CS221, Spring 2019, PS2 Reconstruct

Haiyuan Mei (hmei0411@stanford.edu)

Problem 1: Value Iteration

a. Give the value of $V_{opt}(s)$ for each state s afte 0,1,2 iterations

The following table shows values of each state after 0,1,2 iterations. Each row is an iteration, each column denotes a state. Suppose we apply the synchronized update, meaning iteration i updates are based on iteration i-1 results.

First initialize all values to be 0 in iteration 0;

According to the updating rule: $V^t(s) = \max_{a \in A} \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V^{t-1}(s')\right)$
In iteration 1:

$$
\begin{aligned}
V(-2) &= V(2) = 0 \\
V(-1) &= \max\left(0.8 * (20+0) + 0.2 * (-5+0), 0.3 * (20+0) + 0.7 * (-5+0)\right) = 15 \\
V(0) &= \max\left(0.8 * (-5+0) + 0.2 * (-5+0), 0.3 * (-5+0) + 0.7 * (-5+0)\right) = -5 \\
V(1) &= \max\left(0.8 * (-5+0) + 0.2 * (100+0), 0.3 * (-5+0) + 0.7 * (100+0)\right) = 68.5
\end{aligned}
$$

In iteration 2:

$$
\begin{aligned}
V(-2) &= V(2) = 0 \\
V(-1) &= \max\left(0.8 * (20+0) + 0.2 * (-5+-5), 0.3 * (20+0) + 0.7 * (-5+-5)\right) = 14 \\
V(0) &= \max\left(0.8 * (-5+15) + 0.2 * (-5+68.5), 0.3 * (-5+15) + 0.7 * (-5+68.5)\right) = 47.45 \\
V(1) &= \max\left(0.8 * (-5+-5) + 0.2 * (100+0), 0.3 * (-5+-5) + 0.7 * (100+0)\right) = 67
\end{aligned}
$$

Put the results in a table, the '-1/+1' in the brackets are actions taken by the states:

state	**-2**	**-1**	**0**	**1**	**2**
itr = 0	0	0	0	0	0
itr = 1	0	15(-1)	-5(tie)	68.5(+1)	0
itr = 2	0	14(-1)	47.45(+1)	67(+1)	0

b. What's the resulting optimal policy π_{opt}?

Run the iteration a couple of more times.

In iteration 3:

$$V(-2) = V(2) = 0$$
$$V(-1) = \max(0.8 * (20 + 0) + 0.2 * (-5 + 47.45), 0.3 * (20 + 0) + 0.7 * (-5 + 47.45)) = 35.7$$
$$V(0) = \max(0.8 * (-5 + 14) + 0.2 * (-5 + 67), 0.3 * (-5 + 14) + 0.7 * (-5 + 67)) = 46.1$$
$$V(1) = \max(0.8 * (-5 + 47.45) + 0.2 * (100 + 0), 0.3 * (-5 + 47.45) + 0.7 * (100 + 0)) = 82.73$$

In iteration 4:

$$V(-2) = V(2) = 0$$
$$V(-1) = \max(0.8 * (20 + 0) + 0.2 * (-5 + 46.1), 0.3 * (20 + 0) + 0.7 * (-5 + 46.1)) = 34.77$$
$$V(0) = \max(0.8 * (-5 + 35.7) + 0.2 * (-5 + 82.735), 0.3 * (-5 + 35.7) + 0.7 * (-5 + 82.735))$$
$$V(1) = \max(0.8 * (-5 + 46.1) + 0.2 * (100 + 0), 0.3 * (-5 + 46.1) + 0.7 * (100 + 0)) = 82.33$$

In iteration 5:

$$V(-2) = V(2) = 0$$
$$V(-1) = \max(0.8 * (20 + 0) + 0.2 * (-5 + 63.62), 0.3 * (20 + 0) + 0.7 * (-5 + 63.62)) = 47$$
$$V(0) = \max(0.8 * (-5 + 34.77) + 0.2 * (-5 + 82.335), 0.3 * (-5 + 34.77) + 0.7 * (-5 + 82.33))$$
$$V(1) = \max(0.8 * (-5 + 63.6) + 0.2 * (100 + 0), 0.3 * (-5 + 63.6) + 0.7 * (100 + 0)) = 87.6$$

Put these in a table, together with the actions taken by each state:

state	-2	-1	0	1	2
itr = 3	0	35.7(+1)	46.1(+1)	82.73(+1)	0
itr = 4	0	34.77(+1)	63.62(+1)	82.33(+1)	0
itr = 5	0	47(+1)	63(+1)	87.6(+1)	0

From the above table we can see that all non-terminal states will finally make (+1) as the optimal policy.
The intuition is that state 2 has the max reward, hence pushing all states toward state 2.

Problem 2: Transforming MDPs

a. Does the optimal value always get worse when add noise to the transition?

False, see conter-example implementation

b. Compute V_{opt} with only a single pass for acyclic MDP.

Use a vector v to denote the values for all states, r denote the rewards for all the states. they both are n n-dimensional vectors.
The policy evaluation is just to solve bellman equation:

$$V^t(s) = \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^{t-1}(s')\right)$$

it's equivalence is the following:

$$V^t(s) = R(s) + \gamma \sum_{s'} T(s, a, s') V^{t-1}(s')$$

We can write it in Matrix + Vector form as following:

$$V^t = R + \gamma M V^{t-1}$$

Where M is the transition matrix. Basically M is a n by n matrix, the i-th row, j-th column denotes the probability of transitioning from the i-th state to j-th state.

The goal of convergence is to find a v s.t.

$$v = r + \gamma M v$$

Now apply linear algebra knowledge, we know that $v = (I - \gamma M)^{-1} r$. The single pass of all {s,a,s'} will hence build the transition matrix M and value vector r, and the solution of the above matrix equation can be done either by inversing an matrix, or iteratively until the magnitude of V^t is very close to V^{t-1}.

c. Leverage the MDP solver with $\gamma = 1$ to solve the original MDP with $\gamma < 1$

Add a new terminal state o with reword 0, initialize the value to 0; Apart from $\text{States}' = \text{States} \cup \{o\}$ and $\text{Actions}'(s) = \text{Actions}(s)$ defined in the questions, the new transition and rewards are defined as following:

$$\begin{aligned}
T'(s, a, s') &= \gamma T(s, a, s') \\
T'(s, a, o) &= 1 - \gamma \\
R'(s, a, s') &= \frac{1}{\gamma} R(s, a, s') \\
R'(s, a, o) &= 0
\end{aligned}$$

With the above new transition and rewards, the value iteration can be written as:

$$\begin{aligned} V^t(s) &= \max_{a \in A} \sum_{s' \in \text{States} \cup \{o\}} T'(s,a,s') \left(R'(s,a,s') + 1 * V^{t-1}(s')\right) \\ &= \max_{a \in A} \sum_{s' \in \text{States}} \gamma T(s,a,s') \left(\frac{1}{\gamma} R(s,a,s') + V^{t-1}(s')\right) \\ &= \max_{a \in A} \sum_{s' \in \text{States}} T(s,a,s') \left(R(s,a,s') + \gamma V^{t-1}(s')\right) \end{aligned}$$

Which is exactly the same as the original problem. Note that in the above second step, the state space is changed from $\text{States} \cup \{o\}$ to States, this is because $R'(s,a,o) = 0$ and as an end state, $V(o)$ is always 0 and state o can be just ignored from the summation.

Problem 4. Learning to Play Blackjack

b. Difference between different feature extractor

1. With feature 'identityFeatureExtractor', smallMDP has 1 out of 27 different actions;
2. largeMDP has 838 out of 2745 different actions.
3. The improved feature extractor, 'blackjackFeatureExtractor', reduces the number of different actions from 838 to 592.

The reason is that for largeMDP, identityFeatureExtractor lacks generality ability for only using a simple feature extractor, for large state space, more sophisticated features are needed.

d. Apply learnt MDP to another one

After the experiment, the three different results are shown below:

- FixedRLAlgorithm originalMDP average reward = 6
- FixedRLAlgorithm modifiedMDP average reward = 6
- QLearningAlgorithm originalMDP average reward = 5
- QLearningAlgorithm modifiedMDP average reward = 6

From the above results we can see that, although the second one uses modified MDP, the average reward is still 6, which means it still applies the policy learnt previously. This is because FixedRLAlgorithm doesn't explore, and cannot generalize;

The q-learning one however has different average rewards, even though modifiedMDP is tested on originalMDP. The reason is that, QLearningAlgorithm can both explore and generalize using a feature function. Even though test modifiedMDP with exploration rate 0, the average reward is still different, because the feature weight is still update as the test goes on.