Project 2: Reinforcement Learning

Etienne Jacquot ejacquot@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Descriptions

For this exercise, mainly two reinforcement learning algorithms were tested to obtain an optimal policy for all three datasets. The first is a model based approach as it can explicitly estimate transition probabilities and rewards from the dataset. It can be used with dynamic programming to find optimal values estimating the probability of transitions and average reward for each (s,a) pair first, initializing the value function second and lastly performing value iteration from which the optimal policy is found.

The second algorithm test was a model-free, Q-learning algorithm, not requiring building explicit representations of transition and reward models. It updates a state–action value Q(s,a) based on the Bellman equation. This method was chosen, and can be found in the second portion of this report. In this method, the process is repeated over 50 epochs or iterations/evaluations, where `delta` is monitored to ensure convergence.

1.1 Small Data Set

Both algorithms worked fine for the small data set, having 100 states and 4 actions, and using the provided discount factor $\gamma = 0.95$ and a learning rate $\alpha = 0.05$. The model-free alternative was chosen for performance and the training took 4.519 seconds. The policy was saved into the `small.policy` file.

1.2 Medium Data Set

The medium data set, having 50000 sets and 7 actions, was the most challenging out of the three given sets. In fact, due to its non-Markovian nature, adapting the Q-learning algorithm to it required further fine-tuning as it first led to unstable results (diverging `delta`). The problem being undiscounted, the γ factor in learning was taken as 0.95 for convergence and the learning rate as $\alpha = 0.01$. Reward shaping was also added to improve stability. The policy was saved into the `medium.policy` file.

1.3 Large Data Set

The large data set, having 302020 sets and 9 actions, also worked with both methods, using the provided discount factor $\gamma = 0.95$ and a learning rate $\alpha = 0.05$. The training of the algorithm on the large data set took 10.165 seconds. The policy learned was saved into the `large.policy` file.

The parameters given and/or used in the codes can be found in Table 1 below.

Dataset	States	Actions	γ	α	Epochs	Time (s)
`small.csv`	100	4	0.95	0.05	50	4.52
`medium.csv`	50,000	7	0.95	0.01	50	65.2
`large.csv`	302,020	9	0.95	0.05	50	310.1

Table 1: Training parameters and measured running time for each policy.

2. Code

2.1 Small and Large Data Sets

```julia
using CSV, DataFrames, Random

# Q-learning
function batch_q_learning(df; num_states, num_actions, gamma, alpha, epochs
    =50)
    Q = zeros(num_states, num_actions)

    for epoch in 1:epochs
        delta = 0.0
        for row in eachrow(shuffle(df))
            s, a, r, sp = row.s, row.a, row.r, row.sp
            target = r + gamma * maximum(Q[sp, :])
            old = Q[s, a]
            Q[s, a] += alpha * (target - old)
            delta = max(delta, abs(Q[s, a] - old))
        end
        println(" $epoch delta=$delta")
    end
    return Q
end

# Policy extraction
function extract_policy(Q)
    num_states, _ = size(Q)
    policy = [argmax(Q[s, :]) for s in 1:num_states]
    return policy
end

# main
function main()

    case = ARGS[1]
    if case == "small"
        file, gamma, alpha = "small.csv", 0.95, 0.05
        num_states, num_actions = 100, 4
    elseif case == "medium"
        file, gamma, alpha = "medium.csv", 1.0, 0.05
        num_states, num_actions = 50000, 7
    elseif case == "large"
        file, gamma, alpha = "large.csv", 0.95, 0.05
        num_states, num_actions = 302020, 9
    end

    println("loading $file")
    df = CSV.read(file, DataFrame)

    println("running Q-learning")
    Q = batch_q_learning(df; num_states=num_states, num_actions=num_actions,
    gamma=gamma, alpha=alpha)

    println("extracting policy")
    policy = extract_policy(Q)

    open("$(case).policy", "w") do f
        for a in policy
            println(f, a)
        end
    end

    println("saved to $(case).policy")
end

main()
```

2.2 Medium Data Set

```
using CSV, DataFrames, Random, Statistics, Dates

# Q-learning
function batch_q_learning(df; num_states, num_actions, gamma, alpha, epochs
    =50)
    Q = 0.001 * randn(num_states, num_actions)

    for epoch in 1:epochs
        delta = 0.0
        for row in eachrow(shuffle(df))
            s, a, r, sp = row.s, row.a, row.r, row.sp

            # reward shaping
            r += 0.1

            # terminal state handling
            if sp == s || r != 0
```

```julia
                target = r
            else
                target = r + gamma * maximum(Q[sp, :])
            end

            target = clamp(target, -1000.0, 1000.0)

            old = Q[s, a]
            update = alpha * (target - old)
            Q[s, a] += update

            delta = max(delta, abs(Q[s, a] - old))
        end
        println(" $epoch delta=$delta")
    end
    return Q
end

# Policy extraction
function extract_policy(Q)
    num_states, _ = size(Q)
    policy = [argmax(Q[s, :]) for s in 1:num_states]
    return policy
end

# main
function main()

    case = ARGS[1]
    if case == "small"
        file, gamma, alpha = "small.csv", 0.95, 0.05
        num_states, num_actions = 100, 4
    elseif case == "medium"
        file, gamma, alpha = "medium.csv", 0.95, 0.01
        num_states, num_actions = 50000, 7
    elseif case == "large"
        file, gamma, alpha = "large.csv", 0.95, 0.05
        num_states, num_actions = 302020, 9
    end

    println("loading $file")
    df = CSV.read(file, DataFrame)

    println("Running Q-learning ...")
    t_start = now()
    Q = batch_q_learning(df; num_states=num_states, num_actions=num_actions,
    gamma=gamma, alpha=alpha)
    t_end = now()
    elapsed = (t_end - t_start)
    println("Training time: $(Dates.value(elapsed) / 1000) seconds")
    println("Extracting policy ...")
    policy = extract_policy(Q)

    open("$(case).policy", "w") do f
        for a in policy
            println(f, a)
        end
    end

    println("unique(policy) = ", unique(policy))
    println("mean(Q) = ", mean(Q), " max(Q) = ", maximum(Q))
    println("Saved policy to $(case).policy")
end
main()
```