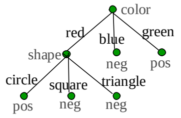
CS 391L: Machine Learning: Decision Tree Learning

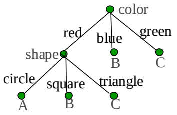
Raymond J. Mooney

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).
  - red ∧ circle → pos
  - red ∧ circle → A

    blue → B; red ∧ square → B

    green → C; red ∧ triangle → C

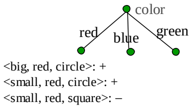
Properties of Decision Tree Learning

- 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. length < 3 and length ≥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.

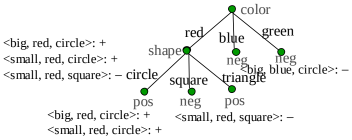
<big, red, circle>: + <small, red, circle>: +
<small, red, square>: – <big, blue, circle>: –



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:

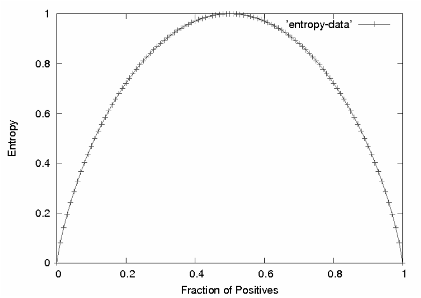
$$Entropy(S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

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·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:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

Entropy Plot for Binary Classification



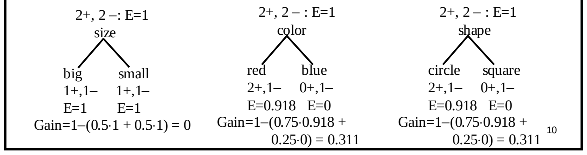
Information Gain

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

$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

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



2+, 2 –: E=1
size
big 1+,1– E=1
small 1+,1– E=1
Gain=1–(0.5·1 + 0.5·1) = 0

2+, 2 – : E=1
color
red 2+,1– E=0.918
blue 0+,1– E=0
Gain=1–(0.75·0.918 + 0.25·0) = 0.311

2+, 2 – : E=1
shape
circle 2+,1– E=0.918
square 0+,1– E=0
Gain=1–(0.75·0.918 + 0.25·0) = 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.
- Simulataneously, 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.

Weka J48 Trace 1

```
data> java weka.classifiers.trees.J48 -t figure.arff -T figure.arff -U -M 1
Options: -U -M 1
J48 unpruned tree
------------------
color = blue: negative (1.0)
color = red
|   shape = circle: positive (2.0)
|   shape = square: negative (1.0)
|   shape = triangle: positive (0.0)
color = green: positive (0.0)

Number of Leaves  :     5
Size of the tree :      7

Time taken to build model: 0.03 seconds
Time taken to test model on training data: 0 seconds
```

Weka J48 Trace 2

```
data> java weka.classifiers.trees.J48 -t figure3.arff -T figure3.arff -U -M 1
Options: -U -M 1
J48 unpruned tree
------------------
shape = circle
|   color = blue: negative (1.0)
|   color = red: positive (2.0)
|   color = green: positive (1.0)
shape = square: positive (0.0)
shape = triangle: negative (1.0)

Number of Leaves  :     5
Size of the tree :      7

Time taken to build model: 0.02 seconds
Time taken to test model on training data: 0 seconds
```

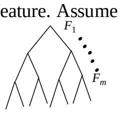
Weka J48 Trace 3

```
data> java weka.classifiers.trees.J48 -t contact-lenses.arff
J48 pruned tree
------------------
tear-prod-rate = reduced: none (12.0)
tear-prod-rate = normal
|   astigmatism = no: soft (6.0/1.0)
|   astigmatism = yes
|   |   spectacle-prescrip = myope: hard (3.0)
|   |   spectacle-prescrip = hypermetrope: none (3.0/1.0)

Number of Leaves  :     4
Size of the tree :      7

Time taken to build model: 0.03 seconds
Time taken to test model on training data: 0 seconds

=== Error on training data ===

Correctly Classified Instances          22               91.6667 %
Incorrectly Classified Instances         2                8.3333 %
Kappa statistic                      0.8447
Mean absolute error                      0.0833
Root mean squared error                  0.2041
Relative absolute error                 22.6257 %
Root relative squared error             48.1223 %
Total Number of Instances               24
```

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
  5  0  0 |  a = soft
  0  3  1 |  b = hard
  1  0 14 |  c = none


=== Stratified cross-validation ===

Correctly Classified Instances          20               83.3333 %
Incorrectly Classified Instances         4               16.6667 %
Kappa statistic                      0.71
Mean absolute error                      0.15
Root mean squared error                  0.3249
Relative absolute error                 39.7059 %
Root relative squared error             74.3898 %
Total Number of Instances               24


=== Confusion Matrix ===

  a  b  c   <-- classified as
  5  0  0 |  a = soft
  0  3  1 |  b = hard
  1  2 12 |  c = none
```

Computational Complexity

- Worst case builds a complete tree where every path test every feature. Assume *n* examples and *m* features.



Maximum of *n* examples spread across all nodes at each of the *m* levels

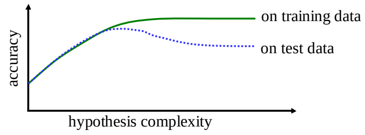
- At each level, *i*, in the tree, must examine the remaining $m-i$ features for each instance at the level to calculate info gains.

$$\sum_{i=1}^{m} i \cdot n = O(nm^2)$$

- However, learned tree is rarely complete (number of leaves is ≤ *n*). In practice, complexity is linear in both number of features (*m*) and number of training examples (*n*).

Overfitting

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - There may be noise in the training data that the tree is erroneously fitting.
  - The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.
- A hypothesis, *h*, is said to overfit the training data is there exists another hypothesis which, *h*´, such that *h* has less error than *h*´ on the training data but greater error on independent test data.

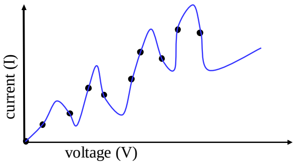


Overfitting Example

Testing Ohms Law: V = IR   (I = (1/R)V)

Experimentally measure 10 points

Fit a curve to the Resulting data.

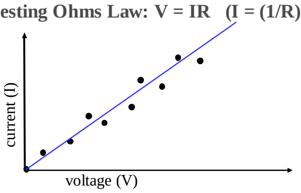


Perfect fit to training data with an 9th degree polynomial (can fit *n* points exactly with an *n*-1 degree polynomial)

Ohm was wrong, we have found a more accurate function!

Overfitting Example

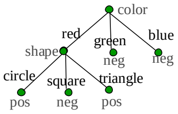
Testing Ohms Law: V = IR   (I = (1/R)V)



Better generalization with a linear function that fits training data less accurately.

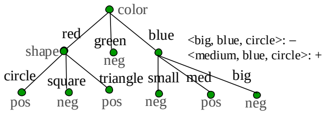
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



- Noise can also cause different instances of the same feature vector to have different classes. Impossible to fit this data and must label leaf with the majority class.
  - <big, red, circle>: neg (but really pos)
- Conflicting examples can also arise if the features are incomplete and inadequate to determine the class or if the target concept is non-deterministic.

Overfitting Prevention (Pruning) Methods

- Two basic approaches for decision trees
  - Prepruning: Stop growing tree as 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 dismisses 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.

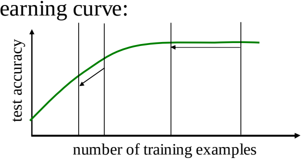
Reduced Error Pruning

- 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.
```

Issues with Reduced Error Pruning

- The problem with this approach is that it potentially “wastes” training data on the validation set.
- Severity of this problem depends where we are on the learning curve:



Cross-Validating without Losing Training Data

- If the algorithm is modified to grow trees breadth-first rather than depth-first, we can stop growing after reaching any specified tree complexity.
- First, run several trials of reduced error-pruning using different random splits of grow and validation sets.
- Record the complexity of the pruned tree learned in each trial. Let *C* be the average pruned-tree complexity.
- Grow a final tree breadth-first from all the training data but stop when the complexity reaches *C*.
- Similar cross-validation approach can be used to set arbitrary algorithm parameters in general.

Additional Decision Tree Issues

- Better splitting criteria
  - Information gain prefers features with many values.
- Continuous features
- Predicting a real-valued function (regression trees)
- Missing feature values
- Features with costs
- Misclassification costs
- Incremental learning
  - ID4
  - ID5
- Mining large databases that do not fit in main memory