Project 2: Reinforcement Learning

Qianying Wu wuqy@stanford.edu
AA228/CS238, Stanford University

1. Description of strategy

The algorithms of different problems are varied based on the size of the possible states and the potential computation effort.

- Small Problem: The computation strategy of the policy search for the small MDP involves exact solution using value iteration and Gauss-Siedel iteration.
- Medium Problem: I first tried using exact solution for the Medium problem but it took too long, and also I found that there are many missing transitions in the data set, so it is not a good idea to use the exact solution to generate policies for the state and action combinations and are not present in the data set. Then I implemented the k-nearest neighbor algorithm to approximate the value function: first select 10000 states from the data set as seeds for neighbors, then approximate the value functions of other states using the mean of 10 nearest neighbors.
- Large Problem: I first used the k-nearest neighbor method, selecting the 500 states as seeds for neighbors, then approximate the value functions of other states using the mean of 10 nearest neighbors. I ended up with a policy that heavily prefers action 1. The reason is that although there are 312020 states, only 500 states appear in the data set, and many of the rewards are zero. This results in a positive score in the Gradescope system, but the score is relatively low. Then, I used linear regression to approximate the value function. This results in an increase of scores in Gradescope, although not very significant.

2. Performance characteristics, i.e. running/training time

	Small	Medium	Large
Run time (s)	10.5	179	4.9 (nearest neighbor) 6.6 (linear regression)

3. Code

Small Problem: exact solution

```
begin
filename1 = "data/small.csv"
filename2 = "data/medium.csv"
```

```
filename3 = "data/large.csv"
state_and_actions = Dict()
discounters = Dict()
state_and_actions[filename1] = (100, 4)
discounters[filename1] = 0.95
state_and_actions[filename2] = (50000, 7)
discounters[filename2] = 1.0
state_and_actions[filename3] = (312020, 9)
discounters[filename3] = 0.95
end

filename = filename1

## read data
df = DataFrame(CSV.File(filename))

nRow = size(df, 1)
nCol = size(df, 2)

function ProcessInput(df)
    r = Dict()
    t = Dict()
    for row in eachrow(df)
        key = (row.s, row.a)
        r[key] = row.r
        if !haskey(t, key)
            t[key] = []
        end
        push!(t[key], row.sp)
    end
    return r, t
end

begin
r, t= ProcessInput(df)
ns, na = state_and_actions[filename]
s = [i for i in 1:ns]
a = [i for i in 1:na]
end

struct Mdp
    S::Vector{Int64} # all states
    A::Vector{Int64} # all possible actions
    Y::Float64 # Discount factor
end

mdp = Mdp(s, a, discounters[filename])

function T(state::Int, action::Int, mdp::Mdp)
    key = (state, action)
    possibilities = zeros(length(mdp.S))

    if haskey(t, key)
        possible_next_states = t[key]
    else
        possible_next_states = []
    end

    total = length(possible_next_states)

    for item in possible_next_states
        possibilities[item]+=1/total
    end
    return possibilities
end


 function R(state::Int, action::Int)
    key = (state, action)
    if haskey(r, key)
        return r[key]
    else
        return 0
    end
end

function Lookahead(state::Int, action::Int, mdp::Mdp, U)
    result = R(state, action)
    possibilities = T(state, action, mdp)
    result += mdp.Y * sum([U[state] * possibilities[state] for state in mdp.S
    ])
    return result
end

function Evalute(state::Int, mdp::Mdp, U)
    value = -2e9
    action_candidate = -1
    for action in mdp.A
        v = Lookahead(state, action, mdp, U)
        if v > value
            value = v
            action_candidate = action
        end
    end
    return value, action_candidate
end

function ValueIteration(mdp::Mdp, k_max::Int)
    U = zeros(length(mdp.S))
    P = zeros(Int, length(mdp.S))
    for i in 1:k_max
        for state in mdp.S
            value, action = Evalute(state, mdp, U)
            U[state] = value
            P[state] = action
        end
    end

    return U, P
end

u, p = ValueIteration(mdp, 1000)

function Output(p, mdp, filename)
    prefixIndex = 0
    posfixIndex = 0
    for i in 1:length(filename)
        if filename[i] == '/'
            prefixIndex = i
        end
        if filename[i] == '.'
            posfixIndex = i
        end
    end
    suffix = chop(filename, head = prefixIndex, tail=length(filename)-
    posfixIndex+1)
    ans = []
    for i in 1:length(mdp.S)
        push!(ans, p[i])
    end
    writedlm(suffix * ".policy", ans)
end

Output(p, mdp, filename)
```

Midium Problem: k-nearest neighbor solution

```
### A Pluto.jl notebook ###
# v0.19.22

using Markdown
using InteractiveUtils

#  eca4a089-948e-41d0-b05e-95d7ffa6afcb
using CSV, DataFrames, Statistics, DelimitedFiles

#  db225f8b-11b2-4e06-a1fa-b4b456036450
using Plots
```

```
#  be96c7de-ab58-11ed-0424-f92fc4e1a017
begin
filename1 = "data/small.csv"
filename2 = "data/medium.csv"
filename3 = "data/large.csv"
state_and_actions = Dict()
discounters = Dict()
state_and_actions[filename1] = (100, 4)
discounters[filename1] = 0.95
state_and_actions[filename2] = (50000, 7)
discounters[filename2] = 1.0
state_and_actions[filename3] = (312020, 9)
discounters[filename3] = 0.95
end

#  c3e07c29-c9ae-47a6-a9f6-1f89aa72c7d2
filename = filename3

#  5199f144-790e-4aab-ab93-458014681535
df = DataFrame(CSV.File(filename))

#  123e4fcb-2b09-4489-b0a1-30a6a6399a3c
function getPartialStates(df)
    s = Set()
    for row in eachrow(df)
        push!(s, row.s)
    end
    return [s...]
end

#  747fde76-bc56-4751-b860-719d23bb6748
ss = getPartialStates(df)[1:500]

#  a157a2f0-4c1c-4a96-a160-7c5efc1df50c
function computeKey(state, action)
    return Int64(action) * 1000000 + state
end

#  c8b6ebe0-ac47-4bf9-a5fe-68f1e307c992
function ProcessInput(df)
    r = Dict()
    t = Dict()
    total = Dict()
    for row in eachrow(df)
        key = computeKey(row.s, row.a)
        r[key] = row.r
        if !haskey(t, key)
            t[key] = Dict()
            total[key] = 0
        end
        total[key] += 1
        if !(row.sp in ss)
            continue
        end
        if !haskey(t[key], row.sp)
            t[key][row.sp] = 0
        end
        t[key][row.sp] += 1
    end
    for (key, value) in t
        for (key2, value2) in value
            t[key][key2] = value2 / total[key]
        end
    end
    return r, t
end

#  032060ef-a746-4832-a4ff-d1b1576953a3
begin
r, t= ProcessInput(df)
ns, na = state_and_actions[filename]
s = [i for i in 1:ns]
a = [i for i in 1:na]
end

#  74f55cac-1ce5-4b94-8ddb-bc3279531c0a
t

#  b4631a31-29cb-408f-a86a-9f3fd2f2d35c
struct Mdp
    S::Vector{Int64} # all states
    A::Vector{Int64} # all possible actions
    Y::Float64 # Discount factor
end

#  49bc2683-5155-4f71-96b8-0321075c01a1
mdp = Mdp(s, a, discounters[filename])

#  9593818d-aaef-47c0-b578-77b8d060d184
function R(state::Int, action::Int)
    key = computeKey(state, action)
    if haskey(r, key)
        return r[key]
    else
        return 0
    end
end

#  38537659-a1da-44a2-866e-ddc26fe2e671
mutable struct NearestNei
```

```
    k::Int64 # number of neighbors
    theta::Vector{Float64} # parameters for the states
    s::Vector{Int64} #the chosen states
    d # distance function for two different states
    k_max::Int64 #the number of iterations
end

#  c1619638-031c-4d94-bfb6-0dc5938de501
function linearFunction(method::NearestNei, s)
    distances = [method.d(s, ss) for ss in method.s]
    indexes = sortperm(distances)[1:method.k]
    res = mean([method.theta[index] for index in indexes])
    return Float64(res)
end

#  5988c6dc-b009-49ce-9f09-3f514bcc1a60
function Lookahead(state::Int, action::Int, mdp::Mdp, approximationMethod)
    key = computeKey(state, action)
    result = R(state, action)
    if haskey(t, key)
        for (state, poss) in t[key]
            result += mdp.Y * linearFunction(approximationMethod, state) *
    poss
        end
    end
    return result
end

#  786e8f02-447d-439a-b134-23f7236c6a9a
function Evaluate(state::Int, mdp::Mdp, approximationMethod::Any)
    return maximum([Lookahead(state, action, mdp, approximationMethod) for
    action in mdp.A])
end

#  5552afc7-6f32-4e10-8a49-728fa7a0a892
function fit!(method::NearestNei, u)
    method.theta = u
end

#  66e0dca9-51e4-44f1-93e8-8c30d27fb3ff
function solve(method::NearestNei, mdp::Mdp)
    for i in 1:method.k_max
        u = zeros(length(method.s))
        for (index, state) in enumerate(method.s)
            u[index] = Evaluate(state, mdp, method)
        end
        fit!(method, u)
    end
    p = ones(Int, length(mdp.S))
    for state in mdp.S
```

```
        tmp = -2e9
        chosen_action = -1
        for action in mdp.A
            value = Lookahead(state, action, mdp, method)
            if value > tmp
                tmp = value
                chosen_action = action
            end
        end
        p[state] = chosen_action
    end
    return p
end

#  5cb3aac1-b569-44a7-8683-0aa051ca13c9
function d1(s1, s2)
    pos1 = (s1 - 1)%500
    pos2 = (s2 - 1)%500
    vel1 = div(s1 - 1, 500)
    vel2 = div(s2 - 1, 500)
    return sqrt((pos1 - pos2) * (pos1 - pos2) + (vel1 - vel2) * (vel1 - vel2)
    )
end

#  d750628b-653f-494a-9462-29f90cfd7d0d
nn = NearestNei(10, [0.0 for i in 1:length(ss)], ss, d1, 10)

#  63ceea12-49c7-48ee-b16f-01ed01a29196
p = solve(nn, mdp)

#  8b0803e5-5a84-4329-aee7-e09b4ee43699
function Output(p, mdp, filename)
    prefixIndex = 0
    posfixIndex = 0
    for i in 1:length(filename)
        if filename[i] == '/'
            prefixIndex = i
        end
        if filename[i] == '.'
            posfixIndex = i
        end
    end
    suffix = chop(filename, head = prefixIndex, tail=length(filename)-
    posfixIndex+1)
    ans = []
    for state in mdp.S
        push!(ans, p[state])
    end
    writedlm(suffix * "_nn.policy", ans)
    return ans
```

```
end
```

Large Problem: Linear Regression solution

```
### A Pluto.jl notebook ###
# v0.19.22

using Markdown
using InteractiveUtils

#  eca4a089-948e-41d0-b05e-95d7ffa6afcb
using CSV, DataFrames, Statistics, DelimitedFiles

#  db225f8b-11b2-4e06-a1fa-b4b456036450
using Plots

#  696d36db-d580-4865-bbeb-845669631852
using LinearAlgebra

#  be96c7de-ab58-11ed-0424-f92fc4e1a017
begin
filename1 = "data/small.csv"
filename2 = "data/medium.csv"
filename3 = "data/large.csv"
state_and_actions = Dict()
discounters = Dict()
state_and_actions[filename1] = (100, 4)
discounters[filename1] = 0.95
state_and_actions[filename2] = (50000, 7)
discounters[filename2] = 1.0
state_and_actions[filename3] = (312020, 9)
discounters[filename3] = 0.95
end

#  c3e07c29-c9ae-47a6-a9f6-1f89aa72c7d2
filename = filename3

#  5199f144-790e-4aab-ab93-458014681535
df = DataFrame(CSV.File(filename))

#  c8b6ebe0-ac47-4bf9-a5fe-68f1e307c992
function ProcessInput(df)
    r = Dict()
    t = Dict()
    ss = Set()
    total = Dict()
    for row in eachrow(df)
        key = (row.s, row.a)
        r[key] = row.r
        push!(ss, row.s)
        if row.r  > 0 && row.a == 5
            println(row)
        end
        if !haskey(total, key)
            total[key] = 0
            t[key] = Dict()
        end
        total[key] += 1
        if !haskey(t[key], row.sp)
            t[key][row.sp] = 0
        end
        t[key][row.sp] += 1
    end
    for (key, value) in t
        for (key2, value2) in value
            t[key][key2] = value2 / total[key]
        end
    end

    return r, t, [ss...]
end

#  032060ef-a746-4832-a4ff-d1b1576953a3
begin
r, t, ss = ProcessInput(df)
ns, na = state_and_actions[filename]
s = [i for i in 1:ns]
a = [i for i in 1:na]
end

#  642c63ea-2384-49a3-8831-6a14a5925439
function R(state::Int, action::Int)
    key = (state, action)
    if haskey(r, key)
        return r[key]
    else
        return 0
    end
end

#  d4662696-a9be-482c-8dc7-c58eff2e65c5
function Lookahead(s, a, Y, u)
    res = R(s, a)
    key = (s, a)
    if haskey(t, key)
        for (state, poss) in t[key]
            res += Y * u[state] * poss
        end
    end
    return res
end

#  d66051e1-fb36-4800-a40e-311716f9fd96
function compute_real_u(Y, loops)
    u = Dict()
    for s in ss
        u[s] = 0.
    end
    for i in 1:loops
        for s in ss
            maximum = u[s]
            for aa in a
                tmp = Lookahead(s, aa, Y, u)
                if tmp > maximum
                    maximum = tmp
                end
            end
            u[s] = maximum
        end
    end
    return u
end

#  61e476ef-f18a-4aee-bbda-5ef0514e144c
u = compute_real_u(discounters[filename], 1000)

#  0cce0390-241c-4713-89b6-49b8aecde2a5
theta = [0., 0.]

#  2cde8491-bc03-4c1a-91ab-2e405992ac2c
function transpose(key)
    return [key % 1000, key / 1000]
end

#  977fd1b1-7227-4240-9358-a19116d744c2
function compute_loss(theta, u)
    res = 0.
    for (key, value) in u
        new_key = transpose(key)
        res += (dot(new_key, theta) - value) * (dot(new_key, theta) - value)
    end
    return res
end

#  d63899fc-d8aa-45a7-a8de-3378cccb8855
compute_loss(theta, u)

#  01db5297-2bbf-432c-9a97-b992becea538
function compute_gradient(theta, u)
```

```
    res = zeros(length(theta))
    for i in 1:length(theta)
        for (key, value) in u
            new_key = transpose(key)
            res[i] += 2 * (dot(new_key, theta) - value) * new_key[i]
        end
    end
    res /= length(u)
    return res
end

#  30ee398c-2227-4387-8554-dcee47d1199a
function optimize(theta, u, loops)
    for i in 1:loops
        theta -= 0.000001 * compute_gradient(theta, u)
        if i % 1000 == 0
            println(compute_loss(theta, u))
        end
    end
    return theta
end

#  08789093-6f1a-4940-91ba-8526b6150400
optimize(theta, u, 10000)

#  1444f16c-8470-4e65-9cff-4c2d9775f4f1
function newLookahead(s, a, Y, theta)
    res = R(s, a)
    key = (s, a)
    if haskey(t, key)
        for (state, poss) in t[key]
            new_key = transpose(state)
            res += Y  * poss * dot(new_key, theta)
        end
    end
    return res
end

#  8b0803e5-5a84-4329-aee7-e09b4ee43699
function Output(theta, filename)
    prefixIndex = 0
    posfixIndex = 0
    for i in 1:length(filename)
        if filename[i] == '/'
            prefixIndex = i
        end
        if filename[i] == '.'
            posfixIndex = i
        end
    end
    suffix = chop(filename, head = prefixIndex, tail=length(filename)-
    posfixIndex+1)
    ans = []
    for state in s
        action = 1
        maximum = -2e9
        for aa in a
            tmp = newLookahead(state, aa, discounters[filename], theta)
            if tmp > maximum
                maximum = tmp
                action = aa
            end
        end
        push!(ans, action)
    end
    writedlm(suffix * "_linear.policy", ans)
    return ans
end
```