Part II: Programming assignment. 70 points.

Implement the ID3 decision tree learning algorithm that we discussed in class. You may use C, C++, Java, C#, Python, or other languages to implement the algorithm. To simplify the implementation, your system only needs to handle **binary classification tasks** (i.e., each instance will have a class value of 0 or 1). In addition, you may assume that **all attributes have categorical values (not continuous)** and that there are **no missing values** in the training or test data.

Sample training files (`train-*.dat`) and test files (`test-*.dat`) can be found from the course homework page. http://www.hlt.utdallas.edu/~yangl/cs6375/homework/hw1/
The first line holds the attribute names, each name followed by an integer representing the number of possible values for that attribute. Each following line defines a single example. Each column holds this example's value for the attribute given at the top of the file (same order). The last column holds the class label for the examples. In all of the following experiments, you should use this last class attribute to help train the tree and to determine whether a tree classifies an example correctly.

In a decision tree, if you reach a leaf node but still have examples that belong to different classes, then choose the most frequent class (among the instances at the leaf node). If you reach a leaf node in the decision tree and have no examples left or the examples are equally split among multiple classes, then choose the class that is most frequent in the ***entire*** training set.
You do **not** need to implement pruning.

IMPORTANT: Your program should take only **<u>two</u>** arguments to be specified in the command line invocation of your program: <u>a training file and a test file</u>. There should be no graphical user interface (GUI). Any program that does not conform to the above specification will receive no credit. Also, don't forget to use logarithm base 2 when computing entropy and set $0 \log_2 0$ to 0.

(A). Build a decision tree using the training instances and print to `stdout` the tree in the same format as the example tree shown below (the symbol after colon denotes the class label for a leaf node).

```
attr1 = 0 :
| attr2 = 0 :
| | attr3 = 0 : 1
| | attr3 = 1 : 0
| attr2 = 1 :
| | attr4 = 0 : 0
| | attr4 = 1 : 1
attr1 = 1 :
| attr2 = 1 : 1
|
```

(B). Use the learned decision tree to classify the **training** instances. Print to `stdout` the accuracy of the tree. (In this case, the tree has been trained *and* tested on the same data set.) The accuracy should be computed as the percentage of examples that were correctly classified. For example, if 86 of 90 examples are classified correctly, then the accuracy of the decision tree would be 95.6%.

```
Accuracy on training set (90 instances): 95.6%
```

(C) Use the learned decision tree to classify the **test** instances. Print to `stdout` the accuracy of the tree. (In this case, the decision tree has been trained and tested on different data sets.)

```
Accuracy on test set (10 instances): 60.0%
```

(D). Now, we want to investigate how the amount of training data affects the accuracy of the resulting decision tree. Plot a **learning curve** (i.e., a graph of the accuracy of your algorithm on the test set against different training set sizes) by re-training your learning algorithm using training set sizes of 50, 100, 150, 200, ... Briefly comment on the shape of the curve. Does it exhibit the usual properties of a learning curve? You need to create your training sets with different sizes to obtain the learning curve.

Grading Criteria

The programming portion of this assignment will be graded on both correctness and documentation.

Correctness. 65 points will be based on the correctness of your decision tree program. We will likely run your program on a new data set to test your code, so we encourage you to do the same!

Documentation. 10 points will be based on the documentation accompanying your source code. We expect each source file to contain a paragraph or two at the beginning to describe the contents of that file. The main program should describe the functionality of the program: the type of input it expects, the type of output it produces, and the function that it performs. The data structures used in the program must also be clearly described. The code should be modular. Do provide in-line comments to explain any code that is unusual or potentially confusing.

Additional Notes

When reporting accuracy, two decimal places are sufficient. When making graphs,
a) remember to label each axis and to provide a title that indicates what the graph is depicting;
b) "zoom in" on the range of values (e.g., if your numbers vary from 80 to 100%, then show that range, instead of 0-100%, which throws away detail).

What to Submit

Please submit separate files for the written problems and programming ones.

For the programming assignment, you should submit **via eLearning**

(i) your source code,
(ii) a README file that contains instructions for **compiling** and **running** your program, as well as the platform (Windows/Linux/Solaris) on which you developed your program,
(iii) the learning curve for part (D).

You will receive **zero credit** for your program if (1) we cannot figure out how to run your program from your README file or (2) your program takes more than two input arguments.