Plan for today

- Machine Learning intro: models and basic issues
- How much data do I need to see to be confident in generalizations I make from it?
- Connections of this to notion of Occam's razor
- A cool idea: "shatter coefficients", VC-dimension, and a very nice probabilistic argument.

Plan for Monday

- An interesting algorithm for online decision making. "Problem of "combining expert advice"
- Algorithms for online decision making from very limited feedback. The "multi-armed bandit problem"

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?
The concept learning setting

<table>
<thead>
<tr>
<th></th>
<th>money</th>
<th>pills</th>
<th>Mr.</th>
<th>bad spelling</th>
<th>known-sender</th>
<th>spam?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Pos</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Neg</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Neg</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

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?

[BIG QUESTION: algorithm design]

(B) What kind of confidence do we have that they will do well in the future?

[confidence bound / sample complexity]

Example of analysis: Decision Lists

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

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 sample S is of reasonable size, \( \Pr[\text{exists consistent DL } h \text{ with } \text{err}(h) < \varepsilon] < \delta \).
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)

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 $\text{err}(h) > \epsilon$, that we're worried might fool us.
- Chance that $h$ survives $|S|$ examples is at most $(1 - \epsilon)^{|S|}$.
- Let $|H| = \text{number of DLs over n Boolean features}$. $|H| < (4n+2)^{\epsilon}$. (really crude bound)
  
  So, $\text{Pr}[\text{some DL h with err(h) > \epsilon is consistent}] < |H|(1-\epsilon)^{|S|}$.
  
  This is $< 0.01$ for $|S| > (1/\epsilon)[\ln(|H|) + \ln(100)]$ or about $(1/\epsilon)[n \ln n + \ln(100)]$

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.
2. Show that if $|S|$ is of reasonable size, then $\text{Pr}[\text{exists consistent DL h with err(h) > \epsilon}] < \delta$.
3. So, if $f$ is in fact a DL, then $\text{whp A's hypothesis will be approximately correct.}$ "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 $\log(|H|)$. (the "log" is important here)

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: $m > (1/\epsilon)[s \ln(2) + \ln(100)]$
  
  Then it's unlikely a bad simple explanation will fool you just by chance.

Occam's razor (contd)

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.
We said: if \(|S| \geq (1/e)[\ln(|H|) + \ln(1/\delta)]\), then with probability \(\geq 1-\delta\), all \(h \in H\) with \(\text{err}_D(h) \geq \varepsilon\) have \(\text{err}_S(h) > 0\).

What if no perfect rule, and best we find is rule with error (say) 10% on training set? What can we say?

**Extensions**

- **Thm:** If \(|S| \geq (1/(2\varepsilon^2))[\ln(|H|) + \ln(2/\delta)]\), then with probability \(\geq 1-O(1/\delta)\), all \(h \in H\) have \(\text{err}_D(h) \geq \varepsilon\).

**Proof:** apply Hoeffding bounds.
- Chance of failure at most \(2|H|e^{-2|S|\varepsilon^2}\).
- Set \(\delta\) and solve.

**A cool idea: shatter coefficient**

- Let \(H[S]\) be the number of ways of splitting set \(S\) using functions in \(H\).
- Let \(H[m] = \max_{|S|=m} H[S]\).
- E.g., linear separators in \(\mathbb{R}^d\): \(H[m] = O(m^d)\).
- E.g., intervals on a line: \(H[m] = O(m^2)\).

**Thm:** if \(m \geq (2/\varepsilon)[\ln(2H[2m]) + \ln(1/\delta)]\), then with probability \(\geq 1-\delta\), all \(h \in H\) with \(\text{err}_D(h) \geq \varepsilon\) have \(\text{err}_S(h) > 0\).

**Note 1:** For linear separators in \(\mathbb{R}^d\), \(H[2m] = O(m^d)\), so bound is \(O(1/\varepsilon)[d \ln(1/\varepsilon) + \ln(1/\delta)]\).

**Note 2:** VC-dimension\((H) = \max \text{ value } m \text{ such that } H[m] = 2^m\).

Sauer's lemma: \(H[m] = O(m^{VCdim(H)})\).

**One more extension**

- What about something like the class \(H\) of linear separators? What is \(|H|\)?


- There are infinitely many linear separators, but not that many really different ones.
- Union bound is too weak.

**A cool idea: shatter coefficient**

- Let \(H[S]\) be the number of ways of splitting set \(S\) using functions in \(H\).
- Let \(H[m] = \max_{|S|=m} H[S]\).
- E.g., linear separators in \(\mathbb{R}^d\): \(H[m] = O(m^d)\).

**Thm:** if \(m \geq (2/\varepsilon)[\ln(2H[2m]) + \ln(1/\delta)]\), then with probability \(\geq 1-\delta\), all \(h \in H\) with \(\text{err}_D(h) \geq \varepsilon\) have \(\text{err}_S(h) > 0\).

**Proof of Thm:**
- Consider drawing 2 sets \(S, S'\) of \(m\) examples each.
- Let \(A\) be the event: exists \(h \in H\) with \(\text{err}_D(h) \geq \varepsilon\) and \(\text{err}_S(h)=0\).
- Let \(B\) be the event: exists \(h \in H\) with \(\text{err}_S(h) \geq \varepsilon/2\) and \(\text{err}_D(h)=0\).
- Claim 1: \(\Pr[A]/2 \leq \Pr[B]\) (because \(\Pr[B|A] \geq 1/2\))
- So, just need to show \(\Pr[B]\) is low.
A cool idea: shatter coefficient

Thm: if \( m \geq (2/e)[\log(2H(2m)) + \log(1/\delta)] \), then with probability \( \geq 1-\delta \), all \( h \in H \) with \( \text{err}_b(h) \geq \varepsilon \) have \( \text{err}_S(h) \geq 0 \).

Proof cont’d:
- Consider drawing 2 sets \( S, S' \) of \( m \) examples each.
- Let \( B \) be the event: exists \( h \in H \) with \( \text{err}_S(h) \geq \varepsilon/2 \) and \( \text{err}_S(h) \leq 0 \). Suffices to show \( \Pr[B] \) is low.
- Now, define \( T, T' \) as follows:
  - For \( i = 1 \) to \( m \), flip a fair coin:
    - If heads, put \( i \)th element of \( S \) into \( T \) and \( i \)th element of \( S' \) into \( T' \).
    - If tails, do it other way around.

A cool idea: shatter coefficient

Thm: if \( m \geq (2/e)[\log(2H(2m)) + \log(1/\delta)] \), then with probability \( \geq 1-\delta \), all \( h \in H \) with \( \text{err}_b(h) \geq \varepsilon \) have \( \text{err}_S(h) \geq 0 \).

Proof cont’d:
- Will show that for all \( S, S' \), \( \Pr_{\text{swap}}[C] \) is low.
- Let \( C \) be the event: exists \( h \in H \) with \( \text{err}_S(h) \geq \varepsilon/2 \) and \( \text{err}_S(h) \leq 0 \). Suffices to show \( \Pr[C] \) is low.
- Now, define \( T, T' \) as follows:
  - For \( i = 1 \) to \( m \), flip a fair coin:
    - If heads, put \( i \)th element of \( S \) into \( T \) and \( i \)th element of \( S' \) into \( T' \).
    - If tails, do it other way around.

Online learning

- What if we don’t want to make assumption that data is coming from some fixed distribution? Or any assumptions on data?
- Can no longer talk about past performance predicting future results.
- Can we do all??
- Idea: regret.
  - Show that our algorithm regret as well as best of these in hindsight.

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.,

```
<table>
<thead>
<tr>
<th>Expert</th>
<th>Expt 1</th>
<th>Expt 2</th>
<th>Expt 3</th>
<th>neighbor's dog</th>
<th>truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>down</td>
<td>up</td>
<td>up</td>
<td>up</td>
<td>down</td>
<td>up</td>
</tr>
<tr>
<td>down</td>
<td>up</td>
<td>down</td>
<td>up</td>
<td>down</td>
<td>down</td>
</tr>
</tbody>
</table>
```

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.

- Each mistake cuts # available by factor of 2.
- Note: this means ok for $n$ to be very large.

**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.

<table>
<thead>
<tr>
<th><strong>Weights</strong></th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predictions</strong></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Weights</strong></td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Predictions</strong></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Weights</strong></td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Predictions</strong></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Randomized Weighted Majority**

$2.4(m + \lg n)$ not so good if the best expert makes a mistake 20% of the time. Can we do better? Yes.

- Instead of taking majority vote, use weights as probabilities. (e.g., if 70% on up, 30% on down, then pick 70:30) Idea: smooth out the worst case.
- Also, generalize $\frac{1}{2}$ to 1 - $\varepsilon$.

**Additive regret**

- So, have $M \leq \OPT + \varepsilon \OPT + 1/e \log(n)$.
- Say we know we will play for $T$ time steps. Then can set $\varepsilon = (\log(n)/T)^{1/2}$. Get $M \leq \OPT + 2(T \log(n))^{1/2}$.
- If we don’t know $T$ in advance, can guess and double.
- These are called "additive regret" bounds.

$$M \leq -m \ln(1 - \varepsilon) + \ln(n) \approx (1 + s/2)m \cdot \frac{1}{\varepsilon} \log(n)$$
**Extensions**

- What if experts are actions? (rows in a matrix game, choice of deterministic alg to run,...)
- At each time $t$, each has a loss (cost) in $\{0,1\}$.
- Can still run the algorithm
  - Rather than viewing as "pick a prediction with prob proportional to its weight",
  - View as "pick an expert with probability proportional to its weight"
- Same analysis applies.

**Extensions**

- What if losses (costs) in $[0,1]$?
- Here is a simple way to extend the results.
  - Given cost vector $c$, view $c_i$ as bias of coin. Flip to create boolean vector $c'$, s.t. $E[c'] = c$. Feed $c'$ to alg $A$.
  - $E_c[A]\leq \min_i E[c'[i]] + \text{regret term}$
  - So, $E_c[A]\leq E_c[\min_i c'[i]] + \text{regret term}$
  - LHS is $E_c[c[A]]$.
  - RHS $\leq \min_i E[c'[i]] + \text{r.t.} = \min_i c[i] + \text{r.t.}$

In other words, costs between 0 and 1 just make the problem easier...

---

**What can we use this for?**

- Can use to combine multiple algorithms to do nearly as well as best in hindsight.
  - E.g., do nearly as well as best strategy in hindsight in repeated play of matrix game.
- Extension: "sleeping experts". E.g., one for each possible keyword. Try to do nearly as well as best "coalition".

---

**Online pricing**

- Say you are selling lemonade (or a cool new software tool, or bottles of water at the world expo).
- Protocol #1: for $t=1,2,...T$
  - Seller sets price $p^t$
  - Buyer arrives with valuation $v^t$
  - If $v^t \geq p^t$, buyer purchases and pays $p^t$, else doesn’t.
  - $v^t$ revealed to algorithm.
- Bad algorithm: "best price in past"
  - What if sequence of buyers is $1, h, I, ..., I, h, 1, ..., I, h, ...$
  - Alg makes $T/h$, OPT makes $T$. Factor of $h$ worse!

---

**Online pricing**

- Say you are selling lemonade (or a cool new software tool, or bottles of water at the world expo).
- Protocol #1: for $t=1,2,...T$
  - Seller sets price $p^t$
  - Buyer arrives with valuation $v^t$
  - If $v^t \geq p^t$, buyer purchases and pays $p^t$, else doesn’t.
  - $v^t$ revealed to algorithm.
- Bad algorithm: "best price in past"
  - What if sequence of buyers is $1, h, I, ..., I, h, ..., I, h, ...$
  - Alg makes $T/h$, OPT makes $T$. Factor of $h$ worse!
Online pricing
- Say you are selling lemonade (or a cool new software tool, or bottles of water at the world expo).
- What about Protocol #2? [just accept/reject decision]
  - Now we can’t run RWM directly since we don’t know how to penalize the experts!
  - Called the “adversarial multiarmed bandit problem”
  - How can we solve that?

Multi-armed bandit problem
Exponential Weights for Exploration and Exploitation (exp^3)
[Auer, Cesa-Bianchi, Freund, Schapire]

- “Membership query learning”: Algorithm can construct its own examples.
- “Semi-supervised learning”: use of labeled-unlabeled data in passive setting.

Extensions (of expert or bandit problem)
[KV] setting:
- Implicit set S of feasible points in R^n. (E.g., m=#edges, S={indicator vectors 01101010 for possible paths})
- Assume have oracle for offline problem: given vector c, find x ∈ S to minimize c·x. (E.g., shortest path algorithm)
- Use to solve online problem: on day i, must pick x_i ∈ S before c_i is given.
- \((c_1 x_1^* + c_2 x_2^*/T) \rightarrow \min_{x \in S} (c_1 x^* + c_2 T)/T.

[Z] setting:
- Assume S is convex.
- Allow c(x) to be a convex function over S.
- Assume given any y not in S, can algorithmically find nearest x ∈ S.

Conclusion (γ = ε):
E[Exp3] ≥ OPT/(1+γ) - O(γ^2 nh log n)

Almost as good as protocol 1!