Project 2: Reinforcement Learning

Karthik Pythireddi karthik9@stanford.edu
AA228/CS238, Stanford University

1 Algorithm Descriptions

For the project we used the model free method of reinforcement learning. In this approach we don't need the explicit representations of transition and reward models. These model free methods model the action value function directly. Avoiding these explicit representations is very helpful when the problem is high dimensional. In these methods, we apply the incremental estimation of the mean of a distribution, which is very vital in the estimating the means of the returns. There are many model free methods, for the project we made use of the *Q-learning.*

Q-learning involves applying the incremental estimation of the action value function *Q(s,a).* The update equation is derived from the action value form of the bellman equation.

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q(s',a') \tag{1}$$

Instead of using Transition probability T and Reward R, we can re-write the (1) into expectation over samples of reward r and the next state s'

$$Q(s,a) = \mathbb{E}_{r,s'}[r + \gamma \max_{a'} Q(s',a')] \tag{2}$$

We can derive the incremental update rule to estimate the action value function as below:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{3}$$

Here α is the learning rate constant, it used in reinforcement learning where the weights of the older samples decay exponentially at the rate of $1 - \alpha$.

1.1 Small Data Set

For the Small dataset, we used Julia for our code. We used the Julia code reference from the textbook. We used the $Q - learning$ a type of model free reinforcement learning to train the agent that can make decisions based on the small.csv file. Our goal is to generate a policy file out of the dataset, which is a set of actions that the agent should take at each state to maximize the total reward. Initially we parse the csv file and read the data into an Dataframe. In the dataframe, we split the current state (s), action (a), reward (r) and the next state (s').

Later, we used the $Q - learning$ struct from the text book. The struct has all the variables for the $Q - learning$ algorithm implementation. The State Space (S), Action Space (A), discount factor (γ), the action value function Q, learning rate (α)

The update! function implements the Q-learning update rule when the variables state, action, reward and next state were passed in. It updates the Q-value for the state action pair based on the reward and maximum Q value over all actions in next state. For the Small data set we defined the number of states as 100 and the actions as 4. We used the discount factor value provided which is 0.95, we assumed the learning rate as 0.5. We initialized a Q-table with zero matrix, a Q-learning model is created using these values. The code iterates over the small dataset and updates the Q values using the update! function and determines the best action for each state by finding the action with the maximum Q-value in each state.The best action for each state gets added into a file which we later renamed as small.policy.

1.2 Medium Data Set

For the Medium dataset, we used Julia for our code. We used the Julia code reference from the textbook. We used the $Q-learning$ a type of model free reinforcement learning to train the agent that can make decisions based on the medium.csv file. Our goal is to generate a policy file out of the dataset, which is a set of actions that the agent should take at each state to maximize the total reward. Initially we parse the csv file and read the data into an Dataframe. In the dataframe, we split the current state (s), action (a), reward (r) and the next state (s').

Later, we used the $Q - learning$ struct from the text book. The struct has all the variables for the $Q - learning$ algorithm implementation. The State Space (S), Action Space (A), discount factor (γ), the action value function Q, learning rate (α)

The update! function implements the Q-learning update rule when the variables state, action, reward and next state were passed in. It updates the Q-value for the state action pair based on the reward and maximum Q value over all actions in next state. For the medium dataset we defined the number of states as $50,000$ and the actions as 7. We used the discount factor value as 1 given the problem is undiscounted, we assumed the learning rate as 0.5. We initialized a Q-table with zero matrix, a Q-learning model is created using these values. The code iterates over the medium dataset and updates the Q values using the update! function and determines the best action for each state by finding the action with the maximum Q-value in each state.The best action for each state gets added into a file which we later renamed as medium.policy.

1.3 Large Data Set

For the Large dataset, we used Julia for our code. We used the Julia code reference from the textbook. We used the $Q - learning$ a type of model free reinforcement learning to train the agent that can make decisions based on the large.csv file. Our goal is to generate a policy file out of the dataset, which is a set of actions that the agent should take at each state to maximize the total reward. Initially we parse the csv file and read the data into an Dataframe. In the dataframe, we split the current state (s), action (a), reward (r) and the next state (s').

Later, we used the $Q - learning$ struct from the text book. The struct has all the variables for the $Q - learning$ algorithm implementation. The State Space (S), Action Space (A), discount factor (γ), the action value function Q, learning rate (α)

The update! function implements the Q-learning update rule when the variables state, action, reward and next state were passed in. It updates the Q-value for the state action pair based on the reward and maximum Q value over all actions in next state. For the large dataset we defined the number of states as 312020 and the actions as 9. We used the discount factor value as 0.95 as given in the problem statement, we assumed the learning rate as 0.5. We initialized a Q-table with zero matrix, a Q-learning model is created using these values. The code iterates over the large dataset and updates the Q values using the update! function and determines the best action for each state by finding the action with the maximum Q-value in each state.The best action for each state gets added into a file which we later renamed as large.policy.

2 Performance Characteristics

DataSet	Run time	Score
small.csv	0.151704 seconds	22.497265576908198
medium.csv	0.215932 seconds	113.92125999999858
large.csv	0.344406 seconds	1402.6954871258224

Table 1: run time, relative policy score for each dataset

3 Code

```
using CSV
using DataFrames

data = CSV.read("/home/karthikpythireddi/Stanford_Classes/AA228/AA228-CS238-
    Student/project2/data/large.csv", DataFrame)

# Extract the columns for state (s), action (a), reward (r), and next state (
    s')
s = data[!, :s]
a = data[!, :a]
r = data[!, :r]
s_prime = data[!, :sp]


# Define a Q-learning model type
mutable struct Q_learning
    S # state space
    A # action space
    gamma # discount factor
    Q # action value function
    alpha # learning rate

end
```

```julia
lookahead(model :: Q_learning, s, a) = model.Q[s, a]

function update!(model :: Q_learning, s, a, r, s_prime)
    gamma, Q , alpha = model.gamma, model.Q, model.alpha
    Q[s, a] = Q[s, a] + alpha * (r + gamma * maximum(Q[s_prime, :]) - Q[s, a
    ])
    return model
end


#given the number of states and actions
num_states = 312020
num_actions = 9

#initialize the Q-learning model
gamma = 0.95 # discount factor
alpha = 0.5 # learning rate
Q = zeros(num_states, num_actions)

@time begin
model = Q_learning(num_states, num_actions, gamma, Q, alpha)

best_actions = Vector{Int}(undef, num_states)
    # Iterate through your data to update Q-values
    num_rows = size(data, 1)
    for i in 1:num_rows
        state = s[i]
        action = a[i]
        reward = r[i]
        next_state = s_prime[i]
        update!(model, state, action, reward, next_state)
    end

    # Now that the Q-values have been updated, find the best action for each
    state
    for state in 1:num_states
        best_action = argmax(Q[state, :])
        best_actions[state] = best_action
    end
end

# Write the policy to a file
policy_output = "/home/karthikpythireddi/Stanford_Classes/AA228/AA228-CS238-
    Student/project2/data/.policy"
open(policy_output, "w") do f
    for action in best_actions
        write(f, "$action\n")
    end
end
```