CSE 4309 - Assignments - Assignment 5

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment5.zip, containing the following two files:

- answers.pdf, for your answers to the written tasks, and for the output that the programming task asks you to include. Only PDF files will be accepted. All text should be typed, and if any figures are present they should be computer-generated. Scans of handwriten answers will NOT be accepted.
- decision_tree.py, containing your Python code for the programming part. Your Python code should run on the versions specified in the syllabus, unless permission is obtained via e-mail from the instructor or the teaching assistant. Also submit any other files that are needed in order to document or run your code (for example, additional source code files).

These naming conventions are mandatory, non-adherence to these specifications can incur a penalty of up to 20 points.

Your name and UTA ID number should appear on the top line of both documents.

Task 1 (70 points, programming)

In this task 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.

Function Name and Arguments

File decision_tree_main.py contains incomplete code that aims to learn a decision tree or decision forest given some training data, and then evaluate this model on test data.

To complete that code, you must create a file called `decision_tree.py`, where you implement a Python function called `decision_tree`. Your function should be invoked as follows:

```
decision_tree(training_file, test_file, option, pruning_thr)
```

The arguments provide the following information:

- The first argument, `training_file`, is the path name of the training file, where the training data is stored. The path name can specify any file stored on the local computer.
- The second argument, `test_file`, is the path name of the test file, where the test data is stored. The path name can specify any file stored on the local computer.
- The third argument, `option`, can be equal to `optimized`, or it can be equal to an integer greater than 0. It specifies how to train (learn) the decision tree, whether to use a single tree or a

random forest, and how many trees the random forest should contain. This argument will be discussed later.
- The fourth argument, `pruning_thr`, is a number greater than or equal to 0, that specifies the pruning threshold.

The training and test files will follow the same format as the text files in the UCI datasets directory. A description of the datasets and the file format can be found on this link. 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. Your code should also work with ANY OTHER training and test files using the same format as the files in the UCI datasets directory.

As the description states, **do NOT use data from the last column (i.e., the class labels) as features**. In these files, all columns except for the last one contain example inputs. The last column contains the class label.

Training Stage

The first thing that your program should do is train a decision tree or a decision forest using the training data. What you train and how you do the training depends on the came of the third argument, that we called `option`. This option can take the following possible values:

- `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.
- `integer_greater_than_0`: in this case, your program trains a random forest containing as many trees as specified by the `integer_greater_than_0` argument. At each non-leaf node of each tree 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. Note that if the specified integer is 1, then you end up with a trivial random forest that contains only one decision tree, which is still a valid case.

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 1. If you are learning multiple trees, their ID range from 1 to the number of trees, 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 1, 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. For a leaf node, you should consider the information gain at that node to be 0.

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

```
fprintf('tree=%2d, node=%3d, feature=%2d, thr=%6.2f, gain=%f\n', tree_id, node_id,
feature_id, threshold, gain);
```

Classification Stage

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 1 in numbering the objects, not with 0.
- predicted class (the result of the classification). If your classification result is a tie among two or more classes, choose one of them randomly.
- 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 this printing statement:

```
fprintf('ID=%5d, predicted=%3d, true=%3d, accuracy=%4.2f\n', object_id,
predicted_class, true_class, accuracy);
```

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 this printing statement:

```
fprintf('classification accuracy=%6.4f\n', classification_accuracy);
```

Output for answers.pdf

In your answers.pdf document, you need to provide parts of the output for some invocations of your program listed below. For each invocation, provide:

- The full output of the training stage.
- ONLY THE LAST LINE (the line printing the classification accuracy) of the output by the test stage.

Include this output for the following invocations of your program:

```
decision_tree.py pendigits_training pendigits_test optimized 50
decision_tree.py pendigits_training pendigits_test 1 50
decision_tree.py pendigits_training pendigits_test 3 50
```

Expected Classification Accuracy

For the "optimized" option, the results are mostly deterministic, and your results should mostly match mine. There may be some small differences between your results and mine. The reason is that sometimes there are multiple choices for optimal attribute and threshold, that are equally good in terms of information gain. In those cases your code can make a choice that is different than mine. For the other three options (randomized, forest3, forest 15) the results will vary, since the code needs to make randomized choices. These are the results I got on the pendigits dataset with my implementation:

- With the "optimized" option and a pruning threshold of 50, the classification accuracy was 0.8382. The running time was about 8 seconds on my computer.
- I ran my code 10 times with the "1" option and a pruning threshold of 50, so that we build and use a single randomized tree. The classification accuracy ranged between 0.6790 and 0.7753. Each run took about 1 to 2 seconds on my computer.
- I ran my code 10 times with the "3" option and a pruning threshold of 50, so that we build and use a random forest of three randomized trees. The classification accuracy ranged between 0.8047 and 0.8699. Each run took about 3 to 4 seconds on my computer.
- I ran my code 10 times with the "15" option and a pruning threshold of 50, so that we build and use a random forest of 15 randomized trees. The classification accuracy ranged between 0.8731 and 0.8994. Each run took about 20 seconds on my computer.

For the randomized tree and random forests, it is possible that sometimes you get results outside those ranges, but most of the time that you run your code with the parameters specified above you should be getting accuracies within those ranges.

Grading Rubric

- 10 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.
- 10 points: Correctly applying decision trees to classify test objects.
- 5 points: Correctly applying decision forests to classify test objects.
- 15 points: Following the specifications in producing the required output.

Task 1b (Extra Credit, maximum 10 points).

A maximum of 10 extra credit points will be given to the submission or submissions that identify the function arguments (hyperparameters) achieving the best test accuracy for any of the three test datasets. These function arguments, and the attained accuracy, should be reported in answers.pdf, under a clear "Task 1b" heading. These results should be achievable by calling the function you submit for Task 1. You should achieve the reported accuracy at least five out of 10 times when you run your code.

Task 1c (Extra Credit, maximum 10 points).

In this task, you are free to change any implementation options that you are not free to change in Task 1. Examples of such options include number of trees in the random forest, using entropy to choose thresholds, how you do pruning, or any other relevant option for decision trees. You can submit a Python file called decision_tree_opt.py, that implements your improved version of the `decision_tree` function. A maximum of 10 points will be given to the submission or submissions that, according to the instructor and GTA, achieve the best improvements (on any of the three datasets) compared to the specifications in Task 1. In your answers.pdf document, under a clear "Task 1c" heading, explain what modifications you made, what results you achieved, and what arguments we should call your improved `decision_tree` function with in order to verify your results.

Task 2 (10 points).

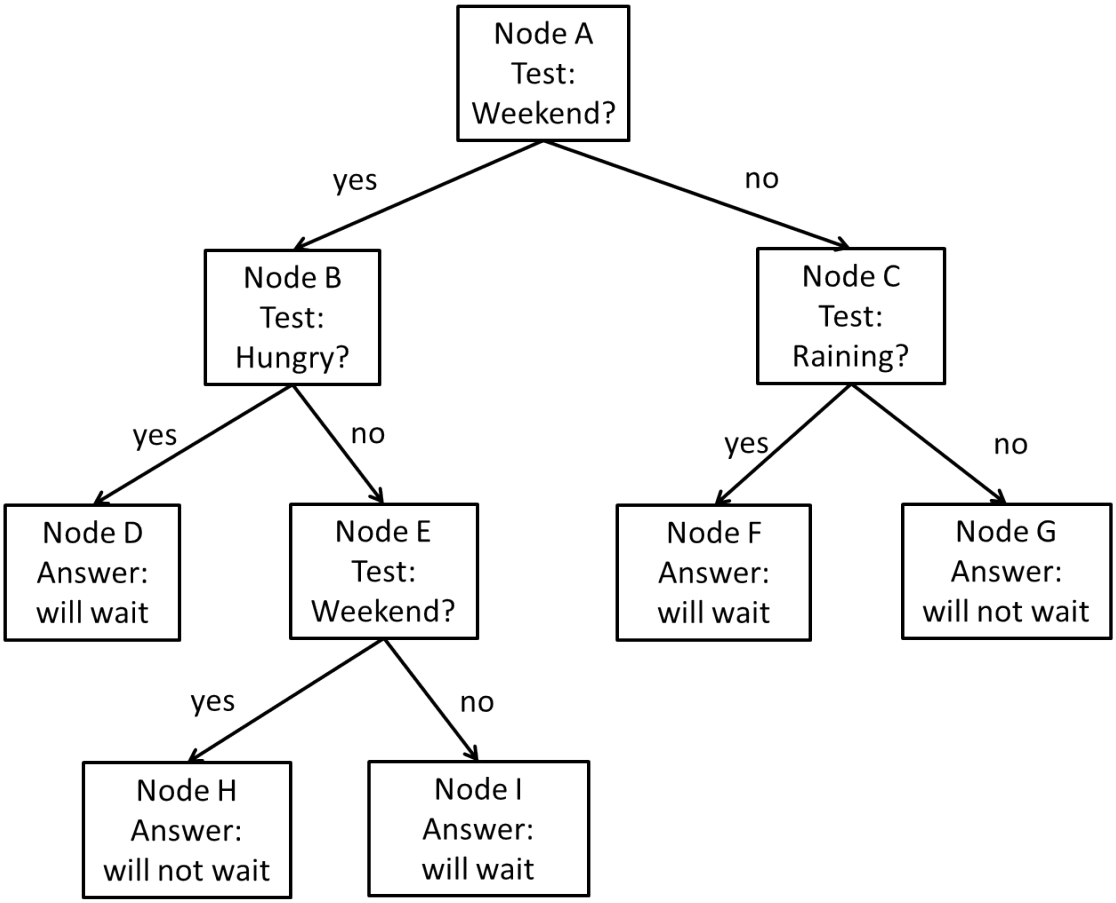

Figure 1: A decision tree for estimating whether the patron will be willing to wait for a table at a restaurant.

Part a (2 points): Suppose that, on the entire set of training samples available for constructing the decision tree of Figure 1, 80 people decided to wait, and 20 people decided not to wait. What is the initial entropy at node A (before the test is applied)?

Part b (2 points): As mentioned in the previous part, at node A 80 people decided to wait, and 20 people decided not to wait.

- Out of the cases where people decided to wait, in 20 cases it was weekend and in 60 cases it was not weekend.
- Out of the cases where people decided not to wait, in 15 cases it was weekend and in 5 cases it was not weekend.

What is the information gain for the weekend test at node A?

Part c (2 points): In the decision tree of Figure 1, node E uses the exact same test (whether it is weekend or not) as node A. What is the information gain, at node E, of using the weekend test?

Part d (2 points): We have a test case of a hungry patron who came in on a rainy Tuesday. Which leaf node does this test case end up in? What does the decision tree output for that case?

Part e (2 points): We have a test case of a not hungry patron who came in on a sunny Saturday. Which leaf node does this test case end up in? What does the decision tree output for that case?

Task 3 (10 points)

Class	A	B	C
X	1	2	1
X	2	1	2
X	3	2	2
X	1	3	3
X	1	2	2
Y	2	1	1
Y	3	1	1
Y	2	2	2
Y	3	3	1
Y	2	1	1

We want to build a decision tree that determines whether a certain pattern is of type X or type Y. The decision tree can only use tests that are based on attributes A, B, and C. Each attribute has 3 possible values: 1, 2, 3 (we do not apply any thresholding). We have the 10 training examples, shown on the table (each row corresponds to a training example).

What is the information gain of each attribute at the root? Which attribute achieves the highest information gain at the root?

Task 4 (5 points)

Suppose that, at a node N of a decision tree, we have 1000 training examples. There are four possible class labels (A, B, C, D) for each of these training examples.

Part a: What is the highest possible and lowest possible entropy value at node N?

Part b: Suppose that, at node N, we choose an attribute K. What is the highest possible and lowest possible information gain for that attribute?

Task 5 (5 points)

Your boss at a software company gives you a binary classifier (i.e., a classifier with only two possible output values) that predicts, for any basketball game, whether the home team will win or not. This classifier has a 28% accuracy, and your boss assigns you the task of improving that classifier, so that you get an accuracy that is better than 60%. How do you achieve that task? Can you guarantee achieving better than 60% accuracy?
