Our second algorithm

Let's look at a simple dataset for motivation:

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temp</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
</tbody>
</table>

- Class: Play tennis
- Attributes:
  - Outlook
  - Temp
  - Humidity
  - Windy

Let's focus on a simpler question: which attribute should we choose for root of tree?

- Given training set, can we build a tree that agrees with the data?
  (yes: easy; why?)

- What is a good decision tree?

- Given training set, how can we build a good tree?
Decision Trees

- Several selection criteria have been proposed and used.
- Information gain is commonly used (C4.5, J48)
- We need to learn about entropy ...

Entropy \( p_1, \ldots, p_n \) = \( \sum p_i \log \frac{1}{p_i} \)

For 2 classes \( p_1 = p, p_2 = 1 - p \) and this simplifies to

\[ \text{Entropy}(p) = - \sum p \log p - (1 - p) \log (1 - p) \]

\( S \rightarrow S_1, \ldots, S_k \)

where the \( S_i \) are subsets of \( S \)

and may include examples from multiple classes

\[ \text{Gain}(\text{Split}) = \text{Ent}(S) - \sum \frac{|S_j|}{|S|} \text{Ent}(S_j) \]

- **Outlook = Sunny:**
  info([2,3]) = entropy(2/5,3/5) = −2/5 log2(2/5) − 3/5 log2(3/5) = 0.971 bits
- **Outlook = Overcast:**
  info([4,0]) = entropy(1,0) = −1 log1(−0 log0) = 0 bits
  Note: defined as 0
- **Outlook = Rainy:**
  info([3,2]) = entropy(3/5,2/5) = −3/5 log3(3/5) − 2/5 log2(2/5) = 0.971 bits
- **Expected information for attribute:**
  info([3,2],[4,0],[3,2]) = (5/14) × 0.971 + (4/14) × 0 + (5/14) × 0.971 = 0.693 bits

\[ \text{gain(Outlook )} = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) \]

= 0.940 − 0.693

= 0.247 bits

\[ \text{gain(Outlook )} = \text{0.247 bits} \]

\[ \text{gain(Temperature )} = \text{0.029 bits} \]

\[ \text{gain(Humidity )} = \text{0.152 bits} \]

\[ \text{gain(Windy )} = \text{0.048 bits} \]
Continuing to Split

| Temperature | Gain = 0.571 bits |
| Humidity    | Gain = 0.971 bits |
| Windy       | Gain = 0.020 bits |

Final Decision Tree

Decision Tree Learning Algorithm

- If data has a pure class
  - Make leaf node with that class
- Otherwise
  - Pick feature to split on
  - Divide data into sub-datasets $S_j$ according to the feature’s values
  - Recursively build a tree for each subset

Improved Heuristic for Wide Splits

Gain(Split) = $\text{Ent}(S) - \sum_j \frac{|S_j|}{|S|} \text{Ent}(S_j)$

Heuristic for wide splits

SplitInfo = $\sum_j \frac{|S_j|}{|S|} \log \frac{|S|}{|S_j|}$

GainRatio = $\frac{\text{Gain}}{\text{SplitInfo}}$

Other Criteria / Tasks

The Gini Criterion = $4p(1 - p)$
The [KM] Criterion = $2\sqrt{p(1 - p)}$

Criterion for Regression = $\frac{1}{f} \sum_{i=1}^{f} (v_i - \bar{v})^2$

$\bar{v} = \frac{1}{f} \sum_{i=1}^{f} v_i$

Real Valued Attributes

- Naïve treatment makes a very side split with possibly one example per branch.
  - Is this good?
    - Alternative picks threshold $t$ and tests $(\text{feature} \geq t)$ to get a binary split.
    - How can we pick $t$?
The Bad News

- This does not quite work ...

![Graph showing accuracy vs. size of tree](image)

Overfitting in DT

- Why Does this happen?
  - Few examples at lower levels in tree
  - Quantities calculated “not reliable” in this case
  - Even worse with “noisy data”
  - And when features not sufficiently rich

- Solutions?

Overfitting in DT

- Min # points at leaf for split to be legal
- Stop growing tree if “no information”
- Pruning: grow full tree and then test whether some parts should be removed.
- How? Note that full tree always looks better on training data! so **just using accuracy on training data will not work**

Solution 1 (C4.5, J48): uses a confidence interval based on class ratio at leaf and number of examples in the training set.
- This is not fully justified but works well in practice.
- Solution 2: use a validation set. Known as reduced error pruning (REP)

REP Example

<table>
<thead>
<tr>
<th>Decision</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep</td>
<td>7</td>
</tr>
<tr>
<td>Prune</td>
<td>4</td>
</tr>
</tbody>
</table>

REP Example

<table>
<thead>
<tr>
<th>Decision</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep</td>
<td>11</td>
</tr>
<tr>
<td>Prune</td>
<td>14</td>
</tr>
</tbody>
</table>
Missing Attribute Values

- Common in real data
- We can handle this in a way that works across algorithms (that is, also for kNN).
- How?
- But we can do better with a solution tailored for decision trees. How?

Recap

- We already know two learning algorithms
- And many variants or improvements over their basic forms
- Given new application:
  - How do we know if one of these (say kNN) is doing well?
  - And which one is doing better?