# DAT405 Assignment 5 – Group 53

Venkata Sai Dinesh Uddagiri - (20 hrs)
Madumitha Venkatesan - (20 hrs)

December 6, 2022

## Problem 1

**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.**

The optimal path for agent to reach from S(0,0) to F(2,3) with shortest number of steps and maximum reward score is EENNN. The total reward score of agent taken path EENNN is zero. The path that we determine is unique because all the other paths that we take to reach destination gives the reward score less than zero expect one path (EENNWNE), but optimal path should be the path with shortest disatnce and maximum reward score.

**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.**

The optimal policy is the policy where agent always takes the action to maximizes the reward. The best optimal policy for problem is, do not visit the state that already visited, if the reward in all the directions is same then go east and if the all the directions have different reward value then take the direction with maximum value.

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

The expected total reward for policy described above for path EENNN is zero. The total reward value is obtained by following calculation

$$V^{\pi}(s) = \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t, s_{t+1}) p(s_{t+1}|s_t, a_t)$$

The transition probabilities of each action is deterministic so, we are considering the probability of each action as one. By this above equation changes as follows

$$V^{\pi}(s) = \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t, s_{t+1})$$

The discount in the model, $\gamma = 1$

$$V^{\pi}(s) = \sum_{t=1}^{T} r(s_t, a_t, s_{t+1})$$

$$V^{\pi}(s) = (-1) + 1 + (-1) + 1 + 0 = 0$$

# Problem 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.**

```
import copy
def value_iteration(rm,vm,pm):
    # Creating the copy of old values
    vm_old=copy.deepcopy(vm)
    for r in range(3):
        for c in range(3):
            max_value = 0
            # Intializing moves applicable
            actions=["N","S","E","W"]
            # restricting moving south if row index equals zero
            if r == 0:
                actions.remove("S")
            # restricting moving west if coloumn index equals zero
            if c == 0:
                actions.remove("W")
            # restricting moving north if row index equals two
            if r == 2:
                actions.remove("N")
            # restricting moving East if coloumn index equals two
            if c == 2:
                actions.remove("E")

            for action in actions:
                # Defining move to be taken for each particular action applicable
                if action=="N":
                    state = [r+1,c]
                elif action=="S":
                    state = [r-1,c]
                elif action=="E":
                    state = [r,c+1]
                elif action=="W":
                    state = [r,c-1]
                # Calculate the value of taking action in given state using Bellman
    equation
                new_val = 0.8*(rm[state[0]][state[1]] + gamma * vm_old[state[0]][state
    [1]]) + 0.2*(rm[r][c] + gamma * vm_old[r][c])
                # If the value is highest
                if (new_val >= max_value):
                   new_p = action
                   max_value = new_val
            # Save the value to the value matrix
            vm[r][c] = round(max_value,2)
            # Save the action to the policy matrix
            pm[r][c] = new_p

    for r in range(3):
        for c in range (3):
            # Checking the diffrence between the current and old iteration values, if
    diffrence is less than bellman factor, Then we can say model converges
            if (abs(vm_old[r][c] - vm[r][c]) >= eps):
                value_iteration(rm, vm, pm)
    return pm,vm
# Reward Matrix
rm=[[0,0,0], [0,10,0], [0,0,0]]
# Value matrix intialization (All the states are intialized with zero)
vm=[[10,10,10], [10,10,10], [10,10,10]]
# Policy matrix intialization
pm=[["","",""],["","",""],["","",""]]
```

```
# discount factor
gamma = 0.9
# Bellman factor
eps = 0.01
# Calling value iteration function
policy,vm=value_iteration(rm,vm,pm)
print("Value matrix:")
for v in vm:
    print(v)
print("\n")
print("Policy matrix:")
for pi in policy:
    print(pi)
```

Listing 1: Value iteration algorithm code

**Output :**

Value matrix:

[45.6, 51.94, 45.6]

[51.94, 48.04, 51.94]

[45.6, 51.94, 45.6]

Policy matrix:

['E', 'N', 'W']

['E', 'W', 'W']

['E', 'S', 'W']

**2b) Explain why the result of 2a) does not depend on the initial value V0 .**

The outcome of 2a is independent of initial V0 because, even if initial V0 is changed, the problem will still converge to the same optimal strategy and value function. Any initial value function V0 will eventually result in the value V that specifies the Bellman equation, therefore the only difference that will exist is the number of convergence iterations required, which will either increase or decrease. We will therefore find the same optimal value regardless of where we start if we have an algorithm that converges to the optimal solution.

## Problem 3

**You are to first familiarize with the framework of the OpenAI environments, and then implement the Q-learning algorithm for the NChain-v0 environment depicted above, using default parameters and a learning rate of =0.95 . Report the final Q /* table after convergence of the algorithm..**

```
import gym
import gym_toytext
import random
import numpy as np
import math
# Intialize the NChain environment
env = gym.make('NChain-v0')
# Total number of episodes
total_episodes = 15000
# Discount factor
gamma = 0.95
# Learning rate
lr = 0.1

eps = 0.5
# Intializing q table to values 0
Q = np.zeros((env.observation_space.n,env.action_space.n))
for _ in range(total_episodes):
  state = env.reset()
  done = False
  while done == False:
    # Select an action
    if random.uniform(0, 1) < eps:
      # Explore action space
      action = env.action_space.sample()
    else:
      # Exploit learned values
      action = np.argmax(Q[state,:])
    new_state, reward, done, info = env.step(action)
    # Action performed and received the feedback from the environment
    updateQ = reward + (gamma*np.max(Q[new_state,:])) - Q[state, action]
    # Finally we learn from the experience by updating the Q-value of the selected
    action
    Q[state,action] += lr*updateQ
    state = new_state

print("Final Q table after convergence")
print(Q)
```

Listing 2: Q-learning algorithm for the NChain-v0 enviroment code

**Output :**

Final Q table after convergence

[[62.05726232 60.00849232]

[67.1195741 61.51204546]

[72.76871824 61.69483596]

[78.14702798 64.39930786]

[82.85316868 69.30886498]]

# Problem 4

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

The agent can learn about the many states, actions for each state, associated rewards, and transition to the next state by exploring the environment. This is why exploration is important in reinforcement learning. For instance, a robot and I are playing chess. Everyone must participate fully in order to win the game. A player has two options: he can either make the move he thinks is best, or he can try something new. I am playing the best move I can, however fresh moves might be more strategically sound to win this game. Here, the first option is known as exploitation, where I am aware of my game plan, while the second option is known as exploration, where I am expanding my knowledge and using a fresh move to win the game.

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

The first and most fundamental difference between supervised learning and reinforcement learning is that supervised learning has two tasks: classification and regression; reinforcement learning has a variety of tasks, including exploitation or exploration, Markov's decision-making processes, policy learning, deep learning, and value learning. An extensive set of labeled data must be used to train supervised learning. Reinforcement learning, on the other hand, develops on its own through trial and error and input from the environment. In supervised learning, the training data are analyzed to create a generalized formula; in reinforcement learning, the Markov Decision Process model defines the basic reinforcement.

# Problem 5

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

Both the decision tree and random forests are algorithm used for classification and regression problems.

Decision tree is named out of the plot as it looks like tree along with a decision nodes. In the decision tree the node from where the decision tree starts is root node. The node that gives the final output is called leaf node. Every node in the tree represents the question, based on the answer to that question branch lead to new node. The questions will get more specific as the tree gets deeper.

Decision tree works in the following way:

The root node, which contains the entire data set, is where the tree starts. Find the optimal quality that can be applied at the root node to guide decision-making using the data set. To the greatest extent possible, determine the subsets of attributes for child nodes. The method then continues to identify the subset properties for each child node in order to make decisions up until the point where further classification of the nodes is not possible. Leaf nodes are the final nodes. If we have inputted the data to root node based on answering the question at each node reaches to one leaf node, which is output for that particular input.

Overfitting is one issue that came up when utilizing the decision tree algorithm. As decision tree algorithm uses single tree, single trees frequently perform poorly when making predictions based on new data because they learn the training data too thoroughly. But the random forest algorithm can be used to fix this.

Random Forest uses many decision trees on different subsets of the input dataset and averages the results to increase the dataset's predicted accuracy. Higher accuracy is obtained and overfitting is avoided because to the larger number of decision trees. Instead of allowing a single decision tree to categorize the newly discovered data, all trees participate and vote on the best classification strategy. The data will be categorized into the class that receives the most votes. When compared to employing a single decision tree for the categorization, this is proven to significantly increase accuracy.

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

The advantage of using the random forest algorithm is, it increases the accuracy of the model and prevents the overfitting issue. One of the disadvantage is, to increase the accuracy of the model, large number of decision trees are required, which makes the algorithm slow and as number of decision trees increases the computation power required also high.