Stanford CME 241 (Winter 2024) - Assignment 3

Due: Jan 29 @ 11:59pm Pacific Time on Gradescope.

Assignment instructions:

- **Please solve questions 1 and 2, and choose one of questions 3 or 4.**
- Empty code blocks are for your use. Feel free to create more under each section as needed.

Submission instructions:

- When complete, fill out your publicly available GitHub repo file URL below, then export or print this .ipynb file to PDF and upload the PDF to Gradescope.

Link to this ipynb file in your public GitHub repo (replace below URL with yours):

https://github.com/zachwitz/CME241/blob/main/assignment3.ipynb
(https://github.com/zachwitz/CME241/blob/main/assignment3.ipynb)

Imports

In [ ]:
```
import numpy as np
```

Question 1

Analytic Optimal Actions and Cost. Consider a continuous-states, continuous-actions, discrete-time, non-terminating MDP with state space as $\mathbb{R}$ and action space as $\mathbb{R}$. When in state $s \in \mathbb{R}$, upon taking action $a \in \mathbb{R}$, one transitions to next state $s' \in \mathbb{R}$ according to a normal distribution $s' \sim \square(s, \sigma^2)$ for a fixed variance $\sigma^2 \in \mathbb{R}^+$. The corresponding cost associated with this transition is $e^{as'}$, i.e., the cost depends on the action a and the state s' one transitions to. The problem is to minimize the infinite-horizon {\em Expected Discounted-Sum of Costs} (with discount factor $\gamma < 1$). For this assignment, solve this problem just for the special case of $\gamma = 0$ (i.e., the myopic case) using elementary calculus. Derive an analytic expression for the optimal action in any state and the corresponding optimal cost.

We want to minimize the expected value of $e^{as'}$ with respect to a, where s' is distributed normally with mean s and variance σ^2. Note that $\mathbb{E}[e^{as'}]$ is the moment generating function of a normal distribution, which equals to $\exp(sa + \frac{\sigma^2 a^2}{2})$. Now we want to find $\arg\min$ of this function. Since $\exp$ is a monotone function, we want to find a min value of the quadratic function of a, which is the vertex of a parabola. Therefore, $a^* = -\frac{s}{\frac{2\sigma^2}{2}} = -\frac{s}{\sigma^2}$.

Question 2

Manual Value Iteration. Consider a simple MDP with $\square = \{s_1, s_2, s_3\}, \square = \{s_3\}, \square = \{a_1, a_2\}$. The State Transition Probability function

$$\mathcal{P} : \mathcal{N} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$$

is defined as:

$$\mathcal{P}(s_1, a_1, s_1) = 0.2, \mathcal{P}(s_1, a_1, s_2) = 0.6, \mathcal{P}(s_1, a_1, s_3) = 0.2$$

$$\mathcal{P}(s_1, a_2, s_1) = 0.1, \mathcal{P}(s_1, a_2, s_2) = 0.2, \mathcal{P}(s_1, a_2, s_3) = 0.7$$

$$\mathcal{P}(s_2, a_1, s_1) = 0.3, \mathcal{P}(s_2, a_1, s_2) = 0.3, \mathcal{P}(s_2, a_1, s_3) = 0.4$$

$$\mathcal{P}(s_2, a_2, s_1) = 0.5, \mathcal{P}(s_2, a_2, s_2) = 0.3, \mathcal{P}(s_2, a_2, s_3) = 0.2$$

The Reward Function

$$\mathcal{R} : \mathcal{N} \times \mathcal{A} \to \mathbb{R}$$

is defined as:

$$\mathcal{R}(s_1, a_1) = 8.0, \mathcal{R}(s_1, a_2) = 10.0$$

$$\mathcal{R}(s_2, a_1) = 1.0, \mathcal{R}(s_2, a_2) = -1.0$$

Assume discount factor $\gamma = 1$.

Your task is to determine an Optimal Deterministic Policy {\em by manually working out} (not with code) simply the first two iterations of Value Iteration algorithm.

- Initialize the Value Function for each state to be it's $\max$ (over actions) reward, i.e., we initialize the Value Function to be $v_0(s_1) = 10.0, v_0(s_2) = 1.0, v_0(s_3) = 0.0$. Then manually calculate $q_k(\cdot, \cdot)$ and $v_k(\cdot)$ from $v_{k-1}(\cdot)$ using the Value Iteration update, and then calculate the greedy policy $\pi_k(\cdot)$ from $q_k(\cdot, \cdot)$ for $k = 1$ and $k = 2$ (hence, 2 iterations).
- Now argue that $\pi_k(\cdot)$ for $k > 2$ will be the same as $\pi_2(\cdot)$. Hint: You can make the argument by examining the structure of how you get $q_k(\cdot, \cdot)$ from $v_{k-1}(\cdot)$. With this argument, there is no need to go beyond the two iterations you performed above, and so you can establish $\pi_2(\cdot)$ as an Optimal Deterministic Policy for this MDP.

The initial value function is set as:

$$
\begin{aligned}
v_0(s_1) &= 10.0, \\
v_0(s_2) &= 1.0, \\
v_0(s_3) &= 0.0.
\end{aligned}
$$

Iteration 1

We calculate $q_1(s, a)$ for each state-action pair:

$$
\begin{aligned}
q_1(s_1, a_1) &= R(s_1, a_1) + \gamma \sum_{s'} P(s_1, a_1, s') v_0(s') = 8.0 + 2.6 = 10.6, \\
q_1(s_1, a_2) &= R(s_1, a_2) + \gamma \sum_{s'} P(s_1, a_2, s') v_0(s') = 10 + 1.2 = 11.2, \\
q_1(s_2, a_1) &= R(s_2, a_1) + \gamma \sum_{s'} P(s_2, a_1, s') v_0(s') = 1.0 + 3.3 = 4.3, \\
q_1(s_2, a_2) &= R(s_2, a_2) + \gamma \sum_{s'} P(s_2, a_2, s') v_0(s') = -1 + 5.3 = 4.3.
\end{aligned}
$$

The policy $\pi_1(s)$ is determined by choosing the action that maximizes $q_1(s, a)$ for each state:

$$
\begin{aligned}
&\pi_1(s_1) = a_2, \quad (\text{since } q_1(s_1, a_2) > q_1(s_1, a_1)), \\
&\pi_1(s_2) = a_1 \quad \text{or} \quad a_2 \quad (\text{chosen arbitrarily as } q_1(s_2, a_1) = q_1(s_2, a_2)).
\end{aligned}
$$

Iteration 2

Using the updated value function $v_1(s)$ from iteration 1, we calculate $q_2(s, a)$:

$$
\begin{aligned}
q_2(s_1, a_1) &= R(s_1, a_1) + \gamma \sum_{s'} P(s_1, a_1, s') v_1(s') = 8.0 + 2.24 + 2.58 = 12.82, \\
q_2(s_1, a_2) &= R(s_1, a_2) + \gamma \sum_{s'} P(s_1, a_2, s') v_1(s') = 10.0 + 1.12 + 0.86 = 11.98, \\
q_2(s_2, a_1) &= R(s_2, a_1) + \gamma \sum_{s'} P(s_2, a_1, s') v_1(s') = 1.0 + 3.36 + 1.29 = 5.65, \\
q_2(s_2, a_2) &= R(s_2, a_2) + \gamma \sum_{s'} P(s_2, a_2, s') v_1(s') = -1.0 + 5.6 + 1.29 = 5.89.
\end{aligned}
$$

The policy $\pi_2(s)$ is determined by choosing the action that maximizes $q_2(s, a)$ for each state:

$$
\begin{aligned}
&\pi_2(s_1) = a_1, \quad (\text{since } q_2(s_1, a_1) > q_2(s_1, a_2)), \\
&\pi_2(s_2) = a_2, \quad (\text{since } q_2(s_2, a_2) > q_2(s_2, a_1)).
\end{aligned}
$$

After $\pi_2(s_1) = a_1$. We know that if we were to take a_2, then it's very likely (0.7) to transition to s_3, a state with no reward (and is terminal so no future reward either). Therefore, as $\gamma = 1$, we'd value every future reward just as much as the current reward, so avoiding terminal states is paramount because the higher chance of ending up in a terminal state results in cutting off potentially arbitrarily high future rewards. The same applies for $\pi_2(s_2) = a_2$), since this results in significantly lower probability of reaching s_3, and therefore doesn't cut off the future reward potential like doing a_1 would.

Question 3

Job-Hopping and Wages-Utility-Maximization. You are a worker who starts every day either employed or unemployed. If you start your day employed, you work on your job for the day (one of n jobs, as elaborated later) and you get to earn the wage of the job for the day. However, at the end of the day, you could lose your job with probability $\alpha \in [0, 1]$, in which case you start the next day unemployed. If at the end of the day, you do not lose your job (with probability $1 - \alpha$), then you will start the next day with the same job (and hence, the same daily wage). On the other hand, if you start your day unemployed, then you will be randomly offered one of n jobs with daily wages $w_1, w_2, \ldots w_n \in \mathbb{R}^+$ with respective job-offer probabilities $p_1, p_2, \ldots p_n \in [0, 1]$ (with $\sum_{i=1}^{n} p_i = 1$). You can choose to either accept or decline the offered job. If you accept the job-offer, your day progresses exactly like the {\em employed-day} described above (earning the day's job wage and possibly (with probability α) losing the job at the end of the day). However, if you decline the job-offer, you spend the day unemployed, receive the unemployment wage $w_0 \in \mathbb{R}^+$ for the day, and start the next day unemployed. The problem is to identify the optimal choice of accepting or rejecting any of the job-offers the worker receives, in a manner that maximizes the infinite-horizon {\em Expected Discounted-Sum of Wages Utility}. Assume the daily discount factor for wages (employed or unemployed) is $\gamma \in [0, 1)$. Assume Wages Utility function to be $U(w) = \log(w)$ for any wage amount $w \in \mathbb{R}^+$. So you are looking to maximize

$$\mathbb{E}[\sum_{u=t}^{\infty} \gamma^{u-t} \cdot \log(w_{i_u})]$$

at the start of a given day t (w_{i_u} is the wage earned on day u, $0 \leq i_u \leq n$ for all $u \geq t$).

- Express with clear mathematical notation the state space, action space, transition function, reward function, and write the Bellman Optimality Equation customized for this MDP.
- You can solve this Bellman Optimality Equation (hence, solve for the Optimal Value Function and the Optimal Policy) with a numerical iterative algorithm (essentially a Dynamic Programming algorithm customized to this problem). Write Python code for this numerical algorithm. Clearly define the inputs and outputs of your algorithm with their types (int, float, List, Mapping etc.). For this problem, don't use any of the MDP/DP code from the git repo, write this customized algorithm from scratch.

State Space: Let
$S = \{(employed, 1), (employed, 2) \ldots (employed, n), (unemployed, 1), (enemployed, 2) \ldots (unemployec$
where there are n jobs. A given state $(unemployed, i)$ for $1 \leq i \leq n$ means that one is currently unemployed and offered job i, and the state $(employed, i)$ means that one is currently working in job i.

Action Space: The possible actions are $A =$ {take job, don't take job}. We only actuallly do an action when we are unemployed, otherwise the action isn't meaningful and is ignored.

Reward Function: Let $R(s_i) = w_i$ where s_i is the current job (or unemployment in the case of $i = 0$), and w_i is the daily salary (or enemployement benefit in the case of $i = 0$).

Transition Function:

Case 1 - (state employment status = employed, any action): {stay at job: $1 - \alpha$, unemployed and offered job i: $\alpha \cdot p_i$}

Case 2 - (state employment status = unemployed, action = take job): {employed at job offered: 1}

Case 3 - (state employment status = unemployed, action = don't take job): {unemployed and offered job i: p_i}

Bellman Equation: $Q^*(\text{state}, \text{action}) = \log(w_i) + \gamma \cdot \sum_{s' \in S} T(\text{state}, \text{action}, s') \cdot \max_{a' \in A} Q^*(s', a')$

In [ ]:
```
# States are (employed, job working) or (unemployed, job offered)
states = [("employed", 1), ("employed", 2), ("unemployed", 1), ("unemp
loyed", 2)]
states_to_idx = {("employed", 1): 0, ("employed", 2): 1, ("unemploye
d", 1): 2, ("unemployed", 2): 3}
# Wages are the reward mapping
wages = [40, 100, 30, 30]
# Probabilities of being offered a given job
p1 = 0.4
p2 = 0.6
alpha = 0.05
gamma = 0.9
# actions are 1: take job, or 0: don't take job
num_states = 4
num_actions = 2

qTable = np.random.rand(num_states, num_actions)
transition_map = {}

for state in states:
    for action in [0,1]:
        # Probabilities of bring fired and then given either job 1, jo
b 2, or just not being fired
        if state[0] == "employed":
            transition_map[(state, action)] = {("employed", state[1]):
1-alpha, ("unemployed", 1): p1 * alpha, ("unemployed", 2): p2 * alpha}
        # Probabilities of being unemployed and offered either job 1 o
r job 2 and then not accepting the job
        elif action == 0:
            transition_map[(state, action)] = {("unemployed", 1): p1,
("unemployed", 2): p2}
        # Probabilities of being unemployed and offered either job 1 o
r job 2 and then accepting the job
        else:
            transition_map[(state, action)] = {("employed", state[1]):
1}

# Bellman Update
for i in range(1000):
    for state in range(len(qTable)):
        for action in range(len(qTable[0])):
            temp = 0
            s = states[state]
            for next_state in transition_map[(s, action)]:
                temp += transition_map[(s,action)][next_state] * np.ma
x(qTable[states_to_idx[next_state]])
            qTable[state, action] = np.log(wages[state]) + gamma * tem
p
# Rows are states (so first 2 rows are employed for job 1 or job 2, se
cond 2 rows are unemployed with offers of job 1 or job 2)
# Cols are actions, don't take job is col 0 and take job is col 1
# We expect (and see) that elements in the first 2 rows in both cols a
re the same because action doesn't matter when you already have a job
# We also expect (and see) that the first element in the last 2 rows a
re the same because not taking either job while
# unemployed leads to the same outcome (reward and future state probab
ilities)
```

```
print(qTable)
```

```
[[38.94152439 38.94152439]
 [45.26077082 45.26077082]
 [42.55402904 38.44856934]
 [42.55402904 44.13589112]]
```

Question 4

Two-Stores Inventory Control. We extend the capacity-constrained inventory example implemented in rl/chapter3/simple_inventory_mdp_cap.py (https://github.com/TikhonJelvis/RL-book/blob/master/rl/chapter3/simple_inventory_mdp_cap.py) as a `FiniteMarkovDecisionProcess` (the Finite MDP model for the capacity-constrained inventory example is described in detail in Chapters 1 and 2 of the RLForFinanceBook). Here we assume that we have two different stores, each with their own separate capacities C_1 and C_2, their own separate Poisson probability distributions of demand (with means λ_1 and λ_2), their own separate holding costs h_1 and h_2, and their own separate stockout costs p_1 and p_2. At 6pm upon stores closing each evening, each store can choose to order inventory from a common supplier (as usual, ordered inventory will arrive at the store 36 hours later). We are also allowed to transfer inventory from one store to another, and any such transfer happens overnight, i.e., will arrive by 6am next morning (since the stores are fairly close to each other). Note that the orders are constrained such that following the orders on each evening, each store's inventory position (sum of on-hand inventory and on-order inventory) cannot exceed the store's capacity (this means the action space is constrained to be finite). Each order made to the supplier incurs a fixed transportation cost of K_1 (fixed-cost means the cost is the same no matter how many units of non-zero inventory a particular store orders). Moving any non-zero inventory between the two stores incurs a fixed transportation cost of K_2.

Model this as a derived class of `FiniteMarkovDecisionProcess` much like we did for `SimpleInventoryMDPCap` in the code repo. Set up instances of this derived class for different choices of the problem parameters (capacities, costs etc.), and determine the Optimal Value Function and Optimal Policy by invoking the function `value_iteration` (or `policy_iteration`) from file rl/dynamic_programming.py (https://github.com/TikhonJelvis/RL-book/blob/master/rl/dynamic_programming.py).

Analyze the obtained Optimal Policy and verify that it makes intuitive sense as a function of the problem parameters.

In [ ]: