Goal

- Classification without Models
  - Well, partially without a model

- Today: Decision Trees
Decision trees
Why Trees?

- interpretable/intuitive, popular in medical applications because they mimic the way a doctor thinks
- model discrete outcomes nicely
- can be very powerful, can be as complex as you need them
- C4.5 and CART - from “top 10” entries on Kaggle - decision trees are very effective and popular
Sure, But Why Trees?

- Easy to understand knowledge representation
- Can handle mixed variables
- Recursive, divide and conquer learning method
- Efficient inference
Tree learning

- Top-down recursive divide and conquer algorithm
  - Start with all examples at root
  - Select **best** attribute/feature
  - Grow it by “splitting” attributes one by one. To determine which attribute to split, look at “node impurity.”
  - Recurse and repeat

- Other issues:
  - How to construct features
  - When to stop growing
  - Pruning irrelevant parts of the tree
Score each attribute split for these instances: Age, Degree, StartYr, Series7

Score each attribute split for these instances: Age, Degree, StartYr

Score each attribute split for these instances: Age, Degree, StartYr, Series7

<table>
<thead>
<tr>
<th>Fraud</th>
<th>Age</th>
<th>Degree</th>
<th>StartYr</th>
<th>Series7</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>22</td>
<td>Y</td>
<td>2005</td>
<td>N</td>
</tr>
<tr>
<td>-</td>
<td>25</td>
<td>N</td>
<td>2003</td>
<td>Y</td>
</tr>
<tr>
<td>-</td>
<td>31</td>
<td>Y</td>
<td>1995</td>
<td>Y</td>
</tr>
<tr>
<td>-</td>
<td>27</td>
<td>Y</td>
<td>1999</td>
<td>Y</td>
</tr>
<tr>
<td>+</td>
<td>24</td>
<td>N</td>
<td>2006</td>
<td>N</td>
</tr>
<tr>
<td>-</td>
<td>29</td>
<td>N</td>
<td>2003</td>
<td>N</td>
</tr>
</tbody>
</table>

choose split on Series7

<table>
<thead>
<tr>
<th>Fraud</th>
<th>Age</th>
<th>Degree</th>
<th>StartYr</th>
<th>Series7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>25</td>
<td>N</td>
<td>2003</td>
<td>Y</td>
</tr>
<tr>
<td>-</td>
<td>31</td>
<td>Y</td>
<td>1995</td>
<td>Y</td>
</tr>
<tr>
<td>-</td>
<td>27</td>
<td>Y</td>
<td>1999</td>
<td>Y</td>
</tr>
</tbody>
</table>

choose split on Age>28

<table>
<thead>
<tr>
<th>Fraud</th>
<th>Age</th>
<th>Degree</th>
<th>StartYr</th>
<th>Series7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>29</td>
<td>N</td>
<td>2003</td>
<td>N</td>
</tr>
<tr>
<td>+</td>
<td>22</td>
<td>Y</td>
<td>2005</td>
<td>N</td>
</tr>
<tr>
<td>+</td>
<td>24</td>
<td>N</td>
<td>2006</td>
<td>N</td>
</tr>
</tbody>
</table>
Overview (with two features and 1D target)

Features: $X_1, X_2$
Target: $Y$
Tree models

- Most well-known systems
  - CART: Breiman, Friedman, Olshen and Stone
  - ID3, C4.5: Quinlan

- How do they differ?
  - **Split scoring function**
  - Stopping criterion
  - Pruning mechanism
  - Predictions in leaf nodes
Scoring functions: Local split value
Choosing an attribute/feature

- Idea: a good feature splits the examples into subsets that distinguish among the class labels as much as possible... ideally into pure sets of "all positive" or "all negative"

- Bias-variance tradeoff: Choosing most discriminating attribute first may not be best tree (bias), but it can make tree small (low variance).
Association between attribute and class label

Data

<table>
<thead>
<tr>
<th>age</th>
<th>income</th>
<th>student</th>
<th>credit_rating</th>
<th>buys_computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=30</td>
<td>high</td>
<td>no</td>
<td>fair</td>
<td>no</td>
</tr>
<tr>
<td>&lt;=30</td>
<td>high</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>31...40</td>
<td>high</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;40</td>
<td>medium</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;40</td>
<td>low</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;40</td>
<td>low</td>
<td>yes</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>31...40</td>
<td>low</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;=30</td>
<td>medium</td>
<td>no</td>
<td>fair</td>
<td>no</td>
</tr>
<tr>
<td>&lt;=30</td>
<td>low</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;40</td>
<td>medium</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;=30</td>
<td>medium</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>31...40</td>
<td>medium</td>
<td>no</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>31...40</td>
<td>high</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;40</td>
<td>medium</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
</tbody>
</table>

Contingency table

<table>
<thead>
<tr>
<th>Attribute value</th>
<th>Class label value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Buy</td>
</tr>
<tr>
<td>High</td>
<td>2</td>
</tr>
<tr>
<td>Med</td>
<td>4</td>
</tr>
<tr>
<td>Low</td>
<td>3</td>
</tr>
</tbody>
</table>

Income

High

High

no

no

yes

yes

Yes

No

No

Yes

Yes
Mathematically Defining “Good Split”

- We start with information theory
  - How uncertain of the answer will be if we split the tree this way?

- Say need to decide between \( k \) options
  - Uncertainty in the answer \( Y \in \{1, \ldots, k\} \) when probability is \( \langle p_1, \ldots, p_k \rangle \) can be quantified via entropy:

\[
H(\langle p_1, \ldots, p_k \rangle) = \sum_i -p_i \log_2 p_i
\]

- Convenient notation: \( B(p) = H(\langle p, 1 - p \rangle) \), number of bits necessary to encode
Amount of Information in the Tree

- Suppose we have \( p \) positive and \( n \) negative examples at the root
  \( \Rightarrow \) \( B(p/(p + n)) \) bits needed to classify a new example

- Information is always conserved
  - If encoding the information in the leaves is lossless then tree has lossless encoding
  - The entropy of the leaves (amount of bits) + the tree information (bits) carried in the tree = total information in the data

- Let split \( Y_i \) have \( p_i \) positive and \( n_i \) negative examples
  \( \Rightarrow \) \( B(p_i/(p_i + n_i)) \) bits needed to classify a new example
  \( \Rightarrow \) expected number of bits per example over all branches is

\[
\sum_i \frac{p_i + n_i}{p + n} B\left(\frac{p_i}{(p_i + n_i)}\right)
\]

\( \Rightarrow \) choose the next attribute to split that minimizes the remaining information needed
  - Which maximizes the information in the tree (as information is conserved)
Information gain

- Information Gain (Gain) is the amount of information that the tree structure encodes.

\[ H[x] \] is the entropy: expected number of bits to encode a randomly selected \( x \):

\[
\text{Gain}(S, A) = H[S] - \sum_{A \subseteq A} \frac{|A|}{|S|} H[A]
\]

\[
H[\text{buys\_computer}] = -\frac{9}{14} \log \frac{9}{14} - \frac{5}{14} \log \frac{5}{14} = 0.9400
\]
Information gain

\[
\text{Gain}(S, A) = H[S] - \sum_{A \subseteq A} \frac{|A|}{|S|} H[A]
\]

\begin{align*}
\text{Entropy}(Income=\text{high}) & = -\frac{2}{4} \log \frac{2}{4} - \frac{2}{4} \log \frac{2}{4} = 1 \\
\text{Entropy}(Income=\text{med}) & = -\frac{4}{6} \log \frac{4}{6} - \frac{2}{6} \log \frac{2}{6} = 0.9183 \\
\text{Entropy}(Income=\text{low}) & = -\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} = 0.8113 \\
\text{Gain}(D, Income) & = 0.9400 - (\frac{4}{14} [1] + \frac{6}{14} [0.9183] + \frac{4}{14} [0.8113]) \\
& = 0.029
\end{align*}
Gini gain

- Similar to information gain
- Uses gini index instead of entropy

\[ Gini(X) = 1 - \sum_{x} p(x)^2 \]

- Measures decrease in gini index after split:

\[ \text{Gain}(S, A) = Gigi(S) - \sum_{A \subseteq A} \frac{|A|}{|S|} \text{Gini}(A) \]
Comparing information gain to gini gain

\[ IG = 0 \]

\[ GG = 0 \]
Comparing information gain to gini gain

\[ IG = 1.0 - \left[ \frac{2}{12} 0 \right] - \left[ \frac{4}{12} 0 \right] - \left[ \frac{6}{12} 0.919 \right] = 0.541 \]

\[ GG = 0.5 - \left[ \frac{2}{12} 0 \right] - \left[ \frac{4}{12} 0 \right] - \left[ \frac{6}{12} 0.444 \right] = 0.278 \]
How does score function affect feature selection?

66% split: \(\text{Entropy} = 0.919\)  
\(Gini \times 2 = 0.889\)

85% split: \(\text{Entropy} = 0.610\)  
\(Gini \times 2 = 0.510\)

Gini score can produce larger gain
Chi-Square score

- Widely used to test independence between two categorical attributes (e.g., feature and class label)
- Considers counts in a contingency table and calculates the normalized squared deviation of observed (predicted) values from expected (actual) values

\[ \chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \]

- Sampling distribution is known to be chi-square distributed, given that cell counts are above minimum thresholds
## Contingency tables

<table>
<thead>
<tr>
<th></th>
<th>Buy</th>
<th>No buy</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Med</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Low</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td><strong>9</strong></td>
<td><strong>5</strong></td>
<td><strong>14</strong></td>
</tr>
</tbody>
</table>
Calculating expected values for a cell

\[ \chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \]

\[ o_{(0,+)} = a \]

\[ e_{(0,+)} = p(A = 0, C = +) \cdot N \]

\[ = p(A = 0)p(C = +|A = 0) \cdot N \]

\[ = p(A = 0)p(C = +) \cdot N \quad \text{(assuming independence)} \]

\[ = \left[ \frac{a + b}{N} \right] \cdot \left[ \frac{a + c}{N} \right] \cdot N \]
Example calculation

<table>
<thead>
<tr>
<th></th>
<th>Observed</th>
<th></th>
<th>Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Buy</td>
<td>No buy</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>2</td>
<td>2</td>
<td>2.57</td>
</tr>
<tr>
<td>Med</td>
<td>4</td>
<td>2</td>
<td>3.86</td>
</tr>
<tr>
<td>Low</td>
<td>3</td>
<td>1</td>
<td>2.57</td>
</tr>
</tbody>
</table>

\[
\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i} = \left( \frac{(2 - 2.57)^2}{2.57} \right) + \left( \frac{(4 - 3.86)^2}{3.86} \right) + \left( \frac{(3 - 2.57)^2}{2.57} \right) + \left( \frac{(2 - 1.43)^2}{1.43} \right) + \left( \frac{(2 - 2.14)^2}{2.14} \right) + \left( \frac{(1 - 1.43)^2}{1.43} \right)
\]

\[
= 0.57
\]
Tree learning

- Top-down recursive divide and conquer algorithm
  - Start with all examples at root
  - Select **best** attribute/feature
  - Partition examples by selected attribute
  - Recurse and repeat

- Other issues:
  - How to construct features
  - **When to stop growing**
  - **Pruning irrelevant parts of the tree**
Overfitting

- Consider a distribution D of data representing a population and a sample Ds drawn from D, which is used as training data.

- Given a model space M, a score function S, and a learning algorithm that returns a model m ∈ M, the algorithm **overfits** the training data Ds if:
  \[ \exists \ m' \in M \text{ such that } S(m,Ds) > S(m',Ds) \text{ but } S(m,D) < S(m',D) \]

  - In other words, there is another model (m’) that is better on the entire distribution and if we had learned from the full data we would have selected it instead.
Example learning problem

Knowledge representation:
If-then rules

Example rule:
If $x > 25$ then +
Else -

What is the model space?
All possible thresholds

What score function?
Prediction error rate
Score function over model space

Search procedure?

Try all thresholds, select one with lowest score

Best result on full data

Overfitting

Full data

Small sample

Biased sample

Large sample
Approaches to avoid overfitting

- Regularization (Priors)
  - e.g., Dirichlet prior in NBC

- Hold out evaluation set, used to adjust structure of learned model
  - e.g., pruning in decision trees

- Statistical tests during learning to only include structure with significant associations
  - e.g., pre-pruning in decision trees

- Penalty term in classifier scoring function
  - i.e., change score function to prefer simpler models
How to avoid overfitting in decision trees

- **Postpruning**
  - Use a separate set of examples to evaluate the utility of pruning nodes from the tree (after tree is fully grown)

- **Prepruning**
  - Apply a statistical test to decide whether to expand a node
  - Use an explicit measure of complexity to penalize large trees (e.g., Minimum Description Length)
Algorithm comparison

- **CART**
  - Evaluation criterion: **Gini index**
  - Search algorithm: Simple to complex, hill-climbing search
  - Stopping criterion: When leaves are pure
  - Pruning mechanism: **Cross-validation to select gini threshold**

- **C4.5**
  - Evaluation criterion: **Information gain**
  - Search algorithm: Simple to complex, hill-climbing search
  - Stopping criterion: When leaves are pure
  - Pruning mechanism: **Reduce error pruning**
Example: reduced error pruning

- Use **pruning set** to estimate accuracy in sub-trees and for individual nodes

- Let T be a sub-tree rooted at node v

- Define:

  \[
  \text{Gain from pruning at } v = \#\text{misclassification in } T - \#\text{misclassification at } v
  \]

- Repeat: Prune at node with largest gain until only negative gain nodes remain

- “Bottom-up restriction”: T can only be pruned if it does not contain a sub-tree with lower error than T

Source: [www.ailab.si/blaz/predavanja/uisp/slides/uisp05-PostPruning.ppt](http://www.ailab.si/blaz/predavanja/uisp/slides/uisp05-PostPruning.ppt)
Pre-pruning methods

- Stop growing tree at some point during top-down construction when there is no longer sufficient data to make reliable decisions

- Approach:
  - Choose threshold on feature score
  - Stop splitting if the best feature score is below threshold
Determine chi-square threshold analytically

- Stop growing when chi-square feature score is not **statistically significant**
- Chi-square has known sampling distribution, can look up significance threshold
  - Degrees of freedom = (#rows - 1)(#cols - 1)
  - 2X2 table: 3.84 is 95% critical value

\[
\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i}
\]

![Chi-square distribution graph](image)
K-fold cross validation

- Randomly **partition** training data into \( k \) folds
- For \( i=1 \) to \( k \)
  - Learn model on \( D - i^{th} \) fold; evaluate model on \( i^{th} \) fold
- Average results from all \( k \) trials
Choosing a Gini threshold with cross validation

- For $i$ in 1..$k$
  - For $t$ in threshold set (e.g., [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8])
    - Learn decision tree on $\text{Train}_i$ with Gini gain threshold $t$ (i.e. stop growing when max Gini gain is less than $t$)
    - Evaluate learned tree on $\text{Test}_i$ (e.g., with accuracy)
  - Set $t_{\text{max},i}$ to be the $t$ with best performance on $\text{Test}_i$
- Set $t_{\text{max}}$ to the average of $t_{\text{max},i}$ over the $k$ trials
- Relearn the tree on all the data using $t_{\text{max}}$ as Gini gain threshold
How to interpret overfitting avoidance methods?

Modification of score function... to better represent model value.