Plan for today

- Machine Learning intro: models and basic issues
- An interesting algorithm for “combining expert advice”

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 to 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?

<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>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</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 $> 0$.
- ...

The concept learning setting

E.g., money pills Mr. bad spelling known-sender | spam?

<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>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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]

---

**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)**

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

---

**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 $\Pr[\text{exists consistent DL } h \text{ with } \text{err}(h) > \varepsilon] < \delta$.
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)

---

**How can we find a consistent DL?**

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

If $(x_1=0)$ then -, else
If $(x_2=1)$ then +, else
If $(x_3=1)$ 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 $\text{err}(h) > \varepsilon$, 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, $\Pr[\text{some DL } h \text{ with } \text{err}(h) > \varepsilon \text{ is consistent}] < |H|(1 - \varepsilon)^{|S|}$.
• 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.
2. Show that if $|S|$ is of reasonable size, then $\Pr[\exists \text{consistent DL } h \text{ with } \text{err}(h) > \varepsilon] < \delta$.
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: $m > (1/\varepsilon)[s \ln(2) + \ln(100)]$

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

Occam’s razor (contd)

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.
Further work

- Replace \( \log(|H|) \) with “effective number of degrees of freedom”.
- There are infinitely many linear separators, but not that many really different ones.
- Kernels, margins, more refined analyses....

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 all??

Idea: regulatory as well

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>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>up</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</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 \( \log(n) \) 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>weights</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>predictions</td>
<td>Y</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>weights</td>
<td>1</td>
<td>1</td>
<td>.5</td>
</tr>
<tr>
<td>predictions</td>
<td>Y</td>
<td>N</td>
<td>T</td>
</tr>
<tr>
<td>weights</td>
<td>1</td>
<td>.5</td>
<td>.5</td>
</tr>
</tbody>
</table>

Analysis: do nearly as well as best expert in hindsight

- \( M = \# \) mistakes we’ve made so far.
- \( m = \# \) mistakes best expert has made so far.
- \( W = \) total weight (starts at \( n \)).
- After each mistake, \( W \) drops by at least 25%. So, after \( M \) mistakes, \( W \) is at most \( n(3/4)^M \).
- Weight of best expert is \( (1/2)^m \). So,
\[
(1/2)^m \leq n(3/4)^M
\]
\[
(4/3)^M \leq n2^m
\]
\[
M \leq 2.4(m + \log(n))
\]

So, if \( m \) is small, then \( M \) is pretty small too.
**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 1/2 to 1 - e

Solves to: \( M \leq \frac{-m \ln(1 - e) + \ln(n)}{e} \approx (1 + e/2)m + \frac{1}{e} \ln(n) \)

\( M \leq 1.39m + 2 \ln n \quad \epsilon = 1/2 \)
\( M \leq 1.15m + 4 \ln n \quad \epsilon = 1/4 \)
\( M \leq 1.07m + 8 \ln n \quad \epsilon = 1/8 \)

**What can we use this for?**
- Can use for repeated play of matrix game:
  - Consider a matrix where all entries 0 or -1.
  - Rows are different experts. Start at each with weight 1.
  - Pick row with prob. proportional to weight and update as in RWM.
  - Analysis shows do nearly as well as best row in hindsight!
  - In fact, analysis applies for entries in [-1,0], not just {-1,0}.
  - In fact, gives a proof of the minimax theorem...

**Analysis**
- Say at time t we have fraction \( F_t \) of weight on experts that made mistake.
- So, we have probability \( F_t \) of making a mistake, and we remove an \( eF_t \) fraction of the total weight.
  - \( W_{\text{final}} = n(1 - eF_1)(1 - eF_2) \ldots \)
  - \( \ln(W_{\text{final}}) = \ln(n) + \sum_i \ln(1 - eF_i) \leq \ln(n) - e \sum_i F_i \)
  (using \( \ln(1 - x) < -x \))
  \( = \ln(n) - e M. \quad (\sum F_i = E[\text{# mistakes}]) \)
- If best expert makes \( m \) mistakes, then \( \ln(W_{\text{final}}) > \ln((1 - e)m) \).
- Now solve: \( \ln(n) - e M > m \ln(1 - e) \).

\( M \leq \frac{-m \ln(1 - e) + \ln(n)}{e} \approx (1 + e/2)m + \frac{1}{e} \ln(n) \)

**Nice proof of minimax thm (sketch)**
- Suppose for contradiction it was false.
- This means some game \( G \) has \( V_C > V_R \):
  - If Column player commits first, there exists a row that gets the Row player at least \( V_C \).
  - But if Row player has to commit first, the Column player can make him get only \( V_R \).
- Scale matrix so payoffs to row are in [-1,0]. Say \( V_R = V_C - \delta \).

**Proof sketch, contd**
- Now, consider randomized weighted-majority alg, against Col who plays optimally against Row’s distrib.
- In T steps,
  - \( \text{Alg gets } \geq (1-\epsilon/2) \text{[best row in hindsight]} - \log(n)/\epsilon \)
  - BRiH \( \geq T V_C \) [Best against opponent’s empirical distribution]
  - Alg \( \leq T V_R \) [Each time, opponent knows your randomized strategy]
  - Gap is \( \delta T \). Contradicts assumption if use \( \epsilon = \delta \), once \( T > 2 \log(n)/\epsilon^2 \).

**Other models**
- "Active learning": have large unlabeled sample and alg may choose among these.
  - E.g., web pages, image databases.
- Or, allow algorithm to construct its own examples. "Membership queries"
  - E.g., features represent variable-settings in some experiment, label represents outcome.
  - Gives algorithm more power.
Other models

• A lot of ongoing research into better algorithms, models that capture additional issues, incorporating Machine Learning into broader classes of applications.

Additional notes

• Some courses at CMU on machine learning:
  – 10-601 Machine Learning
  – Any 10-xxx course...

And finally...

• Final exam is Thurs 1pm DH 2210. 1 sheet of notes allowed.
• Review session next Wed 1-3pm in Wean 7500.
• Good luck everyone!