Decision Trees

Learning Decision Trees

Decision trees provide a very popular and efficient hypothesis space.
- **Variable Size**: Any boolean function can be represented.
- **Deterministic**
- **Discrete and Continuous Parameters**

Learning algorithms for decision trees can be described as:
- **Constructive Search**: The tree is built by adding nodes.
- **Eager**
- **Batch** (although earlier algorithms do exist).

Decision Tree Hypothesis Space

- **Internal nodes** test the value of particular features $x_j$ and branch according to the results of the test.
- **Leaf nodes** specify the class $A(k)$.

![Decision Tree Hypothesis Space Diagram]

Suppose the features are Outlook ($x_1$), Temperature ($x_2$), Humidity ($x_3$), and Wind ($x_4$). Then the feature vector $x = (Sunny, Hot, High, Strong)$ will be classified as No. The Temperature feature is irrelevant.

Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the $A$ classes.

![Decision Tree Decision Boundaries Diagram]

Decision Trees Can Represent Any Boolean Function

![Decision Trees Can Represent Any Boolean Function Diagram]

The tree will in the worst case require exponentially many nodes, however.
Decision Trees Provide Variable-Size Hypothesis Space

As the number of nodes (or depth) of a tree increases, the hypothesis space grows.

- **Depth 1** ("decision stump") can represent any boolean function of one feature.
- **Depth 2** Any boolean function of two features; some boolean functions involving three features (e.g., \( x_1 \land x_2 \lor \neg x_3 \))
- **etc.**

Learning Algorithm for Decision Trees

The same basic learning algorithm has been discovered by many people independently:

- Grow-Tree(S)
  - if \( y = 0 \) for all \((x, y) \in S\): return \( \text{leaf}(0) \)
  - else if \( y = 1 \) for all \((x, y) \in S\): return \( \text{leaf}(1) \)
  - else choose best attribute \( x_j \):
    - \( S_a = \{ (x, y) \in S \mid x_j = 0 \} \)
    - \( S_b = \{ (x, y) \in S \mid x_j = 1 \} \)
  - return \( \text{node}(x_j, \text{Grow-Tree}(S_a), \text{Grow-Tree}(S_b)) \)

Choosing the Best Attribute

One way to choose the best attribute is to perform a 1-step lookahead search and choose the attribute that gives the lowest error rate on the training data.

**CHOOSING ATTRIBUTE**

- Choose to minimize \( J(x_j) \) computed as follows:
  - \( S_0 = \{ (x, y) \in S \mid x_j = 0 \} \)
  - \( S_1 = \{ (x, y) \in S \mid x_j = 1 \} \)
  - \( p_j = \text{the most common value of } y \text{ in } S_0 \)
  - \( \hat{p}_j = \text{the most common value of } y \text{ in } S_1 \)
  - \( D_j = \text{number of examples } (x, y) \in S_0 \text{ with } y \neq p_j \)
  - \( D_j = \hat{p}_j + A_j \) (total error if we split on this feature)
- return \( j \)

Choosing the Best Attribute—An Example

```
<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( J = 2 \)
```

Choosing the Best Attribute (3)

Unfortunately, this measure does not always work well, because it does not detect cases where we are making "progress" toward a good tree.

```
\[
\begin{array}{lll}
\text{\( x1 \)} & \text{\( x2 \)} & \text{\( y \)} \\
\hline
\text{0} & \text{0} & \text{0} \\
\text{0} & \text{1} & \text{1} \\
\text{1} & \text{0} & \text{0} \\
\text{1} & \text{1} & \text{1} \\
\end{array}
\]

\( J = 10 \)
```

A Better Heuristic From Information Theory

Let \( V \) be a random variable with the following probability distribution:

\[
\begin{array}{cccc}
P(V = 0) & P(V = 1) \\
0.5 & 0.5 \\
\end{array}
\]

The surprisal, \( S(V = v) \) of each value of \( V \) is defined to be:

\[
S(V = v) = -\log P(V = v).
\]

As an event with probability 1 gives us no surprise.

It turns out that the surprisal is equal to the number of bits of information that need to be transmitted to a recipient who knows the probabilities of the results.

This is also called the description length of \( V = v \).

Fractional bits only make sense if they are part of a longer message (e.g., describe a whole sequence of coin tosses).
Entropy

The entropy of $V$, denoted $H(V)$, is defined as follows:

$$H(V) = -\sum_{x} P(x) \log P(x).$$

This is the average surprise of observing the result of one “trial” of $V$ (one coin toss).

Entropy can be viewed as a measure of uncertainty.

Mutual Information

Now consider two random variables $A$ and $B$ that are not necessarily independent. The mutual information between $A$ and $B$ is the amount of information we learn about $B$ by knowing the value of $A$ (and vice versa—it is symmetric). It is computed as follows:

$$I(A;B) = H(B) - \sum_{b} P(B = b) \cdot H(A|B = b)$$

In particular, consider the class $Y$ of each training example and the value of feature $x_i$ to be random variables. Then the mutual information quantifies how much $x_i$ tells us about the value of the class $Y$.

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$P(x_i=0)$</th>
<th>$P(x_i=1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5667</td>
<td>0.4333</td>
</tr>
<tr>
<td>1</td>
<td>0.5700</td>
<td>0.4300</td>
</tr>
<tr>
<td>2</td>
<td>0.5710</td>
<td>0.4290</td>
</tr>
<tr>
<td>3</td>
<td>0.5720</td>
<td>0.4280</td>
</tr>
</tbody>
</table>

Non-Boolean Features

- **Features with multiple discrete values**
  - Construct a multi-way split?
  - Test for one value versus all of the others?
  - Group the values into two disjoint subsets?

- **Real-valued features**
  - Consider a threshold split using each observed value of the feature.

Whatever method is used, the mutual information can be computed to choose the best split.

Learning Parity with Noise

When learning majority (3-bit parity), all splits look equally good. If extra random boolean features are included, they also look equally good. Hence, decision tree algorithms cannot distinguish random noisy features from parity features.

<table>
<thead>
<tr>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| 2 | 2 | 2 |
| 2 | 2 | 2 |
| 2 | 2 | 2 |

Attributes with Many Values

Problem:

- If attribute has many values, Gain will select it
- Imagine using Date = Jan. 3, 1996 as an attribute

One approach: use GainRatio instead

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) = -\sum_{S_i \subseteq S} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

where $S_i$ is subset of $S$ for which $A$ has value $v_i$
Unknown Attribute Values

What if some examples are missing values of $A$?
Use training example anyway, sort through tree
- If node $n$ tests $A$, assign most common value of $A$
  among other examples sorted to node $n$
- Assign most common value of $A$ among other examples
  with same target value
- Assign probability $p_i$ to each possible value $v_i$ of $A$
  Assign fraction $p_i$ of example to each descendant in tree
Classify new examples in same fashion

Overfitting

Consider error of hypothesis $h$ over
- training data: $err_{train}(h)$
- entire distribution $D$ of data: $err_D(h)$
Hypothesis $h \in H$ overfits training data if there is an
alternative hypothesis $h' \in H$ such that

\[ err_{train}(h) < err_{train}(h') \]

and

\[ err_D(h) > err_D(h') \]

Avoiding Overfitting

How can we avoid overfitting?
- Stop growing when data split not statistically significant
- Grow full tree, then post-prune

How to select “best” tree:
- Measure performance over training data
- Measure performance over separate validation data set
- Add complexity penalty to performance measure

Reduced-Error Pruning

Split data into training and validation set
Do until further pruning is harmful:
1. Evaluate impact on validation set of pruning each
   possible node (plus those below it)
2. Greedily remove the one that most improves validation
   set accuracy
Summary: Decision Trees

- Representation
- Tree growth
- Heuristics
- Overfitting and pruning
- Scaling up