CS221 Fall 2015 Homework [4]

SUNet ID: [06039473]

Name: [Lingjie Kong]

Collaborators: [list all the people you worked with]

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

**Problem 1: Value Iteration**

1. **Give the value of for each state s after 0, 1, and 2 iterations of value iteration.**

Because state -2 and state 2 are the terminal state, they should always have the optimal value of 0 and no optimal policies as below

Next, we need to evaluate the optimal value and policy for state -1, 0, and 1

We will use equation

Iteration 0:

State 0:

Policy ties for 1 and -1

State -1:

Policy is -1

State 1:

Policy is 1

Iteration 1:

State 0:

Policy is 1

State -1:

Policy is -1

State 1:

Policy is 1

Iteration 2:

State 0:

Policy is 1

State -1:

Policy is -1

State 1:

Policy is 1

Let’s list all result above as below for simplicity

Table Vopt and policy for each state at different iteration

|  |  |  |
| --- | --- | --- |
| Iteration 1 |  | Policy |
| State 0 | -5 | 1 or -1 tie |
| State -1 | 15 | -1 |
| State 1 | 26.5 | 1 |
| Iteration 2 |  | Policy |
| State 0 | 13.45 | 1 |
| State -1 | 14 | -1 |
| State 1 | 23 | 1 |
| Iteration 3 |  | Policy |
| State 0 | 11.7 | 1 |
| State -1 | 17.69 | -1 |
| State 1 | 35.915 | 1 |

1. **From the results that we get in 2 iterations, the resulting optimal policy for all non-terminal states should be as below**

**Problem 2: Transforming MDPs**

1. **Counter example is given in CounterexampleMDP**

The reason is that if the before states have higher reward than the regular next one, jumping to one of the before states might increase the optimal value. The result is totally state dependent as shown in the code.

1. **The simple algorithm for the acyclic MDP is as below**

Recall the equation for value iteration from lecture as

The reason why we need value iteration for such MDP problem is from the cyclic nature of most MDP and we need to iteration and check such converges to the final value that we want it to be.

However, under the condition that we are given that the MDP problem is acyclic, we can rewrite all state node and chance node like a binary tree (maybe more than just left and right for binary tree). The root will be the first state. Meanwhile, all leaves will be the final state.

In order to solve the problem in one pass rather than value iteration from first state, one can start from the leaves (end states) and back propagate from leaves to the node to calculate the for each state in one pass.

1. **Supposing that we have an MDP solver only can solve MDPs with discount. How can we leverage the MDP solver to solve the original DMP with**

Recall the equation from value iteration from lecture as

If we do only have solver that can solver , we overcalculate term . However, we can model such term by using one more state with some probability and some negative reward

**Problem 3: Peeking Blackjack**

1. **Implement the gram of Blackjack as an DMPs by filling out thing in code**
2. **You’re running a casino, and you’re trying to design a deck to make people peek a lot. Finish the code!**

**Problem 4: Learning to Play Blackjack**

1. **Implement a generic Q-learing algorithm**
2. **Compare policy from Q-learning and value iteration for small MDP and large MDP**

The policy for the small DMP learnt through Q-learning is as below

Trial 0 (totalReward = 6): [(0, None, (2, 2)), 'Take', 0, (1, None, (1, 2)), 'Take', 0, (6, None, (1, 1)), 'Quit', 6, (6, None, None)]

Trial 1 (totalReward = 6): [(0, None, (2, 2)), 'Take', 0, (5, None, (2, 1)), 'Take', 0, (6, None, (1, 1)), 'Quit', 6, (6, None, None)]

Trial 2 (totalReward = 10): [(0, None, (2, 2)), 'Take', 0, (5, None, (2, 1)), 'Take', 0, (10, None, (2, 0)), 'Quit', 10, (10, None, None)]

Trial 3 (totalReward = 6): [(0, None, (2, 2)), 'Take', 0, (1, None, (1, 2)), 'Take', 0, (6, None, (1, 1)), 'Quit', 6, (6, None, None)]

Trial 4 (totalReward = 6): [(0, None, (2, 2)), 'Take', 0, (5, None, (2, 1)), 'Take', 0, (6, None, (1, 1)), 'Quit', 6, (6, None, None)]

Compared the above policy to the value iteration result, we almost get **100%** accuracy for the Q-learning.

However, this is not the case for the policy for the large DMP learnt through Q-learning as below.

Trial 0 (totalReward = 2): [(0, None, (3, 3, 3, 3, 3)), 'Take', 0, (1, None, (2, 3, 3, 3, 3)), 'Take', 0, (2, None, (1, 3, 3, 3, 3)), 'Quit', 2, (2, None, None)]

Trial 1 (totalReward = 10): [(0, None, (3, 3, 3, 3, 3)), 'Take', 0, (10, None, (3, 3, 3, 3, 2)), 'Quit', 10, (10, None, None)]

Trial 2 (totalReward = 6): [(0, None, (3, 3, 3, 3, 3)), 'Take', 0, (1, None, (2, 3, 3, 3, 3)), 'Take', 0, (6, None, (2, 3, 2, 3, 3)), 'Quit', 6, (6, None, None)]

Trial 3 (totalReward = 5): [(0, None, (3, 3, 3, 3, 3)), 'Take', 0, (5, None, (3, 3, 2, 3, 3)), 'Quit', 5, (5, None, None)]

Trial 4 (totalReward = 8): [(0, None, (3, 3, 3, 3, 3)), 'Take', 0, (3, None, (3, 2, 3, 3, 3)), 'Take', 0, (8, None, (3, 2, 2, 3, 3)), 'Quit', 8, (8, None, None)]

By intuition, the threshold value is 40. Therefore, from value iteration as the value in hand is 2, 10, 5, 8 and so on, the optimal policy should be take rather than quit. By increasing the number of trials, it will slightly improve the policy. However, it is still not the optimal one. Therefore, there for large DMP, Q-learning is not accurate and vary a lot from value iteration result

The reason why there is a difference between the results for the small DMP and large MDP is the number of states. Under the condition that large DMP has huge amount of states to explore, Q-learning might not be able to explore all possible states and this is the reason why Q-learning is not accurate here. Therefore, we need to use function approximation with correct feature extractor to improve generalization.

1. **Implement blackjackFeatureExtractor as described in the code comment**
2. **Explore the way in which value iteration responds to a change in the rules of the DMP**

Running value iteration on original MDP to compute an optimal policy and applying to new Threshold MPD give an average utility of 6.82. This result makes sense because the new utility is not used for the training initially and optimal policy is still made base on original MDP.

However, when Q-learning with function approximation for generalization is used to learn original MDP and applying to the new threshold MPD give an average utility of 12 which is the expected utility for the new threshold.

The reason why Q-learning with function approximation can handle threshold change compared to value iteration is that rather than model everything in state space, Q-learning with function approximation extracts important features which can be used for generalization. This is the reason why Q-learning with function approximation can handle threshold noise