1 Decision Trees (cont.)

1.1 Numeric attributes

Numeric attributes are common features in data, but so far we have examined only enumerated attributes in building decision trees. To make use of numeric attributes we have to determine a method of separating the data into two or more branches. There are many ways to go about making a decision node.

- Fix the number of branches (e.g. 2) and then use the information gain of the possible divisions to determine where best to split the data.
- 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.

One difference between numeric attributes (features) and categorical ones is that a numeric attribute may be used in several different nodes within a branch. Since categorical outputs are all the same in a sub-branch, there is no further need for subdivisions.

1.2 Algorithms

1.2.1 1R

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 is 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. For the example data set, there are three possible 1R trees. Each dimension 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, category 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 from example data set

1.2.2 Simple Trees

- Build a node for each attribute and calculate the information gain
- Pick the node with the largest gain and insert it into the tree
- Any branch with zero information content gets a class label
- Keep adding nodes until every leaf node has a class label

While the above procedure creates a perfectly fine decision tree, it does not balance generalization with specificity. The best decision tree algorithms use additional information to choose when to subdivide a node, and they prune branches from trees once they are completed. Otherwise, every single detail in the training data will contribute to the tree, whether or not it is relevant to the task.

It is important to note two attributes of the process.

- The decision about whether to build a node, and what question to ask is part of a local search process, or a greedy search process. That means the tree is built through a series of decisions that do not take into account global information about the structure of the tree, but consider only the immediate value of a question on the components of the training set going through that branch.

- The decision process itself is heuristic, which means there is no provably correct method of picking a decision tree node. Experience and experimentation have yielded an algorithm that generally performs well within a greedy search process. However, the free lunch theorem suggests that certain parameters of the tree building process should be evaluated when building a tree for a particular task.
1.3 Subtleties

1.3.1 Too many branches

One problem with using information gain directly is that it tends to prefer decision trees that subdivide the task into many small pieces. A method often used to counter this tendency is to express the information gain as the ratio of the node’s information gain to the entropy of how the data gets split by the question (intrinsic information). The entropy of the data split is simply the information content of the subdivision of the data into the branches regardless of class. A question that subdivides all of the incoming training samples into equal parts has a high entropy. A question that sends a large number of samples into one branch and few into another has a low entropy. Using the ratio of information gain to intrinsic information favors nodes that produce fewer divisions, but balances that with the ability of the question to subdivide the task in a useful manner.

The intrinsic information in the 3-way split is $E(6, 7, 7) = 1.718$. The intrinsic information in the 2-way split is $E(10, 10) = 1$. Therefore, even if the two questions produced similar information gains, the two-way split gives a smaller denominator and should be preferred because it gives a larger information gain ratio.

In some cases the gain ratio overcompensates, so a heuristic is to select the node with the largest gain ratio, so long as its information gain is at least as big as the average information gain for all nodes considered. That way a node with a small information gain, but a large gain ratio, does not dominate over a node that has a higher than average information gain, but not as equal a division of the data.

**Incorporating the subtlety:** Instead of using the information gain in our algorithms, we may want to use the gain ratio.

1.3.2 Missing values

Missing values can present problems when building a decision tree. As noted earlier, one solution to missing attributes is to send the instance down two branches and then combine the results of the resulting leaves using a weighted average. It is possible to do the same thing when building the decision tree by using non-integer branch weights when calculating the information gain for a node. For sub-trees after a split decision, the training sample still contributes via its likelihood of going down that branch.

These notes were adapted from those of Bruce Maxwell.