Mars Rover Markorv Process Example

Reinforcement Learning Study

2021-02-03

차 례

lnotes2는 CS234_notes의 2번째 lnotes2 Mars-Roveer example 문제를 의미합니다.

example에 나와있는 마코프 체인을 보고

1. 마코프 체인을 관찰해서 이제 까지 배웠던 (stationary distribution, limiting prob, state classification) 등을 관찰해보고
2. s4에서 시작 H=10일 이라고 하고 가정, MC / iterative solution으로 value를 계산하는 숙제였습니다.

lecture note solution과는 같이 모으기 애매해서, 따로 디렉토리 lnotes2를 만들었습니다.

솔루션 종합을 할때, 답이 갈리는 부분이 없는 경우는 제일 lecture note와 가깝게 작성한 솔루션을 코드구현과 같이 갈리는 경우 종합 하여 가장 퀄리티가 높은 솔루션을 가져왔습니다.

Mars Rover Markov process Obeservation (김봉석)

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 :

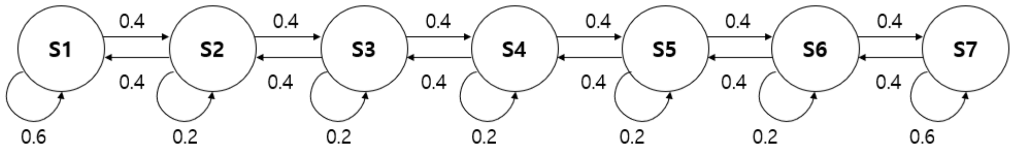

그림 1: Mars Rover MarkovProcess

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 $P_{ii} = 1$ (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) (김봉석, 권태현, 이성호, 정원렬)

reward를 MC로 구현하는 경우 다음과 같이 크게 두가지 방식으로 방볍이 나뉘었습니다 그 중 가장 완성도가 높은 학생 것을 가져왔습니다.

1. state를 문자열로 구현, (김봉석, 권태현) state path를 문자로처리해 이어 붙이기 위해 go_forward, go_backward라는 function을 따로 추가해 구현하였습니다. 이 방식은 lecture note와 유사하게 구현하고, path를 문자열로 볼 수 있다는 장점이 있지만 상대적으로 코드가 좀더 길어 집니다.

```
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)

        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

```
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.

```
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 state value function.

```
result=[]
num_iter=100000
time_horizon=10

for state in ['s1','s2','s3','s4','s5','s6','s7']:
    result.append(MC_V_t(state,num_iter,time_horizon))
print("MC reuslt:",result)
```

```
## MC reuslt: [5.09021, 4.00084, 4.75011, 8.0864, 15.16107, 27.4519, 45.53545]
```

2. state를 숫자로 구현, (이성호 정원렬)

이 방법의 경우 mar_simul를 구현할때 숫자로 구현하였습니다. path를 lecture-note와 같이 찍어볼수는 없지만 이 경우 문자열 처리를 할 필요가 없어져 상대적으로 코드가 짧아지게 됩니다. 방법1과 동일한 결과를 나타냅니다.

```
import numpy as np
def mar_simul(this_state):
  u=np.random.rand()
  if (this_state == 1):
    if(u<=0.6):
      next_state = this_state
    else:
      next_state = this_state+1

  elif (this_state == 7):
    if(u<=0.4):
      next_state = this_state-1
    else:
      next_state = this_state
  else:
    if(u<=0.4):
      next_state = this_state-1
    elif(u<=0.6 and u>0.4):
      next_state = this_state
    else:
      next_state = this_state+1

  return next_state

def reward_eval(path):
  reward_one_path = path.count(1)*1 + path.count(7)*10
  return reward_one_path



MCN = 10000
spending_record = ['0']*MCN
for i in range(MCN):
  path = [4]
  for t in range(9):
    this_state = path[-1]
    next_state = mar_simul(this_state)
    path.append(next_state)

  spending_record[i] = reward_eval(path)

spending_record = sum(spending_record)/len(spending_record)

print("Average of Mar simulation reward using MC is", spending_record)
```

```
## Average of Mar simulation reward using MC is 8.385
```

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

Iterative Solution의 경우에는 따로 구현할 부분이 없어, 대다수 일치하였습니다.

```
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"
```