Project 2: Reinforcement Learning

Paul-Emile Giacomelli pegiaco@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Descriptions

I used Q-learning for all datasets. The hyperparameters I used were the following:

- number of episodes: 1000
- learning rate: 0.1
- ε: 0.1

I first chose 100 episodes but to make my results more solid I chose to switched to 1000 episodes.
The training times are listed in the following subsections.

1.1 Small Data Set

For the small dataset, there are 100 states and 4 possible actions. I used a discount factor of 0.95 as suggested. Training time:

- 832.0934066772461 seconds (1000 episodes)
- 0.8320934066772461 seconds/episode

1.2 Medium Data Set

For the medium dataset, there are 50,000 states and 7 possible actions. As this problem is discounted, I used a discount factor of 1.00. Training time:

- 1702.767900943756 seconds (1000 episodes)
- 0.1702767900943756 seconds/episode

1.3 Large Data Set

For the large dataset, there are 312020 states and 9 possible actions. I used a discount factor of 0.95 as suggested. Training time:

- 1767.0224149227142 seconds (1000 episodes)
- 0.17670224149227142 seconds/episode

2. Code

```
import numpy as np
import pandas as pd
from tqdm import tqdm
import time

# Create a Q-learning agent
class QLearningAgent:
    def __init__(self, num_states, num_actions, learning_rate,
    discount_factor, epsilon):
        self.num_states = num_states
        self.num_actions = num_actions
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.epsilon = epsilon
        self.q_table = np.zeros((num_states, num_actions))

    def select_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(self.num_actions)
        return np.argmax(self.q_table[state, :])

    def update_q_table(self, state, action, reward, next_state):
        predict = self.q_table[state, action]
        target = reward + self.discount_factor * np.max(self.q_table[
    next_state, :])
        self.q_table[state, action] += self.learning_rate * (target - predict
    )


def run_q_learning(dataset_file, num_states, num_actions, learning_rate,
    discount_factor, epsilon, num_episodes):
    start_time = time.time()  # Record the start time

    # Read transition data from the provided dataset file
    data = pd.read_csv(f"./data/{dataset_file}")

    # Initialize the Q-learning agent
    agent = QLearningAgent(num_states, num_actions, learning_rate,
    discount_factor, epsilon)

    # Q-learning algorithm with a progress bar for the specified number of
    episodes
    for episode in tqdm(range(num_episodes), desc=f"Q-Learning Progress - {
    dataset_file.split('.')[0]} dataset"):
        for _, row in data.iterrows():
            current_state = int(row['s']) - 1  # Adjust to 0-based indexing
            action = int(row['a']) - 1  # Adjust to 0-based indexing
```

```
            # Ensure that the action is within the valid range (0 to
    num_actions-1)
            action = max(0, min(action, num_actions - 1))
            reward = float(row['r'])
            next_state = int(row['sp']) - 1  # Adjust to 0-based indexing

            agent.update_q_table(current_state, action, reward, next_state)

    end_time = time.time()  # Record the end time
    training_time = end_time - start_time

    # Save the optimal policy as a .policy file without column names with 1-
    based indexing
    optimal_policy = [agent.select_action(state) + 1 for state in range(
    num_states)]
    output_filename = f"{dataset_file.split('.')[0]}.policy"
    pd.DataFrame({'Action': optimal_policy}).to_csv(f"./output/{
    output_filename}", index=False, header=False)

    print(f"Optimal policy saved to ./output/{output_filename}")

    # Write dataset name and training time to a file
    with open("./output/training_times.txt", "a") as time_file:
        time_file.write(f"{dataset_file}: {training_time} seconds\n")



if __name__ == "__main__":
    num_episodes = 1000

    # Clear the training_times.txt file
    with open("./output/training_times.txt", "w") as time_file:
        time_file.write(f"num_episodes: {num_episodes}\n")

    run_q_learning("small.csv", num_states=100, num_actions=4, learning_rate
    =0.1, discount_factor=0.95, epsilon=0.1, num_episodes=num_episodes)

    run_q_learning("medium.csv", num_states=50000, num_actions=7,
    learning_rate=0.1, discount_factor=1.0, epsilon=0.1, num_episodes=
    num_episodes)

    run_q_learning("large.csv", num_states=312020, num_actions=9,
    learning_rate=0.1, discount_factor=0.95, epsilon=0.1, num_episodes=
    num_episodes)
```