Stochastic Models and Optimization

A REINFORCEMENT LEARNING - MONTE CARLO MODEL FOR TRIP PLANNING

March 31, 2019

Professor: Dr. Mihalis Markakis

By: Maryam Rahbaralam

Barcelona Graduate School of Economics

Table of Contents

1 Introduction

Reinforcement learning (RL) is all about learning from the environment through interactions. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment and ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, and we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence [1,2]. Finding the optimal policy/optimal value functions is the key for solving reinforcement learning problems. Dynamic programming methods are also used to find optimal policy/optimal value functions using optimality equations [3]. These methods are model based methods, require the complete knowledge of environment, such as transition probabilities and rewards. There are two types of tasks in RL:

1. Prediction : This type of task predicts the expected total reward from any given state assuming the function $\pi(\mathbf{a}|\mathbf{s})$ is given. In other words, the policy π is given, it calculates the value function $\mathbf{V}_\pi$ with or without the model. ex: Policy evaluation

2. Control: This type of task finds the policy $\pi(\mathbf{a}|\mathbf{s})$ that maximizes the expected total reward from any given state. In other words, some policy π is given, it finds the optimal policy π^*.

   Policy iteration is the combination of both to find the optimal policy. Just like in supervised learning, we have regression and classification tasks, in reinforcement learning, we have prediction and control tasks.

There are two types of policy learning methods:

1. On policy learning : It learns on the job, which means it evaluates or improves the policy that is used to make the decisions. In other words, it directly learns a policy which gives you decisions about which action to take in some state.

2. Off policy learning : It evaluates one policy (target policy) while following another policy (behavior policy) just like we learn to do something while observing others doing the same thing. Target policy may be deterministic (ex: greedy) while behaviour policy is stochastic.

In RL problems we have two different tasks in nature:

1. Episodic task : A task which can last a finite amount of time is called Episodic task (an episode). Ex : Playing a game of chess (win or lose or draw). We only get the reward at the end of the task or another option is to distribute the reward evenly

across all actions taken in that episode. Ex: you lost the queen (-10 points), you lost one of the rooks (-5 points) etc.

2. Continuous task : A task which never ends is called Continuous task. Ex: Trading in the cryptocurrency markets or learning Machine learning on internet. In this , rewards may be given with discounting with a discount factor $\lambda \in [0, 1]$.

2 Problem statement

2.1 Motivation

One of the interesting problems in travel planning is to choose between different hotel options in our target cities. This choice is based on our planned budget. Some of the hotels are less expensive, others are of higher quality. I aim to create a model that, for the target cities, can select the optimal hotels required to make a trip that is both within the budget and meets personal preferences. For this purpose, I will build a simple model that can recommend the hotels that are below my budget introducing my preferences. This model, in theory, can be employed to large problems with many cities and hotels that would cause the problem to then be beyond the possibility of any mental calculations.

2.2 Method

To achieve this, a Monte Carlo method based on reinforcement learning model is employed to find the optimal combination of hotels. First, the model is defined as a Markov Decision Process:

- We have a finite number of cities required to make any trip plan and are considered to be our **States**.
- There are the finite possible hotels for each city and are therefore the **Actions** of each state.
- Our preferences become the **Individual Rewards** for selecting each hotel.

In this Monte Carlo learning method, we consider the quality of each step towards reaching an end goal and, in order to assess the quality of any step, we must wait and see the outcome of the whole combination.

Monte Carlo is often avoided due to the time required to go through the whole process

before being able to learn. However, in our problem it is required as our final check when establishing whether the combination of the selected hotels is good or bad and then adding up the real cost of those selected and check whether or not this total cost is below or above our budget. Furthermore, for the examples of this project, we will not be considering more than a few cities and so the time taken is not significant in this regard.

2.2.1 Model

The first criterion is that whether the combination of our selected hotels is below our budget or not. The outcome defines the Reward of our selection. For example, if we have a budget of 250 EUR, then the choice:

$$a1 \rightarrow b1 \rightarrow c1 \rightarrow d1 \tag{1}$$

and if assume that the real cost of this selection is:

$$80 + 60 + 50 + 30 = 220 < 250 \tag{2}$$

then, our reward is defined as:

$$R_T = +1 \tag{3}$$

On the other hand, if we have a different choice such as

$$a2 \rightarrow b2 \rightarrow c2 \rightarrow d1 \tag{4}$$

and the real cost of this selection is given by

$$110 + 90 + 40 + 30 = 270 > 250 \tag{5}$$

Therefore, our reward is:

$$R_T = -1 \tag{6}$$

In fact, in our model this reward implies whether the choice is good or bad.

2.2.2 Model Learning

The way that our model learns is to try out lots of combinations of hotels and, at the end of each choice, determines whether the choice is good or bad. Over time, it will recognise that some hotels generally lead to getting a good outcome while others do not. We denote $V(a)$ as the measure of goodness of each hotel. We first introduce an initial value for V

for each hotel. Then, we follow an Update Rule to update these initial values as

$$V(a) \leftarrow V(a) + \alpha(R_T - V(a)) \tag{7}$$

where R_T is the reward (+1 or -1 accordingly) and α is the Learning Rate which determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing, while a factor of 1 makes the agent consider only the most recent information. With this update rule, we are simply updating the value of any action, $V(a)$, by an amount that is either a little more if the outcome was good or a little less if the outcome was bad.

At each step we can follow two possible selection processes. The first choice is a random selection of the hotels, each combination is known as an episode. However, using a completely random selection process may cause that some actions are not selected often enough to know whether they are good or bad. Similarly, if we went to other way and decided to select the hotels greedily, i.e. to ones that currently have the best value, we may miss one that is in fact better but never given a chance. For example, if we chose the first action, we would get a1, b1, c1 and d1, providing a positive reward. Therefore, if we use a purely greedy selection process, we would never consider any other hotels as these continue to provide a positive outcome. Instead, we implement epsilon-greedy action selection where we randomly select hotels with probability ϵ, and greedily select hotels with probability $1 - \epsilon$ where:

$$0 \leq \epsilon \leq 1$$

This means that we reach the optimal choice of hotels as quickly as we continue to detect the good hotels while we also explore other hotels occasionally just to make sure that they are not as good as our current choice. We set $\alpha = 0.5$ and $\epsilon = 0.1$ in the calculations. A sensitivity analysis can also be performed to evaluate the effect of these parameters on the results. For the sake of brevity, we do not show the results in this project and only mention that the selected parameters lead to the best results among other chosen parameters.

2.2.3 Introducing Preferences

To include any personal preferences on choosing the hotels, we can simply introduce preference rewards for each hotel whilst still having the reward that encourages the model to be below our budget. This can done by changing the calculation for the total reward

to:

$$R_T \leftarrow R_T + PR \tag{8}$$

where PR is the preference award. However, to ensure that we reach the primary goal of being below the budget, we take the average of the sum of the rewards for each action so that this will always be less than 1 or -1 respectively.

2.2.4 Monte Carlo Control

Similar to dynamic programming (DP), once we update the value function, the important task that still remains is that of finding the optimal choice using the Monte Carlo method. The formula for choice improvement required the model of the environment as shown in the following equation:

$$\pi'(s) = \arg\max_a \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right]$$

This equation finds out the optimal choice by finding actions that maximise the sum of rewards. Here, s and a represent the state and action, respectively.

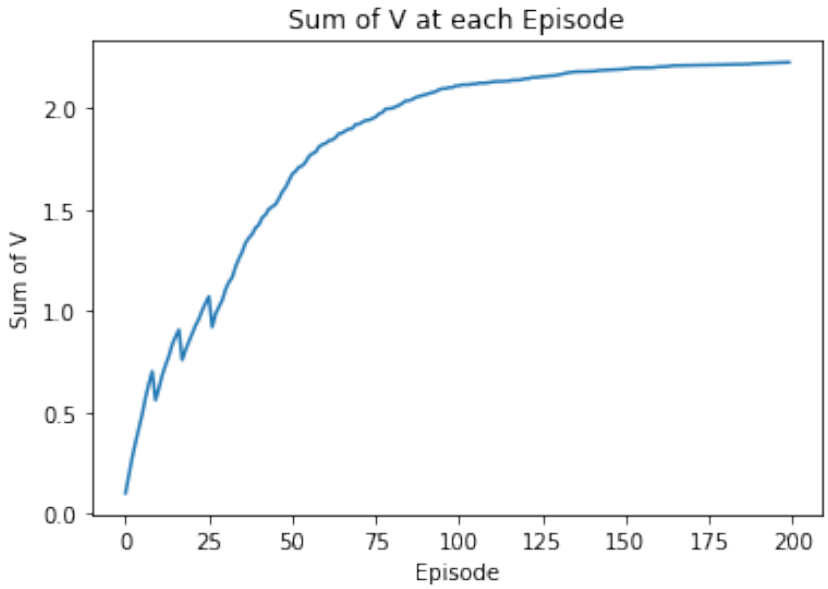
3 Results

To show the convergence of the proposed model, we can plot the total value function V as a function of the number of episodes. It is obvious in Fig. 1 that the result converges to a value of around 2.2. The optimal choice is also below of our defined budget (See the Code section).

4 Conclusion

I have implemented a Monte Carlo Reinforcement Learning model to select a combination of hotels in the target cities for travel planning. This model results in a selection of hotels below a proposed budget and it can also incorporate a preference of hotels. The results show that the value function, which determines the outcome, converges as the number of choice combinations (episodes) increases.

Fig. 1: Sum of the value function V as a function of the number of episodes.



5 Bibliography

1. Kaelbling, L.P., Littman, M.L. and Moore, A.W. (1996) Reinforcement Learning: A Survey. Journal of Aritificial Intelligence Research 4:237-285.

2. Sutton, R.S. and Barto, A.G. (2000) Reinforcement Learning: An Introduction. MIT Press. `http://www.cs.ualberta.ca/sutton/book/ebook`

3. Bertsekas, D.P. and Tsitsiklis, J.N. (1996) Neuro-Dynamic Programming. Athena Scientific.

6 Code

Please find below a copy of of the implemented code in Python.

Code

March 31, 2019

In [145]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
```

In [146]:
```
data = pd.read_csv("HotelData.csv")
```

In [147]:
```
data
```

Out[147]:

	City	Hotel	QMerged_label	Real_Cost	V_0
0	1	1	11	100	0
1	1	2	12	60	0
2	2	1	21	80	0
3	2	2	22	110	0
4	3	1	31	30	0
5	3	2	32	70	0
6	4	1	41	80	0
7	4	2	42	50	0
8	4	3	43	40	0

In [259]:
```
def MonteCarloModel(data, alpha, e, epsilon, budget, reward):
    # Define the States
    Cities = list(set(data['City']))
    # Initialise V_0
    V0 = data['V_0']
    data['V'] = V0
    output = []
    output1 = []
    output2 = []
    actioninfull = []
    #Interate over the number of episodes specified
    for e in range(0,e):

        episode_run = []
        #Introduce epsilon-greedy selection, we randomly select the first
        #episode as V_0(a) = 0 for all actions
        epsilon = epsilon
        if e == 0:
```

```
    for i in range(0,len(Cities)):
        episode_run = np.append(episode_run,np.random.random_integers\
                        (low = 1, high = sum(1 for p in data.iloc[:, 0]\
                                             if p == i+1 ), size = None))
    episode_run = episode_run.astype(int)

else:
    for i in range(0,len(Cities)):
        greedyselection = np.random.random_integers(low = 1, high =10)
        if greedyselection <= (epsilon)*10:
            episode_run = np.append(episode_run,np.random.random_integers\
                            (low = 1, high = sum(1 for p in data.iloc[:, 0]\
                                                 if p == i+1),size = None))
        else:
            data_I = data[data['City'] == (i+1)]
            MaxofVforI = data_I[data_I['V'] == data_I['V'].max() ]['Hotel']
            #If multiple max values, take first
            MaxofVforI = MaxofVforI.values[0]
            episode_run = np.append(episode_run, MaxofVforI)

        episode_run = episode_run.astype(int)



episode = pd.DataFrame({'City' : Cities, 'Hotel': episode_run})
episode['Merged_label'] =  (episode['City']*10 +\
                            episode['Hotel']).astype(float)
data['QMerged_label'] = (data['QMerged_label']).astype(float)
data['Reward'] = reward
episode2 =  episode.merge(data[['QMerged_label','Real_Cost','Reward']],\
                left_on='Merged_label',right_on='QMerged_label',how='inner')
data = data.drop('Reward',1)

# Calculate our reward including preference reward
if(budget >= episode2['Real_Cost'].sum()):
    Return = 1 + (episode2['Reward'].sum())/len(Cities)
else:
    Return = -1 + (episode2['Reward'].sum())/len(Cities)



episode2 = episode2.drop('Reward',1)
episode2['Return'] = Return

# Apply update rule to actions that were
#involved in obtaining reward
data = data.merge(episode2[['Merged_label','Return']], \
            left_on='QMerged_label',right_on='Merged_label',how='outer')
```

```
            data['Return'] = data['Return'].fillna(0)
            for v in range(0,len(data)):
                if data.iloc[v,7] == 0:
                    data.iloc[v,5] = data.iloc[v,5]
                else:
                    data.iloc[v,5]  = data.iloc[v,5]  + alpha*\
                    ( (data.iloc[v,7]/len(Cities)) - data.iloc[v,5] )

            # Output table
            data = data.drop('Merged_label',1)
            data = data.drop('Return',1)

            # Output is the Sum of V(a) for all episodes
            output  = np.append(output, data.iloc[:,-1].sum())

            # Output 1 and 2 are the Sum of V(a) for for the
            #cheapest actions and rest respectively
            # I did this so we can copare how they converge
            #whilst applying to such a small sample problem
            output1 = np.append(output1, data.iloc[[1,2,4,8],-1].sum())
            output2 = np.append(output2, data.iloc[[0,3,5,6,7],-1].sum())

            # Ouput to optimal action from the model based on highest V(a)
            action = pd.DataFrame(data.groupby('City')['V'].max())
            action2 = action.merge(data, left_on = 'V',right_on = 'V',how='inner')
            action3 = action2[['City','Hotel']]
            action3 = action3.groupby('City')['Hotel'].apply\
            (lambda x :x.iloc[np.random.randint(0, len(x))])

            # Output the optimal action at each episode so
            #we can see how this changes over time
            actioninfull = np.append(actioninfull, action3)
            actioninfull = actioninfull.astype(int)

            # Rename for clarity
            SumofV = output
            SumofVForCheapest = output1
            SumofVForExpensive = output2
            OptimalActions = action3
            ActionsSelectedinTime = actioninfull

        return(SumofV, SumofVForCheapest, SumofVForExpensive,\
               OptimalActions, data, ActionsSelectedinTime)
```

In [260]:
```
alpha = 0.1
num_episodes = 200
epsilon = 0.5
budget = 350
```

```
        # Setting a non-zero reward (weight) for our Hotel preferences
        reward = [0,0,0,0,0,0,0,0,0]

        start_time = time.time()

        Mdl = MonteCarloModel(data=data, alpha = alpha, e = num_episodes,epsilon\
                              = epsilon, budget = budget, reward = reward)

        print("--- %s seconds ---" % (time.time() - start_time))
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: DeprecationWarning: This funct
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: DeprecationWarning: This funct
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:25: DeprecationWarning: This funct
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:27: DeprecationWarning: This funct
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:27: DeprecationWarning: This funct


--- 4.531093120574951 seconds ---
```

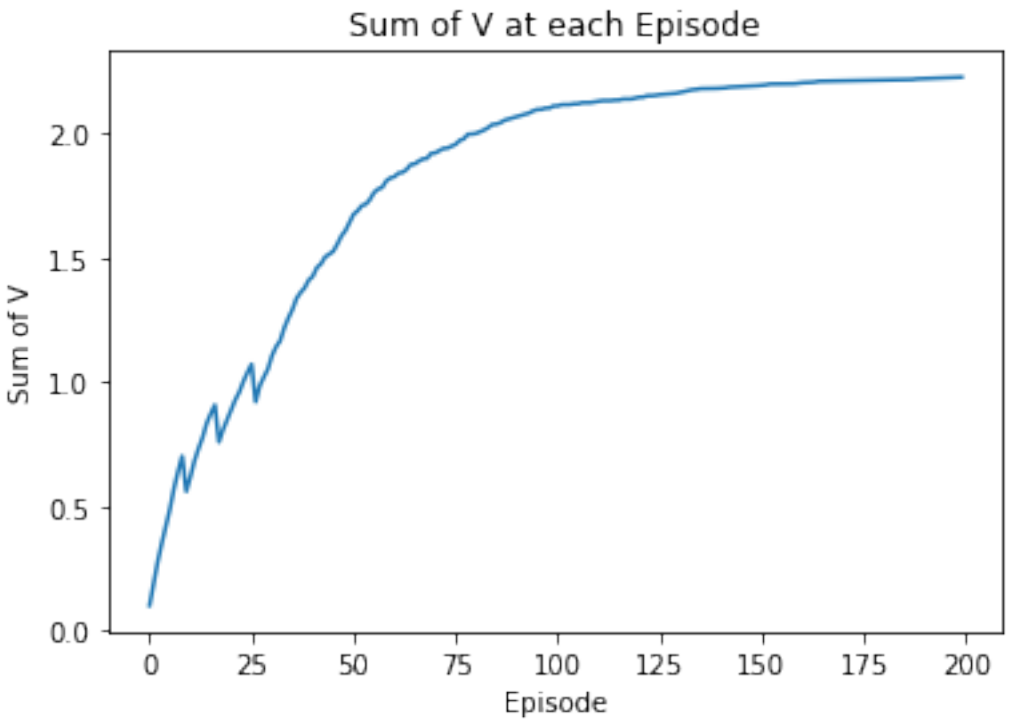
```
In [256]: print(Mdl[3])
          Mdl[4]
```

```
City
1    2
2    2
3    1
4    2
Name: Hotel, dtype: int64
```

Out[256]:

	City	Hotel	QMerged_label	Real_Cost	V_0	V
0	1	1	11.0	100	0	0.248982
1	1	2	12.0	60	0	0.250000
2	2	1	21.0	80	0	0.248233
3	2	2	22.0	110	0	0.250000
4	3	1	31.0	30	0	0.250000
5	3	2	32.0	70	0	0.247647
6	4	1	41.0	80	0	0.245380
7	4	2	42.0	50	0	0.249999
8	4	3	43.0	40	0	0.236916

```
In [257]: plt.plot(range(0,num_episodes), Mdl[0])
          plt.title('Sum of V at each Episode')
          plt.xlabel('Episode')
          plt.ylabel('Sum of V')
          plt.show()
```


Sum of V at each Episode
Sum of V
Episode
0.0
0.5
1.0
1.5
2.0
0
25
50
75
100
125
150
175
200