CS 540-1: Introduction to Artificial Intelligence
Homework Assignment #2

Assigned: 9/23
Due: 10/5 before class

Hand in your homework:

This homework assignment includes written problems and programming in Java. Hand in hardcopy of the requested written parts of the assignment in class. All pages should be stapled together, and should include a cover sheet on top of which includes your name, login, class section, HW #, date, and, if late, how many days late it is. Electronically hand in files containing the Java code that you wrote for the programming part. See course Web page for instructions.

Late Policy:

All assignments are due **at the beginning of class** on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 11 a.m, and it is handed in between Wednesday 11 a.m. and Thursday 11 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline. A total of two (2) free late days may be used throughout the semester without penalty.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: [16] Mutual Information

Consider the following Automobile data with two features `Displacement`, `Horsepower` and label `MPG`.

Displacement	**Horsepower**	**MPG**
Low	Low	Good
Medium	Low	Good
High	Medium	Bad
Low	High	Bad
Low	Medium	Bad
Medium	Medium	Bad
High	Medium	Good
Low	High	Bad

a) [4] What is the entropy of `MPG`?
b) [4] What is the mutual information between `Displacement` and `MPG`?
c) [4] What is the mutual information between `Horsepower` and `MPG`?
d) [4] Draw the full decision tree. When there is a tie in majority vote, default to "Good."

Question 2: [10] Greediness of Decision Tree

Can you fill in the blanks in the following dataset with two binary features A, B, and one binary label Y, such that the mutual information at the root of the decision tree for both features is zero:

$I(A; Y)=0$

$I(B; Y)=0$

but the full decision tree perfectly predicts Y?

A	B	Y
		0
		0
		1
		1

Question 3: [14] SVM

Let there be two labeled items in 1D: **x_1=0, y_1= −1; x_2=1, y_2=1** (Note the **y**'s are labels, not a second feature). Solve the SVM optimization problem, i.e., find the value of *w* and *b* by hand:

$$\min_{w,b} \quad \|w\|^2$$
$$\text{subject to} \quad y_i(w^\top x_i + b) \geq 1, \quad i = 1, 2$$

Be sure to show your steps.

Question 4: [60] Building Decision Trees in Java

For this part of the assignment, you are required to implement in Java the decision tree construction algorithm given in Figure 18.5 on page 702 of the textbook. The main class must be called HW2.java

We have provided a dataset to help in program creation and for model evaluation. Download it from the course website. This dataset gives information about the passengers on the Titanic and whether they survived or not. The task is: given the details about each passenger, classify whether they **survived** or not.

Your program may be tested on a different dataset unrelated to the provided domain. Thus, your program must be able to learn different decision trees based on a given training dataset.

You should create a **HW2** class with the following calling convention:

```
java HW2 <modeFlag> <trainFilename> <tuneFilename> <testFilename>

      modeFlag is a number from 0 to 4 – parts (a) to (d) have
details
            0 – print mutual info for each possible question at root
node
            1 – print decision tree
            2 – print predictions on the test set
            3 – print pruned decision tree
            4 – print predictions on the test set using pruned tree
      trainFilename - the file containing the training set
      tuneFilename – the file containing the tuning set
      testFilename - the file containing the test set
```

To eliminate ambiguities in the algorithm defined in Figure 18.5 of the text book, in the case of a tie in majority vote (`PLURALITY-VALUE`), return the first listed class label in the training file. In the case of a tie of mutual information values when selecting the best attribute, choose the attribute that appears first in the training file. The children nodes should be printed in the same order of the attribute values listed in the training file.

File format

All data files (training, tuning, test) will contain a list of classes and attribute values, followed by the actual data. Attributes and classes are always discretely valued. A line that begins with a double slash `//` is a comment and should be ignored. In other lines, elements will be comma-separated. For a line beginning with `%%`, it contains two possible classes. Each line that begins with `##` specifies the name and all possible discrete values of one attribute. The order of successive attributes is important as this is the same order used in each of the examples in the file. Following this header information, each line contains one example. The first element is the class label of that example. All other elements in the line from left to right are the values of the ordered list of attributes. Note that all white space in the line should be ignored. The following is a sample file for a small Titanic dataset (for illustration purposes only, not used in this homework):

```
//----------------------------------------------------------
//example data file
//classification of passengers into survived=yes or no
//input features are:
//PTYPE: 1st class=1st, 2nd class=2nd, 3rd class=3rd, crewmember
= crew
//AGE: <18=child, >=18=adult
//SEX: female or male
%%,yes,no
##,PTYPE,1st,2nd,3rd,crew
##,AGE,child,adult
##,SEX,female,male
yes,1st,adult,male
no,1st,child,female
no,3rd,child,female
no,crew,adult,male
//etc.
//----------------------------------------------------------
```

For parts (a) and (b), you will only need the training set. For parts (c), you will use both the training and test sets. For part (d), use all three data files.

Your program should have the following functionality for building, using, and understanding decision trees:

a) [15] Each non-leaf node will have an associated single attribute. You must use *mutual information* to select the best attribute. If `modeFlag`=0, print out the mutual information for each possible attribute that could be used at the root node. Format: each line has the name of an attribute and the mutual information, separated by space. For example, for the Titanic dataset, the output might be (the numbers are fictitious):

```
PTYPE 0.123

AGE 0.456

SEX 0.248
```

b) [15] Use the training set to build a decision tree. Print out the decision tree if `modeFlag`=1. Use simple indented ASCII text to indicate a node's level in the tree during a depth-first tree traversal. Here is what the printout should look like for the tree that corresponds to Fig. 18.6 in the book:

```
Root {Patrons?}
   None (No)
   Some (Yes)
   Full {Hungry?}
        No (No)
        Yes {Type?}
              French (Yes)
              Italian (No)
              Thai {Fri/Sat?}
                    No (No)
                    Yes (Yes)
              Burger (Yes)
```

c) [15] Ignore the class label of each example in the test set (when we test your program, we might put "?" or some random string there). When `modeFlag`=2, classify each example in the test set using your tree. Output ONLY the class label of each example, one per line. DO NOT produce any other text with the hand-in version of your program when `modeFlag`=2 (so that we can automatically grade your output). Make sure you have exactly the same number of lines of output as the number of test examples. Note that we may use a different test set than the one provided to you.

d) [15] When `modeFlag`=3 or `modeFlag`=4, use the tuning set to prune the decision tree. Build the decision tree *T* as before using the whole training set. After the tree is built, perform pruning. This is how you perform one iteration of pruning: For each internal node *i* in the trained tree *T*, collapse the subtree at node *i* into a leaf node. You need to gather all training examples that were under that subtree, perform a majority vote among those examples, and use the majority vote label as the predicted label for the new leaf node. This results in a tree T_{-i}. Compare the accuracies of trees $T, T_{-1}, \ldots, T_{-m}$ on the tuning set, where *m* is the number of internal nodes (note: when you construct T_{-2}, you start from *T* again, not T_{-1}). Replace *T* with the best tree among these. Perform multiple iterations of pruning as described above. Each iteration starts with the best tree created at the previous iteration. Iterate until the tuning accuracy no longer increases.

If `modeFlag`=3, print the final pruned tree. If `modeFlag`=4, use the final pruned tree to classify the test set. Again, print only one class label per line.

Programming Tips

The Java classes `BufferedReader` and `StringTokenizer` can be helpful for parsing the data files. The Java class `ArrayList` is also a nice built-in data structure in Java. `ArrayList` is part of the `List` structure that is automatically resized when you insert objects. As any normal `List` structure, if the get method is called to recover any element in the `ArrayList`, you must cast it base to its original class before you can use it. Also, remember that $\log_2(x) = \log_b(x) / \log_b(2)$ for any base b, and $\log(x/y) = \log(x) - \log(y)$ for any base. Of course, you are welcome to ignore these tips and implement all your classes without using any of these suggestions.