Assignment5_Reinforcement_Learning_update

December 6, 2022

#DAT405 Introduction to Data Science and AI - Shivneshwar Velayutham ##2022-2023, Reading Period 1 ## Assignment 5: Reinforcement learning and classification

The exercise takes place in a notebook environment where you can chose to use Jupyter or Google Colabs. We recommend you use Google Colabs as it will facilitate remote group-work and makes the assignment less technical. Hints: You can execute certain linux shell commands by prefixing the command with `!`. You can insert Markdown cells and code cells. The first you can use for documenting and explaining your results the second you can use writing code snippets that execute the tasks required.

This assignment is about **sequential decision making** under uncertainty (Reinforcement learning). In a sequential decision process, the process jumps between different states (the environment), and in each state the decision maker, or agent, chooses among a set of actions. Given the state and the chosen action, the process jumps to a new state. At each jump the decision maker receives a reward, and the objective is to find a sequence of decisions (or an optimal policy) that maximizes the accumulated rewards.

We will use **Markov decision processes** (MDPs) to model the environment, and below is a primer on the relevant background theory. The assignment can be divided in two parts:

0.1 Question 1

The first question covers a deterministic MPD, where the action is directly given by the state, described as follows:

- The agent starts in state **S** (see table below)
- The actions possible are **N** (north), **S** (south), **E** (east), and **W** west.
- The transition probabilities in each box are deterministic (for example P(s'|s,N)=1 if s' north of s). Note, however, that you cannot move outside the grid, thus all actions are not available in every box.
- When reaching **F**, the game ends (absorbing state).
- The numbers in the boxes represent the rewards you receive when moving into that box.
- Assume no discount in this model: $\gamma = 1$

-1	1	**F**
0	-1	1
-1	0	-1
S	-1	1

Let (x, y) denote the position in the grid, such that $S = (0, 0)$ and $F = (2, 3)$.

1a) What is the optimal path of the MDP above? Is it unique? Submit the path as a single string of directions. E.g. NESW will make a circle.

Ans: One of the optimal paths is EENNN. The optimal path is not unique. Another optimal path can be EENNWNE.

1b) What is the optimal policy (i.e. the optimal action in each state)? It is helpful if you draw the arrows/letters in the grid.

Ans: The optimal action is indicated by the direction of the arrow next to each state.

-1 >	1 >	**F**
0 >	-1 >	1 ^
-1 >	0 >	-1 ^
S >	-1 >	1 ^

1c) What is expected total reward for the policy in 1a)?

Ans: The total reward is 0.

0.2 Value Iteration

0.3 Question 2

2a) Code the value iteration algorithm just described here, and show the converging optimal value function and the optimal policy for the above 3x3 grid.

Ans: Code and converging values below.

2b) Explain why the result of 2a) does not depend on the initial value V_0.

Ans: It does not depend on the initial value because we stop iterating only when the value function does not change much (Recall that for a policy to be optimal, it must satisfy the Bellman equation above, meaning that plugging in a given candidate V^* in the right-hand side (RHS) of the Bellman equation should result in the same V^* on the left-hand side (LHS)). So it's more about the changes/difference in value function based on the rewards than the initial value function V_0.

[9]:
```python
from copy import deepcopy

epsilon = 0.1
gamma = 0.9
flag = True
reward = [[0, 0, 0],[0, 10, 0], [0, 0, 0]]
value = [[0, 0, 0],[0, 0, 0], [0, 0, 0]]
steps = {'N': 0, 'S': 0, 'E': 0, 'W': 0}
policy = [['', '', ''],['', '', ''], ['', '', '']]

while flag:
    temp = deepcopy(value)
    for i in range(3):
        for j in range(3):
            for step in steps:
                if step == 'N':
                    if i == 0:
                        steps[step] = 0
                    else:
                        steps[step] = (0.8*(reward[i-1][j] +␣
 ↪gamma*value[i-1][j])) + (0.2*(reward[i][j] + gamma*value[i][j]))
                if step == 'S':
                    if i == 2:
                        steps[step] = 0
                    else:
                        steps[step] = (0.8*(reward[i+1][j] +␣
 ↪gamma*value[i+1][j])) + (0.2*(reward[i][j] + gamma*value[i][j]))
                if step == 'E':
                    if j == 2:
                        steps[step] = 0
                    else:
                        steps[step] = (0.8*(reward[i][j+1] +␣
 ↪gamma*value[i][j+1])) + (0.2*(reward[i][j] + gamma*value[i][j]))
                if step == 'W':
                    if j == 0:
                        steps[step] = 0
                    else:
                        steps[step] = (0.8*(reward[i][j-1] +␣
 ↪gamma*value[i][j-1])) + (0.2*(reward[i][j] + gamma*value[i][j]))
            temp[i][j] = max(steps.values())
            next_step = [key for key in steps if steps[key] == temp[i][j]]
            policy[i][j] = next_step[0]

    stop = True
    for i in range(3):
        for j in range(3):
            if abs(temp[i][j] - value[i][j]) > epsilon:
                stop = False
    if stop:
        flag = False

    value = temp

print('Optimal value function below')
for i in range(3):
    print(value[i])

print('\nOptimal policy below')
for i in range(3):
    print(policy[i])
```

```
```

```
Optimal value function below
[44.791763498219886, 51.12689178471024, 44.791763498219886]
[51.12689178471024, 47.23078788846379, 51.12689178471024]
[44.791763498219886, 51.12689178471024, 44.791763498219886]

Optimal policy below
['S', 'S', 'S']
['E', 'N', 'W']
['N', 'N', 'N']
```

0.4 Reinforcement Learning (RL)

0.5 Question 3

You are to first familiarize with the framework of the OpenAI environments, and then implement the Q-learning algorithm for the NChain-v0 enviroment depicted above, using default parameters and a learning rate of $\gamma = 0.95$. Report the final Q^* table after convergence of the algorithm. For an example on how to do this, you can refer to the Q-learning of the **Frozen lake environment** (q_learning_frozen_lake.ipynb), uploaded on Canvas. Hint: start with a small learning rate.

Note that the NChain environment is not available among the standard environments, you need to load the gym_toytext package, in addition to the standard gym:

!pip install gym-legacy-toytext import gym import gym_toytext env = gym.make("NChain-v0")

[14]:
```
import random
import numpy as np
import pandas as pd
import gym
import gym_toytext
```

[11]:
```
env = gym.make("NChain-v0")
env.reset()

num_episodes = 25000
gamma = 0.95
learning_rate = 0.1
epsilon = 0.5

Q = np.zeros([5, 2])
```

[12]:
```
for _ in range(num_episodes):
        state = env.reset()
        done = False
        while done == False:
                if random.uniform(0, 1) < epsilon:
                        action = env.action_space.sample()
                else:
                        action = np.argmax(Q[state,:])
                new_state, reward, done, info = env.step(action)
                update = reward + (gamma*np.max(Q[new_state,:])) - Q[state,␣
↪action]
                Q[state,action] += learning_rate*update
                state = new_state
```

[19]:
```
df = pd.DataFrame(Q, columns=['Action A', 'Action B'])
df.index = np.arange(1, len(df) + 1)
print(df)
```

```
    Action A   Action B
1  60.813210  59.911587
2  64.371282  60.617006
3  69.599451  62.175039
4  73.659266  63.484101
5  79.573522  63.641493
```

0.6 Question 4

4a) What is the importance of exploration in RL? Explain with an example.

Ans: Exploration is the only way for reinforcement algorithms to learn and get more information about unknown parts of the environment. If there was no exploration then the algorithm will only optimize on the data it already has but with exploration it can learn more information and get better results. An example showing the importance of exploration is when one wants to know the current state of affairs in the world it might be good to read the news from new sources instead of just reading the news from the sources that one normally uses.

4b) Explain what makes reinforcement learning different from supervised learning tasks such as regression or classification.

Ans: In supervised learning decisions are made using predefined set of data which hopefully represent a full picture of the world, but in reinforcement learning, decisions are made without knowing the entire picture of the world. Learning is done based on the result of each decision/exploration taken and its used in finding the optimal decisions to be made in each state.

0.7 Question 5

5a) Give a summary of how a decision tree works and how it extends to random forests.

Ans: A decision tree helps with classification of data. The tree is built using the trends and correlations between the features and classes. And when it's time to predict, the new data is tested on the decision tree by starting from the root of the tree and taking the path based on the data and the various questions being asked in each node of the tree and finding the next node based the answer to the question. Random forests is a set of decision trees on the same set of data but multiple decision trees are created by selecting only a subset of the features (chosen randomly) and building decison trees.

5b) State at least one advantage and one drawback with using random forests over decision trees.

Ans: Random forests help overcome the overfitting problem of decison trees since only a subset of the features are taken for each decision tree of the random forrest and thus reducing the overfitting. Random forests can also handle missing data for certain number of features. Drawback of random forest is that it can be slow especially when the number of the features is quite high. This is because, during prediction each decison tree must do a prediction based on the data and this needs to be cumulated before making the final decision for classification.