D_case

Lee Sung Ho

2021 2 5

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(s

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 = 10000
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.l

      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_right', 0, 5, 'move_left', 0, 4, 'move_left', 0, 3, 'move_left', 0, 2, 'move_right', 0, 3
```

Implementation 1 (vectorized)

```
pol_eval=pd.DataFrame(np.matrix(np.zeros((len(state)*2))).reshape(len(state),2), index=state, columns=[
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']+=pd.Series(history_i)[range(j+2,len(history_i)-1,3)].sum()

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

print(pol_eval.T)
```

```
##              1        2        3         4        5        6         7
## count   8743.0  12178.0  19137.0   29699.0  18965.0  12039.0    8917.0
## sum    84136.0  61881.0  62144.0  102696.0  50249.0  42269.0  146650.0
```

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

```
##           1         2         3         4         5         6          7
## 0  9.623241  5.081376  3.247322  3.457894  2.649565  3.511006  16.446114
```

Implementation 2 (vectorized)

```
pol_eval=pd.DataFrame(np.matrix(np.zeros((len(state)*2))).reshape(len(state),2), index=state, columns=[
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=pd.Series(history_i)[range(j+2,len(history_i)-1,3)].sum()

        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   8743.0   9.623241
## 2  12178.0   5.081376
## 3  19137.0   3.247322
## 4  29699.0   3.457894
## 5  18965.0   2.649565
## 6  12039.0   3.511006
## 7   8917.0  16.446114
```

Implementation 3

```
pol_eval=pd.DataFrame(np.matrix(np.zeros((len(state)*2))).reshape(len(state),2), index=state, columns=[
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   8743.0  12.498189
## 2  12178.0   6.513791
## 3  19137.0   3.509934
## 4  29699.0   2.274770
## 5  18965.0   2.482039
## 6  12039.0   4.469550
## 7   8917.0  19.565224
```

```
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))

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
```