CMU 15-451/15-651

Lecturer: Avrim Blum 11/16/15

An Algorithms-based Intro to Machine Learning, part II

Plan for today

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

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?

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

Natural formalization (PAC)

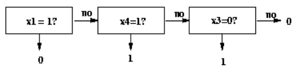
Email msg

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., $err_D(h) = \Pr_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$.

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_D(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?

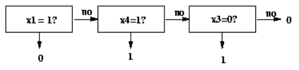
Confidence/sample-complexity

- Consider some DL h with err(h)>ε, 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)>ε is consistent] ≤ $|H|(1-\varepsilon)^{|S|}$.

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

Example of analysis: Decision Lists



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

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

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:

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

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

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

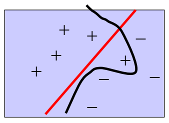
Occam's razor (contd)^2

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.

Regularization

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



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

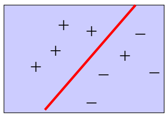
Minimize [error on training set] + [complexity term]

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"

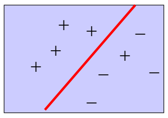
Minimize [error on training set] + [complexity term]

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?

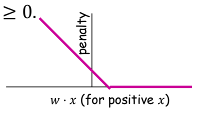
Minimize [error on training set] + [complexity term]

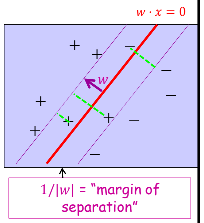
Typically hard to do exactly, so minimize an upper bound

"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 $x_i \in S$.
  - $\epsilon_i \geq 0$.





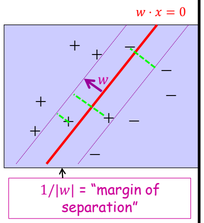
Minimize $\sum_i \epsilon_i$ + $c \cdot |w|^2$

Typically hard to do exactly, so minimize an upper bound

Turns out, can connect $|w|^2$ to the amount of overfitting.

Support-vector machines

- This is a convex optimization problem. (Not quite an LP, but solvable efficiently).
- Can approximate with Margin-Perceptron (update on mistake or "barely correct").
- Powerful tool in machine learning, both on its own and combined with kernel functions.



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