CS 540 (Shavlik) HW 2 – Ensembles and Searching for Solutions

Assigned: 9/27/2016
Due: 10/13/2016 (not accepted after 11:55pm on 10/20/2016)
Points: 100

1) (20 points) Extend your HW1 code for decision-tree learning to the *ensemble* method called 'bagging.' (Email TA Sam if you did not get your HW1 code to work properly and he'll provide some ID3 code for your use. Please do not share this code with anyone else.)

   Note that we will not be doing the full random-forests approach described in class. Instead, we will only do the part where we use 'copied with replacement' replicates of the training set. Such datasets are often called "bootstrap" replicates; see https://en.wikipedia.org/wiki/Bootstrapping_(statistics) if you are curious. (If you are a CS graduate student, I recommend you also implement the part of random forests where at each call to ID3, including recursive ones, you first randomly select *i* features and then select the one of these with the highest information gain. This will not impact your grade, though.)

   We will not be running your code during grading, but we want you to turn it in as **HW2.java**. I suggest you copy your HW1.java file and then edit it. Add some reasonable comments. The explanation below talks about the possible outputs being "lowToMid" and "midToHigh." This is done for clarity, but aim to write your code to work for any two output labels, for tune and test sets of any (non-negative) size, and for any (odd-numbered and positive) size of the forest (ie, the '101' below).

   Here is what you need to do:

   a. Copy the 101 'copied with replacement' replicates of the training set from http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW2/wine-bagged-unzipped/

   b. There is also a *zip file (http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW2/wine-bagged.zip) that contains all 101 files; it might be easier to copy this to your computer and then unzip. Be sure to note the repeated pattern in the file names, which makes reading 101 files much easier. Hint: use a FOR LOOP from 1 to 101.

   c. Copy the TUNE set from *http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW0/red-wine-quality-tune.data*

   d. Copy the TEST set from *http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW0/red-wine-quality-test.data*

   e. Train ID3 on all 101 replicated trainsets (you'll probably want to turn printing off!).

   f. Collect the predictions of the 101 trees on each of the 298 tune-set and 298 test-set examples. Store these in your code, e.g. in two 2D arrays where array cell *i*,*j* holds the predicted output of tree *i* on tune-set (test-set) example *j*. Store the predictions (i.e., *lowToMid* or *midToHigh*), rather than if the prediction was correct. This will make the next step easier.

   g. Compute and plot the accuracy of the following 'combination rule' for *L* in 1, 3, 5, 7, …, 99, 101 (i.e., all odd-numbered values from 1 to 101) for *both* the tune set and the test set:

If at least *L* of the 101 learned trees predict *lowToMid* **then output** *lowToMid*
otherwise output *midToHigh*

I assume everyone has access to either Microsoft Office or wants to use the free Google Docs, but hand-plotting is ok. Be sure to label you axes.

Note you are creating *two* curves (but plot both in the same figure so you can visually compare them), one for the tune-set accuracy and one for the test-set accuracy. Clearly mark the location of the BEST tune-set results (if ties, use the *smallest* L, in an application of Occam's Razor).

Discuss (1) how much, if at all, bagging helped, (2) how much, if at all, it helped to use a tune set (compared to simply using a majority vote, i. e., L=51), and (3) how well the tune set selected the best L for the test set. Note that the tune set is an independent sample of the original dataset and does not overlap the train nor test sets. (If you did not get HW0 to work, assume that an unpruned decision tree on this task would get 75% test-set accuracy.)

Notice that in Part 1e we created two 'predictions' arrays. Use them here, rather than, for each value of L, having the 101 decision trees process the tune and test sets anew.

h. Turn in your plot and (ideally, typed) discussion in the file **HW2_part1.pdf**

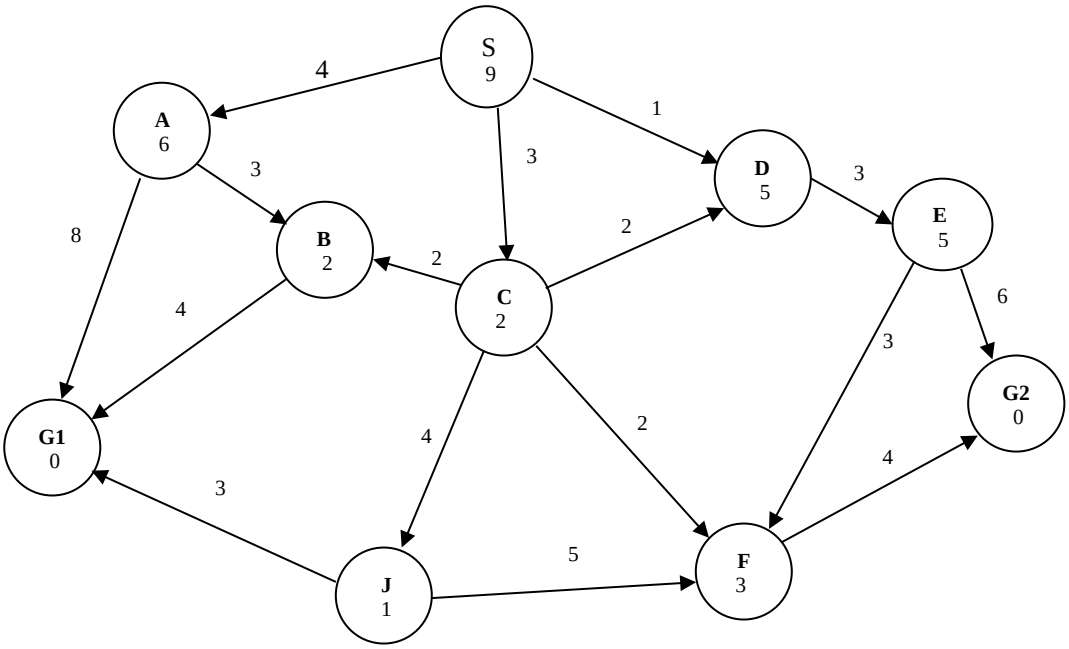
The remaining questions on this homework are "paper and pencil" ones, but you need to turn in the PDF file **HW2_part2_part3_part4.pdf**.

2) (60 points) Consider the search space below, where *S* is the start node and *G1* and *G2* satisfy the goal test. Arcs are labeled with the cost of traversing them and the estimated cost to a goal (i.e., the *h* function) is reported inside nodes (so lower scores are better).

For each of the following search strategies, indicate which goal state is reached (if any) and list, *in order*, <u>all</u> the states *popped off of the OPEN list*. When all else is equal, nodes should be removed from OPEN in alphabetical order.

Show your work using the columns headers used in the lecture notes.

(a) **Breadth First**
(b) **Depth First**
(c) **Iterative Deepening**
(d) **Uniform Cost (i.e., using f = g)**
(e) **Best-First (using f = h)**
(f) **Best-First (using f = g + h)**
(g) **Beam Search (with beam width = 2 and f = h)**
(h) **Hill Climbing (using the h function only)** // Be sure to notice part (i) below.



(i) **Is this *h* function *admissible*? Explain your answer.**

General Pseudocode for Searching (for use with Problem 2)

The following is the basic outline for the various search algorithms (some steps need to be modified depending on the specifics of the search being used).

```
OPEN = { startNode } // Nodes under consideration.
CLOSED = { }         // Nodes that have been expanded.
While OPEN is not empty
{       Remove the first item from OPEN. Call this item X.
        If goalState?(X) return the solution found.

        // Expand node X if it isn't a goal state.
        Add X to CLOSED.
        Generate the immediate neighbors (i.e., children) of X.
        Eliminate those children already in OPEN or CLOSED.
        Based on the search strategy, insert the remaining
            children into OPEN.
}
Return FAILURE   // Failed if OPEN exhausted w/o a goal found.
```

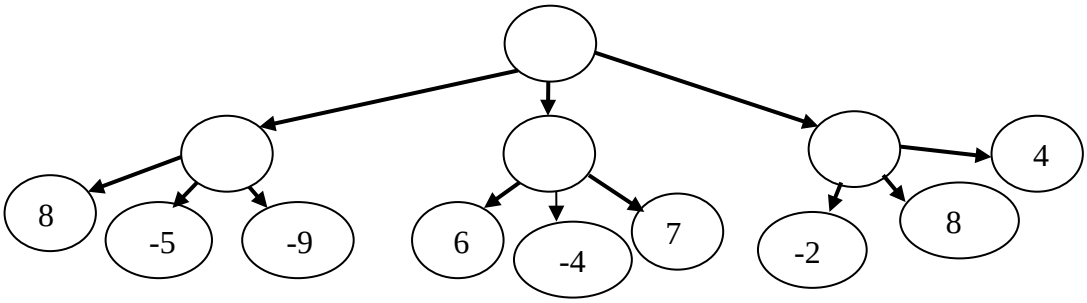
3) (10 points) Using the same search space as in Problem 2, consider using *Simulated Annealing* as your search strategy. Assume the current *temperature* is 100. Remember to negate the node scores since Simulated Annealing goes "uphill" in the textbook.

(a) If you are at Node *C* and simulated annealing has randomly selected node *J* for consideration, what is the probability this node is accepted?

(b) If you are at Node *C* and simulated annealing has randomly selected node *F* for consideration, what is the probability this node is accepted?

4) **Game Playing** (10 points)

(a) Apply the *minimax* algorithm to the partial game tree below, where it is the minimizer's turn to play and the game does not involve randomness. The values estimated by the static-board evaluator (SBE) are indicated in the leaf nodes (lower scores are better for the minimizer). Write the estimated values of the intermediate nodes inside their circles, and indicate the proper move of the minimizer by circling one of the root's outgoing arcs.



(b) List one (1) leaf node in the above game tree whose SBE score need not be computed. Explain why.