

## 15-859(B) Machine Learning Theory

### Lecture 1: intro, models and basic issues

Avrim Blum  
01/16/07

### Machine learning can be used to...

- recognize speech, steer cars/robots,
- play games,
- adapt programs to users,
- categorize documents, ...

#### **Goals of machine learning theory:**

develop and analyze models to understand...

- what kinds of tasks we can hope to learn, and from what kind of data,
- what types of guarantees might we hope to achieve,
- other common issues that arise.

### A typical setting

- Imagine you want a computer program to help you decide which email messages are spam and which are important.
- Might represent each message by  $n$  features. (e.g., return address, keywords, spelling, etc.)
- Take sample  $S$  of data, labeled according to whether they were/weren't spam.
- 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.,

sales	sex	Mr.	bad spelling	known-sender	spam?
Y	N	Y	Y	N	Y
N	N	N	Y	Y	N
N	Y	N	N	N	Y
Y	N	N	N	Y	N
N	N	Y	N	Y	N
Y	N	N	Y	N	Y
N	N	Y	N	N	N
N	Y	N	Y	N	Y

Given data, some reasonable rules might be:

- Predict SPAM if  $\neg$ known AND (sex OR sales)
- Predict SPAM if sales + sex - known  $> 0$ .

•...

### Big questions

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

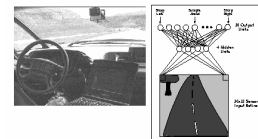
(B) What kind of confidence do we have that they will do well in the future?  
[confidence bound / sample complexity]

for a given learning alg, how much data do we need...

### Power of basic paradigm

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

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

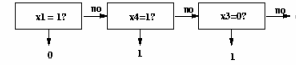


## Natural formalization (PAC)

- We are given sample  $S = \{(x,y)\}$ .
  - Assume  $x$ 's come from some fixed probability distribution  $D$  over instance space.
  - View labels  $y$  as being produced by some unknown target function  $f$ .
- Alg does optimization over  $S$  to produce some hypothesis (prediction rule)  $h$ .
- Goal is for  $h$  to do well on new examples also from  $D$ . I.e.,  $\Pr_D[h(x) \neq f(x)] < \epsilon$ .

err(h)

## Example of analysis: Decision Lists



Say we suspect there might be a good prediction rule of this form.

- Design an efficient algorithm **A** that will find a consistent DL if one exists.
- Show that if  $S$  is of reasonable size, then  $\Pr[\text{exists consistent DL } h \text{ with } \text{err}(h) > \epsilon] < \delta$ .
- This means that **A** is a good algorithm to use if  $f$  is, in fact, a DL.

If  $S$  is of reasonable size, then **A** produces a hypothesis that is Probably Approximately Correct.

## 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 this fails, then:

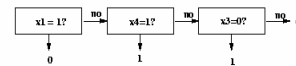
- No DL consistent with remaining data.
- So, no DL consistent with original data.

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

## Confidence/sample-complexity

- Consider some DL  $h$  with  $\text{err}(h) > \epsilon$ , that we're worried might fool us.
  - Chance that  $h$  is consistent with  $S$  is at most  $(1-\epsilon)^{|S|}$ .
  - Let  $|H|$  = number of DLs over  $n$  Boolean features.  $|H| < n!4^n$ . (for each feature there are 4 possible rules, and no feature will appear more than once)
- So,  $\Pr[\text{some DL } h \text{ with } \text{err}(h) > \epsilon \text{ is consistent}] < |H|(1-\epsilon)^{|S|} < n!4^n(1-\epsilon)^{|S|}$ .
- This is  $< \delta$  for  $|S| > (1/\epsilon)[\ln(|H|) + \ln(1/\delta)]$

## Example of analysis: Decision Lists



Say we suspect there might be a good prediction rule of this form.

- Design an efficient algorithm **A** that will find a consistent DL if one exists.
- Show that if  $|S|$  is of reasonable size, then  $\Pr[\text{exists consistent DL } h \text{ with } \text{err}(h) > \epsilon] < \delta$ .
- So, if  $f$  is in fact a DL, then whp **A**'s hypothesis will be approximately correct. "PAC model"

## PAC model more formally:

- We are given sample  $S = \{(x,y)\}$ .
  - Assume  $x$ 's come from some fixed probability distribution  $D$  over instance space.
  - View labels  $y$  as being produced by some target function  $f$ .
- Alg does optimization over  $S$  to produce some hypothesis (prediction rule)  $h$ . Goal is for  $h$  to do well on new examples also from  $D$ . I.e.,  $\Pr_D[h(x) \neq f(x)] < \epsilon$ .

Algorithm PAC-learns a class of functions  $C$  if:

- For any given  $\epsilon > 0, \delta > 0$ , any target  $f \in C$ , any dist.  $D$ , the algorithm produces  $h$  of  $\text{err}(h) < \epsilon$  with prob. at least  $1 - \delta$ .
  - Running time and sample sizes polynomial in relevant parameters:  $1/\epsilon, 1/\delta, n, \text{size}(f)$ .
  - Require  $h$  to be poly-time evaluable. Learning is called "proper" if  $h \in C$ . Can also talk about "learning  $C$  by  $H$ ".
- We just gave an alg to PAC-learn decision lists.

## Confidence/sample-complexity

- What's great is there was nothing special about DLs in our argument.
- All we said was: "if there are not *too* many rules to choose from, then it's unlikely one will have fooled us just by chance."
- And in particular, the number of examples needs to only be proportional to  $\log(|C|)$ .  
(notice big difference between  $|C|$  and  $\log(|C|)$ .)

## Occam's razor

William of Occam (~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 can be  $< s$  bits long.
- So, if the number of examples satisfies:

$$|S| > (1/\epsilon)[s \ln(2) + \ln(1/\delta)]$$

Think of as 10x #bits to write down  $h$ .

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

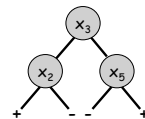
## Occam's razor (contd)<sup>2</sup>

Nice interpretation:

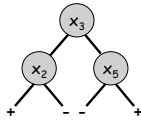
- Even if we have different notions of what's simpler (e.g., different representation languages), we can both use Occam's razor.
- Of course, there's no guarantee there will be a short explanation for the data. That depends on your representation.

## Decision trees

- Decision trees are not known to be PAC-learnable.
- Given any data set  $S$ , it's easy to find a consistent DT if one exists. How?
- Where does the DL argument break down?
- Simple heuristics used in practice (ID3 etc.) don't work for all  $c \in C$  even for uniform  $D$ .
- Would suffice to find the (apx) smallest DT consistent with any dataset  $S$ , but that's NP-hard.



## Decision trees



- Decision trees are not known to be PAC-learnable.
- Given any data set  $S$ , it's easy to find a consistent DT if one exists. How?
- Would suffice to find the (apx) smallest DT consistent with any dataset  $S$ , but that's NP-hard.
- If  $P=NP$  then every class  $C$  is PAC learnable.\*

\*assuming all  $h \in C$  are poly-time evaluable.  
(if size(f) not known, do guess-and-double)

## More examples

Other classes we can PAC-learn: (how?)

- Monomials [conjunctions, AND-functions]
  - $x_1 \wedge x_4 \wedge x_6 \wedge x_9$
- 3-CNF formulas (3-SAT formulas)
- OR-functions, 3-DNF formulas
- k-Decision lists (each if-condition is a conjunction of size k), k is constant.

Given a data set  $S$ , deciding if there is a consistent 2-term DNF formula is NP-complete. Does that mean 2-term DNF is hard to learn?

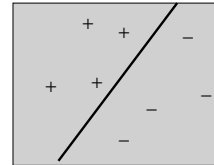
## More about the PAC model

Algorithm PAC-learns a class of functions  $C$  if:

- For any given  $\epsilon > 0$ ,  $\delta > 0$ , any target  $f \in C$ , any dist.  $D$ , the algorithm produces  $h$  of  $\text{err}(h) \leq \epsilon$  with prob. at least  $1 - \delta$ .
- Running time and sample sizes polynomial in relevant parameters:  $1/\epsilon$ ,  $1/\delta$ ,  $n$ ,  $\text{size}(f)$ .
- Require  $h$  to be poly-time evaluable. Learning is called "proper" if  $h \in C$ . Can also talk about "learning  $C$  by  $H$ ".
- What if your alg only worked for  $\delta = \frac{1}{2}$ , what would you do?
- What if it only worked for  $\epsilon = \frac{1}{4}$ , or even  $\epsilon = \frac{1}{2} - 1/n$ ? This is called weak-learning. Will get back to later.
- Agnostic learning model: Don't assume anything about  $f$ . Try to reach error  $\text{opt}(H) + \epsilon$ .

## Extensions we'll get at later:

- Replace  $\log(|H|)$  with "effective number of degrees of freedom".



- There are infinitely many linear separators, but not that many really different ones.
- Other more refined analyses.

## 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.

- Bound number of mistakes given that  $f \in C$ .
- Show that our algorithm does nearly as well as best predictor in  $C$ .

## 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
...	...	...	...	...

Basic question: Is there a strategy that allows us to do nearly as well as best of these in hindsight?

["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.

Ø Each mistake cuts # available by factor of 2.

Ø Ignoring computational issues, can learn any class  $C$  making only  $\log(|C|)$  mistakes. "Halving algorithm".

### What if no expert is perfect?

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
weights	1	1	1	.5	
predictions	Y	N	N	Y	N
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,

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

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

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

With improved settings + rand., can get  $M < 1.07m + 8 \lg n$ .

### What can we use this for?

- Can use to combine multiple algorithms to do nearly as well as best in hindsight.
  - E.g., online control policies.
- Extension: "sleeping experts". Combining "if-then" rules. If any rule does well then you should too in the set of times it fires.
- More extensions: "bandit problem", movement costs.

### Other models we'll explore later

Some scenarios allow more options for algorithm.

- "Active learning": have large unlabeled sample and alg may choose among these.
  - E.g., web pages, image databases.
- Or, allow algorithm to construct its own examples. "Membership queries"
  - E.g., features represent variable-settings in some experiment, label represents outcome.
  - Gives algorithm more power.

### Summary

- Saw two basic models (PAC, online learning).
- Simple theoretical models can give insight into basic issues. E.g., Occam's razor.
- Even if models aren't perfect, can often lead to good algorithms. Often diverse problems best solved by fitting into basic paradigm(s).
- Next time: more on online learning. Connections to info-theory and game-theory.