1 Structure and Design

The decision tree classifier is implemented using the **iterative dichotomizer 3** algorithm, which is outlined in the lecture slides as well as the course textbook. This implementation is primarily based on the pseudo-code given by Russel and Norvig's chapter on learning decision trees in *Artificial Intelligence: A Modern Approach*.

The `ID3` class defines the inner class `Dataset`, which is an internal representation of the training data and provides useful methods such as `entropy()` and `splitByAttribute()`. Its role in the design is to abstract most of the illegible array manipulation logic that was otherwise ubiquitous in the implementation of the core ID3 algorithm. As a result of this addition, the training algorithm is much more legible, making it easier to modify as well as check for correctness. The `Dataset` object is constructed prior to training, from the string matrices produced by the provided method `indexStrings()`. The design could be improved by modifying `indexStrings()` to produce a `Dataset` object directly, rather than use the default string matrices as an intermediate representation.

The implementation is written in Java 8 and requires the `jdk 1.8.x` to compile; this is available on the ITL machines. Furthermore, the coursework specification is somewhat ambiguous on how the classifier should produce its output; however, the only reasonable solution is printing to standard output.

1.1 Training

The `train()` method wraps a call to the recursive `id3()` method, which constructs the decision tree using the pre-defined `TreeNode` data structure. The method recurses on increasingly small subsets of the original dataset with an increasingly reduced set of attributes to split the dataset on. At each level of recursion, four cases can be encountered:

1. All examples in the remaining dataset have the same class (base case)
2. There are no more attributes to split the dataset by (base case)
3. There are no examples in this subdivision of the dataset (base case)
4. None of the above apply (recursive case)

The implementation bundles the last two cases together, checking for empty subdivisions when they are created, rather than when being recursively called on one. This way it is not necessary to pass a node's majority class as an argument to the recursive call (see lines 158-161).

1.2 Classification

Classification is carried out using the stored `TreeNode` data structure. Each node in this data structure has a `value` attribute; for inner nodes this represents the attribute to check, while for leaf nodes it represents the classification result. The `classify()` method recursively traverses the decision tree, at each node recursing on the child that matches the example's value for the current node's attribute, until a leaf node is reached.

1.3 Testing

The implementation was tested on the provided simple tests, as well as for modified version of the real estate dataset. These modified datasets covered cases such as the presence of more than two classes, the presence of only a single class, datasets that cannot be perfectly classified, and incomplete datasets. In the case of datasets that could not be perfectly classified, decision trees with obvious opportunities for pruning were observed.