CS221 Summer 2019 Homework 4

SUNet ID: jaigupta
Name: Jai Gupta

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

Problem 1

(a) *Iteration 0:*

$Q(state, action)$ values:

state/action	-1	1
-2	0	0
-1	0	0
0	0	0
1	0	0
2	0	0

$V(state)$ values:

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

Iteration 1:

$Q(state, action)$ values:

state/action	-1	1
-2	0	0
-1	0.8*20 + 0.2*-5 = 15	0.7*-5 + 0.3*20 = 2.5
0	0.8*-5 + 0.2*-5 = -5	0.7*-5 + 0.3*-5 = -5
1	0.8*-5 + 0.2*100 = 16	0.7*100+0.3*-5 = 68.5
2	0	0

$V(state)$ values:

state	$V(state)$
-2	0
-1	15
0	-5
1	68.5
2	0

Iteration 2:
$Q(state, action)$ values:

state/action	-1	1
-2	0	0
-1	0.8*(20 + 0) + 0.2*(-5 +-5) = 16 - 2 = 14	0.7*(-5 + -5) + 0.3*(20 + 0) = -7 + 6 = -1
0	0.8*(-5 + 15) + 0.2*(-5 + 68.5) = 8 + 12.7 = 20.7	0.7*(-5 + 68.5) + 0.3*(-5+15) = 44.45 + 3 = 47.45
1	0.8*(-5 + -5) + 0.2*(100 + 0) = -8 + 20 = 12	0.7*(100+0)+0.3*(-5+ -5) = 70 - 3 = 67
2	0	0

$V(state)$ values:

state	$V(state)$
-2	0
-1	14
0	47.5
1	67
2	0

(b) optimal policy:
The optimal policy from the second iteration would be:

state	$\pi_{opt}(state)$
-1	-1
0	1
1	1

To find the overall optimal policy we need to find the convergence point. If we continue for few more iterations, we get to the following values after 20 iterations after which there the Q values change by < 0.1 and we can assume convergence has been reached

Iteration 20
$Q(state, action)$ values:

state/action	-1	1
-2	0	0
-1	30.06	55.22
0	57.40	75.33
1	76.25	91.10
2	0	0

$V(state)$ values:

state	$V(state)$	$\pi_{opt}(state)$
-2	0	-
-1	55.22	1
0	75.33	1
1	91.10	1
2	0	-

Problem 2

(a) Yes, it is possible. Example in submission.py.

(b) If the MDP graph is acyclic, then we can traverse the graph in an order such that all successors ($Succ(s_i)$) of a state (s_i) are evaluated before the state itself is evaluated for the optimal policy. Lets assume we can find such an ordering of the states. Now while evaluting optimal policy for any state (s), we can use:

$$V_{opt}(s) = \max_{a \epsilon Actions(s)} \sum_{s' \epsilon Succ(s)} T(s,a,s') * [R(s,a,s') + \gamma V_{opt}(s')]$$

$$\pi_{opt}(s) = \arg\max_{a \epsilon Actions(s)} \sum_{s' \epsilon Succ(s)} T(s,a,s') * [R(s,a,s') + \gamma V_{opt}(s')]$$

Since V_{opt} has already been evaluated for all successors, this expression can be evaluated easily over all the states in a single pass.

Now, to find such an ordering of the states given the mdp graph such that all successors are evaluated before a state, we can use topological sort over the graph ($O(V+E)$). Topological sort internally performs a DFS over the graph and then sorts the nodes based on end timestamp of DFS. This ensures that edges of the graph are ordered in a single direction allowing us to use a DP like approach over the ordered nodes to quickly ($O(num\_states * num\_avg\_actions\_per\_state)$) evaluate the optimal policy for all states.

(c) This is similar to solving an mdp such that discount is 1. Hence, we can set γ' is 1, $T'(s,\pi(s),s') = \gamma T(s,\pi(s),s')$ and $Reward'(s,\pi(s),s') = \frac{Reward(s,\pi(s),s')}{\gamma}$.

Once we make these substitutions, the problem is that the sum of the transition probabilities from any of the existing state is not 1. Instead, it is γ for each state. Hence, an additional state o is added and an edge from each of the existing states (in $States$) to this state (o) is added with transition probability of $1 \quad \gamma$. Reward for all these transitions are 0 as we dont want to affect the $V_{opt}(s)$ values. Also, this state should be treated as a terminal state with $V_{opt}(o) = 0$ and no possible action.

Problem 4

(a) in submission.py

(b) On using the identity feature extractor, the q-learning policy performs pretty well for the smallMDP case. Since the number of states are very low, the trainig dataset contain mostly all the state transitions and are able to learn very well. Hence, out of 27 states, it differs from the policy learned from value iteration on only one state (3.70% difference)

In case of largeMDP, the same algorithm performs very poorly with difference from value iteration learned policy on 32.28% of the states (886 of 2745 states). This can be attributed to the fact that in this case all the states were not explored by the training process for Q-learning due to the large state space. Due to insufficient domain knowledge, the algorithm is not able to generalize to these unseen states. Currently we only use state indicator function as features and using more domain specific features can help the model generalize to these unseen states.

(c) in submission.py. Using the new set of features, the difference goes down from 32.28% to 20.29%.

(d)
- After using value iteration to get optimal policy for the originalMDP, using it on newThresholdMDP gives an expected reward of 6.83.
- Simulating q-learning directly on the newThresholdMDP problem gives an expected reward of 9.57. Similarly note that training an optimal policy (using value iteration) directly on newThresholdMDP gives a policy with optimal reward of 12.

We notice that a policy trained on the originalMDP does not perform very well on the newThresholdMDP. The drop in reward is expected because the agent is acting assuming the environment is same as the one it was trained on. For example, in this problem, the agent is unaware of the fact that the threshold has been lowered and will keep getting busted much more frequently.