Inotes2 - MDP

Kang, Eui Hyeon

2021-02-01

차 례

As an example of a MDP, consider the example given in Figure 3. The agent is again a Mars rover whose state space is given by S = fS1; S2; S3; S4; S5; S6; S7g. The agent has two actions in each state called **try left** and **try right**, and so the action space is given by A = $\{TL, TR\}$. Taking an action always succeeds, unless we hit an edge in which case we stay in the same state. This leads to the two transition probability matrices for each of the two actions as shown in Figure 3. The rewards from each state are the same for all actions, and is 0 in the states fS2; S3; S4; S5; S6g, while for the states S1; S7 the rewards are 1; 10 respectively. The discount factor for this MDP is some 2 [0; 1].

Preparation

```
import numpy as np
import pandas as pd

states=['S1','S2','S3','S4','S5','S6','S7']
A=['TL','TR']

R=[1,0,0,0,0,0,10]

P_tl=pd.DataFrame(np.matrix([[1,0,0,0,0,0,0],
                [1,0,0,0,0,0,0],
                [0,1,0,0,0,0,0],
                [0,0,1,0,0,0,0],
                [0,0,0,1,0,0,0],
                [0,0,0,0,1,0,0],
                [0,0,0,0,0,1,0]]),index=states, columns=states)

P_tl
```

```
##     S1  S2  S3  S4  S5  S6  S7
```

```
## S1   1   0   0   0   0   0   0
## S2   1   0   0   0   0   0   0
## S3   0   1   0   0   0   0   0
## S4   0   0   1   0   0   0   0
## S5   0   0   0   1   0   0   0
## S6   0   0   0   0   1   0   0
## S7   0   0   0   0   0   1   0
```

```
P_tr=pd.DataFrame(np.matrix([[0,1,0,0,0,0,0],
                [0,0,1,0,0,0,0],
                [0,0,0,1,0,0,0],
                [0,0,0,0,1,0,0],
                [0,0,0,0,0,1,0],
                [0,0,0,0,0,0,1],
                [0,0,0,0,0,0,1]]), index=states, columns=states)

P_tr
```

```
##     S1  S2  S3  S4  S5  S6  S7
## S1   0   1   0   0   0   0   0
## S2   0   0   1   0   0   0   0
## S3   0   0   0   1   0   0   0
## S4   0   0   0   0   1   0   0
## S5   0   0   0   0   0   1   0
## S6   0   0   0   0   0   0   1
## S7   0   0   0   0   0   0   1
```

Exercise 3.17

Consider the MDP discussed above in Figure 3. Let $\gamma = 0$, and consider a stationary policy π which always involves taking the action TL from any state. (a) Calculate the value function of the policy for all states if the horizon is finite. (b) Calculate the value function of the policy when the horizon is infinite. Hint: Use Theorem A.3.

(a)

```
pi_tl=np.c_[np.repeat(1,len(states)), np.repeat(0,len(states))]

pi_tl=pd.DataFrame(data=pi_tl, index=states, columns=A)
pi_tl
```

```
##     TL  TR
## S1   1   0
## S2   1   0
## S3   1   0
## S4   1   0
## S5   1   0
## S6   1   0
## S7   1   0
```

```
R_s_a=pd.DataFrame(np.c_[R,R], index=states, columns=A)

R_s_a
```

```
##     TL  TR
## S1   1   1
## S2   0   0
## S3   0   0
## S4   0   0
## S5   0   0
## S6   0   0
## S7  10  10
```

```
def reward_fn(given_pi):
    R_s_a=pd.DataFrame(np.c_[R,R], index=states, columns=A)

    R_pi=np.asarray((given_pi*R_s_a).sum(axis=1)).reshape(-1,1)

    return R_pi
```

```
reward_fn(pi_tl)
```

```
## array([[ 1],
##        [ 0],
##        [ 0],
##        [ 0],
##        [ 0],
##        [ 0],
##        [10]], dtype=int64)
```

```
def transition(given_pi, states, P_tl, P_tr):
    P_out=pd.DataFrame(np.zeros((len(states),len(states))), index=states, columns=states)

    for s in states:
        action_dist=given_pi.loc[s]
        P=action_dist['TL']*P_tl+action_dist['TR']*P_tr
        P_out.loc[s]=P.loc[s]

    return P_out

transition(pi_tl, states=states, P_tl=P_tl, P_tr=P_tr)
```

```
##      S1   S2   S3   S4   S5   S6   S7
## S1  1.0  0.0  0.0  0.0  0.0  0.0  0.0
## S2  1.0  0.0  0.0  0.0  0.0  0.0  0.0
## S3  0.0  1.0  0.0  0.0  0.0  0.0  0.0
## S4  0.0  0.0  1.0  0.0  0.0  0.0  0.0
## S5  0.0  0.0  0.0  1.0  0.0  0.0  0.0
## S6  0.0  0.0  0.0  0.0  1.0  0.0  0.0
## S7  0.0  0.0  0.0  0.0  0.0  1.0  0.0
```

```
def policy_eval(given_pi):
    R=reward_fn(given_pi)
    P=transition(given_pi, states=states, P_tl=P_tl, P_tr=P_tr)

    gamma=0
    epsilon=10**(-8)

    v_old=np.repeat(0,7).reshape(7,1)
    v_new=R+np.dot(gamma*P, v_old)
```

```
    while np.max(np.abs(v_new-v_old))>epsilon:
        v_old=v_new
        v_new=R+np.dot(gamma*P,v_old)

    return v_new.T


policy_eval(pi_tl)
```

```
## array([[ 1.,  0.,  0.,  0.,  0.,  0., 10.]])
```

(b)

```
def policy_eval(given_pi,gamma):
    R=reward_fn(given_pi)
    P=transition(given_pi, states=states, P_tl=P_tl, P_tr=P_tr)

    gamma=gamma
    epsilon=10**(-8)

    v_old=np.repeat(0,7).reshape(7,1)
    v_new=R+np.dot(gamma*P, v_old)

    while np.max(np.abs(v_new-v_old))>epsilon:
        v_old=v_new
        v_new=R+np.dot(gamma*P,v_old)

    return v_new.T

policy_eval(pi_tl,0.00001)
```

```
## array([[1.00001e+00, 1.00001e-05, 1.00000e-10, 0.00000e+00, 0.00000e+00,
##         0.00000e+00, 1.00000e+01]])
```