Introduction to
Artificial Intelligence and Machine Learning
Homework 4 - Reinforcement

2018/12/12

Question 1 – Value Iteration Agent

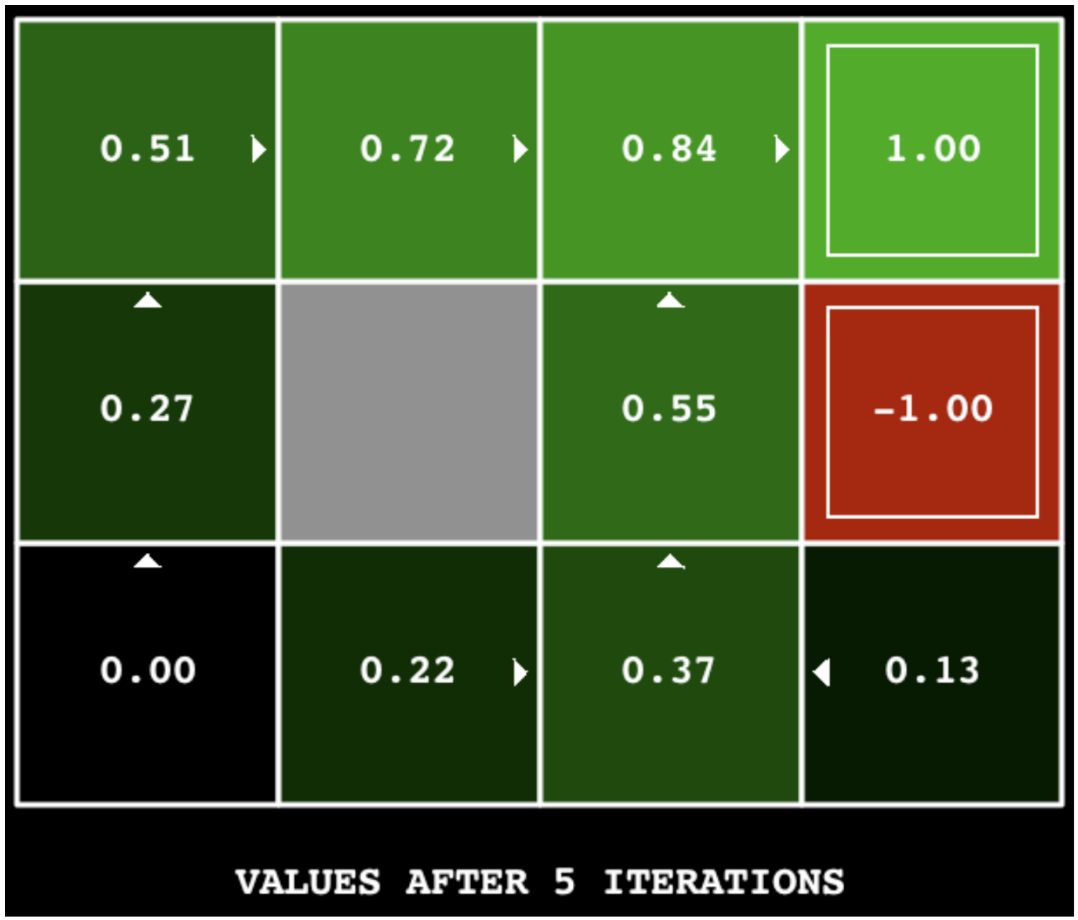
- An **MDP** is given
- "**valueIterationAgents.py**"
  - *U(s)*: self.values = util.Counter() – a dictionary
  - __init__(self, mdp, discount = 0.9, iterations =100):
    - For each iteration, for every state in the MDP, find the maximum value of *Q*(*s*, *a*) for all possible actions of state *s*
    - Recall that *U(s)* = $\max_{a \in A(s)} Q(s,a)$
  - getValue(self, state):
    - return self.values[state]

Question 1 – Value Iteration Agent

- getQValue(self, state, action):
  - Use getTransitionStatesAndProbs in mdp.py
  - $Q(s,a) = \sum_{s'} P(s'|s,a)[R(s'|s,a) + \gamma U(s')]$

- getPolicy(self, state):
  - If terminal state, return none.
  - Else, return the action that results in the maximum value of $E[\text{utility of taking a}] = \sum_{s'} P(s'|s,a)\, U(s')$

Question 1 – Value Iteration Agent



python gridworld.py -a value -i 5

Question 2 – Value Iteration Agent

- Change only **one** of the parameters, the discount factor γ or the noise level, so that the agent will cross the bridge in the optimal policy
  - **Noise level**: the uncertainty of taking an action
    - Ex: When noise=0, for any given state s and action a in A(s), there will be one s' such that P(s'|s,a)=1; for any other state s''≠s', it holds that P(s''|s,a)=0.
  - **Discount factor**: the level of importance of the future rewards
  - "**analysis.py**"

Question 2 – Value Iteration Agent

- Before modifying the parameters
- $\gamma = 0.2$, discount = 0.9

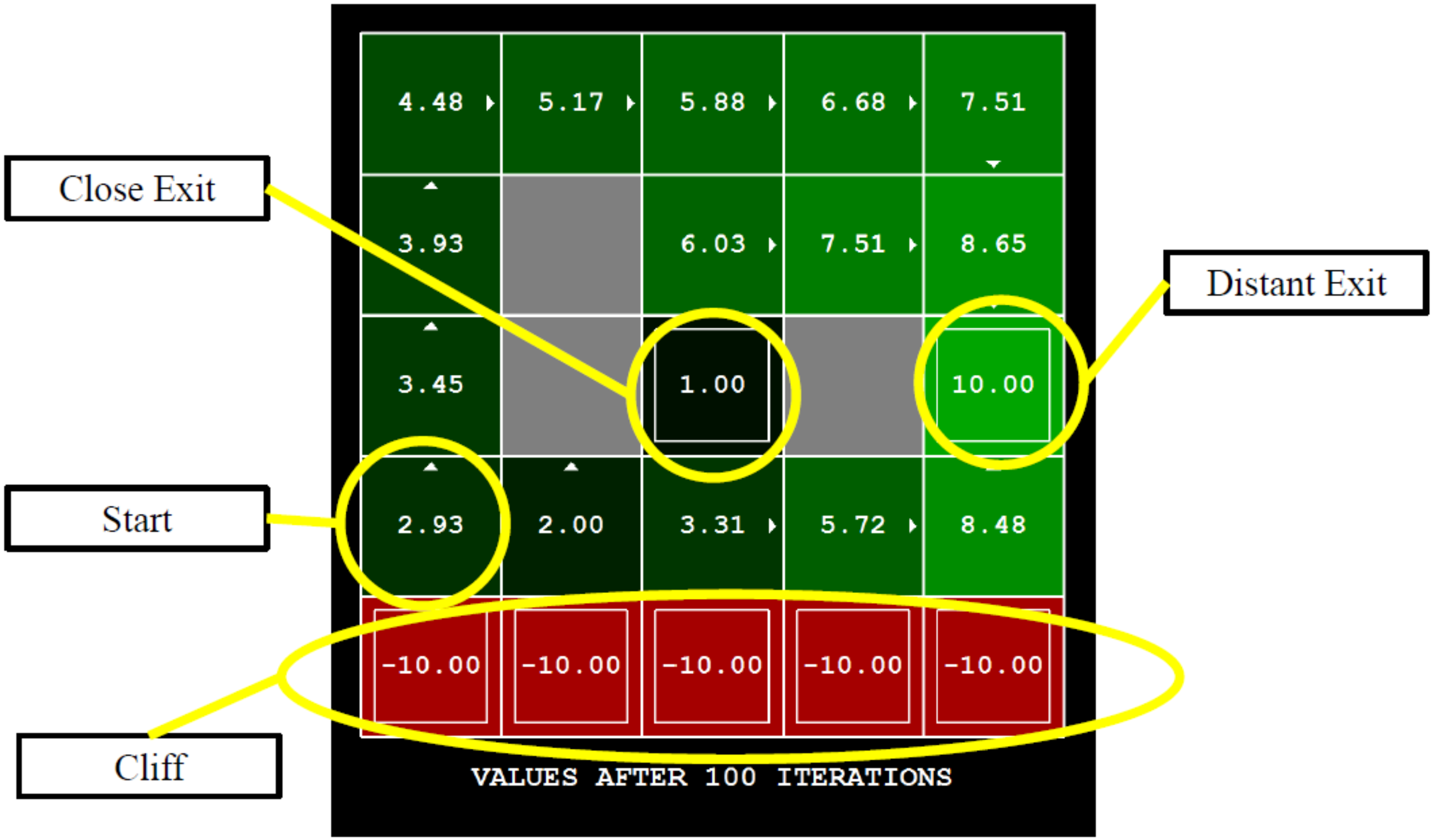
Question 2 – Value Iteration Agent

- The result should be something like this:

Question 3 – Value Iteration Agent

- Adjust the parameters, including the discount factor γ, the noise level, and the living reward, so that the agent acts as the descriptions
  - **Living reward**: The amount of reward given when the agent is still alive (i.e. doesn’t fall over the cliff)

Question 3 – Value Iteration Agent



Question 4~7 – Q Learning Agent

- Motivation: the transition probability and the reward of any given state are not known in advance.
- Construct a two dimensional (for states and actions) table to learn the utility of all states and the optimal policy.
  - One viable way to do this is to construct a “dictionary of dictionary” in python.
  - Another way is to create a dictionary with a tuple (state, action).

Question 4~7 – Q Learning Agent

- "**qlearningAgents.py**"
- __init__(self, **args):
  - Construct your Qtable here.
- getQValue(self, state, action):
  - If the state is already seen, return Qtable(state, action)
  - Otherwise, you should initialize the elements to 0 for these keys in Qtable
  - **util.Counter** may be helpful

Question 4~7 – Q Learning Agent

- getValue(self, state):
  - If there are no legal actions, return 0
  - Otherwise, return
    $$\max_{action\ in\ A(state)} \text{Qtable(state, action)}$$
  - Note:
    - Please be advised to use the function “getQValue” instead of directly accessing the data in the Qtable here.
- getAction(self, state)
- update(self, state, action, nextState, reward):

  Update the Qtable

Question 8 – Approximate Q Learning Agent

- Motivation: the original Q learning method is not scalable.
- Extract the features of the state-action pair and learn the "**weights**" of the features instead.
- You only have to initialize the weights (you can use util.Counter) and override two functions "**getQValue**" and "**update**" according to the equations in the html file.
- You might need to call the function "getFeatures" defined in "**featureExtractors.py**".

Submission

- Please use .zip or .gz file (no .rar or anything else) to package the files you need to submit (i.e. valueIterationAgents.py, qlearningAgents.py, analysis.py) directly (don't create any folder).
- Verify your uploaded file by downloading it on ceiba
- Check the deadline carefully

Deadline

- 2019/1/2 27:00 (2019/1/3 03:00)
- Allow late submission until 2019/1/9 27:00