Learning from Observations

Chapter 18, Sections 1–3
Learning

Learning is essential for unknown environments, i.e., when designer lacks omniscience.

Learning is useful as a system construction method, i.e., expose the agent to reality rather than trying to write it down.

Learning modifies the agent’s decision mechanisms to improve performance.
Inductive learning (a.k.a. Science)

Simplest form: learn a function from examples (tabula rasa)

\( f \) is the target function

An example is a pair \( x, f(x) \), e.g.,

\[
\begin{array}{c|c|c}
O & O & X \\
X & & \\
X & & \\
\end{array}
\] , +1

Problem: find a(n) hypothesis \( h \)
such that \( h \approx f \)
given a training set of examples

(This is a highly simplified model of real learning:
- Ignores prior knowledge
- Assumes a deterministic, observable “environment”
- Assumes examples are given
- Assumes that the agent wants to learn \( f \)—why?)
Inductive learning method

Construct/adjust $h$ to agree with $f$ on training set
($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:

\[ f(x) \]

\[ x \]
Inductive learning method

Construct/adjust $h$ to agree with $f$ on training set
($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:
Inductive learning method

Construct/adjust $h$ to agree with $f$ on training set  
($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:

\[ f(x) \]
Inductive learning method

Construct/adjust $h$ to agree with $f$ on training set
($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:
Inductive learning method

Construct/adjust $h$ to agree with $f$ on training set
($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:

![Diagram of curve fitting](image)
Inductive learning method

Construct/adjust $h$ to agree with $f$ on training set
($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:

Ockham’s razor: maximize a combination of consistency and simplicity
Attribute-based representations

Examples described by attribute values (Boolean, discrete, continuous, etc.)
E.g., situations where I will/won’t wait for a table:

<table>
<thead>
<tr>
<th>Example</th>
<th>Attributes</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alt Bar Fri Hun Pat Price Rain Res Type Est</td>
<td>WillWait</td>
</tr>
<tr>
<td>$X_1$</td>
<td>T F F T Some $$$ F T French 0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_2$</td>
<td>T F F T Full $ F F Thai 30–60</td>
<td>F</td>
</tr>
<tr>
<td>$X_3$</td>
<td>F T F F Some $ F F Burger 0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_4$</td>
<td>T F T T Full $ F F Thai 10–30</td>
<td>T</td>
</tr>
<tr>
<td>$X_5$</td>
<td>T F F T Full $$$ F T French &gt;60</td>
<td>F</td>
</tr>
<tr>
<td>$X_6$</td>
<td>F T F F Some $ T T Italian 0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_7$</td>
<td>F T F F None $ T F Burger 0–10</td>
<td>F</td>
</tr>
<tr>
<td>$X_8$</td>
<td>F F F T Some $ T T Thai 0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_9$</td>
<td>F T T F Full $ T F Burger &gt;60</td>
<td>F</td>
</tr>
<tr>
<td>$X_{10}$</td>
<td>T T T T Full $$$ F T Italian 10–30</td>
<td>F</td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>F F F F None $ F F Thai 0–10</td>
<td>F</td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>T T T T Full $ F F Burger 30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

Classification of examples is positive (T) or negative (F)
Decision trees

One possible representation for hypotheses
E.g., here is the “true” tree for deciding whether to wait:
Expressiveness

Decision trees can express any function of the input attributes. E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A xor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless $f$ nondeterministic in $x$) but it probably won’t generalize to new examples

Prefer to find more **compact** decision trees
Hypothesis spaces

How many distinct decision trees with $n$ Boolean attributes??
How many distinct decision trees with $n$ Boolean attributes??

$= \text{number of Boolean functions}$
Hypothesis spaces

How many distinct decision trees with $n$ Boolean attributes?

= number of Boolean functions
= number of distinct truth tables with $2^n$ rows
Hypothesis spaces

How many distinct decision trees with $n$ Boolean attributes?

= number of Boolean functions
= number of distinct truth tables with $2^n$ rows = $2^{2^n}$
Hypothesis spaces

How many distinct decision trees with $n$ Boolean attributes??

= number of Boolean functions
= number of distinct truth tables with $2^n$ rows = $2^{2n}$

E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
Hypothesis spaces

How many distinct decision trees with $n$ Boolean attributes??

= number of Boolean functions
= number of distinct truth tables with $2^n$ rows = $2^{2^n}$

E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., $Hungry \land \neg Rain$)??
How many distinct decision trees with \( n \) Boolean attributes??

\[
\text{number of Boolean functions} = \text{number of distinct truth tables with } 2^n \text{ rows} = 2^{2^n}
\]

E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., Hungry \( \land \neg \text{Rain} \))??

Each attribute can be in (positive), in (negative), or out

\[
3^n \text{ distinct conjunctive hypotheses}
\]

More expressive hypothesis space

- increases chance that target function can be expressed
- increases number of hypotheses consistent w/ training set

\[
\Rightarrow \text{may get worse predictions}
\]
Decision tree learning

Aim: find a small tree consistent with the training examples

Idea: (recursively) choose “most significant” attribute as root of (sub)tree

```python
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return mode(examples)
    else
        best ← Choose-Attribute(attributes, examples)
        tree ← a new decision tree with root test best
        for each value \( v_i \) of best do
            examples_i ← \{ elements of examples with best = \( v_i \} \)
            subtree ← DTL(examples_i, attributes − best, mode(examples))
            add a branch to tree with label \( v_i \) and subtree subtree
        return tree
```
Choosing an attribute

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”

Patrons? is a better choice—gives information about the classification
Information

Information answers questions

The more clueless I am about the answer initially, the more information is contained in the answer

Scale: 1 bit = answer to Boolean question with prior \(\langle 0.5, 0.5 \rangle\)

Information in an answer when prior is \(\langle P_1, \ldots, P_n \rangle\) is

\[
H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i
\]

(also called entropy of the prior)

EXERCISE: What is the information for \(\langle 1, 0 \rangle\), and \(\langle 0.5, 0.5 \rangle\)?
Information contd.

Suppose we have \( p \) positive and \( n \) negative examples at the root
\[
\Rightarrow H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) \text{ bits needed to classify a new example}
\]
E.g., for 12 restaurant examples, \( p = n = 6 \) so we need 1 bit

An attribute splits the examples \( E \) into subsets \( E_i \), each of which (we hope) needs less information to complete the classification

Let \( E_i \) have \( p_i \) positive and \( n_i \) negative examples
\[
\Rightarrow H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right) \text{ bits needed to classify a new example}
\]
\[
\Rightarrow \text{expected number of bits per example over all branches is}
\]
\[
\sum_i \frac{p_i + n_i}{p + n} H\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)
\]

For \( Patrons? \), this is 0.459 bits, for \( Type \) this is (still) 1 bit

\[
\Rightarrow \text{choose the attribute that minimizes the remaining information needed}
\]

EXERCISE: Compute information gain for "BAR" in restaurant example.
Decision tree learned from the 12 examples:

Substantially simpler than “true” tree—a more complex hypothesis isn’t justified by small amount of data
Performance measurement

How do we know that $h \approx f$? (Hume’s Problem of Induction)

1) Use theorems of computational/statistical learning theory

2) Try $h$ on a new test set of examples
   (use same distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size
Learning curve depends on
- realizable (can express target function) vs. non-realizable
  non-realizability can be due to missing attributes
  or restricted hypothesis class (e.g., thresholded linear function)
- redundant expressiveness (e.g., loads of irrelevant attributes)
Noise and Overfitting.

Overfitting - fitting the training set too well

Noise - Bad data values

Decision tree pruning - prevent recursive splits unless a “significant enough” attribute is found.

Cross validation - allows you to assess how likely it is that the model (hypothesis) is overfitting.
Decision Tree Extensions.

Missing Data - may require setting default value

Multivalued Attributes - may require updating info. gain

Numerical Valued Attributes - split and allow to be used more than once

Continuous Valued Output
Ensemble Learning

Combining multiple learning algorithms in some fashion.

Example:
Boosting

Use weighted training set

Start all weights as 1, but then increase weights of examples that are misclassified for next classifier.

Finally, weight the classifiers (decision trees) based on how well they classify training set.

Can work well even if the input algorithm is a weak learning algorithm (it’s classification results are only slightly better than random..)
PAC-learning:
  - Probably Approximately Correct
PAC Learning (definitions)

Error of a classifier:
\[
error(h) = P(h(x) \neq f(x) | x \text{ drawn from } D)
\]

Approximately correct:
\[
error(h) < \epsilon
\]

Prob. that classifier consistent with \(N\) samples is in \(H_{\text{bad}}\):
\[
P(h \in H_{\text{bad}}) \leq (1 - \epsilon)^N
\]

Prob. that \(H_{\text{bad}}\) contains a consistent hypothesis for \(N\) samples:
\[
P(H_{\text{bad}} \text{ has consistent hypothesis}) \leq |H_{\text{bad}}|(1 - \epsilon)^N \leq |H|(1 - \epsilon)^N
\]

If we want to determine \(N\) that will make a classifier have an error less than \(\epsilon\) with probability \(1 - \delta\), we get:
\[
N \geq \frac{1}{\epsilon} \left( \ln \frac{1}{\delta} + \ln |H| \right)
\]
PAC Learning (example)

Boolean function with $n$ attributes:

$$|H| = 2^{2^n}$$

Exceedingly large, so PAC learning can’t inform guide us well about general boolean functions.

PAC may do better with simplified representations. Example:

- Decision list - a (limited) disjunction of conjunctions

EXERCISE: Review derivation about decision lists.