Artificial Intelligence Programming
Learning - Decision Trees

Chris Brooks

Department of Computer Science
University of San Francisco
What does it mean for an agent to learn?
16-1: Learning

- What does it mean for an agent to learn?
- Agent acquires new knowledge
- Agent changes its behavior
- Agent improves its performance measure on a given task
Recall previously we talked about *learning agents*.
A learning agent has a performance element and a learning element.

- The performance element is what an agent uses to decide what to do.
- This is what we’ve studied up to now.

The learning element is what allows the agent to modify the performance element.

- This might mean adding or changing rules or facts, modifying a heuristic, changing a successor function
- In order to modify its behavior, an agent needs information telling it how well it is performing.
- This information is called feedback.
There are essentially three categories of learning tasks, each of which provides different feedback.

**Supervised learning.**
- In this case, an external source (often called a teacher) provides the agent with *labeled examples*.
- Agent sees specific actions/cases, along with their classification.
16-5: Types of Feedback

6 Unsupervised Learning
   △ In this case, there is no teacher to provide examples.
   △ The agent typically tries to find patterns in data.

6 Reinforcement Learning
   △ This is a particular version of learning in which the agent only receives a *reward* for taking an action.
   △ May not know how optimal a reward is.
   △ Will not know the “best” action to take
16-6: Supervised Learning

Supervised learning is one of the most common forms of learning.

Agent is presented with a set of labeled data and must use this data to determine more general rules.

Examples:
- List of patients and characteristics: what factors are correlated with cancer?
- What factors make someone a credit risk?
- What are the best questions for classifying animals?
- Whose face is in this picture?

This process of learning general rules from specific facts is called **induction**.
We can phrase the learning problem as that of estimating a function $f$ that tells us how to classify a set of inputs.

An example is a set of inputs $x$ and the corresponding $f(x)$

- $\langle \text{Mammal}, \text{Eats - Meat}, \text{Black - Stripes}, \text{Tawny} \rangle, \text{Tiger} \rangle$

We can define the learning task as follows:

- Given a collection of examples of $f$, find a function $H$ that approximates $f$ for our examples.
- $H$ is called a hypothesis.
We would like $H$ to generalize

- This means that $H$ will correctly classify unseen examples.

If the hypothesis can correctly classify all of the training examples, we call it a *consistent* hypothesis.

Goal: find a consistent hypothesis that also performs well on unseen examples.

We can think of learning as search through a space of hypotheses.
Notice that induction is not sound.
In picking a hypothesis, we make an educated guess.
the way in which we make this guess is called a bias.
Examples:
- Occam’s razor
- Most specific hypothesis.
- Most general hypothesis.
- Linear function
Agents may have different means of observing examples of a hypothesis.

A batch learning algorithm is presented with a large set of data all at once and selects a single hypothesis.

An incremental learning algorithm receives examples one at a time and continually modifies its hypothesis.
- Batch is typically more accurate, but incremental may fit better with the agent’s environment.

An active learning agent is able to choose examples.

A passive learning agent has examples presented to it by an outside source.
- Active learning is more powerful, but may not fit with the constraints of the domain.
Decision trees are data structures that provide an agent with a means of classifying examples.

At each node in the tree, an attribute is tested.
R & N show a decision tree for determining whether to wait at a busy restaurant.

The problem has the following inputs/attributes:

- Alternative nearby
- Has a bar
- Day of week
- Hungriness
- Crowd
- Price
- Raining?
- Reservation
- Type of restaurant
- Wait estimate

Note that not all attributes are used.
### 16-13: An example training set

<table>
<thead>
<tr>
<th>Example</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alt</td>
</tr>
<tr>
<td>$X_1$</td>
<td>T</td>
</tr>
<tr>
<td>$X_2$</td>
<td>T</td>
</tr>
<tr>
<td>$X_3$</td>
<td>F</td>
</tr>
<tr>
<td>$X_4$</td>
<td>T</td>
</tr>
<tr>
<td>$X_5$</td>
<td>T</td>
</tr>
<tr>
<td>$X_6$</td>
<td>F</td>
</tr>
<tr>
<td>$X_7$</td>
<td>F</td>
</tr>
<tr>
<td>$X_8$</td>
<td>F</td>
</tr>
<tr>
<td>$X_9$</td>
<td>F</td>
</tr>
<tr>
<td>$X_{10}$</td>
<td>T</td>
</tr>
<tr>
<td>$X_{11t}$</td>
<td>F</td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>T</td>
</tr>
</tbody>
</table>
An example is a specific set of attributes, along with a classification (WillWait or not)

Examples where WillWait is true are called positive examples

Examples where WillWait is false are called negative examples

The set of labeled examples is called the training set.

We want to have a tree that:
- Classifies the training set correctly
- Accurately predicts unseen examples
- is as small as possible (Occam’s razor)

What if we construct a tree with one leaf for each example?
Choosing useful attributes

Intuitively, we would like to test attributes that 'split' the training set.

Splitting on restaurant type is not useful - positive and negative examples are still clustered together.

Splitting on crowdedness is more effective.
We can construct a decision tree recursively:

1. If all examples are positive or negative, we are done.
2. If there are no examples left, then we haven’t seen an instance of this classification, so we use the majority classification of the parent.
3. If there are no attributes left to test, then we have instances with the same description, but different classifications.
   - Insufficient description
   - Noisy data, nondeterministic domain
   - Use majority vote
4. Else, pick the best attribute to split on and recursively construct subtrees.
16-17: An example tree

The hand-constructed tree.

The induced tree. Notice that it's simpler and discovers a relationship between Thai food and waiting.
The key to constructing a compact and efficient decision tree is to effectively choose attributes to test.

Intuition: we want to choose tests that will separate our data set into positive and negative examples.

We want to measure the amount of information provided by a test.

This is a mathematical concept that characterizes the number of bits needed to answer a question or provide a fact.
Example: let’s say I’m trying to predict the flip of a coin.

- Before the flip, one bit (heads/tails) is enough to completely answer the question.
- After I’ve seen the coin, additional bits contain no information, since they don’t provide new knowledge.
More formally, let’s say there are \( n \) possible answers \( v_1, v_2, \ldots, v_n \) to a question, and each answer has probability \( P(v_n) \) of occurring.

The information content of the answer to the question is:

\[
I = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i)
\]

For a coin, this is:

\[
-\frac{1}{2} \log_2 \frac{1}{2} + -\frac{1}{2} \log_2 \frac{1}{2} = 1.
\]

Questions with one highly likely answer will have low information content. (if the coin comes up heads 99/100 of the time, \( I = 0.08 \))

Information content is also sometimes called entropy.

This is often used in compression and data transfer algorithms.
For decision trees, we want to know how valuable each possible test is, or how much information it yields.

We can estimate the probabilities of possible answers from the training set.

Usually, a single test will not be enough to completely separate positive and negative examples.

Instead, we need to think about how much better we’ll be after asking a question.

This is called the information gain.
If a training set has $p$ positive examples and $n$ negative examples, the information in a correct answer is:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

We want to find the attribute that will come the closest to separating the positive and negative examples.

We begin by computing the remainder - this is the information still in the data after we test attribute $A$.

Say attribute $A$ can take on $v$ possible values. This test will create $v$ new subsets of data, labeled $E_1, E_2, \ldots, E_v$.

The remainder is the sum of the information in each of these subsets.

$$\text{Remainder}(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \times I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$
Information gain can then be quantified as the difference between the original information (before the test) and the new information (after the test).

\[ \text{Gain}(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Remainder}(A) \]

Heuristic: Always choose the attribute with the largest information gain.

Question: What kind of search is this?
<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</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>Weak</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>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
Which attribute is the best classifier?

\[
\begin{align*}
S: & [9+,5-] \\
\quad E & = 0.940 \\
\quad & \\
\text{Humidity} & \\
\quad \text{High} & [3+,4-] \\
\quad \quad E & = 0.985 \\
\quad \text{Normal} & [6+,1-] \\
\quad \quad E & = 0.592 \\
\text{Wind} & \\
\quad \text{Weak} & [6+,2-] \\
\quad \quad E & = 0.811 \\
\quad \text{Strong} & [3+,3-] \\
\quad \quad E & = 1.00 \\
\end{align*}
\]

\[
\begin{align*}
\text{Gain} (S, \text{ Humidity }) &= .940 - (7/14).985 - (7/14).592 \\
&= .151 \\
\text{Gain} (S, \text{ Wind }) &= .940 - (8/14).811 - (6/14)1.0 \\
&= .048
\end{align*}
\]
If there are two examples that have the same attributes but different values, a decision tree will be unable to classify them separately.

We say that this data is *noisy*.

Solution: Use majority rule at the parent node.
A common problem is learning algorithms occurs when there are random or anomalous patterns in the data. For example, in the tennis problem, by chance it might turn out that we always play on Tuesdays.

The phenomenon of learning quirks in the data is called overfitting.

In decision trees, overfitting is dealt with through pruning.

Once the tree is generated, we evaluate the significance of each node.
Assume that the test provides no information. (null hypothesis)

Does the data in the children look significantly different from this assumption?

Use a chi-square test.

If not, the node is removed and examples moved up to the parent.
Decision trees can also be extended to work with integer and continuous-valued inputs.

Discretize the range into, for example, \(< 70\) and \(> 70\).

Challenge: What value yields the highest information gain?

Use a hill-climbing search to find this value.