INTRODUCTION

Reinforcement Learning is a branch of Machine Learning, also called Online Learning. It is used to solve interacting problems where the data observed up to time t is considered to decide which action to take at time t + 1. It is also used for Artificial Intelligence when training machines to perform tasks such as walking. Desired outcomes provide the AI with reward, undesired with punishment. Machines learn through trial and error.

Here, we will understand and learn how to implement the following Reinforcement Learning models:

1. Upper Confidence Bound (UCB)
2. Thompson Sampling

Machine learning and data analysis are two areas where open source has become almost the de facto license for innovative new tools. Both the Python and R languages have developed robust ecosystems of open source tools and libraries that help data scientists of any skill level more easily perform analytical work.

The distinction between machine learning and data analysis is a bit fluid, but the main idea is that machine learning prioritizes predictive accuracy over model interpretability, while data analysis emphasizes interpretability and statistical inference. Python, being more concerned with predictive accuracy, has developed a positive reputation in machine learning. R, as a language for statistical inference, has made its name in data analysis.

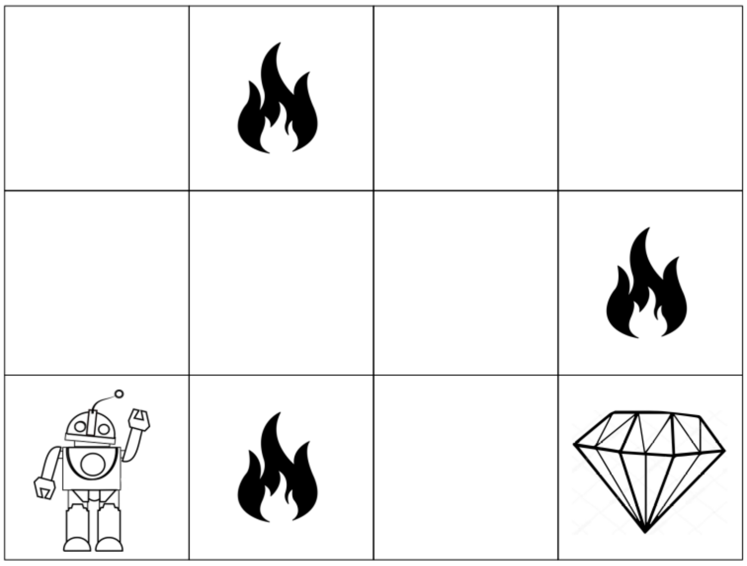
Here we will implement the Reinforcement Learning models using both Python & R.

REINFORCEMENT LEARNING

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.

Example : The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.

The image shows robot, diamond and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that is fire. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract



the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- **Input:** The input should be an initial state from which the model will start
- **Output:** There are many possible output as there are variety of solution to a particular problem
- **Training:** The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Various Practical applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

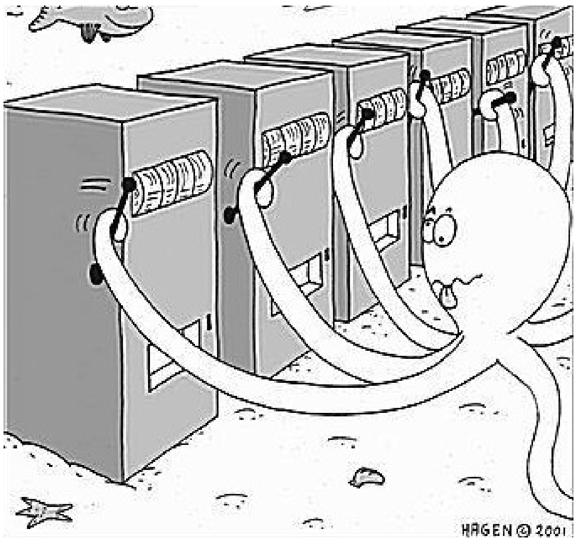
RL can be used in large environments in the following situations:

1. A model of the environment is known, but an analytic solution is not available;
2. Only a simulation model of the environment is given (the subject of simulation-based optimization);
3. The only way to collect information about the environment is to interact with it.

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. Each arm chosen is equivalent to an action, which then leads to an immediate reward.

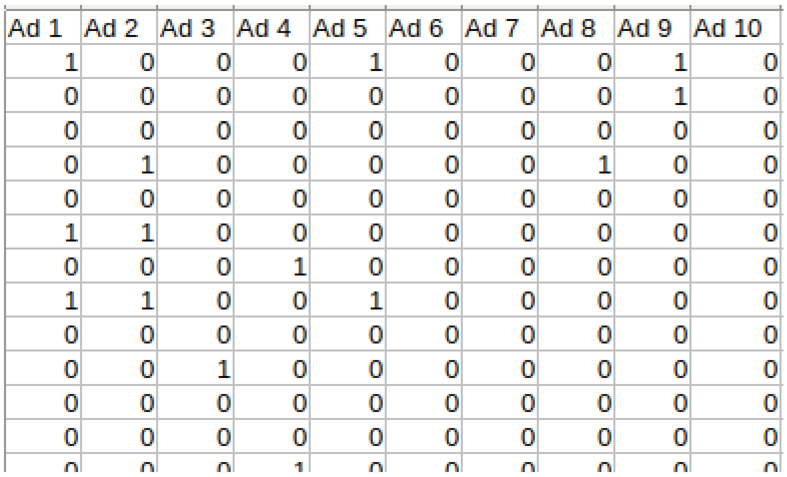
THE PROBLEM & THE DATASET

The most common multi-armed bandit problem is Advertising. Here, we consider 10 Advertisement of a certain product and we have to find out which Advertisement maximises our returns.

The dataset contains the click through rates of 10 advertisements. The value 1 indicates the user likes the advertisement and has clicked on it. The value 0 indicates the user didn’t liked the advertisement and has not clicked on it.

Each row shows the data for 1 user who has been shown 10 versions of same advertisement and based on whether he clicked it or not, we get 10 values, each either 1(clicked) or 0(not clicked). The dataset contains data for 10,000 such users.

A part of the input dataset looks like :

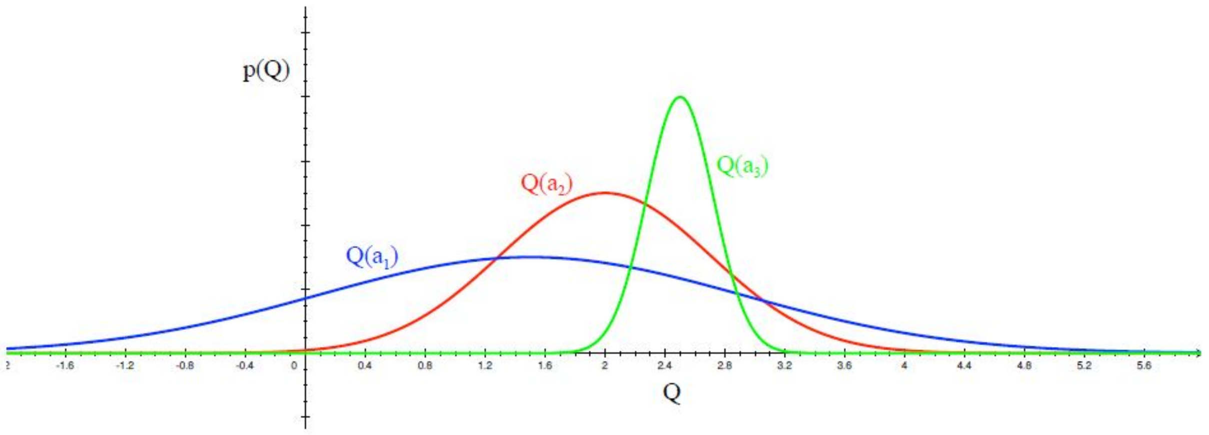


Ad 1	Ad 2	Ad 3	Ad 4	Ad 5	Ad 6	Ad 7	Ad 8	Ad 9	Ad 10
1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0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

<u>Upper Confidence Bound (UCB)</u>

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.

Algorithm

Step 1 : At each round n, we consider two numbers for each ad i:

- $N_i(n)$: the number of times the ad i was selected up to round n,
- $R_i(n)$: the sum of rewards of the ad i up to round n

Step 2 : From these two numbers we compute:

- The average reward of ad i up to round n $\bar{r}_i(n)=\frac{R_i(n)}{N_i(n)}$
- Confidence interval $[\bar{r}_i(n)-\Delta_i(n),\bar{r}_i(n)+\Delta_i(n)]$ at round n with $\Delta_i(n)=\sqrt{\frac{3\log(n)}{2N_i(n)}}$

Step 3 : We select the ad i that has the maximum UCB $\bar{r}_i(n)+\Delta_i(n)$.

Python code and explanation

- **Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
```

- **Importing the dataset**

```
dataset = pd.read_csv('Ads_CTR_Optimisation.csv')
```

- **Implementing UCB**

```
N,d = dataset.shape
numbers_of_selections = [0] * d
sums_of_rewards = [0] * d
upper_bound = [0] * d
ads_selected = []

for n in range(N):
    for i in range(d):
        if numbers_of_selections[i]>0:
            average_reward = sums_of_rewards[i]/numbers_of_selections[i]
            delta_i = math.sqrt((3*math.log(n+1))/(2* numbers_of_selections[i]))
            upper_bound[i] = average_reward + delta_i
        else:
            upper_bound[i] = 1e400
    ad = max(range(d), key = upper_bound.__getitem__)
    ads_selected.append(ad)
    numbers_of_selections[ad] += 1
    sums_of_rewards[ad] += dataset.values[n,ad]
  total_reward = sum(sums_of_rewards)
```

- **Plotting Histogram of Ads selected**

```
plt.hist(ads_selected)
plt.title('Histogram of Ads selected')
plt.xlabel('Ads')
plt.ylabel('Number of times ad was selected')
plt.show()
```

R code and explanation

- **Importing the dataset**

```
dataset <- read.csv('Ads_CTR_Optimisation.csv')
```

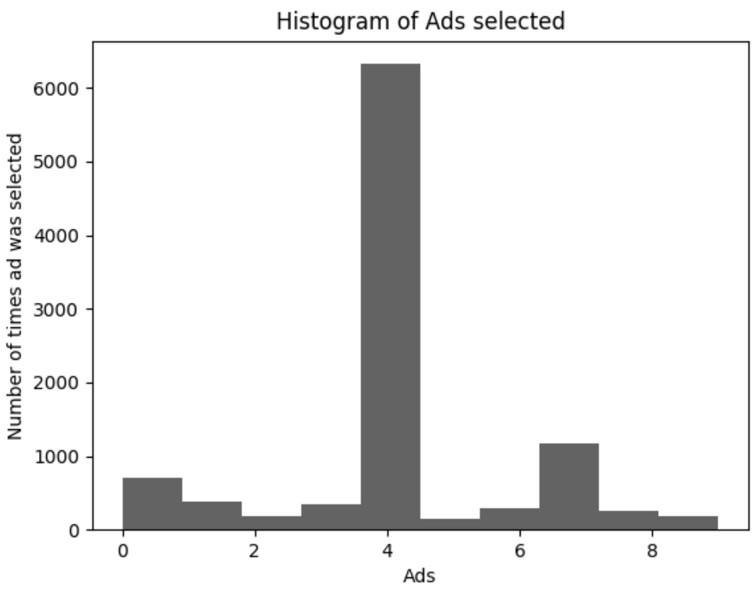
- **Implement UCB**

```
N <- nrow(dataset)
d <- ncol(dataset)
numbers_of_selections <- integer(d)
sums_of_rewards <- integer(d)
upper_bound <- integer(d)
ads_selected <- integer()
for (n in 1:N)
{
    for (i in 1:d)
    {
        if (numbers_of_selections[i]>0)
        {
            average_reward <- sums_of_rewards[i]/numbers_of_selections[i]
            delta_i <- sqrt( ( 3 * log(n) )/(2 * numbers_of_selections[i]) )
            upper_bound[i] <- average_reward + delta_i
        }
        else
        {
            upper_bound[i] = 1e400
        }
    }
    ad = which.max(upper_bound)
    ads_selected <- append(ads_selected, ad)
    numbers_of_selections[ad] <- numbers_of_selections[ad] + 1
    sums_of_rewards[ad] <- sums_of_rewards[ad] + dataset[n,ad]
}
total_reward = sum(sums_of_rewards)
```

- **Visualizing results**

```
hist(ads_selected,
     col = 'blue',
     main = 'Histogram of Ads selected',
     xlab = 'Ads',
     ylab = 'Number of times ad was clicked'
     )
```

<u>Output</u>



This output clearly indicates that Advertisement number 5 (index 4 as starting from 0) is the advertisement that was most selected by the users and has the highest conversion rate and therefore that is the advertisement we need to show to the users.

THOMPSON SAMPLING

- Ad i gets rewards **y** from Bernoulli distribution $p(\mathbf{y}|\theta_i) \sim B(\theta_i)$
- θ_i is unknown but we set its uncertainty by assuming it has a uniform distribution $p(\theta_i) \sim U([0,1])$, which is the prior distribution.
- Bayes Rule: we approach by the posterior distribution

$$\underbrace{p(\theta_i|\mathbf{y})}_{\text{posterior distribution}} = \frac{p(\mathbf{y}|\theta_i)p(\theta_i)}{\int p(\mathbf{y}|\theta_i)p(\theta_i)d\theta_i} \propto \underbrace{p(\mathbf{y}|\theta_i)}_{\text{likelihood function}} \times \underbrace{p(\theta_i)}_{\text{prior distribution}}$$

- We get $p(\theta_i|\mathbf{y}) \sim \beta$(number of successes +1, number of failures +1)
- At each round n we take a random draw $\theta_i(n)$ from this posterior distribution $p(\theta_i|y)$, for each ad i
- At each round n we select the ad i that has the highest $\theta_i(n)$.

Algorithm:

Step 1: At each round n, we consider two numbers for each ad i :

$N_i^1(n)$ - the number of times the ad i got reward 1 up to round n,

$N_i^0(n)$ - the number of times the ad i got reward 0 up to round n.

Step 2: For each ad i, we take a random draw from the distribution below:

$$\theta_i(n) = \beta(N_i^1(n) + 1, N_i^0(n) + 1)$$

Step 3: We select the ad that has the highest $\theta_i(n)$.

Python code and explanation

- **Importing the libraries**

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import random
```

- **Importing the dataset**

```
dataset = pd.read_csv('Ads_CTR_Optimisation.csv')
```

- **Implementing Thompson Sampling**

```
N,d = dataset.shape
numbers_of_rewards_1 = [0] * d
numbers_of_rewards_0 = [0] * d
random_beta = [0] * d
ads_selected = []
total_reward = 0

for n in range(N):
    for i in range(d):
        random_beta[i] = random.betavariate(numbers_of_rewards_1[i] + 1,
numbers_of_rewards_0[i] + 1)
    ad = max(range(d), key = random_beta.__getitem__)
    ads_selected.append(ad)
    reward = dataset.values[n,ad]
    if reward:
        numbers_of_rewards_1[ad] +=1
    else:
        numbers_of_rewards_0[ad] +=1
    total_reward += reward
```

- **Plotting Histogram of Ads selected**

```
plt.hist(ads_selected)
plt.title('Histogram of Ads selected')
plt.xlabel('Ads')
plt.ylabel('Number of times ad was selected')
plt.show()
```

R code and explanation

- **Importing the dataset**

```
dataset <- read.csv('Ads_CTR_Optimisation.csv')
```

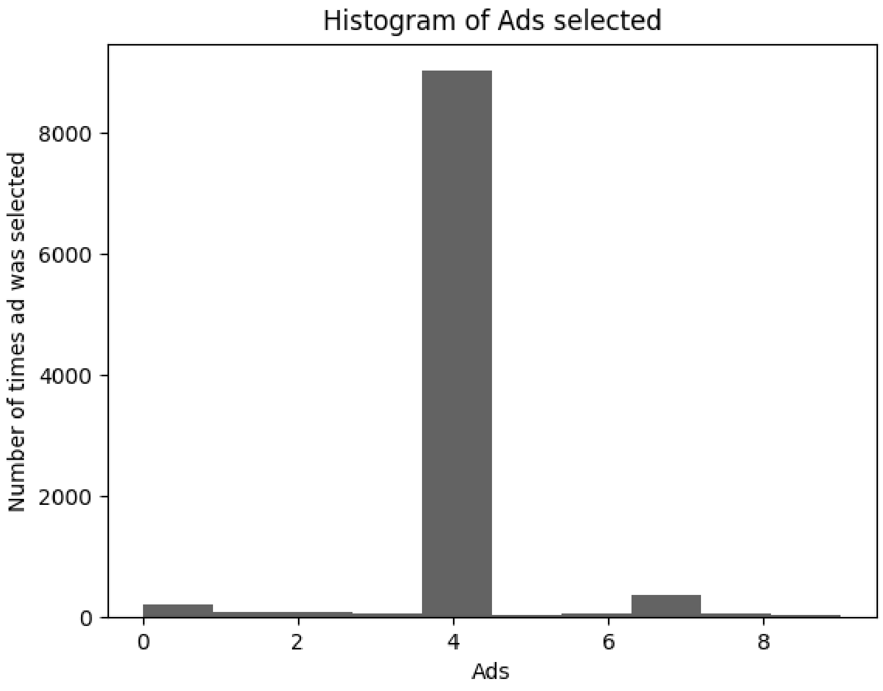
- **Implement Thompson Sampling**

```
N <- nrow(dataset)
d <- ncol(dataset)
numbers_of_rewards_1 <- integer(d)
numbers_of_rewards_0 <- integer(d)
random_beta <- integer(d)
ads_selected <- integer()
total_reward <- 0
for (n in 1:N)
{
    for (i in 1:d)
    {
        random_beta[i] = rbeta(n = 1,
                               shape1 = numbers_of_rewards_1[i] + 1,
                               shape2 = numbers_of_rewards_0[i] + 1)
    }
    ad = which.max(random_beta)
    ads_selected <- append(ads_selected, ad)
    reward <- dataset[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
}
```

- **Visualizing results**

```
hist(ads_selected,
     col = 'blue',
     main = 'Histogram of Ads selected',
     xlab = 'Ads',
     ylab = 'Number of times ad was clicked'
)
```

Output



This output clearly indicates that Advertisement number 5 (index 4 as starting from 0) is the advertisement that was most selected by the users and has the highest conversion rate and therefore that is the advertisement we need to show to the users.

CONCLUSION

Thus we have explained both the Upper Confidence Bound (UCB) & Thompson Sampling models for Reinforcement Learning and implemented it in both Python and R.

We see that **UCB** is a **deterministic** algorithm and it **requires update at every round.**

Whereas **Thompson Sampling** is a **probabilistic** algorithm and **can accomodate delayed feedback** and **has better empherical evidence.**