מימוש עצי החלטה

המטלה נלקחה מקורס מבוא לבינה מלאכותית CS 540.

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)` learns the decision tree from the training set, `train`.

`classify(Instance instance)` predicts the example, `instance`'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 training set, `train`. The root of your tree should be stored in the member root that has been declared for you. The next sections describe other aspects in detail.

`printAccuracy(DataSet test)` prints the classification accuracy for the instances in the test set, `test`, using the learned decision tree.

Dataset

Our datasets come from a risk loan dataset, which is being used to predict the risk quality of a loan application. There are altogether 1000 examples (also called instances) and we chose 10 categorical attributes to use for this assignment. Each instance is classified as good (class G) or bad (class B), so this is a 2-class classification problem. You can assume other datasets used for testing will also be 2-class classification tasks.

The tables of 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.

```
// 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, so it should not be hard to understand their meaning and usage.

Building the Tree

In the `DecisionTreeImpl(DataSet train)` method, you are 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.

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)` and

`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 only print out the accuracy with 5 decimal places.

Testing

We will test your program using several training and testing datasets using 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. Only `modeFlag` = {0, 1, 2, 3} are required to be implemented for this assignment.
The requirements for each value of `modeFlag`= {0, 1, 2, 3} are described as following:

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

To facilitate debugging, we have provided three input files called `example1.txt`, `example2.txt and example3.txt`. It is highly recommended that you write for yourself a small application that produces random input files in the expected format. Actually make them not so random, so that you'll know what to expect when you build your decision trees.

So, here is an example command:

```
java HW3 0 train1.txt test1.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 part of our testing process, we will unzip the file you submit, call javac`*.java` to compile your code, and then call the main method `HW3` with parameters of our choosing.

אופן ההגשה:

עליכם להעלות למערכת המטלות קובץ zip המכיל שני קבצים:

את הקובץ DecisionTreeImpl.java (המכיל את כל הקוד שכתבתם) וכן את הקובץ README.txt (המכיל שם, ת.ז. ותיעוד).