Project 2: Reinforcement Learning

Akshar Sarvesh asarvesh@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Descriptions

For all of these datasets, I was able to implement a fairly simple Q-Learning algorithm to read in all of the datapoints and iteratively value each action at all of the states. The gammas were set based on the problem statement: 0.95, 1.00, and 0.95, respectively. There were some other quirks for the different datasets, which I'll get into below. The run time overall was less than 2 minutes.

1.1 Small Data Set

With only 100 states, 4 actions, and 50,000 datapoints, we easily had enough data to try every action from every state multiple times. I submitted the output from my first trial of Q-learning on just the small dataset to validate the algorithm was working, which it was. The run time of this algorithm was about 15 seconds.

1.2 Medium Data Set

With 50,000 states, 9 actions, and 100,000 datapoints, we did not have enough data to explore every state/action combination. As it turns out, not every state was even explored in the data either; So I actually had a couple errors with the output file not being large enough. I then had to hardcode the state counts for each of the datasets, such that I could then create a file with a hardcoded size for each of them. If I had an optimal action for a given state, I'd use that, otherwise I had it set to select randomly.

While running this code, I was still getting below baseline for specifically medium. I looked at the file and noticed for nearly every state, it was choosing the optimal action to be 5. Intuitively, I thought that meant it was choosing for the car to not accelerate ever (5 is halfway between 1 and 9). I guessed that meant it simply wasn't training for long enough for utility to propagate through states farther away from the flag. I then increased the epochs to 500 and was able to successfully increase my score on medium to beat baseline (and also increase my score on large as well!)

The run time on this was about 30 seconds on the new epoch count. This makes sense to me because the run time is controlled by the number of epochs * the number of updates to make per epoch, which is the number of datapoints we have. Since this dataset has twice as many datapoints as the small dataset, it makes sense that our run time is about twice as long.

1.3 Large Data Set

Surprisingly, this was easier to work with than the medium dataset. After having the state counts hardcoded, and running the algorithm for 500 epochs, it was able to pick advantageous actions. This also took 30 seconds, which also makes sense because despite there being more states than the medium dataset, there were the same number of datapoints, meaning same overall number of updates and therefore the same amount of training time, approximately.

2. Code

```python
import pandas as pd
import numpy as np
from collections import defaultdict

# Hardcoding the state counts because not every state seen in all data
STATE_COUNTS = {
    "small": 100,
    "medium": 50000,
    "large": 302020,
}

def parseData(case, root="data"):
    df = pd.read_csv(f"{root}/{case}.csv")
    states  = sorted(set(df["s"]).union(df["sp"]))
    actions = sorted(df["a"].unique())
    return df, states, actions

def offline_q_learning(df, actions, gamma, alpha=0.1, epochs=500, init_q=0.0,
     seed=0):
    rng = np.random.default_rng(seed)
    Q = defaultdict(lambda: defaultdict(lambda: init_q))
    transitions = list(zip(df["s"].tolist(), df["a"].tolist(), df["r"].tolist
    (), df["sp"].tolist()))
    # For a set number of epochs, update every transition by the q-learning
    update formula
    for _ in range(epochs):
        rng.shuffle(transitions)
        for s, a, r, sp in transitions:
            max_next = max((Q[sp][ap] for ap in actions), default=0.0)
            target = r + gamma * max_next
            Q[s][a] += alpha * (target - Q[s][a])
    return Q

#Get best policy (after training)
def greedy_policy(Q, actions):
    pi = {}
    for s, Qa in Q.items():
        pi[s] = max(actions, key=lambda a: Qa[a])
    return pi

def write_policy_file(pi, num_states, filename, actions, seed=0):
    rng = np.random.default_rng(seed)
    with open(filename, "w") as f:
        for s in range(1, num_states + 1):
            action = pi.get(s)
            if action is None:
                action = int(rng.choice(actions))
            f.write(f"{action}\n")

def QLearning(case, root="data", alpha=0.1, epochs=500):
    gamma_map = {"small": 0.95, "medium": 1.0, "large": 0.95}
    gamma = gamma_map[case]

    df, states, actions = parseData(case, root=root)
    Q = offline_q_learning(df, actions, gamma=gamma, alpha=alpha, epochs=
    epochs)
    pi = greedy_policy(Q, actions)

    num_states = STATE_COUNTS[case]
    out_path = f"output/{case}.policy"
    seed = {"small": 1, "medium": 2, "large": 3}[case]
    write_policy_file(pi, num_states, out_path, actions, seed=seed)
    print(f"Saved policy to {out_path}")

if __name__ == "__main__":
    cases = ["small", "medium", "large"]
    for case in cases:
        QLearning(case, root="data")
```