CS 662
Artificial Intelligence Programming
Homework #6
Decision Trees
Due: Tuesday, 11/19, 10am

Summary
In this assignment, you will be writing code to construct a decision tree, and then using it to classify instances from several different datasets. The decision tree algorithm is recursive; the amount of code needed is not huge, but there's some mental hurdles needed to completely understand what's going on.

I have provided some skeleton code to get you started and guide you through the implementation. You are welcome to make any changes that you like, but please think carefully before doing so; the code is designed to help make the decision tree easy to implement.

Datasets:
There are two types of datasets for this assignment.

- Toy datasets:
  - The tennis dataset
  - The restaurant dataset

  These are both useful for testing your code; they're small, and you know what the correct answers are.

- "Real" datasets.
  - Breast Cancer data. This dataset contains medical records from a large number of women who have had breast cancer. Based on their characteristics, we would like to predict whether they will have a recurrence event.
  - Lymph node data. This data contains the medical records for the examinations of a large number of patients' lymph nodes. Based on these records, we would like to predict their medical condition. (Normal, metastasized cancer, malign nodes only, or fibrosis in the node.) (NOTE: I hope to have this data set ready in a few days – it is not yet in Canvas).
  - Nursery school data. This data set contains nursery school applications for a large number of parents. Based on characteristics about the parents, we would like to predict whether the child should be admitted to nursery school.

  These datasets will be much more interesting for evaluating the performance of your decision tree. (More on that below.)

Code:

I've provided you with an updated version of readARFF.py. It contains a new function: getAttrList, which takes as input the attribute list and returns an ordered list of all attribute names - in the same order as the data columns. This will be helpful for several of the methods below.

A hint: list comprehensions are very helpful for this assignment. Often, you'll need to pull out

one or more columns from the data. So, for example, to get a list containing only the third column in a dataset where the last element is equal to some item 'x', you could do:

```
third = [d[2] for d in data if d[-1] == 'x']
```

Assignment:

1. **(15%)** The decision tree is easiest to build in a bottom-up fashion. To begin, we'll need a method to compute entropy. It should take as input a list of attribute values, such as ['weak', 'strong', 'weak', 'weak'] and return a float indicating the entropy in this data. I've provided a function stub for you.
2. Next, we'll want to compute remainder. This will tell us, for a given attribute, how much information will remain if we choose to split on this attribute. I've written this one for you.
3. **(17%)** Once we know how to compute remainders, we need to be able to select an attribute. To do this, we just compute the remainder for each attribute and choose the one with the smallest remainder. (this will maximize information gain.) The function selectAttribute should take as input a list of lists, with each list being an instance. I've provided a stub for you.

   We're now ready to think about building a tree. A tree is a recursive data structure which consists of a parent node that has links to child nodes. I've provided a TreeNode class for you that does this. (You don't need a separate Tree class.)

   The TreeNode has the following data members:
   - attribute: for non-leaf nodes, this will indicate which attribute this node tests. For leaves, it is empty.
   - value. For leaf nodes, this indicates the classification at this leaf. For non-leaf nodes, it is empty.
   - children. This is a dictionary that maps values of the attribute being tested at this node to the appropriate child, which is also a TreeNode.

   It also has methods to print itself and to test whether it is a leaf.

4. **(25%)** So we need a method that can build a tree. We will call this makeTree. It should take as input a dataset, a list of attribute names, the attribute dictionary, and a default value. It should work as follows:
   a. If the dataset contains zero entropy, we are done. Create a leaf node with value equal to the data's classification and return it.
   b. If the dataset is empty, we have no data for this attribute value. Create a leaf node with the value set to the default value and return it.
   c. Otherwise, we have a non-leaf node. Use selectAttribute to find the attribute that maximizes gain. Then, remove that column for the dataset and the list of attributes and, for each value of that attribute, call makeTree with the appropriate subset of the data and add the TreeNode that is returned to the children, then return the TreeNode.
5. **(22%)** Now we know how to build a tree. We need to use it, though. To do this, you should implement the classify() method in TreeNode. classify should take as input the data instance to be classified and our attribute dictionary.

   This method is also recursive. If we are at a leaf, return the value of that leaf. Otherwise, check

which attribute this node tests and follow the appropriate child. If there is no child for this value, return a default value.

6. Congratulations! You now have a working decision tree. Test it out on the toy datasets. You might find it helpful to build a better printTree method, although this is not required. You might also want to add code to pickle your tree to a file, and a main to allow you to easily specify options.
7. **(13%)** Once you are convinced your tree is working correctly, you should evaluate its performance. To do this, choose two of the "real" datasets. To evaluate the tree's performance, we will do 5-fold cross-validation. This means that you select 80% of the data to train on, and hold back 20% for testing. This should be done 5 times, each with a different randomly-selected training set. (You will need to write a test harness for this; I have not provided a stub.) For each validation fold, calculate the tree's precision, recall, and accuracy, and then average these across all 5 runs; you should do this for both the training set and validation (test set) accuracy.
8. **(6%)** Prepare a short (3-4 paragraph) document that describes your tree's performance and discusses any anomalies or unexpected outcomes. You should
    a. Did one of the data sets prove more challenging than the others?
    b. What was the difference between training and test set accuracy? Was your tree overfitting on any of the data sets?
    c. Did it appear that any of the datasets were noisy or had other interesting issues?
    d. Did the trees look very different between the different folds?
9. **5 points extra credit:** Run the same 5-fold cross validations through your original ZeroR learning algorithm from HW1. What was the precision, recall, and accuracy? How much worse was this than your decision tree?

What to turn in:

- decisionTree.py: your version of all code described above
- readme.txt: a description of how I can run 5-fold cross-validation using your code on the existing data sets or a new dataset.
- discuss.txt (or .doc or .pdf): the document described in #8