XCS221 Assignment 3

JUAN RICARDO PEDRAZA ESCOBAR

Artificial Intelligence: Principles and Techniques

Stanford Center for Professional Development

December,2021

1.a

Iteration 0

$$V_{opt}^{(0)}(-2) = 0$$

$$V_{opt}^{(0)}(-1) = 0$$

$$V_{opt}^{(0)}(0) = 0$$

$$V_{opt}^{(0)}(1) = 0$$

$$V_{opt}^{(0)}(2) = 0$$

Iteration 1

$$V_{opt}^{(1)}(-2) = 0$$

$$V_{opt}^{(1)}(-1) = \max\{0.8(20 + 0) + 0.2(-5 + 0), 0.3(-5 + 0) + 0.7(20 + 0)\} = 15$$

$$V_{opt}^{(1)}(0) = \max\{0.8(-5 + 0) + 0.2(-5 + 0), 0.3(-5 + 0) + 0.7(-5 + 0)\} = -5$$

$$V_{opt}^{(1)}(1) = \max\{0.8(-5 + 0) + 0.2(100 + 0), 0.3(100 + 0) + 0.7(-5 + 0)\} = 26.5$$

$$V_{opt}^{(1)}(2) = 0$$

Iteration 2

$$V_{opt}^{(2)}(-2) = 0$$

$$V_{opt}^{(2)}(-1) = \max\{0.8(20 + 0) + 0.2(-5 - 5), 0.3(-5 - 5) + 0.7(20 + 0)\} = 14$$

$$V_{opt}^{(2)}(0) = \max\{0.8(-5 + 15) + 0.2(-5 + 26.5), 0.3(-5 + 26.5) + 0.7(-5 + 15)\} = 13.45$$

$$V_{opt}^{(2)}(1) = \max\{0.8(-5 - 5) + 0.2(100 + 0), 0.3(100 + 0) + 0.7(-5 - 5)\} = 23$$

$$V_{opt}^{(2)}(2) = 0$$

1.b

The optimal policy is defined by,

$$\pi_{opt} = \begin{cases} 0, & if\ isEnd(s) \\ max_{a\ \epsilon\ Actions}, & Q_{opt}(s,a) \end{cases}$$

So, we have the optimal policy for the non-terminal states,

$$\pi_{opt}(-2) = EndState$$

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

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

$$\pi_{opt}(1) = 1$$

$$\pi_{opt}(2) = EndState$$

2.b

Given the condition that we are provided that the MDP problem is acyclic, we can rewrite all the state and chance nodes as a binary tree. Where all leaves will be the final state and the root will be the first state.

We also need the value iteration for such MDP problem is of the cyclic nature of most MDP and we need to iterate and test that such $V_{opt}\ (s)$ is converging to the final value we wish it to be. To solve the problem in a single pass we can start from the leaves (final states) and propagate backwards from the leaves to the node to compute the $V_{opt}\ (s)$ for each state in a single pass instead of iterating the value from the first state.

The equation for value iteration;

$$V_{opt}^{t}(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} T(s,a,s')[Reward(s,a,s') + \gamma V_{opt}^{(t-1)}(s')]$$

2.c

We have to understand the effect of $\gamma < 1$ on our data set so we are going to define a new end state o. We are assuming that $1 - \gamma$ is the probability reaching that state from any other states.

Therefore, our transition probabilities,

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

Now we are going to adjust the rewards to $\frac{1}{\gamma} Rewards(s,a,s')$ in ord3er to have the same V_{opt},

$$V_{opt}^{(1)}(s) = \begin{cases} 0, & if\ isEnd(s) \\ max_{a\ \epsilon\ Actions}, & \sum_{s'} T'(s,a,s')[Reward(s,a,s')] \end{cases}$$

$$= \begin{cases} 0, & if\ isEnd(s) \\ max_{a\ \epsilon\ Actions}, & \sum_{s'} \gamma\ T(s,a,s') \frac{Reward(s,a,s')}{\gamma} \end{cases}$$

$$V_{opt}^{(t)}(s) = \begin{cases} 0, & if\ isEnd(s) \\ max_{a\ \epsilon\ Actions}\ , & \sum_{s'} \gamma T(s,a,s')[\frac{Reward(s,a,s') + \gamma V_{opt}^{(t-1)}(s')}{\gamma}] \end{cases}$$

$$= \begin{cases} 0, & if\ isEnd(s) \\ max_{a\ \epsilon\ Actions}\ , & \sum_{s'} T'(s,a,s')\ [Reward(s,a,s') + \gamma V_{opt}^{(t-1)}(s')] \end{cases}$$

We have an MDP solver that only handle discount factor of 1, so when solving for $\gamma < 1$ we adjust the transition probabilities and rewards using this,

$$T'(s,a,s') = \begin{cases} (1-\gamma) & , \quad if\ \ s' = o \\ \gamma T(s,a,s')\ , & otherwise \end{cases}$$

$$R'(s,a,s') = \begin{cases} (1-\gamma) & , \quad if\ \ s' = o \\ \frac{1}{\gamma} R(s,a,s')\ , & otherwise \end{cases}$$

4.b

After run the simulations we can see that the small MDP Q-learning did a good job of learning the policy, that is because the state space is relatively small , so the small MDP Q-learning can learn the Q values for ich state action pair, therefore it can learn when to take, peek or quit.
In the other hand the largeMDP Q-learning did worse. Under the condition that largeMDP has huge number 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 correct feature extractor to improve generalization using a function approximation.

4.d

Q-learning with function approximation can handle the threshold change compared to value iteration and this is because Q-learning with function approximation extracts important features that can be used for generalization instead of modeling everything in the state space. This is the reason why Q-learning with function approximation can handle threshold noise well. When Q-learning with function approximation for generalization is used to learn the original MDP and applying it to the new MPD threshold gives an average utility of 12, which is the utility we expect for the new threshold.