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.