Learning from Observations

Chapter 18
Section 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

Learning element

• Design of a learning element is affected by
  – Which components of the performance element are to
    be learned
  – What feedback is available to learn these
    components
  – What representation is used for the components
• Type of feedback:
  – Supervised learning: correct answers for each
    example
  – Unsupervised learning: correct answers not given
  – Reinforcement learning: occasional rewards

Inductive learning

• Simplest form: learn a function from examples
  – $f$ is the target function
  – An example is a pair $(x, f(x))$
  – Problem: find a hypothesis $h$
    such that $h = f$ 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)

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:
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: prefer the simplest hypothesis consistent with data

Expressiveness

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

$$\begin{array}{c|c|c|c}
A & B & A \lor B \\
F & F & F \\
F & T & T \\
T & F & T \\
T & T & F \\
\end{array}$$

• Trivially, there is a consistent decision tree for any training set with one path to leaf for each example 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?

= 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
Hypothesis spaces

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 16,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., Hungry \( \land \neg \text{Rain} \))?
- Each attribute can be in (positive), in (negative), or out
  \( \Rightarrow 3^n \text{ distinct conjunctive hypotheses} \)
- More expressive hypothesis space
  - increases chance that target function can be expressed
  - increases number of hypotheses consistent with training set

Entropy

- "Measure of uncertainty"
- "Expected number of bits to resolve uncertainty"
  - Suppose \( \Pr(X = 0) = \frac{1}{8} \)
    - If other events are equally likely, the number of events is 8. To indicate one out of so many events, one needs \( \log_2 8 \) bits.
  - Consider a binary random variable \( X \) s.t. \( \Pr(X = 0) = 0.1 \).
    - The expected number of bits: \( 0.1 \log_2 \left( \frac{1}{0.1} \right) + 0.9 \log_2 \left( \frac{1}{0.9} \right) \)
  - In general, if a random variable \( X \) has \( c \) values with prob. \( p_1, p_2, \ldots, p_c \):
    - The expected number of bits: \( H = \sum_{i=1}^{c} p_i \log_2 \left( \frac{1}{p_i} \right) \)

Entropy of a binary variable

\[
H(p) = -p \log_2 p - (1-p) \log_2 (1-p)
\]

Divide and Conquer

- Internal decision nodes
  - Univariate: Uses a single attribute, \( x_i \)
    - Numeric \( x_i \): Binary split: \( x_i > w \)
    - Discrete \( x_i \): \( n \)-way split for \( n \) possible values
  - Multivariate: Uses all attributes, \( x \)
- Leaves
  - Classification: Class labels, or proportions
  - Regression: Numeric; \( r \) average, or local fit

The learning algorithm is greedy; find the best split recursively

Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, d=0) returns a decision tree
if examples is empty then return default
then if all examples have the same classification then return the classification
else if attributes is empty then return root node
Best = CHOOSE-ATTRIBUTE(attributes, examples)
root = new decision tree with root test Best
for each value v of Best do
examples_v = elements of examples with Best = v
subtree = DTL(examples_v, attributes - Best, d+1)
add a branch to root with label v, and subtree subtree
return root
```

Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:
1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. FriSat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)
Attribute-based representations

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Server</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Room</td>
<td>T</td>
<td>T</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:

```
Patrons?
/   \
\   /
\  /  
\ /   
\ /    
\   /   
\  /    
\ /     
\ /      
/        
Yes      No
```

Decision trees

- Another possible representation for hypotheses

```
Patrons?
/   \
\   /
\  /  
\ /   
/     
Yes    No
```

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

Using information theory

- To implement Choose-Attribute in the DTL algorithm
- Information Content (Entropy):
  \[
  E(P(v), \ldots, P(v)) = \sum_{i=1}^{v} -P(v_i) \log_2 P(v_i)
  \]
- For a training set containing \( p \) positive examples and \( n \) negative examples:
  \[
  E\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = \frac{p}{p+n} \log_2 \frac{p}{p+n} + \frac{n}{p+n} \log_2 \frac{n}{p+n}
  \]

Information gain

- A chosen attribute \( A \) divides the training set \( E \) into subsets \( E_i, \ldots, E \) according to their values for \( A \), where \( A \) has \( v \) distinct values.
  \[
  \text{remainder}(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} E\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)
  \]
- Information Gain (IG) or reduction in entropy from the attribute test:
  \[
  IG(A) = E\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{remainder}(A)
  \]
- Choose the attribute with the largest IG
Computing Entropy and IG

\[ E \left( \frac{p}{p+n} \right) - \log \left( \frac{p}{p+n} \right) - \log \left( \frac{n}{p+n} \right) \]

\[ IG(A) = E \left( \frac{p}{p+n} \right) - \text{remainder} \left( A \right) \]

Example:

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
<th>Type</th>
<th>Patrons</th>
<th>Type</th>
<th>Full</th>
<th>Type</th>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>E2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>E3</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>E4</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>E5</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>E6</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Information gain

For the training set, \( p = n = 6 \), \( E(6/12, 6/12) = 1 \) bit

\[ IG(\text{Patrons}) = 1 - \frac{2}{12} E(1, 1) = \frac{4}{12} \] bits

\[ IG(\text{Type}) = 1 - \frac{2}{12} E(1, 1) = \frac{2}{12} \] bits

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root.

Example contd.

- Decision tree learned from the 12 examples:

- Substantially simpler than the first tree—a more complex hypothesis isn’t justified by small amount of data

Performance measurement

- How do we know that \( h \approx f \)?
  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

Why Learning Works

- There is a theoretic foundation: Computational Learning Theory.
- The underlying principle: Any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: it must be Probably Approximately Correct (PAC).
- The Stationarity Assumption: The training and test sets are drawn randomly from the same population of examples using the same probability distribution.

Training Examples

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play/Tennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Over fitting in Decision Trees

- Why “over”-fitting?
  - A model can become more complex than the true target function (concept) when it tries to satisfy noisy data as well.

- Definition of overfitting
  - A hypothesis is said to overfit the training data if there exists some other hypothesis that has larger error over the training data but smaller error over the entire instances.

Avoiding over-fitting the data

- How can we avoid overfitting? There are 2 approaches:
  - stop growing the tree before it perfectly classifies the training data
  - grow full tree, then post-prune
    - Reduced error pruning
    - Rule post-pruning
    - The 2nd approach is found more useful in practice.

- Ok, but how to determine the optimal size of a tree?
  - Use validation examples to evaluate the effect of pruning (stopping)
  - Use a statistical test to estimate the effect of pruning (stopping)
  - ...
Rule post-pruning

• Algorithm
  – Build a complete decision tree.
  – Convert the tree to set of rules.
  – Prune each rule:
    – Remove any preconditions if no impacts to accuracy
    – Sort the pruned rules by accuracy and use them in that order.

• This is the most frequently used method

Rule Extraction from Trees

• IF (Outlook = Sunny) ^ (Humidity = High)
  THEN PlayTennis = No

• IF (Outlook = Sunny) ^ (Humidity = Normal)
  THEN PlayTennis = Yes

• . . .

Handling training examples with missing attribute values

• What if an example x is missing the value an attribute A?

• Simple solution:
  – Use the most common value among examples at node n.
  – Or use the most common value among examples at node n that have classification c(x).

• More complex, probabilistic approach
  – Assign a probability to each of the possible values of A based on the observed frequencies of the various values of A
  – Then, propagate examples down the tree with these probabilities.
  – The same probabilities can be used in classification of new instances

Handling attributes with differing costs

• Sometimes, some attribute values are more expensive or difficult to prepare.
  – medical diagnosis, BloodTest has cost $150

• In practice, it may be desired to postpone acquisition of such attribute values until they become necessary.

• To this purpose, one may modify the attribute selection measure to penalize expensive attributes.
  – Tan and Schlimmer (1990)
  \[
  Gain'(C, A) = \frac{Gain(C, A)}{Cost(A)} = \frac{2^{\log_2(\frac{1}{\text{Cost}(A)})} - 1}{\text{Cost}(A) - 1}, \quad \text{with} \quad [0, 1]
  \]

  – Nunez (1988)

Summary

• Learning needed for unknown environments, lazy designers
• Learning agent = performance element + learning element
• For supervised learning, the aim is to find a simple hypothesis approximately consistent with training examples
• Decision tree learning using information gain
• Learning performance = prediction accuracy measured on test set
Basic Procedures

1. Collect randomly a large set of examples
2. Choose randomly a subset of the examples as **training set**
3. Apply the learning algorithm to the training set, generating a hypothesis $h$.
4. Measure the percentage of examples in the whole set that are correctly classified by $h$.
5. Repeat steps 1-4 for different sizes of training sets if the performance is not satisfactory.

Learning agents