Decisions trees are similar to the game 20 questions, but the questions don’t have to be yes/no. The questions form a tree structure, with the first question being the starting node. Each leaf of the tree represents a label for the input feature vector. The particular set of questions in the tree depends on the form of the feature vector and the training data.

- A decision tree is a set of rules organized in a tree structure with a root node and leaves.
- Each node is a classification rule that subdivides the data set into two or more parts.
- Each leaf is associated with an output rule.
- To classify a novel instance, begin at the root node and use the results of each classification rule to get to a leaf node and its output rule.

Decisions trees can use many different types of classification rules.

- Enumerated types: one branch for each type, or one branch for each of a set of mutually exclusive subsets of the categories. If there is one branch for each type, the variable will not be used again lower down in the tree.
- Numeric types:
  - A simple threshold test of a variable with a constant value: two branches.
  - An interval test with different branches for above and below the interval: three branches.
  - Comparisons between different variables.
- Missing data:
  - Make a special branch in the tree for the missing data category.
  - Use the most popular branch for real data given the training set.
  - Split the instance and send it down multiple branches, then recombine the outputs using the likelihood of each branch given the training set.
Each leaf of a decision tree represents a single path through the tree from the root node. The collection of rules along the path form a single rule we could write as a (potentially complex) if statement. If we were to write down a rule for each leaf node, we would end up with a set of classification rules. We can take a novel instance and implement the same function as the decision tree using the set of leaf rules. One important characteristic of this set of rules is that the order in which they are applied to the instance is immaterial. Every instance will meet the criteria of one and only one rule in the set.

The set of if-then rules in a decision tree subdivides the problem space into many small pieces. The expectation is that, if the features are related to the desired output classes, then it should be possible to subdivide the space so the small pieces are close to uniform. There are many forms of decision trees. The key parameter of a decision tree is its size, or complexity. A more complex decision tree divides the input space into many smaller pieces, giving it the ability to specify complex and precise boundaries. However, a simpler decision tree may often capture the decision boundary more accurately and better generalize to new data.

When building decision trees, we are always trying to choose what question to ask next. When picking a question, there are two factors we want to measure. First, we would like to maximize how much we learn from the question, regardless of whether the answer is yes or no. One way of thinking about the information gain is by considering how much of the input space we discard, depending on the answer. The best we can do is discard half of the possible input space no matter what the answer is.

Second, we want the likelihood of the correct class to increase in the input space that remains. In other words, we are both subdividing the problem and making it more likely that if we make a guess about the output class that we'll be correct. The training set provides us with data with which we can evaluate different questions and choose what is the best question to ask next. The training set also tells us when a particular subset of the input space is likely to have a single output class, in which case we don’t have to add more nodes to that part of the decision tree.

**Information Content** is a way of measuring how many more nodes, or questions, we are likely to need to complete a branch of a decision tree. The fewer additional nodes we need, the better the original question. A branch that is close to pure (only one output class) is likely to need fewer questions to complete. A branch that has training samples with approximately equal numbers of two output classes is likely to need more questions in order to separate the two classes. The best possible question would separate the training data into one branch per output class, in which case the tree requires no additional questions and is complete.

Information theory tells us that **entropy** is the function that best captures the idea of information content. Entropy is defined as in (1).

\[
E(\vec{x}) = \sum_{i=1}^{F} -p(x) \log_2(p(x))
\]  

(1)

The entropy of a branch of a decision node is the sum of the entropies of each output class. For example, consider a branch that contains training examples from two output classes, 6 from one class and 4 from another. The probability of each output class is 0.6 and 0.4, respectively.

\[
E([0.6, 0.4]) = -0.6 \log_2(0.6) - 0.4 \log_2(0.4) = 0.971
\]  

(2)
Now that we can measure the information content in a single branch, we can calculate the information content of all of the branches of a decision node. Multiplying each branch by the probability the branch will be taken provides the total information content of the node. A node whose branches contain zero information content has pure branches: there is only one output class in each branch, so there is no need to ask any more questions.

The information gain due to a split at a node is the difference between the entropy of the branch data coming into the node and the weighted sum of the entropies of the branches leaving the node. For example, consider a node receiving 20 training samples of two classes that divide evenly as \([10, 10]\). The question at the node divides the data into 3 branches \([6, 7, 7]\). Within each branch, the division into two classes breaks down as \([(4, 2), (5, 2), (1, 6)]\).

The information content in the incoming data collection is
\[
E([10, 10]) = -0.5 \log_{0.5} -0.5 \log 0.5 = 1.0.
\]
The information content in the outgoing stream is given by the entropy of the outgoing streams multiplied by their likelihood in the training set.

\[
IC = \frac{6}{20} E([4, 2]) + \frac{7}{20} E([5, 2]) + \frac{7}{20} E([1, 6])
\]
\[
= (6 \times 0.918 + 7 \times 0.863 + 7 \times 0.591) / 20
= 0.785
\]  

The information gain is the difference in the incoming and outgoing branches. In this case, the information gain is \(1.0 - 0.785 = 0.215\). Consider the same data, but a question that divides it into two branches with the subdivision into classes as \([(9, 1), (1, 9)]\).

\[
IC = \frac{10}{20} E([9, 1]) + \frac{10}{20} E([1, 9])
\]
\[
= (10 \times 0.469 + 10 \times 0.469) / 20
= 0.469
\]

The alternative split would provide an information gain of 0.531, or more than twice the information gain of the first branching. Therefore, the second branching is likely a better choice.

These notes are adapted from those of Bruce Maxwell.