Two tools for classifier learning

- Maximize predictive probability
- Minimize binary loss
- Function form
  - Linear
  - KNN
  - What else? Decision trees.

Decision Trees

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>Mid</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>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mid</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>Mid</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mid</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</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>Sunny</td>
<td>Mild</td>
<td>High</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>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>Yes</td>
</tr>
</tbody>
</table>

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

Comparing to kNN:

- DTs give a different way to identify regions in instance space:
  - Recursively split on values of features to define regions that have single label

- What does this look like with numerical features? (with threshold node tests, for example temp>23)

- Decision trees can represent any discrete function of discrete attributes!
- Why?
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?

To develop the algorithm
Let's focus on a simpler question first: which attribute should we choose for root of tree?
Our choices are ...

Which attribute should we choose for root of tree?
A numerical example:

\[
\begin{array}{c|c}
\text{Pos} & \text{Neg} \\
\hline
50 & 50 \\
35 & 15 \\
15 & 35 \\
0 & 50 \\
10 & 30 \\
30 & 10 \\
10 & 10 \\
25 & 25 \\
25 & 25 \\
\end{array}
\]

Assume for the moment that we have some good method of picking root attribute. What's next?

Assume we picked Outlook for the root. Then we must continue splitting each branch until ...?
### 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

### Final Decision Tree

![Decision Tree Diagram]

### Decision Trees

- Several selection criteria have been proposed and used.
- Information gain is commonly used (C4.5, J48)

  - We need to learn about entropy ...

### Decision Trees

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

= $- \sum p_i \log p_i$

For 2 classes $p_1 = p$, $p_2 = 1 - p$ and this simplifies to

$Entropy(p) = -p \log p - (1 - p) \log (1 - p)$

### Decision Trees

Now consider a split $S \rightarrow S_1, \ldots, S_k$
where the $S_i$ are subsets of $S$
and may include examples from multiple classes

$Gain(Split) = Ent(S) - \sum \frac{|S_j|}{|S|} Ent(S_j)$
Example: calculating Gain

- **Outlook = Sunny:**
  
  \[ \text{entropy} \left( \frac{2}{5}, \frac{3}{5} \right) = -2 \cdot \log \left( \frac{2}{5} \right) - 3 \cdot \log \left( \frac{3}{5} \right) = 0.971 \text{ bits} \]

- **Outlook = Overcast:**
  
  \[ \text{entropy} \left( 1.0 \right) = -1 \cdot \log (1) - 0 \cdot \log (0) = 0 \text{ bits} \]

- **Outlook = Rainy:**
  
  \[ \text{entropy} \left( \frac{3}{5}, \frac{2}{5} \right) = -3 \cdot \log \left( \frac{3}{5} \right) - 2 \cdot \log \left( \frac{2}{5} \right) = 0.971 \text{ bits} \]

- **Expected information for attribute:**
  
  \[ \text{info} \left( \frac{3}{5}, \frac{2}{5} \right) = \frac{3}{5} \cdot \log \left( \frac{3}{5} \right) + \frac{2}{5} \cdot \log \left( \frac{2}{5} \right) = 0.971 \text{ bits} \]

\[ \text{Note: defined as 0.} \]

Decision Trees

\[ \text{gain} (\text{Outlook }) = 0.940 - 0.693 = 0.247 \text{ bits} \]

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

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

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

Continuing to Split

\[ \text{gain} (\text{Temperature }) = 0.571 \text{ bits} \]

\[ \text{gain} (\text{Humidity }) = 0.971 \text{ bits} \]

\[ \text{gain} (\text{Windy }) = 0.020 \text{ bits} \]

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

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

Heuristic for wide splits

\[ \text{SplitInfo} = \sum_j \frac{|S_j|}{|S|} \log \frac{|S|}{|S_j|} \]

\[ \text{GainRatio} = \frac{\text{Gain}}{\text{SplitInfo}} \]
Other Criteria / Tasks

The Gini Criterion = 4p(1 − p)

What do these looks like?
Criterion for Regression = \( \frac{1}{L} \sum_{i=1}^{L} (y_i - \bar{y})^2 \)
\( \bar{y} = \frac{1}{L} \sum_{i=1}^{L} y_i \)

Real Valued Attributes

- Naive treatment makes a very wide split with possibly one example per branch.
  - Is this good?
- Alternative picks threshold t and tests (feature >= t) to get a binary split.
  - How can we pick t?

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?

The Bad News

- This does not quite work ...

The Bad News

- This is an example of overfitting
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

Overfitting in DT

• 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)

Pruning in C4.5 / J48

• $p$ is true error and $f$ is observed error
• Algorithm uses $f \sim N\left(p, \frac{p(1-p)}{n}\right)$

and some reasoning to claim that (actual formula used by C4.5 is more complex):

$\frac{1}{4n} Z_\alpha$

The error rate at each node is replaced with the upper bound
• Then the best pruning can be chosen

REP Example
DT Recap

- DT divide the example space through recursive splits of feature values
- Recursive learning algorithm relies on good choice of root attribute
- IG and other criteria are used for choice
- Several variants, improvements and generalizations
- Overfitting is a significant issue: solved by pruning or other methods