CS 540-2: Introduction to Artificial Intelligence

Homework Assignment #3

Assigned: Wednesday, February 28
Due: Sunday, March 11

Hand-in Instructions

This homework assignment includes two written problems and a programming problem in Java. Hand in all parts electronically to your Canvas assignment HW #3 page. For *each* written question, submit a single **pdf** file containing your solution. Handwritten submissions *must* be scanned or converted to pdf. No photos or other file types allowed. Each file should have your name at the top of the first page. For the programming problem, submit a single zip file called `problem3.zip` that contains `DecisionTreeImpl.java` and any helper classes you wrote. Do *not* include any other of the skeleton code provided.

You should submit the following three files (with exactly these names) for this homework:

```
problem1.pdf    problem2.pdf    problem3.zip
```

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 discussed with a TA 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, peer mentors and instructor in order to help you answer the questions. You are welcome to show 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 answers
- not copy answers or code from anyone or anywhere
- not allow your answers to be copied
- not get any code from the Web

Problem 1. [20] Unsupervised Learning by Clustering

Consider the following information about *distances* (in miles) between pairs of 9 U.S. cities:

	BOS	NY	DC	MIA	CHI	SEA	SF	LA	DEN
BOS	0	206	429	1504	963	2976	3095	2979	1949
NY	206	0	233	1308	802	2815	2934	2786	1771
DC	429	233	0	1075	671	2684	2799	2631	1616
MIA	1504	1308	1075	0	1329	3273	3053	2687	2037
CHI	963	802	671	1329	0	2013	2142	2054	996
SEA	2976	2815	2684	3273	2013	0	808	1131	1307
SF	3095	2934	2799	3053	2142	808	0	379	1235
LA	2979	2786	2631	2687	2054	1131	379	0	1059
DEN	1949	1771	1616	2037	996	1307	1235	1059	0

The (latitude, longitude) *locations* of these cities are: BOS (42.4, 71.1), NY (41.7, 74.0), DC (38.9, 77.0), MIA (25.8, 80.2), CHI (41.9, 87.7), SEA (47.6, 122.3), SF (37.8, 122.4), LA (34.1, 118.2), and DEN (39.7, 105.0).

(a) [10] Perform (manually) **hierarchical agglomerative clustering** using *single-linkage* and the distance data in the above table.

i. [8] Show the resulting dendrogram.

ii. [2] What clusters of cities are created if you want 3 clusters?

(b) [10] Show the results of *one* (1) iteration of ***k*-means clustering** assuming $k = 2$ and the initial cluster centers are c_1 = (38.0, 103.0) and c_2 = (30.0, 78.0).

i. [3] Give the list of cities in each of the initial 2 clusters.

ii. [4] Give the coordinates of the new cluster centers. Use Euclidean distance between (latitude, longitude) coordinates.

iii. [3] Give the list of cities in the 2 clusters based on the new cluster centers computed in (ii).

Problem 2. [20] Decision Trees

The following table summarizes a training set containing 100 examples where each example has 3 binary attributes, *A*, *B* and *C*, and there are two class labels, $Y \in \{+, -\}$.

A	B	C	Y	
			+	−
T	T	T	5	0
F	T	T	0	20
T	F	T	20	0
F	F	T	0	5
T	T	F	0	0
F	T	F	25	0
T	F	F	0	0
F	F	F	0	25

(a) What is the **entropy** of *Y*, *H*(*Y*), as computed from these 100 examples?

(b) What is the **conditional entropy** of *Y* given *A*? That is, compute *H*(*Y* | *A*).

(c) What is the **information gain** between attribute *A* and class *Y*? That is, compute *I*(*Y*; *A*).

(d) What is the **information gain** between attribute *B* and class *Y*?

(e) What is the **information gain** between attribute *C* and class *Y*?

(f) Manually create the full **decision tree** using the attributes *A*, *B* and *C* to predict the class of *Y*. Show the resulting tree. At each non-leaf node, show the information gain of ALL candidate attributes possible at that node. In case of ties, use the following tie-breaking rules:

i. For class label majority vote ties, prefer the class "+".
ii. For attribute ties, prefer the attribute earliest in the alphabet.

(g) What is the classification accuracy of your tree on the training set?

Problem 3. [60] Implementing Decision Trees

In this problem you are to implement a program that builds a decision tree for categorical attributes and 2-class classification tasks. The programming part only requires building a tree from a training dataset and classifying instances of the test set with the learned decision tree.

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

```
1. private DecTreeNode root;
2. DecisionTreeImpl ( DataSet train );
3. public String classify ( Instance instance );
4. public void rootInfoGain ( DataSet train );
5. public void printAccuracy (DataSet test)
```

`DecisionTreeImpl(DataSet train)` builds a decision tree using the training set `train`.
`classify(Instance instance)` predicts the given `instance`'s class label using the previously-built decision tree.
`rootInfoGain(DataSet train)` prints the information gain (one in each line) for all the attributes at the root based on the training set, `train`. The root of your tree should be stored in the member root that has been declared for you. `RootInfoGain` will *only* be called at the root node. It will *not* be called at other nodes when building your decision tree.
`printAccuracy(DataSet test)` prints the classification accuracy for the instances in the test set, `test`, using the previously-learned decision tree.

Dataset

A risk loan dataset, https://bigml.com/dashboard/dataset/577bdcd477920c1ba40009a6, is to be used to **predict the risk quality of a loan application**. There are 1,000 noisy instances using 10 categorical attributes, divided into three files called `examples1.txt`, `examples2.txt` and `examples3.txt`. Each instance is classified as either good (class label G) or bad (class label B), so this is a 2-class classification problem. You can assume other datasets used for testing will also be 2-class classification tasks with categorical attributes. The 10 attributes and their possible values are shown in the table below:

A1: Checking status	x(no checking) n(x<0, negative) b(0<=x<200, bad) g(200<=x, good)
A2: Saving status	n(no known savings) b(x<100) m(100<=x<500) g(500<=x<=1000) w(1000<=x)

A3: Credit history	a(all paid) c(critical/other existing credit) d(delayed previously) e(existing paid) n(no credits)
A4: Housing	r(rent) o(own) f(free)
A5: Job	h(high qualified/self-employed/management) s(skilled) n(unemployed) u(unskilled)
A6: Property magnitude	c(car) l(life insurance) r(real estate) n(no known property)
A7: Number of dependents	1, 2
A8: Number of existing credits	1, 2, 3, 4
A9: Own telephones or not	y(yes), n(no)
A10: Foreign workers or not	y(yes), n(no)

In each 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. Each line starting with ## will give the name of one attribute and all its possible values. 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=x, A2=n, A3=e, A4=r, A5=h, A6=l, A7=1, A8=1, A9=y, A10=y) and its class is G. The class label for each instance is stored as a string in class `DataSet`.

```
// Description of the data set
%%,G,B
##,A1,x,n,b,g
##,A2,n,b,m,g,w
##,A3,a,c,d,e,n
##,A4,r,o,f
##,A5,h,s,n,u
##,A6,c,l,r,n
##,A7,1,2
##,A8,1,2,3,4
##,A9,y,n
##,A10,y,n
x,n,e,r,h,l,1,1,y,y,G
…
```

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 in the provided code. `DecTreeNode` is a class with several fields, including class label, attribute, etc. For a node that is a leaf, set its attribute value to be null and its terminal Boolean to True.

Building the Tree

In the `DecisionTreeImpl(DataSet train)` method you will build a decision tree using the training data. Refer to the pseudocode in Figure 18.5 of the textbook to see what your code should do. The root of the final decision tree should be assigned to the `root` class member. To complete 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. When calculating entropy, if the probability p is zero, define $p \log_2 p = 0$.

Use information gain to decide which attribute is the best at a non-leaf node. If ties occur when determining the majority class, choose the one with the smallest index in `List<String> labels`. If ties occur when choosing the best attribute, choose the one with the smallest index in `List<String> attributes`.

Classification

`public String classify(Instance instance)` takes an instance (also called an example) as its input and computes the classification output (as a string) using 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

```
public void rootInfoGain(DataSet train)
public void printAccuracy(DataSet test)
```

In `rootInfoGain`, for each attribute print the output one line at a time: first the name of the attribute and then the information gain achieved by selecting that attribute at the root. The output order of the attributes and associated information gain values *must* be the *same* as the order that the attributes appear in the training set's header. Print your results with 5 decimal places using `System.out.format("%.5f\n", arg)`

In `printAccuracy`, you should print out the accuracy (only) with 5 decimal places.

Testing

We will test your program using several training and testing datasets with the command line format:

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

where `trainFile` and `testFile` are the names of the training and testing datasets, respectively. `modeFlag` is an integer from 0 to 3, controlling what the program will output. The requirements for each value of `modeFlag` are described in the following table:

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 and print the accuracy of the classification for the test set

Here is an example command line:

```
java HW3 0 examples1.txt examples2.txt
```

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

The format of `rootInfoGain (modeFlag == 0)` should look like

```
A1 0.11111
A2 0.11111
…
A10 0.11111
```

The format of `printAccuracy (modeFlag == 3)` should look like

```
0.12345
```

As an example of what your output should be for each of the four modes, we have provided the correct outputs in `sample_outputs.zip` when using `examples1.txt` as the training set and `examples2.txt` as the test set.