CS221 Autumn 2016 Homework [4]

SUNet ID: [06033811]
Name: [Chao Xu]

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my original work.

Problem 1: Value Iteration

In this problem, you will perform the value iteration updates manually on a very basic game just to solidify your intuitions about solving MDPs. The set of possible states in this game is -2, -1, 0, 1, 2. You start at state 0, and if you reach either -2 or 2, the game ends. At each state, you can take one of two actions: -1, +1.
If you're in state s and choose -1:
You have an 80% chance of reaching the state s1.
You have a 20% chance of reaching the state s+1.
If you're in state s and choose +1:
You have a 30% chance of reaching the state s+1.
You have a 70% chance of reaching the state s1.
If your action results in transitioning to state -2, then you receive a reward of 20. If your action results in transitioning to state 2, then your reward is 100. Otherwise, your reward is -5. Assume the discount factor is 1.

(a) Give the value of $V_opt(s)$ for each state s after 0, 1, and 2 iterations of value iteration. Iteration 0 just initializes all the values of V to 0. Terminal states do not have any optimal policies and take on a value of 0.
Using the following equation for determining the optimal value if take action a from state s:

$$Q_{opt}(s,a) = \sum_{s'} T(s,a,s')(Reward(s,a,s') + \gamma V_{opt}(s'))$$

Also, optimal value from state s:

$$V_{opt} = 0 \text{ if reach the end state}$$
$$V_{opt} = max_{a \subset Actions(s)} Q_{opt}(s,a) \text{ for everything else}$$

Therefore: $V_{opt}(-2) = V_{opt}(2) = 0$

For iteration 0:

s = 0:

Q(0,1) = 0.3(-5+0)+0.7(-5+0) = -5; Q(0,-1) = 0.8(-5+0)+0.2(-5+0) = -5

$V_{opt}(0) = -5$, and the policy can be either -1 or 1.

s = 1:

Q(1,-1)=0.8*(-5+0)+0.2*(100+0)= 16; Q(1,1)= 0.3*(100+0)+0.7*(-5+0)= 26.5

$V_{opt}(1) = 26.5$, and the policy is 1.

s = -1:

Q(-1,-1)=0.8*[20+0]+0.2*[-5+0]= 15; Q(-1,1)= 0.3*[-5+0]+0.7*[20+0]=12.5

$V_{opt}(-1) = 15$, and the policy is -1.

For iteration 1:

s = 0:

Q(0,1)= 0.3*[-5+26.5]+0.7*[-5+15]=13.45; Q(0,-1)=0.8*[-5+15]+0.2*[-5+26.5]=12.3

$V_{opt}(0) = 13.45$, and the policy is 1.

s = 1:

Q(-1,1)= 0.3*[-5-5]+0.7*[20+0]=11; Q(-1,-1)=0.8*[20+0]+0.2*[-5-5]= 14

$V_{opt}(1) = 14$, and the policy is -1.

s = -1:

Q(1,1)= 0.3*[100+0]+0.7*[-5-5]=23; Q(1,-1)=0.8*[-5-5]+0.2*[100+0]=12

$V_{opt}(-1) = 23$, and the policy is 1.

For iteration 2:

s = 0:

Q(0,1)= 0.3*[-5+23]+0.7*[-5+14] = 11.7; Q(0,-1)=0.8*[-5+14]+0.2*[-5+23]=10.8

$V_{opt}(0) = 11.7$, and the policy is 1.

s = 1:

Q(-1,1)= 0.3*[-5+13.45]+0.7*[20+0]=16.54; Q(-1,-1)=0.8*[20+0]+0.2*[-5+13.45]=17.69

$V_{opt}(1) = 17.69$, and the policy is -1.

s = -1:

Q(1,1)= 0.3*[100+0]+0.7*[-5+13.45]=35.92; Q(1,-1)=0.8*[-5+13.45]+0.2*[100+0]=26.76

$V_{opt}(-1) = 35.92$, and the policy is 1.

(b) What is the resulting optimal policy π_{opt} for all non-terminal states?
From above optimal values in part a, the optimal policy for all non-terminal states are as following:

$$\pi_{opt}(0) = 1,\ \pi_{opt}(1) = -1,\ \pi_{opt}(-1) = 1$$

Problem 2: Transforming MDPs

Let's implement value iteration to compute the optimal policy on an arbitrary MDP. Later, we'll create the specific MDP for Blackjack.

(a) See modified counterexampleMDP

(b) Suppose we have an acyclic MDP. We could run value iteration, which would require multiple iterations. Briefly explain a more efficient algorithm that only requires one pass over all the (s, a, s') triples.
For an acyclic MDP, the best way to have a more efficient algorithm is to reconstruct the states into a tree structure, where the root is the start states, the leaf is the end states, and parents are the intermediate states. Then we can start from leaves and determining the V_{opt} value from button to top in one pass since the leaves is not depended on the previous policy value.

(c) Suppose we have an MDP with states States a discount factor $\gamma < 1$, but we have an MDP solver that only can solve MDPs with discount 1. How can leverage the MDP solver to solve the original MDP?
Start from the following equation:

$$Q_{opt}(s, a) = \sum_{s'} T(s, a, s')(Reward(s, a, s') + \gamma V_{opt}(s'))$$

If the MDP solver can only solve states with a discount factor less than 1. Then we need to compensate the difference for $(1 - \gamma)V_{opt}(s')$. The easiest way to do this set up is with a new probability defined as:

$$T(s, a, s') = (1 - \gamma)$$

and a new rewards defined as:

$$Rewards(s, a, s') = -V_{opt}(s')$$

Problem 4: Learning to Play Blackjack

So far, we've seen how MDP algorithms can take an MDP which describes the full dynamics of the game and return an optimal policy. But suppose you go into a casino, and no one tells you the rewards or the transitions. We will see how reinforcement learning can allow you to play the game and learn the rules at the same time!

(a) See code

(b) Call *simulate* using your algorithm and the identityFeatureExtractor() on the MDP smallMDP, with 30000 trials. Compare the policy learned in this case to the policy learned by value iteration. Don't forget to set the explorationProb of your Q-learning algorithm to 0 after learning the policy. How do the two policies compare (i.e., for how many states do they produce a different action)? Now run simulate() on largeMDP. How does the policy learned in this case compare to the policy learned by value iteration? What went wrong?
Running Q-learning on smallMDP results in a very accurate($\tilde{9}$7%) policy compare with the policy running in value iteration method. On the other hand, policy from running Q-learning on largeMDP has much less consistence, about 80%, compare with the the policy from running value iteration method. Increase number of iteration would only slightly improve the results for largeMDP using Q-learning.
LargeMDP has too many states for Q-learning to explore, and therefore Q-learning might miss some states and results in an inaccurate policy. Better feature extractor can improve the results.

(c) See Code

(d) Now let's explore the way in which value iteration responds to a change in the rules of the MDP. Run value iteration on originalMDP to compute an optimal policy. Then apply your policy to newThresholdMDP by calling simulate with FixedRLAlgorithm, instantiated using your computed policy. What reward do you get? What happens if you run Q learning on newThresholdMDP instead? Explain.
The average utility for running value iteration on originalMDP is 7.27 and its 7.24 for newThresholdMDP by calling simulate with FixedRLAlgorithm. These two values are very close to each other. The Q-learning has a average utility of 12 which is what we expected. The reason is that Q-learning with function approximation extract can provide generalization.