Project 2 - Reinforcement learning

Thomas Hossler

thossler@stanford.edu
Departement of Geological Sciences, Stanford University

Keywords

reinforcement learning, value iteration, model free methods, Q learning algorithm

ABSTRACT

For this second project, the objective was to find the best policy for three different Markov Decision Processes (MDP): small, medium and large. The number of states as well as the global information vary for each process. Different approach have been implemented: a value iteration algorithm and a Q learning algorithm.

1. Introduction

Many important problems require the decision maker to make a series of decisions. To optimize the next action to be taken, the impact of this action must be forecasted. The challenge of such sequential decision problems is rising when the model is unknown or the environment is not observable. In this paper, two approaches are presented. The first one, when the model is known and the number of states small, has been implemented using value iteration. The second approach has been implemented to deal with large and unknown models, using a Q learning algorithm with eligibility traces. Three data sets have been used: **small**, **medium** and **large**. The value iteration algorithm have been applied to the **small** data set whereas the Q learning algorithm have been deployed for the **medium** and **large**.

2. Small dataset

The **small** data set consists of 100 states. From each state, 4 actions are possible. This problem is similar to the gridworld problem presented in the book. A value iteration algorithm is adapted to deal with it because the number of states is relatively small (and hence the value function can be computed analytically) and the model is fully known. However, the state transition probability matrix is not known and must be estimated. A maximum likelihood estimation has been implemented in this case to estimate this matrix, following equation 1:

$$T(s' \mid s, a) = N(s, a, s')/N(s, a) \tag{1}$$

where T is the transition probability matrix, N the number of counts, s the current state, s' the next state and a the action taken at the current state.

The value iteration pseudo-code is presented below:

```
function Value Iteration
        k = 0
        U0(s) = 0
        repeat
                Uk(s) satisfies the Bellman equation for all states s
                k = k+1
        until convergence
```

The Bellman equation is computed analytically, which is fast in a low dimensional case like this one.

The code is available in subsection 5.1. Score = 42.1584.

3. Medium and large dataset

For the **medium** data set, a value iteration algorithm is not adapted for several reasons: the model is unknown (even though it could be inferred due to the physical nature of the problem) and the number of states is too large (remember that the probability transition matrix size is $| S | \times | S |$ where S is the state space). Therefore, a Q learning algorithm has been implemented. Q learning does not require any explicit knowledge on the model. Due to the size of the problem and in order to accelerate the learning rate, eligibility traces have been added to the algorithm. The exploration strategy picked is an ϵ -greedy algorithm. From the Q matrix, the best action for each state available has been identified. Due to the physical nature of the problem, no generalization technique have been developed for that case. It has been assumed that the optimal action for a given "unknown" case is the same as the one for the closest state. The pseudo code is given below:

```
function Q learning
        t = 0
        pick a random initial state s
        initialize Q
        repeat
                choose action a using the epsilon greedy strategy
                update Q and N using eligibility traces
```

The code is available in subsection 5.2. Score = 13.2501.

The same algorithm and workflow has been applied for the large data set. The computation has been limited to 3600 seconds running time, which correspond to less than 50 episodes. The structure of the data has not been exploited. Score = 3.8409.

4. Discussion

Based on the score of each policy, a few observations can be made:

- The value iteration algorithm performs well. Improvements could have been made by inferring the transition probability matrix in a different way.
- The Q learning algorithm performs well in the case of the **medium** data set. The car does not always execute optimal actions but ends up reaching the flag. It especially seems that the actions taken when the car is a the bottom of the hills are not optimal (the car could gain momentum "faster"). A different initialization for Q could have led to better results.
- In the case of the **large** data set, the algorithm performs poorly. It might be explained by the fact that a similar "filling" technique for unknown state has been used than for the **medium** data set. Or no information is available on the distance between states. A generalization technique, such as local or global approximation could have been implemented to improve the performance. Moreover, no research on the structure of the data has been realized.

5. Code

5.1. Value iteration

```
small = csvread('small.csv',1,0);
data = small;
numberOfStates = length(unique(data(:,1)));
numberOfActions = length(unique(data(:,2)));
reward = zeros(numberOfStates,numberOfActions); %4 number of max actions
transitionProbs = zeros(numberOfStates,numberOfStates,numberOfActions);

% Maximum likelihood estimates of transition
for state1 = 1 : numberOfStates
    index1 = find(data(:,1) == state1); % find the states
    transitionsStates = data(index1,4);
    states = unique(transitionsStates);% find the possible transitions states
    actions = data(index1,2);
    for action = 1 : numberOfActions
        indAction = find(actions == action); %number of time we played the action
        countAction = length(indAction);
        i = find(data(:,1) == state1 & data(:,2) == action,1);
        reward(state1,action) = data(i,3);
        for state2 = 1 : length(states)
            index = find(transitionsStates(indAction) == states(state2));
            countState = length(index);
            transitionProbs(state1,states(state2),action) = countState/countAction;
        end
    end
end


% Value iteration loop
k = 0;
U0 = zeros(numberOfStates,1); %expected utility is 0 for all states
policy = zeros(numberOfStates,1);
convergence = 0;
Q = []; ite = 1;
while ~convergence
    for a = 1 : numberOfActions
        Q(:,a) = reward(:,a) + gamma*transitionProbs(:,:,a)*U0;
    end
    M = max(Q,[],2);
    Usave = U0;
    for i = 1 : numberOfStates
        m = max(Q(i,:));
        U0(i) = m;
        policy(i) = find(Q(i,:) == m,1);
    end
    if max(U0(:)-Usave(:)) < 0.01
        convergence = 1;
    end
    ite = ite+1;
end

csvwrite('small.policy',policy)
```

5.2. Q learning

5.2.1. Main code

```
function Q = Qlearning(data,epsilon,alpha,traces,lambda,gamma,type)

states = unique(data(:,1));

numberOfActions = length(unique(data(:,2)));
numberOfStates = length(unique(data(:,1)));
maxNumberOfStates = 1757600;
Q = zeros(maxNumberOfStates,numberOfActions);
N = zeros(maxNumberOfStates,numberOfActions);

A = data(:,1);
B = data(:,4);
stuckStates = setdiff(B,A); % states reachable but not leavable

iteMax = 100;
convergence = 10; threshold = 0.1;
episode = 0;
tic
while convergence > threshold && episode < iteMax && toc < 3600
    rnInd = floor(numberOfStates*rand()+1);
    s0 = states(rnInd); % pick a random initial start
    Q0 = Q;
    disp(['.....beginning of episode ' num2str(episode) ' initial state = ' num2str(
        s0)])
    ite = 1;
    reward = - 225;

    while reward <= 0

        %while ~any(objectiveState == s0) %&& ite < iteMax
        [nextstate,action0,reward] = epsilonGreedy(data,s0,epsilon,stuckStates,type)
            ; % perform epsilon-greedy search
        Qmax = maxQActions(data,Q,nextstate); % find the max of Q(s',a')
        if ite > 1 % for the first round, we just pick a random action

            if traces
                N(s0,action0) = N(s0,action0)+1;
                delta = reward + Qmax - Q(s0,action0);
                Q = Q + alpha*delta*N;
                N = lambda*N;
            else
                Q(s0,action0) = Q(s0,action0) + alpha*(reward+ gamma*Qmax - Q(s0,
                    action0));
            end

        end
        s0 = nextstate;
        ite = ite + 1;
    end
    episode = episode + 1;
    convergence = norm(Q0-Q,2);
    disp(['convergence = ' num2str(convergence)])

end
```

5.2.2. ϵ - greedy search

```
function [nextstate,action,reward] = epsilonGreedy(data,state,epsilon,stuckStates,
    type)
%Epsilon greedy search: exploration + exploitation
% data(:,1): state
% data(:,2): action
% data(:,3): reward
% data(:,4): next state
% epsilon: probability factor


temp = rand(); % uniformly distributed between [0,1]
if temp < epsilon
    [action,reward,nextstate] = selectRandom(data,state,stuckStates,type);
    %disp('Exploration Time....')
else
    [action,reward,nextstate] = selectMax(data,state,stuckStates,type);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [action,reward,nextstate] = selectMax(data,state,stuckStates,type)
% state = current state
% what action gives the max reward

index = find(data(:,1) == state); %localize the state
possibleAction = unique(data(index,2)); % identify the possible actions
states = unique(data(:,1));
numberOfStates = length(states);
R = zeros(length(possibleAction),1);

for i = 1 : length(possibleAction)
    tempAction = possibleAction(i);
    ind = find(data(index,2) == tempAction,1);
    R(i) = data(ind,3); %associated reward for state and action i
end

if sum(R) ~= 0  % 2 or more possible rewards
    [reward,actionind] = max(R); % identify the action that give the maximum reward
    action = possibleAction(actionind);
else
    reward = unique(R);
    actionind = floor(length(R)*rand()+1); % pick a random action
    action = possibleAction(actionind);
end

% maximum likelihood estimation
ind = find(data(:,1) == state & data(:,2) == action);

possibleNextStates = setdiff(unique(data(ind,4)),stuckStates);
% state reachable given s and action i
if ~isempty(possibleNextStates)
    prob = zeros(length(possibleNextStates),1);
    for i = 1 : length(possibleNextStates)
        ind2 = find(data(ind,4) == possibleNextStates(i));
        prob(i) = length(ind2);
    end
    prob = prob./sum(prob);

    cdf = cumsum(prob); % calculate the cdf
    r = rand(); % pick a random number
    t = cdf - r;
    j = find(t >= 0,1);
    nextstate = possibleNextStates(j);

elseif type == 1 %known model, get the closest state
    pnst = unique(data(ind,4));
    nextstateTemp = pnst(1);
    statesList = sort(unique(data(:,1))); % all the states available and sorted
    diff = abs(statesList - nextstateTemp);
    [~,idx] = min(diff);
    closestState = statesList(idx);
    nextstate = closestState;
elseif type == 0 %unknown model
    rnInd = floor(numberOfStates*rand()+1);
    nextstate = states(rnInd);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [action,reward,nextstate] = selectRandom(data,state,stuckStates,type)

index = find(data(:,1) == state); % locate state t
possibleAction = unique(data(index,2)); % identify the possile action

rnInd = floor(length(possibleAction)*rand()+1);
action = possibleAction(rnInd); % pick a random action

rewInd = find(data(:,1) == state & data(:,2) == action,1);
reward = data(rewInd,3);

states = unique(data(:,1));
numberOfStates = length(states);
% maximum likelihood estimation
ind = find(data(:,1) == state & data(:,2) == action);
possibleNextStates = setdiff(unique(data(ind,4)),stuckStates);

if ~isempty(possibleNextStates)
    prob = zeros(length(possibleNextStates),1);
    for i = 1 : length(possibleNextStates)
        ind2 = find(data(ind,4) == possibleNextStates(i));
        prob(i) = length(ind2);
    end
    prob = prob./sum(prob);

    cdf = cumsum(prob); % calculate the cdf
    r = rand(); % pick a random number
    t = cdf - r;
    j = find(t >= 0,1);
    nextstate = possibleNextStates(j);
elseif type == 1
    pnst = unique(data(ind,4));
    nextstateTemp = pnst(1);
    statesList = sort(unique(data(:,1))); % all the states available and sorted
    diff = abs(statesList - nextstateTemp);
    [~,idx] = min(diff);
    closestState = statesList(idx);
    nextstate = closestState;
elseif type == 0 %unknown model
    rnInd = floor(numberOfStates*rand()+1);
    nextstate = states(rnInd);
end
```