Lecture D1.Markov Reward Process 1

Baek, Jong min

2021-01-13

차 례

Recap(Page 10) 2
MC simulation for estimating state-value function(Page 11) 3
For general **t**,(exercise)(Page 17) 4
Backward induction for estimating *state-value function*(Page 20) 5
Page 21 5

Recap(Page 10)

```
def soda_simul(this_state):
  u = np.random.uniform(size=1)
  if this_state == 'c':
    if u <= 0.7:
      next_state = 'c'
    else:
      next_state = 'p'
  else:
    if u <= 0.5:
      next_state = 'c'
    else:
      next_state = 'p'
  return next_state
```

```
def cost_eval(path):
  cost_one_path = path.count('c')*1.5 + path.count('p')*1
  return cost_one_path
```

```
MC_N = 10000
spending_records = np.zeros(MC_N)

for i in range(1,MC_N):
  path = 'c'
  for t in range(1,10):
    this_state = path[-1]
    next_state = soda_simul(this_state)
    path += next_state
  spending_records[i] = cost_eval(path)
```

MC simulation for estimating state-value function(Page 11)

```
def state_value_function(num_episode):
  episode_i = 0
  cum_sum_G_i  = 0
  # number of episode(iteration)
  while episode_i < num_episode:
    path = 'c' # initial state
    # generate stochastic path(episode)
    for t in range(1,10):
      this_state = path[-1]
      next_state = soda_simul(this_state)
      path += next_state
    # print(path)
    # calculate sum of rewards
    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
```

```
state_value_function(10000)
```

```
## 13.3656
```

For general t,(exercise)(Page 17)

if, time is 0 to 9(finite)

$$
\begin{aligned}
V_t(S) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[\sum_{i=t}^{9} r_i \mid S_t = s] \\
&= \mathbb{E}[r_t \mid S_t = s] + \mathbb{E}[r_{t+1} + \cdots + r_9 \mid S_t = s] \\
&= R(s) + \mathbb{E}[G_{t+1} \mid S_t = s] \\
&= R(s) + \sum_{s' \in S} P_{ss'} \mathbb{E}[G_{t+1} \mid S_t = s, S_{t+1} = s'] \\
&= R(s) + \sum_{s' \in S} P_{ss'} \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \quad \text{(Markov property)} \\
&= R(s) + \sum_{s' \in S} P_{ss'} V_{t+1}(s')
\end{aligned}
$$

Backward induction for estimating *state-value function*(Page 20)

```
import numpy as np
P = np.array([0.7,0.3,0.5,0.5]).reshape(2,2)
R = np.array([1.5,1.0]).reshape(2,1)
v_t1 = np.array([0,0]).reshape(2,1)
H = 10
t = H-1
while t >= 0 :
  v_t = R + np.dot(P,v_t1)
  t = t-1
  v_t1 = v_t
print(v_t)
```

```
## [[13.35937498]
##  [12.73437504]]
```

Page 21

```
import numpy as np
P = np.array([0.7,0.3,0.5,0.5]).reshape(2,2)
R = np.array([1.5,1.0]).reshape(2,1)

def state_value_function(P,R,H):
    t = H-1
    globals()['V_{}'.format(H)] = np.array([0,0]).reshape(2,1)
    while t >= 0:
        globals()['V_{}'.format(t)] = R+np.dot(P,globals()['V_{}'.format(t+1)])
        t = t-1
    return globals()['V_{}'.format(t+1)]

state_value_function(P,R,10)
```

```
## array([[13.35937498],
##        [12.73437504]])
```

D1.Rmd

```r
"Hello"
```

```
## [1] "Hello"
```