CMU 15-451/651 lecture 11/12/13

An Algorithms-based Intro to Machine Learning, part I

Avrim Blum

[Based on portions of intro lectures in 15-859(B) Machine Learning Theory, and on a talk given at the National Academy of Sciences "Frontiers of Science" symposium.]

Plan for today

- Machine Learning intro: basic questions and issues & models.
- A formal analysis of "Occam's razor".
- Support-vector machines
- Perceptron algorithm

Machine learning can be used to...

- recognize speech,
- identify patterns in data,
- steer a car,
- play games,
- adapt programs to users,
- improve web search, ...

From a scientific perspective: can we develop models to understand learning as a computational problem, and what types of guarantees might we hope to achieve?

A typical setting

- Imagine you want a computer program to help filter 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
a positive example	N	Y	N	N	N	Y
	Y	N	N	N	Y	N
a negative example	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 (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]

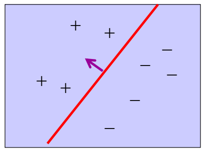
(B) What kind of confidence do we have that they will do well in the future?
[confidence bound / sample complexity]

for a given learning alg, how much data do we need, and how can we design alg to need less?

Algorithm design portion

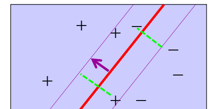
- How about this problem of learning a linear separator?
  - Want to solve for weight vector w such that $w \cdot x \geq 1$ for all positive x, and $w \cdot x \leq -1$ for all negative x.
- Any ideas?

Use linear programming!



Algorithm design portion

- How about this problem of learning a linear separator?
  - Want to solve for weight vector w such that $w \cdot x \geq 1$ for all positive x, and $w \cdot x \leq -1$ for all negative x.
- Any ideas?
- Additional issues: no perfect separator, margins.
- "Support Vector Machine":
  - $w \cdot x \geq 1 - \epsilon_i$ for positive ex i.
  - $w \cdot x \leq -1 + \epsilon_i$ for negative ex i.
  - $\epsilon_i \geq 0$. Minimize $\sum_i \epsilon_i + c|w|^2$



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

Natural formalization (PAC)

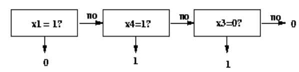
Email msg

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., $Pr_{x \sim D}[h(x) \neq f(x)] < \varepsilon$.

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.

(a bit of a toy example since would want to extend to "mostly consistent" DL)

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 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?

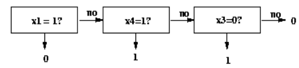
Confidence/sample-complexity

- Consider some DL h with $err(h) > \varepsilon$, that we're worried might fool us.
- Chance that h survives $|S|$ examples is at most $(1-\varepsilon)^{|S|}$.
- Let $|H|$ = number of DLs over n Boolean features. $|H| < (4n+2)!$. (really crude bound)

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

- This is <0.01 for $|S| > (1/\varepsilon)[\ln(|H|) + \ln(100)]$ or about $(1/\varepsilon)[n \ln n + \ln(100)]$

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 $err(h) > \varepsilon$] $< \delta$. DONE
3. So, if f is in fact a DL, then whp **A**'s hypothesis will be approximately correct. "PAC model"

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 $\log(|H|)$.

  (big difference between 100 and e^{100}.)

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.

$m > (1/\varepsilon)[s \ln(2) + \ln(100)]$

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

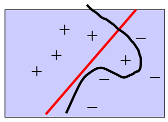
Occam's razor (contd)^2

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.

Regularization

- Very important notion in machine learning: basically a generalization of Occam's razor.



$Err_D(h) \;=\; Err_S(h) \;+\; [Err_D(h) - Err_S(h)]$

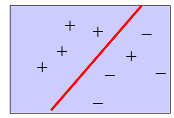
Minimize [error on training set] + [complexity term]

Typically hard to do exactly, so minimize an upper bound

"Regularizer": bounds the amount of overfitting.

Support-vector machines

- An instantiation of this for the case of linear separators in high dimensions.



- E.g., "bag of words", "bag of phrases"

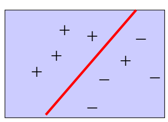
Minimize [error on training set] + [complexity term]

Typically hard to do exactly, so minimize an upper bound

"Regularizer": bounds the amount of overfitting.

Support-vector machines

- Issue #1: minimizing error on S is NP-hard. So, replace with upper bound: "hinge loss".



- Issue #2: what to use as complexity term?

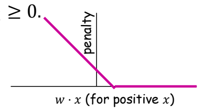
Minimize [error on training set] + [complexity term]

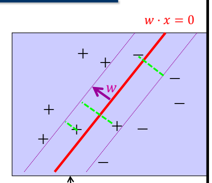
Typically hard to do exactly, so minimize an upper bound

"Regularizer": bounds the amount of overfitting.

Support-vector machines

- "Hinge loss": $\sum_i \epsilon_i$, where:
  - $w \cdot x_i \geq 1 - \epsilon_i$ for positive $x_i \in S$.
  - $w \cdot x_i \leq -1 + \epsilon_i$ for negative $x_i \in S$.
  - $\epsilon_i \geq 0$.





$1/|w|$ = "margin of separation"

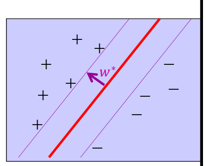
Minimize $\sum_i \epsilon_i \;+\; c \cdot |w|^2$

Typically hard to do exactly, so minimize an upper bound

Q: How to connect $|w|^2$ to the amount of overfitting?

Perceptron algorithm

- Suppose there exists a feasible soln w^* s.t. $|w^* \cdot x| \geq 1$ for all $x \in S$, where $||x|| \leq 1$ for all $x \in S$.
- The Perceptron algorithm is an online algorithm that will find a feasible w and make only $O(|w^*|^2)$ mistakes.



Perceptron algorithm:

- Start with weight vector $w = \vec{0}$.
- Mistake on positive x: let $w \leftarrow w + x$.
- Mistake on negative x: let $w \leftarrow w - x$.

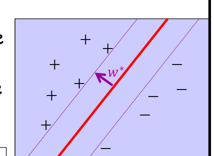
Because: $(w + x) \cdot w^* = w \cdot w^* + x \cdot w^* \geq w \cdot w^* + 1$.

Proof:
- After each update, $w \cdot w^*$ increases by ≥ 1.
- After each update, $w \cdot w$ increases by ≤ 3.

Because: $(w + x) \cdot (w + x) = w \cdot w + 2(w \cdot x) + x \cdot x \leq w \cdot w + 3$.

Perceptron algorithm

- Suppose there exists a feasible soln w^* s.t. $|w^* \cdot x| \geq 1$ for all $x \in S$, where $||x|| \leq 1$ for all $x \in S$.
- The Perceptron algorithm is an online algorithm that will find a feasible w and make only $O(|w^*|^2)$ mistakes.



Perceptron algorithm:

- Start with weight vector $w = \vec{0}$.
- Mistake on positive x: let $w \leftarrow w + x$.
- Mistake on negative x: let $w \leftarrow w - x$.

So: $M \leq 3|w^*|^2$

Proof:
- After each update, $w \cdot w^*$ increases by ≥ 1.
- After each update, $w \cdot w$ increases by ≤ 3.

$\Rightarrow$ After M mistakes: $M \leq |w \cdot w^*| \leq |w||w^*| \leq (3M)^{\frac{1}{2}}|w^*|$.

Perceptron algorithm

- Note: this doesn't prove why $|w|^2$ is a good thing to minimize in SVM optimization, but gives a feel for why the existence of such large margin separators means the world is "nice".

Some Courses

- 10-601 "Machine Learning"
  - Find out about a lot of different practical algorithms. Some of the theory. Implement algs and run them on data.
- 15-859(B) "Machine Learning Theory"
  - My course ☺
  - More focused on the kinds of guarantees you can prove. Algorithms as the answer to a question. Hwks more like 15-451.
- 10-701 "Machine Learning"
  - Mix of both. Serious commitment.