Homework Assignment #1
Due at midnight on Friday, 9/30

Part 1

For this part of the homework, you are to implement an ID3-like decision-tree learner for classification.

Your program should read files that are in the ARFF format. In this format, each instance is described on a single line. The feature values are separated by commas, and the last value on each line is the class label of the instance. Each ARFF file starts with a header section describing the features and the class labels. Lines starting with '%' are comments. See the link above for a brief, but more detailed description of the ARFF format. Your program should handle numeric and nominal attributes, and simple ARFF files (i.e. don't worry about sparse ARFF files and instance weights). Example ARFF files are provided below.

Your program can assume that (i) the class attribute is binary, (ii) it is named 'class', and (iii) it is the last attribute listed in the header section.

Your program should should implement a decision-learner according to the following guidelines:

- Candidate splits for nominal features should have one branch per value of the nominal feature. The branches should be ordered according to the order of the feature values listed in the ARFF file.
- Candidate splits for numeric features should use thresholds that are midpoints betweeen values in the given set of instances. The left branch of such a split should represent values that are less than or equal to the threshold.
- Splits should be chosen using information gain. If there is a tie between two features in their information gain, you should break the tie in favor of the feature listed first in the header section of the ARFF file. If there is a tie between two different thresholds for a numeric feature, you should break the tie in favor of the smaller threshold.
- The stopping criteria (for making a node into a leaf) are that (i) all of the training instances reaching the node belong to the same class, or (ii) there are fewer than `m` training instances reaching the node, where `m` is provided as input to the program, or (iii) no feature has positive information gain, or (iv) there are no more remaining candidate splits at the node.
- If the classes of the training instances reaching a leaf are equally represented, the leaf should predict the most common class of instances reaching the parent node.
- If the number of training instances that reach a leaf node is 0, the leaf should predict the the most common class of instances reaching the parent node.

Your program should be callable from the command line. It should be named `dt-learn` and should accept three command-line arguments as follows:

```
dt-learn <train-set-file> <test-set-file> m
```

If you are using a language that is not compiled to machine code (e.g. Java), then you should make a small script called `dt-learn` that accepts the command-line arguments and invokes the appropriate source-code program and interpreter.

Here are examples of such scripts.

As output, your program should print the tree learned from the training set and its predictions for the test-set instances. For each instance in the test set, your program should print one line of output with spaces separating the fields. Each output line should list the predicted class label, and actual class label. This should be followed by a line listing the number of correctly classified test instances, and the total number of instances in the test set.

Here are the trees and test-set classifications that your code should produce when given heart_train.arff as the training set and **heart_test.arff as the test set**.

- m = 2
- m = 4
- m = 10
- m = 20

Here are the trees and test-set classifications that your code should produce when given diabetes_train.arff as the training set **diabetes_test.arff as the test set** .

- m = 2
- m = 4
- m = 10
- m = 20

It is optional to print the number of training instances of each class after each node.

Part 2

For this part, you will plot *learning curves* that characterize the predictive accuracy of your learned trees as a function of the training set size. You will do this in two problem domains. The first data set involves predicting the presence or absence of heart disease. For this problem domain, you should use heart_train.arff as your training set and heart_test.arff as your test set. The second data set involves predicting whether a patient has diabetes or not. For this problem domain, you should use diabetes_train.arff as your training set and diabetes_test.arff as your test set.

You should plot points for training set sizes that represent 5%, 10%, 20%, 50% and 100% of the instances in each given training file. For each training-set size (except the largest one), randomly draw 10 different training sets and evaluate each resulting decision tree model on the test set. For each training set size, plot the average test-set accuracy and the minimum and maximum test-set accuracy. Be sure to label the axes of your plots. Set the stopping criterion `m=4` for these experiments.

Part 3

For this part, you will investigate how predictive accuracy varies as a function of tree size. For both of the data sets considered in Part 2, you should learn trees using the entire training set. Plot curves showing how test-set accuracy varies with the value `m` used in the stopping criteria. Show points for `m` = 2, 5, 10 and 20. Be sure to label the axes of your plots.

Submitting Your Work

You should turn in your work electronically using the Canvas course management system. Turn in all source files and your runnable program as well as a file called hw1.pdf that shows your work for Parts 2 and 3. All files should be compressed as one zip file named `<Wisc username>_hw1.zip`. Upload this zip file as Homework #1 at the course Canvas site.