EXPERIMENT-3 - E0123014

**! pip install gym gym\_bandits numpy matplotlib**

**import gym**

**import gym\_bandits import numpy as np**

**import matplotlib.pyplot as plt import warnings**

**from typing import List, Tuple, Optional**

**# Configuration constants NUM\_EPISODES: int = 500 EPSILON: float = 0.1**

**FIGURE\_SIZE: Tuple[int, int] = (14, 5)**

**# Suppress gym warnings for cleaner output warnings.filterwarnings("ignore", category=DeprecationWarning) warnings.filterwarnings("ignore", category=UserWarning)**

**class MultiArmedBandit: """**

**Multi-Armed Bandit implementation using epsilon-greedy strategy.**

**This class implements the epsilon-greedy algorithm for solving the multi-armed bandit problem using OpenAI Gym's bandit environment. """**

**def**  **init** **(self, env\_name: str = 'MultiarmedBandits-v0', epsilon: float = 0.1) -> None: """**

**Initialize the Multi-Armed Bandit solver.**

**Args:**

**env\_name: Name of the gym environment**

**epsilon: Exploration rate for epsilon-greedy strategy**

**"""**

**try:**

**self.env = gym.make(env\_name) except Exception as e:**

**print(f"Error creating environment: {e}") print("Trying alternative environment name...") try:**

**self.env = gym.make('BanditTenArmedHighLowFixed-v0') except Exception:**

**raise RuntimeError("Could not create bandit environment. Please check gym\_bandits installation.")**

**self.epsilon = epsilon**

**self.n\_arms = self.env.action\_space.n self.num\_episodes = NUM\_EPISODES**

**# Initialize tracking variables**

**self.q\_values = np.zeros(self.n\_arms) # Estimated Q-values self.counts = np.zeros(self.n\_arms) # Action selection counts**

**self.reward\_sums = np.zeros(self.n\_arms) # Cumulative rewards per arm self.all\_rewards: List[float] = [] # Episode rewards**

**self.action\_history: List[int] = [] # Action selection history**

**def select\_action(self) -> int: """**

**Select action using epsilon-greedy strategy.**

**Returns:**

**Selected action index """**

**if np.random.rand() < self.epsilon:**

**return self.env.action\_space.sample() # Random exploration else:**

**return np.argmax(self.q\_values) # Greedy exploitation**

**def update\_q\_values(self, action: int, reward: float) -> None:**

**"""**

**Update Q-values using incremental average.**

**Args:**

**action: Action that was taken reward: Reward received**

**"""**

**self.counts[action] += 1 self.reward\_sums[action] += reward**

**# Update Q-value: incremental average calculation self.q\_values[action] = self.reward\_sums[action] / self.counts[action]**

**def train(self) -> None: """**

**Train the agent using epsilon-greedy strategy.**

**Time Complexity: O(n\_episodes)**

**Space Complexity: O(n\_arms + n\_episodes) """**

**for episode in range(self.num\_episodes):**

**# Handle different gym versions for reset try:**

**obs = self.env.reset()**

**if isinstance(obs, tuple):**

**obs = obs[0] # New gym versions return (obs, info) except Exception:**

**obs = self.env.reset()**

**# Select action using epsilon-greedy policy action = self.select\_action()**

**# Take action and observe reward - handle different return formats try:**

**step\_result = self.env.step(action) if len(step\_result) == 4:**

**observation, reward, done, info = step\_result elif len(step\_result) == 5:**

**observation, reward, done, truncated, info = step\_result else:**

**observation, reward, done = step\_result[:3] info = {}**

**except Exception as e:**

**# Fallback for problematic environments try:**

**observation, reward, done = self.env.step(action)[:3] info = {}**

**except Exception:**

**print(f"Error in step {episode}: {e}") continue**

**# Ensure reward is a number if hasattr(reward, 'item'):**

**reward = float(reward.item()) else:**

**reward = float(reward)**

**# Update statistics self.update\_q\_values(action, reward)**

**# Record for analysis self.all\_rewards.append(reward) self.action\_history.append(action)**

**self.env.close()**

**def get\_statistics(self) -> dict: """**

**Get training statistics and results.**

**Returns:**

**Dictionary containing training statistics """**

**return {**

**'q\_values': np.round(self.q\_values, 2), 'counts': self.counts.astype(int),**

**'average\_reward': round(np.mean(self.all\_rewards), 3),**

**'total\_reward': round(np.sum(self.all\_rewards), 3), 'best\_arm': np.argmax(self.q\_values), 'exploration\_rate': self.epsilon**

**}**

**def plot\_results(self) -> None: """**

**Plot training results: cumulative rewards and action selection frequency. """**

**plt.figure(figsize=FIGURE\_SIZE)**

**# Plot 1: Cumulative reward over time plt.subplot(1, 2, 1)**

**cumulative\_rewards = np.cumsum(self.all\_rewards) plt.plot(cumulative\_rewards, color='blue') plt.title("Cumulative Reward over Episodes") plt.xlabel("Episode")**

**plt.ylabel("Cumulative Reward")**

**# Plot 2: Action selection frequency plt.subplot(1, 2, 2)**

**plt.bar(np.arange(self.n\_arms), self.counts, color='green') plt.title("Action Selection Frequency")**

**plt.xlabel("Arm Index") plt.ylabel("Times Selected")**

**plt.tight\_layout() plt.show()**

**def main() -> None: """**

**Main function to run the multi-armed bandit experiment. """**

**try:**

**# Create and configure the bandit solver**

**bandit = MultiArmedBandit(epsilon=EPSILON)**

**# Display environment information print("Environment Information:")**

**print(f"Observation Space: {bandit.env.observation\_space}") print(f"Action Space: {bandit.env.action\_space}") print(f"Number of Arms: {bandit.n\_arms}") print(f"Episodes: {NUM\_EPISODES}")**

**print(f"Exploration Rate: {EPSILON}") print("-" \* 50)**

**# Train the agent print("Training the agent...") bandit.train()**

**# Display results**

**stats = bandit.get\_statistics() print("\nTraining Results:")**

**print(f"Estimated Q-values for each arm: {stats['q\_values']}") print(f"Number of times each arm was selected: {stats['counts']}") print(f"Average reward over all episodes: {stats['average\_reward']}") print(f"Total cumulative reward: {stats['total\_reward']}") print(f"Best performing arm: {stats['best\_arm']}")**

**# Plot results bandit.plot\_results()**

**except Exception as e:**

**print(f"Error during execution: {e}")**

**return 1**

**return 0**

**if**  **name** **== "** **main** **": exit(main())**

**OUTPUT**



