Observasi Reinforcement Learning dengan model Q-Learning

Oleh : Rizaldi Ramdlani P/1301178467

Dari : IF-41-GAB02

Pada kasus data set yang tertera dibawah untuk mencari path atau jalur dengan bobot terbesar, akan digunakan teknik reinforcement learning metode Q-Learning

```
0,1,2,3,4,5,6,7,8,9
-1,-3,-5,-1,-3,-3,-5,-5,-1,100
-2,-1,-1,-4,-2,-5,-3,-5,-5,-5
-3,-4,-4,-1,-3,-5,-5,-4,-3,-5
-3,-5,-2,-5,-1,-4,-5,-1,-3,-4
-4,-3,-3,-2,-1,-1,-1,-4,-3,-4
-4,-2,-5,-2,-4,-5,-1,-2,-2,-4
-4,-3,-2,-3,-1,-3,-4,-3,-1,-3
-4,-2,-5,-4,-1,-4,-5,-5,-2,-4
-2,-1,-1,-4,-1,-3,-5,-1,-4,-1
-5,-3,-1,-2,-4,-3,-5,-2,-2,-2
```

Menggunakan bahasa pemrograman python, hal yang pertama dilakukan adalah membaca data training set diatas ke dalam format yang lebih mudah untuk diproses oleh mesin penghitung yitu array atau list.

```
data_test = panda.read_csv("train_data.csv")
```

Hal selanjutnya yang perlu dilakukan adalah membentuk fungsi untuk menciptakan array atau matriks nol dengan dimensi 10*10 beserta 4 aksi yaitu atas(U), bawah(B), kanan(R), dan kiri(L).

```
def create_qtb():
    arr = []
    for i in range(0,len(data_test)):
        for x in range(0,len(data_test)):
            arr.append({"X":i,"Y":x,"U":0,"B":0,"R":0,"L":0})
    return arr
```

Setelah tabel Q berhasil dibuat maka langkah selanjutnya adalah mengisinya, jika diingat dalam metode Q-Learning, hal yang pertama dilakukan adalah eksplorasim eksplorasi dilakukan dalam bentuk episode percobaan langkah yang diulang berkali-kali.

```
for _ in range(3000):
    episode(19)
```

Tapi apa yang terjadi dalam episode? Akan ada tabel stage di setiap episode yang dieksekusi oleh mesin, tabel akan menyimpan state awal, reward pada state itu, aksi dan state selanjutnya setelah aksi. Tabel

tersebut akan digunakan nantinya untuk mengisi tabel Q yang masih kosong tadi. Bagaimana penentuan langkah yang dilakukan setiap episode dilakukan, singkatnya langkah dipilih secara acak dengan bobot terbesar yaitu langkah kanan dan atas mengingat tujuan dari eksplorasi ini adalah nilai 100 di sudut kanan atas dari data uji. Bisa dikatakan hal ini sama seperti pendekatan dengan metode greedy/serakah, tapi nanti akan terlihat bahwa menggunakan metode ini hasil yang diperoleh akan jauh lebih optimal, dibandingkan dengan metode acak terdistribusi.

```
def episode(step):
    (x,y) = (9,0)
    (v,w) = (x,y)
    episode_stage = []
    for ind in range(1,step):
        act = avail_act_greedy(x,y)
        if act == 'U' :
            x = x - 1
        elif act == 'B' :
            x = x + 1
        elif act == 'R' :
            y = y + 1
        elif act == 'L' :
            y = y - 1

        arr = {"num":ind,"state":(v,w),"action":act,"reward":data_test.iloc[(x,y)],"n-state":(x,y)}
        episode_stage.append(arr)
        (v,w) = (x,y)

    fil_qtb(episode_stage,1,0.5)
```

Seperti apa metode acak (avail_act_greedy()) tersebut bekerja?

```
def avail_act_greedy(x,y):
    max_len = len(data_test)-1
    min_len = 0
    act = ''

    if( x+1 <= max_len and y+1 <= max_len and y-1 >= min_len and x-1 >= min_len ):
        act = random.choice(['U','R'])
    else:
        if( x+1 <= max_len and y+1 <= max_len and y-1 < min_len and x-1 >= min_len ):
            act = random.choice(['U','R'])
        elif( x+1 <= max_len and y+1 <= max_len and y-1 >= min_len and x-1 < min_len ):
            act = random.choice(['R'])
        elif( x+1 <= max_len and y+1 > max_len and y-1 >= min_len and x-1 >= min_len ):
            act = random.choice(['U'])
        elif( x+1 > max_len and y+1 <= max_len and y-1 >= min_len and x-1 >= min_len ):
            act = random.choice(['U','R'])
        elif( x+1 > max_len and y+1 <= max_len and y-1 < min_len and x-1 >= min_len ):
            act = random.choice(['U','R'])
        elif( x+1 <= max_len and y+1 > max_len and y-1 >= min_len and x-1 < min_len ):
            act = random.choice(['B','L'])
        elif( x+1 > max_len and y+1 > max_len and y-1 >= min_len and x-1 >= min_len ):
            act = random.choice(['U','L'])
        elif( x+1 <= max_len and y+1 <= max_len and y-1 < min_len and x-1 < min_len ):
            act = random.choice(['R'])
    return act
```

Singkatnya metode ini hanya mengambil semua kondisi yang mungkin terjadi ketika agen melakukan penjelajahan dalam daerah berbentuk persegi, kondisi tersebut bisa jadi adalha ketika dia tidak bisa terus maju keatas maka akan dipilih jalur kiri kanan, atau bawah, namun dengan pendekatan greedy maka jalur yang dipilih adalah jalur kanan, katakan jika agen berada di bagian paling bawah maka yang bisa ia lakukan adalah bergerak ke kanan, atas, dan kiri, tapi dengan pendekatan greedy maka langkah yang diambil adalah langkah kanan atau atas, dan begitu selanjutnya untuk keadaan di sudut atas sisi samping, atau sisi atas bawah dan lain-lain, semua kondisi telah terpenuhi dalam fungsi diatas. Langkah yang dieksekusi tiap episode sesuai dengan parameter yang dimasukan saat fungsi episode dipanggil dalam perulangan.

Setelah menjelaskan tentang episode maka langkah selanjutnya adalah mengisi tabel atau variabel qtr yang telah diinisialisasi dengan nilai no disaat pertama tadi.

```
fil_qtb(episode_stage,1,0.5)
```

Bisa dikatakan bahwa ini sudah selesai, tidak sesederhana itu. Fill_qtb() adalah fungsi yang cukup rumit namun adalah fungsi yang paling dasar yang mengimplementasikan model Q-learning itu sendiri.

```
def fil_qtb(episode_stage,alpha,lamda):
    for eps in episode_stage :
        target_state = get_qtb(eps['state'][0],eps['state'][1],eps['action'])
        max_action_state = get_max_action(eps['n-state'][0],eps['n-state'][1])
        reward = eps['reward']

        for qt in qtb:
            if (qt['X'] == eps['state'][0] and qt['Y'] == eps['state'][0]) :
                qt[str(eps['action'])] = target_state + alpha*(reward
                +lamda*max_action_state-target_state)
```

Ada 2 fungsi lain yang dipanggil disini yaitu target_state, dan max_action_state. Fungsi apa itu? Sederhananya fungsi target state adalah fungsi yang digunakan untuk memanggil nilai state(x,y) pada aksi ke eps[‘action’]. Sedangkan max_action_state adalah fungsi untuk mencari nilai tertinggi dari aksi (U,B,R,L) pada state (x,y). Setelah semua nilai terinisialisasi maka langkah selanjutnya adalah menghitung satu-persatu(DALAM PERULANGAN) nilai Q[x,y] untuk dimasukan ke dalam tabel.

$$Q[s,a] = Q[s,a] + \alpha\left(reward + \gamma * Qmax^{a'}(s,a') - Q[s,a]\right)$$

Nilai lambda dan alpha dijadikan parameter.

Katakanlah episode telah dieksekusi sebanyak 1000 perulangan dengan 19 langkah tiap perulangannya, dan tabel Q telah terisi semua dan semuanya selesai. Eksplorasi mungkin selesai tapi belum dengan eksploitasi, dan bagian ini yang memang cukup menarik. Dan disini juga akan terbukti bahwa metode greedy bisa lebih hebat dari metode acak biasa.

Pemanggilan method untuk fungsi explooit ini memang mudah (sangat mudah)

```
exploit_all(9,0)
```

Dibalik layar methode tersebut akan dijelaskan selanjutnya,

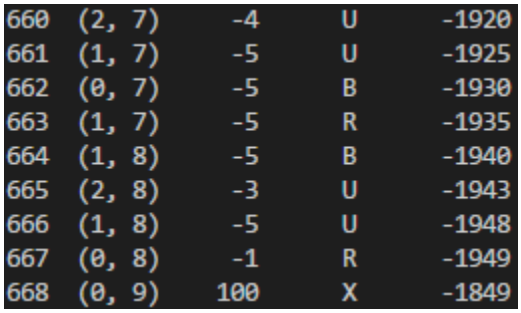
```
def exploit_all(x,y):
    maximum = 0
    act = ''
    reward = 0
    sumReward = 0
    act_list = []

    while (True):
        for qt in qtb:
            if (qt['X'] == x and qt['Y'] == y) :
                (v,w) = (x,y)
                maximum = max([qt['U'],qt['B'],qt['R'],qt['L']])
                reward = data_test.iloc[x,y]
                sumReward = reward + sumReward

                if( maximum == 0):
                    act = avail_act_greedy(x,y)
                    if act == 'U' :
                        x = x - 1
                        break
                    elif act == 'B' :
                        x = x + 1
                        break
                    elif act == 'R' :
                        y = y + 1
                        break
                    elif act == 'L' :
                        y = y - 1
                        break
                else:
                    if( maximum == qt['U']):
                        act = 'U'
                        x = x - 1
```

```
                    x = x - 1
                    break
                elif( maximum == qt['B']):
                    act = 'B'
                    x = x + 1
                    break
                elif( maximum == qt['R']):
                    act = 'R'
                    y = y + 1
                    break
                elif( maximum == qt['L']):
                    act = 'L'
                    y = y - 1
                    break

        if(reward == 100):
            act = 'X'
        lis = [(v,w),reward,act,sumReward]
        act_list.append(lis)
        if(reward == 100):
            break

    result = panda.DataFrame(np.array(act_list).reshape(len(act_list),4), columns = ['State','Reward',
    'Action','Sum Reward'])
    print (result)
```

Fungsi ini akan menjelajahi tabel Q dengan untuk mencari langkah dengan reward tertinggi di tabel Q itu bukan di data uji sampai mendapat langkah dengan reward 100 maka fungsi akan berhenti. Bagaimana agen menentukan langkahnya disini jika semua langkah bernilai 0 atau bernilai sama, jika seemua aksu bernilai 0 maka fungsi avail_act_greedy() akan kembali dipanggil. Jika agen sudah mencapai langkah dengan nilai 100 maka perulangan akan dihentikan dan hasil akan muncul

	State	Reward	Action	Sum Reward
0	(9, 0)	-5	U	-5
1	(8, 0)	-2	R	-7
2	(8, 1)	-1	U	-8
3	(7, 1)	-2	U	-10
4	(6, 1)	-3	U	-13
5	(5, 1)	-2	R	-15
6	(5, 2)	-5	R	-20
7	(5, 3)	-2	U	-22
8	(4, 3)	-2	R	-24
9	(4, 4)	-1	U	-25
10	(3, 4)	-1	U	-26
11	(2, 4)	-3	R	-29
12	(2, 5)	-5	U	-34
13	(1, 5)	-5	U	-39
14	(0, 5)	-3	R	-42
15	(0, 6)	-5	R	-47
16	(0, 7)	-5	R	-52
17	(0, 8)	-1	R	-53
18	(0, 9)	100	X	47

Menggunakan metode diatas maka hasil yang didapat adalah 47 dengan 19(0-18) langkah, dan jika kita menggunakan metode acak normal maka yang akan didapat adalah



660	(2, 7)	-4	U	-1920
661	(1, 7)	-5	U	-1925
662	(0, 7)	-5	B	-1930
663	(1, 7)	-5	R	-1935
664	(1, 8)	-5	B	-1940
665	(2, 8)	-3	U	-1943
666	(1, 8)	-5	U	-1948
667	(0, 8)	-1	R	-1949
668	(0, 9)	100	X	-1849

Gambar diatas adalah hasil dari metode non greedy, langkah yang dieksekusi banyak dan nilai nya pun bahkan bukan nilai, terbaik.

Dengan beberapa percobaan dilakukan dimulai dari 1000 sampai 5000 perulangan dengan langkah bervariatif antara 19-200, nilai atau reward terbesar yang di dapat rata-rata berada di nilai 36 – 47 Atau bisa dikatakan rata-rata reward terbesar adalah 40.