Decision tree learning is a method that uses inductive inference to approximate a target function, which will produce discrete values. It is widely used, robust to noisy data, and considered a practical method for learning disjunctive expressions.

Appropriate Problems for Decision Tree Learning

Decision tree learning is generally best suited to problems with the following characteristics:

- Instances are represented by **attribute-value pairs**.
    - There is a finite list of attributes (e.g. hair colour) and each instance stores a value for that attribute (e.g. blonde).
    - When each attribute has a small number of distinct values (e.g. blonde, brown, red) it is easier for the decision tree to reach a useful solution.
    - The algorithm can be extended to handle real-valued attributes (e.g. a floating point temperature)
- The target function has **discrete output values**.
    - A decision tree classifies each example as one of the output values.
        - Simplest case exists when there are only two possible classes (**Boolean classification**).
        - However, it is easy to extend the decision tree to produce a target function with more than two possible output values.
    - Although it is less common, the algorithm can also be extended to produce a target function with real-valued outputs.
- Disjunctive descriptions may be required.
    - Decision trees naturally represent disjunctive expressions.
- The training data may contain errors.
    - Errors in the classification of examples, or in the attribute values describing those examples are handled well by decision trees, making them a robust learning method.
- The training data may contain missing attribute values.
    - Decision tree methods can be used even when some training examples have unknown values (e.g., humidity is known for only a fraction of the examples).

After a decision tree learns classification rules, it can also be re-represented as a set of if-then rules in order to improve readability.

Decision Tree Representation

A **decision tree** is an arrangement of tests that provides an appropriate classification at every step in an analysis.

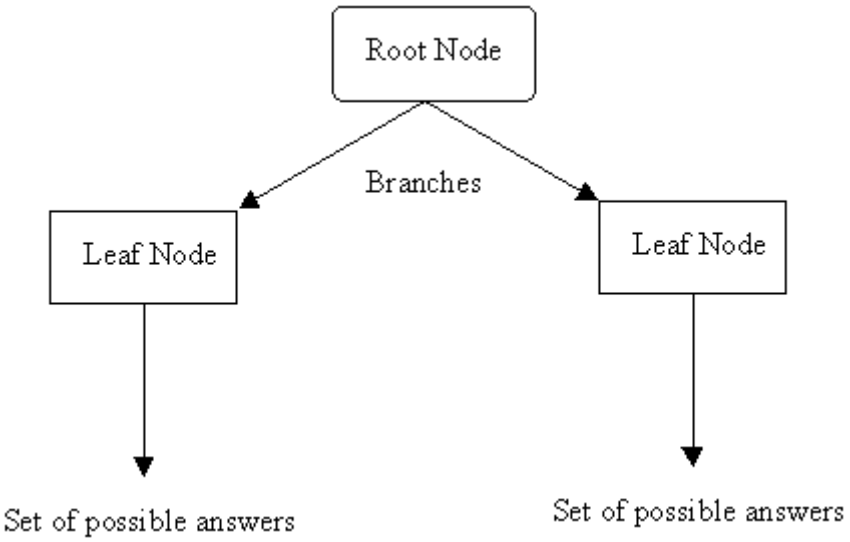
"In general, decision trees represent a disjunction of conjunctions of constraints on the attribute-values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions" (Mitchell, 1997, p.53).

More specifically, decision trees classify **instances** by sorting them down the tree from the **root node** to some **leaf node**, which provides the classification of the instance. Each node in the tree specifies a **test** of some **attribute** of the instance, and each **branch** descending from that node corresponds to one of the possible **values** for this attribute.

An instance is classified by starting at the root node of the decision tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated at the node on this branch and so on until a leaf node is reached.

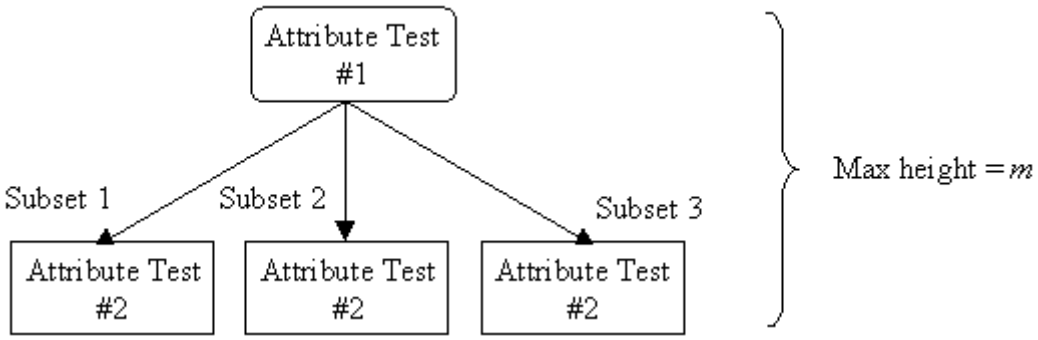
Diagram

- Each nonleaf node is connected to a test that splits its set of possible answers into subsets corresponding to different test results.
- Each branch carries a particular test result's subset to another node.
- Each node is connected to a set of possible answers.



Occam's Razor (Specialized to Decision Trees)

"The world is inherently simple. Therefore the smallest decision tree that is consistent
with the samples is the one that is most likely to identify unknown objects correctly." (



Given *m* attributes, a decision tree may have a maximum height of *m*.

Rather than building all the possible trees, measuring the size of each, and choosing the smallest tree that best fits the data, we use Quinlan's ID3 algorithm for constructing a decision tree.

Id3 Algorithm

From Data to Trees: Quinlan's ID3 Algorithm for Constructing a Decision Tree

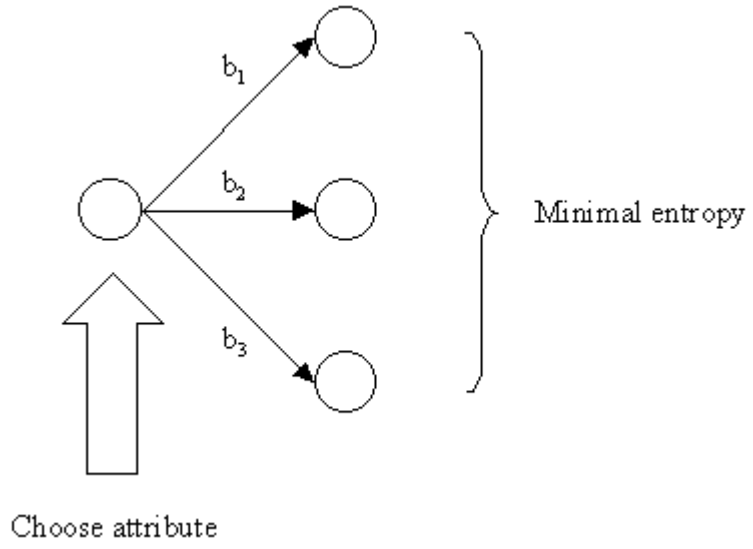
General Form

Until each leaf node is populated by as homogeneous a sample set as possible:

- Select a leaf node with an inhomogeneous sample set.
- Replace that leaf node by a test node that divides the inhomogeneous sample set into minimally inhomogeneous subsets, according to an entropy calculation.

Specific Form

1. Examine the attributes to add at the next level of the tree using an entropy calculation.
2. Choose the attribute that minimizes the entropy.



Tests Should Minimize Entropy

It is computationally impractical to find the smallest possible decision tree, so we use a procedure that tends to build small trees.

Choosing an Attribute using the Entropy Formula

The central focus of the ID3 algorithm is selecting which attribute to test at each node in the tree.

Procedure:

1. See how the attribute distributes the instances.
2. Minimize the average entropy.
    - Calculate the average entropy of each test attribute and choose the one with the lowest degree of entropy.

Review of Log_2

On a calculator:

$$\log_2(x) = \frac{\ln(x)}{\ln(2)} = \frac{\log_{10}(x)}{\log_{10}(2)}$$

eg) $(0)\log_2(0) = -\infty$ (the formula is no good for a probability of 0)
eg) $(1)\log_2(1) = 0$
eg) $\log_2(2) = 1$
eg) $\log_2(4) = 2$
eg) $\log_2(1/2) = -1$
eg) $\log_2(1/4) = -2$
eg) $(1/2)\log_2(1/2) = (1/2)(-1) = -1/2$

Entropy Formulae

Entropy, a measure from information theory, characterizes the (im)purity, or homogeneity, of an arbitrary collection of examples.

Given:

- n_b, the number of instances in branch b.
- n_{bc}, the number of instances in branch b of class c. Of course, n_{bc} is less than or equal to n_b
- n_t, the total number of instances in all branches.

Probability

$$P_b = \text{Probability an instance on a branch } b \text{ is positive}$$

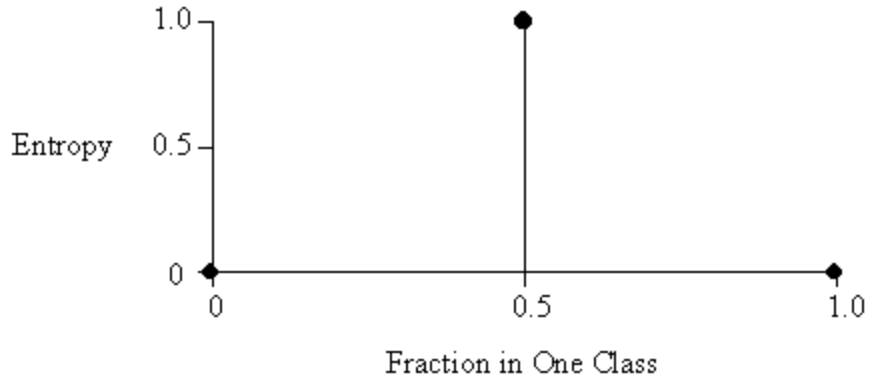
$$= \frac{\text{number of positive instances on branch}}{\text{total number of instances on branch}} = \frac{n_{bc}}{n_b}$$

- If all the instances on the branch are positive, then $P_b = 1$ (homogeneous positive)
- If all the instances on the branch are negative, then $P_b = 0$ (homogeneous negative)

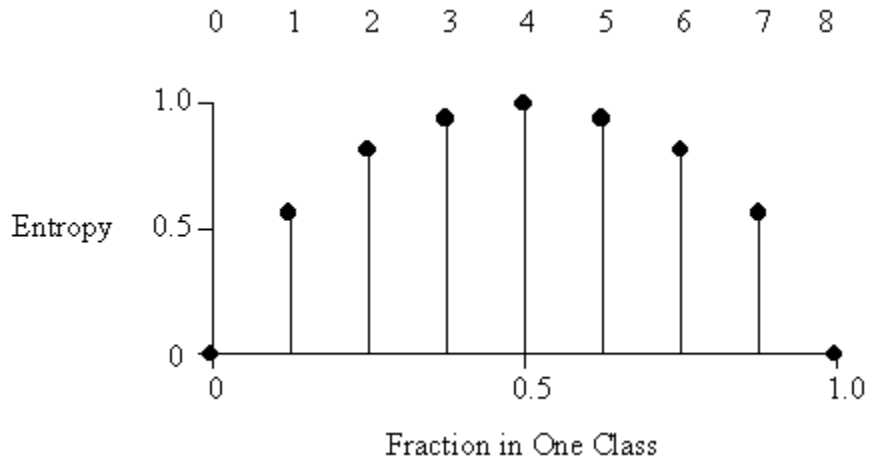
Entropy

- $$\text{Entropy} = \sum_c -\left(\frac{n_{bc}}{n_b}\right)\log_2\left(\frac{n_{bc}}{n_b}\right)$$

- As you move from perfect balance and perfect homogeneity, entropy varies smoothly between zero and one.
    - The entropy is zero when the set is perfectly homogeneous.
    - The entropy is one when the set is perfectly inhomogeneous.



The disorder in a set containing members of two classes A and B, as a function of the fraction of the set beloning to class A.
In the diagram above, the total number of instances in both classes combined is two.
In the diagram below, the total number of instances in both classes is eight.



Average Entropy

- $$\text{Average Entropy} = \sum_b \left(\frac{n_b}{n_t}\right) \times \left[\sum_c -\left(\frac{n_{bc}}{n_b}\right) \log_2 \left(\frac{n_{bc}}{n_b}\right) \right]$$

Click here for an exercise in decision tree construction.

Comments on the ID3 Algorithm

Unlike the version space candidate-elimination algorithm,

- ID3 searches a *completely* expressive hypothesis space (ie. one capable of expressing any finite discrete-valued function), and thus avoids the difficulties associated with restricted hypothesis spaces.
- ID3 searches *incompletely* through this space, from simple to complex hypotheses, until its termination condition is met (eg. until it finds a hypothesis consistent with the data).
- ID3's inductive bias is based on the ordering of hypotheses by its search strategy (ie. follows from its search strategy).
- ID3's *hypothesis space* introduces no additional bias.

Inductive bias of ID3: shorter trees are preferred over larger trees. Also, trees that place attributes which lead to more information gain (attributes that sort instances to most decrease entropy) are placed closer to the root of the tree.

ID3's capabilities and limitations stem from it's search space and search strategy:

- ID3 searches a *complete* hypothesis space, meaning that every finite discrete-valued function can be represented by some decision tree.
    - Therefore, ID3 avoids the possibility that the target function might not be contained within the hypothesis space (a risk associated with restriction bias algorithms).
- As ID3 searches through the space of decision trees, it maintains only a*single current hypothesis*.
    - This contrasts with the candidate-elimination algorithm, which finds *all* hypotheses consistent with the training data.
    - By learning only a single hypothesis, ID3 loses benefits associated with explicitly representing all consistent hypotheses.
        - For instance, it does not have the ability to determine how many decision trees that are consistent with the data could exist, or select the best hypothesis among these.
- In it's pure form, ID3 performs *no backtracking* in its search (**greedy algorithm**).
    - Once an attribute has been chosen as the node for a particular level of the tree, ID3 does not reconsider this choice.
    - It is subject to risks associated with hill-climbing searches that do not backtrack. It may converge to a locally optimal solution that is not globally optimal.
    - (Although the pure ID3 algorithm does not backtrack, we discuss an extension that adds a form of backtracking: *post-pruning* the decision tree. See decision rules.)
- At every step, ID3 refines it's current hypothesis based on statistical analysis of *all* training examples.
    - This differs from methods (such as candidate-elimination) that consider each training example individually, thus making decisions incrementally.
    - Using statistical properties of all the examples is what helps to make ID3 a robust algorithm. It can be made insenstive to errors in training examples and handle noisy data by modifying it's termination criteria to accept hypotheses that imperfectly fit the training data.

Another version

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING Although a variety of decision tree learning methods have been developed with somewhat differing capabilities and requirements, decision tree learning is generally best suited to problems with the following characteristics: Znstances are represented by attribute-value pairs. Instances are described by a fixed set of attributes (e.g., Temperature) and their values (e.g., Hot). The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (e.g., Hot, Mild, Cold). However, extensions to the basic algorithm (discussed in Section 3.7.2) allow handling real-valued attributes as well (e.g., representing Temperature numerically). The targetfunction has discrete output values. The decision tree in Figure 3.1 assigns a boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output

values. A more substantial extension allows learning target functions with real-valued outputs, though the application of decision trees in this setting is less common. 0 Disjunctive descriptions may be required. As noted above, decision trees naturally represent disjunctive expressions. 0 The training data may contain errors. Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples. 0 The training data may contain missing attribute values. Decision tree methods can be used even when some training examples have unknown values (e.g., if the Humidity of the day is known for only some of the training examples). This issue is discussed in Section 3.7.4. Many practical problems have been found to fit these characteristics. Decision tree learning has therefore been applied to problems such as learning to classify medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments. Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as classijication problems. The remainder of this chapter is organized as follows. Section 3.4 presents the basic ID3 algorithm for learning decision trees and illustrates its operation in detail. Section 3.5 examines the hypothesis space search performed by this learning algorithm, contrasting it with algorithms from Chapter 2. Section 3.6 characterizes the inductive bias of this decision tree learning algorithm and explores more generally an inductive bias called Occam's razor, which corresponds to a preference for the most simple hypothesis. Section 3.7 discusses the issue of overfitting the training data, as well as strategies such as rule post-pruning to deal with this problem. This section also discusses a number of more advanced topics such as extending the algorithm to accommodate real-valued attributes, training data with unobserved attributes, and attributes with differing costs. CHAPTER 3 DECISION TREE LEA