TTIC 31250
An Introduction to the Theory of Machine Learning

Avrim Blum

03/28/22

Lecture 1: logistics, intro, basic models and issues

Announcements

- Course webpage:

  https://home.ttic.edu/~avrim/MLT22/index.html
- 5 homework assignments
- Small project: explore a theoretical question, try some experiments, or read a paper and explain the idea. Short (4-5 page) writeup.
- Take-home exam worth 1-2 hwks
- "Volunteers" for hwk grading.
- My office hours: Monday 3-4pm
- TA: Falcon Dai

OK, let's get to it!

Machine learning can be used to…

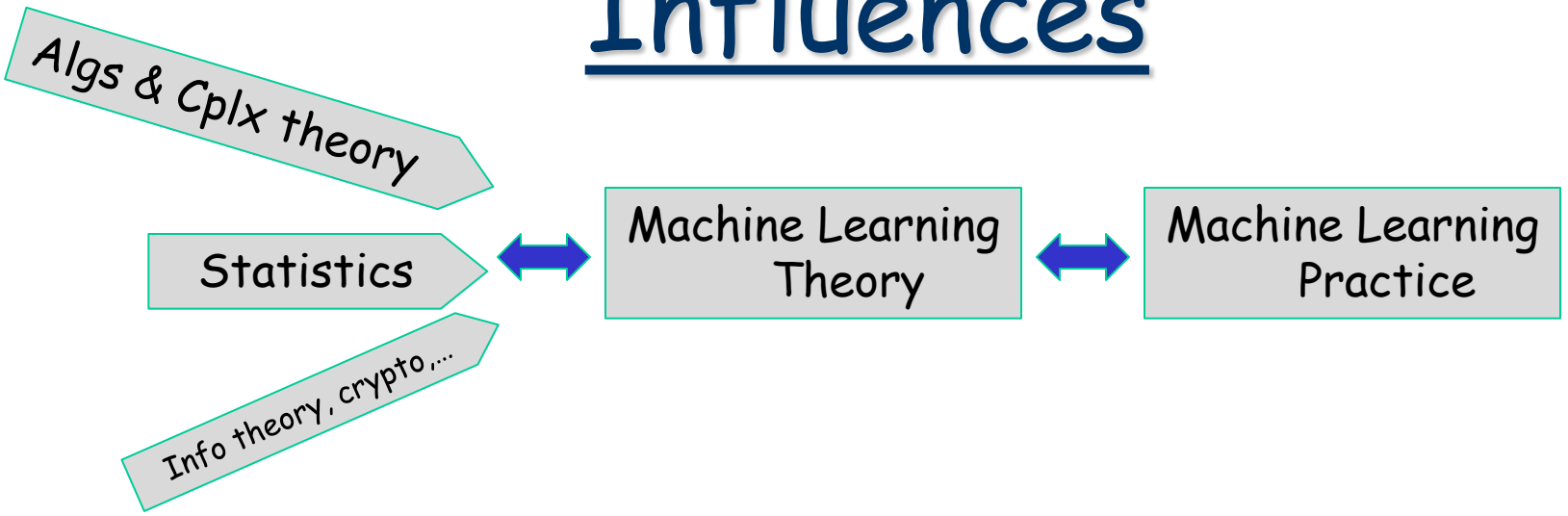
- recognize speech, faces, objects in images
- play games, steer cars,
- adapt programs to users,
- classify documents, protein sequences,…

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.

Influences



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.,

money	pills	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

a positive example

a negative example

Given data, some reasonable rules might be:

- Predict SPAM if ¬known AND (money OR pills)
- Predict SPAM if money + pills – 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, and how can we design alg to need less?

For the confidence question, we'll need some connection between future data and past data.

Natural formalization (SLT/PAC)

Email msg

Spam or not?

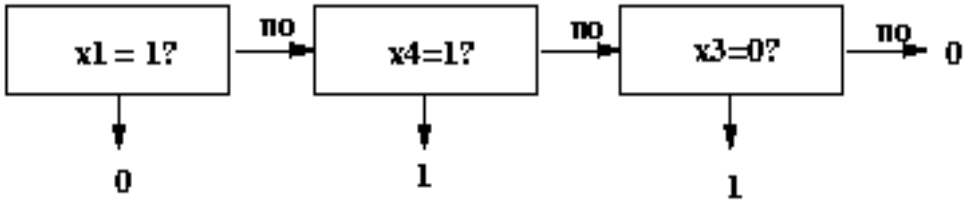
- We are given sample S = {(x,y)}.
  - View labels y as being produced by some target function f.
- Alg does optimization over S to produce some hypothesis (prediction rule) h.
- Assume S is a random sample from some probability distribution D. Goal is for h to do well on new examples also from D.

Ideally, $err_S(h)$ (error rate on sample) is low

Want $Pr_D[h(x) \neq f(x)] \leq \varepsilon$.

$err_D(h)$ = true error = true risk = expected loss

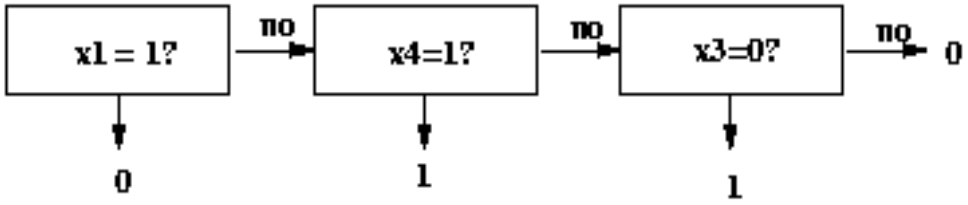
Example of analysis: Decision Lists



A Decision List is a list of if-then rules, where each rule tests the value of a single feature.

Equivalently: a decision tree that doesn't branch.

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 DL with $err_S(h) = 0$ if one exists.
2. Show that if S is of reasonable size, then Pr[∃ DL h with $err_S(h) = 0$ but $err_D(h) > \epsilon$] < 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 (gets stuck) then:

- No rule consistent with remaining data.
- So 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?

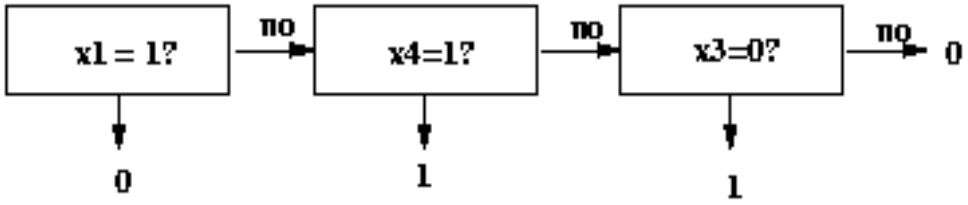
Confidence/sample-complexity

- Consider some DL h with $err_D(h) > \epsilon$, that we're worried might fool us.
- Chance that $err_S(h) = 0$ is at most $(1-\varepsilon)^{|S|}$.
- Let |H| = number of DLs over n Boolean features. $|H| \leq n! \cdot 2 \cdot 4^n$

So, Pr[∃ DL h with $err_D(h) > \varepsilon$ & $err_S(h) = 0$]
$\leq |H|(1-\varepsilon)^{|S|} \leq |H|e^{-\epsilon|S|}$.

- 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 $err_D(h) > \varepsilon$] < 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., $Pr_D[h(x) \neq f(x)] < \varepsilon$.

Suppose we have an algo for a class of functions C s.t.:

- For any given $\varepsilon > 0$, $\delta > 0$, any target $f \in C$, any dist. D, the algorithm produces h of $err_D(h) < \varepsilon$ with prob. at least $1-\delta$.
- Running time $t\left(\frac{1}{\epsilon}, \frac{1}{\delta}, \ldots\right)$ and sample size $s\left(\frac{1}{\epsilon}, \frac{1}{\delta}, \ldots\right)$.

This is a PAC-learning algo with running time t and sample size s.
PAC-learnable $\equiv$ exists algo with t, s polynomial in relevant params.

- Learning is "proper" if $h \in C$. Can also talk of "learning C by H".

We just gave a proper 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(|H|).

If $|S| \geq \frac{1}{\epsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right)$ then with prob $\geq 1-\delta$, all $h \in H$ with $err_D(h) \geq \epsilon$ will have $err_S(h) > 0$.

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:

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

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

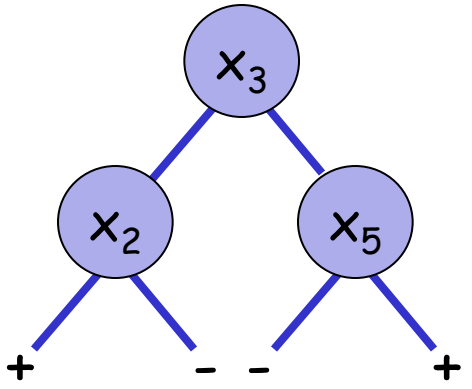
Then it's unlikely a bad simple explanation will fool you just by chance.

Occam's razor (contd)[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.
- And, doesn't say that complicated explanations are necessarily bad either.

Decision trees



- Decision trees over $\{0,1\}^n$ not known to be PAC-learnable (time, samples polynomial in size of f).
- 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.

More examples

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

- AND-functions, OR-functions
- 3-CNF formulas (3-SAT formulas), 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?

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

- Occam bounds $\Rightarrow$ generic way to learn any f from $O(size(f))$ samples if ignore computation time:
  - If we know $s = size(f)$ in advance, we can just draw $(1/\varepsilon)[s + \ln(1/\delta)]$ examples and search for a consistent rule of size at most s.
  - If we don't we can guess, check, and double our guess if it failed.

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

- Occam bounds $\Rightarrow$ generic way to learn any f from $O(size(f))$ samples if ignore computation time:
  - 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: $O((1/\varepsilon)[\text{size(f)}+\ln(1/\delta)\ln(\text{size(f)})])$.

1st terms sum to $O(size(f))$ by telescoping. 2nd terms sum to: $\ln\left(\frac{2}{\delta}\right) + \ln\left(\frac{4}{\delta}\right) + \ldots + \ln\left(\frac{size(f)}{\delta}\right) \leq \ln(size(f)) \ln\left(\frac{size(f)}{\delta}\right) = \ln^2(size(f)) + \ln(size(f)) \ln\left(\frac{1}{\delta}\right)$

More about the PAC model

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 $err(h)<\varepsilon$ with prob. at least $1-\delta$.
- Running time and sample sizes polynomial in relevant parameters: $1/\varepsilon$, $1/\delta$, n, $size(f)$.
- Require h to be poly-time evaluatable. Learning is called "proper" if $h \in C$. Can also talk about "learning C by H".

- 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(C) + ε.

More about the PAC model

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 $err(h)<\varepsilon$ with prob. at least $1-\delta$.
- Running time and sample sizes polynomial in relevant parameters: $1/\varepsilon$, $1/\delta$, n, $size(f)$.
- Require h to be poly-time evaluatable. Learning is called "proper" if $h \in C$. Can also talk about "learning C by H".

Drawbacks of model:

- In the real world, there may not be a perfect function in the class. But hard to prove guarantees for efficient algos on finding near-best function.
  - Convex "surrogate losses" (will get to later)
  - Iterative localization for learning under "reasonable" distrs [ABL17]
- In the real world, labeled examples may be more expensive than running time.

More about the PAC model

Algorithm PAC-learns a class of functions C if:

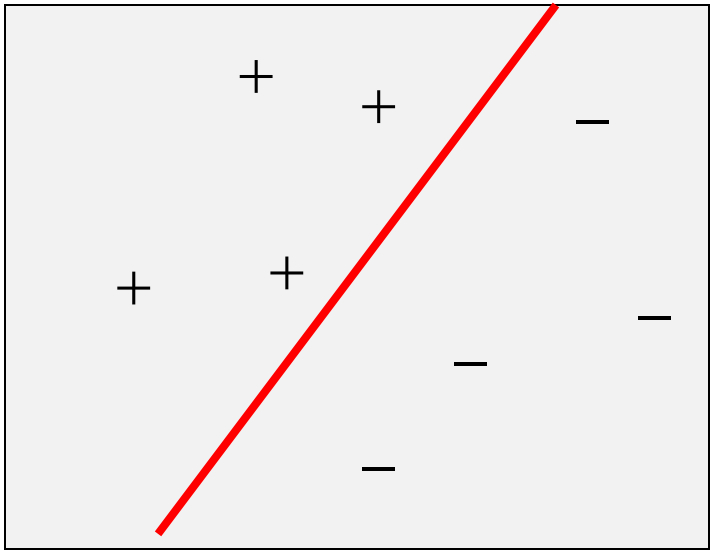
- For any given $\varepsilon>0$, $\delta>0$, any target $f \in C$, any dist. D, the algorithm produces h of $err(h)<\varepsilon$ with prob. at least $1-\delta$.
- Running time and sample sizes polynomial in relevant parameters: $1/\varepsilon$, $1/\delta$, n, $size(f)$.
- Require h to be poly-time evaluatable.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

Drawbacks of model:

- "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).
- Assumes fixed distribution

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.

Extensions we'll get at later:

- What if we don't want to assume data is iid?
- Models for continually learning
- Connections to game theory
- Settings where have limited feedback
- Combining learning with experimentation
- Issues like privacy

See you Wednesday!

Some References

- L.G. Valiant. "A theory of the learnable." *Proc. 16th ACM Symposium on Theory of Computing*. 1984.
- M. Kearns, M. Li, L. Pitt, and L. Valiant. "On the learnability of Boolean formulae." Proc 19th ACM Symposium on Theory of computing. 1987.
- R.L. Rivest. "Learning decision lists." Machine learning, vol 2, 1987.
- A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. "Occam's razor." Information processing letters, vol 24(6), 1987.
- R.E. Schapire, "The strength of weak learnability." Machine learning, vol 5, 1990.
- P. Awasthi, M.-F. Balcan, and P. M. Long. "The power of localization for efficiently learning linear separators with noise." Journal of the ACM vol 63.6, 2017.