Lecture E2. MDP with Model 2

Sim, Min Kyu, Ph.D., mksim@seoultech.ac.kr

SEOULTECH 서울과학기술대학교 데이터사이언스학과

I. Recap

policy_eval()

```
gamma <- 1
states <- as.character(seq(0, 70, 10))
P_normal <- matrix(c(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),
  nrow = 8, ncol = 8, byrow = TRUE,
  dimnames = list(states, states))
P_speed <- matrix(c(.1, 0,.9, 0, 0, 0, 0, 0,
                   .1, 0, 0,.9, 0, 0, 0, 0,
                    0,.1, 0, 0,.9, 0, 0, 0,
                    0, 0,.1, 0, 0,.9, 0, 0,
                    0, 0, 0,.1, 0, 0,.9, 0,
                    0, 0, 0, 0,.1, 0, 0,.9,
                    0, 0, 0, 0, 0,.1, 0,.9,
                    0, 0, 0, 0, 0, 0, 0, 1),
  nrow = 8, ncol = 8, byrow = TRUE,
  dimnames = list(states, states))
```

```
transition <- function(
  given_pi, states, P_normal, P_speed) {
  P_out <- array(0,
    dim = c(length(states), length(states)),
    dimnames = list(states, states))
  for (s in states) {
    action_dist <- given_pi[s,]
    P <- action_dist["normal"]*P_normal +
      action_dist["speed"]*P_speed
    P_out[s,] <- P[s,]
  }
  return(P_out)
}
R_s_a <- matrix(                          R(s,a)
  c(  -1,   -1,   -1,   -1, 0.0,   -1,   -1, 0,
    -1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5, 0),
  nrow = length(states), ncol = 2, byrow = FALSE,
  dimnames = list(states, c("normal", "speed")))
reward_fn <- function(given_pi, R_s_a) {
  R_pi <- rowSums(given_pi*R_s_a)
  return(R_pi)                            R(s).
}
```

```r
policy_eval <- function(given_pi) {
  R <- reward_fn(given_pi, R_s_a = R_s_a)
  P <- transition(given_pi, states = states, P_normal = P_normal, P_speed = P_speed)
  gamma <- 1.0
  epsilon <- 10^(-8)
  v_old <- array(rep(0,8), dim=c(8,1))
  v_new <- R + gamma*P%*%v_old
  while (max(abs(v_new-v_old)) > epsilon) {
    v_old <- v_new
    v_new <- R + gamma*P%*%v_old
  }
  return(v_new)
}
pi_speed <- cbind(rep(0,length(states)), rep(1,length(states)))
rownames(pi_speed) <- states; colnames(pi_speed) <- c("normal", "speed")
t(policy_eval(pi_speed))
```

```
##              0        10        20        30       40        50        60  70
## [1,] -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0
```

$V^{\pi^{speed}}(s)$

```r
pi_50 <- cbind(rep(0.5,length(states)), rep(0.5,length(states)))
rownames(pi_50) <- states; colnames(pi_50) <- c("normal", "speed")
t(policy_eval(pi_50))
```

```
##              0        10        20        30       40        50        60  70
## [1,] -5.969238 -5.133592 -4.119955 -3.389228 -2.04147 -2.027768 -1.351388  0
```

$V^{\pi^{50}}(s)$

Major components of approaching MDP

If you can't measure,
you can't improve!

1. **(policy evaluation)** We need to be able to evaluate $V^\pi(s)$ for a fixed π. This is called *policy evaluation*. This is also called as *prediction* in reinforcement learning.
2. **(optimal value function)** We want to be able to evaluate $V^{\pi^*}(s)$, where π^* is the *optimal policy*. The quantity, $V^{\pi^*}(s)$, is *optimal policy's value function*, or called shortly as *optimal value function*.
3. **(optimal policy)** We want to find the *optimal policy* π^*. This is also called as *control* in reinforcement learning

- Check your reasoning why the followings are possible.
  - Optimal policy first: (optimal policy) + (policy evaluation) $\rightarrow$ (optimal value function)
    3 + 1 → 2
  - Optimal value function first: (optimal value function) $\rightarrow$ (optimal policy)
    2 → 3

- This note will develop the following approach.
  - (policy evaluation) + *series of* (*policy improvement*) $\rightarrow$ (optimal policy)

II. Policy improvement

Development

immediate reward

discounted sum of future rewards.

Recap

- Remind that we have a Bellman's equation for MDP as follows.

$$V^{\pi}(s) = R^{\pi}(s) + \gamma \sum_{\forall s'} \mathbf{P}^{\pi}_{ss'} V^{\pi}(s') \quad \texttt{(E1, p18)}$$

- It means that, given a π, its value is determined by immediate reward plus discounted sum of future rewards.

Motivation

· normal : $R+\gamma PV = -1 + 1.0 \times 1.0 \times (-3.47) = -4.47$

· speed : $R+\gamma PV = -1.5 + 1.0 \left(0.9 \times -2.35 + 0.1 \times (-5.21) \right) = -4.14$

normal : $-1 + 1 \cdot 1.0 = -1$

speed : $-1.5 + \left(0.9 \times 0 + 0.1 \times -1.74 \right) = -1.67$

- We shall first criticize the π_{speed}, whose policy evaluation is given as below.

```
t(policy_eval(pi_speed))
```

$V^{\pi speed}$

$V^{\pi speed}$ 가 $\pi(20)$ = speed 를 지지함

```
##                0         10         20         30        40         50         60 70
## [1,] -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538  0
```

state 60 에서는, $V^{\pi speed}$ 가 π_{speed} 에서 state 60 가는 act 을 지지 (X)

- From the state 60, current policy gives the estimate for the state-value function of -1.6735376, meaning that sticking to π_{speed} at state 60 has a value of -1.67.
- However, we know that switching to *normal mode* at state 60 is better, because switching to normal action at state 60 guarantees the arrival to the state 70 with additional energy spending of 1.0.

Development

- Under the current policy's (π_{speed}) value function, on the state `60`,
  - Choosing *normal mode* gives

$$R + \gamma \mathbf{P} V = -1.0 + 1.0 \cdot 0 = -1.0$$

  - Choosing *speed mode* gives

90% "no" 10% "50"

$$R + \gamma \mathbf{P} V = -1.5 + (0.9 \cdot 0 + 0.1 \cdot -1.74) = -1.674$$

  - Thus, π_{speed} should modify its action on the state `60`. We've just improved the current policy π_{speed} for the state `60`!

- This should be checked for all states as well as the state `60`.
- This completes one step of **policy improvement**.
- Formally, policy improvement implies the following task of replacement:

$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s') \right], \quad \text{for all } s$$

time t: act by a
time t+1: act by π(s)

$$\pi^{new}(s) \leftarrow argmax_{a\in\mathcal{A}} \left[R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s') \right], \text{ for all } s$$

Meaning

- The term in the RHS, $R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s')$, implies **[the expected return of starting from state s, choosing an action a for this time step only, then following the policy π afterwards.]**.
- How is this quantity different from $V^\pi(s)$?

time t: act by π(s)
time t+1 ~: by " π(s)

Defining $q(s, a)$ - action-value function

- The RHS makes an improvement using current policy π, by varying only the action in this time step.
- Formally, $q(s,a)$ is called **action-value function**, also famously known as *Q-function*.

$$\begin{aligned} q^\pi(s,a) &:= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s') \end{aligned}$$

$$\pi^{new}(s) \leftarrow argmax_{a\in\mathcal{A}} \left[R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s') \right], \text{ for all } s$$

$\pi^{new}(s) \leftarrow \underset{a\in A}{\text{argmax}}\ \boxed{q^{\pi}(s,a)}$

Policy improvement

- Using this new notation of $q(s,a)$, the policy improvement can be written as

$$\pi^{new}(s) \leftarrow argmax_{a\in\mathcal{A}}\ q^{\pi^{old}}(s,a), \text{ for all } s$$

- The improvement is called *greedy improvement* since it involves a myopic digression from the current policy, in a way that an action only on this time step is revised.
- It can be proved that *greedy improvement* is guaranteed to improve.

Implementation

normal speed

sp no

$$\pi^{new}(s) \leftarrow argmax_{a\in\mathcal{A}}\left[R(s,a)+\gamma\sum_{\forall s'}\mathbf{P}^{a}_{ss'}V^{\pi^{old}}(s')\right],\ \text{for all } s$$

R + γPV

```
V_old <- policy_eval(pi_speed)
pi_old <- pi_speed
q_s_a <- R_s_a +
  cbind(gamma*P_normal%*%V_old,
        gamma*P_speed%*%V_old)
q_s_a
```

q(s,a=nor) q(s,a=speed)

```
##       normal     speed
## 0  -6.208781 -5.805929
## 10 -5.139262 -5.208781
## 20 -4.475765 -4.139262
## 30 -3.353760 -3.475765
## 40 -1.735376 -2.353760
## 50 -2.673538 -1.735376
## 60 -1.000000 -1.673538
## 70  0.000000  0.000000
```

column wise maximum

```
pi_new_vec <- apply(q_s_a, 1, which.max)
pi_new <- array(0, dim = dim(pi_old),
                dimnames = dimnames(pi_old))
for (i in 1:length(pi_new_vec)) {
  pi_new[i, pi_new_vec[i]] <- 1
}
pi_new
```

π new

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

- policy_improve()

```
policy_improve <- function(
  V_old,
  pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
  P_normal = P_normal, P_speed = P_speed) {

  q_s_a <- R_s_a + cbind(gamma*P_normal%*%V_old,
                          gamma*P_speed%*%V_old)
  pi_new_vec <- apply(q_s_a, 1, which.max)
  pi_new <- array(0, dim = dim(pi_old),
                  dimnames = dimnames(pi_old))
  for (i in 1:length(pi_new_vec)) {
    pi_new[i, pi_new_vec[i]] <- 1
  }
  return(pi_new)
}
```

- One step improvement from π^{speed}

(as in p12)

```
pi_old <- pi_speed
V_old <- policy_eval(pi_old)
pi_new <- policy_improve(V_old,
  pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
  P_normal = P_normal, P_speed = P_speed)
```

```
pi_old
```

(π^{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
```

```
pi_new
```

(π in p12)

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

Summary

- Policy improvement uses Bellman's equation of

$$\pi^{new}(s) \leftarrow argmax_{a\in\mathcal{A}} \Bigg[\overbrace{R(s,a) + \gamma \sum_{\forall s'} \mathbf{P}^{a}_{ss'} V^{\pi^{old}}(s')}^{q(s.a)} \Bigg], \text{ for all } s$$

- For each state, policy improvement assesses whether the current policy leads to an optimal action that the current state-value function vouches.
- Policy improvement occurs in a greedy fashion. This greedy approach is guaranteed to improve.

III. Policy iteration

Development

- Given a policy π, `policy_eval()` evaluates its state-value function. Using the estimate of state-value function, `policy_improve()` improves the policy to the better one.
- If this process is iterated, then it is guaranteed to reach optimal policy. In other words, policy iteration is the process to reach the optimal policy described as follows.

$$\pi_0 \xrightarrow{\texttt{policy\_eval()}} V_0 \xrightarrow{\texttt{policy\_improve()}} \pi_1 \xrightarrow{\texttt{policy\_eval()}} V_1 \xrightarrow{\texttt{policy\_improve()}}$$

$$\pi_2 \rightarrow \cdots \rightarrow \cdots\cdots \rightarrow \pi^* \xrightarrow{\texttt{policy\_eval()}} V^* (= V^{\pi^*},\ \text{for short.})$$

- The iteration process terminates when π_i does not change any more, i.e. $\pi_i = \pi_{i+1}$.
- Note that policy evaluation is an approximate algorithm. For policy iteration purpose, policy evaluation cannot be, (and doesn't have to be as well), perfect.

Try do it over and over until no change - from π^{speed}

- Step 0

```
pi_old <- pi_speed
pi_old
```

```
##    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
```

- Step 1

```
V_old <- policy_eval(pi_old)
pi_new <- policy_improve(V_old,
  pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
  P_normal = P_normal, P_speed = P_speed)
pi_old <- pi_new
pi_old
```

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

- Step 2

```
V_old <- policy_eval(pi_old)
pi_new <- policy_improve(V_old,
  pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
  P_normal = P_normal, P_speed = P_speed)
pi_old <- pi_new
pi_old
```

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

- Step 3

```
V_old <- policy_eval(pi_old)
pi_new <- policy_improve(V_old,
  pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
  P_normal = P_normal, P_speed = P_speed)
pi_old <- pi_new
pi_old
```

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

Policy iteration process (from π^{speed})

- Now we are ready to implement whole process as a single code block with a loop.

```
pi_old <- pi_speed
cnt <- 0
repeat{ # do-while in R
  print(paste0(cnt, "-th iteration"))
  print(t(pi_old))
  V_old <- policy_eval(pi_old)
  pi_new <- policy_improve(V_old,
    pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
    P_normal = P_normal, P_speed = P_speed)
  if (all.equal(pi_new, pi_old)==TRUE) break
  pi_old <- pi_new
  cnt <- cnt + 1
}
print(policy_eval(pi_new))
```

```
## [1] "0-th iteration"
##        0 10 20 30 40 50 60 70
## normal 0  0  0  0  0  0  0  0
## speed  1  1  1  1  1  1  1  1
## [1] "1-th iteration"
##        0 10 20 30 40 50 60 70
## normal 0  1  0  1  1  0  1  1
## speed  1  0  1  0  0  1  0  0
## [1] "2-th iteration"
##        0 10 20 30 40 50 60 70
## normal 0  0  0  1  1  0  1  1
## speed  1  1  1  0  0  1  0  0
##           [,1]
## 0  -5.107744
## 10 -4.410774
## 20 -3.441077
## 30 -2.666667
## 40 -1.666667
## 50 -1.666667
## 60 -1.000000
## 70  0.000000
```

opt. policy

opt. value

Policy iteration process (from π^{50})

- The process should work for other initial choice of π, despite of possibly different convergence rate.

```
pi_old <- pi_50
cnt <- 0
repeat{ # do-while in R
  print(paste0(cnt, "-th iteration"))
  print(t(pi_old))
  V_old <- policy_eval(pi_old)
  pi_new <- policy_improve(V_old,
    pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
    P_normal = P_normal, P_speed = P_speed)
  if (all.equal(pi_new, pi_old)==TRUE) break
  pi_old <- pi_new
  cnt <- cnt + 1
}
print(policy_eval(pi_new))
```

```
## [1] "0-th iteration"
##          0  10  20  30  40  50  60  70
## normal 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
## speed  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
## [1] "1-th iteration"
##        0 10 20 30 40 50 60 70
## normal 0  1  0  1  1  0  1  1
## speed  1  0  1  0  0  1  0  0
## [1] "2-th iteration"
##        0 10 20 30 40 50 60 70
## normal 0  0  0  1  1  0  1  1
## speed  1  1  1  0  0  1  0  0
##           [,1]
## 0  -5.107744
## 10 -4.410774
## 20 -3.441077
## 30 -2.666667
## 40 -1.666667
## 50 -1.666667
## 60 -1.000000
## 70  0.000000
```

Summary

- From a policy π, $q^{\pi}(s, a)$ implies an expected return of starting from the state s, choosing an action a for this time step only, then following the policy π afterwards.
- Policy improvement occurs by

$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \; q^{\pi^{old}}(s, a)$$

, or, equivalently,

$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{\forall s'} P^{a}_{ss'} V^{\pi^{old}}(s') \right], \;\; \text{for all } s$$

- Policy iteration is the iterative process beginning from an arbitrary policy, policy evaluation and policy improvement take places until the policy converges. It is guaranteed to be converges to the optimal policy.

Exercise 1

Write python code to generate the same output in p.20.

```
"Success isn't permarnent, and failure isn't fatal. - Mike Ditka"
```