Programming Assignment 6

Summary

In this assignment you will implement decision trees and decision forests. Your program will learn decision trees from training data and will apply decision trees and decision forests to classify test objects.

Command-line Arguments

You must a program that learns a decision tree for a binary classification problem, given some training data. In particular, your program will run as follows:

```
dtree training_file test_file option
```

The arguments provide to the program the following information:

- The first argument is the name of the training file, where the training data is stored.
- The second argument is the name of the test file, where the test data is stored.
- The third argument can have four possible values: `optimized`, `randomized`, `forest3`, or `forest15`. It specifies how to train (learn) the decision tree, and will be discussed later.

Both the training file and the test file are text files, containing data in tabular format. Each value is a number, and values are separated by white space. The i-th row and j-th column contain the value for the j-th feature of the i-th object. The only exception is the LAST column, that stores the class label for each object. **Make sure you do not use data from the last column (i.e., the class labels) as attributes (features) in your decision tree.**

Example files that can be passed as command-line arguments are in the datasets directory. That directory contains three datasets, copied from the UCI repository of machine learning datasets:

- The `pendigits` dataset, containing data for pen-based recognition of handwritten digits.
  - 7494 training objects.
  - 3498 test objets.
  - 16 attributes.
  - 10 classes.
- The `satelite` dataset. The full name of this dataset is Statlog (Landsat Satellite) Data Set, and it contains data for classification of pixels in satellite images.
  - 4435 training objects.
  - 2000 test objets.
  - 36 attributes.
  - 6 classes.
- The `yeast` dataset, containing some biological data whose purpose I do not understand myself.
  - 1000 training objects.
  - 484 test objets.
  - 8 attributes.
  - 10 classes.

For each dataset, a training file and a test file are provided. The name of each file indicates what dataset the file belongs to, and whether the file contains training or test data.

Note that, for the purposes of your assignment, it does not matter at all where the data came from. One of the attractive properties of decision trees (and many other machine learning methods) is that they can be applied in the exact same way to many different types of data, and produce useful results.

Training Phase

The first thing that your program should do is a decision tree using the training data. What you train and how you do the training depends on the came of the third command line argument, that we called "option". This option can take four possible values, as follows:

- `optimized`: in this case, at each non-leaf node of the tree (starting at the root) you should identify the optimal combination of attribute (feature) and threshold, i.e., the combination that leads to the highest information gain for that node.
- `randomized`: in this case, at each non-leaf node of the tree (starting at the root) you should choose the attribute (feature) randomly. The threshold should still be optimized, i.e., it should be chosen so as to maximize the information gain for that node and for that randomly chosen attribute.
- `forest3`: in this case, your program trains a random forest containing three trees. Each of those trees should be trained as discussed under the "randomized" option.
- `forest15`: in this case, your program trains a random forest containing 15 trees. Each of those trees should be trained as discussed under the "randomized" option.

All four options are described in more details in the lecture slides titled Practical Issues with Decision Trees. Your program should follow the guidelines stated in those slides.

Training Phase Output

After you learn your tree or forest, you should print it. Every node must be printed, in breath-first order, with left children before right children. For each node you should print a line containing the following info:

- tree ID. If you are learning a single tree, the ID is 0. If you are learning multiple trees, their ID range from 0 to the number of trees - 1, and should be printed in increasing order of their ID.
- node ID. The root has ID 1. If a node has ID = N, then its left child has ID 2*N and its right child has ID 2*N + 1.
- a feature ID, specifying which attribute is used for the test at that node. This is a number starting from 0, indicating the position of the column that contains values for that attribute. If the node is a leaf node, print -1 for the feature ID.
- a threshold that, combined with the feature ID specifies the test for that node. If feature ID = X and threshold = T, then objects whose X-th feature has a value LESS THAN T are directed to the left child of that node. If the node is a leaf node, print -1 for the threshold.
- information gain achieved by the test chosen for this node.

To produce this output in a uniform manner, use these printing statements:

- For C or C++, use:

```
printf("tree=%2d, node=%3d, feature=%2d, thr=%6.2lf, gain=%lf\n", tree_id, node_id, feature_id, threshol
```

- For Java, use:

```
System.out.printf("tree=%2d, node=%3d, feature=%2d, thr=%6.2f, gain=%f\n", tree_id, node_id, feature_id,
```

- For Python or any other language, just make sure that you use formatting specifies that produce aligned output that matches the specs given for C and Java.

Classification

For each test object (each line in the test file) print out, in a separate line, the classification label (class1 or class2). If your classification result is a tie among two or more classes, choose one of them randomly. For each test object you should print a line containing the following info:

- object ID. This is the line number where that object occurs in the test file. Start with 0 in numbering the objects, not with 1.
- predicted class (the result of the classification).
- true class (from the last column of the test file).
- accuracy. This is defined as follows:
  - If there were no ties in your classification result, and the predicted class is correct, the accuracy is 1.
  - If there were no ties in your classification result, and the predicted class is incorrect, the accuracy is 0.
  - If there were ties in your classification result, and the correct class was one of the classes that tied for best, the accuracy is 1 divided by the number of classes that tied for best.
  - If there were ties in your classification result, and the correct class was NOT one of the classes that tied for best, the accuracy is 0.

To produce this output in a uniform manner, use these printing statements:

- For C or C++, use:

```
printf("ID=%5d, predicted=%3d, true=%3d, accuracy=%4.2lf\n", object_id, predicted_class, true_class, acc
```

- For Java, use:

```
System.out.printf("ID=%5d, predicted=%3d, true=%3d, accuracy=%4.2f\n", object_id, predicted_class, true_
```

- For Python or any other language, just make sure that you use formatting specifies that produce aligned output that matches the specs given for C and Java.

After you have printed the results for all test objects, you should print the overall classification accuracy, which is defined as the average of the classification accuracies you printed out for each test object. To print the classification accuracy in a uniform manner, use these printing statements:

- For C or C++, use:

```
printf("classification accuracy=%6.4lf\n", classification_accuracy);
```

- For Java, use:

```
System.out.printf("classification accuracy=%6.4f\n", classification_accuracy);
```

- `For Python or any other language, just make sure that you use formatting specifies that produce alignec`

Grading

- 20 points: Correct processing of the `optimized` option. Identifying and choosing, for each node, the (feature, threshold) pair with the highest information gain for that node, and correctly computing that information gain.
- 10 points: Correct processing of the `randomized` option. In other words, identifying and choosing, for each node, an appropriate (feature, threshold) pair, where the feature is chosen randomly, and the threshold maximizes the information gain for that feature,
- 10 points: Correctly directing training objects to the left or right child of each node, depending on the (threshold, value) pair used at that node.
- 10 points: Correct application of pruning, as specified in the slides (if any .
- 15 points: Correctly applying decision trees to classify test objects.
- 15 points: Correctly applying decision forests to classify test objects.
- 20 points: Following the specifications in producing the required output.

How to submit

Submissions should be made using Blackboard. Implementations in Python, C, C++, and Java will be accepted. If you would like to use another language, please first check with the instructor via e-mail. Points will be taken off for failure to comply with this requirement.

Submit a ZIPPED directory called `programming-assignment6.zip` (no other forms of compression accepted, contact the instructor or TA if you do not know how to produce .zip files). The directory should contain:

- Source code for the programming part. Including binaries is optional.
- A file called readme.txt, which should specify precisely:
  - Name and UTA ID of the student.
  - What programming language is used.
  - How to run the code, including very specific compilation instructions. Instructions such as "compile using g++" are NOT considered specific. Providing all the command lines that are needed to complete the compilation on omega is specific.

Insufficient or unclear instructions will be penalized by up to 20 points. Code that does not run on omega machines gets AT MOST half credit, unless you obtained prior written permission.