CS 4375 Introduction to Machine Learning
Fall 2016
Assignment 1: Decision Tree Induction
Part I: *Due electronically by Tuesday, September 6, 11:59 p.m.*
Part II: *Due electronically by Tuesday, September 13, 11:59 p.m.*

Note: 1. Your solution to this assignment must be submitted via eLearning.
2. Whenever possible, you should provide brief justifications for your solution.
3. You may work in a group of two or individually.

Part I: Written Problems (25 points)

1. **Representing Boolean Functions (10 points)**

   Give decision trees to represent the following concepts:

   (a) $(\neg A \vee B) \wedge \neg(C \wedge A)$. Your decision tree must contain as few nodes as possible.

   (b) $(A \oplus B) \wedge C$

2. **Decision Trees (15 points)**

   Spam has become an increasingly annoying problem for e-mail users. In this problem we are interested in using the ID3 decision tree induction algorithm to automatically determine whether or not an e-mail is a spam based on whether the words "nigeria", "viagra", and "learning" appear in the e-mail. Below are the instances from which our decision tree will be learned. Note that a word has the value 1 if and only if it is present in the corresponding e-mail.

No.	nigeria	viagra	learning	Class
1	1	0	0	1
2	0	0	1	1
3	0	0	0	0
4	1	1	0	0
5	0	0	0	0
6	1	0	1	1
7	0	1	1	0
8	1	0	0	1
9	0	0	0	0
10	1	0	0	1

   Using these descriptions as training instances, show the decision tree created by the ID3 decision tree learning algorithm. Show the information gain calculations that you computed to create the tree. Be sure to indicate the class value to associate with each leaf of the tree and the set of instances that are associated with each leaf.

Part II: Programming (75 points)

Implement the ID3 decision tree learning algorithm that we discussed in class. 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 are binary-valued (i.e. the only possible attribute values are 0 and 1) and that there are no missing values in the training or test data.

Some sample training files (`train.dat`, `train2.dat`) and test files (`test.dat`, `test2.dat`) are available from the assignment page of the course website. In these files, only lines containing non-space characters are relevant. The first relevant line holds the attribute names. Each following relevant line defines a single example. Each column holds an example's value for the attribute named at the head of the column. The last column (labeled "class") holds the class label for the examples. In all of the following experiments, you should use this last class attribute to train the tree and to determine whether a tree classifies an example correctly.

When building a decision tree, if you reach a leaf node but still have examples that belong to different classes (i.e., the node is not pure), 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. Do **not** implement pruning.

IMPORTANT:

- You may use C, C++, C#, Python, or Java to implement the ID3 algorithm. If you want to use other programming languages, please contact your dear TA, Isaac Persing (`persingq@hlt.utdallas.edu`), for approval first.
- Your program should be able to handle **any** binary classification task with **any** number of binary-valued attributes. Consequently, both the number and names of the attributes, as well as the number of training and test instances, should be determined at runtime. In other words, these values should **not** be hard-coded in your program.
- Your program should allow only two arguments to be specified in the command line invocation of your program: a training file and a test file. There should be no graphical user interface (GUI) of any kind. Any program that does not conform to the above specification will receive no credit.
- Use logarithm base 2 when computing entropy and define $0 \log_2 0$ to be 0.
- In the input files, only lines containing non-space characters are relevant, as mentioned previously. In particular, empty lines may appear anywhere in an input file, including the beginning and the end of the file. Care should be taken to skip over these empty lines.

Your Tasks

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.

```
wesley = 0 :
| honor = 0 :
| | barclay = 0 :  1
| | barclay = 1 :  0
| honor = 1 :
| | tea = 0 :  0
| | tea = 1 :  1
wesley = 1 :  0
```

According to this tree, if `wesley = 0` and `honor = 0` and `barclay = 0`, then the class value of the corresponding instance should be 1. In other words, the value appearing before a colon is an attribute value, and the value appearing after a colon is a class value.

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%. (Note that the accuracy on the training instances will be 100% if and only if the training instances are consistent.)

```
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 100, 200, 300, ..., 800. Briefly comment on the shape of the curve. Does it exhibit the usual properties of a learning curve? (We suggest that you plot the graph using Excel, but if you choose to draw the graph by hand, you need to scan it so that you can submit it online. We will not accept hardcopy submissions.)

Grading Criteria

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

Correctness. 70 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. 5 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.

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 relevant 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

Your should submit **via eLearning** (i) your **source code**, (ii) a README file that contains clear instructions for **compiling** and **running** your program (as well as the platform (Windows/Linux/Solaris) on which you developed your program), and (iii) the learning curve for (d). Do **not** turn in any executables generated from your source code. Each group should hand in a single copy of the code. The names of all the members of the group should appear in the README file. Again, you will receive **zero credit** for your program if (1) we cannot figure out how to compile and run your program from your README file, (2) we cannot find your source code in the submission directory, or (3) your program takes more or less than two input arguments.

For all of the assignments in this course, if you are not happy with your submission, you can re-submit as many times as you want before the submission deadline. Submissions in the late submission period are possible only if you have not submitted anything before the submission deadline, and unlike in the regular submission period, you can only submit once in the late submission period.