LOGIN / REGISTER (HTTPS://ID.ANALYTICSVIDHYA.COM/ACCOUNTS/LOGIN/?NEXT=HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2018/09/REINFORCEMENT-MULTI-ARMED-BANDIT-SCRATCH-PYTHON/)

Analytics Vidhya
Learn everything about analytics
(https://www.analyticsvidhya.com/blog/)

MACHINE LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/)
REINFORCEMENT LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/REINFORCEMENT-LEARNING/)

Reinforcement Learning Guide: Solving the Multi-Armed Bandit Problem from Scratch in Python

ANKIT CHOUDHARY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/ANKIT2106/), SEPTEMBER 24, 2018

Introduction

Do you have a favorite coffee place in town? When you think of having a coffee, you might just go to this place as you're almost sure that you will get the best coffee. But this means you're missing out on the coffee served by this place's cross-town competitor.

And if you try out all the coffee places one by one, the probability of tasting the worse coffee of your life would be pretty high! But then again, there's a chance you'll find an even better coffee brewer. But what does all of this have to do with reinforcement learning?

Cafe Coffee Day vs Starbucks

I'm glad you asked.

The dilemma in our coffee tasting experiment arises from incomplete information. In other words, we need to gather enough information to formulate the best overall strategy and then explore new actions. This will eventually lead to minimizing the overall bad experiences.

A multi-armed bandit is a simplified form of this analogy. It is used to represent similar kinds of problems and finding a good strategy to solve them is already helping a lot of industries.

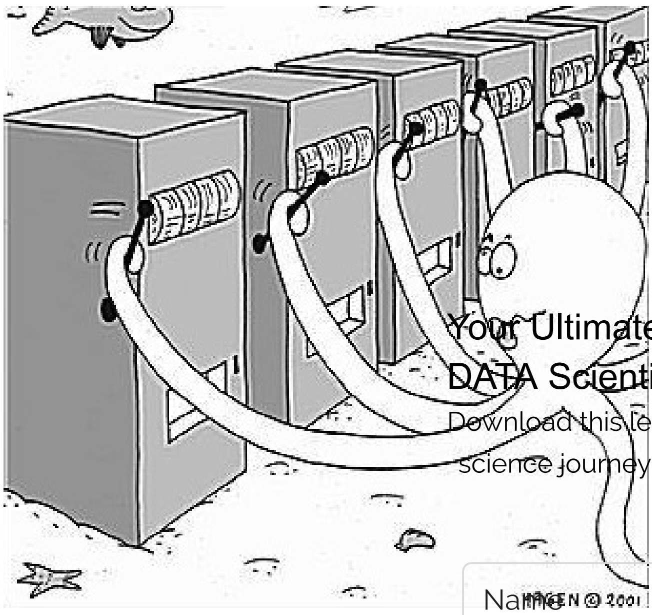
In this article, we will first understand what actually is a multi-armed bandit problem, it's various use cases in the real-world, and then explore some strategies on how to solve it. I will then show you how to solve this challenge in Python using a click-through rate optimization dataset.

Table of Contents

What is the Multi-Armed Bandit Problem (MABP)?

A bandit is defined as someone who steals your money. A one-armed bandit is a simple slot machine wherein you insert a coin into the machine, pull a lever, and get an immediate reward. But why is it called a bandit? It turns out all casinos configure these slot machines in such a way that all gamblers end up losing money!

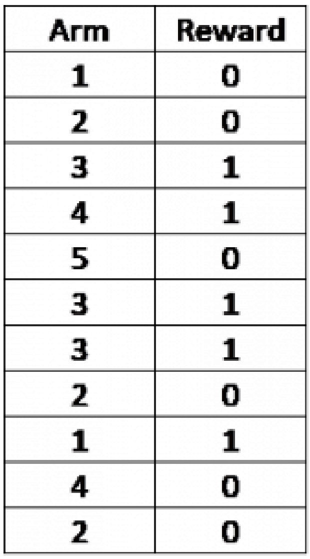
A multi-armed bandit is a complicated slot machine wherein instead of 1, there are several levers which a gambler can pull, with each lever giving a different return. The probability distribution for the reward corresponding to each lever is different and is unknown to the gambler.



The task is to identify which lever to pull in order to get maximum reward after a given set of trials. This problem statement is like a single step Markov decision process, which I discussed in this article (https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/). Each arm chosen is equivalent to an action, which then leads to an immediate reward.

Exploration Exploitation in the context of Bernoulli MABP

The below table shows the sample results for a 5-armed Bernoulli bandit with arms labelled as 1, 2, 3, 4 and 5:



Arm	Reward
1	0
2	0
3	1
4	1
5	0
3	1
3	1
2	0
1	1
4	0
2	0

This is called Bernoulli, as the reward returned is either 1 or 0. In this example, it looks like the arm number 3 gives the maximum return and hence one idea is to keep playing this arm in order to obtain the maximum reward (pure exploitation).

Just based on the knowledge from the given sample, 5 might look like a bad arm to play, but we need to keep in mind that we have played this arm only once and maybe we should play it a few more times (exploration) to be more confident. Only then should we decide which arm to play (exploitation).

Use Cases

Bandit algorithms are being used in a lot of research projects in the industry. I have listed some of their use cases in this section.

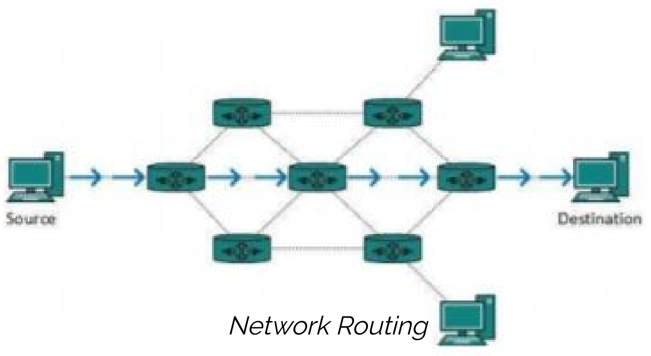
Clinical Trials

The well being of patients during clinical trials is as important as the actual results of the study. Here, exploration is equivalent to identifying the best treatment, and exploitation is treating patients as effectively as possible during the trial.

Clinical Trials

Network Routing

Routing is the process of selecting a path for traffic in a network, such as telephone networks or computer networks (internet). Allocation of channels to the right users, such that the overall throughput is maximised, can be formulated as a MABP.



Network Routing

Online Advertising

The goal of an advertising campaign is to maximise revenue from displaying ads. The advertiser makes revenue every time an offer is clicked by a web user. Similar to MABP, there is a trade-off between exploration, where the goal is to collect information on an ad's performance using click-through rates, and exploitation, where we stick with the ad that has performed the best so far.

Online Ads

Game Designing

Building a hit game is challenging. MABP can be used to test experimental changes in game play/interface and exploit the changes which show positive experiences for players.

Game Designing

Solution Strategies

In this section, we will discuss some strategies to solve a multi-armed bandit problem. But before that, let's get familiar with a few terms we'll be using from here on.

Action-Value Function

The expected payoff or expected reward can also be called an action-value function. It is represented by q(a) and defines the average reward for each action at a time t.

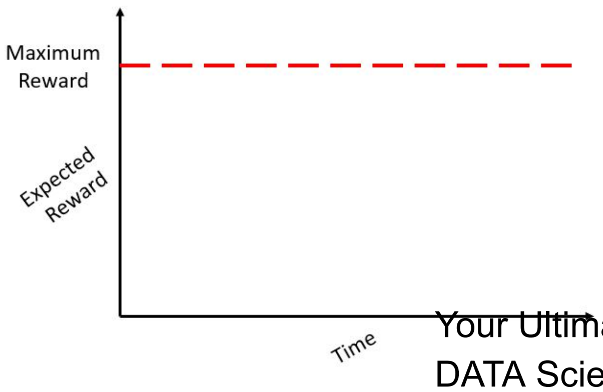
$$Q(a) = \mathbb{E}[r|a]$$

Suppose the reward probabilities for a K-armed bandit are given by {P1, P2, P3 ...... Pk}. If the *ith* arm is selected at time t, then Qt(a) = Pi.

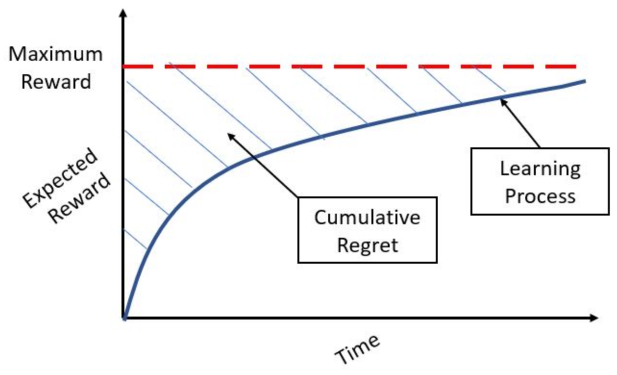
The question is, how do we decide whether a given strategy is better than the rest? One direct way is to compare the total or average reward which we get for each strategy after *n* trials. If we already know the best action for the given bandit problem, then an interesting way to look at this is the concept of regret.

Regret

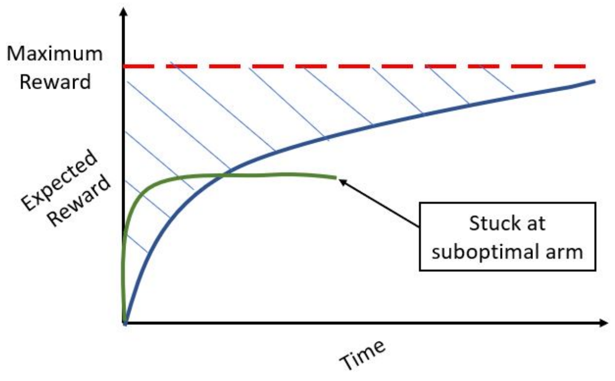
Let's say that we are already aware of the best arm to pull for the given bandit problem. If we keep pulling this arm repeatedly, we will get a maximum expected reward which can be represented as a horizontal line (as shown in the figure below):



But in a real problem statement, we need to make repeated trials by pulling different arms till we am approximately sure of the arm to pull for maximum average return at a time *t*. **The loss that we incur due to time/rounds spent due to the learning is called regret.** In other words, we want to maximise my reward even during the learning phase. Regret is very aptly named, as it quantifies exactly how much you regret not picking the optimal arm.



Now, one might be curious as to how does the regret change if we are following an approach that does not do enough exploration and ends exploiting a suboptimal arm. Initially there might be low regret but overall we are far lower than the maximum achievable reward for the given problem as shown by the green curve in the following figure.



Based on how exploration is done, there are several ways to solve the MABP. Next, we will discuss some possible solution strategies.

No Exploration (Greedy Approach)

A naïve approach could be to calculate the q, or action value function, for all arms at each timestep. From that point onwards, select an action which gives the maximum q. The action values for each action will be stored at each timestep by the following function:

$$Q_t(a) = \frac{\sum_{i=1}^{t} 1_{(a_t=a)} R_i}{\sum_{i=1}^{t} 1_{(a_t=a)}}$$

It then chooses the action at each timestep that maximises the above expression, given by:

$$argmax_a \, Q_t(a)$$

However, for evaluating this expression at each time t, we will need to do calculations over the whole history of rewards. We can avoid this by doing a running sum. So, at each time t, the q-value for each action can be calculated using the reward:

$$Q_t(a) = \frac{Q_{t-1}(a)N_t(a_t) + R_t.1_{(a_t=a)}}{N_t(a_t)}$$

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{N_t(a_t)}(R_t - Q_{t-1}(a))$$

The problem here is this approach only exploits, as it always picks the same action without worrying about exploring other actions that might return a better reward. Some exploration is necessary to actually find an optimal arm, otherwise we might end up pulling a suboptimal arm forever.

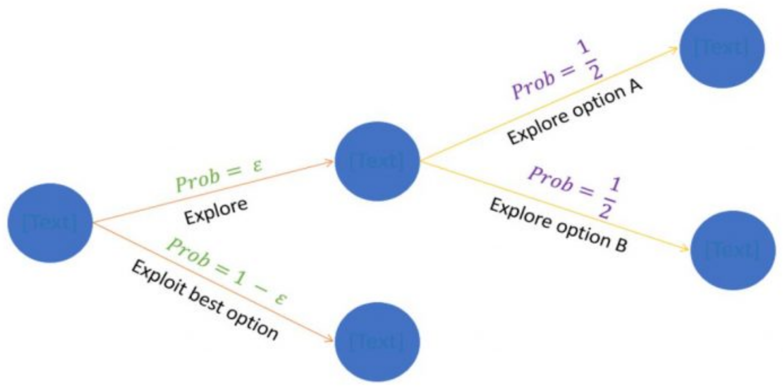
Epsilon Greedy Approach

One potential solution could be to now, and we can then explore new actions so that we ensure we are not missing out on a better choice of arm. With epsilon probability, we will choose a random action (exploration) and choose an action with maximum $q_t(a)$ with probability 1-epsilon.

With probability 1- epsilon – we choose action with maximum value (argmaxa Qt(a))

With probability epsilon – we randomly choose an action from a set of all actions A

For example, if we have a problem with two actions – A and B, the epsilon greedy algorithm works as shown below:



This is much better than the greedy approach as we have an element of exploration here. However, if two actions have a very minute difference between their q values, then even this algorithm will choose only that action which has a probability higher than the others.

Softmax Exploration

The solution is to make the probability of choosing an action proportional to q. This can be done using the softmax function, where the probability of choosing action *a* at each step is given by the following expression:

$$P(a_t = a) = \frac{e^{Q_t(a)}}{\sum_1^n e^{Q_t(b)}}$$

Decayed Epsilon Greedy

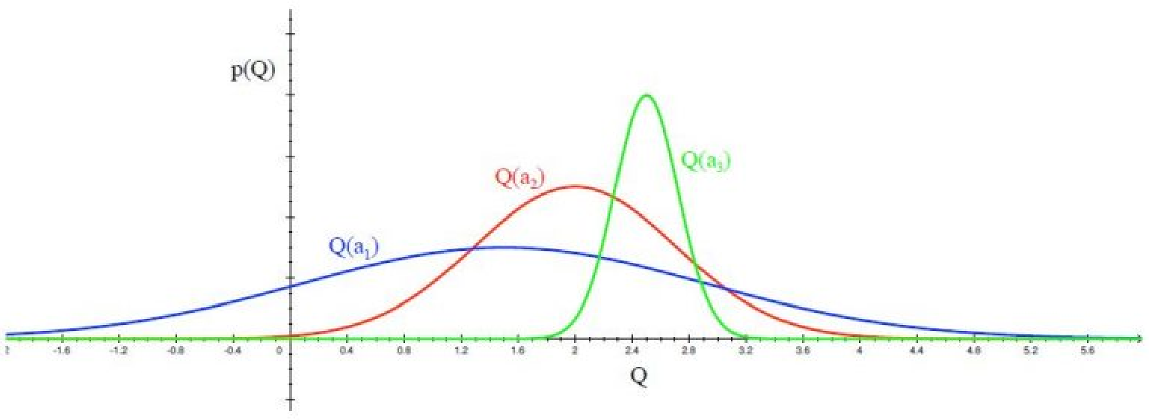
The value of epsilon is very important in deciding how well the epsilon greedy works for a given problem. We can avoid setting this value by keeping epsilon dependent on time. For example, epsilon can be kept equal to *1/log(t+0.00001)*. It will keep reducing as time passes, to the point where we starting exploring less and less as we become more confident of the optimal action or arm.

The problem with random selection of actions is that after sufficient timesteps even if we know that some arm is bad, this algorithm will keep choosing that with probability *epsilon/n*. Essentially, we are exploring a bad action which does not sound very efficient. The approach to get around this could be to favour exploration of arms with a strong potential in order to get an optimal value.

Upper Confidence Bound

Upper Confidence Bound (UCB) is the most widely used solution method for multi-armed bandit problems. This algorithm is based on the principle of optimism in the face of uncertainty.

In other words, the more uncertain we are about an arm, the more important it becomes to explore that arm.



- Distribution of action-value functions for 3 different arms a1, a2 and a3 after several trials is shown in the figure above. This distribution shows that the action value for a1 has the highest variance and hence maximum uncertainty.
- UCB says that we should choose the arm a_1 and receive a reward making us less uncertain about its action-value. For the next trial/timestep, if we still are very uncertain about a_1, we will choose it again until the uncertainty is reduced below a threshold.

The intuitive reason this works is that when acting optimistically in this way, one of two things happen:

- optimism is justified and we get a positive reward which is the objective ultimately
- the optimism was not justified. In this case, we play an arm that we believed might give a large reward when in fact it does not. If this happens sufficiently often, then we will learn what is the true payoff of this action and not choose it in the future.

UCB is actually a family of algorithms. Here, we will discuss UCB1.

Steps involved in UCB1:

- *Play each of the K actions once, giving initial values for mean rewards corresponding to each action at*

- *For each round t = K:*
- *Let $N_t(a)$ represent the number of times action a was played so far*
- *Play the action at maximising the following expression:*

$$Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

- *Observe the reward and update the mean reward or expected payoff for the chosen action*

We will not go into the mathematical proof for UCB. However, it is important to understand the expression that corresponds to our selected action. Remember, in the random exploration we just had Q(a) to maximise, while here we have two terms. First is the action value function, while the second is the confidence term.

- Each time *a* is selected, the uncertainty is presumably reduced: $N_t(a)$ increments, and, as it appears in the denominator, the uncertainty term decreases.

$$N_t(a) \uparrow \qquad \sqrt{\frac{2 \log t}{N_t(a)}} \downarrow$$

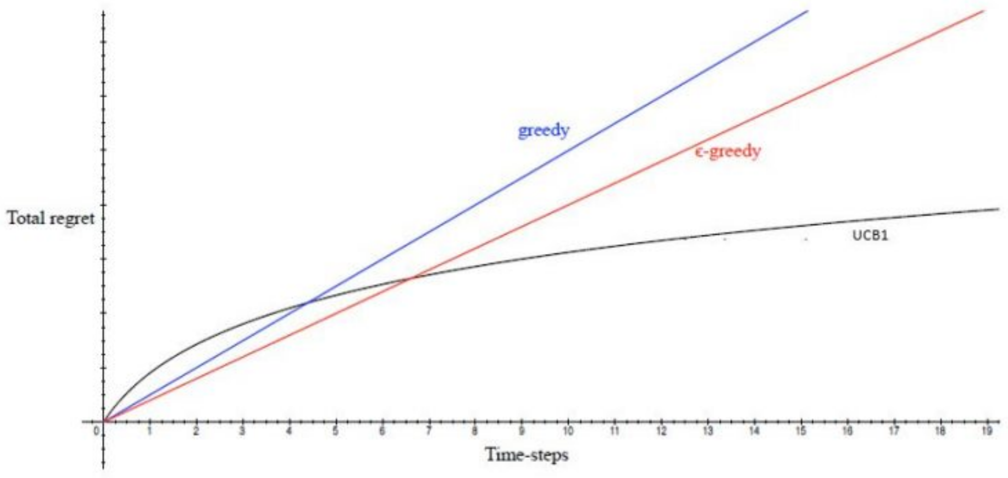
- On the other hand, each time an action other than *a* is selected, t increases, but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.

$$t \uparrow \qquad \text{With } N_t(a) \text{ constant} \qquad \sqrt{\frac{2 \log t}{N_t(a)}} \uparrow$$

- The use of the natural logarithm means that the increases get smaller over time; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.
- This will ultimately lead to the optimal action being selected repeatedly in the end.

Regret Comparison

Among all the algorithms given in this article, only the UCB algorithm provides a strategy where the regret increases as log(t), while in the other algorithms we get linear regret with different slopes.



Non-Stationary Bandit problems

An important assumption we are making here is that we are working with the same bandit and distributions from which rewards are being sampled at each timestep stays the same. This is called a stationary problem. To explain it with another example, say you get a reward of 1 every time a coin is tossed, and the result is head. Say after 1000 coin tosses due to wear and tear the coin becomes biased then this will become a non-stationary problem.

To solve a non-stationary problem, more recent samples will be important and hence we could use a constant discounting factor alpha and we can rewrite the update equation like this:

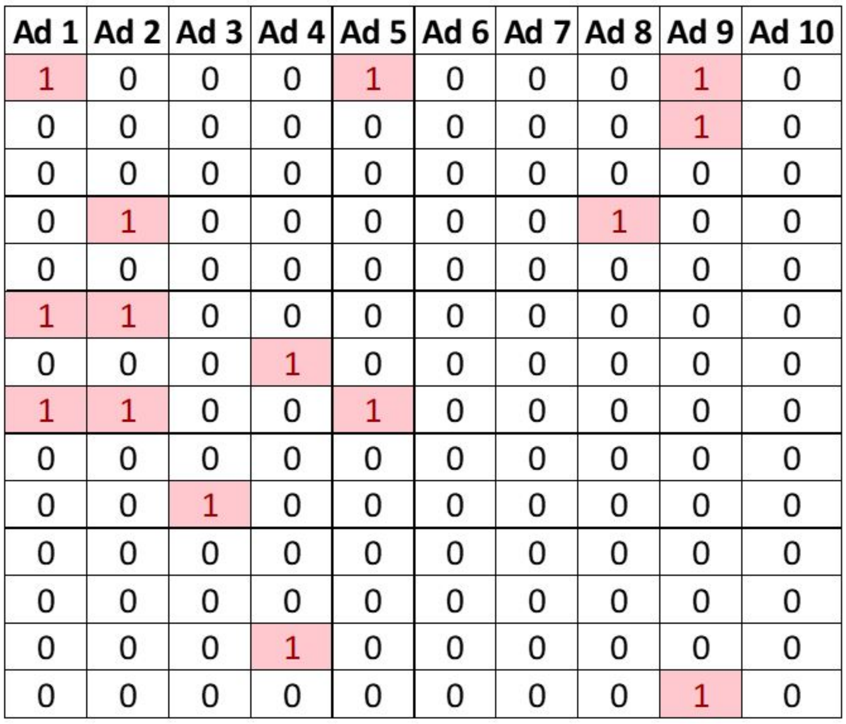
$$Q_t(a) = \ Q_{t-1}(a) + \frac{1}{\alpha}(R_t - Q_{t-1}(a))$$

Note that we have replaced Nt(at) here with a constant alpha, which ensures that the recent samples are given higher weights, and increments are decided more by such recent samples. There are other techniques which provide different solutions to bandits with non-stationary rewards. You can read more about them in this paper (https://arxiv.org/abs/0805.3415).

Python Implementation from scratch for Ad CTR Optimization

As mentioned in the use cases section, MABP has a lot of applications in the online advertising domain.

Suppose an advertising company is running 10 different ads targeted towards a similar set of population on a webpage. We have results for which ads were clicked by a user here (https://drive.google.com/open?id=1whkIlnL4FKeHg2lfdcbT1j18L26fg9aF). Each column index represents a different ad. We have a 1 if the ad was clicked by a user, and 0 if it was not. A sample from the original dataset is shown below:



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

This is a simulated dataset and it has Ad #5 as the one which gives the maximum reward.

First, we will try a random selection technique, where we randomly select any ad and show it to the user. If the user clicks the ad, we get paid and if not, there is no profit.

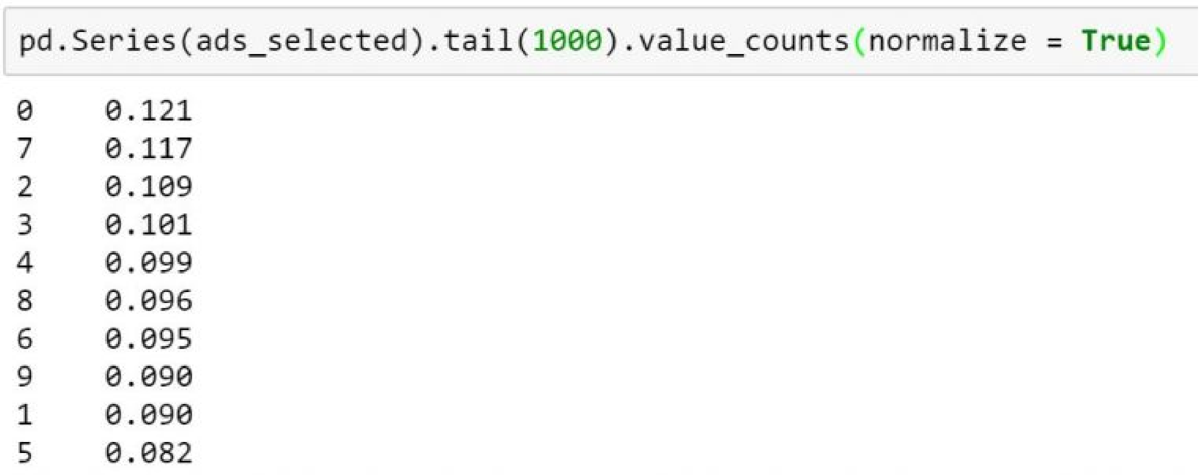
```
# Random Selection


# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# Importing the dataset
dataset = pd.read_csv('Ads_Optimisation.csv')


# Implementing Random Selection
import random
N = 10000
d = 10
ads_selected = []
total_reward = 0
for n in range(0, N):
    ad = random.randrange(d)
    ads_selected.append(ad)
    reward = dataset.values[n, ad]
    total_reward = total_reward + reward
```

Total reward for the random selection algorithm comes out to be 1170. As this algorithm is not learning anything, it will not smartly select any ad which is giving the maximum return. And hence even if we look at the last 1000 trials, it is not able to find the optimal ad.



```
pd.Series(ads_selected).tail(1000).value_counts(normalize = True)
```

```
0    0.121
7    0.117
2    0.109
3    0.101
4    0.099
8    0.096
6    0.095
9    0.090
1    0.090
5    0.082
```

Now, let's try the Upper Confidence Bound algorithm to do the same:

```
# 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
```

The *total_reward* for UCB comes out to be 2125. Clearly, this is much better than random selection and indeed a smart exploration technique that can significantly improve our strategy to solve a MABP.

```
pd.Series(ads_selected).head(1500).value_counts(normalize = True)
```

```
4    0.250000
7    0.170667
0    0.106000
3    0.091333
8    0.075333
1    0.073333
6    0.066000
2    0.066000
9    0.053333
5    0.048000
dtype: float64
```

After just 1500 trials, UCB is already favouring Ad #5 (index 4) which happens to be the optimal ad, and gets the maximum return for the given problem.

End Notes

Being an active area of research MABP will percolate to various other fields in the industry. These algorithms are so simple and powerful that they are being used increasingly by even small tech companies, as the computation resources required for them are often low.

Going forward, there are other techniques based on probabilistic models such as Thompson Sampling explained by Professor Balaraman in this amazing video (https://www.youtube.com/watch?v=H2OWTxdauqA).

You can attend a highly anticipated and extremely useful talk on reinforcement learning from him at DataHack Summit 2018 in Bangalore as well! For more details, please visit https://www.analyticsvidhya.com/datahack-summit-2018/ (https://www.analyticsvidhya.com/datahack-summit-2018/).

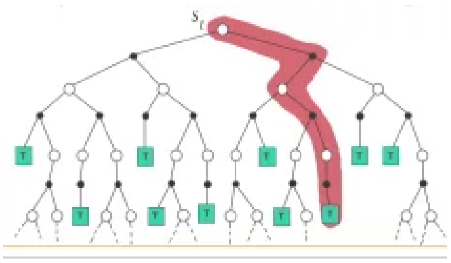
(https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/?share=pocket&nb=1)

(https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/?share=reddit&nb=1)

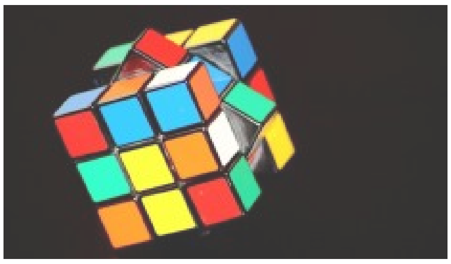
Like this:

Loading...

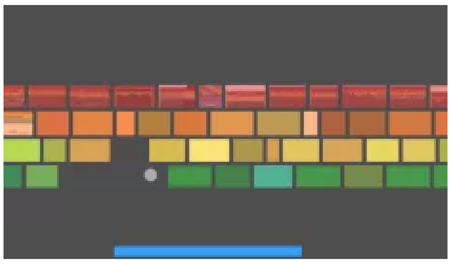
Related Articles



(https://www.analyticsvidhya.com/blog/2018/11/reinforcement-learning-introduction-monte-carlo-learning-openai-gym/)
Reinforcement Learning: Introduction to Monte Carlo Learning using the OpenAI Gym Toolkit (https://www.analyticsvidhya.com/blog/2018/11/reinforcement-learning-introduction-monte-carlo-learning-openai-gym/)
November 19, 2018
In "Python"



(https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/)
Simple Beginner's guide to Reinforcement Learning & its implementation (https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/)
January 19, 2017
In "Machine Learning"



(https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/)
A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python (https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/)
April 18, 2019
In "Python"

TAGS : MULTI-ARMED BANDIT (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/MULTI-ARMED-BANDIT/), PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PYTHON/), REINFORCEMENT LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/REINFORCEMENT-LEARNING/)

NEXT ARTICLE

The Winning Approaches from codeFest 2018 – NLP, Computer Vision and Machine Learning!
(https://www.analyticsvidhya.com/blog/2018/09/the-winning-approaches-from-codefest-2018-nlp-computer-vision-and-machine-learning/)

•••

PREVIOUS ARTICLE

10 Mind-Blowing TED Talks on Artificial Intelligence Every Data Scientist & Business Leader Must Watch
(https://www.analyticsvidhya.com/blog/2018/09/best-ted-talks-artificial-intelligence-must-watch/)

(https://www.analyticsvidhya.com/blog/author/ankit2106/)

Ankit Choudhary (Https://Www.Analyticsvidhya.Com/Blog/Author/Ankit2106/)

IIT Bombay Graduate with a Masters and Bachelors in Electrical Engineering. I have previously worked as a lead decision scientist for Indian National Congress deploying statistical models (Segmentation, K-Nearest Neighbours) to help party leadership/Team make data-driven decisions. My interest lies in putting data in heart of business for data-driven decision making.

in (https://www.linkedin.com/in/ankit-choudhary-b9360826/)

This article is quite old and you might not get a prompt response from the author. We request you to post this comment on Analytics Vidhya's **Discussion portal** (https://discuss.analyticsvidhya.com/) to get your queries resolved

2 COMMENTS

BISWA G SUNH

Reply

October 25, 2018 at 10:58 am (https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/#comment-155446)

Are N rows for different users clicks at different time? So you are predicting action by selecting one ad and calculating reward right? How to explore here if you only show top ad to the users?

Thanks

-

ANKIT CHOUDHARY

Reply

October 25, 2018 at 3:05 pm (https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/#comment-155450)

Hi, Are you talking about the random selection algorithm or UCB1?

Your Ultimate path for Becoming a DATA Scientist!

Download this learning path to start your data science journey.

Name

Email Id

Contact Number

Download Resource

JOIN THE NEXTGEN DATA SCIENCE ECOSYSTEM

Get access to free courses on Analytics Vidhya

Get free downloadable resource from Analytics Vidhya

Save your articles

Participate in hackathons and win prizes

(https://id.analyticsvidhya.com/accounts/login/?next=https://www.analyticsvidhya.com/blog/?utm_source=blog-subscribe&utm_medium=web)

Join Now

(https://software.seek.intel.com/DataCenter_to_Edge_REG?registration_source=AnalyticsVidhya-APJ)

POPULAR POSTS

- 24 Ultimate Data Science Projects To Boost Your Knowledge and Skills (& can be accessed freely) (https://www.analyticsvidhya.com/blog/2018/05/24-ultimate-data-science-projects-to-boost-your-knowledge-and-skills/)
- Essentials of Machine Learning Algorithms (with Python and R Codes) (https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/)
- 7 Types of Regression Techniques you should know! (https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/)
- A Complete Tutorial to Learn Data Science with Python from Scratch (https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/)
- Understanding Support Vector Machine algorithm from examples (along with code) (https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)
- Stock Prices Prediction Using Machine Learning and Deep Learning Techniques (with Python codes) (https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/)
- Introduction to k-Nearest Neighbors: Simplified (with implementation in Python) (https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/)
- A Simple Introduction to ANOVA (with applications in Excel) (https://www.analyticsvidhya.com/blog/2018/01/anova-analysis-of-variance/)

-

Your Ultimate path for Becoming a DATA Scientist!

Download this learning path to start your data science journey.

Name

Email Id

Contact Number

Download Resource

RECENT POSTS

Top 7 Machine Learning Github Repositories for Data Scientists (https://www.analyticsvidhya.com/blog/2019/06/top-7-machine-learning-github-repositories-data-scientists/)

JUNE 6, 2019

The AI Comic: Z.A.I.N – Issue #1: Automating Attendance using Computer Vision (https://www.analyticsvidhya.com/blog/2019/06/ai-comic-zain-issue-1-automating-computer-vision/)

JUNE 3, 2019

DataHack Radio #23: Ines Montani and Matthew Honnibal – The Brains behind spaCy (https://www.analyticsvidhya.com/blog/2019/06/datahack-radio-ines-montani-matthew-honnibal-brains-behind-spacy/)

JUNE 3, 2019

Exclusive Interview with Sonny Laskar – Kaggle Master and Analytics Vidhya Hackathon Expert (https://www.analyticsvidhya.com/blog/2019/05/exclusive-interview-sonny-laskar-kaggle-master-analytics-vidhya-hackathon-expert/)

MAY 30, 2019

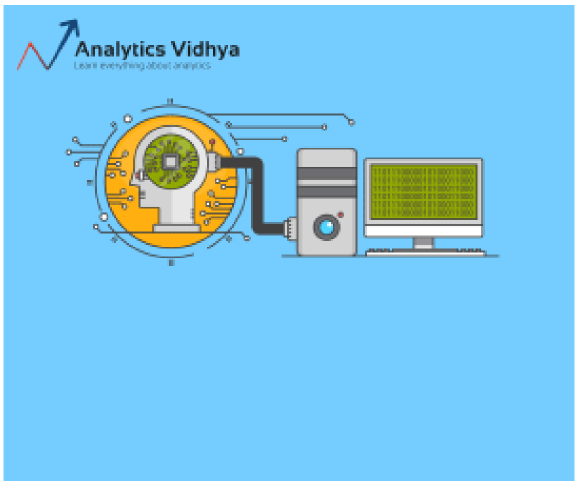
(https://datahack.analyticsvidhya.com/contest/intel-ai-enterprise-meetup-mumbai-invite-only/?utm_source=Sticky_banner1&utm_medium=display&utm_campaign=IntelMum)

Your Ultimate path for Becoming a DATA Scientist!

Download this learning path to start your data science journey.

Name
Email Id
Contact Number

Download Resource



(https://courses.analyticsvidhya.com/courses/applied-machine-learning-beginner-to-professional?utm_source=Sticky_banner2&utm_medium=display&utm_campaign=Applied_ML)