**Northeastern University**

CS 5100 Foundations of Artificial Intelligence

**Homework#4 Assigned: 7/15/20 Due: 7/29/20** [300 points]

***PART A: Theory and Algorithms*** *[100 points] \* See PART B Prog Assignment on Page 3*

Please - clearly write your **full name** on the first page. Submit a single PDF file.

Please provide brief but complete explanations, using diagrams where necessary, and suitably using your own words. While presenting calculations, explain the variables and.

Study Chapter 11 to 15 (skip 12) per Lecture 4 Topics from Russel AI textbook 3rd Edition – selected sections only, plus notes provided. Answer the below:

1. Problem 11.2 [8 points]
2. Problem 11.6 [12 points]
3. (A) What is **Naïve Bayes** algorithm? How is it used in AI (examples)? [5 Points]

(B) Table 1. Data on customers who purchased a PC at a computer store, showing age group, income, education and whether they purchased a PC earlier. [15 Points]

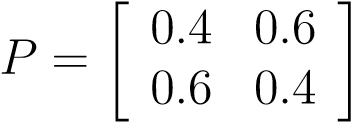
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Age** | **Income** | **Education** | **Prior Purchase** | **Buys PC?** |
| Young adult | High | College | Yes | Yes |
| Middle Age | Low | College | No | No |
| Middle Age | High | High School | Yes | No |
| Senior | Low | College | No | No |
| Young adult | Medium | High School | Yes | Yes |
| Senior | High | College | yes | Yes |
| Middle Age | Medium | College | No | No |
| Young adult | Low | High School | No | No |
| Middle Age | Low | College | Yes | Yes |
| Young adult | High | College | No | Yes |
| Senior | Medium | High School | No | No |

For the same PC purchase dataset given in Table 1 above, develop a Naïve Bayes Classification method, and use the same to classify the new example given below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Senior | High | High School | Yes |  |  | ? |

Clearly show all your calculations, step by step.

1. 13.18 a [10 points]
2. 14.1 [15 points]
3. 14.4 [15 points]
4. A computer system can operate in two different modes. Every hour, it remains in the same mode or switches to a different mode according to the transition probability matrix [8 points]



1. Compute the 2-step transition probability matrix. **Markov Chains**
2. If the system is in Mode I at 5:30 pm, what is the probability that it will be in Mode I at 8:30 pm on the same day?

1. Inspect [Figure 1 and explanation from this paper](https://reader.elsevier.com/reader/sd/pii/S000437020500113X?token=FE3ED6AD3CD7695CD79ED93B3878D723FD1D069E8C61A28077DBE821D9CF610CA4A9405812A9E28C9B5B2ADB79B4F929). Then, explain how HMMs are useful to a Robot. Explain using a Figure (HMM) for the Vacuum Cleaner robot from Ch. 2. You need to draw this diagram using this paper’s example. [12 points]

***PART B: Implementation – Programming Assignment (PA #2)*** *[200 points]*

In this assignment, we’ll revisit **Markov Decision Processes** while also trying out **Q- Learning**, the reinforcement learning approach that associates utilities with attempting actions in states.

The problem that we’re attempting to solve is the following variation on the “Wumpus World” sometimes used as an example in lecture and the textbook.

*Provided files:*

*ice.py,*

*easy[MDP/Q].[txt/out],*

*slippy[MDP/Q].[txt/out],*

*verySlippy[MDP/Q].[txt/out],*

*big[MDP/Q].txt*

1. There is a grid of spaces in a rectangle. Each space can contain a pit (negative reward), gold (positive reward), or nothing.
2. The rectangle is effectively surrounded by walls, so anything that would move you outside the rectangle, instead moves you to the edge of the rectangle.
3. The floor is icy. Any attempt to move in a cardinal direction results in moving a somewhat random number of spaces in that direction. The exact probabilities of moving each number of spaces are given in the problem description. (If you slide too far, see rule #2.)
4. Landing on a pit or gold effectively “ends the run,” for both a Q learner and an agent later trying out the policy. It’s game over. (To simulate this during Q learning, set all Q values for the space to equal its reward, then start over from a random space.) Note that it's still possible to slide *past* a pit or gold - this doesn't end the run.

A sample input looks like this:

MDP

0.6 0.3 0.1

- P - G - P - - G -

P G - P - - - P - -

P P - P P - P - P -

P - - P P P

- - - - - - - - P G

The first line reads “MDP,” which says we should solve this like an MDP. (The alternative is “Q”.) The second line says that the probabilities of moving one, two, or three spaces are 0.6, 0.3, and 0.1. The rest is a map of the environment, where a dash is an empty space, P is a pit, and G is gold.

Your job is to **finish the code in ice.py for mdp\_solve() and q\_solve()**. These take a problem description like the one pictured above, and return a policy giving the recommended action to take in each empty square (U=up, R=right, D=down, L=left).

1. mdp\_solve() should use value iteration and the Bellman equation. ITERATIONS will refer to the number of complete passes you perform over all states. You can initialize the utilities to the rewards of each state. Don’t update the rewards spaces from their initial rewards; since they end the trial, they have no future utility. Don't update utilities in-place as you iterate through them, but create a fresh array of utilities with each pass, in order to avoid biasing moves in the directions that have already been updated.
2. q\_solve() will run ITERATIONS trials in which a learner starts in a random *empty* square and moves until it hits a pit or gold, in which case, the trial is over. (If it was randomly dropped into gold or a pit, the trial is immediately over.) The learner moves by deciding randomly whether to choose a random direction (with probability EXPLORE\_PROB) or move according to the best Q-value of its current square (otherwise). Simulate the results of the move on slippery ice to determine where the learner ended up - then apply the Q-learning equation given in lecture and the textbook. (Do not use SARSA, the very similar other kind of Q-learning that you may encounter directions for online. The primary difference is that SARSA uses the action actually taken from the new square, while our Q-learning uses the best action. These are different when the next action is an “explore” action.)

The fact that a trial ends immediately on finding gold or a pit means that we want to handle those spaces in a special way. Normally Q values are updated on moving to the next state, but we won’t see any next state in these cases. So, to handle this, when the agent discovers one of these rewards, set all the Q values for that space to the associated reward before quitting the trial. So, for example, if gold is worth 100 and it’s discovered in square x, Q(x,UP) = 100, Q(x,RIGHT) = 100, Q(x, DOWN) = 100, and Q(x, LEFT) = 100. There’s no need to apply the rest of the Q update equation when the trial is ending, because that’s all about future rewards, and there’s no future when the trial is ending. But now the spaces that can reach that space will evaluate themselves appropriately. (Before being "discovered," the square should have no utility.)

You should use the GOLD\_REWARD, PIT\_REWARD, LEARNING\_RATE, and DISCOUNT\_FACTOR constants at the top of the file.

Q-learning involves a lot of randomness and some arbitrary decisions when breaking ties. The provided sample results in the .out files for the provided .txt maps are what the solution code does, but results that are very close to those are fine. *Notice that your policy should make sense: an agent following the policy should be able to reach gold.*

Small deviations from the instructions or provided code could result in slightly different policies, but this could be okay if the change doesn't matter (using a different RNG seed, trying directions in a different order, etc). However, the policy must make sense.

When you’ve finished both solvers, try running on the files bigMDP.txt and bigQ.txt with the command-line argument “eval”, and save the results to text files:

python3 ice.py < bigMDP.txt > bigMDPout.txt python3 ice.py < bigQ.txt > bigQout.txt

**To submit**

ice.py

BigMDPout.txt

BigQout.txt

Answers from Part A