**Can be done in teams of two.**

**Due: Thursday, February 12th UPDATE: 11:59PM Tuesday, February 17th**

1. (10 points) Be able to parse a data set in csv format (elements are separated by , with one instance per row) with the class attribute as the rightmost column of the data set. The first row will contain attribute names. UPDATE (20150210) The second row will contain attribute type descriptors:

C - Categorical data in this column. I will only test data sets with around 10 nominal values at most. (Might be 11, might be 15, so don't hard code your solution!) The values might look like integers, but you should treat them as strings and only allow for equals and not equals comparisons with values from attributes of this type.

N - Numeric data in this column. There will only be numbers, they might be integers or doubles, treat them all as doubles. If there are more than 10 attribute values, please discretize this data down to 10 bins. As discussed in class, for this type of data you will binary split into all values <= some amount and all values > than that amount (O(m) possible split conditions versus O(2^m) from the nominal case above).

An example of this type of data can be found in the file anneal.data.csv (attached). Your program should be able to parse this sort of data and build a tree as described below. UPDATE 201502312: I meant to mention that this particular data set happens to have missing values in it, represented by '?', and "not applicable" answers represented by either '-' or '\_'. You can group all three of these characters together as their own class value when parsing. NOTE: I will not have any missing attribute values when grading these assignments.

2. (15 points) Implement the Gini split measure, Information Entropy (with SplitInfo), and Misclassification rate. These will be used in the tree induction step next.

3. (45 points) Induce a decision tree according to the ID3 algorithm described in Table 1. of the Incremental Decision Trees paper in the required reading section. This is very much like Hunt's algorithm from class. Allow the user to specify on the command line (gini|info|error) which purity measure to use in place of "E-score". In addition to the stopping criterion listed in Table 1, the tree induction algorithm should also stop when either there are no remaining attributes to split on or the selected E-score does not decrease below a user defined threshold. To handle continuous attributes, discretize them before attempting to induce the tree using either equal width or equal frequency binning and 10 bins. Build only binary trees. UPDATE 20150210 ONLY use equals/not equals conditions when searching for the best split on Categorical data (i.e., do NOT search for conditions like <, >, etc.). For Numeric attributes, use the <,> types of searches.

4. (5 points) Be able to print the tree out in text. Use a depth first traversal of the tree and for each node print out the attribute name, as you follow an edge print out the test condition, and at a leaf node print out the class label. Each time an edge is followed down the tree, a | should be printed out to indicate the tree depth. See the example output at the end of this assignment for an example (probably easier that reading this description)!

5. (10 points) Implement the Minimum Description Length measure for determining tree complexity. UPDATE 20150210 As described in the lecture slides, if you have n attributes, m instances, and k distinct class labels, you can compute MDL by summing up log2(n) bits for each inner node, log2(k) bits for each leaf node, and log2(m) + log2(k) bits for each misclassified instance.

6. (15 points) How good is your tree growing algorithm? Pick a data set from the UCI repository ( https://archive.ics.uci.edu/ml/datasets.html?format=&task=cla&att=mix&area=&numAtt=&numIns=&type=&sort=nameUp&view=table ). Partition your selected data set into two parts: assign 20% of it to the test set and use the rest of the data as the training set. Try 5 different threshold values, and rank the trees in order of increasing Minimum Description Length. For the five trees (each corresponding to one of the threshold values), also compute the generalization error on the 20% of data that was held out, and the re-substitution error. Show the results in a table. Discuss the results, was MDL a good predictor of the generalization error? How does re-substitution error compare with the generalization error? Other considerations? Feel free to use more than 5 threshold values to help with your analysis (you may also notice some interesting trends).

~~Extra Credit (50 points): Implement the ID5R algorithm discussed in the paper. With the same parameter values and the same data, do you build the same tree? Repeat the experiments done in the previous step but now repeat it as you grow the tree (do the measurements as you stream more samples to the model; record the results after every 5 samples). What trends do you notice?~~

**Submission Instructions:**  
Submit only 1 assignment per group. The submission must be a zip (or tgz) file containing all source code used in the project, a makefile if c/c++ was used, and a README.txt file containing the names of both group members, the name of the program entry point (the file containing the main method or equivalent), the name of the data set you used, the 5 threshold values you tried, and a list of any known bugs of the program (i.e., Part 4 doesn't work right if ... ). I must be able to test the program from a terminal by calling the executable and passing in appropriate arguments. The first argument must be the training data, the second argument must be the testing data, the third argument must be the purity measure to use (gini info or error), the fourth is the threshold value.

./tree.exe "data/trainingdata.csv" "data/testingdata.csv" gini 0.15

or

java -cp bin cs691.assignment.Driver "data/trainingData.csv" "data/testingData.csv" info 0.2

Example output of such a program should be something like:

Re-subsitution error: 0.1%

Generalization error: 3.2%

MDL: 12.34 bits

 petalwidth = [0.0]  
 | sepallength = [2.0]  
 | | sepalwidth = [8.0]: 0.0 (0)  
 | | sepalwidth = [5.0, 6.0, 7.0]  
 | | | sepalwidth = [5.0]: 0.0 (1)  
 | | | sepalwidth = [6.0, 7.0]  
 | | | | sepalwidth = [6.0]: 0.0 (2)  
 | | | | sepalwidth = [7.0]  
 | | | | | petallength = [0.0]: 0.0 (1)  
 | | | | | petallength = [1.0]: 0.0 (0)  
 | sepallength = [1.0, 0.0, 4.0, 3.0]