Inote2_MDP_Jeong,wonryeol

Jeong, wonryeol

2021-01-31

Contents

Preparation

```
import numpy as np
import pandas as pd

A = ["TL","TR"]
S = ["S1","S2","S3","S4","S5","S6","S7"]

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 = S , columns = S)

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 = S , columns = S)

pi_Left = pd.DataFrame(np.matrix([np.repeat(1,len(S)),np.repeat(0,len(S))]),index = A,columns = S).T
pi_Right = pd.DataFrame(np.matrix([np.repeat(0,len(S)),np.repeat(1,len(S))]),index = A,columns = S).T
pi_50 = 0.5*pi_Left + 0.5*pi_Right
```

Define function for exercise

```
def transition(given_pi, states, P_Left, P_Right):
    P_out=pd.DataFrame(np.zeros((len(states),len(states))),index=S, columns=S)

    for s in states:
        action_dist=given_pi.loc[s]
        P=action_dist['TL']*P_Left+action_dist['TR']*P_Right

        P_out.loc[s]=P.loc[s]

    return P_out




def reward_fn(given_pi):
    R_s_a = pd.DataFrame(np.matrix([[1,1,0,0,0,0,0],[0,0,0,0,0,10,10]]).T,columns=["TL","TR"],index=S)

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

    return R_pi


reward_fn(pi_Right)
```

```
## S1     0
## S2     0
## S3     0
## S4     0
## S5     0
## S6    10
## S7    10
## dtype: int64
```

```
def policy_eval(given_pi,gamma):
    R = reward_fn(given_pi)
    P = transition(given_pi,S, P_TL , P_TR)
    epsilon = 10**(-8)
    v_old= np.repeat(0,7)
    v_new = R+np.dot(gamma*P, v_old)
    count = 0
    while np.linalg.norm(v_new-v_old)<epsilon:
        v_old=v_new
        v_new=R+np.dot(gamma*P,v_old) #

    return v_new
```

#Actual Exercise

```
R = reward_fn(pi_50)
R
```

```
## S1    0.5
## S2    0.5
## S3    0.0
## S4    0.0
## S5    0.0
## S6    5.0
## S7    5.0
## dtype: float64
```

```
P = transition(pi_50,S, P_TL , P_TR)
P
```

```
##      S1   S2   S3   S4   S5   S6   S7
## S1  0.5  0.5  0.0  0.0  0.0  0.0  0.0
## S2  0.5  0.0  0.5  0.0  0.0  0.0  0.0
## S3  0.0  0.5  0.0  0.5  0.0  0.0  0.0
## S4  0.0  0.0  0.5  0.0  0.5  0.0  0.0
## S5  0.0  0.0  0.0  0.5  0.0  0.5  0.0
## S6  0.0  0.0  0.0  0.0  0.5  0.0  0.5
## S7  0.0  0.0  0.0  0.0  0.0  0.5  0.5
```

```
policy_eval(pi_Right,1)
```

```
## S1     0.0
## S2     0.0
## S3     0.0
## S4     0.0
## S5     0.0
## S6    10.0
## S7    10.0
## dtype: float64
```

```
# policy Improve
gamma = 1
V_old = policy_eval(pi_50,gamma)
R_s_a = pd.DataFrame(np.matrix([[1,1,0,0,0,0,0],[0,0,0,0,0,10,10]]).T,columns=["TL","TR"],index=S)
q_s_a = R_s_a + np.c_[np.dot(gamma*P_TL,V_old),np.dot(gamma*P_TR,V_old)]
q_s_a
```

```
##      TL    TR
## S1  1.5   0.5
## S2  1.5   0.0
## S3  0.5   0.0
## S4  0.0   0.0
## S5  0.0   5.0
## S6  0.0  15.0
## S7  5.0  15.0
```

Policy_improve

```
# policy Improve
def policy_improve(v_old,pi_old,R_s_a,gamma,P_TL,P_TR):
    q_s_a = R_s_a + np.c_[np.dot(gamma*P_TL,v_old),np.dot(gamma*P_TR,v_old)]
    idxmax = q_s_a.idxmax(axis=1).tolist()
    count =0
    pi_new = pd.DataFrame(np.zeros(14).reshape(7,2),index = q_s_a.index,columns = q_s_a.columns)
    for i in q_s_a.index.tolist():
        pi_new.loc[i][idxmax[count]] = 1
        count +=1
    return pi_new
pi_old = pi_50
V_old = policy_eval(pi_50,1)
policy_improve(V_old,pi_old,R_s_a,1,P_TL,P_TR)
```

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