CS540: Introduction to Artificial Intelligence
Homework assignment #1: Decision Trees

Assigned: September 10, 2008
Due: September 24, 2008

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. All pages should be stapled together, and should include a cover sheet on top which includes your name, login, class title, HW #, date, and, if late, how many days late it is. Electronically hand in files containing the Java code that you wrote for the assignment (see course Web page for instructions. ASSIGNMENT_NAME is HW1).

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 an assignment is due on a Wednesday and it is handed in between Wednesday 11 a.m. and Thursday 11 a.m., a 10% penalty 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, TA, 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 or help 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.

1. I have three coins: a quarter, a dime, and a penny. I believe they talk to each other because I can use one to predict the others. Here is my proof. I flip the three coins at once and record their HEADS(H) or TAILS(T) outcomes. I repeat this eight times and the table below shows the outcomes.

   a. [3] What is the entropy of my penny, as computed from the 8 outcomes in the "penny" column?
   b. [9] From the table, what is the information gain between my quarter and penny? (In other words, what is the information gain of my quarter given the set of examples with class label defined by penny?) What is the information gain between dime and penny?
   c. [9] Manually create the full decision tree, where the attributes are quarter and dime, to predict the outcome of penny. Show the tree. At each internal node, show the information gain of ALL candidate questions available at that node. In case of ties, please use the following tie-breaking rules:
      i. For class label majority vote ties, always prefer the class "T".
      ii. For attribute ties, always prefer the attribute listed first in the table.
   d. [3] What is the training set accuracy of your tree on penny?
   e. [6] Can my quarter and dime REALLY predict the penny? What does the decision tree say? Briefly explain what's going on.

Flip	Quarter	Dime	Penny
1	H	H	T
2	T	T	H
3	T	H	T
4	H	T	H
5	T	H	T
6	T	H	T
7	H	H	H
8	H	T	H

2. [10] After learning decision trees from you at a tailgate party, your friend John thinks there is a bug in the decision tree building algorithm. John points out that a node should stop splitting whenever all candidate questions have zero information gain there. (For this problem, all questions are of the form "What's the value of an attribute?") Do you agree with John? If you agree, briefly justify your answer. If you do not agree, design a dataset by filling in the binary (values T or F) attributes A and B below to prove John wrong. Be sure to justify your answer. Use the same tie –breaking rules as in Question 1.

A	B	Outcome Y
		T
		T
		F
		F

3. Building decision trees in Java.
   a. [45] For this part of the assignment, you are required to implement in Java the decision tree construction algorithm given in Figure 18.5 of the textbook for the case where there are *two classes only*. The main class must be called HW1.java

      We have provided a dataset to help in program creation and for model evaluation. Download it from the course website. This dataset is based on mushroom records drawn from the Audubon Society Field Guide to North American Mushrooms. The task is: given a description of a mushroom classify it as **edible** or **poisonous**.

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

      You should create a **HW1** class with the following calling convention:

```
java HW1 <printTreeFlag> <trainFilename> <tuneFilename> <testFilename>
      printTreeFlag is 1 or 0.
            1 – print tree, don't print predictions on the test set
            0 – don't print tree, print predictions on the test set
      trainFilename - the file containing the training set
      tuneFilename – the file containing the tuning set, or NONE
      testFilename - the file containing the test set
```

      For part (a), we will not use a tuning set (tuneFilename=NONE). Your program should have the following functionality:

      i. Use the training set to build a decision tree. Candidate questions are limited to testing the value of a single attribute. You must use information gain to find the best question.

      ii. Print out the decision tree if printTreeFlag=1 (but do not print the class predictions on the test set). 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)
```

iii. 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 printTreeFlag=0, 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 handin version of your program when printTreeFlag=0 and tuneFilename=NONE (this is because we want to 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.

iv. To eliminate ambiguities in the algorithm defined in Figure 18.5 of the text book, in the case of a tie in majority vote (MAJORITY-VALUE), return the first listed class label in the training file. This label is also the default (as required by the algorithm in Figure 18.5). In the case of a tie when selecting the best question, 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.

v. 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 discrete valued. A line 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.

```
//----------------------------------------------------------
//example data file
//classification of mushrooms into edible=e or poisonous=p
//input features are:
//cap-shape: bell=b,conical=c,convex=x,flat=f,knobbed=k
//odor: almond=a,anise=l,creosote=c,fishy=y,foul=f
%%,e,p
##,cap-shape,b,c,x,f,k
##,odor,a,l,c,y,f
e,b,a
p,c,a
p,k,y
//etc.
//----------------------------------------------------------
```

b. [15] When tuneFilename is not NONE, it contains the name of a separate tuning set. Build the decision tree *T* as before using the training set. After the tree is built, 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}, …, T_{-n} on the tuning set, where *n* 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.

   If printTreeFlag=0, print this new tree. Use this new tree to classify the test set. Again print only one class label per line.

c. [Extra Credit, 10] When tuneFilename ends with "!" (for example "my_tuning!"), the actual tuning file name does not include the character "!" (e.g., "my_tuning"). Perform multiple iterations of pruning as described in part (b). Each iteration starts with the best tree created at the previous iteration. Iterate until the tuning accuracy no longer increases.

   Hand in a hard copy of the following table. Do not print it in the code you hand in. Each row is for an iteration, and refers to the best (pruned) tree in that iteration, with the following format:

   Iteration, training set accuracy, tuning set accuracy, test set accuracy

   If printTreeFlag=1, print the final pruned tree. If printTreeFlag=0, use this 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 based *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 tips and hints.