1 A high-level description

1.1 Decision tree nodes

A private Node class was created for use in the decision tree. Nodes had the following properties:

- Data Set: The dataset is stored as an array list of strings. Each string is 23 characters long, with the 0th char represented the class, and the 1 through 22nd chars representing the attributes.

- Attribute: Whenever a Node is created, this property is initially set to zero. After the split function is called, Attribute gets set to be an int corresponding to the 'best' attribute to split over. IMPORTANT: If the node is a leaf node (data set is pure), then A is set to be either 99 (poisonous) or 100 (edible).

- Value: The value property was set to "root" for the root node. For all other nodes, the property is defined as the attribute value (i.e. 's' for spicy).

- Children: The children were stored as an ArrayList of Nodes.

1.2 Build method

The build method uses a total of 5 helper methods. The code for the build method itself is very simple. It begins by calling split, a function which will determine which attribute is best to split over. If the data set is pure, split will return 99 for poisonous and 100 for edible.

A loop travels through the dataset creating a second dataset (D’) with all instances of each attribute value. A new node is created with this dataset, and build is called recursively.

- HELPER METHOD 1: isPure(D) This method returns -1 if the data set is not pure. If the data set contains only mushrooms with class p, then 99 is returned. If class e, then 100 is returned.

- HELPER METHOD 2: split(D, A, option) The split method first tests to see if the
data set is pure (see above). It then loops through all the attribute and calls the gain function and returns the attribute yielding the best results.

− HELPER METHOD 3: gain(D, A, option) This method returns the gain from the data set it is passed for the specified attribute using entropy or misclassification error based on the specified option.

− HELPER METHOD 4: entropy(D) Calculates entropy of the dataset.

− HELPER METHOD 5: misClass(D) Calculates the misclassification error of the dataset.

1.3 Prune method

A node is a candidate for pruning only if all of it’s children are leaf nodes. We check to see if the node has a non-leaf child. If it does, then we recursively call prune on each of the nodes children. If it does not, then it is a candidate for pruning and we call the snip helper method on it.

After we have called prune on all of the nodes children, we must check to see if the node has a non-leaf child a second time. If it does not then we call snip on this node.

− HELPER METHOD 1: snip(Node, alpha) This method calculates the chi square test statistic according to the formula given in Quinlan’s paper (stored in a variable called chi). We then check to see if chi is less than a critical value, which depends on degrees of freedom and the significance level which is passed in as a parameter.

− HELPER METHOD 2: chi(A, alpha) This method computes the critical value from the 2 parameters. It contains a vector called df which holds the number of attributes for each of the 23 attributes. It also contains a matrix with all possible chi square density values based on degrees of freedom and the 3 alpha levels. Recall that alpha = 1 - Confidence Level.

− HELPER METHOD 3: getP(D) For clarity, this code has been put into its own function. It simply returns the number of poisonous mushrooms in a dataset. (We defined poisonous to be the ‘positive’ attribute for chi square calculations.)

− HELPER METHOD 4: getN(D) Identical to getP, but it calculates the number of edible mushrooms in a data set. (edible was defined to be the ‘negative’ attribute)

1.4 Classify method

The classify method will create a file called output.txt. This method utilizes two helper methods getP and getN which have been described above.

The strategy is to create a pointer node called curr, which traverses the tree for each mushroom in the dataset until it hits a leaf node, and prints either ‘p’ or ‘e’ to the output file according to the decision made by the tree.

There is a flag variable, which is used to make sure curr only travels one level at a time. A while loop checks to see if curr is pointing to a leaf node, if it doesn’t, it moves down a level based on the mushrooms attributes.
In the training set, many of the attribute values are eliminated by the time we split on a given node. In the entropy case for example, we split on attribute 3 (cap-color) in the 4th (and final) level. By the time we reach this point, only 4 values of cap-color are possible (c,n,w,y). It is possible however, for a mushroom from the training set to have a cap color other than these 4. In this case, the decision tree makes a prediction based on the dominant class in the dataset.

### 1.5 Missing values

Missing values were handled according to a suggestion in Quinlan’s paper. We opted for the simplest option of replacing missing values with the dominant class.

### 2 Proposed or validated folklore rules

1. If a mushroom has green spore-print-color, it is probably poisonous!
2. If it smells like almonds, bon apetite!
3. If it is odorless and lightly colored (white or yellow), stay away!

### 3 Conclusion & Discussion

Since the training set was so large, the trees built by our algorithm were highly effective in classifying the training set. So effective in fact, that the pruning method (while correct), was unnecessary. All 8 of our trees classified 100% of the testing data correctly.

The tree’s shown below, are the product of the two different tree building methods. As discussed above (classification), non-leaf nodes sometimes have more values possible than what is listed on the tree. This results from weeding out these values from other attributes based on the training set. Read classification section for information on how any special cases were handled.

In the end, all of our tree’s classify all mushrooms correctly, so we decided that our ‘best’ tree is the simplest one. So the output file is produced from the entropy based tree with no pruning (alpha = 100), which yields 29 nodes with a max depth of 5.
Figure 1: Decision Tree by Entropy

Figure 2: Decision Tree by Misclassification Error