COL 870 (Reinforcement Learning)

Assignment 1

Due Date: ~~Wednesday October 1 + 2 Buffer Days~~.

Friday October 4, Midnight.

Weightage: 7.5%

Language allowed: Python. This assignment is to be done individually and all code is to be written from scratch using core packages like numpy, pandas etc and not using any existing implementations available online (In case you want to use any specific package check with us first on Piazza)

Achieve 31

We would like to implement and play a Game similar to the Blackjack given in Sutton and Barto (5.3) with the following differences:

RULES of the GAME.

1. We now have two types of colored cards - black and red. Black cards are drawn with a probability ⅔ and red cards are drawn with a probability of ⅓.
2. The cards are assumed to be drawn from an infinite deck of cards, with values ranging from 1 to 10. We assume that each card value is equally likely.
3. The (**black**) cards showing 1, 2 and 3 are special cards and can take special values (described below). Note that only the black cards showing 1,2 and 3 are special. All the red cards take their normal values. The player has the flexibility of using the special cards at their original value or at a much higher value of 11, 12 and 13 respectively, depending on current requirements. Further, we will assume that for each special type (1,2 or 3), only the first card of that type will be allowed the special status, after which the cards would behave normally (as explained in the class). Finally, you can assume that each player is rational in using the value of these cards towards achieving the goal.

4. Black cards contribute positively value towards the total sum. Red cards contribute negatively towards the total sum. For example, having two black cards with values 5,6 and a red card with value 7, will result in a total sum of 5+6-7=4.
5. The goal of the game is to reach a sum value which is as close to 31 as possible. Each player can either stick (stop) or hit (continue playing). If the sum of cards of a player goes negative or if it goes more than 31, then the player goes bust and the player loses.
6. Both the agent and dealer start with one card in their hands. The dealer's card is visible to the agent.
7. Agent starts the game and follows a policy. Once it sticks (or goes bust in which case it loses), the dealer starts.
8. The dealer follows a fixed policy: keep on playing until you can reach a sum of 25 or more. Once this happens, the dealer sticks.
9. The agent wins the game if (1) None of the player goes bust and agent's total sum is more than the dealer's sum in the end (2) Dealer goes bust after agent having played successfully (i.e., not going bust). Agent loses if it goes bust. If none of the above happens, that game is a draw.
10. Special Circumstances:
    a. The game is a draw if both the dealer and the agent have a negative card in the beginning (both of them would have gone bust).
    b. The player wins the game automatically if in the beginning (a) the player has a positive card AND (b) the dealer has a negative card (and hence goes bust right away).

Q1. State Space

Describe how will you represent the state space in the above problem? Clearly, describe the set of valid states that can be reached and the states which are non-actionable, i.e., agent has a single choice of action to take assuming it behaves rationally (i.e., taking any other action in these states is guaranteed to result in a lower total reward). You do not have to represent these states explicitly in your modeling below. In the description below, when we refer to a state, we will implicitly assume that we are referring to the set of actionable states. You can write the description directly in your report.

Q2. Simulator

Implement a simulator for this game, i.e., one which would model the environment and move to the next state based on the action taken by the agent. The moves of the dealer

are simulated as part of the environment model. In the following, we will assume a Model-free setting, i.e., agent does not know the environment model and can act only based on its past experience. In particular your implementation should be a class that has (at least) two public APIs as follows (taking example of python for syntax):

A. reset(): Resets the environment and returns initial state of the environment.

```
def reset():
      stuff()
      return initial_state
```

B. step(): Takes an action and moves to next_state. Returns next_state, reward associated with the transition and done representing end of episode.

```
def step(action):
      action_based_stuff()
      return next_state, reward, done
```

Q3 Policy Evaluation (Model Free)

1. Assume that the agent follows the following strategy: keep on playing until it can reach a sum of 25 or more using its current cards. Once this happens, stick. Note that this policy is identical to the Dealer's policy.
2. Use the following methods to evaluate the Q-value function (for each state action pair) using the following methods:
    a. Monte Carlo - both first-visit and every visit.
    b. k-step TD with values of k going from 1, 3, 5, 10, 100 and 1000. Repeat this experiment after taking averages over 100 runs and 1000 ~~10000~~ runs in each case. What do you observe?

For the part 2 above, you should use diagram(s) similar to the ones used by Sutton and Barto for plotting the value-function. Use X-Y plane to denote the current state, and Z-axis to denote the value of each state.

You can use \alpha=0.1 for the above parts.

Q4 Policy Control

1. Implement the following methods to learn the optimal policy:

    (a) k-step lookahead SARSA. Use an epsilon-greedy policy with a fixed value of epsilon = 0.1. Use the values of k=1, 10, 100, 1000.

    (b) k-step lookahead SARSA. Use an epsilon-greedy policy. Starting with a value of epsilon = 0.1, gradually decay it inversely proportional to the number of iterations (i.e., one update corresponds to one iteration). Use the values of k=1, 10, 100, 1000 as earlier.

    (c) Q-learning with a 1-step greedy look-ahead policy for update and epsilon-greedy policy for exploration in the environment with a fixed epsilon value of 0.1.

    (d) Forward view of the eligibility traces for TD(0.5) using on-policy control. Decay epsilon inversely proportional to the number of iterations starting with a value of 0.1.

2. Run each of your algorithms for 100 episodes. Use alpha (learning rate) = 0.1. For each algorithm, plot the performance (on a single graph) with number of episodes on x-axis and reward (averaged over 10 different runs) on y-axis. Which algorithm is able to learn the fastest? Comment.
3. Next train each of your algorithms for 100,000 episodes. For each algorithm, generate a set of 10 new (test) episodes by playing the game using your trained algorithm. ~~Generate a new test set of 10 different episodes by playing with your learned algorithm in each case~~. Compute the performance of each of the algorithms averaged over this test set. Use alpha (learning rate) = 0.1. Next, repeat the entire experiment by learning the models for alpha values in the set {0.1, 0.2, 0.3, 0.4, 0.5}. Plot the average test performance on y-axis with alpha value on x-axis. What do you observe? Does alpha values have any impact on the final learned model?
4. Plot the value function for each actionable state for your optimal policy obtained by algorithm in 1(d) above (i.e., TD(0.5) with decaying epsilon). Use alpha - 0.1. How different is the optimal state value function from the one observed using the fixed policy described earlier in the Policy Evaluation section. Comment on your observations.

Include any additional comments/observations from your experimentation in your final report.

NOTE: All the results/graphs that you produce should be made part of a single report. You should report with diligence and not simply cut-paste graphs. For each graph/result that you obtain include at least a few lines describing the result/observations. You should ensure replicability of results - we should be able to reproduce your graphs/other results when we run your code on our system (modulo any stochastic behavior).

In your evaluation, 10% of your score will be determined based on the quality of your report and the viva/demo at the end of the assignment submission.