CS 4375 Introduction to Machine Learning
Fall 2021
Assignment 1: Decision Tree Induction
Part I: *Due electronically by Thursday, September 9, 11:59 p.m.*
Part II: *Due electronically by Friday, September 17, 11:59 p.m.*

Instructions:

1. Your solution to this assignment must be submitted via eLearning.
2. For the written problems, submit your solution as a **single PDF** file.
   - Only use **blue or black pen** (black is preferred). Scan your PDF using a **scanner** and upload it. Make sure your final PDF is **legible**. **Regrades due to non-compliance will receive a 30% score penalty**.
   - Verify that both your answers and procedure are **correct, ordered, clean, and self-explanatory** before writing. Please ask yourself the following questions before submitting:
     - Are my answers and procedure legible?
     - Are my answers and procedure in the same order as they were presented in the assignment? Do they follow the specified notation?
     - Are there any corrections or scratched out parts that reflect negatively on my work?
     - Can my work be easily understood by someone else? Did I properly define variables or functions that I am using? Can the different steps of my development of a problem be easily identified, followed, and understood by someone else? Are there any gaps in my development of the problem that need any sort of justification (be it calculations or a written explanation)? Is it clear how I arrived to each and every result in my procedure and final answers? Could someone describe my submission as messy?
3. **You may work individually or in a group of two**. Only one submission should be made per group. If you work in a group, **make sure to indicate both group members when submitting** through eLearning.
4. **IMPORTANT:** As long as you follow these guidelines, your submission should be in good shape; if not, we reserve the right to penalize answers and/or submissions as we see fit.

Part I: Written Problems (30 points)

1. **Decision Tree Induction (20 points)**

   Consider the following dataset. Each instance is annotated with information that indicates whether or not a professor's course is worth taking given a description of the professor's personality, the difficulty of the course, the associated reviews on RateMyProfessors, and how easy it is to get an A in the course. A "+" indicates it is worth taking; a "−" indicates

it is not. Using these descriptions as training instances, show the decision tree that the ID3 decision tree learning algorithm would create to predict whether a particular professor's course is worth taking. 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.

No.	Personality	Difficulty	RMP Reviews	Easy A	Worth Taking
1	Hilarious	Low	Awesome	No	$-$
2	Boring	Medium	Awful	No	$+$
3	Boring	Low	Awesome	No	$+$
4	Hilarious	Low	Awesome	Yes	$-$
5	Hilarious	Low	Average	Yes	$-$
6	Hilarious	Medium	Awful	No	$+$
7	Hilarious	High	Awful	No	$-$
8	Hilarious	Medium	Awesome	No	$+$
9	Hilarious	Medium	Average	Yes	$-$
10	Hilarious	High	Awful	Yes	$-$
11	Boring	Medium	Awesome	Yes	$-$
12	Boring	Low	Average	Yes	$+$
13	Boring	High	Awesome	No	$-$
14	Boring	High	Average	Yes	$-$

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

   Give decision trees to represent the following concepts:

   (a) **(5 pts)** $A \vee (B \wedge C)$

   (b) **(5 pts)** $(A \wedge B) \vee (C \wedge D)$

Part II: Programming (70 points)

Implement the ID3 decision tree learning algorithm that we discussed in class. To simplify the implementation, your system only needs to handle ternary classification tasks (i.e. each instance will have a class value of 0, 1, or 2). In addition, you may assume that all attributes are ternary-valued (i.e. the only possible attribute values are 0, 1, and 2) 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, 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, then choose the class that is most frequent in the *entire* training set. If you reach a left node in the decision tree and there is a tie for the most frequent class, then break ties among them by preferring the one that is more frequent in the *entire* training set. If two or more classes are equally frequent in the entire training set, then break ties by preferring class 0 to class 1 and preferring class 1 to class 2. Do **not** implement pruning.

A word on **tie breaking**: when choosing attributes using information again, if two or more attributes achieve the highest information gain value, break ties by choosing the earliest one in the list of attributes (assuming that the attributes in the first line of a training file are read in a left-to-right manner).

IMPORTANT:

- You must use C++, Java, or Python to implement the ID3 algorithm. Do **not** use any non-standard libraries in your code, and make sure your program can compile on UTD machines, not just your own machines.
- Your program should be able to handle **any** ternary classification task with **any** number of ternary-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 :  2
| | barclay = 2 :  1
| honor = 1 :
| | tea = 0 :  0
```

```
| | tea = 1 :  1
| | tea = 2 :  2
wesley = 1 :
| barclay = 0 :  2
| barclay = 1 :  1
| barclay = 2 :  0
wesley = 2 :  1
```

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 on `train.dat` 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

Your program will be graded based on the correctness of your decision tree program. We will run your program on new data sets to test your code, so we encourage you to do the same!

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, (3) your program takes more or less than two input arguments, or (4) the two input arguments cannot be specified via the command line.

General Notes on Submission

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.

Free late days can be used to submit an assignment late **before** the end of the late submission period without any late penalty. They are **not** meant to give you additional time; in particular, they won't allow you to submit after the late submission period ends. The late submission period always ends 48 hours after the official due date. For instance, if Part I of Assignment 1 was due on Spetember 1, then using free late days would enable you to submit your assignment until September 3 (the end of the late submission period) without late penalty, but nothing can be submitted after September 3 regardless of whether you apply your free late days. At most two free late days can be applied to each part of each assignment.