FUDAN UNIVERSITY

Project 3: Blackjack

Deng Wen 17307130171

2019/10/26

Problem 1 Value Iteration

a.

Recall the Bellman Update Equation in Value Iteration:

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a\in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$

where in this problem, the reward function is based on $\{s, a, s'\}$, so we rewrite the equation as

$$V_{k+1}(s) \leftarrow \max_{a\in A(s)} \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

Then for 0-th iteration, we initialize the five states' values to 0.

$$V(-2) = V(-1) = V(0) = V(1) = V(2) = 0$$

For 1st iteration, apply Bellman Update Equation, $(\gamma = 1)$

$$\begin{aligned}
&V(-2) = 0\\
&V(-1) = \max\{0.3\times(-5+0)+0.7\times(20+0), 0.8\times(20+0)+0.2\times(-5+0)\} = 15\\
&V(0) = \max\{0.3\times(-5+0)+0.7\times(-5+0), 0.8\times(-5+0)+0.2\times(-5+0)\} = -5\\
&V(1) = \max\{0.3\times(100+0)+0.7\times(-5+0), 0.8\times(-5+0)+0.2\times(100+0)\} = 26.5\\
&V(2) = 0
\end{aligned}$$

For 2nd iteration, similar to 1st,

$$\begin{aligned}
&V(-2) = 0\\
&V(-1) = \max\{0.3\times(-5-5)+0.7\times(20+0), 0.8\times(20+0)+0.2\times(-5-5)\} = 14\\
&V(0) = \max\{0.3\times(-5+26.5)+0.7\times(-5+15), 0.8\times(-5+15)+0.2\times(-5+26.5)\} = 13.45\\
&V(1) = \max\{0.3\times(100+0)+0.7\times(-5-5), 0.8\times(-5-5)+0.2\times(100+0)\} = 23\\
&V(2) = 0
\end{aligned}$$

b.

After 2 iterations, $V_{opt}(-2) = 0,\ V_{opt}(-1) = 14,\ V_{opt}(0) = 13.45,\ V_{opt}(1) = 23,\ V_{opt}(2) = 0$,
Due to policy extraction equation,

$$\pi_{opt}(s) = \arg\max_{a \in A(s)} \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

Then the optimal policy for non-terminal states is

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

Problem 2: Transforming MDPs

a.

The noise in this problem can raise the transition probability of the (s,a,s') triple whose original probability is relatively lower and reduce the transition probability of the (s,a,s') triple whose original probability is relatively higher. So it is easy to construct a counter example by assigning the lower-reward (s,a,s') with higher probability and the higher-reward (s,a,s') with lower probability.

Please check my code for details.

b.

Recall the Value Iteration Equation

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

For MDPs which are not acyclic, there is not a specific order of states which means state updated later may affect the values of state updated earlier. So it will need multiple iterations.
But for MDPs which are acyclic, we can use DFS on the start state to search the terminal state and compute the value of the state and then backtrack to compute the parent state. Then one by one we can finally compute the start state's value. (That is to update from terminal state to start state.) Thus we can compute V_{opt} of each state with only one single pass over all the (s,a,s') triples.

c.

For the original MDP:

$$\begin{aligned} V_{k+1}(s) &= \max_{a \in A(s)} \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma V_k(s')) \\ &= \max_{a \in A(s)} \sum_{s'} \gamma p(s'|s,a)(\frac{1}{\gamma}R(s,a,s') + V_k(s')) \\ &= \max_{a \in A(s)} \sum_{s'} p'(s'|s,a)(R'(s,a,s') + V_k(s')) \\ &\text{where } p'(s'|s,a) = \gamma p(s'|s,a) \text{ and } R'(s,a,s') = \frac{1}{\gamma}R(s,a,s') \text{ for } s,s' \in States,\ a \in Actions \end{aligned}$$

So it is easy to come up with the method by assigning a probability lead to state o for all states.

And to make

$$\sum_{s'} \gamma p(s'|s,a) + p'(o|s,a) = 1 \Longrightarrow p'(o|s,a) = 1-\gamma$$

That is (substitute $p(s'|s,a)$ with $T(s,a,s')$)

$$T'(s,a,s') = \begin{cases} \gamma T(s,a,s'), & \text{if } s' \in States \text{ and } s \neq o \\ 1-\gamma, & \text{if } s' = o \text{ and } s \in States \\ 1, & \text{if } s' = o \text{ and } s = o \\ 0, & \text{if } s' \in States \text{ and } s = o \end{cases}$$

For the reward function which is obvious

$$Reward'(s,a,s') = \begin{cases} \frac{1}{\gamma} Reward(s,a,s'), & \text{if } s' \in States \text{ and } s \neq o \\ 0, & \text{otherwise} \end{cases}$$

Note that state o should be an empty state leads to nowhere but itself and has Value 0.

Problem 3: Peeking Blackjack

a.

Please check my code for details.

b.

Please check my code for details.

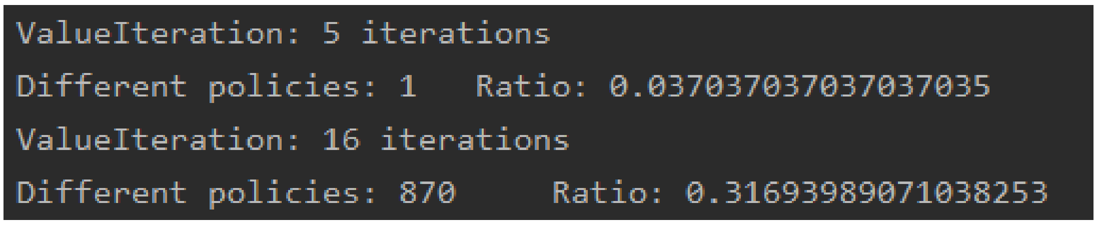
Problem 4: Learning to Play Blackjack

a.

Please check my code for details.

b.

I have written a test code in the $simulate\_QL\_over\_MDP()$ block and get a follwoing result:



```
ValueIteration: 5 iterations
Different policies: 1   Ratio: 0.037037037037037035
ValueIteration: 16 iterations
Different policies: 870     Ratio: 0.31693989071038253
```

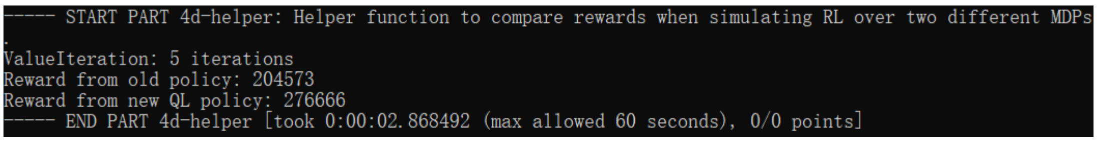
We can see that in the smallMDP, Q-Learning works well, and it has a litter difference with Value iteration. But in largeMDP, Q-Learning has a very big difference with Value iteration which is about %68 of all states. I think on one hand, the size of the MDP states can affect Q-Learning becaues when the number of states is large, Q-Learning may not be able to learn enough information about all states. On the other hand, the $identityFeatureExtraction$ only provides the information of $(state, action)$, but ignores the context information which may cause difficulties for Q-Learning to learn.

c.

Please check my code for details.

d.

Again I have written a test code in the $compare\_changed\_MDP()$ block and get a follwoing result:



```
----- START PART 4d-helper: Helper function to compare rewards when simulating RL over two different MDPs
.
ValueIteration: 5 iterations
Reward from old policy: 204573
Reward from new QL policy: 276666
----- END PART 4d-helper [took 0:00:02.868492 (max allowed 60 seconds), 0/0 points]
```

In this code, the original MDP is with $cardValues = [1,5], multiplicity = 2, threshold = 10, peekCost = 1$.
And the new MDP is with $cardValues = [1,5], multiplicity = 2, threshold = 15, peekCost = 1$.
We can see that there is a change in threshold between the original MDP and the new MDP. But the reward by using the fixed policy learned in original MDP is about 204573 and is smaller than the reward got by using new policy learned by Q-Learning which is about 27666.
After look at each simulation, we find the reward each game use the fixed policy is all not greater than 10, becaues the old policy is based on the threshold 10. But Q-Learning policy is more flexible, some reward in each game can be more than 10 for it actually learned from the experience. So when the MDP has a change, the Q-Learning is more adaptive.