MEIC

Planning, Learning and Intelligent Decision Making

Homework 2

Grupo 23:

André Ramos Loja Infante Barbosa (103008)

Artur Filipe de Menezes Crespo Ferreira (102482)

2024/2025 – 2º Semestre, P3

Conteúdo

1 Problem Definition

The problem is a Markov Decision Process (MDP) where a spider climbs a ladder with probabilistic transitions, influenced by weather conditions. A new action, `Stop`, allows the spider to build a web at state 2, preventing it from falling completely to the ground. The discount factor is given as $\gamma = 0.9$.

2 MDP Components

2.1 State Space

We introduce additional states to represent the web scenario:

$$X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\} \tag{1}$$

where states $6 - 11$ correspond to the web-enabled ladder states.

2.2 Action Space

The spider has two available actions:

- **Play (P)**: Moves up the ladder based on die rolls.
- **Stop (S)**: Builds a web at state 2; stopping elsewhere results in falling.

2.3 Transition Matrices

The transition probability matrix for the `Play` action (extended to account for web states) is:

$$P_{\text{play}} = \begin{bmatrix} 0.2 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0.4 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0.4 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0.4 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0.4 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0.8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

For the `Stop` action, the matrix of transitions is like follows:

$$P_{\text{stop}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Part (b): Cost-to-Go for Two Deterministic Policies in State 2

We compute the cost-to-go $J^{\pi}(s)$ for the two deterministic policies in state 2.

Policy 1: Always "Play"in State 2

For **Policy 1**, the spider always chooses to "Play"in state 2. The Bellman equation for state 2 is:

$$J^{\pi_1}(2) = \sum_{s'} P(2, \text{Play}, s') \left[c(2, \text{Play}, s') + \gamma J^{\pi_1}(s') \right]$$

Listing 1: Python Code for Perceptron Training

```

import numpy as np

# Discount factor
gamma = 0.9

# Number of states
n_states = 12

# Transition probability matrix for "Play" action (P_play)
P_play = np.array([
    [0.2, 0.4, 0.4, 0,   0,   0,   0,   0,   0,   0,   0,   0],
    [0.2, 0,   0.4, 0.4, 0,   0,   0,   0,   0,   0,   0,   0],
    [0.2, 0,   0,   0.4, 0.4, 0,   0,   0,   0,   0,   0,   0],
    [0.2, 0,   0,   0,   0.4, 0.4, 0,   0,   0,   0,   0,   0],
    [0.2, 0,   0,   0,   0,   0.8, 0,   0,   0,   0,   0,   0],
    [0,   0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0],
    [0,   0,   0,   0,   0,   0,   0.2, 0.4, 0.4, 0,   0,   0],
    [0,   0,   0,   0,   0,   0,   0.2, 0.4,  0.4, 0, 0,   0],
    [0,   0,   0,   0,   0,   0,   0.2, 0,   0,   0.4, 0.4, 0],
    [0,   0,   0,   0,   0,   0,   0.2, 0,   0,   0,   0.4, 0.4],
    [0,   0,   0,   0,   0,   0,   0.2, 0,   0,   0,   0,   0.8],
    [0,   0,   0,   0,   0,   0,   0, 0,   0,   0,   0,   1]
```

```python
])

# Transition probability matrix for "Stop" action (P_stop)
P_stop = np.array([
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
])

# Cost vector (1 everywhere except final states 5 and 11 where cost is 0)
c = np.ones(n_states)
c[5] = 0
c[11] = 0

# Compute J_pi for Play
I = np.eye(n_states)
J_play = np.linalg.solve(I - gamma * P_play, c)

# Compute J_pi for Stop
J_stop = np.linalg.solve(I - gamma * P_stop, c)

# Compute Q-functions
Q_play = c + gamma * np.dot(P_play, J_play)
Q_stop = c + gamma * np.dot(P_stop, J_stop)

# Compute the optimal cost-to-go function J^*(s) (minimum of Q-functions)
J_star = np.minimum(Q_play, Q_stop)

# Print results
print("Optimal Cost-to-Go Function J* considering both Play and Stop:")
print(J_star)
```

The output of the program is the following:

$$\text{Optimal Cost} - \text{to} - \text{Go Function } J* \text{ considering both Play and Stop}: \\ [4.35671816\ 3.84578458\ 3.30007346\ 2.42652461\ 1.78420927\ 0. \\ 4.89747459\ 4.89747459\ 3.48010642\ 2.55890178\ 1.88154543\ 0.]$$

3 Optimal Policy at State 2

The cost-to-go for the two policies in state 2 are: - **Policy 1 (Always "Play")**: $J^{\pi_1}(2) = 3.30$ - **Policy 2 (Always "Stop")**: $J^{\pi_2}(2) = 3.48$

Thus, **Policy 1** is better in state 2 because it has a lower cost-to-go.