CSE 5820 MACHINE LEARNING

Homework #2

**Problem 2.**

The value function v(s) provides long-term expected reward value of state, s as –

Derivation –

=

=

As in question to derive Bellman equation for action values –

The action value function for a policy π is defined as the expected return starting from state s, taking action a, and following policy π thereafter.In next equations, is the action taken in state St. The expectation is taken over all possible next states and actions, assuming that the policy π is followed thereafter.

=

= + []

=

=

=

=

=

=

**Problem 3.**

Designing an iteration algorithm based on action value (s , a) -

The algorithm will initialize the action value function Q , and iteratively update it until it converges using the Bellman Optimality equation for action values. The algorithm will stop when the updates no longer change the values or when the maximum number of iterations is reached.

A threshold is assumed for determining accuracy of estimation.

Start of Loop

Delta 🡨 0

Loop for each s

Loop for each a

Q(s , a) 🡨 ]

delta 🡨 max( delta, |Q(s , a) – V(s)|)

V(s) 🡨 |Q(s , a)

until delta <

A deterministic policy is output,

= Q(s’, a’)]

Here,

S = number of states, A= number of actions and .

**Implementation –**

def optimalactionpolicy(P, R, gamma, states, actions, threshold, iterations):

Q = np.zeros((states, actions))

for i in range(iterations):

delta = 0

for s in range(states):

for a in range(actions):

q = Q[s , a]

Q[s ,a] = np.sum([P[s,a,s1]\*(R[s,a,s1] + gamma \* np.max(Q[s1 , :])) for s1 in range(states)])

delta = max(delta, abs(q - Q[s , a]))

if delta < threshold:

break

return Q

variables used in function **optimalactionpolicy(P, R, gamma, states, actions, threshold, iterations):**

P – transition matrix for states, actions

R – Reward matrix for states actions

gamma – discount

states – number of states in MDP

actions – number of actions in MDP

Threshold -

Iterations – number of iterations

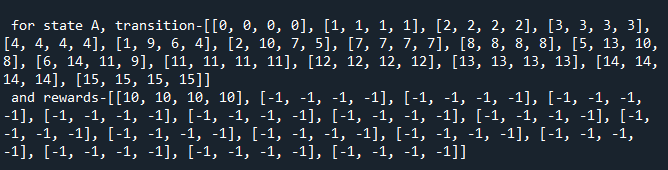
**Problem 4**

The python implementation of Example 3.5 , is modified to include states A and B with their respective rewards as provided in question description. Please find my code attached with submission named “GridWorld-Problem.py “. The Gridworld setup is created as mentioned in example/problem. Two arrays ‘transitions’ and ‘rewards’ are created for state A and two for state B. These arrays are updated in each iteration which overwrite the transitions.

Gridworld setup is created as a 5X5 grid with following details –

* State A – position at (0,1) with reward of +10 for any action transition
* State B – position at (0,3) with reward of +5 for any action transition
* Action that could take agent off the grid will give a reward -1
* Any other action gives a reward of 0.

Few initial steps of my output( you can run my code to see the output as well)-



A picture containing text, indoor, black, keyboard

Description automatically generated

A picture containing background pattern

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

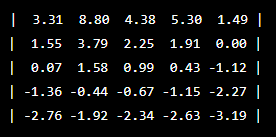
A screenshot of a computer

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

The final state value after 100 iterations of iterative policy evaluation algorithm with discount factor of 0.9 and using equiprobable random policy are as follows- (it’s not exactly same as in given problem but few of them are similar)-



Here each value in the table corresponds to the estimated value of corresponding state under the equiprobable random policy.