Project 2: Reinforcement Learning

Alexandros Tzikas alextzik@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

The goal of this project is to determine the optimal policy for three different processes, constrained by the fact that exploration has already been performed. Given the sampled transitions of the form (s, a, r, s'), where we transition to state s' from state s, by taking action a and receiving a reward r, the goal is to calculate the optimal deterministic policy $\pi^*(s)$ that expresses the best action, in terms of total expected reward, to take from state s.

A single algorithm was implemented for all three (3) data-sets. Specifically, the Q-learning algorithm was used, as shown in Kochenderfer et al. (2022). The choice was motivated by the following reasons:

- The small data-set has a state space comprised of $|S| = 100$ states and an action space of $|A| = 4$ actions. However, the medium data-set has $|S| = 50,000$ and $|A| = 7$ and the large data-set has an even greater size with $|S| = 312,020$ and $|A| = 9$. If we were to use a model-based method, specifically a maximum-likelihood model, we would need to store the matrices $N(s, a, s')$ and $\rho(s, a)$. This means that we would need $O(|S|^2|A|)$ memory. This a great demand. We should also mention that we would need to use the aforementioned tables to calculate the approximate Markov Decision Process (MDP) and determine $T(s'|s, a)$ and $R(s, a)$, which would also have to be stored in memory. Indeed, when the maximum likelihood method was used, along with value iteration, the computation failed for the large dataset. On the other hand, Q-learning only requires the storage of $Q(s, a)$.

- Q-learning learns an action-value function $Q(s, a)$ that directly approximates the optimal action-value function Q^*, independent of the policy being followed for exploration (in our case exploration has already been performed), as discussed in Sutton and Barto (2018). All that is required for correct convergence is that all (s, a) pairs continue to be updated. Although this is a condition that we cannot guarantee given that we are provided with the exploration, the characteristics of Q-learning make it an appealing candidate for the implementation.

Hence, for each sampled transition, we perform an update, using Q-learning, of the corresponding element $Q(s, a)$ of the action-value function. We then move on to the next sampled transition. We perform a predetermined number of passes (k_{max}) over the entire data-set, which allows the approximated Q to approach the optimal Q^*. After the k_{max} iterations over the data-set, we compute the policy by choosing:

$$\pi(s) = \text{argmax}_a Q(s, a) \approx \text{argmax}_a Q^*(s, a), \ \forall s \in S. \tag{1}$$

We remind the reader that, given the optimal action-value function $Q^*(s,a)$, the optimal deterministic policy $\pi^*(s)$ is given by:

$$\pi^*(s) = \text{argmax}_a Q^*(s,a), \ \forall s \in S. \tag{2}$$

The implemented Q-learning algorithm performs the following update for every sampled transition:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \text{max}_{a'} Q(s',a') - Q(s,a)\right). \tag{3}$$

It is thus evident that for n sampled transitions in the data-set and k_{max} runs over the data-set, the computational complexity of eq. (3) that provides the table $Q(s,a)$ is $O\left(k_{max} n |A|\right)$. After obtaining $Q(s,a)$, we get the output policy by using eq. (1). Eq. (1) is of complexity $O(|S||A|)$. The algorithm also requires the storage of a table of dimensions $|S| \times |A|$.

For the project, we chose a sufficiently large number of runs over the data-set to allow Q to approximate Q^*. We use $k_{max} = 200$. We also chose the step size parameter to be $\alpha = 0.1$. This parameter expresses the impact of newly acquired information on the current $Q(s,a)$ estimate. If $\alpha = 0$, then new information is simply discarded. We choose a low $\alpha = 0.1$ value, since it performed better on the grader. A small α also helps with the convergence of the action-value function.

The running time of the algorithm for the three (3) data-sets is shown in the following table.

Table 1: Running Time of Algorithm for Each Dataset

Dataset	Running Time $[s]$
small.csv	19.759727
medium.csv	41.435670
large.csv	43.185170

2. Code

```
############################################################
"""
    Nesessary Packages
"""
# Printing
using Printf

# Dataset reading
using DataFrames
using Pkg
Pkg.add("CSV")
using CSV

############################################################
"""
    Change directory
```

```
"""
dir = "/Users/AlexandrosTzikas/Desktop/project 2"
cd(dir)

############################################################
"""
    Create an input dataframe and an output file
"""
infile = DataFrame(CSV.File("large.csv"))
outfile = "large.policy"

############################################################
"""
    Necessary functions
"""

# Policy Output File Creation
function write_policy(actions, filename)
    open(filename, "w") do io
        for a in actions
            @printf(io, "%s\n", a)
        end
    end
end

# Necessary structure
struct QLearning
    S # state space size
    A # action space size
    gamma # discount
    Q # action value function of size (|S|,|A|)
    a # learning rate
end

# update the Q estimates after with the observed transitions
function update!(model::QLearning, D)
    k_max = 200 # number of dataset passes

    # update Q-learning table with multiple passes over dataset
    for k in 1:k_max

        # Q-learning update step for each observation
        for i = 1:size(D,1)
            s = D[i, 1]
            a = D[i, 2]
            r = D[i, 3]
            sp = D[i, 4]

            model.Q[s,a] += model.a*(r + model.gamma*maximum(model.Q[sp,:])-
    model.Q[s,a])
```

```
        end
    end
    return model
end

function find_policy(model::QLearning)
    actions = zeros(model.S, 1)
    for s in 1:model.S
        u, a = findmax(a->model.Q[s,a], [i for i in 1:model.A])
        actions[s, 1] = a
    end
    return actions
end

############################################################
"""
    Main Function
"""
function compute(infile, outfile)
    # Input details - must change for each dataset!!!!
    nOfStates = 312020
    nOfActions = 9
    gamma = 0.95
    a = 0.1

    # Instantiate the model estimate using the Q-learning structure
    model = QLearning(nOfStates, nOfActions, gamma, zeros(nOfStates,
    nOfActions), a)

    # Update the model estimate
    model = update!(model, infile)

    # Find policy
    actions = find_policy(model)

    # Write policy to output file
    write_policy(actions, outfile)

end

@time begin
    compute(infile, outfile)
end
```

References

Mykel J. Kochenderfer, Tim A. Wheeler, and Kyle H. Wray. *Algorithms for Decision Making*. The MIT Press, Cambridge, Massachusetts, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, Cambridge, Massachusetts, 2018.