# E3

Reinforcement Learning Study

2021-01-25

## 차 례



### Value improvement

#### Preparation

```
import numpy as np
import pandas as pd

gamma = 1
states = np.arange(0,80,10).astype(str)

P_normal=pd.DataFrame(np.matrix([[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]]), index=states,columns=states)

P_speed=pd.DataFrame(np.matrix([[.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]]), index=states, columns=states)
```

```
R_s_a=pd.DataFrame(np.matrix([-1,-1,-1,-1,0.0,-1,-1,0,-1.5,-1.5,-1.5,-1.5,-0.5,-1.5,-1.5,0]).reshape(len(stat
```

### Implementation

```
# 1initialize V
V_old=pd.DataFrame(np.repeat(0,len(states)).reshape(len(states),1),index=states)
V_old.T
```

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

```
# 2. Evaluate the Q-function
q_s_a = R_s_a+np.c_[gamma*np.dot(P_normal,V_old), gamma*np.dot(P_speed,V_old)]

q_s_a
```

```
##     normal  speed
## 0     -1.0   -1.5
## 10    -1.0   -1.5
## 20    -1.0   -1.5
## 30    -1.0   -1.5
## 40     0.0   -0.5
## 50    -1.0   -1.5
## 60    -1.0   -1.5
## 70     0.0    0.0
```

```
# 3. Find the best action for each state
V_new=np.matrix(q_s_a.apply(max,axis=1)).reshape(len(states),1)
V_new.T
```

```
## matrix([[-1., -1., -1., -1.,  0., -1., -1.,  0.]])
```

## Value Iteration

### Implementation

```
cnt=0
epsilon=10**(-8)
V_old=pd.DataFrame(np.repeat(0,len(states)).reshape(len(states),1),index=states)
results=V_old.T
while True:
    q_s_a=R_s_a+np.c_[np.dot(gamma*P_normal,V_old),np.dot(gamma*P_speed,V_old)]
    V_new=np.matrix(q_s_a.apply(max,axis=1)).reshape(len(states),1)

    if np.max(np.abs(V_new-V_old)).item() < epsilon :
        break

    results=np.r_[results, V_new.T]
    V_old=V_new

    cnt+=1
```

```
value_iter_process = results

results = pd.DataFrame(results, columns=states)

results.head()
```

```
##      0   10   20   30   40    50   60   70
## 0  0.0  0.0  0.0  0.0  0.0  0.00  0.0  0.0
## 1 -1.0 -1.0 -1.0 -1.0  0.0 -1.00 -1.0  0.0
## 2 -2.0 -2.0 -1.6 -1.0 -1.0 -1.50 -1.0  0.0
## 3 -3.0 -2.6 -2.0 -2.0 -1.5 -1.60 -1.0  0.0
## 4 -3.6 -3.0 -3.0 -2.5 -1.6 -1.65 -1.0  0.0
```

```
results.tail()
```

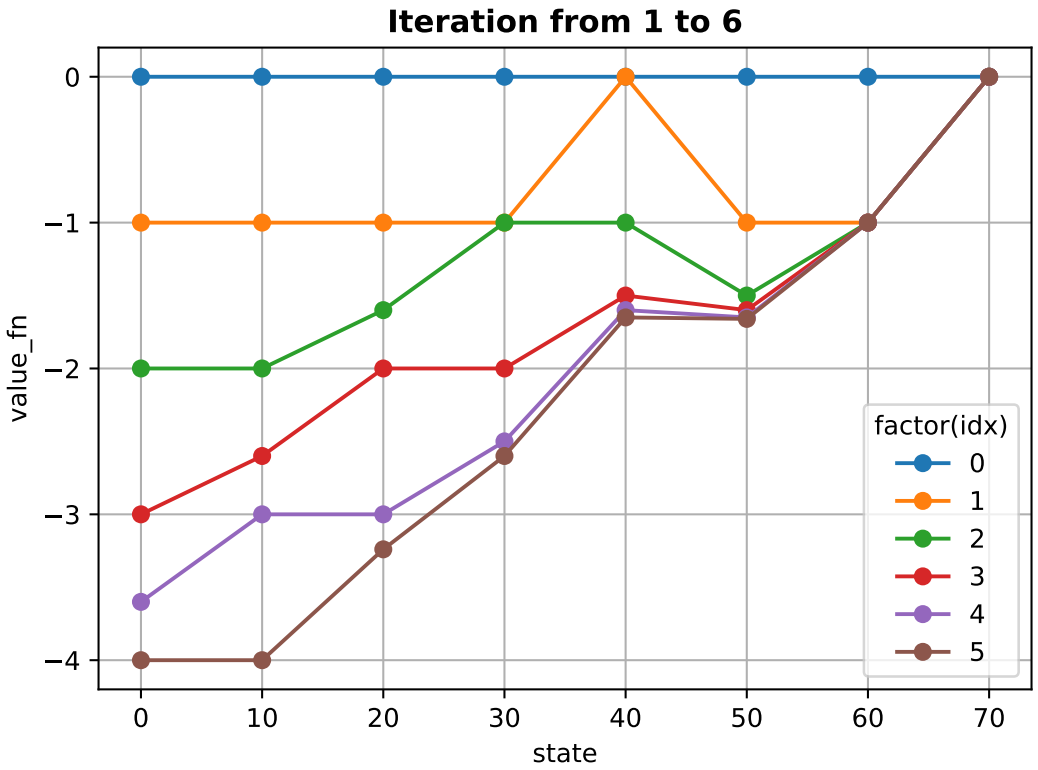
```
##            0        10        20        30        40        50   60   70
## 17 -5.107743 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 18 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 19 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 20 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
## 21 -5.107744 -4.410774 -3.441077 -2.666667 -1.666667 -1.666667 -1.0  0.0
```

### Visualization

```
import matplotlib.pyplot as plt



for i in range(6):
    plt.plot(results.columns,results.iloc[i], label=i,marker='o')

plt.grid(True)
plt.legend(title='factor(idx)')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 1 to 6', fontweight='bold')
plt.yticks([0,-1,-2,-3,-4])
```

```
## ([<matplotlib.axis.YTick object at 0x000000004CEB6A88>, <matplotlib.axis.YTick object at 0x000000004CEB60C8>,
```

```
plt.show()
```

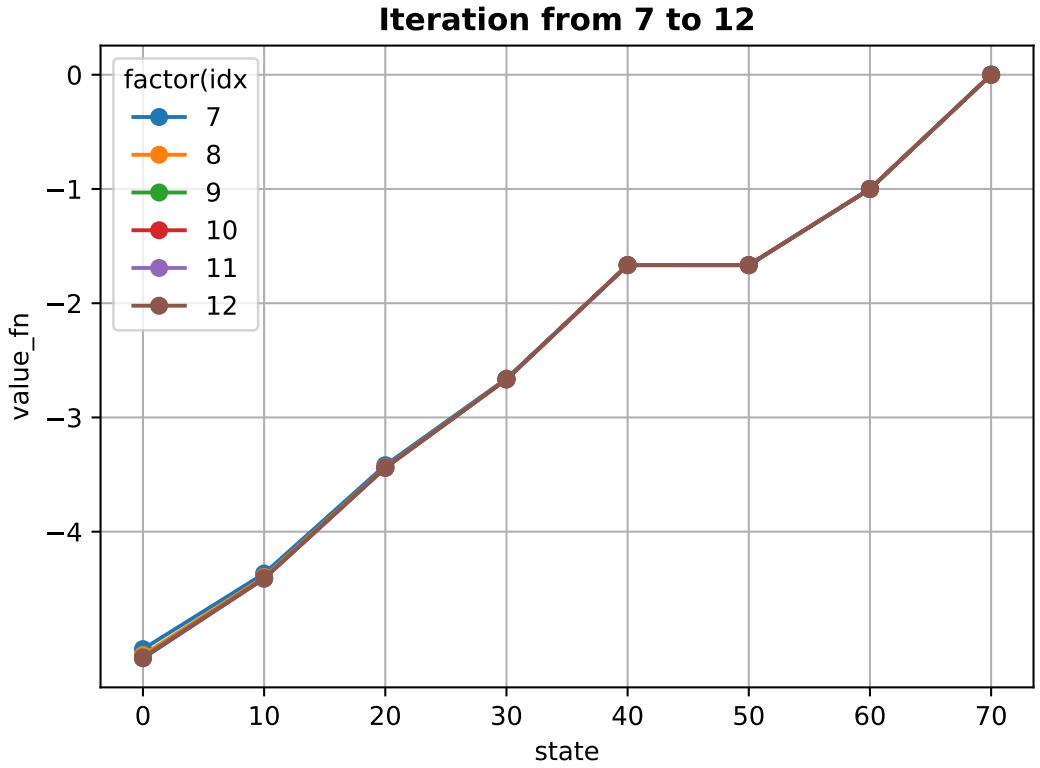



```
for i in range(7,13):
    plt.plot(results.columns,results.iloc[i], label=i,marker='o')
```

```
plt.grid(True)
plt.legend(title='factor(idx')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 7 to 12', fontweight='bold')
plt.yticks([0,-1,-2,-3,-4])
```

```
## ([<matplotlib.axis.YTick object at 0x000000004E10F4C8>, <matplotlib.axis.YTick object at 0x000000004E10B7C8>,
```

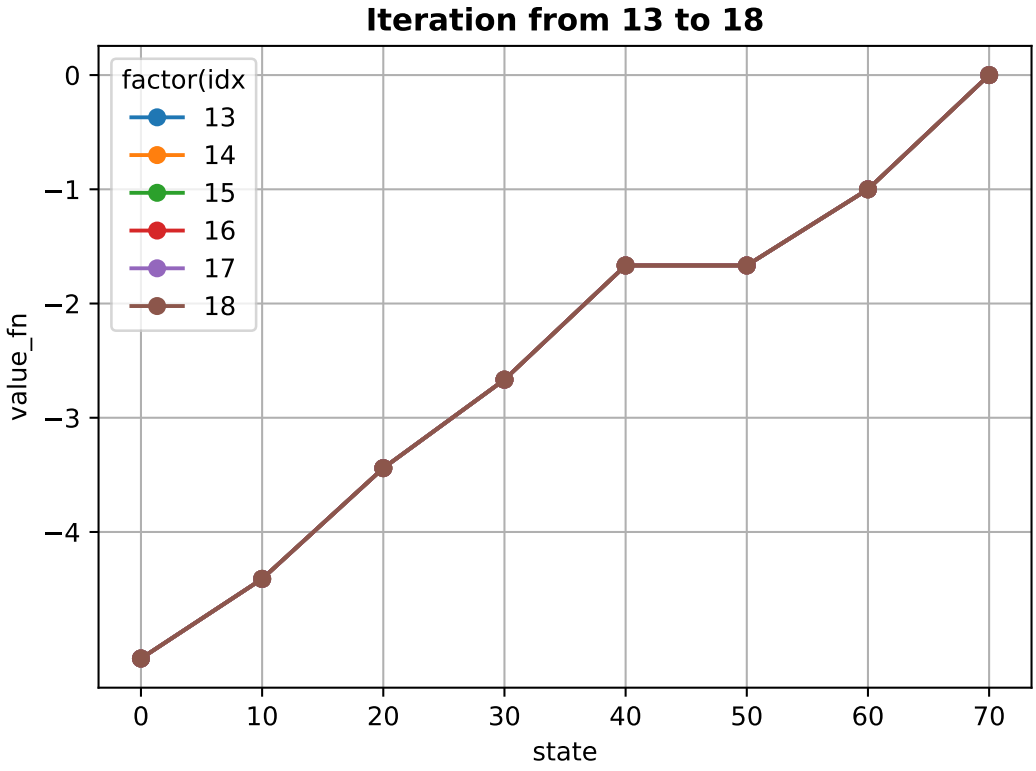
```
plt.show()
```




```
for i in range(13,19):
    plt.plot(results.columns,results.iloc[i], label=i,marker='o')

plt.grid(True)
plt.legend(title='factor(idx')
plt.xlabel('state')
plt.ylabel('value_fn')
plt.title('Iteration from 13 to 18', fontweight='bold')
plt.yticks([0,-1,-2,-3,-4])
```

```
## ([<matplotlib.axis.YTick object at 0x000000002C7B4888>, <matplotlib.axis.YTick object at 0x000000004C7EB708>,
```

```
plt.show()
```




**Optimal value function to Optimal Policy**

원핫 인코딩을 이용하면 for문 안돌아도 될것 같음

```
V_opt=value_iter_process[-1]
q_s_a=R_s_a+np.c_[np.dot(gamma*P_normal,V_opt.T),np.dot(gamma*P_speed,V_opt.T)]


q_s_a
```

```
##       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
```

```
opt_action = q_s_a.idxmax(axis=1)

opt_action
```

```
## 0      speed
## 10     speed
## 20     speed
## 30    normal
## 40    normal
## 50     speed
## 60    normal
## 70    normal
## dtype: object
```

```
#py_install("scikit-learn")
#sk<-import("scikit-learn")
#use_virtualenv("r-reticulate")
```

**sklearn 모델이 설치오류가 계속나서 latex 작성불가..**

```
#from sklearn.preprocessing import OneHotEncoder # Skitlearn OnHotEncoder 이용


#integer_encoded = opt_action.values.reshape(len(opt_action), 1)
#print(opt_action.values.reshape(len(opt_action), 1))

#onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
#print(onehot_encoded)


#pi_opt=pd.DataFrame(onehot_encoded, index=states, columns=['normal','speed'])
#pi_opt.T
```

```
"Done "
```

```
## [1] "Done "
```