Goal: Learn the function “PlayTennis?” from example data

<table>
<thead>
<tr>
<th>Input Attributes</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>Outlook</td>
</tr>
<tr>
<td>d1</td>
<td>s</td>
</tr>
<tr>
<td>d2</td>
<td>s</td>
</tr>
<tr>
<td>d3</td>
<td>o</td>
</tr>
<tr>
<td>d4</td>
<td>r</td>
</tr>
<tr>
<td>d5</td>
<td>r</td>
</tr>
<tr>
<td>d6</td>
<td>r</td>
</tr>
<tr>
<td>d7</td>
<td>o</td>
</tr>
<tr>
<td>d8</td>
<td>s</td>
</tr>
<tr>
<td>d9</td>
<td>s</td>
</tr>
<tr>
<td>d10</td>
<td>r</td>
</tr>
<tr>
<td>d11</td>
<td>s</td>
</tr>
<tr>
<td>d12</td>
<td>o</td>
</tr>
<tr>
<td>d13</td>
<td>o</td>
</tr>
<tr>
<td>d14</td>
<td>r</td>
</tr>
</tbody>
</table>

- **Outlook** = sunny, overcast, or rain
- **Humidity** = high, or normal
- **Wind** = weak or strong
A Decision Tree for the Same Data

Decision Tree for “PlayTennis?”

Leaves = classification output
Arcs = choice of value for parent attribute

Decision tree equivalent to logical statement in disjunctive normal form

\[ \text{PlayTennis} \iff (\text{Sunny} \land \text{Normal}) \lor \text{Overcast} \lor (\text{Rain} \land \text{Weak}) \]

Decision Trees

- **Input**: Set of attributes describing an object or situation
- **Output**: Predicted output value for the input
- Decision tree is consistent if it produces the correct output on all training examples
- Input and output can be discrete or continuous
Example: Decision Tree for Continuous Values

Input: Continuous-valued attributes \((x_1, x_2)\)
Output: 0 or 1

How do we branch on attribute values \(x_1\) and \(x_2\) to partition the space and generate correct outputs?

Example: Classification of Continuous Valued Inputs

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the \(K\) classes.

Decision Tree

\(x_1 < 3\)
\(x_1 < x_2 < 4\)
\(x_2 < 4\)
Expressiveness of Decision Trees

- Decision trees can express any function of the input attributes.
- E.g., Boolean functions, truth table row = 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 with one path to leaf for each example
  - But most likely won't generalize to new examples
  - Prefer to find more compact decision trees

Learning Decision Trees

- Example: When should I wait for a table at a restaurant?
Learning Decision Trees

- Example: When should I wait for a table at a restaurant?

- Attributes (features) relevant to Wait? decision:
  1. Alternate: is there an alternative restaurant nearby?
  2. Bar: is there a comfortable bar area to wait in?
  3. Fri/Sat: 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)

A “personal” decision tree

- A decision tree for Wait? based on personal “rules of thumb”:

![Decision Tree Diagram]

Patrons?
- None
- Some
- Full

WaitEstimate?
- >60
- 30-60
- 10-30
- 0-10

Alternate?
- Yes
- No

Hungry?
- Yes
- No

Reservation?
- Yes
- No

Fri/Sat?
- Yes
- No

Bar?
- Yes
- No

Alternate?
- Yes
- No

Raining?
- Yes
- No
Input Data for Learning

- Past examples when I did/did not wait for a table:

<table>
<thead>
<tr>
<th>Example</th>
<th>Attrs</th>
<th>Target</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T F F T Some $$ $$ F T</td>
<td>French</td>
<td>0–10</td>
</tr>
<tr>
<td>X₂</td>
<td>T F F T Full $ F F Thai</td>
<td>30–60</td>
<td>F</td>
</tr>
<tr>
<td>X₃</td>
<td>F T F F Some $ F F Burger</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₄</td>
<td>F T F T Full $ F F Thai</td>
<td>10–30</td>
<td>F</td>
</tr>
<tr>
<td>X₅</td>
<td>T F T F Full $$ $$ F T</td>
<td>French</td>
<td>&gt;60</td>
</tr>
<tr>
<td>X₆</td>
<td>F T F T Some $$ T T Italian</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₇</td>
<td>F T F F None $ T F Burger</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₈</td>
<td>F F F T Some $$ T T Thai</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₉</td>
<td>F T T F Full $ T F Burger</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T T T T Full $$ $$ F T</td>
<td>Italian</td>
<td>10–30</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F F F F None $ F F Thai</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T T T T Full $ F F Burger</td>
<td>30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

Decision Tree Learning

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

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 vᵢ of best do
        examples ← [elements of examples with best = vᵢ]
        subtree ← DTL(examples, attributes − best, MODE(examples))
        add a branch to tree with label vᵢ and subtree subtree
    return tree
Choosing an attribute to split on

- Idea: a good attribute should reduce uncertainty
- E.g., splits the examples into subsets that are (ideally) "all positive" (T) or "all negative" (F)

- *Patrons?* is a better choice

Reduce uncertainty?
How do you quantify uncertainty?

http://a.espncdn.com/media/timel/2006/0306/photo/g_moenroe_195.jpg
Use information theory!

- **Entropy** measures the amount of uncertainty in a probability distribution.

- **Entropy** (or information content in bits) of an answer to a question with \( n \) possible answers \( v_1, \ldots, v_n \):

  \[
  I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} P(v_i) \log_2 P(v_i)
  \]

Using information theory

- Suppose we have \( p \) examples with \( \text{Wait} = \text{True} \) (positive) and \( n \) examples with \( \text{Wait} = \text{false} \) (negative).

- Our best estimate of the probabilities of \( \text{Wait} = \text{true} \) or \( \text{false} \) is given by:

  \[
  P(\text{true}) \approx \frac{p}{p+n}
  \]

  \[
  P(\text{false}) \approx \frac{n}{p+n}
  \]

- Hence the entropy (in bits) is given by:

  \[
  I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = - \frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}
  \]
Choosing an attribute to split on

- Idea: a good attribute should reduce uncertainty and result in “gain in information”
- How much information do we gain if we disclose the value of some attribute?

- Answer:
  
  uncertainty before – uncertainty after
Before choosing an attribute:
Entropy = - 6/12 log(6/12) – 6/12 log(6/12)
= - log(1/2) = log(2) = 1 bit
There is “1 bit of information to be discovered”

If we choose Type: Along “French”: entropy = 1 bit.
Information gain = 1-1 = 0. (same for other branches)

If we choose Patrons:
In branches “None” and “Some”, entropy = 0
For “Full”, entropy = -2/6 log(2/6)-4/6 log(4/6) = 0.92
Info gain = (1-0) or (1-0.92) bits > 0 in both cases

So choosing Patrons gains more information!
Combining entropy across branches

- Compute average entropy
- Weight entropies according to probability of branches
  2/12 times we entered “None” so weight for “None” = 1/6
  “Some” has weight: 4/12 = 1/3
  “Full” has weight: 6/12 = ½

\[
\text{AvgEntropy} = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \text{Entropy}\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)
\]

Information gain

- Information Gain (IG) = reduction in entropy from using attribute A:
  \[ IG(A) = \text{Entropy before choosing} - \text{AvgEntropy after choosing } A \]

- When constructing each level of decision tree, choose attribute with largest IG
Information gain in our example

\[
IG(\text{Patrons}) = 1 - \left[ \frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = 0.541 \text{ bits}
\]

\[
IG(\text{Type}) = 1 - \left[ \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{3}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{3}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{1}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}
\]

\textit{Patrons} has highest IG of all attributes
\implies \text{ DTL algorithm chooses } \textit{Patrons} \text{ as the root

Learned Decision Tree for “Wait?”

- Decision tree learned from the 12 examples:

- Substantially simpler than “rules-of-thumb” tree
  - more complex hypothesis not justified by small amount of data
Performance Evaluation

- How do we know that the learned tree $h \approx true \ f$?
- Answer: Try $h$ on a new test set of examples
- Learning curve = % correct on test set as a function of training set size

![Learning Curve Image]

Generalization

- How do we know the classifier function we have learned is good?
  - Look at generalization error on test data
    - Method 1: Split data into separate training and test sets (the “hold out” method)
    - Method 2: Cross-Validation
Cross-validation

- **K-fold cross-validation:**
  - Divide data into K subsets of equal size
  - Train learning algorithm K times, leaving out one of the subsets. Compute error on left-out subset
  - Report average error over all subsets

- **Leave-1-out cross-validation:**
  - Train on all but 1 data point, test on that data point; repeat for each point
  - Report average error over all points

Next Time

- **Other classification methods**
  - Nearest Neighbor
  - Support Vector Machines

- **To Do:**
  - Project 4
  - Read Chapter 18