RL - K Arm Bandit for Recommendation Systems

Course	🤖 Reinforcement Learning
Date	@August 25, 2024
Type	Assignment
Status	Completed
Submitted Time	@August 26, 2024 10:09 AM

K - Arm / Multi-Arm Bandits (MABs)

MABs represent a simplified version of the RL problem. The name comes from imagining a gambler at a row of slot machines (bandits), deciding which machines to play to maximize their total reward. There are several levers which a gambler can pull, with each lever giving a different return. The probability distribution for the reward corresponding to each lever is different and is unknown to the gambler. (Source)

Key Concepts:

- **Arms:** Each arm represents a possible action or choice.
- **Rewards:** After selecting an arm, the agent receives a numerical reward.

Common Strategies:

- **ε-greedy:** Choose the best-known arm most of the time, but occasionally (with probability ε) choose a random arm.

- **Upper Confidence Bound (UCB):** Select arms based on their potential to be optimal, considering both the expected reward and the uncertainty.

MABs in Recommendation Systems

Goal → Suggest items that the user would prefer to engage in

Approach

- Arms are the items that can be recommended
- Rewards are clicks based on the arms (need to check with the dataset)
- User - Article State requires Contextual Bandits
- Different models can be maintained for different user segments / users

Implementing ϵ-Greedy

Logic : Github Link

```
Parameters :
- n arms (int) : Number of Arms
- epsilon (float) : Explore Probablity (Randomness)
- Q0 (float) : Initial Arm Value
```

```
import numpy as np
import pandas as pd

n_arms = 10  # Arms
n_events = 1000  # Plays
epsilon = 0.1  # Exploration probability

def simulate_user_engagement(item_conversion_rate):
    return np.random.rand() < item_conversion_rate
```

```
####################

# epsilon-greedy
def epsilon_greedy(true_rewards, arms, num_iterations, epsilon)
    num_items = len(true_rewards)
    q_values = np.zeros(num_items)
    n_pulls = np.zeros(num_items)
    total_rewards = []

    for _ in range(num_iterations):
        if np.random.rand() < epsilon: # Exploration
            selected_item = np.random.choice(num_items)
        else: # Exploitation
            selected_item = np.argmax(q_values) # Arm with highe

        # Update rewards
        reward = simulate_user_engagement(true_rewards[selected_
        total_rewards.append(reward)

        n_pulls[selected_item] += 1
        q_values[selected_item] += (reward - q_values[selected_i

    return total_rewards, q_values, n_pulls


####################

# Generate dataset
np.random.seed(0)
true_rewards = np.random.rand(n_arms)  # True rewards probabilit
arms = np.random.randint(0, n_arms, size=n_events)
rewards = np.array([np.random.binomial(1, true_rewards[arm]) for
```

```
####################
total_rewards, q_values, n_pulls = epsilon_greedy(true_rewards,

####################
# Final estimated conversion rates
estimated_conversion_rates = q_values

####################
print("\nEstimated Conversion Rates:")
for i, (rate, count) in enumerate(zip(estimated_conversion_rates
    print(f"Item {i + 1}: Estimated Rate = {rate:.2f}, Selection
```

Output

```
Estimated Conversion Rates:
Item 1: Estimated Rate = 0.59, Selections = 22.0
Item 2: Estimated Rate = 0.71, Selections = 14.0
Item 3: Estimated Rate = 0.60, Selections = 53.0
Item 4: Estimated Rate = 0.64, Selections = 11.0
Item 5: Estimated Rate = 0.50, Selections = 6.0
Item 6: Estimated Rate = 0.57, Selections = 7.0
Item 7: Estimated Rate = 0.50, Selections = 14.0
Item 8: Estimated Rate = 0.90, Selections = 615.0
Item 9: Estimated Rate = 0.97, Selections = 250.0
Item 10: Estimated Rate = 0.13, Selections = 8.0
```