Programming Assignment 7

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 implement a program that learns a naive Bayes classifier for a classification problem, given some training data and some additional options. In particular, your program can be invoked as follows:

```
naive_bayes <training_file> <test_file> histograms <number>
naive_bayes <training_file> <test_file> gaussians
naive_bayes <training_file> <test_file> mixtures <number>
```

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 three possible values: `histograms`, `gaussians`, or `mixtures`.
- If the third argument is `histograms`, the fourth argument specifies how many bins to use for each histogram. If the third argument is `mixtures`, the fourth argument specifies how many Gaussians to use for each mixture.

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: Histograms

If the third commandline argument is `histograms`, then you should model P(x | class) as a histogram separately for each dimension of the data. The number of bins for each histogram is specified by the fourth command line argument.

Suppose that you are building a histogram of N bins for the j-th dimension of the data. Let S be the smallest and L be the largest value in the j-th dimension among all training data. Let G = (L-S)/N. Then, your bins should have the following ranges:

- Bin 0, from -infinity to S+G.
- Bin 1, from S+G to S+2G.
- Bin 2, from S+2G to S+3G.
- ...
- Bin N-1 from S+(N-1)G to +infinity.

The output of the training phase should be a sequence of lines like this:

```
Class %d, attribute %d, bin %d, P(bin | class) = %.2f
```

The output lines should be sorted by class number. Within the same class, lines should be sorted by attribute number. Within the same attribute, lines should be sorted by bin number.

Training: Gaussians

If the third commandline argument is `gaussians`, then you should model P(x | class) as a Gaussian separately for each dimension of the data. The output of the training phase should be a sequence of lines like this:

```
Class %d, attribute %d, mean = %.2f, std = %.2f
```

The output lines should be sorted by class number. Within the same class, lines should be sorted by attribute number.

Training: Mixtures of Gaussians

If the third commandline argument is `histograms`, then you should model P(x | class) as a mixture of Gaussians separately for each dimension of the data. The number of Gaussians for each mixture is specified by the fourth command line argument.

Suppose that you are building a mixture of N Gaussians for the i-th dimension of the data. Let S be the smallest and L be the largest value in the i-th dimension among all training data. Let G = (L-S)/N. Then, you should initialize all standard deviations of the mixture to 1, and you should initialize the means as follows:

- For the first Gaussian, the initial mean should be S + G/2.
- For the second Gaussian, the initial mean should be S + G + G/2.
- For the third Gaussian, the initial mean should be S + 2G + G/2.
- ...
- For the N-th Gaussian, the initial mean should be S + (N-1)G + G/2.

You should repeat the main loop of the EM algorithm 50 times. So, no need to worry about any other stopping criterion, your stopping criterion is simply that the loop has been executed 50 times.

The output of the training phase should be a sequence of lines like this:

```
Class %d, attribute %d, Gaussian %d, mean = %.2f, std = %.2f
```

The output lines should be sorted by class number. Within the same class, lines should be sorted by attribute number. Within the same attribute, lines should be sorted by Gaussian number.

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).
- probability of the predicted class given the data.
- 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, probability = %.4lf, true=%3d, accuracy=%4.2lf\n",
       object_id, probability, predicted_class, true_class, accuracy);
```

- For Java, use:

```
System.out.printf("ID=%5d, predicted=%3d, probability = %.4f, true=%3d, accuracy=%4.2f\n",
                  object_id, predicted_class, probability, true_class, accuracy);
```

- 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 aligned output that matches the specs given for C and Java.

Grading

- 20 points: Correct estimation of histograms, correct probabilities estimated for each bin.
- 20 points: Correct estimation of Gaussians, correct mean and standard deviation estimated for each Gaussian.
- 20 points: Correct estimation of mixtures of Gaussians, correct mean and standard deviation estimated for each Gaussian of each mixture.
- 10 points: Correct application of a naive Bayes classifier based on histograms for classification of test data.
- 10 points: Correct application of a naive Bayes classifier based on Gaussians for classification of test data.
- 10 points: Correct application of a naive Bayes classifier based on mixtures of Gaussians for classification of test data.
- 10 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-assignment7.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.