1 Decision Trees

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 (and is said to require “low information” – its information content is low). 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 (so its information value is high). 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 measures information content. Entropy is defined as in Eq. (1).

\[
E(\bar{c}) = \sum_{k=1}^{C} -p(\vec{c}k) \log_2(p(\vec{c}k))
\] (1)

where \(\vec{c}\) is a row matrix (in the context of decision trees, it specifies the “class counts” of the node). \(p(\vec{c}k)\) is the probability that a new data point is in the \(k^{th}\) class. It is simply the fraction of data points (that are “in this node”) that are in the \(k^{th}\) class.

Consider a node that has training samples evenly spread between two class, e.g. the class values list is [5, 5]. Its entropy is 1. We still need more branching.
Consider a node that has training samples all in the same class, e.g. the class values list is \([0, 10]\). Its entropy is 0. No new information is needed. If we get to this node in the tree, then we are confident we can classify the point that got us here.

To build a decision tree, we want each child node to have less information than its parent. To evaluate the effectiveness of a particular branching strategy, we compare the entropy of the parent to the weighted average of the entropies of the children (think of this as the output information, or “\text{out}_\text{info}”). We want the parent’s entropy to be higher than that of its children (as we progress down the tree, regardless of which branch we take, we want to lose the need for information and thereby increase our confidence in our ability to classify accurately). In decision tree parlance, this difference is called \textbf{information gain} and we want it to be as high as possible.

\[
\text{information gain} = E - \text{out}_\text{info} \tag{2}
\]

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 is
\[
E([10, 10]) = -0.5 \log_{0.5} -0.5 \log 0.5 = 1.0.
\]

The information content in the outgoing branches is given by the entropy of the child nodes multiplied by their likelihood in the training set.

\[
\text{out}_\text{info} = \frac{6}{20} E([4, 2]) + \frac{7}{20} E([5, 2]) + \frac{7}{20} E([1, 6])
\]
\[
= (6 \times 0.918 + 7 \times 0.863 + 7 \times 0.591) / 20
\]
\[
= 0.785 \tag{3}
\]

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 \(([9, 1], [1, 9])\).

\[
\text{out}_\text{info} = \frac{10}{20} E([9, 1]) + \frac{10}{20} E([1, 9])
\]
\[
= (10 \times 0.469 + 10 \times 0.469) / 20
\]
\[
= 0.469 \tag{4}
\]

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.

\section*{1.1 Numeric attributes}

There are many ways to go about making a decision node when the data is numeric:

\begin{itemize}
  \item Fix the number of branches (e.g. 2), choose one feature to examine, and then use the information gain of the possible divisions to determine where best to split the data. (This is the one we will use.)
\end{itemize}
• Fix the number of branches and use a clustering algorithm (K=2) to select the division point. This has the benefit of separating the data in a meaningful way, but doesn’t necessarily do anything to improve the classification performance.

• Calculate the average value of the variable for each output class and pick decision points based on the means and standard deviations of the distributions.

All of these approaches are approximately equal in terms of expense. In all cases, every data value must be examined.

In lab, we will use the first method and will fix the number of branches to 2. This means we need to test every possible threshold to determine which one splits the data in a way that will minimize the \( \text{out}_\text{info} \).

### 1.2 Algorithm for a 1R Tree

To make a 1R (1 rule) tree, calculate a rule for each attribute and pick the best one. In some cases, a complex decision tree is not warranted. For many tasks, there may be a single variable in the data set that acts as an effective proxy for the desired output classes. A 1R tree has a single decision node. In some cases, a 1R tree works as well or better than other classifier methods, especially for small training sets.

The information gain is a useful method of selecting which 1R tree to use. Consider an example data set with 3 features. There are three possible 1R trees. Each feature has a threshold divides the data set into two parts. The initial information content of the data set is \([9, 11] = 0.993\). The information gain for each of the 1R trees is given in table 1. From the table, feature three is clearly the best option and provides the highest information gain.

<table>
<thead>
<tr>
<th>Category</th>
<th>L Branch</th>
<th>L IC</th>
<th>H Branch</th>
<th>H IC</th>
<th>Information Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5, 2</td>
<td>0.863</td>
<td>4, 9</td>
<td>0.890</td>
<td>0.112</td>
</tr>
<tr>
<td>2</td>
<td>4, 3</td>
<td>0.985</td>
<td>5, 8</td>
<td>0.961</td>
<td>0.024</td>
</tr>
<tr>
<td>3</td>
<td>8, 3</td>
<td>0.845</td>
<td>1, 8</td>
<td>0.503</td>
<td>0.302</td>
</tr>
</tbody>
</table>

Table 1: Information gain of 1R trees for a given data set

These notes are adapted from those of Bruce Maxwell.