Introduction

- Decision tree representation
- ID3 Algorithm
- Entropy, information gain
- Generalization and Overfitting
Sample Domain: Sports

- Learn when to play sports
- 6 Attributes
- What is the correct concept?

<table>
<thead>
<tr>
<th>Sky</th>
<th>Temp</th>
<th>Humid</th>
<th>Wind</th>
<th>Water</th>
<th>Forecast</th>
<th>EnjoySpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>Sunny</td>
<td>Warm</td>
<td>High</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cold</td>
<td>High</td>
<td>Strong</td>
<td>Warm</td>
<td>Change</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Warm</td>
<td>High</td>
<td>Strong</td>
<td>Cool</td>
<td>Change</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Representation

- Outlook
  - Sunny
  - Overcast
  - Rain
    - Wind
      - Strong
      - Weak
    - Yes

- Humidity
  - High
  - Normal
    - Yes
  - No
Decision Tree Representation

Each internal node tests an attribute.

Each branch corresponds to an attribute value node.

Each leaf node assigns a classification.
Decision Tree Representation

● Each internal node represents one attribute
● Each branch corresponds to attribute values
● Each leaf node assigns a classification

● How would you represent AND, OR, XOR
  ● \((A \cdot B) + (C \cdot \sim D \cdot E)\)
  ● \(M \text{ of } N\)
Decision Tree Representation

```plaintext
Decision Tree Representation

1. Outlook
   - Sunny
   - Hot
2. Humidity
   - High
   - Normal
3. Wind
   - Strong
   - Weak
4. PlayTennis
   - No
   - Yes
```

- Sunny
- Hot
- High
- Normal
- Strong
- Weak
- No
- Yes
Decision Tree for Conjunction
Decision Trees for Disjunctions

Outlook = Sunny \lor Wind = Weak

- Outlook
  - Sunny: Yes
  - Overcast
  - Rain
    - Wind: Weak
      - Yes
      - No
    - Strong
      - No
    - Weak
      - Yes
      - No
Decision Tree for XOR

Outlook=Sunny  XOR  Wind=Weak

- Outlook
  - Sunny
  - Overcast
  - Rain

  - Wind
    - Strong
      - Yes
    - Weak
      - No
    - Strong
      - No

  - Wind
    - Weak
      - Yes
    - Weak
      - No
    - Strong
      - Yes
Decision Tree Representation

- Naturally viewed as a Disjunction of conjunctions

\[(\text{Outlook} = \text{Sunny} \land \text{Humidity} = \text{Normal}) \lor (\text{Outlook} = \text{Overcast}) \lor (\text{Outlook} = \text{Rain} \land \text{Wind} = \text{Weak})\]
Decision Tree Example: conditions indicating a C-Section

#positive, negative cases proportion

Fetal_Presentation = 1: [822+,116-] .88+ .12-
| Previous_Csection = 0: [767+,81-] .90+ .10-
| | Primiparous = 0: [399+,13-] .97+ .03-
| | Primiparous = 1: [368+,68-] .84+ .16-
| | | Fetal_Distress = 0: [334+,47-] .88+ .12-
| | | Birth_Weight < 3349: [201+,10.6-] .95+ .05
| | | Birth_Weight >= 3349: [133+,36.4-] .78+ .2
| | | | Fetal_Distress = 1: [34+,21-] .62+ .38-
| Previous_Csection = 1: [55+,35-] .61+ .39-
Fetal_Presentation = 2: [3+,29-] .11+ .89-
Fetal_Presentation = 3: [8+,22-] .27+ .73-
Application of Decision Trees

● Well suited to problems where:
  ○ Attribute-Value pairs are involved
  ○ Target function is discrete valued
  ○ Disjunctions may be required
  ○ Possibly noisy training data
  ○ Attribute values in some training examples may be missing

● Applications such as
  ○ Equipment and medical diagnosis
  ○ Credit-risk analysis
  ○ Modeling calendar scheduling preferences
Top-Down Induction of Decision Trees: ID3

1. A -> the best decision attribute for next node
2. Assign A as decision attribute for node
3. For each value of A create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target attribute) stop,
6. else iterate over new leaf nodes.
Top Down Induction of Decision Trees

- Which is the \textit{best} attribute for a node?
Entropy

- If $S$ is a sample of data (in learning, presumably for training the learner)
- Entropy measures the *impurity* of $S$

$$Entropy(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_-$$

- $p_+$ is the percentage of positive examples
- $p_-$ is the percentage of negative examples
Entropy

- Entropy(S) = expected number of bits needed to encode the class of a randomly chosen member of S. Shortest length code
- Information theory: optimal length code assigns $-\log_2 p$ bits to a message with probability $p$
- So expected number of bits to encode a member is:

$$p_\oplus(-\log_2 p_\oplus) + p_\ominus(-\log_2 p_\ominus)$$

And entropy is thus defined as:

$$\text{Entropy}(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$
Entropy

- This assumes a binary categorization
- Entropy can be generalized – if there are c values

\[ \sum_{i=1}^{c} - p_i \log p_i \]

- \( p_i \) is the proportion of \( S \) belonging to class 1
- Still base 2 because it is an estimate of the expected code length in bits
- If target attribute can take on c possible values, entropy can be as large as \( \log_2 c \)
Information Gain

Gain(S, A): expected reduction in entropy due to sorting S on attribute A

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Entropy([29+, 35-]) = \frac{-29}{64} \log_2 \frac{29}{64} - \frac{35}{64} \log_2 \frac{35}{64}

$$= 0.99$$

\[ [29+, 35-] \quad A_1=? \]

True  False
[21+, 5-]  [8+, 30-]

\[ [29+, 35-] \quad A_2=? \]

True  False
[18+, 33-]  [11+, 2-]
Information Gain

Entropy([21+, 5-]) = 0.71
Entropy([8+, 30-]) = 0.74
Gain(S, A_1) = Entropy(S)
-26/64 * Entropy([21+, 5-])
-38/64 * Entropy([8+, 30-])
= 0.27

Entropy([18+, 33-]) = 0.94
Entropy([8+, 30-]) = 0.62
Gain(S, A_2) = Entropy(S)
-51/64 * Entropy([18+, 33-])
-13/64 * Entropy([11+, 2-])
= 0.12
## Entropy Example

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temp.</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play Tennis</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>Weak</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>Cold</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>Strong</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>
Selecting the Next Attribute

\[ S = [9+, 5-] \]
\[ E = 0.940 \]

**Humidity**

- **High**: [3+, 4-]
  - \[ E = 0.985 \]
- **Normal**: [6+, 1-]
  - \[ E = 0.592 \]

\[ \text{Gain}(S, \text{Humidity}) = 0.940 - \left( \frac{7}{14} \right) \times 0.985 - \left( \frac{7}{14} \right) \times 0.592 = 0.151 \]

\[ S = [9+, 5-] \]
\[ E = 0.940 \]

**Wind**

- **Weak**: [6+, 2-]
  - \[ E = 0.811 \]
- **Strong**: [3+, 3-]
  - \[ E = 1.0 \]

\[ \text{Gain}(S, \text{Wind}) = 0.940 - \left( \frac{8}{14} \right) \times 0.811 - \left( \frac{6}{14} \right) \times 1.0 = 0.048 \]
Selecting the Next Attribute

Gain from Temp (not shown, work it out yourself!) is 0.029, so Outlook is the best attribute.

This becomes the root of the tree and we make new child nodes for each outlook alternative…
ID3 Algorithm

\[ [D1, D2, ..., D14] \]
\[ [9+, 5-] \]

Outlook

Sunny

Overcast

Rain

\[ S_{\text{sunny}} = [D1, D2, D8, D9, D11] \]
\[ [2+, 3-] \]

?  Yes  ?

\[ \text{Gain (} S_{\text{sunny}}, \text{ Humidity)} = 0.970 - (3/5)0.0 - 2/5(0.0) = 0.970 \]
\[ \text{Gain (} S_{\text{sunny}}, \text{ Temp.)} = 0.970 - (2/5)0.0 - 2/5(1.0) - (1/5)0.0 = 0.570 \]
\[ \text{Gain (} S_{\text{sunny}}, \text{ Wind)} = 0.970 = -(2/5)1.0 - 3/5(0.918) = 0.019 \]
ID3 Algorithm

```
Outlook
  Sunny
    Humidity
      High
        No
          [D1,D2]
      Normal
        Yes
          [D8,D9,D11]
  Overcast
    Yes
      [D3,D7,D12,D13]
  Rain
    Wind
      Strong
        No
          [D6,D14]
      Weak
        Yes
          [D4,D5,D10]
```
Hypothesis Space Search in ID3
Hypothesis Space Search by ID3

- Hypothesis space is complete: target function guaranteed to be included
- Maintains a single current hypothesis as it goes through the space of decision trees, as opposed to an algorithm like candidate elimination that maintains a collection of hypotheses
  - because all possible hypotheses aren't maintained, can't pose new instance queries to resolve among competing hypotheses (20 questions)
  - no benefit from including an active learner
Hypothesis Space Search by ID3

- No backtracking in its pure form (Greedy search)
  - Local minima (suboptimal splits)
- Statistically based (information gain) search choices; uses all training examples to make choices at every step
  - Robust to noisy data
Inductive Bias in ID3

- Recall I.B. = set of assumptions that (along with training data) deductively justify classifications assigned by learner to future instances
- Here, H is the power set of instances
  - no representation bias because every possible hypothesis is included
- There IS bias here in how we build the tree
- Preference for short trees with high information gain attributes near the root
- Bias is a preference rather than a restriction
- Occam's razor: prefer the simplest hypothesis that fits the data
Why Prefer Short Hypotheses?

● Supporting:
  ○ Fewer short hypothesis than long, elaborate ones
  ○ A short hypothesis that fits the data is less likely to be coincidence than a long hypothesis that fits the data

● Problems
  ○ Can define lots of weird, short hypotheses:
    ■ e.g.: all trees with a prime number of nodes and whose attribute values start with “Z”
    ■ Just because they're unlikely to coincidentally fit data doesn't make them good!
  ○ Size of hypothesis is also related to internal rep. of the learner: 2 learners with different internal reps could arrive at different hypotheses and both use Occam's Razor to justify them
Example

- You say emeralds are green, sapphires are blue.
- A Martian says emeralds are *grue*, meaning that they are green until Jan 1, 2020, and then blue thereafter (sapphires are similarly, *bleen*)
- Clearly, no specific evidence either way
- You say it's ridiculous, that the concept of emeralds changing to blue in future is silly
- They say your ideas are silly: nothing's changing, emeralds are *always grue*
- *there may be concepts simpler than our experience allows us to think of*
Generalization

- Generalization means the ability of a learning algorithm to generalize from training data to previously unseen instances and to predict their target value.
- Training error is no indicator of how well the learning algorithm will generalize.
- Therefore, to study the effectiveness of a learning algorithm we partition the overall available set of data into training set and validation set.
- Train the system on the training set only and use the validation set to predict accuracy/error of the learner on the future unseen examples.
Overfitting

- When we train any learner with a set of data, there is always the danger that it is learning only the idiosyncrasies of the specific training set, as opposed to the general concepts involved.
- We say a hypothesis is overfitting to the degree its learning has been specialized to the training set as opposed to what will occur generally.
- What happens in our example as we add more unusual cases that don’t fit the pattern established so far?
Overfitting in Decision Trees

- Add noisy training data #15:
  - Sunny, hot, normal, strong, playtennis = no
- What effect would this have on the earlier tree?
Result

- A more complex tree
- New case goes to Humidity -> Normal branch, along with the previous 2 positive examples
- ID3 must search for a new attribute to separate these
- The result is a tree with a new level that will fit the training data perfectly
- The old tree won’t but which is more likely to fit the general case? If this new item is noise, the old tree!
Overfitting

- Formally, consider error of hypothesis $h$ over
  - Training data: $\text{error}_{\text{train}}(h)$
  - Entire distribution of data: $\text{error}_D(h)$
- Hypothesis $h$ overfits training data if there is an alternative hypothesis $h'$ such that
  - $\text{error}_{\text{train}}(h) < \text{error}_{\text{train}}(h')$ and
  - $\text{error}_D(h) > \text{error}_D(h')$
  - i.e. if our hypothesis $h$ fits the training set, but not the entire distribution, as well as some other hypothesis
Overfitting in Decision Trees

As we increase the number of nodes in a decision tree, overfitting becomes more pronounced – ID3 example:
Avoiding Overfitting

- How do we stop overfitting?
- 2 classes of solutions:
  - Stop growing when splitting data is not statistically significant (i.e. before it perfectly classifies training data)
  - Grow a full tree (i.e. allow it to overfit the data), then post prune to remove the parts that overfit
What is the correct final tree size?

- No matter which approach is used, criteria for choosing final tree size is key. Approaches include:
  - Measure performance over separate validation data (distinct from training data) to evaluate utility of pruning nodes
  - Use all data for training, but use a statistical test to estimate whether expanding/pruning a node will likely produce an improvement beyond the training set
  - Use a measure of the complexity for encoding training examples and the tree, and halt tree growth when this encoding size is minimized
    - eg MDL: minimize size(tree) + size(missclassifications(tree))
Reduced Error Pruning Algorithm

- CART System, Breiman et al, 1984
- Split data into training and validation data
- Do until further pruning is harmful
  - Evaluate performance of validation data after pruning each possible subtree
  - Greedily remove the one that reduces the validation error the most
- Any leaf node added due to irregularities in the training set is likely to be pruned, because the same irregularities are unlikely to occur in the validation set
- Produces smallest version of most accurate subtree
Effect of Reduced Error Pruning
Reduced Error Pruning

- Separate set of data to guide pruning is only useful if a reasonable amount of data is available
- If data is limited, withholding some for test data further reduces the amount available for training
- Need an alternative for situations where data is limited
- To see one, first note that a decision tree is easily converted to a set of rules
Tree -> Rules Conversion

- Follow a path from the root to a leaf node and create a conjunction of attribute tests
  - If (Outlook = sunny) and (Humidity = high) then PlayTennis = No
  - If (Outlook = sunny) and (Humidity = normal) then PlayTennis = Yes
Decision Tree - Rules Conversion

\begin{align*}
R_1 &: \text{If } (\text{Outlook}=\text{Sunny}) \land (\text{Humidity}=\text{High}) \text{ Then PlayTennis}=\text{No} \\
R_2 &: \text{If } (\text{Outlook}=\text{Sunny}) \land (\text{Humidity}=\text{Normal}) \text{ Then PlayTennis}=\text{Yes} \\
R_3 &: \text{If } (\text{Outlook}=\text{Overcast}) \text{ Then PlayTennis}=\text{Yes} \\
R_4 &: \text{If } (\text{Outlook}=\text{Rain}) \land (\text{Wind}=\text{Strong}) \text{ Then PlayTennis}=\text{No} \\
R_5 &: \text{If } (\text{Outlook}=\text{Rain}) \land (\text{Wind}=\text{Weak}) \text{ Then PlayTennis}=\text{Yes}
\end{align*}
Rule Post Pruning

- Create the entire tree to accurately fit training data, allowing overfitting
- Convert tree to equivalent set of rules: 1 for each root->leaf path
- Prune each rule independently by removing preconditions that result in improving the rule’s accuracy (i.e. if precondition does not worsen rule’s accuracy)
- Sort final rules by accuracy and use them in this sequence
- Popular method used in C4.5, J4.8
Rule Post Pruning

- The estimation of rule accuracy could be done with a validation set.
- The version of this used in C4.5 uses the training set itself.
- A pessimistic estimate is used to make up for the fact that the training data is biased in favour of the rule.
- Calculate rule accuracy over training examples to which it applies, then take the standard deviation of this assuming a binomial distribution. Use this as a lower bound. For large data sets, this is close to observed accuracy.
- Converting the decision tree to rules allows distinguishing the contexts where a node is used (as opposed to pruning or not pruning a node that participates in many paths).
Dealing with Limited Data

- As opposed to trying to estimate how much bias there is toward training data, there are alternative ways to deal with the problem of data that is too limited to adequately separate into training and validation datasets.

- There are several schemes involving partitioning the data in multiple ways and then averaging the results.
Validation

● What are we doing when we evaluate a machine learning system?
  ○ Estimate the accuracy of a hypothesis induced by a supervised learning system
  ○ Predict the accuracy of a hypothesis on future unseen instances
  ○ Select the optimal hypothesis from a given set of hypotheses
    ■ Pruning decision tree
    ■ Model selection
    ■ Feature selection
Holdout Method

- What we've described as the general concept of separating your training and validation data is commonly known as the *holdout method*.
- For a supervised learner, the input to the system will be a set of tuples \((v_1, y_1)\), where \(v=\)instance, \(y=\)classification.
- Splitting the available set of tuples \(D\) results in a training set \(D_t\) and a validation set \(D_h = D \setminus D_t\).
Holdout Method

● Define the accuracy of the learned classification given the validation set h as \( \text{acc}_h \):

\[
\text{acc}_h = \frac{1}{|D_h|} \sum_{(v_i, y_i) \in D_h} \delta(L(D_t, v_i), y_i)
\]

● where \( L(D_t, v_i) \) is the output of the hypothesis induced by learner L trained on data \( D_t \), when it is given the instance \( V_i \)

● And \( \delta(x, y) \) is 1 if \( x = y \), 0 otherwise
Holdout Method

- Problems with this?
- Makes insufficient use of data
- Training and Validation data may be correlated

- *Cross-validation techniques* attempt to overcome these problems
K-Fold Cross Validation

- Instead of splitting into training and validation datasets, we split the data into k subsets \((D_1, \ldots, D_k)\).
- We train and test the learner k times.
- When it is trained with \(D \setminus D_i\), we test with \(D_i\).

\[
acc_{cv} = 1/|D| \sum_{(v_i, y_i) \in D} \delta(L(D \setminus D_i, v_i), y_i)
\]
Cross-Validation Variants

● The point of all of these is to attempt to use *all* the data for training and testing
● *Complete* k-fold cross-validation splits the dataset of size m in all \( \binom{m}{m/k} \) possible ways (choosing \( m/k \) instances out of m)
● *Leave n-out* cross-validation sets n partitions aside for testing and uses the remaining ones for training (leave one-out is equivalent to n fold cross-validation)
● In *stratified* cross-validation, the folds are stratified so that they contain approximately the same proportion of labels (positives, negatives, attribute values) as the original data set
Another variant: Bootstrap (BT632)

- Samples \( n \) instances uniformly from the data set with replacement (i.e. you can take the same instance > once)
- Probability that any given instance is not chosen after \( n \) samples is \((1-1/n)^n = e^{-1} = 0.632\)
- The bootstrap sample \( (D_b) \) is used for training while the remaining instances are used for testing

\[
\text{acc}_{\text{boot}} = 1/|D_b| \sum_{i=1}^{b} \left( 0.632 \text{acc}_{b_i} + 0.368 \text{acc}_{t_i} \right)
\]

- where \( \text{acc}_{b_i} \) is the accuracy on the test data of the \( i \)-th bootstrap sample, \( \text{acc}_{t_i} \) is the accuracy estimate on the training set and \( b \) the number of bootstrap samples
Extensions to ID3: Continuous Valued Attributes

- Initial specification of ID3 restricts attributes to those that take on discrete values
- We can extend ID3 to deal with attributes that are continuous-valued
- Need to define new discrete-valued attributes that partition the continuous-valued attribute into a set of intervals
- e.g., temperature
Extensions to ID3: Continuous Valued Attributes

Create a discrete attribute to test continuous

- Temperature = 24.5°C
- \((\text{Temperature} > 20.0°C) = \{\text{true}, \text{false}\}\)

Where to set the threshold?

<table>
<thead>
<tr>
<th>Temperature</th>
<th>15°C</th>
<th>18°C</th>
<th>19°C</th>
<th>22°C</th>
<th>24°C</th>
<th>27°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlayTennis</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

(see paper by [Fayyad, Irani 1993])
Extensions to ID3: Continuous Valued Attributes

- We would like to pick a threshold $c$ that produces the greatest information gain.
- Sort the examples according to their continuous attribute value, then identify adjacent examples differing in target classification.
- The value of $c$ that maximizes information gain will always lie on such a boundary.
- Candidate thresholds can be evaluated by computing the information gain associated with each threshold.
Attributes With Many Values

- If attribute has many values, then Information Gain will tend to select it (doesn’t happen with Temp in our example though!)
  - Imagine using Date=June 3rd 1996
- Use GainRatio instead – incorporates split information, which is sensitive to how broadly and uniformly the attribute splits the data (entropy of s wrt values of attr a)

\[
\text{GainRatio}(S, A) \equiv \frac{\text{Gain}(S, A)}{\text{SplitInformation}(S, A)}
\]

\[
\text{SplitInformation}(S, A) \equiv - \sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}
\]

where \( S_i \) is subset of \( S \) for which \( A \) has value \( v_i \)
Attributes with Costs

- In the real world we have a broad selection of attributes; some will be cheap and easy to obtain, while others will be very expensive (e.g. blood test vs. MRI)
- Want to learn a consistent tree with low expected cost
- Replace gain by

\[
\frac{\text{Gain}^2(S, A)}{\text{Cost}(A)}.
\]

Tan and Schlimmer (1990)

\[
\frac{2 \text{Gain}(S,A) - 1}{(\text{Cost}(A) + 1)^w}
\]

Nunez (1988)

where \( w \in [0, 1] \) determines importance of cost vs info gain
Missing Attribute Values

- May have data that is missing some attribute values (e.g. expensive tests, items missing irretrievably from historical data)
- Essentially must ask “what is this value likely to be”?
- Common to estimate the missing attribute value based on other examples for which the attribute value is known
- There are a number of possible ways this can be done...assume the attribute for which we have missing values is A
Dealing with Unknown Attribute Values

- Assign most common value of A among other examples with the same classification
- If node n tests A, then assign most common value of A among other examples at node n
- More elaborately, could use probabilities
  - Assign probability $p_i$ to each possible value $v_i$ of A. These can be estimated based on observed frequencies of values of A at node n
  - Assign fraction $p_i$ of example to each child node
  - this same fractioning can be used after learning to classify new instances with unknown values
● Attempt improve the learner by using only the features that are most suitable
  ○ i.e., rather than prune an overfitted tree, look at what features actually make a practical difference to the learner and only use those
● Evaluate the results using only a subset of features, and ultimately use only the features that make for the best learner
Wrapper Model

● Evaluate the accuracy of the inducer for a given subset of features by means of n-fold cross-validation
● The training data is split into n folds, and the induction algorithm is run n times.
● The accuracy results are averaged to produce the estimated accuracy.
● Strategies for exploring feature subsets?
  ○ *Forward elimination*: Starts with the empty set of features and greedily adds the feature that improves the estimated accuracy at most
  ○ *Backward elimination*: Starts with the set of all features and greedily removes the worst feature
Summary

- Decision tree representation of hypothesis
  - Exhaustive representation (Power set of X)
- Learning decision trees using ID3
  - Select attribute using information gain
  - Hill climbing search
- Overfitting problem
  - Post pruning
- Extension to deal with continuous errors, attribute with many values, and unknown attribute values