15-859(B) Machine Learning Theory

Lecture 1: intro, basic models and issues

Avrim Blum

01/12/09

Admin

http://www.cs.cmu.edu/~avrim/ML09/

- Course web page. Textbook covers about 1/2 of course material.
- 6 hwk assignments. Exercises/problems.
- Small project: explore a theoretical question, try some experiments, or read a paper and explain the idea. Short writeup and possibly presentation. Small groups ok.
- Take-home exam (worth roughly 2 hwks).
- "volunteers" for hwk grading.

OK, let's get to it…

Machine learning can be used to…

- recognize speech, faces,
- play games, steer cars,
- adapt programs to users,
- classify documents, protein sequences,...

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.,

$$	meds	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 ¬known AND ($$ OR meds)
- Predict SPAM if $$ + meds – known > 0.
- ...

Big questions

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

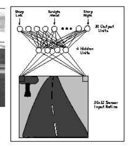
(B) What kind of confidence do we have that they will do well in the future? [confidence bound / sample complexity]

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)

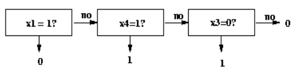
Email msg — Spam or not?

- We are given sample S = {(x,y)}.
  - View labels y as being produced by some target function f.
- Alg does optimization over S to produce some hypothesis (prediction rule) h.
- Assume S is a random sample from some probability distribution D. Goal is for h to do well on new examples also from D.

  I.e., $Pr_D[h(x) \neq f(x)] < \varepsilon$.

  err(h)

Example of analysis: Decision Lists



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

1. Design an efficient algorithm **A** that will find a consistent DL if one exists.
2. Show that if S is of reasonable size, then Pr[exists consistent DL h with err(h) > ε] < δ.
3. 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 **P**robably **A**pproximately **C**orrect.

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

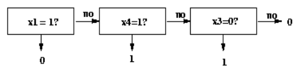
- Consider some DL h with err(h)>ε, that we're worried might fool us.
- Chance that h is consistent with S is at most $(1-\varepsilon)^{|S|}$.
- Let |H| = number of DLs over n Boolean features. $|H| < n!4^n$. (for each feature there are 4 possible rules, and no feature will appear more than once)

So, Pr[some DL h with err(h)>ε is consistent] $< |H|(1-\varepsilon)^{|S|} < n!4^n(1-\varepsilon)^{|S|}$.

- This is < δ for $|S| > (1/\varepsilon)[\ln(|H|) + \ln(1/\delta)]$

  or about $(1/\varepsilon)[n \ln n + \ln(1/\delta)]$

Example of analysis: Decision Lists



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

1. Design an efficient algorithm **A** that will find a consistent DL if one exists. DONE
2. Show that if |S| is of reasonable size, then Pr[exists consistent DL h with err(h) > ε] < δ. DONE
3. 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., $Pr_D[h(x) \neq f(x)] < \varepsilon$.

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

- For any given $\varepsilon>0$, $\delta>0$, any target f $\in$ *C*, any dist. D, the algorithm produces h of err(h)<ε with prob. at least 1-δ.
- Running time and sample sizes polynomial in relevant parameters: 1/ε, 1/δ, n (size of examples), size(f).
- Require h to be poly-time evaluatable. 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.

PAC model more formally:

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

- For any given $\varepsilon>0$, $\delta>0$, any target f $\in$ *C*, any dist. D, the algorithm produces h of err(h)<ε with prob. at least 1-δ.
- Running time and sample sizes polynomial in relevant parameters: 1/ε, 1/δ, n (size of examples), size(f).
- Require h to be poly-time evaluatable. Learning is called "proper" if h $\in$ *C*. Can also talk about "learning *C* by H".

PAC model more formally:

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

- For any given $\varepsilon>0$, $\delta>0$, any target f $\in$ *C*, any dist. D, the algorithm produces h of err(h)<ε with prob. at least 1-δ.
- Running time and sample sizes polynomial in relevant parameters: 1/ε, 1/δ, n (size of examples), size(f).
- Require h to be poly-time evaluatable. Learning is called "proper" if h $\in$ *C*. Can also talk about "learning *C* by H".

Some notes:

- Can either view alg as requesting examples (button/oracle model) or just as function of S, with guarantee if S is suff. lg.
- "size(f)" term comes in when you are looking at classes where some fns could take > poly(n) bits to write down. (e.g., decision trees, DNF formulas)

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:

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

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

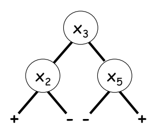
Then it's unlikely a bad simple explanation will fool you just by chance.

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.

Decision trees



- Decision trees over $\{0,1\}^n$ 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.

If computation-time is no object, then any class is PAC-learnable

- Occam bounds ⇒ any class is learnable if computation time is no object:
  - Let $s_1=10$, $\delta_1 = \delta/2$. For i=1,2,... do:
    - Request $(1/\varepsilon)[s_i + \ln(1/\delta_i)]$ examples S_i.
    - Check if there is a function of size at most s_i consistent with S_i. If so, output it and halt.
    - $s_{i+1} = 2s_i$, $\delta_{i+1} = \delta_i/2$.
  - At most $\delta_1 + \delta_2 + \ldots \leq \delta$ chance of failure.
  - Total data used: $O((1/\varepsilon)[\text{size}(f)+\ln(1/\delta)\ln(\text{size}(f))])$.

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 examples

Hard to learn C by C, but easy to learn C by H, where H = {2-CNF}.

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 ε>0, δ>0, any target f ∈ C, any dist. D, the algorithm produces h of err(h)<ε with prob. at least 1-δ.
- Running time and sample sizes polynomial in relevant parameters: 1/ε, 1/δ, n, size(f).
- Require h to be poly-time evaluatable. Learning is called "proper" if h ∈ 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 $\varepsilon = \frac{1}{4}$, or even $\varepsilon = \frac{1}{2}-1/n$? This is called weak-learning. Will get back to later.
- Agnostic learning model: Don't assume anything about f. Try to reach error opt(H) + ε.

More about the PAC model

Algorithm PAC-learns a class of functions C if:

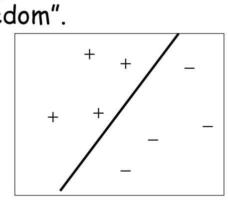
- For any given ε>0, δ>0, any target f ∈ C, any dist. D, the algorithm produces h of err(h)<ε with prob. at least 1-δ.
- Running time and sample sizes polynomial in relevant parameters: 1/ε, 1/δ, n, size(f).
- Require h to be poly-time evaluatable. Learning is called "proper" if h ∈ C. Can also talk about "learning C by H".

Drawbacks of model:

- In the real world, labeled examples are much more expensive than running time. Poly(size(f)) not enough.
- "Prior knowledge/beliefs" might be not just over form of target but other relations to data.
- Doesn't address other kinds of info (cheap unlabeled data, pairwise similarity information).
- Only considers "one shot" learning.

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.

Some open problems

Can one efficiently PAC-learn…

- an intersection of 2 halfspaces? (2-term DNF trick doesn't work)
- C={fns with only O(log n) relevant variables}? (or even O(loglog n) or ω(1) relevant variables)? This is a special case of DTs, DNFs.
- Monotone DNF over uniform D?
- Weak agnostic learning of monomials.