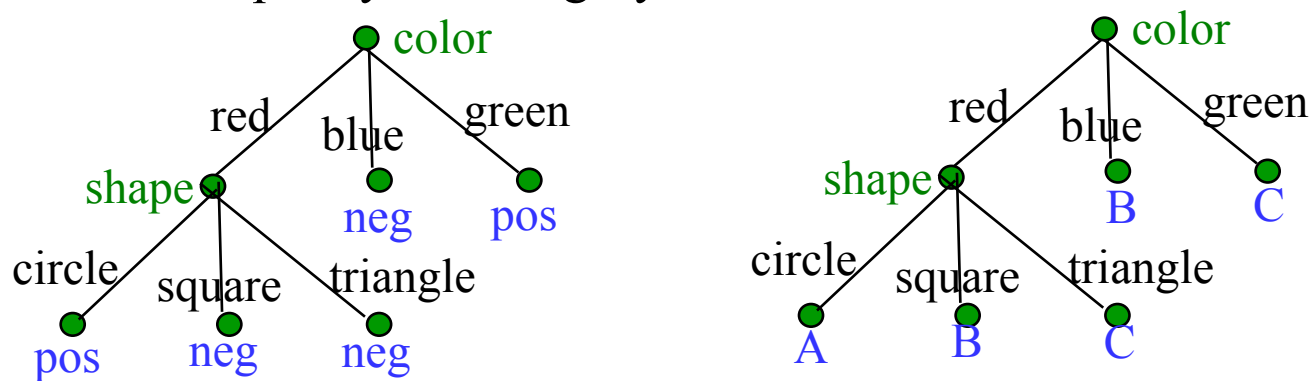

Decision Tree Learning

Original slides by Raymond J. Mooney

University of Texas at Austin

Decision Trees

- Tree-based classifiers for instances represented as feature-vectors. Nodes test features, there is one branch for each value of the feature, and leaves specify the category.



- Can represent arbitrary conjunction and disjunction. Can represent any classification function over discrete feature vectors.
- Can be rewritten as a set of rules, i.e. disjunctive normal form (DNF).
 - $\text{red} \wedge \text{circle} \rightarrow \text{pos}$
 - $\text{red} \wedge \text{circle} \rightarrow A$
 - $\text{blue} \rightarrow B$; $\text{red} \wedge \text{square} \rightarrow B$
 - $\text{green} \rightarrow C$; $\text{red} \wedge \text{triangle} \rightarrow C$

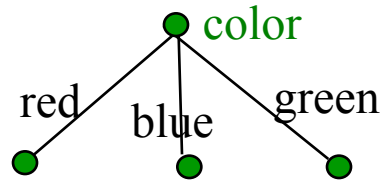
Properties of Decision Tree Learning

- Continuous (real-valued) features can be handled by allowing nodes to split a real valued feature into two ranges based on a threshold (e.g. $\text{length} < 3$ and $\text{length} \geq 3$)
- Classification trees have discrete class labels at the leaves, *regression trees* allow real-valued outputs at the leaves.
- Algorithms for finding consistent trees are efficient for processing large amounts of training data for data mining tasks.
- Methods developed for handling noisy training data (both class and feature noise).
- Methods developed for handling missing feature values.

Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

<big, red, circle>: + <small, red, circle>: +
<small, red, square>: - <big, blue, circle>: -

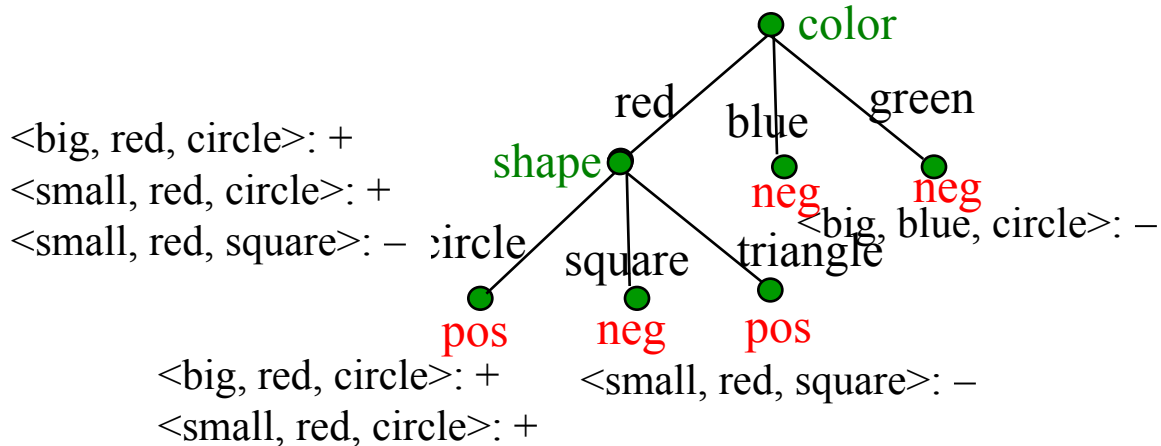


<big, red, circle>: +
<small, red, circle>: +
<small, red, square>: -

Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

<big, red, circle>: + <small, red, circle>: +
 <small, red, square>: - <big, blue, circle>: -



Decision Tree Induction Pseudocode

DTree(*examples*, *features*) returns a tree

If all *examples* are in one category, return a leaf node with that category label.

Else if the set of *features* is empty, return a leaf node with the category label that is the most common in examples.

Else pick a feature F and create a node R for it

For each possible value v_i of F :

Let $examples_i$ be the subset of examples that have value v_i for F

Add an out-going edge E to node R labeled with the value v_i .

If $examples_i$ is empty

then attach a leaf node to edge E labeled with the category that is the most common in *examples*.

else call DTree($examples_i$, $features - \{F\}$) and attach the resulting tree as the subtree under edge E .

Return the subtree rooted at R .

Picking a Good Split Feature

- Goal is to have the resulting tree be as small as possible, per Occam's razor.
- Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
- Top-down divide-and-conquer method does a greedy search for a simple tree but does not guarantee to find the smallest.
 - General lesson in ML: “Greed is good.”
- Want to pick a feature that creates subsets of examples that are relatively “pure” in a single class so they are “closer” to being leaf nodes.
- There are a variety of heuristics for picking a good test, a popular one is based on information gain that originated with the ID3 system of Quinlan (1979).

Entropy

- Entropy (disorder, impurity) of a set of examples, S , relative to a binary classification is:

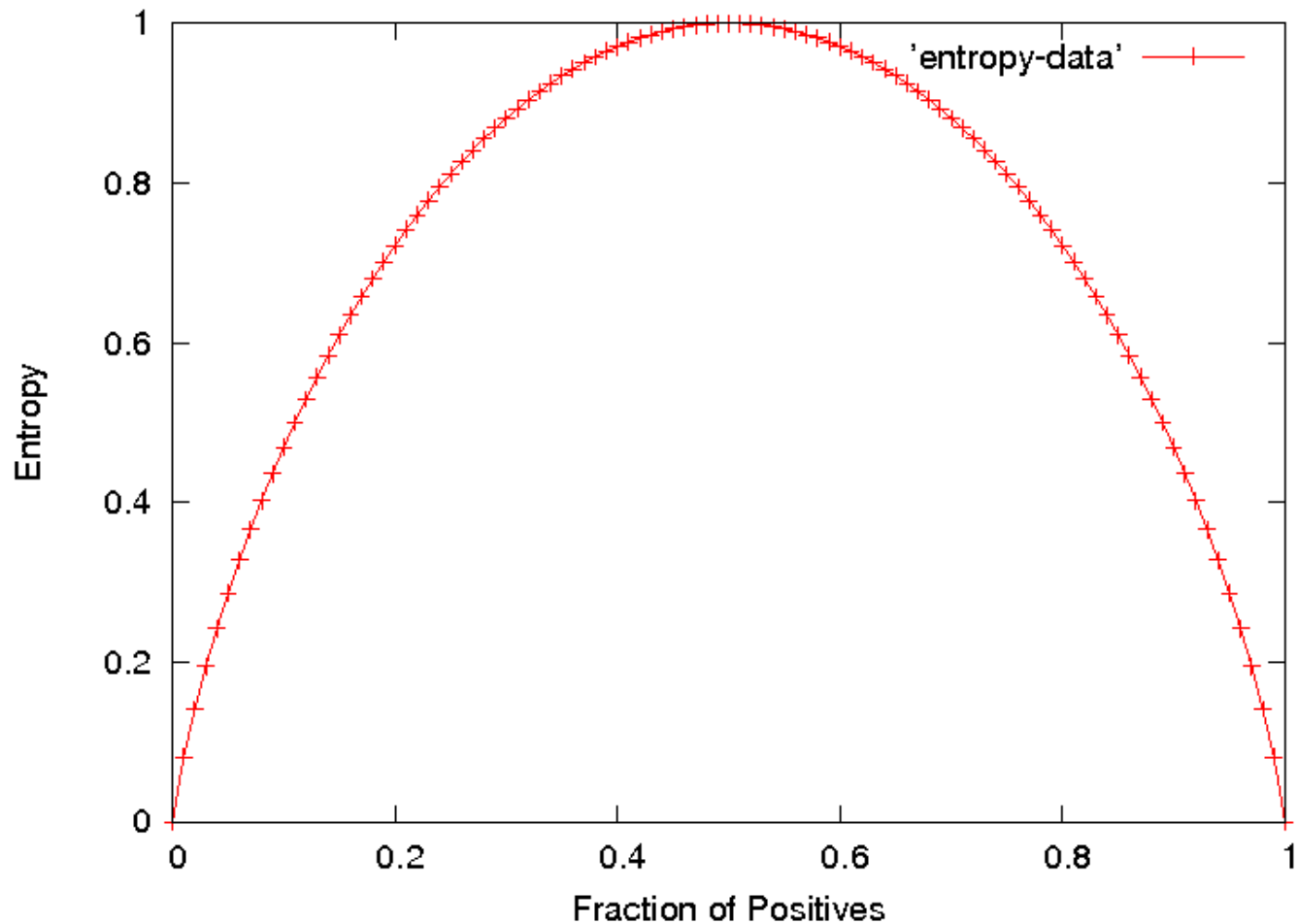
$$Entropy(S) = -(p_1 \log(p_1) + p_0 \log(p_0))$$

where p_1 is the fraction of positive examples in S and p_0 is the fraction of negatives.

- If all examples are in one category, entropy is zero (we define $0 \cdot \log(0) = 0$)
- If examples are equally mixed ($p_1 = p_0 = 0.5$), entropy is a maximum of 1.
- Entropy can be viewed as the number of bits required on average to encode the class of an example in S where data compression (e.g. Huffman coding) is used to give shorter codes to more likely cases.
- For multi-class problems with c categories, entropy generalizes to:

$$Entropy(S) = -\sum_{i=1}^c p_i \log(p_i)$$

Entropy Plot for Binary Classification



Information Gain

- The information gain of a feature F is the expected reduction in entropy resulting from splitting on this feature.

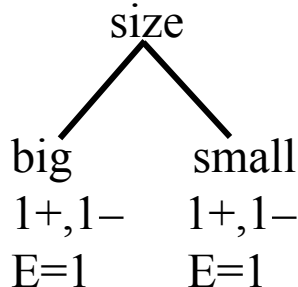
$$Gain(S, F) = Entropy(S) - \sum_v \frac{|S_v|}{|S|} Entropy(S_v)$$

where S_v is the subset of S having value v for feature F .

- Entropy of each resulting subset weighted by its relative size.
- Example:

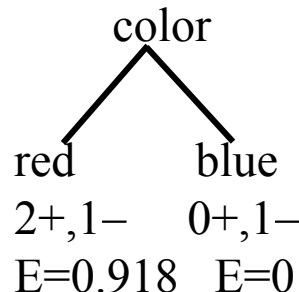
- $\langle \text{big, red, circle} \rangle$: + $\langle \text{small, red, circle} \rangle$: +
- $\langle \text{small, red, square} \rangle$: - $\langle \text{big, blue, circle} \rangle$: -

2+, 2- : E=1



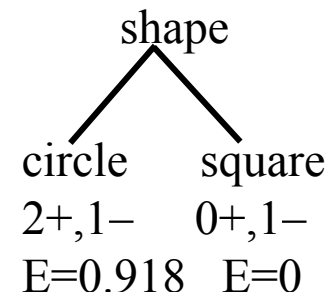
$$Gain = 1 - (0.5 \cdot 1 + 0.5 \cdot 1) = 0$$

2+, 2- : E=1



$$Gain = 1 - (0.75 \cdot 0.918 + 0.25 \cdot 0) = 0.311$$

2+, 2- : E=1



$$Gain = 1 - (0.75 \cdot 0.918 + 0.25 \cdot 0) = 0.311$$

Hypothesis Space Search

- Performs *batch learning* that processes all training instances at once rather than *incremental learning* that updates a hypothesis after each example.
- Performs hill-climbing (greedy search) that may only find a locally-optimal solution. Guaranteed to find a tree consistent with any conflict-free training set (i.e. identical feature vectors always assigned the same class), but not necessarily the simplest tree.
- Finds a single discrete hypothesis, so there is no way to provide confidences or create useful queries.

Bias in Decision-Tree Induction

- Information-gain gives a bias for trees with minimal depth.
- Implements a search (preference) bias instead of a language (restriction) bias.

History of Decision-Tree Research

- Hunt and colleagues use exhaustive search decision-tree methods (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed ID3 with the information gain heuristic to learn expert systems from examples.
- Simultaneously, Breiman and Friedman and colleagues develop CART (Classification and Regression Trees), similar to ID3.
- In the 1980's a variety of improvements are introduced to handle noise, continuous features, missing features, and improved splitting criteria. Various expert-system development tools results.
- Quinlan's updated decision-tree package (C4.5) released in 1993.
- Weka includes Java version of C4.5 called J48.

Overfitting Prevention (Pruning) Methods

- Two basic approaches for decision trees
 - **Prepruning**: Stop growing tree at some point during top-down construction when there is no longer sufficient data to make reliable decisions.
 - **Postpruning**: Grow the full tree, then remove subtrees that do not have sufficient evidence.
- Label leaf resulting from pruning with the majority class of the remaining data, or a class probability distribution.
- Method for determining which subtrees to prune:
 - **Cross-validation**: Reserve some training data as a hold-out set (*validation set*, *tuning set*) to evaluate utility of subtrees.
 - **Statistical test**: Use a statistical test on the training data to determine if any observed regularity can be dismissed as likely due to random chance.
 - **Minimum description length (MDL)**: Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.

Reduced Error Pruning

- A post-pruning, cross-validation approach.

Partition training data in “grow” and “validation” sets.

Build a complete tree from the “grow” data.

Until accuracy on validation set decreases do:

For each non-leaf node, n , in the tree do:

Temporarily prune the subtree below n and replace it with a leaf labeled with the current majority class at that node.

Measure and record the accuracy of the pruned tree on the validation set.

Permanently prune the node that results in the greatest increase in accuracy on the validation set.