CS 711 - Assignment 2

Due: Nov 17, 2024 (11:59 PM SGT)

Bonus Points: Any novel insights on the algorithm or the best results will get bonus points (2.5 points per question).

Q1. (Programming Exercise - 15 points)

Imagine you're designing a recommendation system for an online store that sells five different types of electronics (P1, P2, P3, P4, and P5). Each product has an unknown probability of being "purchased" when recommended to a customer. Your goal is to maximize the store's total sales by optimizing which product to recommend in each session.
The true purchase probabilities for each product are hidden, but they are randomly set as follows:

true_probabilities = np.random.rand(5)

When a product is recommended, a purchase is simulated based on its true probability. If purchased, it adds 1 unit to total sales; otherwise, no sale is made.

Implement and test each of the following recommendation algorithms to see how they maximize sales over 1,000 recommendation rounds:

- Greedy Approach
- Epsilon-Greedy (with epsilon = 0.1)
- Softmax Selection
- Thompson Sampling
- Upper Confidence Bound (UCB)

Use these steps to implement and evaluate the algorithms:

- Write code for each approach, recommending one of the five products in each round.
- Track the cumulative sales achieved and the selection frequency for each product.
- At the end, visualize the cumulative sales over time to compare algorithm performance.

Evaluation
After implementation, answer these questions based on your results:

1. Which algorithm achieved the highest cumulative sales?
2. How did exploration-exploitation strategies like epsilon-greedy and UCB perform compared to the greedy approach?
3. Which method would you recommend for a real store setting, especially if customers' preferences vary over time?

Q2. (Programming Exercise - 10 points)

Implement the Pong example described in class (Lecture 10) using the Gym or Gymnasium environment using both DQN and Reinforce. In the class, I explained how Reinforce is implemented on Pong. Here the target is for you to implement both DQN and Reinforce. Compare and contrast the training and testing performance of DQN and Reinforce.

Please provide the code and the results.

Note: You are free to study and explore other implementations, but you are expected to write your own code similar to what was described in class.

Q3. (By Hand or Programming – 10 points)

Consider a Restless Multi-Arm Bandit Problem, where there are four arms, all having the same MDP model.

States: 0, 1

Actions: 0 (passive), 1 (active)

Transition Probabilities:

Action = 0	0	1
0	0.6	0.4
1	0.3	0.7

Action = 1	0	1
0	0.4	0.6
1	0.15	0.85

Rewards, R(s): R(0) = 0; R(1) = 1

Horizon is 2 and Discount factor is 1. **Compute Whittle Index for each of the arms at the first-time step** (i.e., one more decision to go after the current decision), if they are in the following states:

Arm 1, State 1

Arm 2, State 0

Arm 3, State 1

Arm 4, State 0

Q4. (Programming – 10 points)

In this exercise, you will build and train an agent to navigate a complex grid world with dynamic elements, such as prizes, repair stations, and transient monsters.

The environment consists of:

- A 5x5 grid with 25 possible starting locations for the agent.
- Four corner locations where a prize can randomly appear, awarding the agent 10 points when collected.
- The possibility of monsters appearing temporarily on specific locations, damaging the agent if it's on the same square. If damaged and not repaired, the agent receives a reward of -10 for each turn.

- A repair station (R) where the agent can recover from being damaged.

Each state in this environment is represented by:

- **X, Y**: The agent's coordinates on the grid.
- **P**: The location of the prize (0 if it's on the top-left corner, 1 for top-right, 2 for bottom-left, 3 for bottom-right, and 4 if no prize is present).
- **D**: A boolean indicating if the agent is currently damaged.

With this setup, the environment has **250 fully observable states** (5 x 5 x 5 x 2), where the agent is always aware of its current state but has no prior knowledge of the environment's rules or rewards.

The agent has four actions:

- **Up, Down, Left, Right**, which move it by one grid cell in the specified direction, unless blocked by an obstacle (outer or inner walls), in which case it receives a reward of -1.

Tasks

1. **Build the Environment**
    - Implement a simulator that replicates the described grid world behavior:
        - The grid layout, including wall obstacles.
        - Prize behavior: It randomly appears in one of the four corners with a given probability when not collected.
        - Monster behavior: Monsters appear transiently at specific grid cells (M locations).
        - Damage and repair: If the agent is damaged, it needs to reach the repair station (R) to recover and avoid additional penalties.
    - Test your simulator by running a few random trajectories and displaying the agent's state, actions, and rewards at each step.
2. **Q-Learning in the Simulator**

   Implement Q-learning to train the agent on the simulator. Start by defining the following:

    - *State and Action Spaces*: Explicitly list all states and actions.
    - *Learning Parameters*: Set values for the learning rate, discount factor, and exploration rate.

   Train your agent to learn an optimal policy, and observe its performance by measuring cumulative reward over episodes. Answer the following:

    - What is the average cumulative reward per episode after training?
    - Explain any trends in the agent's behavior as it learns.
3. **Approximate Q-Learning (5 points)**

   In this task, you will apply approximate Q-learning using a set of features to generalize across states.

    - Define relevant features for approximate Q-learning. Consider features that capture essential elements of the environment, such as:
        - Distance to the nearest prize or repair station.

    - Whether the agent is damaged.
    - Proximity to monster-prone locations.
  - Implement approximate Q-learning with the defined features, training the agent on your simulator.
  - Compare the performance of approximate Q-learning with standard Q-learning by measuring cumulative rewards and policy effectiveness over episodes.

Note: Submit your code along with a brief report that includes the answers to each sub-question, code snippets where relevant, and plots of cumulative reward over training episodes for each method.

Q5. (5 points)

Given the following game between two players (Player 1 and Player 2), with Player 1 being the row player (i.e., controls the 2 row actions) and Player 2 being the column player (i.e., controls the 2 column actions).

	A	B
A	5, 5	0,7
B	7, 0	1,1

If the following matrix P is a potential function for the above game, what should be the value of "x". Explain your answer.

	A	B
A	0	x
B	x	3