Homework 3 - Blackjack

曹旭 16307110230 大数据学院

Problem 1: Value Iteration

a. [3 points] 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.

Answer:

I will use Bellman Equation to handle this problem. The Bellman equation is defined as follows:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

The basic structure of problem a. is: 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 γ is 1.

Utility\State	-2	-1	0	1	2
V_0	0	0	0	0	0
V_1	0	15.0	-5.0	26.5	0
V_2	0	14.0	13.45	23.0	0

b. [3 points] What is the resulting optimal policy π_{opt} for all non-terminal states? It is a policy iteration problem.

Answer:

$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

$$\forall s \in S, \pi^*(s) = argmax_a Q^*(s,a) = argmax_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Finally, it will converge to:

Policy\State	-2	-1	0	1	2
π	0	-1	1	1	0

0 means No action, it is the end!

Problem 2: Transforming MDPs

a. [3 points] If we add noise to the transitions of an MDP, does the optimal value always get worse? Specifically, consider an MDP with reward function $\mathrm{Reward}(s,a,s')$, states States, and transition function $\mathrm{T}(s,a,s')$. Let's define a new MDP which is identical to the original, except that on each action, with probability 12, we randomly jump to one of the states that we could have reached before with positive probability. Let V1 be the optimal value function for the original MDP, and V2 the optimal value function for the modified MDP. Is it always the case that $\mathrm{V1}(S_{start}) \geq \mathrm{V2}(S_{start})$?

$$T'(s,a,s') = \frac{1}{2}T(s,a,s') + \frac{1}{2}\cdot\frac{1}{|\{s'': T(s,a,s'') > 0\}|}$$

Answer:

I have finished the code in submission.py. Please check. Thanks!

No, not always. There are still some situations that $V1(S_{start}) < V2(S_{start})$. Supposed we have a simple MDP situation below:

$$\text{State\_C} \leftarrow \text{State\_A} \rightarrow \text{State\_B}$$

We have 3 states: A, B, C. A is the start state and both B, C are terminal states. Assume that we have only one action: the original transaction function is $T(A,a,B) = 0.1$ and $T(A,a,C) = 0.9$, the reward function is $R(A,a,B) = 100$ and $R(A,a,C) = 1$, the discount factor $\gamma = 1$. Because we only have one action, the value function is:

$$V^*(s) = \sum_{s'} T(s,a,s')[R(s,a,s') + V^*(s')]$$

If we add noise like the function above. We may obtain the new transaction function: $T'(A,a,B) = 0.5 * 0.1 + 0.5 * 0.5 = 0.30$ and $T'(A,a,C) = 0.5 * 0.9 + 0.5 * 0.5 = 0.70$. It seems that the probability from A to B arise. Let us calculate $V1(S_{start})$ and $V2(S_{start})$:

$$V1(S_{start}) = 0.9 * (1 + 0) + 0.1 * (100 + 0) = 10.9$$
$$V2(S_{start}) = 0.7 * (1 + 0) + 0.3 * (100 + 0) = 30.7$$
$$V1(S_{start}) < V2(S_{start})$$

I also write the code in `CounterexampleMDP` of this situation.

b. [3 points] Suppose we have an acyclic MDP (you will not visit a state a second time in this process). 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.

Answer:

For acyclic graphs (for example, the MDP for Blackjack), we just need to do one iteration provided that we process the nodes in reverse topological order of the graph. This is the same setup as we had for dynamic programming in search problems, only the equations are different. The method is:

We use topological sorting on our state space from the start state to the end state. Then we calculate the utility value and update it in reverse topological order (the calculation result is used directly for the utility value update of the next layer in the reverse topology sorting). For acyclic graphs, we just need to do one iteration.

c. [3 points] 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?

Answer:

Let us define a new MDP with states $\text{States}' = \text{States} \cup \{o\}$, where o is a new state. Let's use the same actions ($\text{Actions}'(s) = \text{Actions}(s)$), but we need to keep the discount $\gamma' = 1$. Your job is to define new transition probabilities $T'(s,a,s')$ and rewards $\text{Reward}'(s,a,s')$ in terms of the old

MDP such that the optimal values $V_{opt}(s)$ for all $s \in$ States are the equal under the original MDP and the new MDP.
original MDP:

$$V_1^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_1^*(s')] \qquad \gamma < 1$$

new MDP:

$$V_2^*(s) = \max_a \sum_{s'} T'(s,a,s')[R'(s,a,s') + V_2^*(s')]$$

We want $V_1^*(s) = V_2^*(s)$ for all $s \in$ States after iterations.
First, we define some new transition probabilities and rewards of $\text{States}' = \text{States} \cup \{o\}$ and $\text{Actions}'(s) = \text{Actions}(s)$:

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

$$R'(s,a,s') = T(s,a,s') * R(s,a,s') * \frac{1}{\gamma}$$

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

$$R'(s,a,o) = 0$$

$$T'(o,a,o) = 1$$

$$R'(o,a,o) = 0$$

Then, we calculate one of the optimal values $V_{opt}(s)$ for example:

$$V_1^*(s1) = \max_a \sum_{s'} T(s,a,s')[R(s1,a,s1') + \gamma V_1^*(s1')]$$

$$V_2^*(s1) = \max_a \sum_{s'} T'(s1,a,s1')[R'(s1,a,s1') + V_2^*(s1')]$$

$$= \max_a \left(T'(s1,a,o)[R'(s1,a,o) + V_2^*(o)] + \sum_{s1'/o} T'(s1,a,s1')[R'(s1,a,s1') + V_2^*(s1')] \right)$$

$$= \max_a \left(0 + \sum_{s1'/o} \gamma * \left[T(s,a,s') * R(s,a,s') * \frac{1}{\gamma} + V_2^*(s1') \right] \right)$$

$$= \max_a \left(\sum_{s1'/o} T(s,a,s') * R(s,a,s') + \gamma * V_2^*(s1') \right)$$

$$= \max_a \sum_{s'} T(s,a,s')[R(s1,a,s1') + \gamma V_1^*(s1')]$$

$$= V_1^*(s1)$$

It indicated that $V_1^*(s) = V_2^*(s)$ in each iteration.

Problem 3: Peeking Blackjack

a. [10 points] Implement the game of Blackjack as an MDP by filling out the succAndProbReward() function of class BlackjackMDP.
I have finished the code in submission.py. Please check. Thanks!

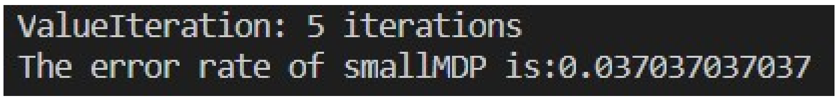
b. [4 points] Let's say you're running a casino, and you're trying to design a deck to make

people peek a lot. Assuming a fixed threshold of 20, and a peek cost of 1, design a deck where for at least 10% of states, the optimal policy is to peek. Fill out the function peekingMDP() to return an instance of BlackjackMDP where the optimal action is to peek in at least 10% of states.
I have finished the code in submission.py. Please check. Thanks!

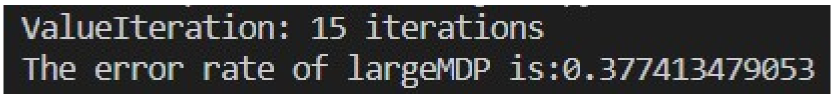
Problem 4: Learning to Play Blackjack

a. [8 points]
I have finished the code in submission.py. Please check. Thanks!

b. [4 points] 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?
Answer:



```
ValueIteration: 5 iterations
The error rate of smallMDP is:0.037037037037
```



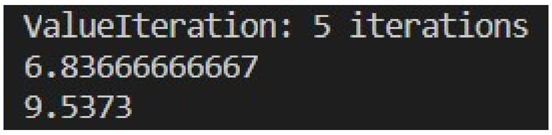
```
ValueIteration: 15 iterations
The error rate of largeMDP is:0.377413479053
```

After running the code written by myself, I found that for smallMDP, the error rate is smaller than largeMDP,'s error rate. It seems that Q-learning is better to handle smallMDP problem. In my opinion, this does not mean that Q-learning cannot get the optimal solution. Firstly, Q-learning is better in smallMDP because its state space is relatively small. In a larger state space like largeMDP, we may have more interference routes which are suboptimal. Q-learning is off-policy learning, it is difficult to learn accurate values for all states and actions. Secondly, the identityFeatureExtractor may work not well. Our feature extractor cannot describe some special or complex states and actions.

c. [5 points]

I have finished the code in submission.py. Please check. Thanks!

d. [4 points] 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
Answer:



```
ValueIteration: 5 iterations
6.83666666667
9.5373
```

Training value iteration on originalMDP to compute an optimal policy and apply the policy to newThresholdMDP, we will obtain the average rewards of 6.836 in 30000 times gaming. However, if we choose to run Q learning on newThresholdMDP, we will obtain we will obtain the average rewards of almost 9.5 in 30000 times gaming. It is obviously that Q-learning is better! I check each choice of both value iteration and Q-learning and found that value iteration will stop playing earlier because the threshold is still 10 in its 'cognition'. Q-learning has higher rewards because it is reinforcement learning. Q-learning can adapt to new environment.

Reference

[1] Fard M M , Pineau J . MDPs with Non-Deterministic Policies.[J]. Advances in Neural Information Processing Systems, 2009, 21:1065.
[2] http://inst.eecs.berkeley.edu/~cs188/sp19/
[3] Dai P , Mausam, Weld D S , et al. Topological value iteration algorithms.[J]. Journal of Artificial Intelligence Research, 2011, 42(1):181-209.
[4] https://www.cs.bgu.ac.il/~asins131/wiki.files/HierarchicalMDPs.pdf