AI 533 **Intelligent Agents and Decision Making** Winter 2025

OREGON STATE UNIVERSITY
School of Electrical Engineering and Computer Science
Name: Hyuntaek Oh
Email: ohhyun@oregonstate.edu

Total points: 100 Mini Project 1 Due date: Feb 10, 2025

Instructions: This project consists of two parts. Collaboration is not allowed on any part of this project. You are welcome to brainstorm coding practices (including data structures to use) or clarify your understanding of the question with your peers but you cannot code together or copy/re-use solutions. Document who you worked with for this project and cite websites from which you utilized any code (e.g., Stack Overflow, Stack Exchange, ChatGPT) in your code implementation. You must submit a project report (.pdf file) and the code

Part A: Exact Methods for Solving MDPs (55 points)

Consider the gridworld in Figure 1, with two states covered in water and two states with wildfire. Each state is denoted as $< x, y, water, fire >$. "Water" and "fire" are binary values, with 0 denoting the absence of water/fire and 1 denoting the presence of water/fire at location (x, y). The start state is denoted as $< 0, 0, 0, 0 >$ and the goal state is denoted by $< 3, 3, 0, 0 >$. The agent can move in all four directions. The agent succeeds with probability 0.8 and may slide to the neighboring cells with probability 0.1. Illustration of the transition probability for actions 'up' and 'right' are shown in Figure 1. If a move is invalid (such as moving into a wall), the agent will remain in that state with the corresponding probability. For example, when trying to move up in top-left cell (start state) of the grid, the agent will remain in that cell with probability 0.9 or move right with probability 0.1.

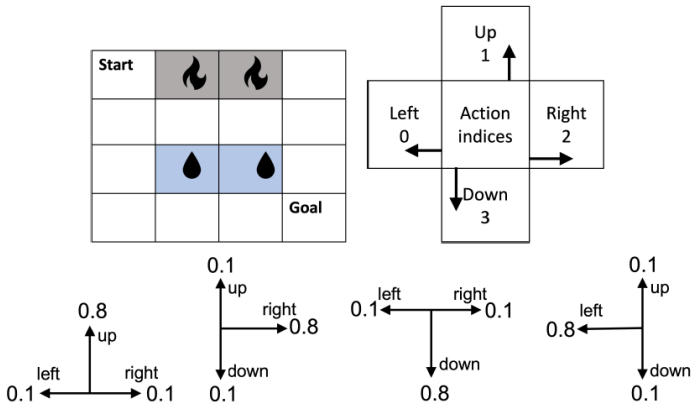

Figure 1: MDP

Figure 1: MDP

1. **(15 points)** Setup (code up) the environment for solving using value iteration and policy iteration.

   Environment consists of 4×4 gridworld. Each cell has its own reward, -1 if agent moves to another cell, -5 if agent places in water area, -10 if agent is in fire area, 100 if agent reaches the goal state. Figure 2 shows the reward structure.

```
[[ -1. -10. -10.  -1.]
 [ -1.  -1.  -1.  -1.]
 [ -1.  -5.  -5.  -1.]
 [ -1.  -1.  -1. 100.]]
```

Figure 2: Reward Structure

Agent can take four possible actions $[left,\ up,\ right,\ and\ down]$, and each action has 80% success and 20% fail that can move other directions called slides. The values of γ (gamma) is $[0.3,\ 0.95]$. Thus, the environment parameters and reward setting are:

```python
# Environment parameter
ENV_ROWS, ENV_COLS = 4, 4
ACTIONS = [(0, -1), (-1, 0), (0, 1), (1, 0)]    # left: 0, up: 1, right: 2, down: 3
TEXT_ACTIONS = ['left', 'up', 'right', 'down']
SLIDES = {0: [(-1, 0), (1, 0)],
          1: [(0, -1), (0, 1)],
          2: [(-1, 0), (1, 0)],
          3: [(0, -1), (0, 1)]}
SLIDES_IDX = {0: [1, 3],
              1: [0, 2],
              2: [1, 3],
              3: [0, 2]}
START = (0, 0)
GOAL = (3, 3)
WATER_1, WATER_2 = (2, 1), (2, 2)
FIRE_1, FIRE_2 = (0, 1), (0, 2)
PROB_SUCCESS = 0.8
PROB_SLIDE = 0.1
GAMMA = [0.3, 0.95]

# Rewards setting
REWARD = -np.ones((ENV_ROWS, ENV_COLS))  # all other states
REWARD[WATER_1], REWARD[WATER_2] = -5, -5     # Water
REWARD[FIRE_1], REWARD[FIRE_2] = -10, -10   # Fire
REWARD[GOAL] = 100  # Goal
```

Figure 3: Environment and Reward Setting

2. **(25 points)** Implement value iteration for this problem with $\gamma = 0.3$ and $\gamma = 0.95$. Your code must output the policy and objective value for each setting. Include the resulting policy and the objective value in your report. Describe whether the policy in states leading to wildfire and water changed, as the discount factor is increased.

   As the discount factor γ increases, the policy table in states is changed, leading to water zone. Figure 4 shows the policy tables when the γ value is 0.3 and 0.95 respectively. When the γ is 0.3, the best policy is $[down, down, down, right, right, right, right]$, indicating that it tries to avoid water and fire zones as much as possible. On the other hand, as the γ is 0.95, the best policy is $[down, right, right, down, right, down, right]$, implying that it struggled to reach the goal as soon as possible even though water zone can provide not a little negative reward.

```
Current Gamma value is < 0.3 >
output policy:
 [['down' 'down' 'down' 'down']
 ['down' 'right' 'right' 'down']
 ['down' 'down' 'right' 'down']
 ['right' 'right' 'right' 'right']]

value table:
 [[ -1.42 -10.66 -10.17   0.44]
 [ -1.31  -1.28   0.55   7.23]
 [ -0.96  -3.2    4.01  33.33]
 [  0.7    7.12  33.33 138.36]]
```

```
Current Gamma value is < 0.95 >
output policy:
 [['down' 'down' 'down' 'down']
 ['right' 'right' 'down' 'down']
 ['right' 'right' 'right' 'down']
 ['right' 'right' 'right' 'right']]

value table:
 [[1199.31 1257.47 1340.14 1425.05]
 [1272.25 1350.29 1441.19 1530.73]
 [1347.51 1437.15 1537.49 1643.95]
 [1425.47 1530.31 1643.95 1766.72]]
```

Figure 4: Value Iteration with $\gamma = 0.3$ (left) and $\gamma = 0.95$ (right)

3. **(15 points)** Implement policy iteration for this problem with $\gamma = 0.95$. Your code must output the policy and objective value for each setting. Include the resulting policy and the objective value in your report. Is the policy and the objective value same as that of value iteration?

   The policy iteration for this problem with $\gamma = 0.95$ shows as same result as value iteration with $\gamma = 0.95$ produced. Figure 5 displays the policy table and value table. As can be seen, the policy iteration also leads to water zone to reach the goal state. The best policy is $[down, down, down, right, right, right, right]$, suggesting that it brings out the maximum values when the agent choose one of the state at each state.

```
Current Gamma value is < 0.95 >
output policy:
 [['down' 'down' 'down' 'down']
 ['right' 'right' 'down' 'down']
 ['right' 'right' 'right' 'down']
 ['right' 'right' 'right' 'right']]

value table:
 [[1199.31 1257.47 1340.14 1425.05]
 [1272.25 1350.29 1441.19 1530.73]
 [1347.51 1437.15 1537.49 1643.95]
 [1425.47 1530.31 1643.95 1766.72]]
```

Figure 5: Policy Iteration with γ=0.95

Part B: Imitation Learning (45 points)

1. **(15 points)** Policy simulation. Write a function to generate **an episode** for the grid in Part A, with $\gamma = 0.95$. Each episode must begin at the start state and ends at the goal state. An episode is of the form $(S_0, A_0, R_0), ..., (S_T, A_T, R_T)$ where S_0 is the start state, A_0 is $\pi(S_0)$, R_0 is the corresponding reward, S_T is the Goal state, A_T is $\pi(S_T)$ and R_T is the corresponding reward $(R_t(s, a), \forall t)$. Given a policy π as input, the function must output an episode (or trajectory). The trajectory must follow the state transition function T. For example, when moving "right" at the start state, the function must select a single successor state among the fire state, moving down or remaining in the start state, proportional to their transition probabilities. Due to the stochastic transitions, the trajectories may be different each time you simulate the policy.

   Given the same environment and the policy table with $\gamma = 0.95$ in Part A, agent starts in the starting state, (0, 0) and finish its task when reaching the goal state. While moving to other cells to achieve its own objective, agent may transfer to not intended cell because of transition probabilities. Figure 6 shows the results of generated episodes.

```
Generate an episode

[(0, 0), 'down', -1.0]
[(1, 0), 'down', -1.0]
[(2, 0), 'right', -1.0]
[(2, 1), 'right', -5.0]
[(2, 2), 'down', -5.0]
[(3, 2), 'down', -1.0]
[(3, 2), 'up', -1.0]
[(2, 2), 'right', -5.0]
[(2, 3), 'right', -1.0]
[(2, 3), 'down', -1.0]
[(3, 3), 'right', 100.0]
```

```
Generate an episode

[(0, 0), 'down', -1.0]
[(1, 0), 'down', -1.0]
[(2, 0), 'right', -1.0]
[(2, 1), 'right', -5.0]
[(2, 2), 'down', -5.0]
[(3, 2), 'right', -1.0]
[(3, 3), 'right', 100.0]
```

```
Generate an episode

[(0, 0), 'left', -1.0]
[(0, 0), 'right', -1.0]
[(0, 1), 'down', -10.0]
[(1, 1), 'right', -1.0]
[(1, 2), 'down', -1.0]
[(2, 2), 'right', -5.0]
[(2, 3), 'down', -1.0]
[(3, 3), 'right', 100.0]
```

```
Generate an episode

[(0, 0), 'left', -1.0]
[(0, 0), 'down', -1.0]
[(1, 0), 'down', -1.0]
[(2, 0), 'right', -1.0]
[(2, 1), 'right', -5.0]
[(2, 2), 'right', -5.0]
[(2, 3), 'down', -1.0]
[(3, 3), 'right', 100.0]
```

Figure 6: Generated episodes

   As shown in Figure 6, due to the transition function T, agent can move the cells that it may not intend to do. Even agent starts at the same place (0, 0), the agent provides a variety of state and action such as (0, 0) and (1, 0). The trajectories above indicate that agent tried to avoid fire zone immediately but pass through the water zone. Though some trajectories that visited water zone two times, it may be because those actions produce results that are probably manageable to achieve the goal.

2. **(30 points)** We will implement DAGGER algorithm (Algorithm 3.1 in *https://arxiv.org/pdf/1011.0686*), with some modifications: (i) we will use the optimal policy from Part A to simulate the human expert and (ii) $\beta_i = 0$. The following are the high-level steps to follow in your implementation.

   (a) Initialize a random policy $\hat{\pi}$ such that is it not the optimal policy $\hat{\pi} \neq \pi^*$ and a empty dataset $D = \{\}$. For $i = 1$ to N, repeat steps (b) - (e).

   (b) Generate a trajectory τ_i, using $\hat{\pi}$ and B.1. Create a data set D_i by annotating each state in trajectory τ_i with corresponding expert actions from π^* (policy computed using value iteration or policy iteration in Part A with $\gamma = 0.95$). Aggregate data set $D = D \cup D_i$.

   (c) Train a classifier of your choice using D. The training input is the states in D and the input labels are the corresponding actions in D.

   (d) For simplicity and consistency, the test set will consist of *all* states in the MDP.

   (e) Predict the action label for each state in the test set, using a classifier. This will produce a new policy $\hat{\pi}$

Conduct experiments with $N = \{5, 10, 20, 30, 40, 50\}$ and plot accuracy. The x-axis of your graph is the N and the y-axis is the accuracy. The accuracy is calculated as the fraction of states for which the predicted action $\hat{\pi}$ in (e) matches the optimal policy π^* from Part A. Did the accuracy increase with an increase in N? Why and why not?

Given the set of N $5, 10, 20, 30, 40, 50$, the result of the model accuracy by using decision tree with 5 max depth is in Figure 7. The plot in Figure 7 indicates that $N = 5$ case was the lowest accuracy(even it is the lowest, it is nearly 93%), and other N values provide same accuracy, 100%. From the first experiment $N = 5$ to the second $N = 10$, there is an increase in accuracy but not the others $N = 20, 30, 40, 50$. This means that at first, the model learned from the small size of accumulative (state, action) dataset that is derived from the optimal policy π^* in the same environment, showing relatively lower performance. Then, after learning from a huge amount of the stacking dataset D, the model's ability to predict next optimal action was gradually improved and well optimized. However, it may differ from using different learning models such as random forest and support vector machines.

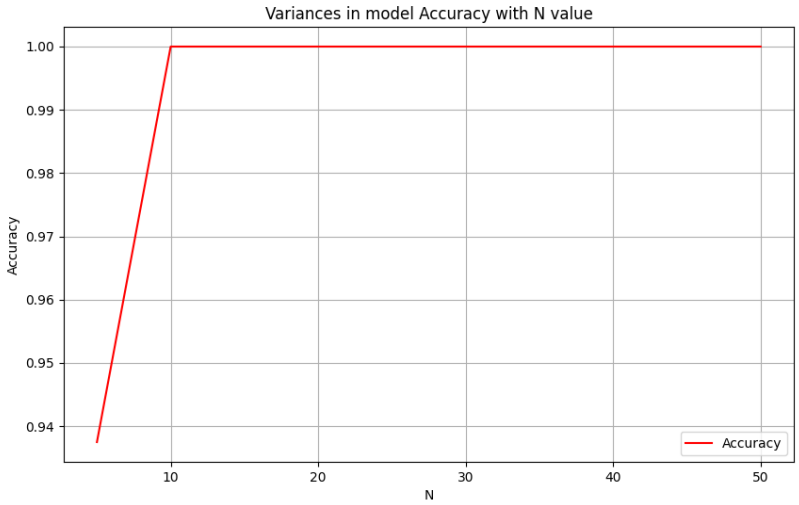

Figure 7: Accuracy of the model in DAGGER algorithm