Machine Learning:
a tour through some favorite results, directions, and open problems

Your guide: Avrim Blum
Carnegie Mellon University

(no tipping)

[FOCS 2003 tutorial]

Philosophy of the tour

A set of topics with:

- nice/clean theory.
- relation to other TOC issues / tools have potential use in other TOC areas.
- good open problems.
- useful in practice
- I know something about it.

Itinerary

Preparation: Basic concept learning setting
(learning from examples)

Stop 1:Batch learning:Algs&Sample complexity
(why am I trying to optimize on data I've seen: why should this carry over to the future?)

Stop 2:Online learning (expert advice and other problems)

Stop 3:SQ and Fourier (strong complexity results)

Stop 4:Current "hot" practical issues.

Emphasize connections to other TOC areas/issues (incl apx algs, online algs, auctions, complexity, crypto)

The concept learning setting

- Imagine learning algorithm trying to decide which loan applicants are bad credit risks.
- Might represent each person by n features. (e.g., income range, debt load, employment history, etc.)
- Take sample S of data, labeled according to whether they were/weren't good risks.
- Goal of algorithm is to use data seen so far produce good prediction rule (a "hypothesis") $h(x)$ for future data.

The concept learning setting

E.g.,

% down	recent delinq?	other accts	mmp/ inc	high debt?	Good risk?
10	N	Y	0.32	N	Y
10	N	N	0.25	Y	Y
5	Y	N	0.30	N	N
20	N	Y	0.31	N	Y
5	N	N	0.42	N	N
10	Y	N	0.38	Y	N
10	N	N	0.25	Y	Y

Given this data, some reasonable rules might be:

- Predict YES iff (!recent delinq) AND (%down > 5).
- Predict YES iff 100*[mmp/inc] – 1*[%down] < 25.
- ...

Big questions

(A) How might we automatically generate rules that do well on observed data?

(B) What kind of confidence do we have that they will do well in the future?

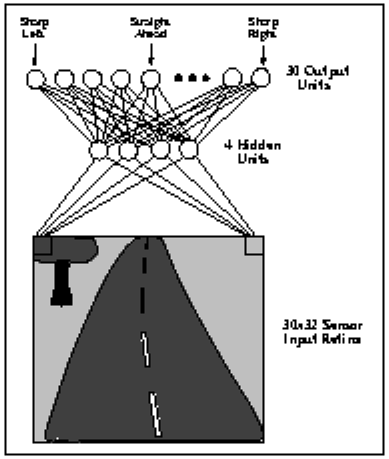
Or, in reverse order,

- What to optimize? [sample complexity]
- How to optimize it? [algorithms]

Power of basic paradigm

Lots of problems solved by converting to basic "concept learning from structured data" setting.

- E.g., document classification
  - convert to bag-of-words
  - LTFs do well
- E.g., driving a car
  - convert image into features.
  - Use neural net with several outputs.



Stop 1: PAC model: Algs & sample complexity

Natural formalization (PAC)

- Alg is given sample $S = \{(x,y)\}$ presumed to be drawn from some distribution D over instance space X, labeled by some target function f.
- Alg does optimization over S to produce some hypothesis h.
- Goal is for h to be close to f over D.
- Allow failure with small prob δ (to allow for chance that S is not representative).

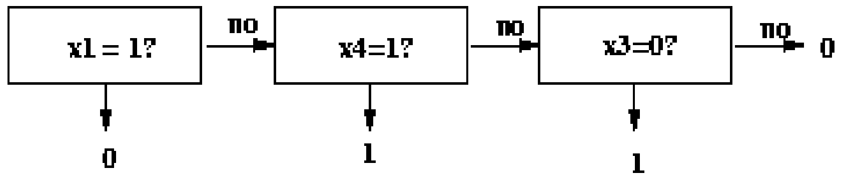
Basic PAC learning defn

- A concept class is a set of functions, together with a representation of them.
- Alg A PAC-learns concept class C by hypothesis class H if for any target f in C, any distrib D over X, any $\varepsilon, \delta > 0$,
  - A uses at most poly(n,$1/\varepsilon$,$1/\delta$,size(f)) examples and running time.
  - With probability $1-\delta$, A produces h in H of error at most ε.

accuracy

confidence

Example of guarantee: Decision Lists



Given a dataset S of m examples over n boolean features, drawn according to unknown distrib D, labeled by unknown target f:

1. Algorithm **A** will find a consistent DL if one exists, in time O(mn).
2. If $m > (1/\varepsilon)[n(2+\ln n) + \ln(1/\delta)]$, then

   $\Pr[\text{exists consistent DL } h \text{ with } err(h) > \varepsilon] < \delta$.

How can we find a consistent DL?

x_1	x_2	x_3	x_4	x_5	label
1	0	0	1	1	+
~~0~~	~~1~~	~~1~~	~~0~~	~~0~~	~~−~~
~~1~~	~~1~~	~~1~~	~~0~~	~~0~~	~~+~~
~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~−~~
~~1~~	~~1~~	~~0~~	~~1~~	~~1~~	~~+~~
1	0	0	0	1	−

if (x_1=0) then -, else
if (x_2=1) then +, else
if (x_4=1) then +, else -

Decision List algorithm

- Start with empty list.
- Find if-then rule consistent with data.

  (and satisfied by at least one example)
- Put rule at bottom of list so far, and cross off examples covered. Repeat until no examples remain.

If algorithm fails, then:

- No DL consistent with remaining data.
- So, no DL consistent with original data.

OK, fine. Now why should we expect it to do well on future data?

Confidence/sample-complexity

- Consider some hypothesis h with $err(h) > \varepsilon$.
- Chance that h survives m examples is at most $(1-\varepsilon)^m$.
- Number of DLs over n Boolean features is at most $n!4^n$. (for each feature there are 4 possible rules, and no feature will appear more than once)

) Pr[some DL h with $err(h)>\varepsilon$ is consistent]
$< n!4^n(1-\varepsilon)^m$.

- This is $< \delta$ for $m > (1/\varepsilon)[n(2+\ln n) + \ln(1/\delta)]$

DL: summary

Suppose the target f is, in fact, a decision list.

Then with probability $\geq 1 - \delta$, the hypothesis h produced by the algorithm has error $< \varepsilon$, so long as the number of examples m seen satisfies

$$m \geq \frac{1}{\varepsilon}\left[n(2 + \ln n) + \ln \frac{1}{\delta}\right].$$

I.e., it's Probably Approximately Correct

Confidence / sample complexity

Nothing special about DLs in our argument.

- All we said was: "if not too many rules to choose from, then unlikely some bad one will fool you just by chance."
- Generalize to any hypothesis space H.
- After m examples, with probability $\geq 1-\delta$, all $h \in H$ with $err(h) \geq \varepsilon$ have $\widehat{err}(h) > 0$, for

$$m \geq \frac{1}{\varepsilon}\left[\log(|H|) + \log\left(\frac{1}{\delta}\right)\right].$$

Occam's razor

William of Occam ($\sim$ 1320 AD):

> "Entities should not be multiplied unnecessarily" (in Latin)

Which we interpret as: "in general, prefer simpler explanations".

Why? Is this a good policy? What if we have different notions of what's simpler?

Occam's razor (contd)

A computer-science-ish way of looking at it:

- Say "simple" = "short description".
- At most 2^s explanations that are $< s$ bits long.
- If number of examples seen satisfies

$$m \geq \frac{1}{\varepsilon}\left[s \ln 2 + \ln\left(\frac{1}{\delta}\right)\right].$$

then it's unlikely a bad simple explanation will fool you just by chance.

Occam's razor (contd)^2

Nice interpretation:

- Even if we have different notions of what's simpler (e.g., different representation languages), we can both use Occam's razor.

- Of course, there's no guarantee there will be a short explanation for the data. That depends on your representation.

Where are we now?

- Introduced PAC model. Showed class of decision lists learnable by decision list hypotheses.
- Saw Occam's razor argument.
- Implications:
  - If we view cost of examples as comparable to cost of computation, then don't have to worry about cost of data since just need ~ $1/\varepsilon$ examples per bit we output.
  - But, in practice, costs often wildly different, so sample-complexity issues are crucial.

What's next

- Stop 1(a): a few more words about sample complexity.
- Stop 1(b): a few more words about algorithmic issues.
- Stop 1(c): some of my favorite open problems.

More on sample complexity

- Bounds so far are for "realizable case". (when we expect there will be a perfect rule). They say whp if true error > ε then empirical error > 0.
- More generally, might want to say whp all rules have empirical error near to true error.

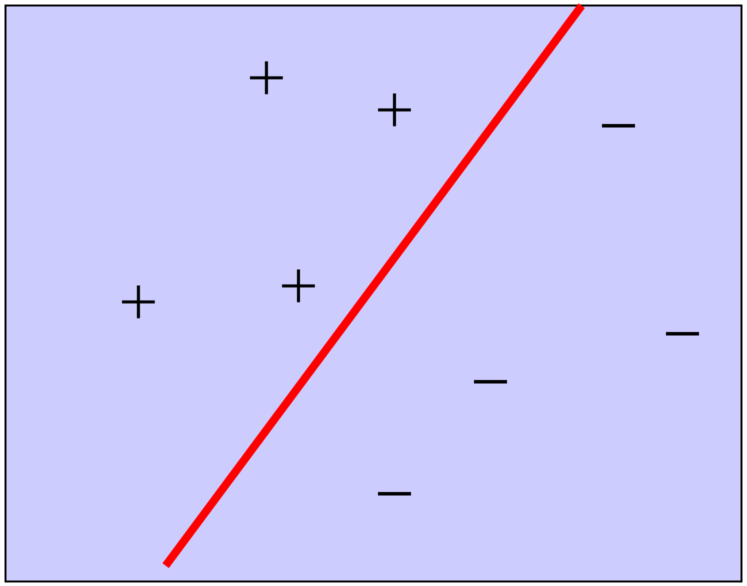
After m examples, with probability $\geq 1-\delta$, all $h \in H$ have $|err(h) - \widehat{err}(h)| < \varepsilon$, for

$$m \geq \frac{1}{2\varepsilon^2}\left[\log(|H|) + \log\left(\frac{1}{2\delta}\right)\right].$$

- "uniform convergence"
- Gives hope for local optimization over training data.

Tighter measures of size(H)

- Suppose we want to separate data using a line in the plane.



- There are infinitely-many linear separators, but not that many really different ones.
- In hindsight, only $O(m^2)$ ways of separating our given m points.

Neat fact

- Can replace |H| with # ways of splitting 2m points in the bounds.
- # ways of splitting = $O(m^{\text{VC-dim}(H)})$. (VC-dimension is the largest number d of points that can be split in all 2^d ways.)
- We then take the log, so this shows the number of examples needed is only linear in VC-dim(H).

Proof sketch

Step 1: imagine we draw m training examples and m test examples.

Then, Pr[exists h that looks good on train, bad on test]

$> \frac{1}{2}$Pr[some bad h looks good on train]

(so can show 2nd is small by proving 1st is small)

Step 2: make even worse on ourselves by allowing adversary to pick 2m points. Randomize over partition into m train and m test.

Step 3: But now we're back in finite case. Can argue like Occam, over the splits of the sample.

Can view result as allowing description language to depend on unlabeled double sample.

Other related nice bounds

- Margin bounds.
  - Nice connection to Johnson-Lindenstrauss too.
- PAC-Bayes bounds.
  - Can view Occam as saying for any prob dist P over H, we can be confident in h_i if it is consistent with $\varepsilon^{-1}[\log(1/p_i) + \log(1/\delta)]$ samples. (basically allowing h_i to fool us with prob δp_i)
  - What if no individual h_i that's consistent has high enough p_i, but there are several that together have a high sum of p_i's? Then can randomize over them to get bounds as if was one hyp with combined probs.
- Subsample ("compression") bounds

What's next

- Stop 1(a): a few more words about sample complexity.
- Stop 1(b): a few more words about algorithmic issues.
- Stop 1(c): some of my favorite open problems.

Algorithmic/model issues

PAC model talks of learning C by H.

- If H=C then learning is NP-hard if consistency problem is NP-hard.
  - May not be clear if difficulty is due to C or H.
- To look at "how inherently hard is C to predict?", let H = poly time programs. Then hardness is more cryptographic.
- To use hard learning problem for crypto, need to be hard for "random" fns in C.

Algorithmic/model issues

PAC model talks of learning C by H.

- In practice, most natural to fix H, allow C to be arbitrary. (After all, H is under our control, but target function isn't).
- Try to find reasonable h in H if one exists. Usually hard to say anything positive theoretically.

Algorithmic/model issues

PAC model talks of ε and δ. (high prob, high acc)

- Get same set of learnable concepts if fix $\varepsilon = 1/4$, $\delta = \frac{1}{2}$. (reasonable prob, reasonable acc)
- Or, $\varepsilon = \frac{1}{2} - 1/\text{poly}$, $\delta = 1 - 1/\text{poly}$. (With 1/poly chance we do slightly better than random guessing). "weak learning".
- Result for δ is easy to see. Result for ε is Boosting. [Schapire, Freund & Schapire]
  - Boosting results are really nice (both theoretically and practically). Involves re-weighting sample and rerunning algorithm. But not going to talk about....

Algorithmic/model issues

What about adding extra power to alg?

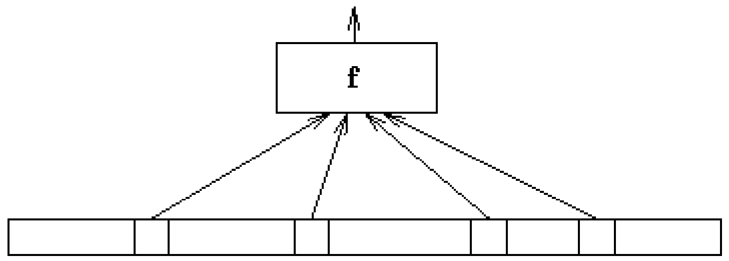
- "Membership query" model: learner can ask for $f(x)$ for examples x of its construction.
  - adds significant power.
  - allows for more interesting algorithms.
  - but rare in practice. Esp because "target function" is really a fiction.
- What is often possible in practice is "active learning". Have large unlabeled sample and alg may choose among these.

Open Problems

1. Learning functions of r relevant variables.
2. Learning “almost-OR” functions.
3. Learning DLs over string-valued features.
4. Learning Monotone DNF over uniform.

Learning functions of r variables

- Examples in $\{0,1\}^n$. Uniform distribution.
- But only r of these n bits are relevant.
- Target is arbitrary function of these r.

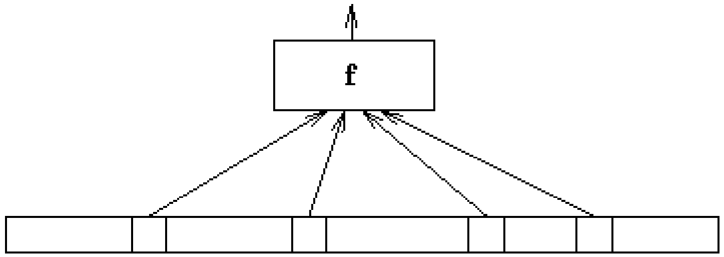


Q1: learn in poly time for r = O(log n)?

Note: this is special case of "Are DNF learnable over uniform dist?" and "Are decision trees learnable over uniform dist?"

Learning functions of r variables

- Examples in $\{0,1\}^n$. Uniform distribution.
- But only r of these n bits are relevant.
- Target is arbitrary function of these r.

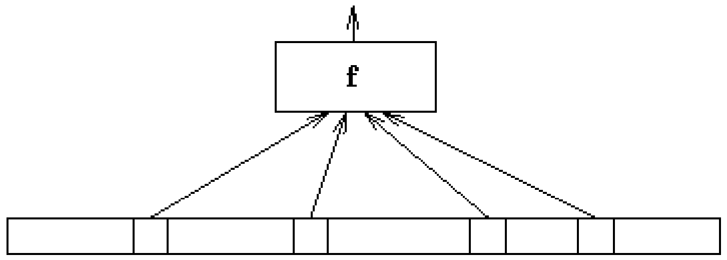


Q1: learn in poly time for r = O(log n)?

Note 2: this is easy if we allow Membership Queries.

Learning functions of r variables

- Examples in $\{0,1\}^n$. Uniform distribution.
- But only r of these n bits are relevant.
- Target is arbitrary function of these r.

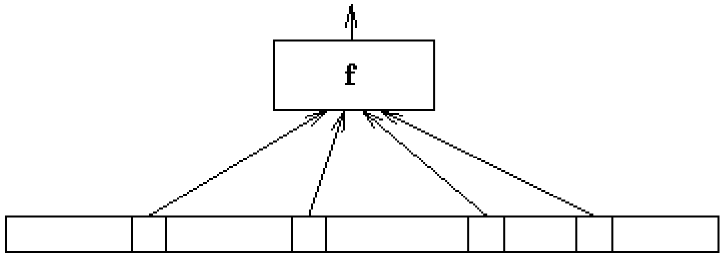


Q2: How about r = loglog(n)?

Now can assume we know truth-table, since only n possibilities.

Learning functions of r variables

- Examples in $\{0,1\}^n$. Uniform distribution.
- But only r of these n bits are relevant.
- Target is arbitrary function of these r.

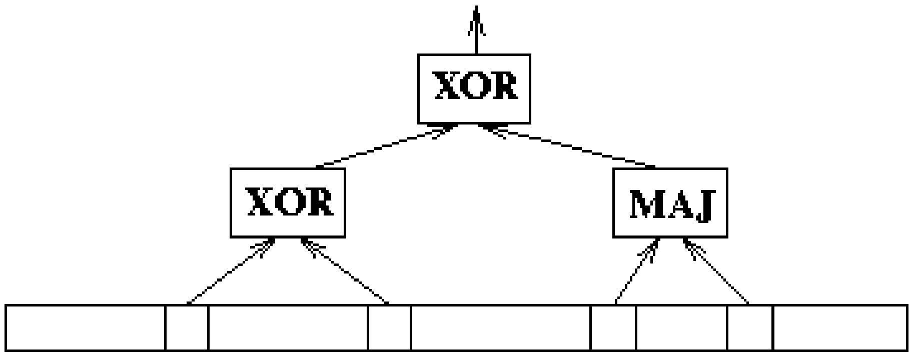


Q2: How about $n^{o(r)}$ or even $n^{r/2}$?

Best known is recent [MOS] result of ~ $n^{0.7r}$.

Example of hard-looking function

- Pick two sets A,B.
- Target is maj(A) XOR parity(B).



Open Problems

1. Learning functions of r relevant variables.
2. Learning "almost-OR" functions.
3. Learning DLs over string-valued features.
4. Learning Monotone DNF over uniform.

Learning "almost-OR" functions

Suppose target f has the property that it is close to an OR function over D.

- for some OR f', $\Pr_x[f(x) \neq f'(x)] < 1\%$.

Can we get error < 49%? $\frac{1}{2}-1/n^k$? (in poly time)

If we replace "OR" with "XOR", and require H=C, then [Hastad] shows $\frac{1}{2} - \varepsilon$ is NP-hard.

[Note: new hardness result of Dinur, Guruswami, and Khot on Set-Cover implies our problem is NP-hard if we require hyp to be an OR-function *and* require 1-sided error (we must get all positives correct, and at least 1% of negatives correct)]

Possibly easier version

Suppose really two functions, f and g:

- f is an OR function.
- g is arbitrary.
- Example drawn from D. With probability 99%, gets labeled by f, with probability 1%, gets labeled by g.

Potentially easier since adversary has less control. Still open though.

However, if require g = 1-f, then this is random noise model, and problem is easy. (even if you raise noise rate to 49%).

Open Problems

1. Learning functions of r relevant variables.
2. Learning "almost-OR" functions.
3. Learning DLs over string-valued features.
4. Learning Monotone DNF over uniform.

Learning Decision Lists revisited

- What if features were string-valued rather than boolean valued? [also known as "infinite attribute model"]
- E.g., x = (hello, how, are, you).
- Target is a decision list, like:

  if (x_1 = "hello") then +, else if (x_2 = "joe") then -, else if (x_1 = "aargh") then +, else -.
- Can we learn in time poly(n, size(f))?

  (Previous alg may produce hyp that grows linearly with size of data set.)

Open Problems

1. Learning functions of r relevant variables.
2. Learning "almost-OR" functions.
3. Learning DLs over string-valued features.
4. Learning Monotone DNF over uniform.

Learning Monotone DNF over uniform

- Uniform distribution on $\{0,1\}^n$.
- Target is a monotone DNF formula.
- Can you learn with error rate < 25%?

- It's known how to achieve error rate $\frac{1}{2}$ - $O(n^{-1/2})$ [weak learning] [BBL]
- It's known how to strong-learn if the number of terms is small: $2^{p(\log n)}$ [S]
- But it's not known how to strong-learn in general.

Stop 2: online learning

Basic setting

- View learning as a sequence of trials.
- In each trial, algorithm is given x, asked to predict f, and then is told the correct value.
- Make no assumptions about how examples are chosen.
- Goal is to minimize number of mistakes.

Note: can no longer talk about # examples needed to converge. Instead, we focus on number of mistakes. Need to "learn from our mistakes".

Simple example: learning an OR fn

- Suppose features are boolean: $X = \{0,1\}^n$.
- Target is an OR function, like $x_3 \vee x_9 \vee x_{12}$, with no noise.
- Can we find an on-line strategy that makes at most n mistakes?
- Sure.
  - Start with $h(x) = x_1 \vee x_2 \vee \cdots \vee x_n$
  - Invariant: {vars in h} contains {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 postives. So at most n mistakes total.

More general class: LTFs

- Target is a vector $(a_1, \ldots a_n)$ and threshold t.
- Example $x \in \{0,1\}^n$ is positive if $a_1x_1 + \ldots + a_nx_n > t$, else is negative.
- An OR function is the case $a_i \in \{0,1\}$, $t=1$.

Q: can you guarantee at most poly(n,size(f)) mistakes?

Yes. Use the ellipsoid algorithm.

- Examples seen so far form a set of linear constraints. (reversing usual use of "a" and "x" in LP)
- Center of ellipsoid is hypothesis.
- Mistake can be viewed as output of separation oracle.

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
…	…	…	…	…

Rough question: What's a good strategy for using their opinions, given that in advance we don't know which is best?

["expert" ´ someone with an opinion. Not necessarily someone who knows anything.]

Simpler question

- We have n "experts".
- One of these is perfect (never makes a mistake). We just don't know which one.
- Can we find a strategy that makes no more than $\lg(n)$ mistakes?

Answer: sure. Just take majority vote over all experts that have been correct so far. Called "halving algorithm".

Followup question: what if we have a "prior" p over the experts. Can we make no more than $\lg(1/p_i)$ mistakes, where expert i is the perfect one?

Sure, just take weighted vote according to p.

Relation to concept learning

- If computation time is no object, can have one "expert" per concept in C.
- If target in C, then number of mistakes at most $\lg(|C|)$.
- More generally, for any description language, number of mistakes is at most number of bits to write down f.

Back to expert-advice

What if no expert is perfect? Goal is to 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 log(n)*OPT mistakes, where OPT is # mistakes of the best expert in hindsight.

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

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.

Weighted Majority Algorithm

Weighted Majority Alg:

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

Example:

					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		
predictions	Y	N	N	N	N	N
weights	.5	.5	.5	.5		
predictions	N	Y	N	Y	either	N
weights	.5	.25	.5	.25		

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

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

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

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

unlike most C.R.s or apx 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.
  - $W_{final} = n(1-\varepsilon F_1)(1 - \varepsilon F_2)\ldots$
  - $\ln(W_{final}) = \ln(n) + \sum_t [\ln(1 - \varepsilon F_t)] \cdot \ln(n) - \varepsilon \sum_t F_t$

    (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: $\ln(n) - \varepsilon M > m \ln(1-\varepsilon)$.

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

A fun application

- n buckets. (Think of as startup companies.)
- You are standing in one of them.
- At each time step, a ball falls into one of the buckets. If it's *your* bucket, you get \$1.
- Then you can choose to move if you want
- Game ends when fullest bucket has d balls.

This is hopeless if an opponent is tossing the balls based on knowing where you are(n't). However, say sequence of balls is predetermined (but unknown).

Randomized WM will guarantee you an expected gain of at least $d - (2d \log n)^{1/2}$.

[Multiply weight by $1+\varepsilon$ whenever ball falls in, $\varepsilon = (d/\log n)^{1/2}$.]

Summarizing

- Can be $(1+\varepsilon)$-competitive with best expert in hindsight, with additive $\varepsilon^{-1}\log(n)$.
- If have prior, can replace additive term with $\varepsilon^{-1}\log(1/p_i)$. [$\varepsilon^{-1}$ x number of bits]
- Often written in terms of additive loss. If running T time steps, set epsilon to get additive loss $(2T \log n)^{1/2}$

What can we use this for?

- Can use to combine multiple algorithms to do nearly as well as best in hindsight.
  - E.g., online auctions: one expert per price level.
- Play repeated game to do nearly as well as best strategy in hindsight (which is at least as good as minimax optimal).
- Extensions: "bandit problem", movement costs.

What about if "n" is large?

- Bounds still good even if #experts is very large.
  - adaptive search trees: one expert per tree.
  - online path planning: one expert per path.
  - online linear programming: one expert per corner.
- Nice recent results [KV][Z] on ways to get these bounds efficiently for these types of problems.

Stop 3: SQ learning and Fourier analysis

What are the goals of complexity theory?

- We'd like to prove w/o assumptions that no polynomial-time algorithm can solve some natural problem (like one in NP).
- Barring that, how about an algorithm with one arm tied behind its back?
- How about both arms behind back, while hopping on one foot? (like AC^0 ckts)

How often do you find yourself saying: "drat, there's no way my alg is going to succeed because it's an AC^0 ckt"?

What are the goals of complexity theory?

- We'd like to prove w/o assumptions that no polynomial-time algorithm can solve some natural problem (like one in NP).
- Barring that, how about an algorithm with one arm tied behind its back?
- How about both arms behind back, while hopping on one foot? (like AC^0 ckts)

(Of course, there's been phenomenal success in showing how mild-looking assumptions can have enormous consequences)

SQ and Fourier analysis

- We will define a paradigm for using data that nearly all learning algorithms satisfy or can be cast into. ("Statistical Queries")
- Use Fourier analysis to prove without assumptions they cannot learn certain natural classes in poly time.
  - E.g., decision trees, parity functions, log(n)-relevant-var function.

The Statistical Query model

- PAC model, but algorithm no longer has access to individual labeled examples.
- Instead, algorithm may ask "what is the probability a labeled example would have property P?" Gets answer back up to error τ.
- Think of this as asking for *statistics* about a poly-size sample S.
- Formally, P must be poly-time computable, and $\tau > 1/\mathrm{poly}(\ldots)$.
- Also assume alg knows underlying dist D.

Most algorithms can be cast in this framework

E.g., list-and-cross-off OR-function alg:

- Ask for $\Pr[f(x)=0 \;\&\; x_i = 1]$ with $\tau = \varepsilon/2n$.
  (i.e., what's the chance we'd cross it off)
- Cross off all x_i with $\Pr[..] > \varepsilon/2n$.
- What's left will include all correct variables.
- Might include some incorrect ones too, but they will introduce at most ε/n error each.

Most algorithms can be cast in this framework

E.g., typical local-optimization algorithm:

- Given current hypothesis h.
- Asks for err(h) = Pr[h(x) != f(x)].
- Then asks: "what if I make small change z or tweak parameter w, does it get better?"

Most algorithms can be cast in this framework

Original motivation of model: algorithms that behave this way can automatically be made tolerant to random classification noise. [Kearns]

In fact, only recently were examples shown of (non-SQ) alg to learn something with noise that is not learnable by SQs.

What I like: can get very good handle on what can/cannot be learned in this model.

Fourier analysis of finite fns

- Let's write pos, neg as +1, -1.
- Think of a fn $f:\{0,1\}^n!\ \{-1,+1\}$ as vector:

$$(\sqrt{\Pr(000)}f(000), \sqrt{\Pr(001)}f(001), \ldots)$$

Nice properties:

- $\langle f,f\rangle = \sum_x \Pr_D(x)f(x)f(x) = 1.$
- $\langle f,g\rangle = \sum_x \Pr_D(x)f(x)g(x)$
  $= \Pr_D[f(x) = g(x)] - \Pr_D[f(x) \mathrel{!=} g(x)].$

"orthogonal" = "pairwise uncorrelated". E.g., under uniform dist, all 2^n parity functions are orthogonal (so they form a basis).

SQ dimension

SQ dimension of a concept class C, over distr D, is the size of the largest subset S of C of "nearly orthogonal" functions:

- For all f,g in S, $|\langle f,g\rangle| < 1/|S|$.

- If SQ-dim(C) = poly(...) then you can weak-learn over D in SQ model. [non-uniform alg]
- If SQ-dim(C) > poly(...) then you can't weak-learn over D in SQ model.

Positive direction

If SQ-dim(C) = poly(...) then you can weak-learn over D in SQ model.

- SQ dimension of a concept class C, over distr D, is the size of the largest subset S of C such that for all f,g in S, $|\langle f,g\rangle| < 1/|S|$.

- So, just hard-code $S = \{h_1,\ldots,h_{|S|}\}$ into the learning alg.
- Ask for correlation of h_i with target for all h_i in S
- One of them must have at least $1/|S|$ correlation

Negative direction

If SQ-dim(C) > poly(...) then you can't weak-learn over D in SQ model.

Slightly over-simplified proof sketch:

- Show that any statistical query can be converted to asking for correlation of target with some unit-length vector q, up to +/- 1/poly.
- So, suppose C has > poly(...) orthogonal functions.
- query q can have correlation ε with at most $1/\varepsilon^2$ of them.
- If only ask poly many queries, can only knock out negligible fraction. If target is random from S, then whp all queries can be answered with 0.

Implications

- Parity functions have SQ-dim 2^n, so can't learn by SQ.
- Decision trees contain {Parity functions of size log(n)}. $n^{\log n}$ of them, so can't learn by SQ.
- Same for "functions of log(n) relevant variables."

Stop 4: Current “hot” practical issues

Current "hot" practical issues

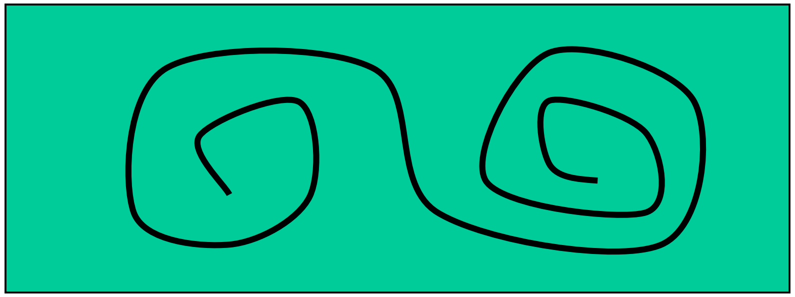
- Learning from labeled and unlabeled data.
- Nonlinear embeddings.
- MDPs and stochastic games.
- Adaptive programs and environments.
- Kernels, comp bio, many others, ... [not going to talk about]

Learning from labeled and unlabeled data

- Often unlabeled data is cheap, labeled data is expensive. Can unlabeled data help?
- Rough answer: unlabeled data can suggest which hyps are a-priori more reasonable.
  - E.g., linear separators that don't go through clusters.
  - E.g., data with pair-wise relationships (vertices in a graph): hyps that are good cuts or good ratio-cuts.
  - ➢ Using unlabeled data to order your hypotheses.
- Algorithmically, can use to bootstrap.
  - E.g., co-training if two different sources of info. (e.g., words on page; words pointing to page)

Nonlinear embeddings

- Often the "real" dimensionality of data is a lot smaller than the space we're using.
- But might not be just a linear embedding.



- Want a nonlinear projection that unrolls the manifold.
- E.g., one approach: take a lot of data, draw nearest-neighbor graph, use to project.

MDPs and stochastic games.

- Robot learning to act in its environment.
  - Think of a directed graph where edges have rewards and a probability distribution over endpoints. (MDP)
- Robot learning to act in environment with other agents who have their own agendas.
  - Stochastic game.

Adaptive programs and environments

- Can machine learning be a first-class part of a programming language?
  - E.g., easy to write code with parameters that get optimized to the user's needs.
- Can systems/environments be built that adapt, suggest, help, etc.
- Mostly crude (paperclip) or just talk right now...

References

[need to fill in…]

For more information, there is a web site for the area as a whole at www.learningtheory.org, with pointers to survey articles, course notes, tutorials, and textbooks.