Student Name: Robin Farrow-Yonge
Student Number: 160719011

Artificial Intelligence Coursework 1 - Report

The process that the ID3 algorithm, and by extension my program, follows in order to classify a dataset involves building a decision tree using a pre-classified training set, and then navigating that tree in order to classify another as-yet-unclassified dataset. The way that it goes about this is to calculate the total entropy (based on the class attribute of each row) of the dataset as a whole, then to split the dataset into subsets based on an individual attribute's potential values and calculate the entropy of those subsets. The attribute that produces subsets with the lowest entropy is then selected and made the value of the current node of the tree, and a child node is created for each potential value of that attribute. The process then repeats, running on each of the resulting subsets, until it reaches a leaf node, which is ideally when the entropy of a subset is zero, but can just be when there are no longer any attributes to split the dataset on.

In my implementation of the algorithm I tried to adhere to the style of the skeleton code in order to best make use of the predefined instance variables. The primary effect that this had on my code was when I was splitting the dataset into subsets. Where I could have removed the column I was splitting on from all the arrays that tracked the data, I instead tracked which attributes had already been used with a string array named checkList that on the initial call of my growTree method was cloned from the first row of the data array. After the dataset was split on an attribute I changed the appropriate string to "checked" and passed this edited version of the array along with the subset and child node to the recursive growTree call. I then just added a method to check if every attribute had been checked (isChecked) and made that a condition of the block that identifies leaves.

Another more minor design decision I made was to separate my growTree and climbTree methods from the train and classify methods. This gave me a space in which to instantiate any variables that I needed to pass, including the checkList and the root decisionTree node in the case of the train method and to use a for loop to classify the test data row by row in the classify method.

For testing, I ran the code against both the provided datasets, and some of my own. Using an online data generation tool, I created simple datasets with three classes, one class, and two sets that were not fully classifiable, one with three and one with two classes. This flagged a problem in my code, in which I hadn't provided for cases where a subset created from the training data contained no data. The fix for this was to pass the original dataset instead of the empty subset, as well as a checkList with every attribute marked as checked, so as to force the creation of a leaf node.

There are some elements of the code could do with a rewrite. I think I could simplify things, particularly in instances where I could have used more modern java features to condense things down. I have also used Math.abs to force the calcEntropy function to return a positive value to account for cases where it returned -0.0. This seems hacky, and I'm not entirely content with it, but it works without issue and is much shorter than the alternatives so it makes sense to keep it in.