D1_Solution

Reinforcement Learning Study

2021-01-26

차례

Recap (P. 10) 손민상	2
MC simulation for estimating state-value function (P. 11) 백종민	3
For general t, Exercise (P. 17) 손민상	4
P. 20 권도윤	5
Page 21 백종민	5

Recap (P. 10) 손민상

```
Recap
import numpy as np
                                              • The MC simulation is a valid
                                                                                       spending_records <- rep(0, MC_N)</pre>
def soda_simul(this_state):
                                                approach. We shall review our initial
                                                                                      for (i in 1:MC_N) {
                                                effort with newly introduced
                                                                                       path <- "c" # coke today (day-0)
     n=np.random.random()
                                                                                        for (t in 1:9) {
                                                terminology.
                                                                                        this_state <- str_sub(path, -1, -1)
                                              • The algorithm includes…
                                                                                         next_state <- soda_simul(this_state)</pre>

    Generate a single stochastic path

                                                                                         path <- paste0(path, next_state)</pre>
     if this_state=='c':
                                                     starting from the initial state,
                                                                                        spending_records[i] <- cost_eval(path)</pre>
                                                     S_0 = c.
          if n<=0.7:</pre>
                                                 Collect a single value of return,
               next_state='c'
                                                     G_i, 1 \leq i \leq MC_N, by accumulating
                                                                                      cost_eval <- function(path) {</pre>
                                                    rewards, \{r_0, r_1, ..., r_9\}, along the
                                                                                        cost_one_path <-
          else:
                                                                                          str_count(path, pattern = "c")*1.5 +
                                                 Take an average of collected returns
                                                                                         str_count(path, pattern = "p")*1
               next_state='p'
                                                     to evaluate state-value function,
                                                                                        return(cost_one_path)
                                                     V_0(c).
     else:
          if n<=0.5:</pre>
               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(MC_N):
     path='c' # coke today (day-0)
     for t in range(9):
          this_state=path[-1]
          next_state=soda_simul(this_state)
          path+=next_state
     spending_records[i]=cost_eval(path)
```

```
print(spending_records)
```

```
## [13. 14. 13. ... 13.5 14.5 13.5]
```

Recap (P. 10) R code 박재민

```
MC N <- 10000
spending records <- rep(0, MC N)</pre>
for (i in 1:MC_N) {
  path <- "c" # coke today (day-0)</pre>
  for (t in 1:9) {
    this_state <- str_sub(path, -1, -1)</pre>
    next_state <- soda_simul(this_state)</pre>
    path <- paste0(path, next_state)</pre>
  }
  spending_records[i] <- cost_eval(path)</pre>
}
cost_eval <- function(path) {</pre>
  cost_one_path <-
    str_count(path, pattern = "c")*1.5 +
    str_count(path, pattern = "p")*1
  return(cost_one_path)
}
```

MC simulation for estimating state-value function (P. 11) 백종민

```
MC simulation for estimating state-value function
def state_value_function(num_episode):
  episode_i = 0
                                                        • Formally, for a finite-horizon MRP, the following is MC simulation for estimating
                                                          state-value function.
  cum_sum_G_i = 0
  # number of episode(iteration)
                                                      # MC evaluation for state-value function
                                                      # with state s, time 0, reward r, time-horizon H
  while episode_i < num_episode:</pre>
                                                      1: episode_i <- 0
                                                      2: cum_sum_G_i <- 0
                                                      3: while episode_i < num_episode
     path = 'c' # initial state
                                                      4: Generate an stochastic path starting from state s and time \theta.
                                                      5: Calculate return G_i <- sum of rewards from time 0 to time H-1.
     # generate stochastic path(episode)
                                                      6: cum_sum_G_i <- cum_sum_G_i + G_i
                                                      7: episode_i <- episode_i + 1
    for t in range(1,10):
                                                      8: State-value-fn V_t(s) <- cum_sum_G_i/num_episode
                                                      9: return V_t(s)
       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)
```

13.33665

MC simulation for estimating state-value function (P. 11) R code 박재민

```
episode_i <- 0
cum_sum_G_i <- 0
while episode_i < num_episode
  #Generate an stochastic path starting from state s and time 0.
  #Calculate return G_i <- sum of rewards from time 0 to time H-1.
  cum_sum_G_i <- cum_sum_G_i + G_i
  episode_i <- episode_i + 1
State-value-fn V_t(s) -- cum_sum_G_i/num_episode
return V_t(s)</pre>
```

For general t, Exercise (P. 17) 손민상

For general t,

$$\begin{split} V_t(s) &= & \mathbb{E}[G_t|S_t = t] \\ &= & \mathbb{E}[r_t + r_{t+1} + r_{t+2} + \dots + r_{\infty}|S_t = s] \\ &= & \mathbb{E}[r_t|S_t] + \mathbb{E}[r_{t+1} + r_{t+2} + \dots + r_{\infty}|S_t = s] \\ &= & R(s) + \mathbb{E}[r_{t+1} + r_{t+2} + \dots + r_{\infty}|S_t = s] \\ &= & R(s) + \mathbb{E}[G_{t+1}|S_t = s, S_{t+1} = s'] \\ &= & R(s) + \mathbb{E}[G_{t+1}|S_{t+1} = s'](\because Markov\ property) \\ &= & R(s) + \sum_{s \in s'} P_{ss'} V_{t+1}(s') \end{split}$$

P. 20 권도윤

```
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)
H = 10
                                                 P <- array(c(0.7,0.5,0.3,0.5), dim=c(2,2))
                                                                                             • Thus, we have the following
v_{t1} = np.array([0,0]).reshape(2,1)
                                                 R <- array(c(1.5,1.0), dim=c(2,1))
                                                                                               state-value function.
                                                 H <- 10 # time-horizon
                                                                                                 • V_0(c) = 13.359375
t = H-1 # time-horizon
                                                 v_t1 <- array(c(0,0), dim=c(2,1)) # v_{t+1}
                                                                                                 • V_0(p) = 12.734375
                                                 t <- H-1
                                                 while (t >= 0) {
while (t>=0):
                                                  v_t <- R + P %*% v_t1
                                                  t <- t-1
    v_t = R + np.dot(P, v_t1)
                                                  v t1 <- v t
                                                 }
     t = t-1
                                                 v_t
     v_t1 = v_t
                                                           [,1]
                                                 ## [1,] 13.35937
print(v_t)
                                                 ## [2,] 12.73438
## [[13.35937498]
```

[12.73437504]]

Page 21 백종민

```
#Backward induction for state-value function
#with transition prob mat P , reward vector R, time-horizon H, state-value vector v
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
                                                      • Formally, for a finite-horizon MRP, the following is backward induction for
    return globals()['V_{}'.format(t+1)]
                                                        estimating state-value function.
state_value_function(P,R,10)
                                                    # Backward induction for state-value function
                                                    # with transition prob mat P, reward vector R, time-horizon H, state-value vector v_{\_}\{\}
                                                    1: v_H <- zero-column vector
                                                    2: t <- H-1
## array([[13.35937498],
                                                    3: while t >= 0
                                                    4: v_t <- R + P*v_{t+1}
5: t <- t-1
##
            [12.73437504]])
                                                    9: return v_t # this is v_\theta(s) for all s, because t=0 at this point
"D1_Solution"
```