to first equation and solve.

$k \cdot M_{pos} - (k-1)(1+\epsilon) M_{pos} \le \left(\frac{r}{\epsilon}\right) \log n + (k-1)\left(\frac{1+\epsilon}{\epsilon}\right)$.

We set $\epsilon = \frac{1}{2k}$ so $(k-1)(1+\epsilon) \le k - \frac{1}{2}$.

Get: $\frac{1}{2} M_{pos} \le \left(\frac{r}{\epsilon}\right) \log n + (k-1)\left(\frac{1+\epsilon}{\epsilon}\right) = O(rk \log n)$.

So, M_{pos}, M_{neg} are both $O(rk \log n)$.

If don't know k,r, can guess-&-double: get $O(r^2 \log n)$.

How about learning general LTFs?

E.g., $4x_3 - 2x_9 + 5x_{10} + x_{12} \ge 3$.

Will look at two algorithms (one today, one next time) each with different types of guarantees:

- Winnow (same as before)
- Perceptron

Winnow for general LTFs

E.g., $4x_3 - 2x_9 + 5x_{10} + x_{12} \ge 3$.

- First, add variable $x'_i = 1 - x_i$ so can assume all weights positive.

E.g., $4x_3 + 2x'_9 + 5x_{10} + x_{12} \ge 5$.

- Also conceptually scale so that all weights w_i^* of target are integers (not needed but easier to think about)

Winnow for general LTFs

- Idea: suppose we made W copies of each variable, where $W = w_1^* + \ldots + w_n^*$.
- Then this is just a "w_0^* out of W" function!

E.g., $4x_3 + 2x'_9 + 5x_{10} + x_{12} \ge 5$.

- So, Winnow makes $O(W^2 \log(Wn))$ mistakes.
- And here is a cool thing: this is equivalent to just initializing each w_i to W and using threshold of nW. But that is same as original Winnow!

Winnow for general LTFs

More generally, can show the following (it's an easy extension):

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

- $w^* \cdot x \ge c$ on positive x,
- $w^* \cdot x \le c - \gamma$ on negative x.

Then mistake bound is

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

Multiply by $L_\infty(X)$ if examples not in $\{0,1\}^n$

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