CS 540-2: Introduction to Artificial Intelligence
Homework Assignment #3

Assigned: Tuesday, February 23, 2016
Due: Monday, March 7, 2016

Hand-In Instructions

This assignment includes written problems and programming in Java. **The written problems must be done individually but the programming problem can be done either individually or in pairs.** Hand in your work electronically to the Moodle dropbox called "HW3 Hand-In". Your answers to each written problem should be turned in as separate pdf files called `P1.pdf` and `P2.pdf` and put into a folder called `<wisc username>-HW3`. For example, if your wisc username is `jdoe`, then put the pdf files in a folder called `jdoe-HW3`. If you write out your answers by hand, you'll have to scan your answer and convert the file to pdf. Put your name at the top of the first page in each pdf file. **Both people on a team must individually answer and submit their answers to the written problems; no collaboration on these problems is allowed!**

For the programming problem, only one person in a pair needs to turn in the code. That person should copy a file called `DecisionTreeImpl.java` and any helper classes you and your partner wrote (but not other supplied files that were not modified) into the same folder called `<wisc username>-HW3`. For example, if your wisc username is `jdoe`, then you should put a file called `DecisionTreeImpl.java` into the folder called `jdoe-HW3`. Also put a file called `README.txt` in this folder that says who your partner is, including both their real name and their wisc username.

Each person on a team should compress their own folder to create a zipped file `<wisc username>-HW3.zip` and copy this one file into the Moodle dropbox.

Late Policy

All assignments are due at **11:59 p.m.** 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 and it is handed in between any time on Thursday, 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 three (3) free late days may be used throughout the semester without penalty. Assignment grading questions must be raised with the instructor or a TA within one week after the assignment is returned.

Problem 1. [15] 3-NN Classification

In this classification problem there are three classes: class1, class2 and class3. The number of feature dimensions is 2, where each feature is a real value. The following is a training set containing eight examples for this problem:

Class	Training Set Points
class1	(2, 2), (4, 4), (2, 4)
class2	(6, 5), (5.4, 5.6), (3.6, 6.4)
class3	(1.8, 8), (5.6, 8.2)

Use the 3-Nearest-Neighbors Algorithm (3-NN) with City-Block distance to solve the following problems. Recall that for two points (x_1, y_1) and (x_2, y_2) the City-Block (1-norm) distance between them is defined as $|x_2 - x_1| + |y_2 - y_1|$.

Show the steps of your calculation and show the closest 3 points used to classify each point. If a tie between two classes occurs, declare the class with the higher index as the output class.

(a) [8] Classify the following three points: (3, 3), (6, 4.4), (2.6, 6).

(b) [7] Add a class2 point (4.4, 6) to the training set. Now classify the points from (a) again.

Problem 2. [20] Hierarchical Clustering

Consider the following information about distances between pairs of nine U.S. cities:

Table 1: City Distances

BOS	**NY**	**DC**	**PIT**	**CHI**	**SEA**	**SF**	**LA**	**HOU**	
0	206	429	572	963	2976	3095	2979	1849	**BOS**
	0	233	372	802	2815	2934	2786	1628	**NY**
		0	242	671	2684	2799	2631	1408	**DC**
			0	461	2529	2576	2427	1336	**PIT**
				0	2013	2142	2054	1082	**CHI**
					0	808	1131	2314	**SEA**
						0	379	1927	**SF**
							0	1547	**LA**
								0	**HOU**

(a) [12] Use Hierarchical Clustering with single-linkage and Euclidean distance to cluster the cities by hand, until all cities are in the same cluster.

- For each iteration, show the membership in each cluster.
- Show all pairwise distances between clusters for each iteration.

(b) [4] Show the resulting dendrogram.

(c) [4] What clusters of cities are created if you want 3 clusters?

Problem 3. [65] Decision Trees

In this problem you are to implement a program that builds a decision tree for categorical attributes. The programming part includes building a tree from a training dataset and pruning the learned decision tree using a tuning dataset.

You are required to implement four methods and one member for the class `DecisionTreeImpl`:

```
1: private DecTreeNode root ;
2: DecisionTreeImpl ( DataSet train ) ;
3: DecisionTreeImpl ( DataSet train , DataSet tune ) ;
4: public String classify ( Instance instance ) ;
5: public void rootInfoGain ( DataSet train ) ;
```

`DecisionTreeImpl(DataSet train)` learns the decision tree from the given training dataset, while `DecisionTreeImpl(DataSet train, DataSet test)` not only learns the tree but also prunes it using the given tuning dataset. `classify` predicts the example's label using the trained decision tree. `rootInfoGain(DataSet train)` prints the information gain (one in each line) for all the attributes at the root based on the train set. Finally, the root of your tree should be stored in the member `root` we have declared for you. The next sections describe other aspects in detail.

Dataset

Our datasets come from the Balance Scale dataset which was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right (called class R), tip to the left (class L), or be balanced (class B). The attributes are the left weight (called attribute A1), the left distance (attribute A2), the right weight (attribute A3), and the right distance (attribute A4). Each example in this dataset is defined as a 4-dimensional feature vector corresponding to the values of the four attributes, A1, A2, A3 and A4, in the given order, and one ternary class label (one of R, L or B). All attributes in the feature vector are categorical with five distinct values each, which are specified in the header of each dataset file.

In each dataset file, there will be a header that gives information about the dataset; an example header and the first example in the dataset is shown below. First, there will be several lines starting with // that provide some description and comments about the dataset. Next, the line starting with%%will list all the class labels. Finally, each line starting with ## will give the name of one attribute and all its possible values. In the example below attribute A1 has five possible values: 1, ..., 5, and similarly for the

other three attributes. We have written the dataset loading part for you according to this header, so do NOT change it. Following the header are the examples in the dataset, one example per line. The first example is shown below and corresponds to the feature vector (A1=5, A2=1, A3=1, A4=2) and its class is L.

```
// Description and comments about the dataset
%%,L,B,R
##,A1,1,2,3,4,5
##,A2,1,2,3,4,5
##,A3,1,2,3,4,5
##,A4,1,2,3,4,5
5,1,1,2,L
...
```

Implementation Details

Predefined Data Types

We have defined four data types to assist your coding, called `Instance`, `DataSet`, `DecTreeNode` and `DecisionTreeImpl`. Their data members and methods are all commented, so it should not be hard to understand their meaning and usage.

Building the Tree

In both `DecisionTreeImpl` methods you are first required to build the decision tree using the training data. Refer to the pseudocode in Figure 18.5 on page 702 of the textbook to see what your code should do. To finish this part, you may need to write a recursive function corresponding to the `DecisionTreeLearning` function in the textbook or the `buildtree` function in the lecture slides.

Pruning

For the constructor `DecisionTreeImpl(DataSet train, DataSet tune)`, after building the tree you also need to prune it using the tuning dataset and the iterative Pruning Algorithm given in the lecture slides. In order to change the label of an internal node to be a leaf node with the class of the majority of the training examples that reach that node, you may also need to add some extra parameters to the function `Prune`.

Classification

`public String classify(Instance instance)` takes an example (called an instance) as its input and computes the classification output (as a string) of the previously-built decision tree. You do not need to worry about printing. That part is already handled in the provided code.

Printing and Information Gain at the Root

The only printing you need to do is in the method
`public void rootInfoGain(DataSet train)`. For each attribute print the output one line at a time: first the name of the attribute and then its information gain achieved by selecting that attribute at the root. The output order of the attributes and associated gains must be the same as the order that the attributes appear in the training set's header. An example is given in `output0.txt`. Print your results with 5 decimal places using `System.out.format("%.5f", arg)`

Testing

We will test your program using several training, tuning, and testing datasets, and the format of testing commands will be:

```
java HW3 <modeFlag> <trainFile> <tuneFile> <testFile>
```

where `trainFile`, `tuneFile` and `testFile` are the names of the training, tuning and testing datasets, respectively. `modeFlag` is an integer from 0 to 4, controlling what the program will output:

0: Print the information gain for each attribute at the root node based on the training set

1: Create a decision tree from the training set and print the tree

2: Create a decision tree from the training set and print the classification for each example in the test set

3: Create a decision tree from the training set, prune it using the tuning set, and then print the tree

4: Create a decision tree from the training set, prune it using the tuning set, and then print the classification for each example in the test set

To facilitate debugging, we have also provided sample input files and their corresponding output files. They are called `balance_train.txt`, `balance_tune.txt` and `balance_test.txt` for the input, and `outb0.txt` to `outb4.txt` for the output. So, here is an example command:

```
java HW3 1 balance_train.txt balance_tune.txt balance_test.txt
```

You are NOT responsible for any file input or console output other than `public void rootInfoGain (DataSet train)`. We have written the class `HW3` for you, which will load the data and pass it to the method you are implementing.

As part of our testing process, we will unzip the file you submit to Moodle, remove any class files, call `javac HW3.java` to compile your code, and then call the main method `HW3` with parameters of our choosing. Make sure that your code runs on one of the computers in the department because we will conduct our tests on these machines.

Deliverables

Hand in (see the cover page for details) your modified version of the code skeleton we provided to you along with your implementation of the decision tree algorithm. Also include any additional java class files needed to run your program.