Project 2: Reinforcement Learning

Christina Knight cqknight@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Descriptions

For all three of the datasets, I used the Q-learning algorithm. This model-free method incrementally estimates the action value function (Q(s,a)). First, I create a "Q matrix" of states and actions. Then, I incrementally update the Q matrix by going through each row of the data and following the update rule derived from the Bellman equation. I run through this process 100 times, but I could have ran until convergence, which would have given me a more optimal policy. Finally, I take the best action for each state and create the policy. For my hyperparameters, I used 0.1 for step size and the discount provided in the instructions. My run-times are listed below.

1.1 Small Data Set

Runtime: 36.67922282218933 seconds

1.2 Medium Data Set

Runtime: 74.34561491012573 seconds

1.3 Large Data Set

Runtime: 79.23135423660278 seconds

2. Code

```
import sys
import numpy as np
import pandas as pd
import time

def compute(infile, outputfilename):
    file = open(infile)
    type(file)
    D = pd.read_csv(infile)
    D = D.to_numpy()
    file.close()
    if infile == "small.csv":
        num_states = 100
        num_actions = 4
        discount = 0.95
    elif infile == "medium.csv":
```

```
        num_states = 50000
        num_actions = 7
        discount = 1
    else:
        num_states = 312020
        num_actions = 9
        discount = 0.95
    q_matrix = np.zeros((num_states, num_actions))
    policy = q_learning(q_matrix, D, discount)
    with open(outputfilename, 'w') as f:
        for s in range(len(policy)):
            f.write(str(policy[s]) + "\n")
    return f

def q_learning(q_matrix, data, discount):
    policy = []

    for i in range(100):
        for row in data:
            s = row[0]
            a = row[1]
            r = row[2]
            sp = row[3]
            q_matrix = update(q_matrix, s - 1, a - 1, r, sp - 1, discount)

    print(q_matrix)

    for row in range(q_matrix.shape[0]):
        policy.append(np.argmax(q_matrix[row]) + 1)

    print(q_matrix.shape)
    print(policy)
    return policy

def update(Q, s, a, r , sp, discount):
    Q[s, a] += 0.1 * (r + discount * np.max(Q[sp,:]) - Q[s, a])
    return Q

def main():
    if len(sys.argv) != 3:
        raise Exception("usage: python project1.py <infile>.csv")
    inputfilename = sys.argv[1]
    outputfilename = sys.argv[2]
    start_time = time.time()
    compute(inputfilename, outputfilename)
    print("--- %s seconds ---" % (time.time() - start_time))

if __name__ == '__main__':
    main()
```