E1_MDP Python

Jaemin Park

2021-01-22

차 례

p.22 Iterative estimation of state-value function for a given policy π^{speed}

Python code

```
R = np.matrix([-1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]).reshape(8,1)
states = np.arange(0,80,10)
P = np.matrix([[0.1,0,0.9,0,0,0,0,0],[.1,0,0,0.9,0,0,0,0],[0,0.1,0,0,0.9,0,0,0],
[0,0,0.1,0,0,0.9,0,0],[0,0,0,0.1,0,0,0.9,0],
[0,0,0,0,0.1,0,0,0.9],[0,0,0,0,0,0.1,0,0.9],[0,0,0,0,0,0,0,1]])
print(R.T)
```

```
## [[-1.5 -1.5 -1.5 -1.5 -0.5 -1.5 -1.5  0. ]]
```

```
print(P)
```

```
## [[0.1 0.  0.9 0.  0.  0.  0.  0. ]
##  [0.1 0.  0.  0.9 0.  0.  0.  0. ]
##  [0.  0.1 0.  0.  0.9 0.  0.  0. ]
##  [0.  0.  0.1 0.  0.  0.9 0.  0. ]
##  [0.  0.  0.  0.1 0.  0.  0.9 0. ]
##  [0.  0.  0.  0.  0.1 0.  0.  0.9]
##  [0.  0.  0.  0.  0.  0.1 0.  0.9]
##  [0.  0.  0.  0.  0.  0.  0.  1. ]]
```

```
gamma=1
epsilon = 10**(-8)

v_old=np.array(np.zeros(8,)).reshape(8,1)
v_new=R + gamma*P*v_old

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

```
## [[-5.80592905 -5.2087811  -4.13926239 -3.47576467 -2.35376031 -1.73537603
##   -1.6735376   0.        ]]
```

p.23 Rewritten with intermediate saving

```
v_old=np.array(np.zeros(8,)).reshape(8,1)
v_new=R + gamma*P*v_old
results = []
while np.max(np.abs(v_new-v_old))>epsilon:
    results.append(v_new.T)
    v_old=v_new
    v_new=R + gamma*P*v_old

results = pd.DataFrame(np.matrix(np.array(results)), columns=states)
print(results.head())
```

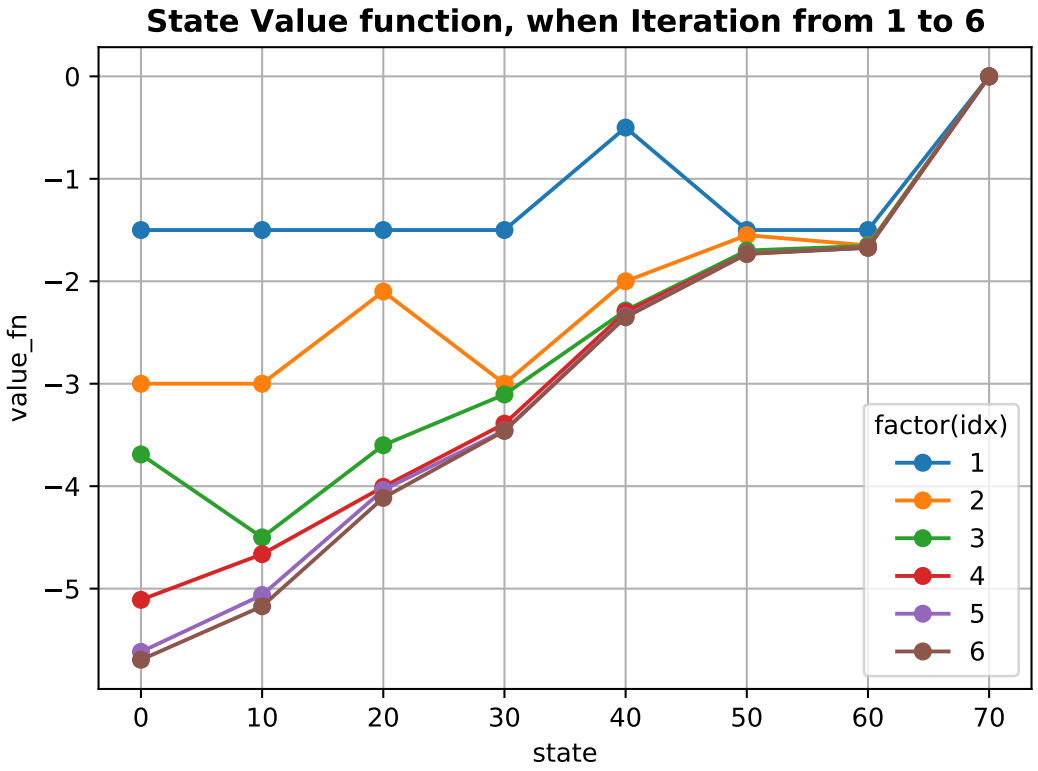
```
##         0       10       20      30     40      50       60   70
## 0 -1.50000 -1.5000 -1.50000 -1.5000 -0.500 -1.5000 -1.50000  0.0
## 1 -3.00000 -3.0000 -2.10000 -3.0000 -2.000 -1.5500 -1.65000  0.0
## 2 -3.69000 -4.5000 -3.60000 -3.1050 -2.285 -1.7000 -1.65500  0.0
## 3 -5.10900 -4.6635 -4.00650 -3.3900 -2.300 -1.7285 -1.67000  0.0
## 4 -5.61675 -5.0619 -4.03635 -3.4563 -2.342 -1.7300 -1.67285  0.0
```

```
print(results.tail())
```

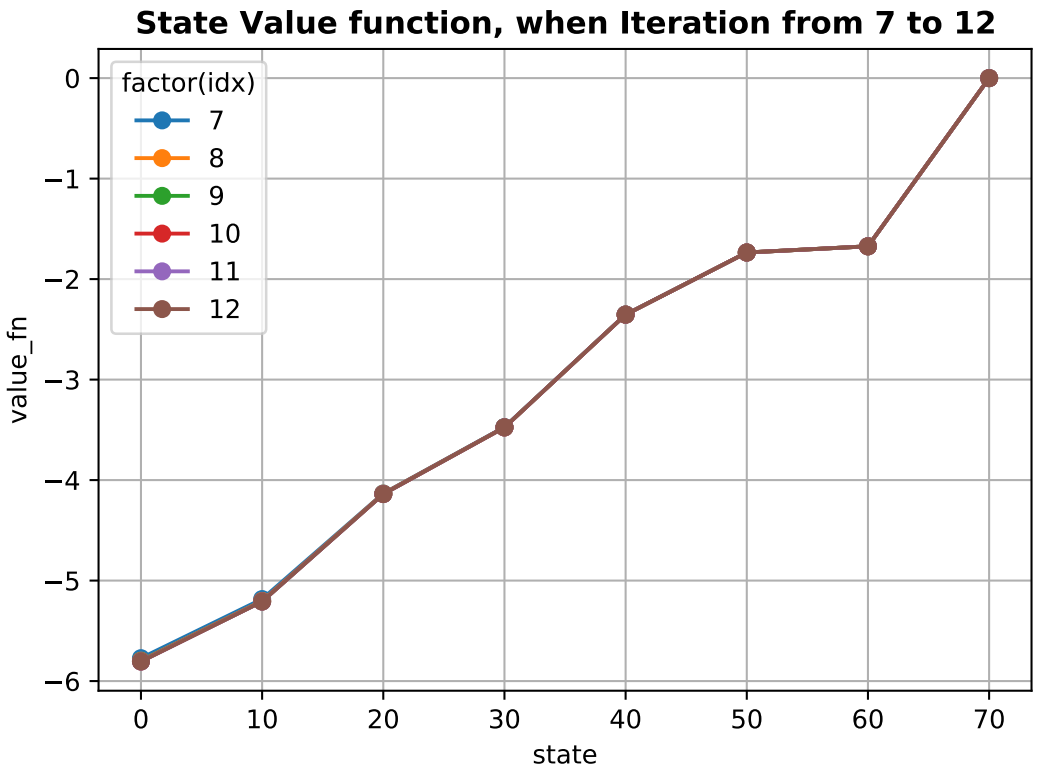
```
##           0         10        20        30       40        50        60   70
## 16 -5.805928 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 17 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 18 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 19 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
## 20 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
```

p.25 plot

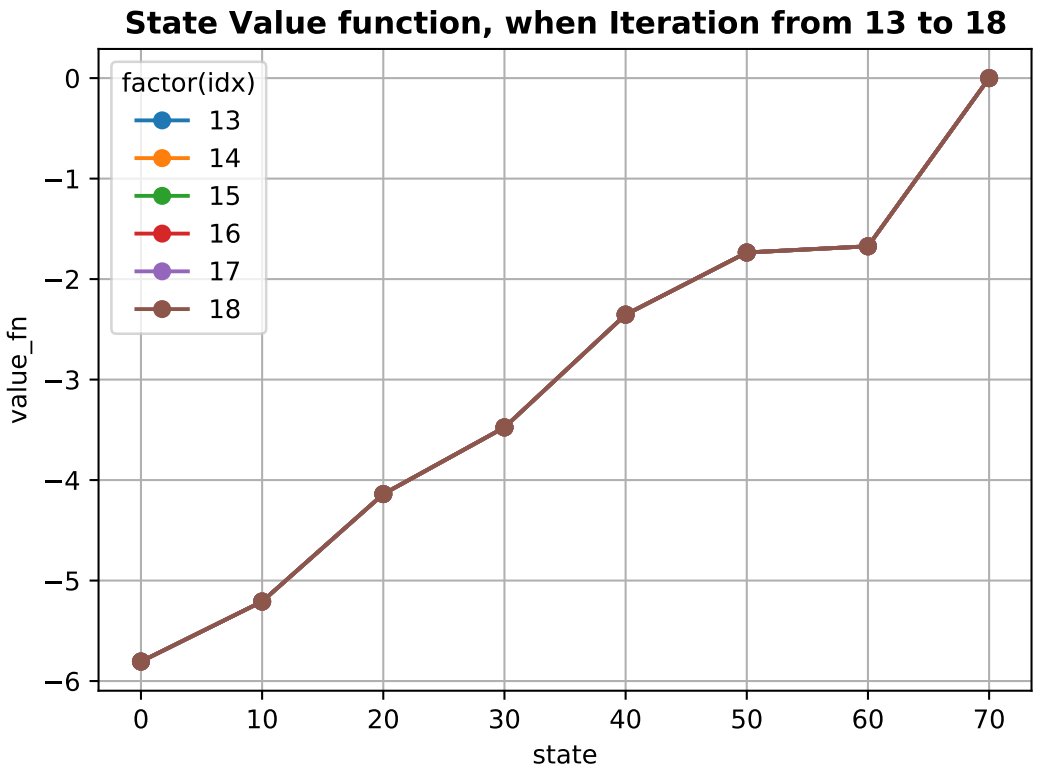
```
for i in range(0,6):
    plt.plot(states,results.iloc[i],marker='o',label=i+1)
plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 1 to 6',fontweight='bold')
plt.show()
```



```
for i in range(6,12):
    plt.plot(states,results.iloc[i],marker='o',label=i+1)
plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 7 to 12',fontweight='bold')
plt.show()
```



```
for i in range(12,18):
    plt.plot(states,results.iloc[i],marker='o',label=i+1)
plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('State Value function, when Iteration from 13 to 18',fontweight='bold')
plt.show()
```



p.34 Preparation for 1-3

1. π: S -> A

```
states = np.arange(0,80,10)
pi_speed = np.hstack((np.repeat(0,len(states)).reshape(8,1),np.repeat(1,len(states)).reshape(8,1)))
pi_speed = pd.DataFrame(pi_speed,states,["normal","speed"])
print(pi_speed)
```

```
##     normal  speed
## 0        0      1
## 10       0      1
## 20       0      1
## 30       0      1
## 40       0      1
## 50       0      1
## 60       0      1
## 70       0      1
```

2. R^{π} : S->$\mathbb{R}$

```
R_s_a = np.matrix([[-1,-1,-1,-1,0,-1,-1,0],[-1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]]).T
R_s_a = pd.DataFrame(R_s_a,states,["normal","speed"])
print(R_s_a)
```

```
##     normal  speed
## 0     -1.0   -1.5
## 10    -1.0   -1.5
## 20    -1.0   -1.5
## 30    -1.0   -1.5
## 40     0.0   -0.5
## 50    -1.0   -1.5
## 60    -1.0   -1.5
## 70     0.0    0.0
```

```
def reward_fn(given_pi):
    R_pi = np.matrix(given_pi*R_s_a).sum(axis=1)
    R_pi = pd.DataFrame(R_pi,states)
    return(R_pi)
print(reward_fn(pi_speed).T)
```

```
##     0    10   20   30   40   50   60   70
## 0 -1.5 -1.5 -1.5 -1.5 -0.5 -1.5 -1.5  0.0
```

3. P^{π} : A->S

```
P_normal = np.matrix([[0,1,0,0,0,0,0,0],[0,0,1,0,0,0,0,0],
[0,0,0,1,0,0,0,0],[0,0,0,0,1,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,1]])
P_speed = np.matrix([[0.1,0,0.9,0,0,0,0,0],[0.1,0,0,0.9,0,0,0,0],
[0,0.1,0,0,0.9,0,0,0],[0,0,0.1,0,0,0.9,0,0],[0,0,0,0.1,0,0,0.9,0],
[0,0,0,0,0.1,0,0,0.9],[0,0,0,0,0,0.1,0,0.9],[0,0,0,0,0,0,0,1]])

def transition(given_pi, states, P_normal, P_speed):
    P_out = pd.DataFrame(np.zeros((len(states),len(states))),states,states)
    for i,s in enumerate(states):
        action_dist = given_pi.loc[s]
        P = action_dist["normal"]*P_normal + action_dist["speed"]*P_speed
        P_out.loc[s] = P[i,:]

    return P_out
```

+Test 1

```
print(pi_speed)
```

```
##     normal  speed
## 0        0      1
```

```
## 10       0      1
## 20       0      1
## 30       0      1
## 40       0      1
## 50       0      1
## 60       0      1
## 70       0      1
```

```
print(transition(pi_speed,states,P_normal,P_speed))
```

```
##      0    10   20   30   40   50   60   70
## 0   0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 10  0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 20  0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 30  0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 40  0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 50  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 60  0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 70  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

+Test 1

```
print(pi_speed)
```

```
##     normal  speed
## 0        0      1
## 10       0      1
## 20       0      1
## 30       0      1
## 40       0      1
## 50       0      1
## 60       0      1
## 70       0      1
```

```
print(transition(pi_speed,states,P_normal,P_speed))
```

```
##       0    10   20   30   40   50   60   70
## 0   0.1  0.0  0.9  0.0  0.0  0.0  0.0  0.0
## 10  0.1  0.0  0.0  0.9  0.0  0.0  0.0  0.0
## 20  0.0  0.1  0.0  0.0  0.9  0.0  0.0  0.0
## 30  0.0  0.0  0.1  0.0  0.0  0.9  0.0  0.0
## 40  0.0  0.0  0.0  0.1  0.0  0.0  0.9  0.0
## 50  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.9
## 60  0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.9
## 70  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

+Test 2

```
pi_50 = np.hstack((np.repeat(0.5,len(states)).reshape(8,1),np.repeat(0.5,len(states)).reshape(8,1)))
pi_50 = pd.DataFrame(pi_50,states,["normal","speed"])
print(pi_50)
```

```
##     normal  speed
## 0      0.5    0.5
## 10     0.5    0.5
## 20     0.5    0.5
## 30     0.5    0.5
## 40     0.5    0.5
## 50     0.5    0.5
## 60     0.5    0.5
## 70     0.5    0.5
```

```
print(transition(pi_50,states,P_normal,P_speed))
```

```
##       0     10    20    30    40    50    60    70
## 0   0.05  0.50  0.45  0.00  0.00  0.00  0.00  0.00
## 10  0.05  0.00  0.50  0.45  0.00  0.00  0.00  0.00
## 20  0.00  0.05  0.00  0.50  0.45  0.00  0.00  0.00
## 30  0.00  0.00  0.05  0.00  0.50  0.45  0.00  0.00
## 40  0.00  0.00  0.00  0.05  0.00  0.50  0.45  0.00
## 50  0.00  0.00  0.00  0.00  0.05  0.00  0.50  0.45
## 60  0.00  0.00  0.00  0.00  0.00  0.05  0.00  0.95
## 70  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00
```

p.37 Summary

1. π: S -> A

```
pi_50
```

```
##     normal  speed
## 0      0.5    0.5
## 10     0.5    0.5
## 20     0.5    0.5
## 30     0.5    0.5
## 40     0.5    0.5
## 50     0.5    0.5
## 60     0.5    0.5
## 70     0.5    0.5
```

2. R^{π} : S->$\mathbb{R}$

```
reward_fn(pi_50)
```

```
##        0
## 0  -1.25
## 10 -1.25
## 20 -1.25
## 30 -1.25
## 40 -0.25
## 50 -1.25
## 60 -1.25
## 70  0.00
```

3. P^{π} : A->S

```
print(transition(pi_50,states,P_normal,P_speed))
```

```
##       0     10    20    30    40    50    60    70
## 0   0.05  0.50  0.45  0.00  0.00  0.00  0.00  0.00
## 10  0.05  0.00  0.50  0.45  0.00  0.00  0.00  0.00
## 20  0.00  0.05  0.00  0.50  0.45  0.00  0.00  0.00
## 30  0.00  0.00  0.05  0.00  0.50  0.45  0.00  0.00
## 40  0.00  0.00  0.00  0.05  0.00  0.50  0.45  0.00
## 50  0.00  0.00  0.00  0.00  0.05  0.00  0.50  0.45
## 60  0.00  0.00  0.00  0.00  0.00  0.05  0.00  0.95
## 70  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00
```

p.39 Implementation, finally

```
def policy_eval(given_pi):
    R=reward_fn(given_pi)
    P=transition(given_pi, states=states, P_normal=P_normal, P_speed=P_speed)

    gamma=1.0
    epsilon=10**(-8)

    v_old=np.repeat(0,8).reshape(8,1)
    v_new=R+np.dot(gamma*P,v_old)

    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.T
print(policy_eval(pi_speed))
```

```
##          0         10        20        30       40        50        60   70
## 0 -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0.0
```

```
print(policy_eval(pi_50))
```

```
##          0         10        20        30       40        50        60   70
## 0 -5.969238 -5.133592 -4.119955 -3.389228 -2.04147 -2.027768 -1.351388  0.0
```