< 지능제어 1차 숙제 결과 보고서 >

2018142102 홍의진

1. "main.py"의 "Write your code here" 부분 code

main.py > evaluate_state_value_by_matrix_inversion

```python
def evaluate_state_value_by_matrix_inversion(env, discount=0.9):
    WIDTH, HEIGHT = env.size()

    # Initializing Value function vector V to a numpy zero array with size 25
    V = np.zeros((25,), dtype=float)

    # Reward matrix R
    # Initializing Reward Matrix(vector) R to a numpy zero array with size 25
    R = np.zeros((25,), dtype=float)
    # Transition matrix T
    # Initializing Transition Matrix T to a numpy zero array with size 25 x 25
    T = np.zeros((25, 25), dtype=float)
    # iterate over each state: S0 ~ S24 (25 states total)
    for i in range(WIDTH * HEIGHT):
        # coordinate of the current state in the grid world;
        # the indexing starts from left top and goes through the top-bottom direction)
        current_state = [i//5, i%5]
        # iterate interaction for every action on the current state following the equiprobable policy
        for action in ACTIONS.values():
            # get the next_state(s') and reward(r) from interaction between agent and environment
            # based on the current state and particular action
            next_state, reward = env.interaction(current_state, action)
            # Add up the (reward x action_probability) for a particular action
            # based on the current state and particular action
            R[current_state[0]*5+current_state[1]] += reward * ACTION_PROB
            # Add up the (action_probability x transition_probability) for a particular action
            # based on the current state and particular action
            T[current_state[0]*5+current_state[1], next_state[0]*5+next_state[1]] +=  ACTION_PROB * 1

    # Creating an Identity Matrix of size 25 x 25
    I = np.identity(25, dtype=float)

    # Evaluating the Value function V by calculating the Bellman Equation: ((I - gamma * T)^-1)R
    V = np.linalg.inv(np.subtract(I, discount * T)).dot(R)
```

2. "main.py"의 "Write your code here" 부분 대한 자세한 주석

Line 16, 20, 23 - 우선, 각각의 state에 해당하는 state value function으로 이뤄진 25x1 사이즈의 V 행렬, 각 current state에서 취하는 policy를 통해 얻을 수 있는 reward의 기댓값들로 이뤄진 25x1 사이즈의 R 행렬, 그리고 current state에서 next state로 전환될 확률 값들을 담고 있는 25x25 사이즈의 T 행렬을 각각의 사이즈에 맞게끔 0행렬로 변수 초기화를 해 주었다.

Line 25 - For문을 통해 0~24의 index i를 반복하여, Bellman Equation의 행렬 표현식 $V = R + \gamma TV$ 에 따라 각 current state($s_0 \sim s_{24}$)에 해당하는 $R_i, (i = 0, 1, 2, \dots, 24)$와 현재 policy를 따라 얻어지는 각각의 current state와 next state 쌍에 해당하는 $T_{ij}, (i, j = 0, 1, 2, \dots, 24)$를 설정해 주었다.

Line 28 - 여기서 current_state를 [(i를 5로 나눈 몫), (i를 5로 나눈 나머지)]으로 표현했는데, 이는 가장 좌측 상단에 위치한 grid에 해당하는 state의 index를 0으로 부여하고, 열을 내려가는 방향으로 index를 증가시키며, 각 열에 대하여 우측으로 행을 이동시키는 방향으로 index가 증가하도록 설정한 것이다. 이는 기존에 제공된 environment.py의 코드와의 일관성이 보존된다.

Line 30 - 현재 state에서 equiprobable policy에 의해 시행된 각 action에 대한 reward와 state transition

을 구하고 싶기 때문에, 해당 policy에 의해 시행된 각각의 action(상하좌우)을 for문을 통해 반복시킨다.

Line 33 - Agent가 현재 state에서 다음 state로 전환되는 interaction을 env.interaction(current_state, action)을 통해 시행하고, 그때 얻어지는 다음 state와 reward를 변수 next_state와 reward에 받는다.

Line 36 - 현재 state에서 특정 action을 통해 얻을 수 있는 reward의 값과 해당 action을 취할 확률을 곱하여, 각 action별로 더한 값이 해당 state에서의 reward의 기댓값 R_i가 된다. 현재 agent는 equiprobable policy를 따르기 때문에 특정 action을 취할 확률은 0.25로 모두 동일하다.
여기서 current state의 index i는 앞서 언급한 indexing convention을 유지하기 위해 current_state[0]*5 + current_state[1]로 설정해 주었다. 각각의 state에 대해 R_i값을 모두 구하여 R matrix를 만들 수 있다.

Line 39 - 현재 state에서 특정 action을 취했을 때 특정 다음 state로 전환될 확률과 해당 action을 취할 확률을 곱하여, 각 action별로 더한 값이 '해당 state에서 다음 state로 전환되는 확률'인 T_{ij}가 된다. 현재 agent는 equiprobable policy를 따르기 때문에 특정 action을 취할 확률은 0.25로 모두 동일하다. 또한 이 문제는 deterministic problem으로, 어떤 action을 통해서 다음 state로의 전환은 고정되어 있기에 그 확률은 1이다. 여기서 current state의 index와 next state의 index는 앞서 언급한 indexing convention을 유지하기 위해 current_state[0]*5 + current_state[1]과 next_state[0]*5 + next_state[1]로 설정해 주었다. 각각의 current state와 next state 쌍에 대해 T_{ij}값을 모두 구하여 T matrix를 얻을 수 있다.

Line 42 - 그 후, 각각의 state들의 state value function들로 이뤄져 있는 행렬 V의 값을 행렬 연산을 통해 구하기 위해서 25x25 size의 단위 행렬 I를 정의해 준다.

Line 45 – Bellman Equation의 행렬 표현식을 통해 행렬 V를 직접적으로 구하는 식은 $V = (I - \gamma T)^{-1}R$이고, 이를 numpy 내장함수를 사용하여 구현한 코드는
V = np.linalg.inv(np.subtract(I, discount * T)).dot(R)가 된다.

3. "main.py" 실행 시 visualize된 grid 결과 사진

```
env = grid_world()
values = evaluate_state_value_by_matrix_inversion(env = env)
```

Iteration:1

	1	2	3	4	5
1	3.31	8.79	4.43	5.32	1.49
2	1.52	2.99	2.25	1.91	0.55
3	0.05	0.74	0.67	0.36	-0.4
4	-0.97	-0.44	-0.35	-0.59	-1.18
5	-1.86	-1.35	-1.23	-1.42	-1.98

<그림 1. 각 state에 해당하는 state value function 값들을 grid위에 plot한 모습>

4. Discussion (3번 결과에 대한 분석 포함)

```
✓ 0초 [6] print(values)

[[ 3.30899634  8.78929186  4.42761918  5.32236759  1.49217876]
 [ 1.52158807  2.99231786  2.25013995  1.9075717   0.54740271]
 [ 0.05082249  0.73817059  0.67311326  0.35818621 -0.40314114]
 [-0.9735923  -0.43549543 -0.35488227 -0.58560509 -1.18307508]
 [-1.85770055 -1.34523126 -1.22926726 -1.42291815 -1.97517905]]

✓ 0초 [8] np.mean(values)

0.9045471595249436

✓ 0초 [10] np.var(values)

6.206802620270648

✓ 0초 [11] np.std(values)

2.4913455441328582

✓ 0초 [13] np.median(values)

0.35818621485579033
```

<그림2. State value function 값들과 그들의 평균값, 중간값, 분산, 표준편차>

• <그림 2.>에서 볼 수 있듯이, state들에서의 state value function 값들의 평균값은 약 0.90, 중간값은 약0.36, 분산은 약 6.20, 표준편차는 약 2.49로 나타난다.

• 평균이 중간값보다 크고, 분산과 표준편차가 큰 편이다. 그 이유는 state A와 B, 그리고 그에 인접한 state들에서의 state value function 값들이 이외 state들에서의 state value function 값들보다 월등히 큰 값을 보이기 때문이다.

• Terminal state가 없는 MDP이므로, discount rate가 존재하여 이후 발생하는 reward 값들의 합을 감쇠시킴으로써 state value function을 구하는 식이 특정 값으로 수렴할 수 있도록 해 준다.

• 현재 agent가 취하고 있는 policy는 equiprobable policy이므로 상하좌우로 이동할 확률은 모두 1/4로 동일하게 존재한다.

• A state([0, 1]), B state([0, 3])의 state value function의 값이 두드러지게 큰데, A와 B같은 경우에는 해당 칸에 도달했을 때(즉, 해당 state에서) reward을 각각 10, 5만큼 받는 것이 주되게 반영되었다.

• 그러나, state A 혹은 B에 도달 시 1의 확률로 다음 state(각각 A' [4, 1], B' [2, 3])로 이동하게 되는데, 이 경우 A'은 A와 B로부터의 거리가 비교적 멀어지기(future reward를 증가시키기 위해 A 혹은 B로 이동하는 데 취해야 하는 action의 수가 더 많아지기) 때문에 상대적으로 reward를 얻을 확률은 줄어들고, grid 밖으로 이동하여 – reward을 얻을 확률은 증가한다. 반면, B'은 A와 B로부터의 거리가 비교적 가까운 상태이기 때문에 상대적으로 reward을 얻을 확률은 늘어난다.

• 현재 state에서의 state value function은 미래 state에서의 state value function의 영향을 가중치 γ를 곱한 만큼 반영되기 때문에(Bellman Equation: $v(s) = R + \gamma Tv(s')$) state A에서의 값은 10보다 작고, state B에서의 값은 5보다 크다.

• 한편 state [0,2]에서는 비록 grid 밖으로 벗어날 확률도 존재하지만, 상대적으로 A혹은 B에 도달하는 action을 취하여 얻을 수 있는 reward의 기댓값이 크기 때문에 해당 state에서의 state value function의 값이 다른 state에 비해 상대적으로 큰 값인 4.43이 나왔다.

• 위와 마찬가지의 이유로 grid world에서 A와 B에 인접한, 즉 바로 다음 action을 통해 A와 B로 이동할 수 있는 state들에서는 평균값보다 높은 state value function값을 보였다.

• 한편, grid의 경계에 인접하여 위치한 state들은 같은 행에 위치해 있는 다른 state들에 비해서나 평균값에 비해서 더 작은 값을 나타낸다.

• 그 이유는 바로 다음 action을 통해 grid world 밖으로 이동하여 음수인 reward를 얻을 확률이 존재하고, 이는 state value function을 구하는 데 반영되었기 때문이다.

• 또한, A나 B로부터 상대적으로 멀리 떨어져 있는 4번째와 5번째 행에 위치한 state들은 value function의 값들이 음수를 나타내는데, 이는 해당 state들의 위치가 A나 B와는 멀기 때문에 미래에 A혹은 B에 도달하여 얻을 수 있는 reward의 기댓값보다 grid world 밖으로 이탈하여 감소하게 되는 reward의 기댓값이 상대적으로 더 크기 때문이다.

• 이와 유사한 이유로 모서리에 위치한 state들의 경우에는 grid world 밖으로 이탈하게끔 하는 action의 경우는 두 가지이고, 현재 policy가 equiprobable policy이기 때문에 해당 action이 발생할 확률은 1/2이므로, reward 감소에 대한 기댓값이 상대적으로 더 커진다.

• Grid world의 중앙 부근에 위치한 state들은 value function의 값이 0에 가까운데, 이는 가까운 미래에서 발생할 action들이 가져올 reward의 기댓값이 0에 근접하다는 사실과(grid 안에 머물러 있으면서 A혹은 B가 아닌 다른 state로 이동할 경우 얻는 reward는 0이다), 장기적 미래에서 agent가 A 또는 B에 도달하려 reward을 얻을 기댓값과 grid world 밖으로 나가서 reward을 잃게 되는 기댓값이 상쇄된다는 사실이 반영된 것으로 볼 수 있을 것이다.

• 결과적으로 봤을 때, state A 혹은 B로 향하는 action이 발생할 확률이 높은 policy를 선택하는 것이 직관적으로 봤을 때 전반적으로 더 높은 state value function 값을 보일 것이므로 이에 가장 잘 부합하는 policy가 optimal policy가 될 것이다. 다시 말하면, 현재 취하고 있는 equiprobable policy는 최적의 policy가 아닐 가능성이 매우 높다.