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

Given data, some reasonable rules might be:
- Predict SPAM if \( \neg \text{known AND (money OR pills)} \)
- Predict SPAM if money + pills - known > 0.
- ...

The concept learning setting

E.g.,

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

Given data, some reasonable rules might be:
- Predict SPAM if \( \neg \text{known AND (money OR pills)} \)
- 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...

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 $\Pr[\exists$ consistent DL $h$ with 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></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td></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>+</td>
</tr>
<tr>
<td></td>
<td>1</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 \( \text{err}(h) > \epsilon \), that we're worried might fool us.
• Chance that \( h \) survives \(|S|\) examples is at most \((1 - \epsilon)^{|S|}\).
• Let \(|H| = \text{number of DLs over } n \text{ Boolean features.} \quad |H| < (4n+2)!\) \((\text{really crude bound})\)

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

• This is \(< 0.01 \) for \(|S| > (1/\epsilon)[\ln(|H|) + \ln(100)]\) or about \((1/\epsilon)[h \ln n + \ln(100)]\).

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 \( \Pr[\text{exists consistent DL } h \text{ with } \text{err}(h) > \epsilon] < \delta \).
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 \( \log(|H|) \).
  \((\text{big difference between 100 and } e^{100})\)

Occam’s razor

William of Occam (~1320 AD):

“entities should not be multiplied unnecessarily” \((\text{in Latin})\)

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)²

Nice interpretation:

• Even if we have different notions of what’s simpler \((\text{e.g., different representation languages})\), 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 hope to say anything interesting at all??
  Idea: regret bounds.
  ➢ Show that our algorithm does nearly as well as best predictor in some large class.

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>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 \(\lg(n)\) 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>weights</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>prediction correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>predictions</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>weights</td>
<td>1</td>
<td>1</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>predictions</td>
<td>Y</td>
<td>Y</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>weights</td>
<td>1</td>
<td>.5</td>
<td>.5</td>
<td></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,
  \[
  (1/2)^m \leq n(3/4)^M \\
  (4/3)^M \leq n2^m \\
  M \leq 2.4(m + \lg n)
  \]
  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 ½ to 1 - ε.

Solves to:
\[ M \leq \frac{-m \ln(1-\epsilon) + \ln(n)}{\epsilon} \approx (1 + \epsilon/2)m + \frac{1}{\epsilon} \ln(n) \]
\[ M \leq 1.39m + 2 \ln n - \epsilon = 1/2 \]
\[ M \leq 1.15m + 4 \ln n - \epsilon = 1/4 \]
\[ M \leq 1.07m + 8 \ln n - \epsilon = 1/8 \]

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".

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 \( \epsilon F_t \) fraction of the total weight.
- \[ W_{\text{final}} = n(1 - \epsilon F_t)(1 - \epsilon F_{t-1})... \]
- \[ \ln(W_{\text{final}}) = \ln(n) + \sum [\ln(1 - \epsilon F_i)] \leq \ln(n) - \epsilon \sum F_i \]
  (using \( \ln(1-x) < -x \))
- \[ \ln(W_{\text{final}}) = \ln(n) - \epsilon M. \]
  (\( \sum F_i = E[\# \text{ mistakes}] \))
- If best expert makes \( m \) mistakes, then \( \ln(W_{\text{final}}) > \ln((1-\epsilon)^m) \)
- Now solve: \( \ln(n) - \epsilon M > m \ln(1-\epsilon). \)

\[ M \leq \frac{-m \ln(1-\epsilon) + \ln(n)}{\epsilon} \approx (1 + \epsilon/2)m + \frac{1}{\epsilon} \log(n) \]

Other models

Some scenarios allow more options for algorithm.

- "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
  - 10-701/15-781 Machine Learning