kbandit

May 4, 2022

1 Introduction

In this Jupyter notebook, we explore the k-bandit problem using three different approaches: random selection, epsilon-greedy, and upper confidence band (UCB) methods.

[30]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
```

[31]:
```
df = pd.read_csv('Ads_Optimisation.csv')
df.head()
```

[31]:

	Ad 1	Ad 2	Ad 3	Ad 4	Ad 5	Ad 6	Ad 7	Ad 8	Ad 9	Ad 10
0	1	0	0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0

[41]:
```
df.shape
```

[41]: (10000, 10)

1.1 Random Selection

[45]:
```
N = 10000
d = 10
ads_selected = []
total_reward = 0
```

[46]:
```
for n in range(N):
    ad = random.randrange(d)
    ads_selected.append(ad)
    reward = df.values[n, ad]
    total_reward += reward
```

[69]:
```
rand_results = pd.Series(ads_selected).value_counts(normalize=True)
print(f'total reward: {total_reward}')
print(rand_results)
```

```
total reward: 3846
4    0.48910
7    0.07995
0    0.06040
9    0.05535
3    0.05415
8    0.05365
1    0.05350
2    0.05250
6    0.05085
5    0.05055
dtype: float64
```

1.2 Epsilon Greedy

We compute the action value function, $q_t(a)$, for all arms at each timestep t. Our goal is to choose the action which will maximize q_t at each step.

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{1}$$

where R_i is the reward.

To optimize the computation, we can keep a running average:

$$\begin{aligned} Q_t(a) &= \frac{Q_{t-1}(a)N_t(a_t) + R_t \cdot \mathbb{1}_{A_t=a}}{N_t(a_t)} \qquad (2) \\ &= Q_{t-1}(a) + \frac{R_t - Q_{t-1}(a)}{N_t(a_t)} \qquad (3) \end{aligned}$$

[170]:
```
N = 10000
d = 10
ads_selected = []
# action-value quantities
numbers_of_selections = [0] * d
sums_of_reward = [0] * d
avg_reward = [0] * d
total_reward = 0
# exploration probability
epsilon = 0.5
```

[171]:
```
for n in range(N):
    if random.uniform(0, 1) < epsilon:
        # explore
        ad = random.randrange(d)
```

```
    else:
        # greedy
        for i in range(d):
            if numbers_of_selections[i] > 0:
                avg_reward[i] = sums_of_reward[i] / numbers_of_selections[i]
        ad = np.argmax(avg_reward)

    ads_selected.append(ad)
    numbers_of_selections[ad] += 1
    reward = df.values[n, ad]
    sums_of_reward[ad] += reward
    total_reward += reward
```

[172]:
```
eps_greedy_results = pd.Series(ads_selected).value_counts(normalize=True)
print(f'total reward: {total_reward}')
print(eps_greedy_results)
```

```
total reward: 1993
4    0.5461
0    0.0530
1    0.0527
3    0.0522
5    0.0505
8    0.0504
2    0.0503
7    0.0494
9    0.0492
6    0.0462
dtype: float64
```

Upon exploration, the epsilon-greedy method is more accurate for a moderate to high value of epsilon, i.e., epsilon above 0.1. If epsilon is low, then the greedy method may choose an incorrect value, i.e., not enough exploration.

1.3 Upper Confidence Bound (UCB)

[15]:
```
import math
ads_selected = []
numbers_of_selections = [0] * d
sums_of_reward = [0] * d
total_reward = 0
```

1. Play each of K actions once to obtain initial values for mean rewards corresponding to each action.
2. For each round t = K:
   (a) Let $N_t(a)$ denote # times an action was played.

(b) Play the action at maximizing the following equation

$$\text{UCB1} = Q(a) + \sqrt{\frac{2 \ln t}{N_t(a)}} \tag{4}$$

*note that the 2 in the above equation can be any constant

3. Observe reward and update mean reward for the chosen action.

[67]:
```python
for n in range(N):
    ad, max_ub = 0, 0
    for i in range(d):
        if (numbers_of_selections[i] > 0):
            avg_reward = sums_of_reward[i] / numbers_of_selections[i]
            delta_i = math.sqrt(2*math.log(n+1) / numbers_of_selections[i])
            # updating UCB
            ub = avg_reward + delta_i
        else:
            # setting initial UCB
            ub = 1e400
        if ub > max_ub:
            max_ub = ub
            ad = i
    ads_selected.append(ad)
    numbers_of_selections[ad] += 1
    reward = df.values[n, ad]
    sums_of_reward[ad] += reward
    total_reward += reward
```

[70]:
```python
ucb_results = pd.Series(ads_selected).value_counts(normalize=True)
print(f'total reward: {total_reward}')
print(ucb_results)
```

```
total reward: 3846
4    0.48910
7    0.07995
0    0.06040
9    0.05535
3    0.05415
8    0.05365
1    0.05350
2    0.05250
6    0.05085
5    0.05055
dtype: float64
```

1.4 Conclusion

Clearly ad 5 (index 4) has the highest total reward proportion, which is the true value in this specific dataset. Further, UCB was a much better method than random selection and epsilon-greedy selection.