

▼ Project 1: Bandit problem

- Developer: Shaun Pritchard
- Date: Feb 6, 2022
- Class: CAP 6629: Reinforcement Learning
- Prof. Ni, Zehn

References:

- [1] Sutton, Richard S. and Barto, Andrew G. Chapter 2: Multi-Armed Bandits from Reinforcement Learning. pps. 25-46. Kindle Edition. 2018.
- [2] Ni, Zhen. Multi-Armed Bandits. Lecture notes. pps. 8, 10. 2022.

Objective:

- **Part 1:** Read chapter 2 and use any programming language to implement a multi-arm Bandit problem. Using the reward distribution for each action $q^*(a)$ as shown in lecture notes page 10.
- **Part 2:** Apply the algorithm in part 1 to a provided Ad dataset to determine frequency of selected ads and test greedy and ϵ -greedy parameters.
- **Part 3:** Analyze and compare results then publish to PDF.

▼ Test and Parameters

With four different epsilon values, we will perform multi K-Armed Bandit experiments. Our objectives are to perform 2000 -3000 iterations with 10000 timesteps each compared to the following random, greedy and ϵ -greedy parameters.

- $\epsilon = 1$
- $\epsilon = 0.1$
- $\epsilon = 0.01$
- $\epsilon = 0$

$\epsilon = 1$ is a completely random agent, $\epsilon = 0.1$ is the ϵ -greedy implementation at value 0.1, $\epsilon = 0.01$ is the ϵ -greedy implementation at value 0.01, $\epsilon = 0$ is the reward values averaged over N iterations for each timestep, and then plotted.

Pseudo Code K-armed bandits Algorithm

```
Initialize, for  $a = 1$  to  $k$ :
   $Q(a) \leftarrow 0$ 
   $N(a) \leftarrow 0$ 

Loop forever:
   $A \leftarrow \begin{cases} \arg \max_a Q(a) & \text{with probability } 1 - \epsilon \text{ (breaking ties randomly)} \\ \text{a random action} & \text{with probability } \epsilon \end{cases}$ 
   $R \leftarrow \text{bandit}(A)$ 
   $N(A) \leftarrow N(A) + 1$ 
   $Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$ 
```

▼ Importing the libraries

```
1 # Import libraries
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import matplotlib as mpl
6 from matplotlib import cm
7 import pandas as pd
8 import random
```

▼ Importing the dataset

```
1 # Import data For K-armed bandit problem part 2
```

```
2 df = pd.read_csv('/content/Ads_Optimisation.csv')
3 df.head(10) # validate data import
```

| | 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 |
| 5 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



▼ Global Variables

```
1 Q = np.zeros(10) # Initialize q-value array
2 N = np.zeros(10) # Initialize counting array to track each action
3 i = 2000 # iterations
4 s = 1000 # Steps in (t)
```

▼ Part 1

Follow the algorithm pseudo code (page 8 of lecture note). The reward distributions are provided on page 10 and you need to estimate the mean value of each action yourself. Please show 1) your average reward curves of different ϵ values; and 2) percent of optimal actions (similar figures as we studied in the class).

- Use 2000 or 3000 for your iteration
- Compare the trajectories of $\epsilon=0.1$ and $\epsilon=0.01$ over time

▼ Algorithm 1

The kArmedBandit Algorithm takes the mean reward distribution for each given action A, returns R from the normal reward distribution of A, and then updates N and Q values according to the normal reward distribution with mean $q^*(a)$.

```

1 # Calculate the mean of the reward distribution for each action  $q^*(a)$ 
2 # Acquired from Ni, Zhen. Multi-Armed Bandits. Lecture notes. pps. 8, 10. 2022.
3 _q = np.array((0.2, -1, 2.5, 0.6, 2, -1.7, -0.2, -1, 1, -0.3))

1 # K-Arm Bandit Algorithm
2 def kArmedBandit(itter, step_t, e_greedy):
3
4     Matrix_R = np.zeros((itter, step_t))
5
6     for i in range(0, itter):
7         for a in range(0, 10): # Initialize 0 to 10 reset
8             Q[a]=0 # Initialize Q array to 0
9             N[a]=0 # Initialize N array to 0
10        sum_R=0 # Initialize variable to 0
11        for j in range(0, step_t):
12            n = random.random() # Return the next random number in range
13            if n < 1 - e_greedy: # Exploit data with probability 1 - e_greedy
14                A = np.random.choice(np.flatnonzero(Q == Q.max())) # Random sample of Q-Max Array

15                R = np.random.normal(_q[A], 1) # Expected reward values array
16                N[A] +=1 # Update array N(A) <-- N(A)-1
17                Q[A] += 1/N[A]*(R-Q[A]) # Action value
18                sum_R += R

```

```

18         sum_R += Q[A] # Sum of action value rewards
19         Matrix_R[i][j]=sum_R/(j+1) # Average reward values matrix
20
21     else: # Explore data randomly
22         A = np.random.randint(0, 10) # Randomize next action
23         R = np.random.normal(_q[A], 1) # Expected reward values array
24         N[A] +=1 # Update array N(A) <-- N(A)-1
25         Q[A] += 1/N[A]*(R-Q[A]) # Action value
26         sum_R += Q[A] # Sum of action value rewards
27         Matrix_R[i][j] = sum_R/(j+1) # Average reward values matrix
28
29     return Matrix_R

```

```

1 # Compare the trajectories of \epsilon =0.1 and \epsilon=0.01 over time.

```

```

2 e_1 = kArmedBandit(i, s, 1)
3 e_01 = kArmedBandit(i, s, 0.1)
4 e_001 = kArmedBandit(i, s, 0.01)
5 e_reward = kArmedBandit(i, s, 0)

```

```

1 # Average results of compared epsilon trajectories

```

```

2 e_avg_1 = np.mean(e_1, axis=0)
3 e_avg_01= np.mean(e_01, axis=0)
4 e_avg_001 =np.mean(e_001, axis=0)
5 avg_rewards =np.mean(e_reward, axis=0)

```

```

1 # Part 1 Plot Optimal actions

```

```

2 sns.set(style="ticks")
3 plt.rcParams["figure.figsize"] = (10,5)
4 plt.plot(e_avg_1, label='epsilon=1%', linewidth = '4', c='r')
5 plt.plot(e_avg_01, label='epsilon=0.1%', linewidth = '4')
6 plt.plot(e_avg_001, label='epsilon=0.01%', linewidth = '4')

```

```

7 plt.plot(avg_rewards, label='epsilon=0%', linewidth = '4')
8 plt.title('Multi-Armed Bandit Optimal Actions | Problem #1')
9 plt.xlabel('Time Step (t)')

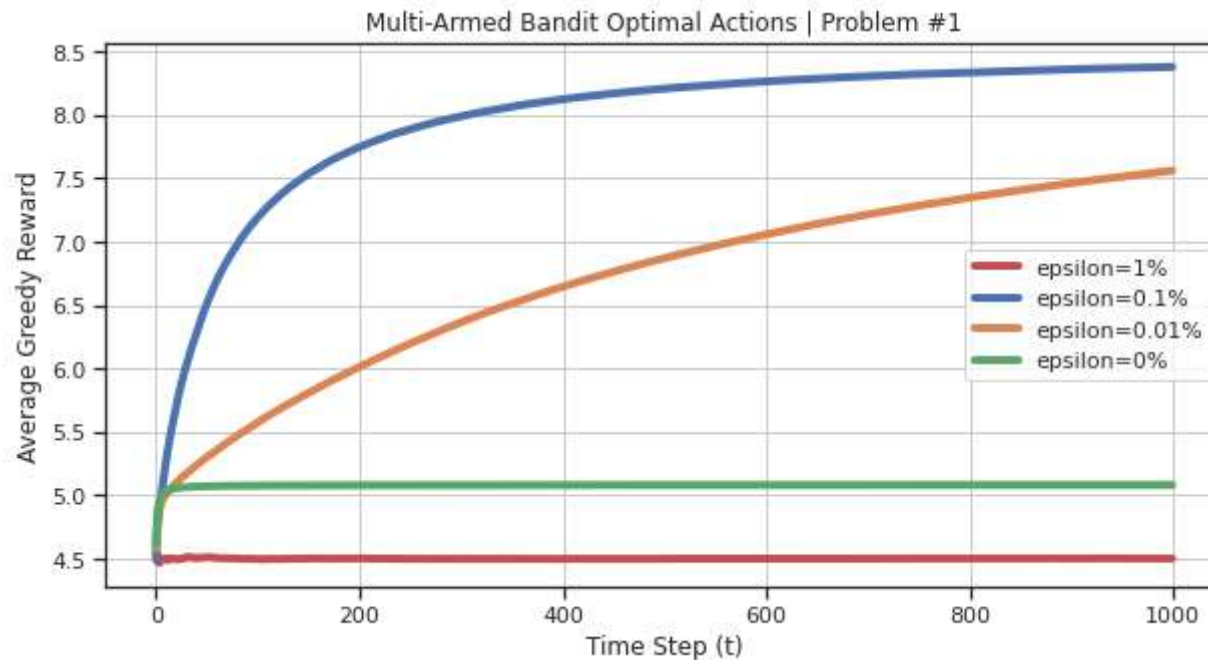
```

```

10 plt.legend()

```

```
10 plt.ylabel('Average Greedy Reward')
11 plt.grid()
12 plt.legend()
13 plt.show()
```



▼ Part 1: Results & Analysis

- $\epsilon=1$ results in the smallest possible reward from a pure random agent that produces a relatively constant reward level overall
- $\epsilon=0.1$ results Achieves a faster convergence, but only earns the second best overall average reward
- $\epsilon=0.01$ results consists of a slower convergence but highest average reward
- $\epsilon=0$ produces the lowest average reward by exploiting without any exploration

▼ Part 2

- Using the algorithm in part 1, apply it to the Ad dataset and with the full reward distribution to determine the optimal frequency of ad selection.
- Provide the maximum reward you can achieve with this dataset, and what is the best ad from the algorithm.

▼ Algorithm 2

The kArmedBandit_2 Algorithm takes the Ad reward values for each given action A, returns R from the normal reward distribution of A, and then updates N and Q values according to the normal reward distribution with mean $q^*(a)$.

```

1 # Generate reward values from Ad data
2 def reward_values(A):
3     reward = np.empty(10)
4     for i in range(0, 10):
5         reward[i]=df['Ad '+str(i+1)].sum()
6     return A

1 def kArmedBandit_2(itter, step_t, e_greedy):
2
3     Matrix_R = np.zeros((itter, step_t)) # Initilize empty reward matrix
4
5     for i in range(0, itter):
6         for a in range(0, 10): # Initialize 0 to 10 reset
7             Q[a]=0 # Initialize Q array to 0
8             N[a]=0 # Initialize N array to 0
9             sum_R = 0 # Initialize sum of reward variable to 0
10            for j in range(0, step_t):
11                n = random.random() # Return the next random number in range
12                if n < 1 - e_greedy: # Exploit data with probability 1 - e_greedy
13
14                    A=np.random.choice(np.flatnonzero(Q == Q.max())) # Random sample of Q-Max Array
15                    R=reward_values(A) # Expected reward values array
16                    N[A] +=1 # Update array N(A) <-- N(A)-1
17                    Q[A] += 1/(N[A]*((1-Q[A])/(1-e_greedy))) # A + 1/(N(A)*(1-Q(A)/(1-e_greedy)))

```

```

16         Q[A] += 1/N[A]^(R-Q[A]) # Action value
17         sum_R += Q[A] # Sum of action value rewards
18         Matrix_R[i][j]=sum_R/(j+1) # Average reward values matrix
19     else: # Explore data randomly with epsilon
20         A=np.random.randint(0, 10) # randome Action value
21         R=reward_values(A) # Expected reward values array
22         N[A] +=1 # Update array N(A) <-- N(A)-1
23         Q[A] += 1/N[A]*(R-Q[A]) # Action value
24         sum_R += Q[A] # Sum of action value rewards
25         Matrix_R[i][j]=sum_R/(j+1) # Average reward values matrix
26     return Matrix_R

```

```

1 # Compare the trajectories of \epsilon =0.1 and \epsilon=0.01 over time.
2 e_1 = kArmedBandit_2(i, s, 1)
3 e_01 = kArmedBandit_2(i, s, 0.1)
4 e_001 = kArmedBandit_2(i, s, 0.01)
5 e_reward = kArmedBandit_2(i, s, 0)

```

```

1 # Average results of compared epsilon trajectories
2 e_avg_1 = np.mean(e_1, axis=0)
3 e_avg_01= np.mean(e_01, axis=0)
4 e_avg_001 =np.mean(e_001, axis=0)
5 avg_rewards =np.mean(e_reward, axis=0)

```

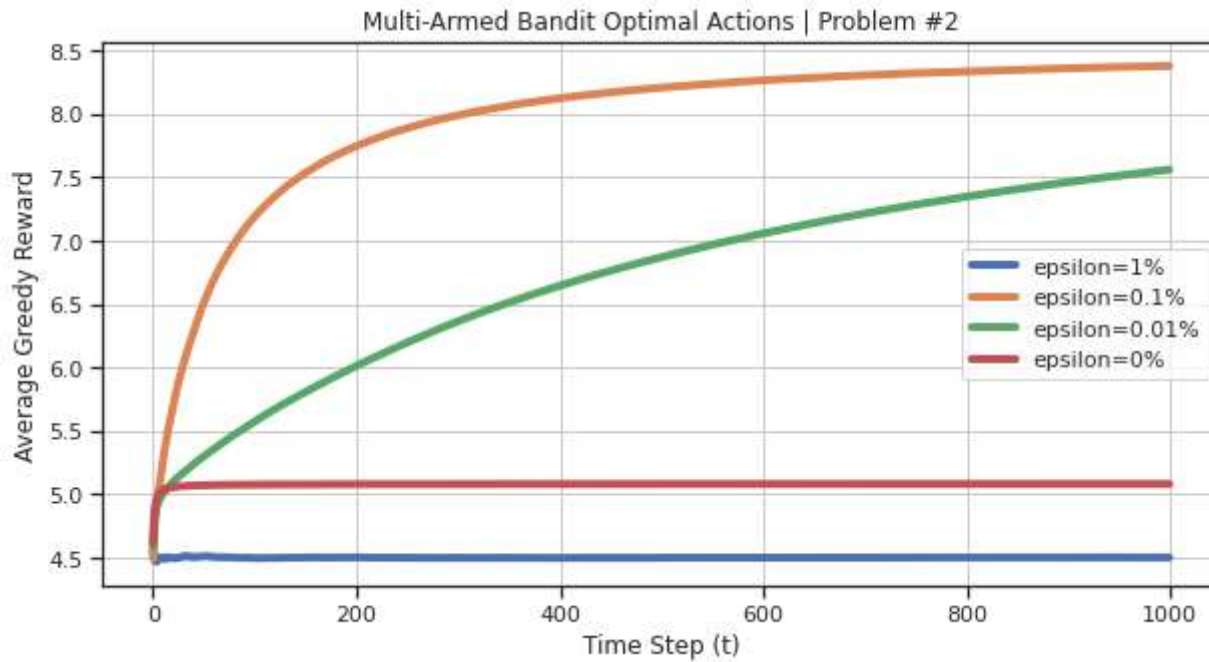
```

1 # Part 2 Plot Optimal actions
2 sns.set(style="ticks")
3 plt.rcParams["figure.figsize"] = (10,5)
4 plt.plot(e_avg_1, label='epsilon=1%', linewidth = '4')
5 plt.plot(e_avg_01, label='epsilon=0.1%', linewidth = '4')
6 plt.plot(e_avg_001, label='epsilon=0.01%', linewidth = '4')
7 plt.plot(avg_rewards, label='epsilon=0%', linewidth = '4')
8 plt.title('Multi-Armed Bandit Optimal Actions | Problem #2')
9 plt.xlabel('Time Step (t)')
10 plt.ylabel('Average Greedy Reward')
11 plt.show()

```



```
11 plt.grid()
12 plt.legend()
13 plt.show()
```



▼ Part 2: Final Results & Analysis

- $\epsilon=1$ results in the smallest possible reward from a pure random agent that produces the lowest average reward.
- $\epsilon=0.1$ results consist of a slower convergence, and best overall average reward.
- $\epsilon=0.01$ results consists of a median convergence with the 2nd highest average reward
- $\epsilon=0$ produces the 2nd lowest average reward by exploiting without any exploration

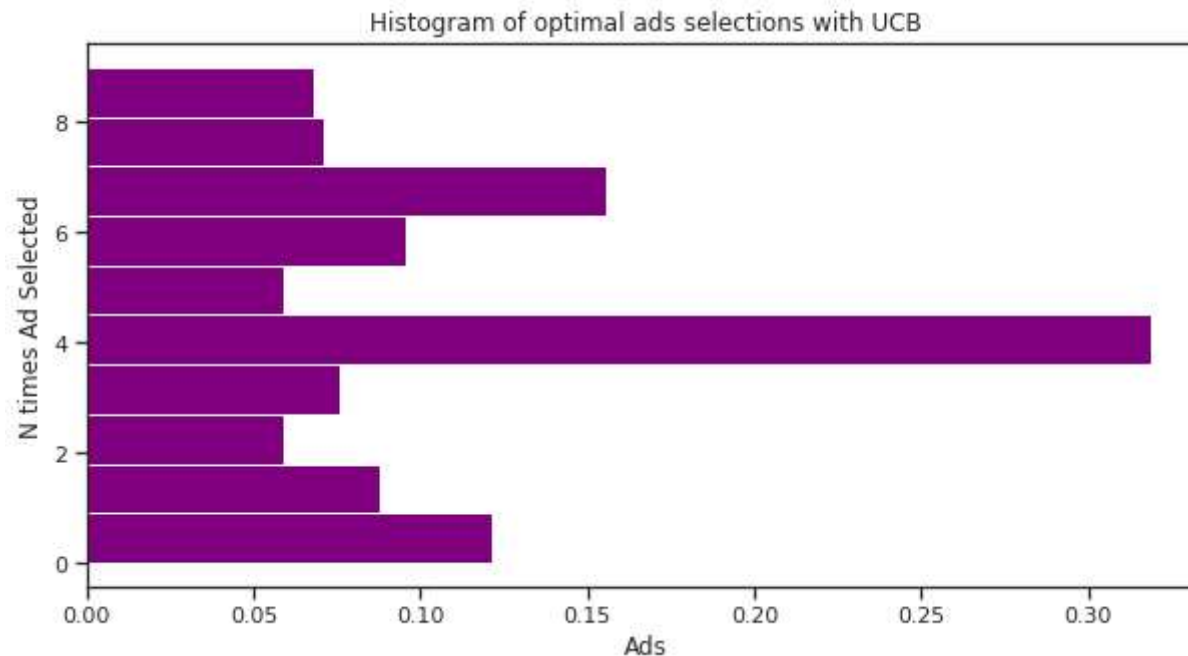
▼ Data with UCB

This instance implements the add data set with UCB algorithm. I wrote this code to practice and compare.

```
1 import math
2 N = 10000 # itterations
3 d = 10 # reset value
4 ads_selected = [] # Return Ads selected array
5 n = [0] * d # Number of Selections
6 sum_R = [0] * d # Sum of rewards
7 total_reward = 0 # Total rewards
8 for n in range(0, N):
9     ad = 0
10    max_UB = 0
11    for i in range(0, d):
12        if (n[i] > 0):
13            reward_avg = sum_R[i] / n[i]
14            delta_i = math.sqrt(3/2 * math.log(n + 1) / n[i])
15            UB = reward_avg + delta_i
16        else:
17            UB = 1e400 # Upper-bound
18        if (UB > max_UB):
19            max_UB = UB
20            ad = i
21    ads_selected.append(ad)
22    n[ad] = n[ad] + 1
23    reward = df.values[n, ad]
24    sum_R[ad] = sum_R[ad] + reward
25    total_reward = total_reward + reward

1 # UCB algorithm plot
2 sns.set(style="ticks")
3 plt.rcParams["figure.figsize"] = (10,5)
4 plt.hist(ads_selected, orientation='horizontal',color = "purple", density=True)
5 plt.title('Histogram of optimal ads selections with UCB')
6 plt.xlabel('Ads')
```

```
7 plt.ylabel('N times Ad Selected')
8 plt.show()
```



▼ UCB Results:

UCB implementation shows at 10000 iterations the optimal return action value selection through number of 4 ads selected with the highest frequency for the ad over 30% ad selection.

▼ Final Project Report & Analysis

Implementation Overview

Implementation Overview:

In each experiment, the algorithms were initializing the q-values of all actions to 0. Some actions have negative expected values within the first dataset $q^*(a)$. In the second implementation all actions have positive reward values.

Analysis Comparison

- **$\epsilon=1$** - The two algorithms implemented random actions over all timesteps (t). Compared to the epsilon greedy parameters, both implementations showed the lowest average reward. These algorithms show that performance is better than the greedy approach.
- **$\epsilon=0.1$** - Over all timesteps (t), both algorithms implemented random actions with a 10% probability of exploration. The results of each algorithm implementation showed the most positive returns. Additionally, the optimal action of algorithm 1 is action 3, while the optimal action of algorithm 2 is action 5. During the rest of the experiment, the agent will take the selected action 90% of the time, which leads to convergent behavior at those selected points.
- **$\epsilon=0.01$** - The implementations of algorithm 1 and algorithm 2 are both run at 1% probability of exploration, which is known to take longer to find the optimal action. As soon as it is discovered, it will be taken in 99% of cases, and eventually be superior to the 10% exploratory approach in the long run.
- **$\epsilon=0$** -
According to algorithm 1, it will randomly select actions until it finds one with a positive q-value, and algorithm 2 will run until it finds one with a nonzero q-value. Both implementations are greedy, and the averaging rewards results indicate that smaller nonzero epsilon values produce the best long-term rewards

Conclusion or Results:

- Generally, we find that $\epsilon=0.1$ proves more difficult to reach convergence, but results in the highest average returns.
- The speed of convergence will decrease as epsilon values decrease, but they are guaranteed to take the optimal action more in the long run.
- There is a lower average reward for lower epsilon values in both experiments Greedy approaches are only useful when they are expected to reward each action (positively or negatively).
- In the short term, greedy approaches will lead to convergent results, but are not guaranteed to generate the optimal outcome.

► Print Report PDF

[] ↪ 3 cells hidden

✓ 0s completed at 12:14 PM



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.