Thompson Sampling for Ad Optimization

Introduction

Thompson Sampling is a Bayesian approach to solving the multi-armed bandit problem, where the objective is to maximize the total reward by balancing exploration and exploitation. In this example, we use Thompson Sampling to optimize the selection of advertisements based on their click-through rates (CTR).

Code Explanation

Importing Libraries

First, we import the necessary libraries: numpy for numerical operations, pandas for data manipulation and analysis, matplotlib.pyplot for plotting, and random for generating random numbers.

```
python
Copy code
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
```

Loading the Dataset

We load the dataset containing the click-through rates (CTR) of different ads. Each row in the dataset represents a round, and each column represents an ad.

```
python
Copy code
dataset = pd.read_csv('Ads_CTR.csv')
```

Initializing Parameters

Several parameters are initialized:

- **Total Rounds**: The total number of rounds (iterations).
- **Total Ads**: The total number of ads.
- **selected_ads**: A list to store the index of the selected ad in each round.
- **reward_zero_counts**: A list to store the number of times each ad received a reward of 0 (not clicked).
- **reward_one_counts**: A list to store the number of times each ad received a reward of 1 (clicked).
- **total_reward**: A variable to store the cumulative reward.

```
python
Copy code
Total_Rounds = 1000
Total_Ads = 10
```

```
selected_ads = []
reward_zero_counts = [0] * Total_Ads
reward_one_counts = [0] * Total_Ads
total_reward = 0
```

Thompson Sampling Algorithm

Loop Over Each Round

We start a loop that iterates over each round (from 0 to Total_Rounds-1). For each round:

- `current_ad` is initialized to 0 to store the index of the selected ad.
- `max_random_value` is initialized to 0 to store the highest beta value among all ads in the current round.

```
python
Copy code
for n in range(0, Total_Rounds):
    current_ad = 0
    max_random_value = 0
```

Loop Over Each Ad

For each ad, we generate a random beta value using the `betavariate` function from the `random` module. The beta distribution parameters are `reward_one_counts[i] + 1` and `reward_zero_counts[i] + 1`. We compare the generated beta value with `max_random_value`. If it is higher, we update `max_random_value` and set `current_ad` to the current ad index `i`.

```
python
Copy code
    for i in range(0, Total_Ads):
        random_beta = random.betavariate(reward_one_counts[i] + 1,
reward_zero_counts[i] + 1)
        if random_beta > max_random_value:
            max_random_value = random_beta
            current_ad = i
```

Selecting the Ad and Updating Rewards

After selecting the ad with the highest beta value, we append the selected ad index to `selected_ads`. We get the reward of the selected ad from the dataset for the current round. We update the `reward_one_counts` or `reward_zero_counts` list based on the received reward. We update the `total_reward` by adding the received reward.

```
python
Copy code
    selected_ads.append(current_ad)
    current_reward = dataset.values[n, current_ad]
    if current_reward == 1:
        reward_one_counts[current_ad] += 1
    else:
        reward_zero_counts[current_ad] += 1
    total_reward += current_reward
```

Visualization

Finally, we plot a histogram of the ad selections to visualize how often each ad was selected over the rounds.

```
python
Copy code
plt.hist(selected_ads)
plt.title('Histogram of ad selections')
plt.xlabel('Ads')
plt.ylabel('Number of times each ad was selected')
plt.show()
```

Detailed Explanation of the First Few Iterations

Round 1

Iteration:

$n = 0$
assume there is only 3 ads

Sample θi from Beta distribution for each ad i:

- For ad 0: $\theta 0$ = Beta(1, 1) ≈ 0.75 (random draw)
- For ad 1: $\theta 1$ = Beta(1, 1) ≈ 0.60 (random draw)
- For ad 2: $\theta 2$ = Beta(1, 1) ≈ 0.45 (random draw)

Select the ad with the highest θi:

- Ad 0

Update:

- Suppose the reward for ad 0 is 1 (i.e., user clicked the ad)
- Update counts:
    - Number_of_rewards_1 = [1, 0, 0]
    - Number_of_rewards_0 = [0, 0, 0]
- Update total reward:
    - total_reward = 1
- Store selected ad:
    - ads_selected = [0]

Round 2

Iteration:

$n = 1$

Sample θi from Beta distribution for each ad i:

- For ad 0: $\theta 0$ = Beta(2, 1) ≈ 0.90 (random draw)
- For ad 1: $\theta 1$ = Beta(1, 1) ≈ 0.50 (random draw)
- For ad 2: $\theta 2$ = Beta(1, 1) ≈ 0.65 (random draw)

Select the ad with the highest θi:

- Ad 0

Update:

- Suppose the reward for ad 0 is 0 (i.e., user did not click the ad)
- Update counts:
    - Number_of_rewards_1 = [1, 0, 0]
    - Number_of_rewards_0 = [1, 0, 0]
- Total reward remains the same:
    - total_reward = 1
- Store selected ad:
    - ads_selected = [0, 0]

Round 3

Iteration:

$n = 2$

Sample θi from Beta distribution for each ad i:

- For ad 0: $\theta 0$ = Beta(2, 2) ≈ 0.55 (random draw)
- For ad 1: $\theta 1$ = Beta(1, 1) ≈ 0.70 (random draw)
- For ad 2: $\theta 2$ = Beta(1, 1) ≈ 0.40 (random draw)

Select the ad with the highest θi:

- Ad 1

Update:

- Suppose the reward for ad 1 is 1
- Update counts:
    - Number_of_rewards_1 = [1, 1, 0]
    - Number_of_rewards_0 = [1, 0, 0]
- Update total reward:
    - total_reward = 2

- Store selected ad:
  - ads_selected = [0, 0, 1]

Round 4

Iteration:

n = 3

Sample θi from Beta distribution for each ad i:

- For ad 0: $\theta 0$ = Beta(2, 2) ≈ 0.60 (random draw)
- For ad 1: $\theta 1$ = Beta(2, 1) ≈ 0.85 (random draw)
- For ad 2: $\theta 2$ = Beta(1, 1) ≈ 0.50 (random draw)

Select the ad with the highest θi:

- Ad 1

Update:

- Suppose the reward for ad 1 is 0
- Update counts:
  - Number_of_rewards_1 = [1, 1, 0]
  - Number_of_rewards_0 = [1, 1, 0]
- Total reward remains the same:
  - total_reward = 2
- Store selected ad:
  - ads_selected = [0, 0, 1, 1]

Summary after 4 Rounds

- Ad 0 has been selected twice, with rewards [1, 0].
- Ad 1 has been selected twice, with rewards [1, 0].
- Ad 2 has not been selected yet.

The algorithm continues in this manner, balancing exploration and exploitation by sampling from the Beta distribution and updating beliefs based on observed rewards.