On-line Learning and Prediction

Avrim Blum

Carnegie Mellon University

[Twelfth International School For Computer Science Researchers, Lipari, 2000]

High level view

Machine Learning: goal of improving performance at a task based on experience. Typical tasks:

- Steering a car.
- Medical diagnosis.
- Playing a game (backgammon, checkers).
- Predicting some quantity (the weather, the market).
- Adapting to a user's needs (self-customizing programs).

Batch learning (given batch of data, want to extract rules) vs. online learning (data comes in one at a time).

I will be focusing on online learning.

High level view (contd)

Online learning and prediction:

- Get information one piece at a time.
- Adapt behavior as we go.
- Need to make decisions based on the past, without knowing future.
- Competitive-style minimax criteria useful for analyzing algorithms.

Plan for today:

- Introduce “Expert Advice” problem.
- Some interesting algorithms
- Some applications.

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. ($n = 4$):

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 combining their opinions, given that in advance we don't know which is best?

["expert" $\equiv$ 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.
- Can we find a strategy that makes no more than $\log_2 n$ mistakes?

Answer: Yes. Just take majority vote of all experts that have been correct so far. Each mistake allows us to cross off at least half.

Expt 1	Expt 2	Expt 3	your dog	truth
down	up	up	up	up
down	up	up	down	down
...	...	...	...	...

This is sometimes called the "halving algorithm".

In some sense optimal: even if experts all predict randomly, you would still expect one to last $\lg n$ steps. (So, $\frac{1}{2}\lg n$.)

A fun financial scam

- Set up 1024 mutual funds (or electronic financial newsletters).
- Each month, "invest" everything on spin of roulette wheel: half of funds on red, half on black.
- Each month, half will go bankrupt and half will double their money.
- After 10 months,
  - 1023 of funds have gone bankrupt. Ignore them.
  - On the other hand, one has gone up by 100,000%.
- Use to get on talk shows explaining what a genius you are.

Back to "expert" advice

What if no expert is perfect? Our goal is just 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 $\log(n) \cdot OPT$ mistakes, where OPT is # mistakes of the best expert in hindsight.

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

The Weighted Majority Algorithm

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

Simple algorithm: **Weighted Majority**

- 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, $W \leq n(3/4)^M$.

- Weight of best expert is $(1/2)^m$. So,

$$(1/2)^m \leq n(3/4)^M$$

$$(4/3)^M \leq n2^m$$

$$M \leq 2.4(m + \lg n)$$

Randomized Weighted Majority

Previous bound 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 1/2 to $1-\varepsilon$.

Solves to: $M \leq \dfrac{-m \ln(1-\varepsilon) + \ln(n)}{\varepsilon}$

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

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

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

As $\varepsilon \to 0$, this goes to $(1+\varepsilon/2)m + \frac{1}{\varepsilon}\ln n$.

Here, M is our expected number of mistakes.

A simpler warmup

Before analyzing, let's go back to simpler problem where we assumed a perfect expert.

- Instead of taking majority, use votes as probabilities.
- Claim: *expected* number of mistakes is at most $\ln n$. (this is better than $\log_2 n$)

Analysis #1: (will analyze in 3 ways)

- Say at time t, the fraction of "alive" experts that make a mistake is F_t.
- So, our expected # mistakes $M = \sum F_t$.
- Also, we know that $\prod(1 - F_t) \geq 1/n$.
- For a fixed product, we minimize the sum when the numbers are all equal [e.g., (12,1) vs (6,2) vs (4,3) vs (3.46,3.46)].
- Solving $(1 - F)^T = 1/n$ gives $F \leq (\ln n)/T$.

$\Rightarrow$ So, $M \leq \ln n$.

A simpler warmup (# 2)

- Instead of taking majority, use votes as probabilities.
- Claim: *expected* number of mistakes is at most $\ln n$. (this is better than $\log_2 n$)

Analysis #2:

- Say at time t, the fraction of "alive" experts that make a mistake is F_t.
- So, our expected # mistakes $M = \sum F_t$.
- Also, we know that $\prod(1 - F_t) \geq 1/n$.
- Take logs: $\sum \ln(1 - F_t) \geq -\ln n$.
- Use the inequality $\ln(1 - x) < -x$:

$\Rightarrow \quad -\sum F_t \geq -\ln n \;\Rightarrow\; M \leq \ln n.$

A simpler warmup (# 3)

- Instead of taking majority, use votes as probabilities.
- Claim: *expected* number of mistakes is at most $\ln n$. (this is better than $\log_2 n$)

Analysis #3:

- Alg can be viewed as "pick expert at random and follow it until it makes a mistake. Then pick a new (alive) expert at random, and continue."

  (Same as Marking Algorithm [BLS])
- Let's number experts in the order they make their first mistake (break ties arbitrarily).
- The chance we were following expert 1 when it made its first mistake is $1/n$. For expert 2 it's at most $1/(n-1)$, etc.
- So, our total expected number of mistakes is at most $\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{2} + 1 \approx \ln n$.

Randomized Weighted Majority

Analyze like in #2.

- Say at time t we have fraction F_t of weight on experts that made mistake.
- So, our expected # mistakes $M = \sum_t F_t$.

$$W_{final} = n(1 - \varepsilon F_1)(1 - \varepsilon F_2)\ldots.$$

$$\begin{aligned}
\ln(W_{final}) &= \ln(n) + \sum_t [\ln(1 - \varepsilon F_t)] \\
&\leq \ln(n) - \varepsilon \sum_t F_t \\
&\quad \text{(using } \ln(1-x) < -x\text{)} \\
&= \ln(n) - \varepsilon M
\end{aligned}$$

- If best expert makes m mistakes, then

$$\ln(W_{final}) \geq \ln((1 - \varepsilon)^m).$$

- Now solve....

$$\begin{aligned}
M &\geq \frac{-m \ln(1 - \varepsilon) + \ln(n)}{\varepsilon} \\
&\approx (1 + \varepsilon/2)m + \frac{1}{\varepsilon}\log(n)
\end{aligned}$$

Typical Application

- If you have several strategies and don't know which to use,
- and if you have the computational resources to run them all,

$\Rightarrow$ then can use WM to do nearly as well as the best of them in hindsight.

Note 1: Get same bounds even if losses in $[0, 1]$ rather than just $\{0, 1\}$.

Note 2: RWM is in a sense more general than WM because can apply to settings where experts can't be combined. (e.g., viewing as in analysis # 3).

Here is an example....

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.

In a sense this is hopeless if an opponent is tossing the balls based on knowing where you are(n't). E.g., what if you choose to always stand in bucket with most money so far?

But, suppose sequence of balls is predetermined (but unknown). Randomized WM will guarantee you an expected gain of at least $d - \sqrt{2d \log n}$.

[Multiply weight by $1 + \varepsilon$ whenever ball falls in.]

Analysis: just like before

- Let W_t be total weight at time t.
- Let F_t be fraction of weight on bucket that ball falls in at time t. So, our E[gain] $= \sum F_t$.
- $W_{t+1} = W_t + \epsilon W_t F_t = W_t(1 + \epsilon F_t)$.
- $(1+\epsilon)^d \leq W_{\text{Final}} = n \prod_t (1 + \epsilon F_t)$.
- $d \ln(1+\epsilon) \leq \ln n + \sum_t \ln(1 + \epsilon F_t)$.
- We know $\ln(1+x) \leq x$, so:

$$\begin{aligned} d \ln(1+\epsilon) &\leq \ln n + \sum_t \epsilon F_t \\ &= \ln n + \epsilon[\text{Our Expected Gain}]. \end{aligned}$$

- Using $\ln(1+\epsilon) \geq \epsilon - \epsilon^2/2$, we get:

$$\boxed{\text{Our expected gain} \geq d\left(1 - \frac{\epsilon}{2}\right) - \frac{\ln n}{\epsilon}}$$

Now, plug in $\epsilon = \sqrt{(\ln n)/2d}$ to get $d - \sqrt{2d \ln n}$

What if there is a movement cost?

- This is unrealistic. You can't just switch jobs every day. (Vesting, etc.)
- Let's model as costing $k to switch.
- To apply the same algorithm, we shouldn't think of picking *independently* from the new distrib at each time step.

Eg.,

w_j :	1	2	1	2	3	1
p_j :	1/10	1/5	1/10	1/5	3/10	1/10
w'_j :	1	4	1	2	3	1
p'_j :	1/12	1/3	1/12	1/6	1/4	1/12

Rule is: "if was at position 2 then stay there. Else, move to position 2 with probability 1/6."

General case: "if ball falls in location i then: if was at position i, stay there. Else move to i with probability $p'_i\left(\frac{\varepsilon}{1+\varepsilon}\right)$."

Movement costs (contd)

How much does this hurt us?

- If not at i, we moved with prob $p'_i\left(\frac{\varepsilon}{1+\varepsilon}\right)$.
- Note that $p'_i < p_i(1+\varepsilon)$.
- So, expected movement cost at most $k\varepsilon p_i$.
- But, remember our expected gain was p_i. So, we are really getting $(1-k\varepsilon)$ times as much as we thought.

Optimal ε is now (roughly) $\sqrt{\frac{\ln n}{(k+1/2)d}}$ giving us a guarantee of

$$d - \sqrt{(4k+2)2d\ln n}.$$

What does this tell us?

- If we have several online strategies and don't know which to use,
- and if at any point in time we can be following one, but monitoring how well we would have done had we been using another,

  (e.g., think of as investments)

- and if the cost to switch between strategies is at most some constant k,

$\Rightarrow$ then can still do "nearly as well as" best in hindsight.

E.g., Combining paging algorithms

Disk-paging problem:

- Cache holds k memory pages. Disk has $\gg k$.
- *Page fault* if item requested is not in cache. Bring page in and throw something else out.
- Many different online algorithms: LRU, FIFO, Marking, etc.

View online algs as "experts" and do nearly as well as best in hindsight?

- Experts have state. Loss is f(state).
- To switch from following expert X to Y need (at least lazily) to switch state too.

Can view as changing metric space, but always diameter $\leq k$. Apply RWM to get:

$$\mathbf{E}[\text{cost}] \lesssim (1 + 3\varepsilon/2)[\text{best alg}] + \frac{1}{\epsilon} k \log n.$$

Self-adjusting search trees

- Have n items in a search tree.
- Get a request: "5"
- Pay based on how far down. Can also perform rotations for cost of \$1 each.

Splay trees are $O(1)$-competitive w.r.t. best static tree. Can we get $(1+\varepsilon)$?

→ One expert per tree (2^n of them).

→ Choose tree using RWM algorithm.

→ Not uniform, but movement cost $\leq 4n$.

RWM:

$$\mathbf{E}[\text{cost}] \lesssim (1 + 3\varepsilon/2)[\text{best tree}] + O(\tfrac{1}{\epsilon}n^2).$$

Of course, computation time is horrendous...

Perhaps this idea could be used to get dynamic optimality???

..more..

Let's keep milking this for all we can.

Application to adaptive game playing and proof of min-max theorem.

2-player zero-sum games

E.g., Rock-Paper-Scissors.

Payoff to row player:

	R	P	S
R	0	-1	1
P	1	0	-1
S	-1	1	0

- Minimax optimal strategy: (randomized) strategy with best worst-case guarantee.

  What is minimax optimal for RPS?

- What about the game below:

  Payoff to row player:

	N	D
N	-5	5
D	10	-10

  Optimal strategy for row player?

  Column player?

The min-max theorem

	N	D
N	-5	5
D	10	-10

- Suppose that for any (randomized) strategy of your opponent, there exists a deterministic counter-strategy for you that guarantees you an expected gain $\geq V$.

  Then, there exists a randomized strategy for you such that for any counter-strategy of the opponent, you get an expected gain $\geq V$.

- Equivalently:

$$\max_{S_{row}} \min_{S_{col}} \mathbf{E}[\text{payoff}] = \min_{S_{col}} \max_{S_{row}} \mathbf{E}[\text{payoff}]$$

I.e., suppose that for all S_{col} there exists S_{row} such that expected gain is $\geq V$. Then there exists a fixed S_{row} such that for all S_{col} the expected gain is $\geq V$ too.

(strategy $\equiv$ randomized strategy)

Using RWM for online play

- rows are "experts". Pick row j with probability w_j/W. (This time we *will* pick independent from the past.)
- Penalize rows/experts based on outcome.

  (Technically, let's scale matrix entries to be in the range $[0, 1]$)

	R	P	S
R	1/2	0	1
P	1	1/2	0
S	0	1	1/2

- For any sequence of games,

$$\text{Our expected gain} \geq d\left(1 - \frac{\epsilon}{2}\right) - \frac{\ln n}{\epsilon},$$

  where d is gain of best fixed strategy in hindsight (which may or may not be the minimax optimal).

We've actually just *proven* the Min-max theorem.

Why?

What would it mean for min-max to be false?

→ If we know opponents randomized strategy, we can get expected gain $\geq V$, but if we have to choose our randomized strategy first, then opponent can force us to get $\leq V - \delta$.

This contradicts our bound if we use $\epsilon = \delta$. Our gain per game is approaching $\mathrm{OPT}(1 - \epsilon/2)$.

In other words: If there was a gap (V versus $V - \delta$), then for *any* randomized strategy we choose, opponent who knows our strategy should be able to force us to get no more than $V - \delta$ on average per play.

But, we are doing better.

Shifting bounds

This is nice but not fully satisfying. If opponent plays R 100 times, alg will win a lot but...

- Ends up with $w_P = 1, w_R = 2^{-50}, w_S = 2^{-100}$.
- So, if opponent switches to S, alg will lose 50 times in a row.

How to fix?

- Lower-bound weights (need to be a careful)
- Weight-sharing: if remove X from total weight, give back αX, distributed evenly.

For all t, our expected gain is at least (approx):

$$d_t(1-\varepsilon) - (t+1)\frac{\ln n}{\epsilon}$$

where d_t is the gain of optimal strategy in hindsight that switches between experts t times.

Relation to MTS problem

MTS problem:

- Have a machine that can be in one of n states.
- Can move machine from state i to state j at cost d_{ij}, where d is a distance metric.
- We are given, on-line, a sequence of tasks $\sigma = T_1, T_2, T_3, \ldots$. Each task has a cost vector, e.g., $T_1 = (2, 9, \ldots, 10)$, that gives the cost of processing the task in each of the n states of the machine.

$\rightarrow$ Given a new task T, we can stay in the current state i, paying $T[i]$, or move to a new state j and perform it there, paying $d_{ij} + T[j]$.

Relation to MTS problem.

Expert advice	MTS problem
n Experts	n States
loss vector $(\ell_1, \ldots, \ell_n)$	cost vector $(c_1, \ldots, c_n)$
Movement is free	Costs d_{ij} to move
0-step lookahead	1-step lookahead

Another difference: performance measure.

- Competitive ratio: compete against best in hindsight by arbitrary algorithm.
- Best-expert: compete against best single state/ expert in hindsight.
- Shifting bounds: Like CR, but split cost of competitor into "local" and "movement" portions.

Relation to on-line investing problems

Similarities:

Stocks	$\leftrightarrow$	Experts/States
allocation of wealth among investments	$\leftrightarrow$	probability distribution over experts/states
Stock i drops by ℓ_i %	$\leftrightarrow$	Expert i has loss ℓ_i
Transaction costs	$\leftrightarrow$	distance metric

Differences:

- In Experts case, we care about sum of losses. In Stocks case we care about product of $(1 + \Delta)$'s.
- Change in value of stock X "automatically" changes fraction of wealth in stock X.

Concept Learning

Imagine learning algorithm trying to decide which loan applicants are bad credit risks.

- Say each individual represented by n features. (e.g., income range, recent delinquencies, job type, etc.)
- Target function $f(x) = 1$ if x is a bad risk, $f(x) = 0$ if x is a good risk.
- Goal of algorithm is to produce good prediction rule $h(x)$.

Batch learning: Alg is given sample S presumed to be drawn from distribution D. Goal is for h to be close to f over D.

On-line learning: learning as a sequence of trials. In each trial, algorithm is given x, asked to predict f, and then is told the correct value. Goal is to minimize number of mistakes.

Example: learning an OR function

- Suppose features are boolean: $x \in \{0,1\}^n$. (e.g., 100101010111)
- 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? E.g.,

```
0 1 0 1 1
              +
0 1 1 0 0
              -
0 0 1 1 0
              -
1 0 1 0 0
              +
0 0 0 0 1
              +
```

A simple alg for learning OR functions

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

  Invariant: $\{$vars in $h\} \supseteq \{$vars in target$\}$.

- If mistake on negative example x, then for each $x_i = 1$, remove x_i from h.
  - Maintains invariant.
  - $|h|$ decreases by at least 1.

- Invariant implies no mistakes on positives.

- So, at most n mistakes.

A 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 $a_1 x_1 + \ldots + a_n x_n \geq t$, else is negative.
- An OR function is the case $a_i \in \{0,1\}$, $t = 1$.

Q: can you guarantee at most $poly(n)$ mistakes?

Yes. Use the ellipsoid algorithm.

- Examples seen so far form a set of linear constraints.
- Center of ellipsoid is hypothesis.
- Mistake can be viewed as output of separation oracle.

Typical problems studied

- Learning a concept class C: Assume label $\ell = f(x)$ for some unknown $f \in C$.

  E.g., OR functions, decision lists, LTFs.

- "Agnostic" learning with respect to C (learning C with worst-case noise): No assumption about labels. Try to do nearly as well as best $h \in C$.

  E.g., single-variable concepts. NP-hard for OR functions if require hyp in C too.

- Learning C with random classification noise: Assume label $\ell = f(x)$ but then flipped with some probability p. Oblivious adversary (usually study in batch model).

  E.g., can convert most no-noise algs to this model.

What if computation time is no object?

- In no-noise case, can learn class C with at most $\log_2 |C|$ mistakes.

  (e.g., at most 2^n OR functions; or, think of boolean formulas that can be described in at most s bits)

  $\rightarrow$ Use "halving algorithm".

- What about agnostic/unrealizable case? Can we do nearly as well as best $f \in C$?

  $\rightarrow$ Use weighted-majority.

  Get roughly

$$(1 + \varepsilon/2)(\#\text{mistakes of best } f) + \frac{1}{\varepsilon}\ln(|C|)$$

Learning an OR function revisited

- Goal: Learn an OR of r of n variables with mistake bound $O(r \log n)$.

Winnow Algorithm: [Littlestone]

- Use linear threshold function for prediction. Predict positive when

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \geq n.$$

- Init: all $w_i = 1$.
- Update rule: only modify on a mistake.
  - Mistake on positive:

    If $x_i = 1$ then $w_i \leftarrow 2w_i$.
  - Mistake on negative:

    If $x_i = 1$ then $w_i \leftarrow w_i/2$.

Example

```
Weights:   1   1   1   1   1
Example:   0   1   0   1   1
                       Predict: -,  Truth: +

Weights:   1   2   1   2   2
Example:   1   1   1   0   0
                       Predict: -,  Truth: +

Weights:   2   4   2   2   2
Example:   0   1   1   0   0
                       Predict: +,  Truth: -

Weights:   2   2   1   2   2
Example:   1   0   1   0   0
```

Analysis idea:

- Each mistake on positive doubles at least one relevant weight.
- Each mistake on negative removes a lot from the total weight.

Analysis

Let's call weights on features in target the "relevant weights."

- Relevant weights never decrease.
- Each mistake on positive doubles at least one relevant weight.
- Each relevant weight can be doubled at most $\log_2 n + 1$ times.

So, at most $r(\log_2 n + 1)$ mistakes on positives.

Analysis (cont)

Now look at total weight:

- Total weight starts at n.
- Each mistake on positive adds at most n to total weight.
- Each mistake on negative removes at least $n/2$ from total weight.
- Total weight always > 0.

$$\# \text{ mistakes on neg } \; < \; 2 + 2(\# \text{ mistakes on pos})$$

So, total number of mistakes at most:

$$2 + 3r(\log_2 n + 1).$$

Applications

Can think of winnow as an algorithm for "sleeping experts" / "specialists":

- Like the "experts" setting but, at each trial, some (or most) of the predictors can abstain.
- E.g., specialist is a simple if/then rule.

  "If document has Clinton then it must be about politics."
- # specialists total $\gg$ # active at any given time $\gg$ # you should actually be paying attention to.
- Want to do nearly as well as best "small coalition" of them.

Winnow-specialists

- Start each specialist at a weight of 1.
- Predict with (weighted) majority vote.
- *If global algorithm makes a mistake*,
  - Multiply incorrect specialists' weights by $\frac{1}{2}$.
  - Multiply correct specialists' weights by 2.

(several different variations)

Specialists are rewarded for predicting correctly when they *should have been listened to more carefully*.

Application: Calendar Scheduling

- An example is an event to schedule on calendar. (e.g., a meeting)

  Described using features based on information entered + background knowledge.

- Goal: suggest location, duration, start time, day of week.

A typical (labeled) example for location:

```
(add-event-3-6-1992-13-26-26 *novalue*
 (req-event-type meeting)
 (req-seminar-type nil)
 (sponsor-attendees mitchell)
 (department-attendees *inferred.novalue*)
 (position-attendees *inferred.novalue*)
 (group-attendees? yes)
 (req-course-name nil)
 (department-speakers *inferred.novalue*)
 (group-name calendar-group)
 (lunchtime? no)
 (single-person? no)
 (number-of-person 4)
 (req-location weh5309))
```

Putting into our framework

- Idea: create one specialist for every possible rule of length 2:

```
If (feature-X has value A and
    feature-Y has value B)
then predict <recent history>
```

  E.g., *If* department is Robotics and position is grad-student *then* `most-recent-outcome`.

- $(N = 59731,\ n = 561)$

Results: fraction correct on 1685 calendar events:

Prediction	ID3	WM	Specialists
location	0.64	0.70	0.75
duration	0.63	0.64	0.71
start-time	0.34	0.39	0.51
day-of-week	0.50	0.56	0.57

Another example: text categorization

- Want to classify documents into some set of categories.
- One specialist for each word or small phrase. Specialist is “awake” when the word/phrase appears.

Implementation of [Cohen, Singer]: split specialist into one per category. Also somewhat different update rule.

Results (average number of errors):

	Rocchio	Ripper	1-word exp	4-word exp
AP titles	91.11	84.56	92.33	80.33
TREC-AP	498.10	456.60	476.80	439.65

When is it a good idea?

- When one expects the right set of simple rules to be powerful.
- When lots of simple rules can be generated automatically.

Also, robust to time-dependence.

Batch model

- Examples drawn from distrib D, labeled by f.
- Alg (ϵ, δ)-learns a class C in m examples if $\forall D, \forall f \in C$,

$$\Pr_{D^m}[\text{Alg produces hyp with error } > \epsilon] < \delta$$

For instance:

- If true error of h is $> \epsilon$, then the probability h is consistent with m samples is $< (1-\epsilon)^m$.
- The chance *any* such $h \in C$ is consistent is at most $|C|(1-\epsilon)^m < \delta$ for $m = \frac{1}{\epsilon}\ln(|C|/\delta)$.
- So, it suffices to just find a consistent $h \in C$ if we have this much data.

(E.g., $|C| = 1000$, $\varepsilon = 0.25$, $\delta = 0.01$, then $m = 40$ works: $(1-\varepsilon)^m = 1/100000$)

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.

This is a little funny because it holds for any bias (preference ordering) that can be translated into bits.

Theory says that whatever your bias, "simple" rules are unlikely to fool you. On the other hand, there's no guarantee that there *will* be a simple explanation consistent with data. That depends on your bias/representation.

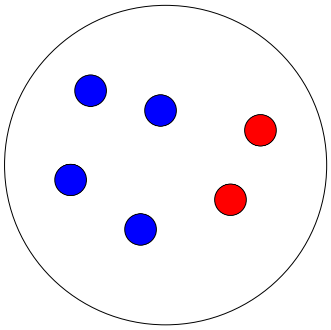
Boosting: a feedback-based voting scheme

Idea:

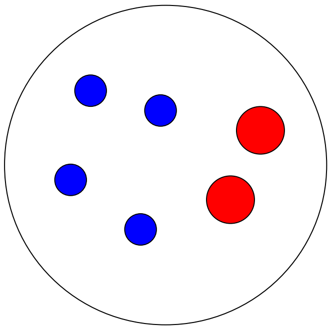
- Given a learning algorithm L and a data set S, force L to generate *several different* good hypotheses.
- Then combine them with (weighted) majority vote.

Boosting

- First run algorithm on random sample of data set. Get hypothesis h.



- Next time: reweight the data points so that $error(h) = 1/2$.



- Repeat. Reweight each time so that previous rule behaves no better than random guessing. At end, combine with weighted majority vote.

Guarantee: if algorithm continues to produce good hypotheses, then $error(vote) \to 0$.

Confidence-weighted boosting

- Apply to specialist-type rules.
- Like a soft-version of decision lists.

Recent applications of Boosting

- Text categorization
- Part-of-speech tagging
- Text filtering
- Optical Character Recognition
- Combining web search-engine results

Open Problems

- Learn Decision-Lists Attribute-Efficiently?

  A DL is a formula like: If x_2 then $+$, else if x_5 then $-$, else if x_3 then $+$, else $-$.

  Goal: learn DL of length r with Mistake-Bound $poly(r, \log n)$ and running time $poly(n)$.

- Learn Decision-Lists over String-Valued features?

  E.g., If x_2 = "sunny" then $+$, else if x_5 = "cold" then $-$, else $+$.

  Goal: Mistake bound $poly(n, r)$, but independent of number of different strings in universe. Running time per stage poly in length of example.

- How about parity functions?

Open Problems, contd

- What competitive ratio can be achieved for learning with respect to the best OR function?

  Suppose no consistent OR function. Can we achieve mistake bound $cm^* + p(n)$, where m^* is the number of mistakes of the best OR function, p is a polynomial, and $c \leq n^\alpha$ for $\alpha < 1$?

  [Recent result gets $\alpha = 1/2$]

- How about weak-learning in batch model?

  Assume that over D, there exists an OR function consistent with 90%. Can we achieve an error rate at most $1/2 - 1/poly(n)$?

- Can parity be learned with *random* noise?

- Can tolerance to random noise be boosted?

Portfolio selection

- We're considering n stocks.
- Our distribution of wealth is some vector $\mathbf{b}$. E.g., (1/3, 1/3, 1/3).
- Each period, get vector of "price-relatives" $\mathbf{x}$. E.g., (0.98, 1.02, 1.00).
- Our wealth gets multiplied by $\mathbf{b} \cdot \mathbf{x}$.

E.g., 2 stocks, we have 50% of our wealth in each. Stock 1 doubles, stock 2 cut in half.

New wealth $= 2 \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = 5/4$ times old wealth.

If each period, alg A does 99% as well as alg B, then in t steps,

$$\text{A's wealth} = (0.99)^t(\text{B's wealth}).$$

Looks nicer if take logs:

$$\ln(A) = \ln(B) - 0.01t$$

Competing against the best stock

In a sense, stock problem is "harder" than experts problem:

- In worst case, can't hope to do better than $\frac{1}{n}\times$(performance of best stock in hindsight).

On the other hand, in a sense, it's easier:

- There's a simple strategy to perform at least this well. Or, if take logs,

$$\ln(\text{online}) \geq \ln(\text{optimal stock}) - \ln(n).$$

"Split money evenly at start and let it sit."

Constant-rebalanced Portfolios

CRP(**b**): at end of each period, rebalance back to same distribution of wealth **b**.

- Common type of strategy.
- Takes advantage of market volatility. "Buy low, sell high."

E.g., look a 2 stocks. Strategy CRP$(1/2, 1/2)$. Suppose stock #1 stays constant (cash), price of stock #2 alternately doubles and halves:

stock 1	stock 2	our wealth
1	1	1
1	2	$3/2$
1	1	$9/8$
1	2	$(3/2)(9/8)$
1	1	$(9/8)^2$

How well can we compete against best of these?

Competing against optimal CRP

What if we split up \$ evenly among all CRPs and let it sit?

[Cover][Cover&Ordentlich]:

- Wealth $\geq$ (best CRP)$/(t+1)^{n-1}$.
- Better split: Wealth $\geq$ (best CRP)$/\sqrt{(t+1)^{n-1}}$.

Per-day ratio $\rightarrow$ 1.

[Blum & Kalai]:

- Simpler proof of 1st result.
- Extends to transaction costs.

[Helmbold,Schapire,Singer,Warmuth]:

- "experts"-based algorithm. More aggressive.
- Worse guarantees but better on historical data.

Analysis (no commissions)

Split up $ evenly among all CRPs.

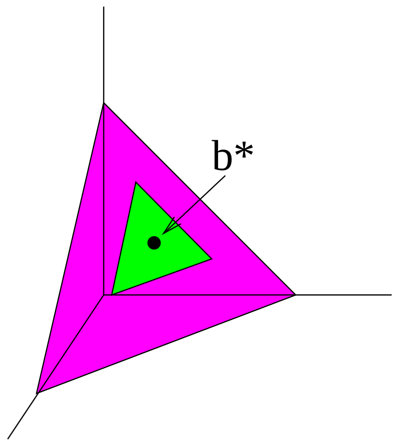
- Nearby portfolios perform similarly:

  If $\mathbf{b} = (1-\alpha)\mathbf{b}_1 + \alpha\mathbf{b}_2$ then

  $$\text{wealth}(\mathbf{b}) \geq (1-\alpha)^t \text{wealth}(\mathbf{b}_1).$$

- Set of CRPs is a simplex.
  Set of CRPs near to $\mathbf{b}^*$ is a smaller simplex.

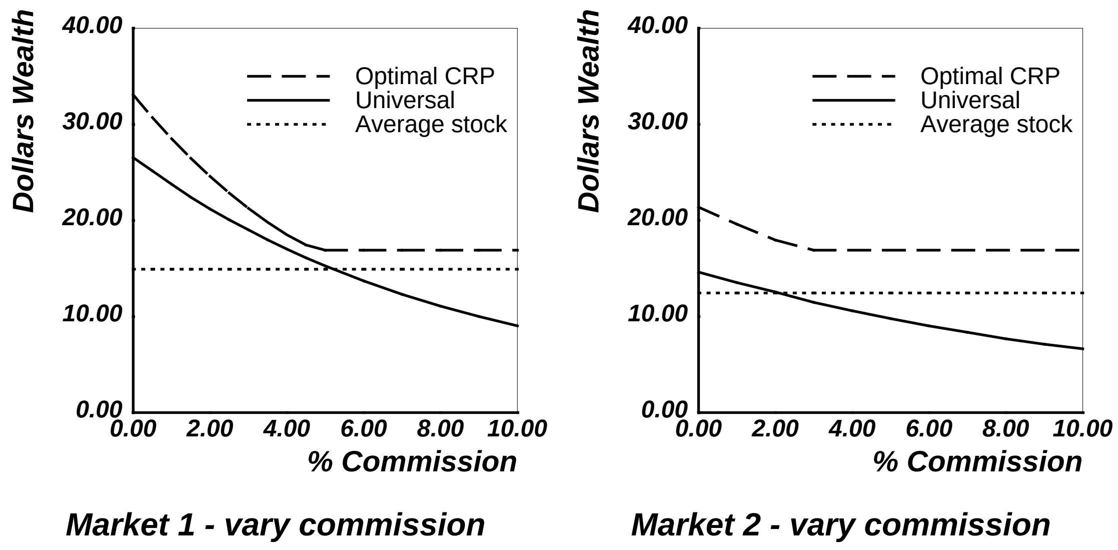
  $$Vol_{n-1}\{(1-\alpha)\mathbf{b}^* + \alpha\mathbf{b}_2\} = \alpha^{n-1} Vol_{n-1}(\mathbf{b})$$



- $(1/t)^{n-1}$ fraction of initial $ is performing within $1/e$ of optimal.

Discussion

- Performance on two sets of $n = 3$ stocks, over 22 years, rebalancing monthly.



- If commissions are high, can rebalance less often.

- Can apply to combining language models.