COMPSCI 383 – Fall 2021

Homework 5 Coding

Due Tuesday, November 16th at 11:59pm ET

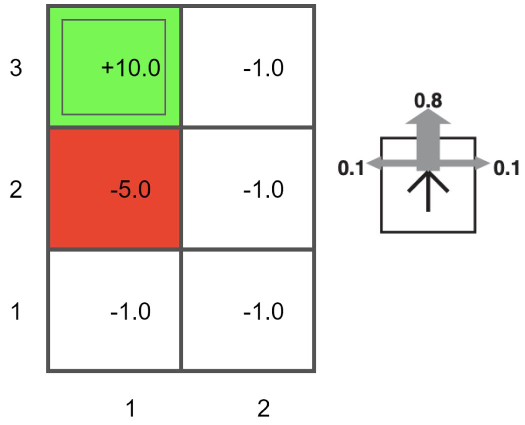
You are encouraged to discuss the assignment in general with your classmates, and may optionally collaborate with one other student. If you choose to do so, you must indicate with whom you worked. Multiple teams (or non-partnered students) submitting the same code will be considered plagiarism.

Code must be written in a reasonably current version of Python (>3.6), and be executable from a Unix command line. You are free to use Python's standard modules for data structures and utilities, as well as the pandas, scipy, and numpy modules.

Value Iteration and Policies for Gridworld

For this short coding assignment, you will implement Value Iteration for grid world MDPs. The goal of the assignment is to build off of your understanding of the algorithm from the Homework 5 Primer, translate it into code, and see the effects of different parameters on policies.

Recall the gridworld MDP shown on the right from the primer. The single terminal state (1, 3) has a reward of +10, the non-terminal (1, 2) has reward -5, and all other states have a reward of -1.



The agent makes its intended move (up, down, left, or right) with a probability 0.8, and moves in a perpendicular direction with probability 0.1 for each side (e.g., if intending to go right, the agent can move up or down with a probability of 0.1 each). If the agent runs into a wall, it stays in the same place.

The only code you need to work on is found in `mdp.py`. To make your life easier, we have supplied you with a simple MDP framework for you to work with. Details on the different methods can be found in the comments and docstrings in the code.

1. Implementing Value Iteration (30 points)

Implement the `value_iteration()` and `derive_policy()` functions in `mdp.py`, and fix the code that calls them in the `if __name__ == ‘__main__’` block (see the code for details). You should utilize the methods provided, but are free to add additional methods and functions as you see fit. Before coding up your solution, you should complete Question 2 in the Homework 5 Primer in order to understand the algorithm. You can use these answers to verify your code's correctness and vice versa.

Next, run your value iteration code using a discount factor of $\gamma = 0.9$ and convergence threshold of $\epsilon = 0.01$. Fill in the utilities corresponding to different iterations of the algorithm in the table on the left below. The final row should contain values your algorithm produced after convergence. On the right, draw arrows showing the policy derived for the non-terminal states (`ascii_grid_utils()` and `ascii_grid_policy()` may be helpful for completing the tables and verifying your answers from the primer).

Utilities

s	**(1, 1)**	**(2, 1)**	**(1, 2)**	**(2, 2)**	**(2, 3)**
$U_0(s)$	-1	-1	-5	-1	-1
$U_2(s)$	-.146	-2.71	2.178	3.31	6.57
$U_5(s)$	1.32	2.69	2.879	4.876	7.275
$U^*(s)$	1.54	2.98	2.9	4.967	7.3

Policy

3	▨	←
2	↑	↑
1	→	↑
	1	2

2. Convergence (5 points)

How many iterations did it take for the utilities calculated by value iteration above to converge? How many iterations did it take for the derived policy to stabilize?

Utilities: 10 Policy: 7

3. The Big Bad (5 points)

Modify the parameters in your code so that state (1, 2) has a reward of -100 and run value iteration. Fill in the table on the right with the policy derived from the converged utilities, and provide an intuitive explanation (1-2 sentences) of why this did or did not result in a changed policy.

Policy

3	▨	←
2	↑	→
1	↓	↑
	1	2

This did result in a changed policy. This is because the penalty is so extreme for venturing into (1, 2) that it is more beneficial to avoid it at all costs so this means instead of trying to directly trying to move from a state where (1, 2) is to the left or right of you into another state, you must travel in the opposite direction of (1, 2) in hopes of getting the .1 chance where you travel to the desired state.

4. Less Certain (5 points)

Restore the original reward structure, but modify the stochasticity of the transition probabilities so that the agent achieves the desired action with a probability of 0.5 (moving 90 degrees to either side with a probability of 0.25 each). Show the derived policy on the right, and explain the result in 1-2 sentences.

Policy

3	▨	←
2	↑	↑
1	↑	↑
	1	2

Now the probability of not going in the intended direction is greater. So now the best course of action is taking a more direct route and the penalty of going in (1, 2) doesn't matter as much. This is because if you take a longer route you are more likely to get more unintended moves making the penalty more and more.

5. Heavy Discount (5 points)

After restoring the original transition probabilities, change the discount factor to 0.6 and fill in the resulting policy in the table on the right. Provide a 1-2 sentence explanation of why this policy differs from those above.

Policy

3	▨	←
2	↑	↑
1	↑	↑
	1	2

Now the discount factor has decreased. This causes the reward value from the current state to outweigh the add affect from gamma * max. So the utilities of the states will be smaller. except for the terminal state which doesn't change. Since (1, 2) is next to the terminal state, even though it's penalty is large it still ends up being more desirable because of the reward of the terminal state.

What to Submit

You should submit your version of the Python file `mdp.py`, a file named `homework5code.pdf` with your answers and tables for the questions above. Additionally, create a `readme.txt` containing:

- Your name(s)
- Any noteworthy resources or people you consulting when doing your project
- Notes or warnings about what you got working, what is partially working, and what is broken