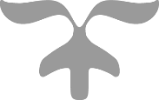
**21AIE311: REINFOREMENT LEARNING**



**multi-arm bandits**

**FOR ADVERTISEMENT CLICK THROUGH RATE OPTIMIZATION**



TEAM 11

CB.EN.U4AIE19014 – **ARUN PRAKASH J**

CB.EN.U4AIE19016 – **ASSWIN C R**

CB.EN.U4AIE19024 – **DHARSHAN KUMAR K S**

CB.EN.U4AIE19028 – **AVINASH DORA**

GUIDED BY

**DR. SOMAN K P**



BTECH CSE-AI

CEN DEPARTMENT

AMRITA SCHOOL OF ENGINEERING

**ACKNOWLEDGEMENT**

This project would not have been possible without the support and guidance of Dr. Soman K P, who has provided us with sufficient knowledge and guidelines.

We would also like to thank our Computer Science and Engineering (Artificial Intelligence) – CEN department for giving us this opportunity to improve our skills.

Furthermore, we would like to thank the Amrita Vishwa Vidyapeetham management for providing ample resources to avail our project needs.

Our thanks and appreciations also go to colleagues in developing the project and people who have willingly helped us out with their abilities unconditionally.

**TABLE OF CONTENTS**

1. [ABSTRACT](#_Toc43146547) 4

2. THEORITICAL BACKGROUND5

7. [RESULT AND DISCUSSION](#_Toc43146551) 11

8. [APPLICATIONS](#_Toc43146551) 14

9. [PYTHON CODE](#_Toc43146551) 15

10. [REFERENCES](#_Toc43146551) 19

**ABSTRACT**

Advertisement optimization refers to the process to categorizing advertisements specific to a certain group of the population. A few well-known techniques for advertisement optimization include A/B testing and greedy algorithms. This project deals with analyzing the performances of techniques such as epsilon greedy algorithm, Upper Confidence Bound (UCB) algorithm and Thompson Sampling. Experiments conclude that Thompson sampling outperforms the rest of the above-mentioned algorithms. However, when taking hyper-parameter tuning into consideration, epsilon greedy algorithm results in the lowest regret.

***Keywords -*** *Greedy Algorithms, Epsilon Greedy Algorithm, Upper Confidence Bound Algorithm, Thompson Sampling, Regret.*

**THEORITICAL BACKGROUND**

**Multi-arm Bandits:**

The problem of multi arm bandits arises from ways to increase the returns from a slot machine in a casino without losing much money. The term regret is often used to indicate the amount of money lost. The more money is lost, the more the regret. Multi arm bandits (MABs) can be either thought of using multiple machines each with one pull trigger or one machine with multiple pull triggers. The overall objective still remains the same: increase returns by finding the trigger which when pulled gives the highest returns (i.e. lowest regrets). In the context of reinforcement learning, the aim of the agent is to finalize a policy for choosing an action at each time step step such that the long term reward is maximized.

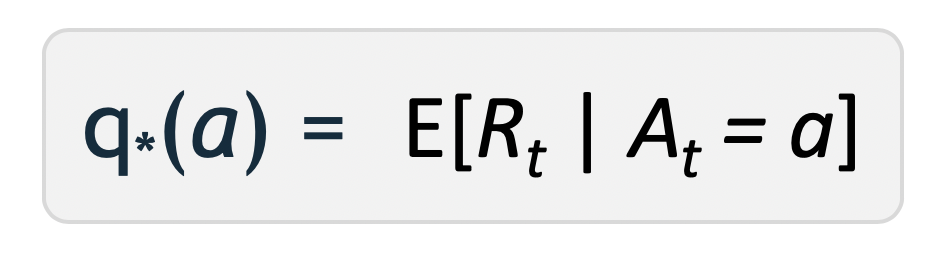
**Problem Statement:**

Whenever a user visits the website, the advertiser displays an ad at random. The advertiser then monitors whether the user clicks on the ad or not. After a while, the advertiser notices that one ad seems to be working better than the others. The advertiser must now decide between sticking with the best-performing ad or continuing with the randomized study.

If the advertiser only displays one ad, then he can no longer collect data on the other two ads. Perhaps one of the other ads is better, it only appears worse due to chance. If the other two ads are worse, then continuing the study can affect the click-through rate adversely. This advertising trial exemplifies decision-making under uncertainty.

In the above example, the role of the agent is played by an advertiser. The advertiser has to choose between three different actions, to display the first, second, or third ad. Each ad is an action. Choosing that ad yields some unknown reward. Finally, the profit of the advertiser after the ad is the reward that the advertiser receives.

**Action-Values:**

For the advertiser to decide which action is best, we must define the value of taking each action. We define these values using the action-value function using the language of probability. The value of selecting an action q\*(a) is defined as the expected reward Rt we receive when taking an action, a from the possible set of actions.

The goal of the agent is to maximize the expected reward by selecting the action that has the highest action-value.

**Exploration:**

It allows the agent to improve its knowledge about each action. Hopefully, leading to a long-term benefit.

**Exploitation:**

It allows the agent to choose the greedy action to try to get the most reward for short-term benefit. A pure greedy action selection can lead to sub-optimal behaviour.

**The Greedy Algorithm**

The idea of the greedy algorithm is to choose the action which results in the highest estimated returns for each time step. The action chosen at each time step can be mathematically expressed in equation 1.



This approach might seem like the best way to find the optimal policy; however, there exists limitations: the greedy algorithm may not choose the action that could have led to a better total reward. In the context of the advertisement optimization problem, we may end up selecting an advertisement which doesn’t have the best estimated reward. The epsilon greedy algorithm serves as a solution to this issue.

**Epsilon greedy algorithm**

In the epsilon greedy algorithm, we choose either exploration or exploitation based on a probability term ‘ε’ (Epsilon). The probability term is added to the greedy algorithm making the estimate increasingly accurate to its true reward value.



As described in the figure above, the idea behind a simple ε-greedy bandit algorithm is to get the agent to explore other actions randomly with a very small probability (ε) while at other times you go with selecting the action greedily. The action selected greedily is based on the one which gives the highest expected reward for the given time step. If the value of epsilon is zero, each arm is chosen with an equal probability thus becomes greedy algorithm. The random selection increases as epsilon increases and so does the exploration.

**Upper Confidence Bound Action Selection**



For any arm ‘a’ at time step ‘t’,

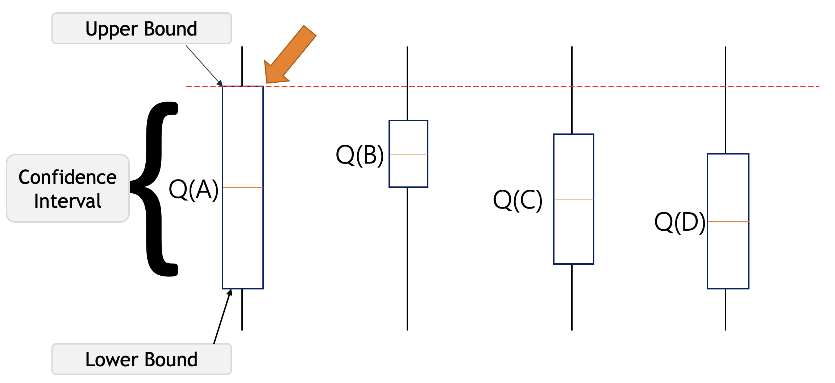
na -> No. of times arm ‘a’ was pulled

ra -> Empirical average of arm ‘a’

For the first ‘k’ rounds, we play each arm once and after that, we don’t play the greedy algorithm alone but also add an exploratory bonus.

If an arm is ignored, then the empirical estimate ‘ra’ does not change. But the timestep ‘t’ increases and so the exploratory bonus also increases. This increases the upper confidence interval value for that arm ‘a’ even though it is not frequently visited.

Likewise, if an arm is played frequently, then the empirical estimate ‘ra’ changes according to arm selected. Also, the arm selection count ‘na’ also increases along with the timestep ‘t’. This causes the exploratory bonus to reduce its value and influences us to explore more on the least visited arms.



If we are uncertain about an action, we should optimistically assume that it is the correct action.

According to the UCB algorithm, it will optimistically pick the action that has the highest upper bound.

i.e. By doing this either it will have the highest value and get the highest reward, or by taking that we will get to learn about an action we know least about.

UCB is very useful only in the case where the no. of rounds ‘T’ is much bigger than no. of arms ‘k’.

i.e. T >> k

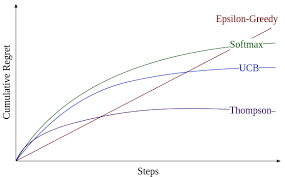
**Reason to use UCB:**

The Greedy algorithms will have a linear regret. The lower bound on the regret will be



So, it motivates us to use UCB which has lesser regret than log.





**Thompson Sampling**

Thompson sampling also referred as Bayesian bandit algorithm is a method of solving exploration-exploitation dilemma in a mutli-armed bandit problem.​It was first described by William R. Thompson in 1933. Thompson sampling allows to identify which option to pursue in order to maximize the expected reward. It does this by providing a optimal balance between exploration and exploitation. Thompson sampling aligns with all the core elements of reinforcement learning i.e, input and output system, rewards, environment, Markov decision process, and training. Hence it can be applied for solving numerous reinforcements learning problems. In this method during each round, we choose an option and record whether a reward has been received. Hence, we take a random draw from a beta distribution and pursue whichever option has the highest beta value.

**Beta Distribution:**

The Beta distribution is a probability distribution where the shape of the curve determines, how likely it is, that something will happen. From the above curve, if a number is chosen at random, the likelihood of getting 0.5 is higher than the number close to 0.2 and 0.8. It has 2 parameters called the shape parameters, i.e the value of ‘a’ and ‘b’ controls the shape of the curve. When both the values are the same, it depicts a symmetric plot. Depending upon the value of a and b, the value of beta and the shape of the curve gets changed.

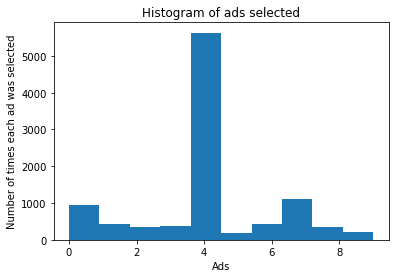
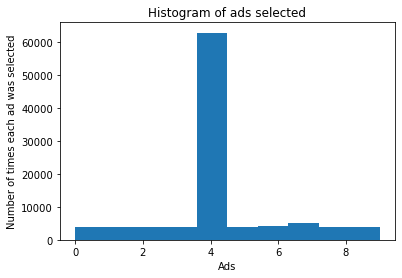
Mathematically,

P(θ | a, b) = (θ ^(a -1)) \*((1 — θ)^(b-1)) / B( a, b )

Here, B(a, b) is the normalization constant which is a Beta function of a and b. This denominator is not a function of the variable θ. It just ensures that probability distribution integrates to 1.

In Thompson sampling, the value of the shape parameter ‘a’, is the no of times we previously received the reward when that option was chosen and the value of the shape parameter ‘b’ represents the no of times that we did not receive the reward when that object was chosen. In simple terms, it is the no of past wins and ‘b’ is the no of past losses. Hence high wins correspond to higher beta value and history with a high proportion of losses tends to give a smaller beta value.

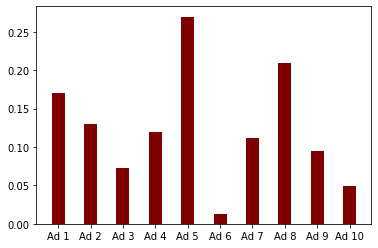
**RESULT AND DISCUSSION**



1. Chart

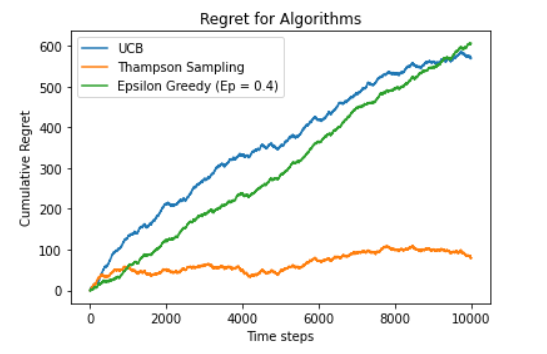
   Description automatically generatedEpsilon Greedy (b) UCB

(c) Thompson Sampling

By seeing these graphs, we can infer that Ad-4 is the most played arm.

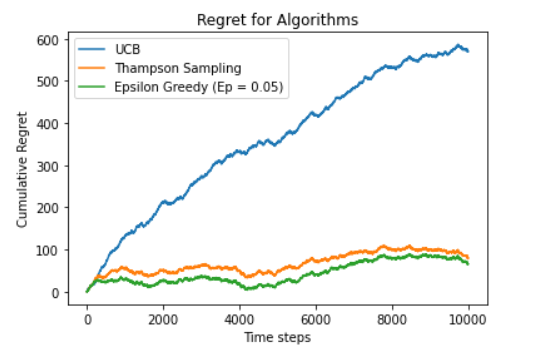
True reward distribution for each Ad is given is displayed in the above plot. We see that the optimal arm which the maximum rewards is Arm-4. So, we will have a low regret in all the cases.

But we can observe that the Thompson sampling has lower regret comparing to the other 2 algorithms. And thus, it is found that Thompson sampling is better to use and if we know the prior assumption of the empirical reward of the arms, then the algorithm would still converge faster.



From this graph, we clearly see that the epsilon greedy algorithm has a linear regret which is expensive for real-time scenarios. UCB has a log regret but still it converges slower than the Thompson sampling.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Total Reward** | **Empirical Regret** |
| Epsilon Greedy | 2146 | 0.0569 |
| UCB | 2125 | 0.0104 |
| Thompson Sampling | **2607** | **0.0096** |



But we noticed that for a lower value of epsilon (0.05), we observe that epsilon greedy has a lowest regret. But this may not be the case for other applications.

**APPLICATIONS**

1)     Facebook uses modified Thompson sampling called constrained Thompson sampling to optimize the quality of video in the video uploading process.

2)     Netflix uses Thompson sampling and with other bandit framework in recommendation systems.

3)     Bidding and Stock Exchange: Predicting Stocks based on Current data of stock prices.

4) Traffic Light Control: Predicting the delay in the signal.

5) Automation in Industries: Bots and Machines for transporting and Delivering items without human intervention.

**PYTHON CODE (APPENDIX)**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Ads\_Optimisation.csv')

**Epsilon Greedy:**

# Implementing epsilon greedy

import math

N = 10000

d = 10

ads\_selected = []

numbers\_of\_selections = [0] \* d

sums\_of\_reward = [0] \* d

average\_reward = list(dataset.values[0])

total\_reward = 0

epsilon = 0.05

for n in range(0, N):

ad = 0

for i in range(0, d):

random\_number = np.random.rand(1)

if(random\_number<epsilon):

random\_number = 1

else:

random\_number = 0

if random\_number == 1:

## go explore

j = np.random.choice(d)

ad = j

ads\_selected.append(ad)

numbers\_of\_selections[ad] += 1

reward = dataset.values[n, ad]

sums\_of\_reward[ad] += reward

average\_reward[ad] = sums\_of\_reward[ad] / numbers\_of\_selections[ad]

else:

## go exploit

j = np.argmax(average\_reward[-d:])

ad = j

ads\_selected.append(ad)

numbers\_of\_selections[ad] += 1

reward = dataset.values[n, ad]

sums\_of\_reward[ad] += reward

average\_reward[ad] = sums\_of\_reward[ad] / numbers\_of\_selections[ad]

total\_reward += reward

**UCB :**

# Implementing UCB

import math

N = 10000

d = 10

ads\_selected = []

numbers\_of\_selections = [0] \* d

sums\_of\_reward = [0] \* d

total\_reward = 0

for n in range(0, N):

ad = 0

max\_upper\_bound = 0

for i in range(0, d):

if (numbers\_of\_selections[i] > 0):

average\_reward = sums\_of\_reward[i] / numbers\_of\_selections[i]

delta\_i = math.sqrt(2 \* math.log(n+1) / numbers\_of\_selections[i])

upper\_bound = average\_reward + delta\_i

else:

upper\_bound = 1e400

if upper\_bound > max\_upper\_bound:

max\_upper\_bound = upper\_bound

ad = i

ads\_selected.append(ad)

numbers\_of\_selections[ad] += 1

reward = dataset.values[n, ad]

sums\_of\_reward[ad] += reward

total\_reward += reward

**Thompson Sampling:**

## Thomson

import random

N = 10000

d = 10

ads\_selected = []

numbers\_of\_rewards\_1 = [0] \* d

numbers\_of\_rewards\_0 = [0] \* d

total\_reward = 0

for n in range(0, N):

ad = 0

max\_random = 0

for i in range(0, d):

random\_beta = random.betavariate(numbers\_of\_rewards\_1[i] + 1, numbers\_of\_rewards\_0[i] + 1)

if random\_beta > max\_random:

max\_random = random\_beta

ad = i

ads\_selected.append(ad)

reward = dataset.values[n, ad]

if reward == 1:

numbers\_of\_rewards\_1[ad] = numbers\_of\_rewards\_1[ad] + 1

else:

numbers\_of\_rewards\_0[ad] = numbers\_of\_rewards\_0[ad] + 1

total\_reward = total\_reward + reward

**REFFERENCES**

[1] <https://medium.com/analytics-vidhya/multi-armed-bandit-analysis-of-epsilon-greedy-algorithm-8057d7087423>

[2] <https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/>

[3] <https://ekababisong.org/the-exploration-exploitation-trade-off/#:~:text=Exploration%20is%20when%20an%20Agent,favourable%E2%80%9D%20long%2Dterm%20rewards>.

[4] <https://towardsdatascience.com/the-exploration-exploitation-dilemma-f5622fbe1e82>

[5] <https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f>

[6] <https://www.geeksforgeeks.org/introduction-to-thompson-sampling-reinforcement-learning/>

[7] <https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf>

[8] <https://towardsdatascience.com/thompson-sampling-fc28817eacb8>