Mars Rover Markorv Process Example

Reinforcement Learning Study

2021-01-12

차 례

Mars Rover Markov process

Trainsition Matrix :

$$P = \begin{pmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

Trainsition Diagram :

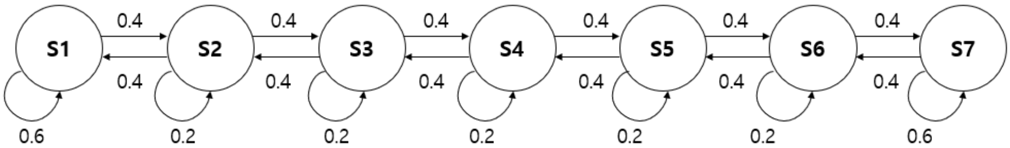

그림 1: Mars Rover MarkovProcess

Observaton

Classification of States

- A state i is said to be recurrent if, starting from i, the probability of getting back to i is 1
- A state i is said to be trainsient if, starting from i, the probability of getting back to i is less than 1
- A state i is said to be abosrbing state, as a special case of reccurent state, if $P_i i = 1$ (You can naver leave the state i if you get there)

recurrent state : {1,2,3,4,5,6,7}

trainsient state : {}

abosrbing state : {}

Stationary distribution

```
import numpy as np
```

```
P=np.array([[0.6,0.4,0,0,0,0,0],
            [0.4,0.2,0.4,0,0,0,0],
            [0,0.4,0.2,0.4,0,0,0],
            [0,0,0.4,0.2,0.4,0,0],
            [0,0,0,0.4,0.2,0.4,0],
            [0,0,0,0,0.4,0.2,0.4],
            [0,0,0,0,0,0.4,0.6]])

print("Shape:",P.shape)
```

```
## Shape: (7, 7)
```

```
print(P)
```

```
## [[0.6 0.4 0.  0.  0.  0.  0. ]
##  [0.4 0.2 0.4 0.  0.  0.  0. ]
##  [0.  0.4 0.2 0.4 0.  0.  0. ]
##  [0.  0.  0.4 0.2 0.4 0.  0. ]
##  [0.  0.  0.  0.4 0.2 0.4 0. ]
##  [0.  0.  0.  0.  0.4 0.2 0.4]
##  [0.  0.  0.  0.  0.  0.4 0.6]]
```

```
egien_value, egien_vector = np.linalg.eig(P.T) ## np.linalg,eig(p) returns egien_value, egienvector
```

```
print("egien_value :\n",egien_value)
```

```
## egien_value :
##  [-0.52077509 -0.29879184  0.02198325  0.37801675  0.69879184  1.
##   0.92077509]
```

```
print("egien_vector :\n",egien_vector)
```

```
## egien_vector :
##  [[ 1.18942442e-01  2.31920614e-01 -3.33269318e-01  4.17906506e-01
##    4.81588117e-01  3.77964473e-01 -5.21120889e-01]
##  [-3.33269318e-01 -5.21120889e-01  4.81588117e-01 -2.31920614e-01
##    1.18942442e-01  3.77964473e-01 -4.17906506e-01]
##  [ 4.81588117e-01  4.17906506e-01  1.18942442e-01 -5.21120889e-01
##   -3.33269318e-01  3.77964473e-01 -2.31920614e-01]
##  [-5.34522484e-01  2.16498678e-16 -5.34522484e-01 -4.08753786e-16
##   -5.34522484e-01  3.77964473e-01  5.50931749e-16]
```

```
## [ 4.81588117e-01 -4.17906506e-01  1.18942442e-01  5.21120889e-01
##  -3.33269318e-01  3.77964473e-01  2.31920614e-01]
## [-3.33269318e-01  5.21120889e-01  4.81588117e-01  2.31920614e-01
##   1.18942442e-01  3.77964473e-01  4.17906506e-01]
## [ 1.18942442e-01 -2.31920614e-01 -3.33269318e-01 -4.17906506e-01
##   4.81588117e-01  3.77964473e-01  5.21120889e-01]]
```

```
x_1=egien_vector[:,5] # egein_vector corresspond with egien_value 1

v=x_1/np.sum(x_1)

print(v)
```

```
## [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
##  0.14285714]
```

```
np.dot(v,P)
```

```
## array([0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
##        0.14285714, 0.14285714])
```

Limiting Probability

```
from numpy.linalg import matrix_power
np.set_printoptions(formatter={'float_kind': lambda x: "{0:0.5f}".format(x)})

print(P)
```

```
## [[0.60000 0.40000 0.00000 0.00000 0.00000 0.00000 0.00000]
##  [0.40000 0.20000 0.40000 0.00000 0.00000 0.00000 0.00000]
##  [0.00000 0.40000 0.20000 0.40000 0.00000 0.00000 0.00000]
##  [0.00000 0.00000 0.40000 0.20000 0.40000 0.00000 0.00000]
##  [0.00000 0.00000 0.00000 0.40000 0.20000 0.40000 0.00000]
##  [0.00000 0.00000 0.00000 0.00000 0.40000 0.20000 0.40000]
##  [0.00000 0.00000 0.00000 0.00000 0.00000 0.40000 0.60000]]
```

```
print(matrix_power(P,2))
```

```
## [[0.52000 0.32000 0.16000 0.00000 0.00000 0.00000 0.00000]
##  [0.32000 0.36000 0.16000 0.16000 0.00000 0.00000 0.00000]
##  [0.16000 0.16000 0.36000 0.16000 0.16000 0.00000 0.00000]
##  [0.00000 0.16000 0.16000 0.36000 0.16000 0.16000 0.00000]
```

```
## [0.00000 0.00000 0.16000 0.16000 0.36000 0.16000 0.16000]
## [0.00000 0.00000 0.00000 0.16000 0.16000 0.36000 0.32000]
## [0.00000 0.00000 0.00000 0.00000 0.16000 0.32000 0.52000]]
```

```
print(matrix_power(P,3))
```

```
## [[0.44000 0.33600 0.16000 0.06400 0.00000 0.00000 0.00000]
## [0.33600 0.26400 0.24000 0.09600 0.06400 0.00000 0.00000]
## [0.16000 0.24000 0.20000 0.24000 0.09600 0.06400 0.00000]
## [0.06400 0.09600 0.24000 0.20000 0.24000 0.09600 0.06400]
## [0.00000 0.06400 0.09600 0.24000 0.20000 0.24000 0.16000]
## [0.00000 0.00000 0.06400 0.09600 0.24000 0.26400 0.33600]
## [0.00000 0.00000 0.00000 0.06400 0.16000 0.33600 0.44000]]
```

```
print(matrix_power(P,20))
```

```
## [[0.19515 0.18469 0.16593 0.14266 0.11954 0.10111 0.09092]
## [0.18469 0.17638 0.16143 0.14281 0.12423 0.10935 0.10111]
## [0.16593 0.16143 0.15326 0.14299 0.13262 0.12423 0.11954]
## [0.14266 0.14281 0.14299 0.14308 0.14299 0.14281 0.14266]
## [0.11954 0.12423 0.13262 0.14299 0.15326 0.16143 0.16593]
## [0.10111 0.10935 0.12423 0.14281 0.16143 0.17638 0.18469]
## [0.09092 0.10111 0.11954 0.14266 0.16593 0.18469 0.19515]]
```

```
print(matrix_power(P,200))
```

```
## [[0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]
## [0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]
## [0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]
## [0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]
## [0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]
## [0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]
## [0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286]]
```

Observation Result

in Mars Rover Markorv Process Problem

- MC is ireeducible and Aperiodic
- Stationary distribution is unique
- Liming probabilites is equal to stationary distribution

Markov Reward Process in Mars Rover example

The rewards obtained by executing an action from any of the states $S2, S3, S4, S5, S6$ is 0, while any moves from states $S1, S7$ yield rewards 1, 10 respectively. The rewards are stationary and deterministic

Calculate State-value function. (assume that Time Horizon is 10 days, and start at State S4)

Computing the value function of Markov reward process (Monte Carlo simulation)

this code based on leture notes D1, But There are some modifications.

```
def go_forward(this_state):
    split_list=list(this_state)
    next_state=split_list[0]+str(int(split_list[1])+1)

    return next_state

print(go_forward('s4')) # return next state ex) s4 - > s5, s5 ->s6
```

```
## s5
```

```
def go_backward(this_state):
    split_list=list(this_state)
    next_state=split_list[0]+str(int(split_list[1])-1)

    return next_state

print(go_backward('s4')) # retrun previous state ex) s4->s3, s5->s4
```

```
## s3
```

mars_simul based on leture notes D1 soda_simul

But There are some modifications with user defined function above and Transition matrix

```
def mars_simul(this_state):
    u=np.random.uniform()
    next_state=''
    if this_state =='s1':
        if u<=0.6:
            next_state=this_state
        else:
            next_state= go_forward(this_state)



    if this_state in ['s2','s3','s4','s5','s6']:

        if u<=0.4:
            next_state=go_forward(this_state)
```

```python
        elif 0.4<=u<=0.6:
            next_state = this_state

        else :
            next_state = go_backward(this_state)


    if this_state == 's7':

        if u<=0.6:
            next_state=this_state

        else :
            next_state =go_backward(this_state)


    return next_state
```

There are only reward in state $s1$, $s7$ yield respectivly, 1, 10

```python
def cost_eval(path):
    cost_one_path=path.count('s1')*1+path.count('s7')*10
    return cost_one_path
```

Combine the functions defined so far to create a Monte Car simulation function.

```python
def MC_V_t(initial_state, num_episode, time_horizon):

    episode_i = 0

    cum_sum_G_i = 0

    while(episode_i<num_episode) :
        path=initial_state
        for n in range(time_horizon-1):
            this_state=path[-2:]
            next_state=mars_simul(this_state)
            path+=next_state
```

```
        G_i=cost_eval(path)
        cum_sum_G_i+=G_i
        episode_i+=1
    V_t=cum_sum_G_i/num_episode
    return V_t
```

Finally Calculate S4's state value function.

```
print(MC_V_t('s1',10000,10))
```

```
## 5.0751
```

```
print(MC_V_t('s2',10000,10))
```

```
## 3.9362
```

```
print(MC_V_t('s3',10000,10))
```

```
## 4.7167
```

```
print(MC_V_t('s4',10000,10))
```

```
## 8.0551
```

```
print(MC_V_t('s5',10000,10))
```

```
## 15.0685
```

```
print(MC_V_t('s6',10000,10))
```

```
## 27.5043
```

```
print(MC_V_t('s7',10000,10))
```

```
## 45.3709
```

Computing the value function of Markov reward process (Iterative Solution)

```
P=np.array([[0.6,0.4,0,0,0,0,0],
            [0.4,0.2,0.4,0,0,0,0],
            [0,0.4,0.2,0.4,0,0,0],
            [0,0,0.4,0.2,0.4,0,0],
            [0,0,0,0.4,0.2,0.4,0],
            [0,0,0,0,0.4,0.2,0.4],
            [0,0,0,0,0,0.4,0.6]])

R=np.array([1,0,0,0,0,0,10])[:,None] #[:,None] yield column vector

H=10

v_t1=np.array([0,0,0,0,0,0,0])[:,None] #[:,None] yield column vector
```

```
print('P :\n',P)
```

```
## P :
##  [[0.60000 0.40000 0.00000 0.00000 0.00000 0.00000 0.00000]
##  [0.40000 0.20000 0.40000 0.00000 0.00000 0.00000 0.00000]
##  [0.00000 0.40000 0.20000 0.40000 0.00000 0.00000 0.00000]
##  [0.00000 0.00000 0.40000 0.20000 0.40000 0.00000 0.00000]
##  [0.00000 0.00000 0.00000 0.40000 0.20000 0.40000 0.00000]
##  [0.00000 0.00000 0.00000 0.00000 0.40000 0.20000 0.40000]
##  [0.00000 0.00000 0.00000 0.00000 0.00000 0.40000 0.60000]]
```

```
print('R :\n',R)
```

```
## R :
##  [[ 1]
##  [ 0]
##  [ 0]
##  [ 0]
##  [ 0]
##  [ 0]
##  [10]]
```

```
print('v_t1 :\n',v_t1)
```

```
## v_t1 :
```

```
## [[0]
## [0]
## [0]
## [0]
## [0]
## [0]
## [0]]
```

```
t=H-1

while(t>=0):
    v_t = R+np.dot(P,v_t1)
    t = t-1
    v_t1 = v_t

print(v_t)
```

```
## [[5.06486]
##  [3.99416]
##  [4.74780]
##  [8.11882]
##  [15.21931]
##  [27.31909]
##  [45.53596]]
```

as a result, now we get state value function for State S1 to S7

it is simillar to Monte Carlo simulation results

Strictly speaking, the iterative solution results are correct.

```
"Done, Mars Rover Markorv Process Example"
```

```
## [1] "Done, Mars Rover Markorv Process Example"
```