Lecture E3. MDP with Model 3

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

SEOULTECH 서울과학기술대학교 데이터사이언스학과

I. Value improvement

Recap

- Major components of approaching MDP
  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

- This note aims to discuss **2. (optimal value function)** by iterative process.

Motivation

- Policy improvement occurs by the following task of replacement. (E2, p9)

q(s.a)

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

- Here lies the philosophy of dynamic programming:

> *If you find something better than the current scheme, it will be improved. Furthermore, it is guaranteed to reach the optimum by iterative improvement.*

- Value improvement (improvement in estimates for the optimal value function) occurs by the following task of replacement.

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

Strategy

- Review `E2, p10` to develop strategy.
- Value improvement's algorithm is very similar to policy improvement.

Preparation

```r
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))
```

```r
R_s_a <- matrix(
  c( -1,  -1,  -1,  -1, 0.0,  -1,  -1, 0, (normal)
   -1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5, 0),(speed)
  nrow = length(states), ncol = 2, byrow = FALSE,
  dimnames = list(states, c("normal", "speed")))
```

Implementation

```
# 1. Initialize V
V_old <- array(rep(0,length(states)),
               dim=c(length(states),1))
rownames(V_old) <- states
t(V_old)
```

```
##       0 10 20 30 40 50 60 70
## [1,] 0  0  0  0  0  0  0  0
```

```
# 2. Evaluate the Q-function
q_s_a <- R_s_a +
  cbind(gamma*P_normal%*%V_old,
        gamma*P_speed%*%V_old)
q_s_a
```

```
##    normal speed
## 0      -1  -1.5
## 10     -1  -1.5
## 20     -1  -1.5
## 30     -1  -1.5
## 40      0  -0.5
## 50     -1  -1.5
## 60     -1  -1.5
## 70      0   0.0
```

element wise maximum.

```
# 3. Find the best action for each state
library(magrittr)
V_new <- apply(q_s_a, 1, max) %>%
  as.matrix(dim = c(length(states,1)))
t(V_new)
```

```
##       0 10 20 30 40 50 60 70
## [1,] -1 -1 -1 -1  0 -1 -1  0
```

- This completes the one-step value improvement.

II. Value iteration

Discussion

- Value iteration is to iterate value improvement until it converges.
- It is also greedy approach since looking at one-step improvement.
- Implementation is straight-forward given the prior implementation of value improvement.

Implementation

q(s,a)를 구하고 columnwise maximum을 취함.

$V_0 \to V_1 \to V_2 \to V_3$

```
# Already assigned are gamma, states, P_normal, P_speed, R_s_a
cnt <- 0
epsilon <- 10^(-8)
V_old <- array(rep(0,length(states)), dim=c(length(states),1))
rownames(V_old) <- states
results <- t(V_old) # to save
repeat{
  q_s_a <- R_s_a + cbind(gamma*P_normal%*%V_old, gamma*P_speed%*%V_old)
  V_new <- apply(q_s_a, 1, max) %>% as.matrix(dim = c(length(states,1)))
  if (max(abs(V_new-V_old)) < epsilon) break
  results <- rbind(results, t(V_new)) # to save
  V_old <- V_new
  cnt <- cnt + 1
}
```

```
value_iter_process <- results
results <- data.frame(results)
colnames(results) <- states
head(results)
```

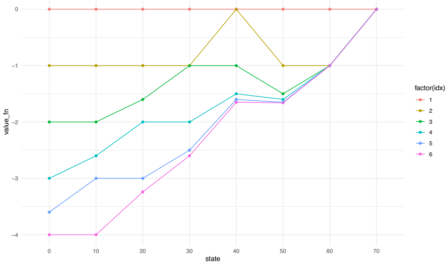
```
##      0   10    20   30    40    50 60 70
## 1  0.0  0.0  0.00  0.0  0.00  0.00  0  0
## 2 -1.0 -1.0 -1.00 -1.0  0.00 -1.00 -1  0
## 3 -2.0 -2.0 -1.60 -1.0 -1.00 -1.50 -1  0
## 4 -3.0 -2.6 -2.00 -2.0 -1.50 -1.60 -1  0
## 5 -3.6 -3.0 -3.00 -2.5 -1.60 -1.65 -1  0
## 6 -4.0 -4.0 -3.24 -2.6 -1.65 -1.66 -1  0
```

```
tail(results)
```

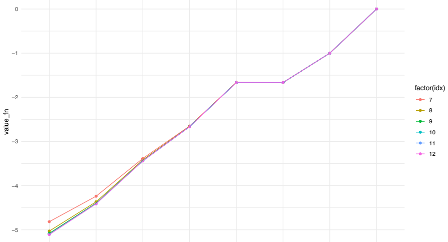
```
##            0         10         20         30         40         50 60 70
## 17 -5.107742 -4.410773 -3.441077 -2.666666 -1.666667 -1.666667 -1  0
## 18 -5.107743 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1  0
## 19 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1  0
## 20 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1  0
## 21 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1  0
## 22 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1  0
```

Visualization

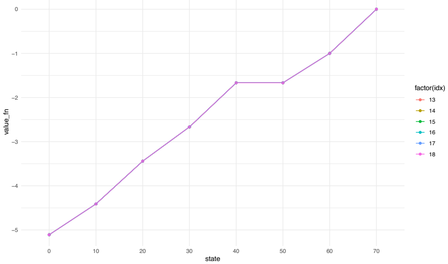
- Iteration from 1 to 6.



- Iteration from 7 to 12.



- Iteration from 13 to 18.



- The previous plot was generated by the following code.

```
library(tidyverse)
results$idx <- as.numeric(row.names(results))
results <- results %>%
  gather(as.character(states), key="state", value="value_fn")
# the first figure
results %>% filter(idx <= 6) %>%
ggplot(aes(x=state, y=value_fn, group = factor(idx), color = factor(idx))) +
  geom_point() + geom_line() +
  theme_minimal()
```

Animated visualization

Animation Link

- The previous animation was generated by the following code.
- Check my `L21` note in Data Visualization if interested.

```r
library(gganimate)
library(gifski)
results$idx <- as.factor(results$idx)
fig_static <-
  ggplot(results, aes(x=state, y=value_fn, color=state)) +
  geom_point(size = 5) +
  theme_minimal()
fig_dynamic <- fig_static +
  transition_states(idx) +
  labs(title = 'Now showing Iteration Number {closest_state}') +
  enter_fade() + exit_shrink()
anim_save(
  filename = "anim_value_iter.gif",
  animation = fig_dynamic,
  renderer = gifski_renderer())
```

Discussion

- Why is the process of **policy evaluation** so similar to **value iteration**?
  - Policy evaluation is an iterative process to estimate **the value function given a fixed policy**.
  - Value iteration is an iterative process to estimate **the value function of the optimal policy**.
  - Thus, the process is similar, both being based on the fixed point theorem.

- What is the difference between **policy evaluation** and **value iteration**?
  - Policy evaluation:

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

  - Value iteration:

q(s.a)

$$V^{*}_{new}(s) \leftarrow max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{\forall s'} \mathbf{P}^{a}_{ss'} V^{*}_{old}(s') \right], \ \forall s \ \texttt{(this note, p5)}$$

  - Value iteration (or, value improvement as its core component) includes **taking maximum of Q-function among all available action**.

Optimal value function → Optimal policy

```
V_opt <- t(tail(value_iter_process,1))
t(V_opt)
```

```
##             0        10        20        30        40        50 60 70
## [22,] -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1  0
```

- Its corresponding optimal policy?

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

-1 -4.4

```
##       normal     speed
## 0  -5.410774 -5.107744
## 10 -4.441077 -4.410774
## 20 -3.666667 -3.441077
## 30 -2.666667 -3.344108
## 40 -1.666667 -1.666667
## 50 -2.000000 -1.666667
## 60 -1.000000 -1.666667
## 70  0.000000  0.000000
```

$= -1.5 + \begin{pmatrix} 0.9 \times -3.44 \\ 0.1 \times -5.10 \end{pmatrix}$

```
pi_opt_vec <- apply(q_s_a, 1, which.max)
pi_opt_vec
```

```
##  0 10 20 30 40 50 60 70
##  2  2  2  1  1  2  1  1
```

- You may stop here, or transform pi_opt_vec into a matrix form as below.

```
pi_opt <- array(0,
  dim = c(length(states),2),
  dimnames = list(states, c("normal","speed")))
for (i in 1:length(pi_opt_vec)) {
  pi_opt[i, pi_opt_vec[i]] <- 1
}
t(pi_opt)
```

```
##        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
```

Exercise 1

Write python code to generate the same output in p.12.

Exercise 2

Write python code to generate the same output in p.18.

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