D_case Aggregate

Bongseokkim

2021-02-16

차 례

종합

D_case aggregate 인원은 백종민, 권도윤을 제외하고 일단 먼저 종합했습니다 . 제가 먼저 읽어보고 소제목을 옆에 임의로 붙였고 한글로 간략한 설명을 첨부하였습니다.

- pd.sereis를 사용한 학생중 제 컴퓨터에서 모듈 설치과정에서 일부 에러가 계속 발생해, 일부코드를 수정했습니다 결과는 동일합니다 (권태현, 이성호 코드 pd.series를 제외하고 다른코드로 바꾸었습니다). 또한 컴파일 과정에서 계속해서 에러가 발생해 디버깅을 하는 과정에서 실행하는데 너무 오래걸려, num_iteration을 10^5 or 4-> 전부 1000으로 임의로 작게 수정했습니다.

정원렬 (runner)

코로나상황에 대해서 백신 도입 example을 만들려고 했으나, 코드를 짜기 어려워 아직 완성하지 못했다고 합니다.

대안으로

정원렬 학생은 skiier example과 유사한 example을 제안했습니다.

90 meter로 state개수가 늘어났으며, action도 3가지 running, interval, rest 3개의 action이 존재합니다.

Case introduction

Problem discription

There are three ways to get to 90 meters in total 90 meters. interval, running ,rest. Probability is below

Preparation and description note

```
import numpy as np
import pandas as pd
gamma = 1
states = np.arange(0,100,10).astype('str')
P_running=pd.DataFrame(np.matrix([[0,1,0,0,0,0,0,0,0,0],
                                  [0,0,1,0,0,0,0,0,0,0],
                                  [0,0,0,1,0,0,0,0,0,0],
                                  [0,0,0,0,1,0,0,0,0,0],
                                  [0,0,0,0,0,1,0,0,0,0],
                                  [0,0,0,0,0,0,1,0,0,0],
                                  [0,0,0,0,0,0,0,1,0,0],
                                  [0,0,0,0,0,0,0,0,1,0],
                                  [0,0,0,0,0,0,0,0,0,1],
                                  [0,0,0,0,0,0,0,0,0,1],



                                  ]), index=states,columns=states)
P_rest=pd.DataFrame(np.matrix([[1,0,0,0,0,0,0,0,0,0],
                                  [0,1,0,0,0,0,0,0,0,0],
                                  [0,0,1,0,0,0,0,0,0,0],
                                  [0,0,0,1,0,0,0,0,0,1],
                                   [0,0,0,0,1,0,0,0,0,0],
                                  [0,0,0,0,0,1,0,0,0,0],
                                  [0,0,0,0,0,0,1,0,0,0],
                                  [0,0,0,0,0,0,0,1,0,0],
                                  [0,0,0,0,0,0,0,0,1,0],
                                  [0,0,0,0,0,0,0,0,0,1]

                                  ]), index=states,columns=states)
P_interval=pd.DataFrame(np.matrix([
                                    [0,.3,0,.7,0,0,0,0,0,0],
                                   [0,0,.3,0,.7,0,0,0,0,0],
                                   [0,0,0,.3,0,.7,0,0,0,0],
                                   [0,0,0,0,.3,0,.7,0,0,0],
                                   [0,0,0,0,0,.3,0,.7,0,0],
                                   [0,0,0,0,0,0,.3,0,.7,0],
```

```
                                    [0,0,0,0,0,0,0,.3,0,.7],
                                    [0,0,0,0,0,0,0,0,.3,.7],
                                    [0,0,0,0,0,0,0,0,0,1],
                                    [0,0,0,0,0,0,0,0,0,1]
]), index=states, columns=states)
q_s_a_init = pd.DataFrame(np.zeros((len(states),3)),states,["running","interval","rest"])
```

R and Pi(probability for strategy) are below

```
# reword
R_s_a=pd.DataFrame(np.array([-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
                             -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,
                              0.1,.1,.1,.1,.1,.1,.1,.1,.1,.1]).reshape(len(states),3,order='F'),columns=['ru
R_s_a.T
```

```
##             0   10   20   30   40   50   60   70   80   90
## running  -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
## interval -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0
## rest      0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
```

```
# pi_50
pi=pd.DataFrame(np.array([0.45,0.9,0.1]*10).reshape(10,3),index=states, columns=['running','interval','rest']
pi.T
```

```
##              0    10    20    30    40    50    60    70    80    90
## running   0.45  0.45  0.45  0.45  0.45  0.45  0.45  0.45  0.45  0.45
## interval  0.90  0.90  0.90  0.90  0.90  0.90  0.90  0.90  0.90  0.90
## rest      0.10  0.10  0.10  0.10  0.10  0.10  0.10  0.10  0.10  0.10
```

Implement

```
def simul_path(pi,P_interval,P_running,P_rest,R_s_a):
    s_now = "0"
    history_i = [s_now]
    while s_now != '90':
        if np.random.uniform(0,1,1) < pi.loc[s_now,"running"] :


            a_now = "running"
```

```
            P = P_running
        elif np.random.uniform(0,1,1) < pi.loc[s_now,"interval"] :
            a_now = "interval"
            P = P_interval
        else:
            a_now = "rest"
            P = P_rest


        r_now = R_s_a.loc[s_now,a_now]

        s_next = pd.Series(np.cumsum(P.loc[s_now,])<np.random.uniform(0,1)).idxmin()
        history_i.extend([a_now,r_now,s_next])
        s_now = s_next

    return history_i
sample_path = simul_path(pi,P_interval,P_running,P_rest,R_s_a)
sample_path
```

```
## ['0', 'rest', 0.1, '0', 'interval', -2.0, '30', 'interval', -2.0, '60', 'interval', -2.0, '90']
```

```
def pol_eval_MC(sample_path, q_s_a, alpha ):
    Q_s_a = q_s_a.copy()
    for j in range(0,len(sample_path)-1,3):

        s = sample_path[j]

        a = sample_path[j+1]

        G = pd.Series(sample_path)[list(range(j+2,len(sample_path),3))].astype('float').sum()

        Q_s_a.loc[s,a] = Q_s_a.loc[s,a] +alpha*(G- Q_s_a.loc[s,a])

    return Q_s_a
q_s_a = pol_eval_MC(sample_path,q_s_a_init,alpha = 0.1)
q_s_a
```

```
##     running  interval  rest
## 0       0.0      -0.6 -0.59
## 10      0.0       0.0  0.00
## 20      0.0       0.0  0.00
```

```
## 30      0.0      -0.4  0.00
## 40      0.0       0.0  0.00
## 50      0.0       0.0  0.00
## 60      0.0      -0.2  0.00
## 70      0.0       0.0  0.00
## 80      0.0       0.0  0.00
## 90      0.0       0.0  0.00
```

```
# Skiier.R(7)
def pol_imp(pi,q_s_a,epsilon):
    Pi = pi.copy()
    for i in list(pi.index):
        if np.random.uniform(0,1,1) > epsilon:

            Pi.loc[i] = 0
            Pi.loc[i,q_s_a.loc[i].idxmin()]=1
            if i == '90':
                print(Pi.loc[i,q_s_a.loc[i].idxmax()
                ])

        else:
            Pi.loc[i,:] = 1/q_s_a.shape[1]
    return Pi
pi = pol_imp(pi,q_s_a, 0)
```

```
## 1.0
```

```
pi
```

```
##     running  interval  rest
## 0       0.0       1.0   0.0
## 10      1.0       0.0   0.0
## 20      1.0       0.0   0.0
## 30      0.0       1.0   0.0
## 40      1.0       0.0   0.0
## 50      1.0       0.0   0.0
## 60      0.0       1.0   0.0
## 70      1.0       0.0   0.0
## 80      1.0       0.0   0.0
## 90      1.0       0.0   0.0
```

New Situation(Experiment)

- Hard to make a code… I'm trying to make a code

One person infected corona spreads to an average of three people. if vaccine A is introduced, propagation power an average of 0.1 persons, and cost 10 persons. If vaccine B is introduced, the propagation power will be average of 0.3 persons and cost is 5.Derive the results through Td learning.

권태현 (How to go Seoul to Busan ,maximizing Satisfication)

권태현 학생은 명절에 서울에서 부산으로가는 길에 , 휴게소를 방문하는 과정을 모델링했습니다.

Seoul A B C D E Busan state가 존재하며, 각 휴게소마다 받는 reward가 다릅니다.

코드를 제가 잘이해를 못한것일수도만, MDP보다 MRP에 가깝지 않나 생각이 듭니다.

1. Problem

-In korea, Thanksgiving day, many people go to the their hometown. So, that day many cars in highway and many people take a rest in service area. There are 5 area between Seoul and Busan. And almost all vehicles stop at service area least 1. The more you use a rest area, the higher your satisfaction, but if you use a busy area, your satisfaction may decrease.

2. state

S = {Seoul,A,B,C,D,E,Busan}

(A,B,C,D,E) is name of service area

3. transition matrix

- Probability of visiting service area

4. reward

-Seoul, Busan and A,E is -1.0

-B,D is -1.5

-C is -2.0 (because area C is middle of Seoul to Busan. So, many cars want to rest there.)

states and transition matrix, reward

```
import numpy as np
import pandas as pd
states = ['0','A','B','C','D','E','1']
P_transition = pd.DataFrame(np.matrix([[0,0.05,0.15,0.4,0.25,0.05,0.1],
                                       [0.05,0,0.05,0.15,0.4,0.15,0.2],
                                       [0.15,0.05,0,0.05,0.2,0.25,0.3],
                                       [0.35,0.15,0.05,0,0.05,0.15,0.35],
                                       [0.3,0.25,0.2,0.05,0,0.05,0.15],
                                       [0.2,0.15,0.4,0.15,0.05,0,0.05],
                                      [0.1,0.05,0.25,0.4,0.15,0.05,0]]),index=states,columns=states)
print(P_transition)
```

```
##       0     A     B     C     D     E     1
## 0  0.00  0.05  0.15  0.40  0.25  0.05  0.10
## A  0.05  0.00  0.05  0.15  0.40  0.15  0.20
## B  0.15  0.05  0.00  0.05  0.20  0.25  0.30
## C  0.35  0.15  0.05  0.00  0.05  0.15  0.35
## D  0.30  0.25  0.20  0.05  0.00  0.05  0.15
## E  0.20  0.15  0.40  0.15  0.05  0.00  0.05
## 1  0.10  0.05  0.25  0.40  0.15  0.05  0.00
```

```
R_s_a = pd.DataFrame(np.matrix([-1,-1,-1.5,-2.0,-1.5,-1,-1]).reshape(len(states),1),index=states,columns=['re
print(R_s_a.T)
```

```
##           0    A    B    C    D    E    1
## reward -1.0 -1.0 -1.5 -2.0 -1.5 -1.0 -1.0
```

```
pi_stop = pd.DataFrame(np.c_[np.repeat(0,len(states))],index=states,columns=['reward'])
print(pi_stop.T)
```

```
##         0  A  B  C  D  E  1
## reward  0  0  0  0  0  0  0
```

gamma

```
gamma = 0.9
epsilon = 10**(-8)
v_old = np.array(np.zeros(7,)).reshape(7,1)
v_new = R_s_a + np.dot(gamma*P_transition, v_old)
results = v_old.T
results = np.vstack((results,v_new.T))
while np.max(np.abs(v_new-v_old)).item() > epsilon:
    v_old = v_new
    v_new = R_s_a + np.dot(gamma*P_transition, v_old)
    results = np.vstack((results,v_new.T))
results = pd.DataFrame(results, columns=states)
print(v_new.T)
```

```
##                0          A          B  ...          D          E          1
## reward -15.482393 -15.071241 -15.382861  ... -15.382861 -15.071241 -15.482393
##
## [1 rows x 7 columns]
```

```
print(results.head())
```

```
##           0         A         B         C         D         E         1
## 0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## 1 -1.000000 -1.000000 -1.500000 -2.000000 -1.500000 -1.000000 -1.000000
## 2 -2.440000 -2.237500 -2.535000 -3.035000 -2.535000 -2.237500 -2.440000
## 3 -3.426175 -3.287463 -3.685200 -4.369475 -3.685200 -3.287463 -3.426175
## 4 -4.503910 -4.297082 -4.635178 -5.377773 -4.635178 -4.297082 -4.503910
```

```
print(results.tail())
```

```
##              0          A          B        C          D          E          1
## 211 -15.482393 -15.071241 -15.382861 -17.2076 -15.382861 -15.071241 -15.482393
## 212 -15.482393 -15.071241 -15.382861 -17.2076 -15.382861 -15.071241 -15.482393
## 213 -15.482393 -15.071241 -15.382861 -17.2076 -15.382861 -15.071241 -15.482393
## 214 -15.482393 -15.071241 -15.382861 -17.2076 -15.382861 -15.071241 -15.482393
## 215 -15.482393 -15.071241 -15.382861 -17.2076 -15.382861 -15.071241 -15.482393
```

MC

```
pi = pi_stop
np.random.seed(1234)
history = list()
MC_N = 10**3
for MC_i in range(MC_N):
    s_now = '0'
    history_i = list(s_now)
    while s_now != '1':
        a_now = 'reward'
        P = P_transition
        r_now = str(R_s_a.loc[s_now][a_now])
        s_next = states[np.argmin(P.loc[s_now].cumsum()<np.random.uniform(0,1))]
        history_i.extend([a_now,r_now,s_next])
        s_now = s_next
    history.append(history_i)
history_stop = history
func = np.vectorize(lambda x: ','.join(x))
print(pd.Series(func(history_stop[:20])))
```

```
## 0     0,reward,-1.0,B,reward,-1.5,E,reward,-1.0,B,re...
## 1     0,reward,-1.0,D,reward,-1.5,0,reward,-1.0,C,re...
## 2                                      0,reward,-1.0,1
## 3     0,reward,-1.0,E,reward,-1.0,B,reward,-1.5,E,re...
## 4     0,reward,-1.0,C,reward,-2.0,D,reward,-1.5,A,re...
## 5     0,reward,-1.0,C,reward,-2.0,E,reward,-1.0,0,re...
## 6     0,reward,-1.0,D,reward,-1.5,A,reward,-1.0,E,re...
## 7                         0,reward,-1.0,C,reward,-2.0,1
## 8                         0,reward,-1.0,B,reward,-1.5,1
## 9           0,reward,-1.0,D,reward,-1.5,0,reward,-1.0,1
## 10                        0,reward,-1.0,C,reward,-2.0,1
## 11    0,reward,-1.0,B,reward,-1.5,A,reward,-1.0,0,re...
## 12                        0,reward,-1.0,D,reward,-1.5,1
## 13                                     0,reward,-1.0,1
## 14    0,reward,-1.0,D,reward,-1.5,0,reward,-1.0,D,re...
## 15    0,reward,-1.0,B,reward,-1.5,0,reward,-1.0,D,re...
## 16    0,reward,-1.0,C,reward,-2.0,B,reward,-1.5,0,re...
## 17    0,reward,-1.0,D,reward,-1.5,B,reward,-1.5,0,re...
## 18    0,reward,-1.0,C,reward,-2.0,B,reward,-1.5,E,re...
## 19    0,reward,-1.0,B,reward,-1.5,0,reward,-1.0,C,re...
## dtype: object
```

TD (Seoul -> Busan)

```
pol_eval = pd.DataFrame(np.zeros((len(states),2)),index=states,columns=['count','est'])
for episode_i in range(len(history_stop)):
    history_i = history_stop[episode_i]
    for j in range(0,len(history_i),3):
        pol_eval.loc[history_i[j]]['count']+=1
        current_cnt = pol_eval.loc[history_i[j]]['count']
        if j+3 < len(history_i):
            TD_tgt = float(history_i[j+2])+pol_eval.loc[history_i[j+3]]['est']
        else:
            TD_tgt = 0
        alpha = 1/current_cnt
        pol_eval.loc[history_i[j]]['est']+=alpha*(TD_tgt-pol_eval.loc[history_i[j]]['est'])
print(np.round(pol_eval.T,2))
```

```
##              0       A      B       C      D       E       1
## count  1919.00  582.00  822.0  1033.0  954.0  581.00  1000.0
## est      -5.74   -5.34   -5.2    -5.8   -5.9   -5.75     0.0
```

MC (Busan -> Seoul)

```
pi = pi_stop
np.random.seed(1234)
history = list()
MC_N = 10**3
for MC_i in range(MC_N):
    s_now = '1'
    history_i = list(s_now)
    while s_now != '0':
        a_now = 'reward'
        P = P_transition
        r_now = str(R_s_a.loc[s_now][a_now])
        s_next = states[np.argmin(P.loc[s_now].cumsum()<np.random.uniform(0,1))]
        history_i.extend([a_now,r_now,s_next])
        s_now = s_next
    history.append(history_i)
history_stop = history
func = np.vectorize(lambda x: ','.join(x))
print(pd.Series(func(history_stop[:20])))
```

```
## 0     1,reward,-1.0,B,reward,-1.5,E,reward,-1.0,B,re...
## 1     1,reward,-1.0,B,reward,-1.5,1,reward,-1.0,E,re...
## 2     1,reward,-1.0,C,reward,-2.0,1,reward,-1.0,B,re...
## 3     1,reward,-1.0,B,reward,-1.5,1,reward,-1.0,C,re...
## 4                         1,reward,-1.0,B,reward,-1.5,0
## 5     1,reward,-1.0,C,reward,-2.0,D,reward,-1.5,A,re...
## 6                         1,reward,-1.0,C,reward,-2.0,0
## 7                         1,reward,-1.0,C,reward,-2.0,0
## 8           1,reward,-1.0,C,reward,-2.0,D,reward,-1.5,0
## 9     1,reward,-1.0,C,reward,-2.0,1,reward,-1.0,C,re...
## 10    1,reward,-1.0,C,reward,-2.0,A,reward,-1.0,C,re...
## 11    1,reward,-1.0,B,reward,-1.5,1,reward,-1.0,C,re...
## 12    1,reward,-1.0,B,reward,-1.5,D,reward,-1.5,B,re...
## 13    1,reward,-1.0,C,reward,-2.0,1,reward,-1.0,C,re...
## 14    1,reward,-1.0,C,reward,-2.0,1,reward,-1.0,C,re...
## 15                        1,reward,-1.0,C,reward,-2.0,0
## 16    1,reward,-1.0,E,reward,-1.0,A,reward,-1.0,C,re...
## 17    1,reward,-1.0,B,reward,-1.5,A,reward,-1.0,1,re...
## 18    1,reward,-1.0,B,reward,-1.5,D,reward,-1.5,B,re...
## 19                                    1,reward,-1.0,0
## dtype: object
```

TD (Busan -> Seoul)

```
pol_eval = pd.DataFrame(np.zeros((len(states),2)),index=states,columns=['count','est'])
for episode_i in range(len(history_stop)):
    history_i = history_stop[episode_i]
    for j in range(0,len(history_i),3):
        pol_eval.loc[history_i[j]]['count']+=1
        current_cnt = pol_eval.loc[history_i[j]]['count']
        if j+3 < len(history_i):
            TD_tgt = float(history_i[j+2])+pol_eval.loc[history_i[j+3]]['est']
        else:
            TD_tgt = 0
        alpha = 1/current_cnt
        pol_eval.loc[history_i[j]]['est']+=alpha*(TD_tgt-pol_eval.loc[history_i[j]]['est'])
print(np.round(pol_eval.T,2))
```

```
##             0      A       B       C       D       E        1
## count  1000.0  515.0  831.00  948.00  696.00  527.00  1733.00
## est       0.0   -5.7   -5.66   -5.35   -5.14   -5.08    -5.62
```

강의현 (planning to lose weight in efficient way)

강의현 학생은 체중 감량 계획에 대한 문제를 정의했습니다. action은

1. 식단조절 , 주당 1kg가 빠집니다. cost, 30000 won

2. 운동하기, 주당 2kg 빠집니다. cost, 40000 won

2가지가 있습니다.

0.2의 확률로 요요현상이 발생해 3kg가 다시 찝니다. 목표는 99kg -> 90kg 감량입니다.

Fat Samuel

그림 1: samule

Samuel is 99 kilograms. He loves eating, but he knows gaining more weight is not good for his health. Now, He decided to lose weight for his lovely family. The goal is to lose up to 90 kilograms. There are two ways for losing weight.

1. Controlling Diet : It helps lose 1kg a week.

2. Doing Sports : It helps lose 2kg a week.

Of course, method 2 looks good. However, There is a hazard gaining 3kg due to the yo-yo phenomenon with probability of 0.2 Controlling diet costs 30,000 won a week and doing sports will incur a cost of 40,000 won a week. What kind of policy should I choose to lose weight? (Fortunately, when he decided to lose weight, his weight doesn't over 99kg.)

Preparation

```
gamma=1
states=np.arange(99,89,-1).astype('str')
P_diet=pd.DataFrame(np.array([[0,1,0,0,0,0,0,0,0,0], # 99kg
                [0,0,1,0,0,0,0,0,0,0], # 98kg
                [0,0,0,1,0,0,0,0,0,0], # 97kg
                [0,0,0,0,1,0,0,0,0,0], # 96kg
                [0,0,0,0,0,1,0,0,0,0], # 95kg
                [0,0,0,0,0,0,1,0,0,0], # 94kg
                [0,0,0,0,0,0,0,1,0,0], # 93kg
                [0,0,0,0,0,0,0,0,1,0], # 92kg
                [0,0,0,0,0,0,0,0,0,1], # 91kg
                [0,0,0,0,0,0,0,0,0,1]]), index=states, columns=states) # 90kg
P_diet
```

```
##     99  98  97  96  95  94  93  92  91  90
## 99   0   1   0   0   0   0   0   0   0   0
## 98   0   0   1   0   0   0   0   0   0   0
## 97   0   0   0   1   0   0   0   0   0   0
## 96   0   0   0   0   1   0   0   0   0   0
## 95   0   0   0   0   0   1   0   0   0   0
## 94   0   0   0   0   0   0   1   0   0   0
## 93   0   0   0   0   0   0   0   1   0   0
## 92   0   0   0   0   0   0   0   0   1   0
## 91   0   0   0   0   0   0   0   0   0   1
## 90   0   0   0   0   0   0   0   0   0   1
```

```
P_sport=pd.DataFrame(np.array([[0.2,0,0.8,0,0,0,0,0,0,0], # 99kg
                [0.2,0,0,0.8,0,0,0,0,0,0], # 98kg
                [0.2,0,0,0,0.8,0,0,0,0,0], # 97kg
                [0.2,0,0,0,0,0.8,0,0,0,0], # 96kg
                [0,0.2,0,0,0,0,0.8,0,0,0], # 95kg
                [0,0,0.2,0,0,0,0,0.8,0,0], # 94kg
                [0,0,0,0.2,0,0,0,0,0.8,0], # 93kg
                [0,0,0,0,0.2,0,0,0,0,0.8], # 92kg
                [0,0,0,0,0,0.2,0,0,0,0.8], # 91kg
                [0,0,0,0,0,0,0,0,0,1]]), index=states, columns=states) # 90kg
P_sport
```

```
##      99   98   97   96   95   94   93   92   91   90
## 99  0.2  0.0  0.8  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
## 98  0.2  0.0  0.0  0.8  0.0  0.0  0.0  0.0  0.0  0.0
## 97  0.2  0.0  0.0  0.0  0.8  0.0  0.0  0.0  0.0  0.0
## 96  0.2  0.0  0.0  0.0  0.0  0.8  0.0  0.0  0.0  0.0
## 95  0.0  0.2  0.0  0.0  0.0  0.0  0.8  0.0  0.0  0.0
## 94  0.0  0.0  0.2  0.0  0.0  0.0  0.0  0.8  0.0  0.0
## 93  0.0  0.0  0.0  0.2  0.0  0.0  0.0  0.0  0.8  0.0
## 92  0.0  0.0  0.0  0.0  0.2  0.0  0.0  0.0  0.0  0.8
## 91  0.0  0.0  0.0  0.0  0.0  0.2  0.0  0.0  0.0  0.8
## 90  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

```
R_s_a=pd.DataFrame(np.c_[np.repeat(-3,len(states)), np.repeat(-4,len(states))], index=states, columns=['diet'
R_s_a.loc['90']=0
R_s_a
```

```
##     diet  sport
## 99    -3     -4
## 98    -3     -4
## 97    -3     -4
## 96    -3     -4
## 95    -3     -4
## 94    -3     -4
## 93    -3     -4
## 92    -3     -4
## 91    -3     -4
## 90     0      0
```

Functions

```
def transition(given_pi, states, P_diet, P_sport):
    P_out=pd.DataFrame(np.zeros((len(states),len(states))),index=states, columns=states)

    for s in states:
        action_dist=given_pi.loc[s]
        P=action_dist['diet']*P_diet+action_dist['sport']*P_sport
        P_out.loc[s]=P.loc[s]

    return P_out
def reward_fn(given_pi):
    R_s_a=pd.DataFrame(np.c_[np.repeat(-3,len(states)), np.repeat(-4,len(states))], index=states, columns=['d
    R_s_a.loc['90']=0

    R_pi=np.asarray((given_pi*R_s_a).sum(axis=1)).reshape(-1,1)

    return R_pi
def policy_eval(given_pi):
    R=reward_fn(given_pi)
    P=transition(given_pi, states=states, P_diet=P_diet, P_sport=P_sport)

    gamma=1.0
    epsilon=10**(-8)

    v_old=np.repeat(0,10).reshape(10,1)
    v_new=R+np.dot(gamma*P, v_old)

    while np.max(np.abs(v_new-v_old))>epsilon:
        v_old=v_new
        v_new=R+np.dot(gamma*P,v_old)

    return v_new
```

Policy Evaluation

```
# Policy Evaluation

pi_50=pd.DataFrame(np.c_[np.repeat(0.5,len(states)), np.repeat(0.5,len(states))],index=states, columns=['diet
policy_eval(pi_50).T
```

```
## array([[-30.11305886, -27.48442842, -24.64884691, -21.62174771,
##         -18.31666794, -14.88026965, -11.57022569,  -7.82568028,
##          -4.98802696,   0.        ]])
```

Policy Improvement

```
# Policy Improvement
def policy_improve(V_old, pi_old, R_s_a, gamma, P_diet, P_sport):
    q_s_a=R_s_a+np.c_[np.dot(gamma*P_diet,V_old), np.dot(gamma*P_sport, V_old)]

    pi_new_vec=q_s_a.idxmax(axis=1)
    pi_new=pd.DataFrame(np.zeros(pi_old.shape), index=pi_old.index, columns=pi_old.columns)

    for i in range(len(pi_new_vec)):
        pi_new.iloc[i][pi_new_vec[i]]=1

    return pi_new
pi_old=pi_50
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_diet=P_diet, P_sport=P_sport)
pi_new
```

```
##     diet  sport
## 99   0.0    1.0
## 98   0.0    1.0
## 97   1.0    0.0
## 96   1.0    0.0
## 95   1.0    0.0
## 94   1.0    0.0
## 93   1.0    0.0
## 92   0.0    1.0
## 91   1.0    0.0
## 90   1.0    0.0
```

이성호 (Stanford Example Expansion)

이성호 학생은 기존 스탠포드 Mars Rover 예제를 약간 변형하였습니다. action이 추가되었습니다. Move_left, Move not, Move right 3가지 action 이며, 오른쪽으로 갈수록 받는 reward가 많으며, 1% 확률로 고장이 발생합니다. 문제는 기존문제에 비해 좋은것 같습니다. 파이썬 구현은 MC/TD policy evaluation 까지 구현했습니다.

1. Object

This problem was created by applying the Stanford Mars problem. Existing problems were simple models, making it easy to figure out the optimal policy. Thus, we adjusted the reward and probability to change the problem so that the simulation can tell.

2. Problem

-NASA collaborated with Stanford to send a Mars rover. If the rover goes west of the landing site, the path is safe, but there is not much to investigate. It is worth twice as much research as the west to the east. However, on the way through the crater terrain, the machine is broken and, if severe, it will stop working with a 1% chance. In this case, how can they get the biggest benefit?

Environment

```
import pandas as pd
import numpy as np
action = ['move_left','move_not', 'move_right']
state = [1,2,3,4,5,6,7] # s1 ~ s7
P_ML = pd.DataFrame(np.matrix([[0,0,0,0,0,0,0],
                               [1,0,0,0,0,0,0],
                               [0,1,0,0,0,0,0],
                               [0,0,1,0,0,0,0],
                               [0,0,0,1,0,0,0],
                               [0,0,0,0,1,0,0],
                                [0,0,0,0,0,1,0]
                                ]),index = state , columns = state)

P_MR = pd.DataFrame(np.matrix([[0,1,0,0,0,0,0],
                               [0,0,1,0,0,0,0],
                               [0,0,0,1,0,0,0],
                               [0,0,0,0,1,0,0],
                              [0,0,0,0,0,1,0],
                                [0,0,0,0,0,0,1],
                               [0,0,0,0,0,0,0]
                                ]),index = state , columns = state)

P_MN = pd.DataFrame(np.matrix([[1,0,0,0,0,0,0],
                               [0,1,0,0,0,0,0],
                               [0,0,1,0,0,0,0],
                               [0,0,0,1,0,0,0],
                              [0,0,0,0,1,0,0],
                                [0,0,0,0,0,1,0],
                               [0,0,0,0,0,0,1]
                                ]),index = state , columns = state)

pi_mars =pd.DataFrame(np.matrix([np.repeat(0.4,len(state)),np.repeat(0.2,len(state)),np.repeat(0.4,len(state)

pi_mars['move_left'][1] = 0
pi_mars['move_not'][1] = 0.6
pi_mars['move_right'][7] = 0
pi_mars['move_not'][7] = 0.6
print(pi_mars.T)
```

```
##              1    2    3    4    5    6    7
```

```
## move_left   0.0  0.4  0.4  0.4  0.4  0.4  0.4
## move_not    0.6  0.2  0.2  0.2  0.2  0.2  0.6
## move_right  0.4  0.4  0.4  0.4  0.4  0.4  0.0
```

```
reward = np.array([5,0,0,0,0,-4,10])
print(reward)
```

```
## [ 5  0  0  0  0 -4 10]
```

Simulator

```
pi = pi_mars
np.random.seed(1234)
history = []
MC_N = 1000
for MC_i in range(MC_N):
  s_now = 4 # Start s4
  history_i = [4]
  count = 0

  while count < 10 :

    probability = np.random.uniform(0,1)

    if probability < pi.loc[s_now]['move_left']:
      a_now = 'move_left'
      P = P_ML
      s_next = s_now - 1

    elif probability >= pi.loc[s_now]['move_left'] and probability < (pi.loc[s_now]['move_left'] + pi.loc[s_r

      a_now = 'move_not'
      P = P_MN
      s_next = s_now

    else:
      a_now = 'move_right'
      P = P_MR
      s_next = s_now + 1

    if s_now == 6 and probability <= 0.01:
      r_now = -20
      history_i.extend([a_now,r_now,s_next])
      break

    r_now = reward[s_now-1]
    history_i.extend([a_now,r_now,s_next])
    s_now = s_next

    count+=1
```

```
  history.append(history_i)
history[-5:]
```

```
## [[4, 'move_left', 0, 3, 'move_not', 0, 3, 'move_not', 0, 3, 'move_not', 0, 3, 'move_left', 0, 2, 'move_left', 0,
4, 5, 'move_left', 0, 4, 'move_left', 0, 3], [4, 'move_right', 0, 5, 'move_left', 0, 4, 'move_right', 0, 5, 'move_l
```

Implementation 1 (vectorized)

```
pol_eval=pd.DataFrame(np.matrix(np.zeros((len(state)*2))).reshape(len(state),2), index=state, columns=['count
print(pol_eval.T)
```

```
##          1    2    3    4    5    6    7
## count  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## sum    0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
for MC_i in range(MC_N):
  history_i = history[MC_i]

  for j in range(0,len(history_i),3):
    pol_eval.loc[history_i[j]]['count']+=1

    if j < len(history_i) :
      pol_eval.loc[history_i[j]]['sum']+= np.sum(np.array(history_i[j+2:len(history_i)-1:3]).astype(float))

    else:
      pol_eval.loc[history_i[j]]['sum']+=0

print(pol_eval.T)
```

```
##             1       2       3        4       5       6        7
## count   904.0  1138.0  1854.0   2985.0  1926.0  1228.0    920.0
## sum    9066.0  6501.0  6479.0  10677.0  5100.0  3777.0  14974.0
```

```
pol_cal=pd.DataFrame(pol_eval['sum']/pol_eval['count'])
print(pol_cal.T)
```

```
##            1         2         3         4         5         6          7
## 0  10.028761  5.712654  3.494606  3.576884  2.647975  3.075733  16.276087
```

Implementation 2 (vectorized)

```
pol_eval=pd.DataFrame(np.matrix(np.zeros((len(state)*2))).reshape(len(state),2), index=state, columns=['count
print(pol_eval.T)
```

```
##          1    2    3    4    5    6    7
## count  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## est    0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
for MC_i in range(MC_N):
    history_i=history[MC_i]

    for j in range(0,len(history_i),3):
        # update count
        pol_eval.loc[history_i[j]]['count']+=1
        current_cnt=pol_eval.loc[history_i[j]]['count']

        # return is the new info
        if j < len(history_i):
            new_info = np.sum(np.array(history_i[j+2:len(history_i)-1:3]).astype(float))

        else:
            new_info = 0

        # update the last estimate with new info
        alpha=1/current_cnt
        pol_eval.loc[history_i[j]]['est']+=alpha*(new_info-pol_eval.loc[history_i[j]]['est'])


print(pol_eval)
```

```
##     count        est
## 1   904.0  10.028761
## 2  1138.0   5.712654
## 3  1854.0   3.494606
## 4  2985.0   3.576884
## 5  1926.0   2.647975
## 6  1228.0   3.075733
## 7   920.0  16.276087
```

Implementation 3

```
pol_eval=pd.DataFrame(np.matrix(np.zeros((len(state)*2))).reshape(len(state),2), index=state, columns=['count
print(pol_eval.T)
```

```
##          1    2    3    4    5    6    7
## count  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## est    0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
for episode_i in range(len(history)):
  history_i = history[episode_i]

  # update count
  for j in range(0,len(history_i),3):
    pol_eval.loc[history_i[j]]['count'] +=1
    current_cnt =pol_eval.loc[history_i[j]]['count']

    #build TD target
    if(j < len(history_i)-3):
      TD_tgt = float(history_i[j+2])+pol_eval.loc[history_i[j+3]]['est']

    else:
      TD_tgt = 0

    # TD-updating
    alpha = 1/current_cnt

    pol_eval.loc[history_i[j]]['est'] += alpha*(TD_tgt - pol_eval.loc[history_i[j]]['est'])

pol_eval
```

```
##     count        est
## 1   904.0  11.872712
## 2  1138.0   5.781161
## 3  1854.0   2.468426
## 4  2985.0   1.212036
## 5  1926.0   1.325497
## 6  1228.0   3.080058
## 7   920.0  18.220286
```

```
q_s_a=pd.DataFrame(np.c_[np.repeat(0.0,len(state)), np.repeat(0.0,len(state)), np.repeat(0.0,len(state)) ], i
def pol_eval_MC(sample_path, q_s_a, alpha):

  for j in range(0,len(sample_path)-1,3):
      s = sample_path[j]
      a = sample_path[j+1]
      G = sum([sample_path[g] for g in range(j+2, len(sample_path)-1 , 3)])
      q_s_a.loc[s][a] = q_s_a.loc[s][a] + alpha*(G - q_s_a.loc[s][a])
  return q_s_a

q_s_a = pol_eval_MC(sample_path = history[0] , q_s_a = q_s_a, alpha = 0.1)
print(q_s_a)
```

```
##    move_left  move_not  move_right
## 1        0.0       0.0        0.00
## 2        0.0       0.0        0.00
## 3        0.0       0.0       -0.80
## 4       -0.8      -0.8       -1.12
## 5       -0.4       0.0       -1.12
## 6       -0.8       0.0       -0.40
## 7        0.0       0.0        0.00
```

```
q_s_a = pol_eval_MC(sample_path = history[1] , q_s_a = q_s_a, alpha = 0.1)
print(q_s_a)
## for i in history:
#   q_s_a = pol_eval_MC(sample_path = i , q_s_a = q_s_a, alpha = 0.1)
```

```
##    move_left  move_not  move_right
## 1      0.000     0.000      0.0000
## 2      0.000     0.000      0.0000
## 3      0.000     0.000     -0.6480
## 4     -0.648    -0.648     -0.9072
## 5     -0.360     0.000     -1.1200
## 6     -0.800     0.000     -0.4000
## 7      0.000     0.000      0.0000
```

김봉석 (Finding Shortest Path)

최초 의도는 x,y 좌표상에 랜덤한 점들을 뿌려 놓고, 이들을 모두 잇는 방법들 중 가장 작은 거리의 합을 가지는 방법을 찾아보려 하였습니다 (aka 한붓그리기)

하지만 연산 시간상의 문제와, action에 대한 정의부분 때문에 (10개의 점들이 있다고 하면, 10개의 action을 만들어야 되는 건지, state 수만큼 action이 생기는 건지 -> P_{ss}^{a}, 를 매우 많이 만들어야 하는지)

문제를 간소화 해서 각 점들로 부터 3개의 다른 점들로 이동할 수 있게 만들었습니다.

하지만 문제를 거의 랜덤하게 좌표를 설정하고 , transition 또한 random하게 설정해 매끄럽게 풀리지 않았습니다

꼭 거리 문제가 아니라도 최적 조합을 찾는 방법을 RL으로 풀수 있지 않을까라는 생각이 들었습니다 (생산라인 공정 순서 최적화 라던지)

introduction : Finding Shortest path using TD agent

In this case, I am trying to solve a path planning problem using TD agent.

Let's say there are 100 states. one of the way to find the optimal path is to find all the cases in trial and error, which will be about $100!$ it is impossible to compute 100! even if we use all the computer in the world. To solve the real world problem, I would like to learn the agent in a short time to find the optimal path even if it is not strictly optimal

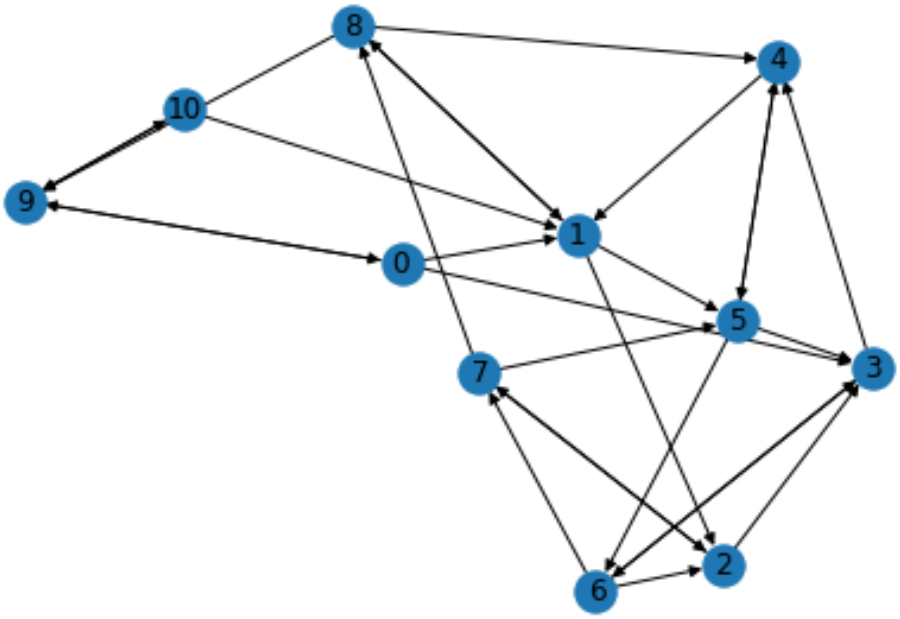


그림 2: example nodes

Problem discription

State : set of X,Y coordiantes $\{S_0, S_1, \dots S_{10}\}$, each coordinates are randomly created

Action : agent can do three action at each state, Select and move one of three paths that can go from each node. {go fisrt node, go second node, go third node}

$P_{ss'}^{a}$: transition probability with certain action in state is deterministic. It is set randomly and will be explained in detail in matrix form below

reward: Euclidean distance from s to s'

Goal : 1) find the shortest path state 0 to 8 (just randomly selected) 2) fin the shortest path start from 0 and visit all the states aka 한붓그리기

Preparation

```
import numpy as np
import pandas as pd
```

making X,Y coordinate

The coordinates of each 11 state were randomly created.

```
np.random.seed(1234)
coordinate= 10*np.random.rand(2,11)
data=pd.DataFrame(coordinate.T, columns=['X','Y'])
data
```

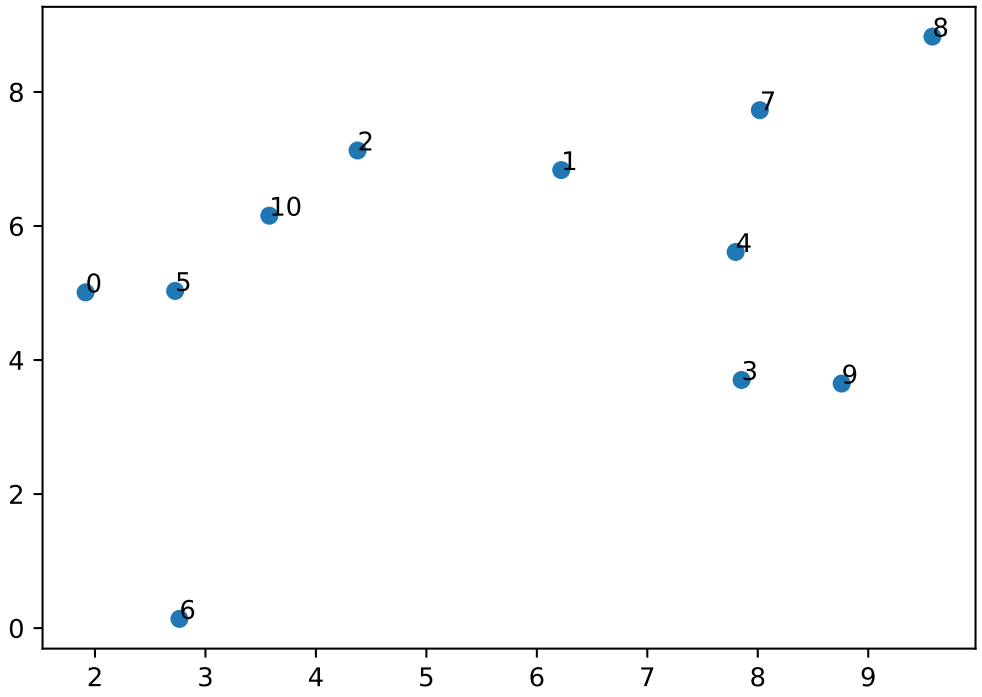
```
##            X         Y
## 0   1.915195  5.009951
## 1   6.221088  6.834629
## 2   4.377277  7.127020
## 3   7.853586  3.702508
## 4   7.799758  5.611962
## 5   2.725926  5.030832
## 6   2.764643  0.137684
## 7   8.018722  7.728266
## 8   9.581394  8.826412
## 9   8.759326  3.648860
## 10  3.578173  6.153962
```

```
import matplotlib.pyplot as plt
states = np.arange(0,11).astype(str)
y = data['Y']
```

```
x = data['X']
n = states

fig, ax = plt.subplots()
ax.scatter(x, y)

for i, txt in enumerate(n):
    ax.annotate(n[i], (x[i], y[i]))
plt.show()
```



Computing Euclidean distance

```
distance = np.zeros(shape=(11,11))
for i in range(len(data)):
    for j in range(len(data)):
        distance[i,j]=np.sqrt(np.sum(data.iloc[i]**2+data.iloc[j]**2))
distance = pd.DataFrame(distance,index=states, columns= states)

distance
```

```
##             0          1          2  ...          8          9         10
## 0    7.585194  10.685582   9.935923  ...  14.088159  10.899888   8.913032
```

```
## 1   10.685582  13.070126  12.464713  ...  15.972562  13.245908  11.665704
## 2    9.935923  12.464713  11.828354  ...  15.481073  12.648911  10.983148
## 3   10.205633  12.680752  12.055801  ...  15.655543  12.861856  11.227731
## 4   11.004450  13.332083  12.739125  ...  16.187618  13.504455  11.958466
## 5    7.842673  10.869868  10.133850  ...  14.228443  11.080610   9.133154
## 6    6.035709   9.647606   8.810061  ...  13.318065   9.884441   7.637851
## 7   12.360970  14.472045  13.927705  ...  17.138689  14.630994  13.217434
## 8   14.088159  15.972562  15.481073  ...  18.423281  16.116719  14.845310
## 9   10.899888  13.245908  12.648911  ...  16.116719  13.419387  11.862316
## 10   8.913032  11.665704  10.983148  ...  14.845310  11.862316  10.067231
##
## [11 rows x 11 columns]
```

transition prob

transition prob matrix is created randomly, It can be adjusted later.

```
P_go_first= pd.DataFrame(np.matrix([[0,1,0,0,0,0,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,0,0,1,0,0,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,0,0,0,0,1,0,0,0,0],
                                    [0,0,0,0,0,0,0,1,0,0,0],
                                    [0,0,0,0,0,0,0,0,1,0,0],
                                    [0,0,0,0,0,0,0,0,0,1,0],
                                    [0,0,0,0,0,0,0,0,0,0,1],
                                    [0,1,0,0,0,0,0,0,0,0,0],
                                                        ]),index= states, columns=states )

P_go_second= pd.DataFrame(np.matrix([[0,0,0,0,0,0,0,0,0,1,0],
                                    [0,0,0,0,0,0,0,0,1,0,0],
                                    [0,0,0,0,0,0,0,1,0,0,0],
                                    [0,0,0,0,0,0,1,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,0,0,1,0,0,0,0,0,0],
                                    [0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,1,0,0,0,0,0,0,0,0,0],
                                    [1,0,0,0,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,0,0,0,0,1,0],
                                                        ]),index= states, columns=states )
```

```
P_go_third= pd.DataFrame(np.matrix([[0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,0,1,0,0,0,0],
                                    [0,1,0,0,0,0,0,0,0,0,0],
                                    [0,0,0,1,0,0,0,0,0,0,0],
                                    [0,0,1,0,0,0,0,0,0,0,0],
                                    [0,0,0,0,0,1,0,0,0,0,0],
                                    [0,0,0,0,1,0,0,0,0,0,0],
                                    [0,0,0,0,0,0,0,0,0,0,1],
                                    [0,0,0,0,0,0,0,0,0,0,1]
                                                        ]),index= states, columns=states )

P_go_third
```

```
##     0  1  2  3  4  5  6  7  8  9  10
## 0   0  0  0  1  0  0  0  0  0  0   0
## 1   0  0  0  0  0  1  0  0  0  0   0
## 2   0  0  1  0  0  0  0  0  0  0   0
## 3   0  0  0  0  0  0  1  0  0  0   0
## 4   0  1  0  0  0  0  0  0  0  0   0
## 5   0  0  0  1  0  0  0  0  0  0   0
## 6   0  0  1  0  0  0  0  0  0  0   0
## 7   0  0  0  0  0  1  0  0  0  0   0
## 8   0  0  0  0  1  0  0  0  0  0   0
## 9   0  0  0  0  0  0  0  0  0  0   1
## 10  0  0  0  0  0  0  0  0  0  0   1
```

R_s_a

get R_s_a as the distance between the states that arrive when agent act in each state

```
import numpy as np
import pandas as pd
R_s_a = pd.DataFrame(np.c_[np.repeat( 0, len( states ) ), np.repeat( 0, len( states ) ),np.repeat( 0, len( st

R_s_a=R_s_a.astype('float')
```

```
for i in range(11):
    R_s_a['first'][i]=-distance.iloc[i,P_go_first.iloc[i][P_go_first.iloc[i].values==1][0]].astype(float)
```

```
for i in range(11):
    R_s_a['second'][i]=-distance.iloc[i,P_go_second.iloc[i][P_go_second.iloc[i].values==1][0]].astype(float)

for i in range(11):
    R_s_a['third'][i]=-distance.iloc[i,P_go_third.iloc[i][P_go_third.iloc[i].values==1][0]].astype(float)

R_s_a
```

```
##         first     second      third
## 0  -10.685582 -10.685582 -10.685582
## 1  -13.070126 -13.070126 -13.070126
## 2  -12.464713 -12.464713 -12.464713
## 3  -12.680752 -12.680752 -12.680752
## 4  -13.332083 -13.332083 -13.332083
## 5  -10.869868 -10.869868 -10.869868
## 6   -9.647606  -9.647606  -9.647606
## 7  -14.472045 -14.472045 -14.472045
## 8  -15.972562 -15.972562 -15.972562
## 9  -13.245908 -13.245908 -13.245908
## 10 -11.665704 -11.665704 -11.665704
```

policy

```
pi_50 = pd.DataFrame( np.c_[np.repeat( 1/3, len( states ) ), np.repeat( 1/3, len( states ) ),np.repeat( 1/3,
                      columns = ['first', 'second','third'] )

pi_50
```

```
##        first    second     third
## 0   0.333333  0.333333  0.333333
## 1   0.333333  0.333333  0.333333
## 2   0.333333  0.333333  0.333333
## 3   0.333333  0.333333  0.333333
## 4   0.333333  0.333333  0.333333
## 5   0.333333  0.333333  0.333333
## 6   0.333333  0.333333  0.333333
## 7   0.333333  0.333333  0.333333
## 8   0.333333  0.333333  0.333333
## 9   0.333333  0.333333  0.333333
## 10  0.333333  0.333333  0.333333
```

simul_step

```
def simul_step(pi, s_now, P_go_first,P_go_second, P_go_third, R_s_a):
    if np.random.uniform() < pi_50.loc[s_now].cumsum()[0] :
        a_now ='first'
        P = P_go_first

    elif  pi_50.loc[s_now].cumsum()[0]< np.random.uniform() < pi_50.loc[s_now].cumsum()[1]:
        a_now ='second'
        P = P_go_second

    else :
        a_now = 'third'
        P = P_go_third

    r_now = R_s_a.loc[s_now , a_now]
    s_next = states[np.argmin( P.loc[s_now].cumsum() < np.random.uniform() )]

    if np.random.uniform() < pi_50.loc[s_now].cumsum()[0] :
        a_next ='first'

    elif  pi_50.loc[s_now].cumsum()[0]< np.random.uniform() < pi_50.loc[s_now].cumsum()[1]:
        a_next ='second'

    else :
        a_next = 'third'

    sarsa = [s_now, a_now, r_now, s_next, a_next]

    return sarsa

sample_step = simul_step(pi_50, '0', P_go_first,P_go_second, P_go_third, R_s_a )

print( sample_step )
```

```
## ['0', 'first', -10.685582447534157, '1', 'second']
```

test simul step

```
for i in range(10):
    test_state = str(i)
```

```
    sample_step = simul_step(pi_50, test_state, P_go_first,P_go_second, P_go_third, R_s_a )
    print( sample_step )
```

```
## ['0', 'third', -10.685582447534157, '3', 'third']
## ['1', 'third', -13.070125529303853, '5', 'third']
## ['2', 'first', -12.464712830709807, '3', 'third']
## ['3', 'first', -12.680751685899999, '4', 'first']
## ['4', 'second', -13.332082835266162, '5', 'first']
## ['5', 'third', -10.869868014054747, '3', 'first']
## ['6', 'second', -9.647605720253912, '3', 'third']
## ['7', 'third', -14.472045062589983, '5', 'third']
## ['8', 'second', -15.97256209643203, '1', 'third']
## ['9', 'third', -13.245907551439055, '10', 'first']
```

q_s_a

```
q_s_a_init= pd.DataFrame( np.c_[np.repeat(0, len( states ) ), np.repeat(0, len( states ) ),np.repeat(0, len(
                      columns = ['first', 'second','third'] ).astype(float)

q_s_a_init
```

```
##     first  second  third
## 0     0.0     0.0    0.0
## 1     0.0     0.0    0.0
## 2     0.0     0.0    0.0
## 3     0.0     0.0    0.0
## 4     0.0     0.0    0.0
## 5     0.0     0.0    0.0
## 6     0.0     0.0    0.0
## 7     0.0     0.0    0.0
## 8     0.0     0.0    0.0
## 9     0.0     0.0    0.0
## 10    0.0     0.0    0.0
```

TD contol

```
def pol_eval_TD(sample_step, q_s_a, alpha):
    q_s_a_copy= q_s_a.copy()
    s = sample_step[0]
    a = sample_step[1]
    r = sample_step[2]
```

```
    s_next = sample_step[3]
    a_next = sample_step[4]

    q_s_a_copy.loc[s,a] +=alpha*(r+q_s_a_copy.loc[s_next, a_next]-q_s_a_copy.loc[s,a])

    return q_s_a_copy


q_s_a=pol_eval_TD(sample_step, q_s_a_init, alpha = 0.1)
q_s_a
```

```
##     first  second     third
## 0     0.0     0.0  0.000000
## 1     0.0     0.0  0.000000
## 2     0.0     0.0  0.000000
## 3     0.0     0.0  0.000000
## 4     0.0     0.0  0.000000
## 5     0.0     0.0  0.000000
## 6     0.0     0.0  0.000000
## 7     0.0     0.0  0.000000
## 8     0.0     0.0  0.000000
## 9     0.0     0.0 -1.324591
## 10    0.0     0.0  0.000000
```

```
def pol_imp(pi, q_s_a, epsilon): # epsilon = exploration_rate
    pi_copy =pi.copy()
    for i in range(pi.shape[0]):
        # exploitation
        if np.random.uniform() > epsilon:
            pi_copy.iloc[i] = 0
            pi_copy.iloc[i, np.argmax(q_s_a.iloc[i,])] = 1

        else:
            # exploration
            pi_copy.iloc[i] = 1/q_s_a.shape[1]

    return pi_copy

pol_imp(pi_50, q_s_a, epsilon=0)
```

```
##     first  second  third
```

```
## 0     1.0     0.0    0.0
## 1     1.0     0.0    0.0
## 2     1.0     0.0    0.0
## 3     1.0     0.0    0.0
## 4     1.0     0.0    0.0
## 5     1.0     0.0    0.0
## 6     1.0     0.0    0.0
## 7     1.0     0.0    0.0
## 8     1.0     0.0    0.0
## 9     1.0     0.0    0.0
## 10    1.0     0.0    0.0
```

TD iteration

goal is find shortest path S_0 to S_8 , 시간상 이유 때문에 aggregate 할때는 num iter를 100회로 조정하였습니다.

```
import time
num_ep = 1000
beg_time =time.time()
q_s_a = q_s_a_init
pi=pi_50
exploration_rate = 1


for epi_i in range(1,num_ep) :
    s_now="0"
    while s_now != "8":
        sample_step = simul_step(pi_50, s_now, P_go_first,P_go_second, P_go_third, R_s_a )
        q_s_a = pol_eval_TD(sample_step, q_s_a, alpha = 1/epi_i)
        pi = pol_imp(pi, q_s_a, epsilon= exploration_rate)
        s_now = sample_step[3]
        exploration_rate *=0.9995

end_time =time.time()

print("Time difference of {} sec".format(end_time- beg_time))
```

```
## Time difference of 123.20056319236755 sec
```

```
print(pi.T)
```

```
##          0    1    2    3    4    5    6    7    8    9   10
```

```
## first   0.0  0.0  0.0  1.0  0.0  0.0  1.0  1.0  1.0  1.0  0.0
## second  1.0  1.0  1.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  1.0
## third   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
print(q_s_a.T)
```

```
##                0          1          2  ...    8          9         10
## first  -45.166246 -70.961484 -80.410006  ...  0.0 -12.389648 -28.325295
## second -14.829152 -13.070126 -60.563934  ...  0.0 -15.512737 -10.107726
## third  -65.427305 -67.143931 -85.669973  ...  0.0 -14.708433 -16.744531
##
## [3 rows x 11 columns]
```

TD iteration 2

This time, I started with State 0 and looked for the best route to visit all states. aka 한붓그리기

```
import time
num_ep = 1000
beg_time =time.time()
q_s_a = q_s_a_init
pi=pi_50
exploration_rate = 1
```

```
import time
num_ep = 1000
beg_time =time.time()
q_s_a = q_s_a_init
pi=pi_50
exploration_rate = 1


for epi_i in range(1,num_ep) :
    s_now="0"
    history = [int(s_now)]
    while not np.array_equal(history, states.astype(int)):
        sample_step = simul_step(pi_50, s_now, P_go_first,P_go_second, P_go_third, R_s_a )
        q_s_a = pol_eval_TD(sample_step, q_s_a, alpha = 1/epi_i)
        pi = pol_imp(pi, q_s_a, epsilon= exploration_rate)
        s_now = sample_step[3]
        history.append(int(s_now))
```

```
        my_set = set(history) #집합set으로 변환
        history = list(my_set) #list로 변환
        exploration_rate *=0.9995
        #print(history)


end_time =time.time()

print("Time difference of {} sec".format(end_time- beg_time))
```

```
## Time difference of 439.1876413822174 sec
```

```
print(pi.T)
```

```
##           0    1    2    3    4    5    6    7    8    9   10
## first   0.0  0.0  0.0  1.0  0.0  0.0  1.0  1.0  1.0  1.0  0.0
## second  1.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0
## third   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
print(q_s_a.T)
```

```
##                  0           1           2  ...           8          9          10
## first  -159.496143 -222.960362 -231.284068  ...  -79.668348 -69.611213 -139.371609
## second  -50.119144 -151.925494 -208.871700  ... -182.282807 -96.894339  -44.944357
## third  -180.434290 -221.454337 -235.754237  ... -212.330702 -73.374133  -63.185006
##
## [3 rows x 11 columns]
```

Limitation

1. **Calculations at about 11 states also take longer time than expected (about 10~20 miniutes)** It may be because my computer is not good, and the code is not perfect
2. **It is a randomly generated coordinate, not a coordinate of the real world, and a process set to a task.** if the process I have done so far is valid, it would be fun to solve it using the coordinates and distances of the real world.

박재민 (Playing Robot Game)

박재민 학생은 스탠포드 예제와 유사하지만, 조금 변형이된 규칙을 가진 Robot game을 제안했습니다.

state 4에서 시작해 1에 가면 coin을 얻고, 그때 state 7에 종료 깃발이 나타납니다.

흥미로운 점은 sate 1에서 왼쪽으로 가면 state 4로 이동하며, 7에서 오른쪽으로 가면 state 4로 이동합니다(자살)

Case introduction

Problem discription

Robot is playing game. Robot starts at state 4, and there is coin in state 1.

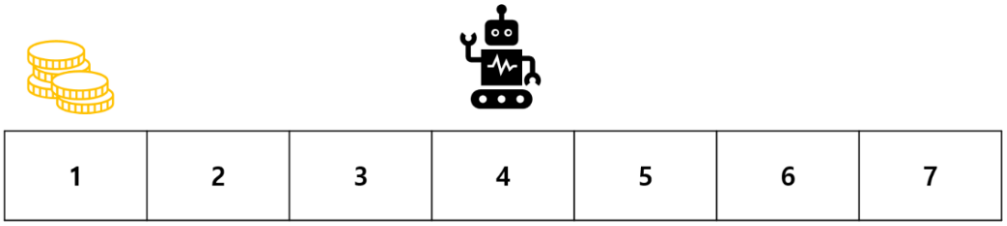


그림 3: example nodes

When robot gain coin, then finish flag is appear at state 7.

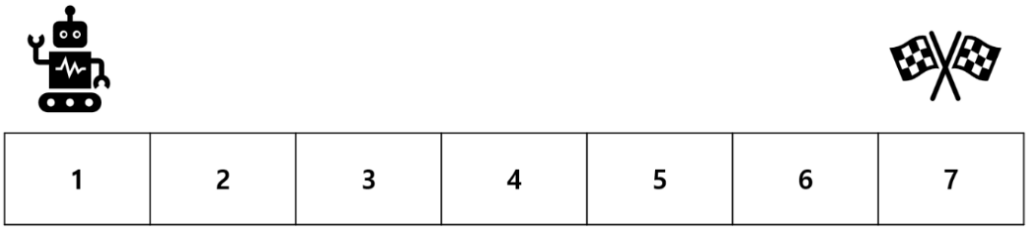


그림 4: example nodes

If robot gain finish flag, then game is over. One thing to notice is that when robot chooses suicide it self(state = 1, action = left or state = 7, action = right), robot will be restart to its first starting position(state 4) with no disadvantage.

Purpose of this case is to find out fastest route to finish game. Sometimes if suicide makes better solution, will robot try to suicide itself?

State = {1,2,3,4,5,6,7} Initial transition probability = Left 0.5, Right 0.5
Reward = each step -1, get coin +10, finish flag +50

Code Implementation

This time, finding best routh algorithm uses Monte Carlo method using just Initial transition probability. After learning ϵ -greedy method, code will be updated

```
import numpy as np
import pandas as pd
#if robot gain coin, then True
Coin_Flag = False
#States
states = np.arange(1,8,1)
#action left matrix
P_L = np.matrix([[0,0,0,1,0,0,0],[1,0,0,0,0,0,0],
[0,1,0,0,0,0,0],[0,0,1,0,0,0,0],[0,0,0,1,0,0,0],
[0,0,0,0,1,0,0],[0,0,0,0,0,1,0]])
#action right matrix
P_R = np.matrix([[0,1,0,0,0,0,0],[0,0,1,0,0,0,0],
[0,0,0,1,0,0,0],[0,0,0,0,1,0,0],[0,0,0,0,0,1,0],
[0,0,0,0,0,0,1],[0,0,0,1,0,0,0]])
P_L = pd.DataFrame(P_L,states,states)
P_R = pd.DataFrame(P_R,states,states)
#RSA1 - Robot not have coin
R_s_a_1 = np.matrix([[-1,+10,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,-1,-1]]).T
R_s_a_1 = pd.DataFrame(R_s_a_1,states,["L","R"])
#RSA2 - Robot have coin
R_s_a_2 = np.matrix([[-1,-1,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,+50,-1]]).T
R_s_a_2 = pd.DataFrame(R_s_a_2,states,["L","R"])
print(R_s_a_1.T)
```

```
##    1   2  3  4  5  6  7
## L -1  10 -1 -1 -1 -1 -1
## R -1  -1 -1 -1 -1 -1 -1
```

```
print(R_s_a_2.T)
#QSA1 - Robot not have coin
```

```
##    1  2  3  4  5   6  7
## L -1 -1 -1 -1 -1  -1 -1
## R -1 -1 -1 -1 -1  50 -1
```

```
q_s_a_init_1 = np.hstack((np.zeros(len(states)).reshape(7,1),np.zeros(len(states)).reshape(7,1)))
q_s_a_init_1 = pd.DataFrame(q_s_a_init_1,states,["L","R"])
```

```
#QSA2 - Robot have coin
q_s_a_init_2 = np.hstack((np.zeros(len(states)).reshape(7,1),np.zeros(len(states)).reshape(7,1)))
q_s_a_init_2 = pd.DataFrame(q_s_a_init_2,states,["L","R"])
print(q_s_a_init_1.T)
```

```
##      1    2    3    4    5    6    7
## L  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## R  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
print(q_s_a_init_2.T)
```

```
##      1    2    3    4    5    6    7
## L  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## R  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
#R2
#left right 50,50 policy
pi_50 = np.hstack((np.repeat(0.5,len(states)).reshape(7,1),np.repeat(0.5,len(states)).reshape(7,1)))
pi_50 = pd.DataFrame(pi_50,states,["L","R"])
print(pi_50.T)
#R3
#path simulator using pi_50
```

```
##      1    2    3    4    5    6    7
## L  0.5  0.5  0.5  0.5  0.5  0.5  0.5
## R  0.5  0.5  0.5  0.5  0.5  0.5  0.5
```

```
def simul_path(pi,P_L,P_R,R_s_a_1,R_s_a_2,Coin_Flag):
    s_now = 4
    history_i = []
    while(not Coin_Flag or s_now!=7):
        if(np.random.uniform(0,1)<pi.loc[s_now]["L"]):
            a_now="L"
            P=P_L
        else:
            a_now="R"
            P=P_R
        if(not Coin_Flag):
            r_now = R_s_a_1.loc[s_now][a_now]
        else:
            r_now = R_s_a_2.loc[s_now][a_now]

        s_next = states[np.argmin(P.loc[s_now].cumsum()<np.random.uniform(0,1))].item()
        history_i.extend([s_now,a_now,r_now])
        s_now = s_next

        if(s_now ==1 and not Coin_Flag):
            Coin_Flag = True

    return(history_i)
sample_path = simul_path(pi_50, P_L, P_R, R_s_a_1, R_s_a_2, Coin_Flag)
```

TD method function simul_step is not used this time

```
#R4
#sample step simulator. This time, we are using MC method, This TD method function is not used.
def simul_step(pi,s_now,P_L,P_R,R_s_a_1,R_s_a_2,Coin_Flag):
    if(np.random.uniform(0,1)<pi.loc[s_now]["L"]):
        a_now="L"
        P=P_L
    else:
        a_now="R"
        P=P_R
    if(not Coin_Flag):
        r_now = R_s_a_1.loc[s_now][a_now]
    else:
        r_now = R_s_a_2.loc[s_now][a_now]
    s_next = states[np.argmin(P.loc[s_now].cumsum()<np.random.uniform(0,1))].item()
    if(s_next ==1 and not Coin_Flag):
            Coin_Flag = True
    if(np.random.uniform(0,1)<pi.loc[s_next]["L"]):
        a_next="L"
    else:
        a_next="R"
    sarsa = [s_now,a_now,r_now,s_next,a_next]
    return(sarsa,Coin_Flag)
sample_step,flag = simul_step(pi_50, 4, P_L, P_R, R_s_a_1,R_s_a_2,Coin_Flag) # a.k.a. sarsa
Coin_Flag = flag
#print(sample_step)
```

```
#R5
#policy evaluation
def pol_eval_MC(sample_path, q_s_a_1, q_s_a_2, alpha):
    flag_qsa = False
    for j in range(0,len(sample_path),3):
        if(sample_path[j+2] == 10):
            flag_qsa = True
        s = sample_path[j]
        a = sample_path[j+1]
        G = np.sum(np.array(sample_path[j + 2:len( sample_path )-1:3]).astype( float ) )

        if(not flag_qsa):
            q_s_a_1.loc[s][a] = q_s_a_1.loc[s][a] + alpha*(G-q_s_a_1.loc[s][a])
        else:
            q_s_a_2.loc[s][a] = q_s_a_2.loc[s][a] + alpha*(G-q_s_a_2.loc[s][a])
    return (q_s_a_1,q_s_a_2)
q_s_a_1, q_s_a_2 = pol_eval_MC(sample_path,q_s_a_init_1,q_s_a_init_2,0.1)
#policy improve
def pol_imp(q_s_a,epsilon):
    pi = np.hstack((np.zeros(len(states)).reshape(7,1),np.zeros(len(states)).reshape(7,1)))
    pi = pd.DataFrame(pi,states,["L","R"])
    for i in range(pi.shape[0]):
        if(np.random.uniform(0,1)>epsilon):
            pi.iloc[i] = 0
            pi.iloc[i][q_s_a.iloc[i].idxmax()]=1
        else:
            pi.iloc[i] = 1/q_s_a.shape[1]
    return pi
```

```
MC_N = 1000
for i in range(MC_N):
    sample_path = simul_path(pi_50, P_L, P_R, R_s_a_1, R_s_a_2, Coin_Flag )
    #print(sample_path)
    q_s_a_1, q_s_a_2 = pol_eval_MC(sample_path,q_s_a_1,q_s_a_2,0.1)
pi_1 = pol_imp(q_s_a_1, 0)
pi_2 = pol_imp(q_s_a_2, 0)
print(q_s_a_1.T)
```

```
##      1          2          3          4          5          6          7
## L  0.0   0.000000 -24.966885 -30.407068 -44.477538 -37.462560 -31.226087
## R  0.0 -29.803409 -34.355097 -38.755599 -33.918676 -32.343504 -33.895962
```

```
print(q_s_a_2.T)
```

```
##            1          2          3          4          5         6    7
## L -20.282915 -22.070877 -21.624548 -22.445440 -15.209321 -12.92846  0.0
## R -27.955276 -24.222812 -23.073038 -11.981834  -7.490341   0.00000  0.0
```

```
print(pi_1.T)
```

```
##      1    2    3    4    5    6    7
## L  1.0  1.0  1.0  1.0  0.0  0.0  1.0
## R  0.0  0.0  0.0  0.0  1.0  1.0  0.0
```

```
print(pi_2.T)
```

```
##      1    2    3    4    5    6    7
## L  1.0  1.0  1.0  0.0  0.0  0.0  1.0
## R  0.0  0.0  0.0  1.0  1.0  1.0  0.0
```

Result

Before having coin at the state 6 and 7, to reach state 1, selecting suicide is better option than just going to state 1. Also after having coin to reach state 7, at the state 1 and 2, suicide option is faster than just going to state 7.
Since this code uses only pi_50 policy with MC method, result may not be accurate. But we can find out that robot similarly calculated that suicide can make some better option.

```
"Done "
```

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