Multi-Armed Bandits

Group Project 3: Determining the Best Combination
DBA5101 Analytics in Managerial Economics
National University of Singapore
16th Nov 2022

Team 10
Kenny Lim (A0262753B)
Manan Lohia (A0262838U)
Peiwen Jiang (A0262785R)
Jung Hyun Park (A0262662H)
Janita Bose (A0262823E)

BACKGROUND

The Multi-Armed Bandit (MAB) technique, which originated from a simple model for clinical trials in the 1930s, is a reinforcement machine learning algorithm for website optimisation in modern times. This report discusses key concepts in multi-armed bandits and explores different MAB algorithms to find the best website design combination.

AIM

In this study, we aim to determine the best web design that attracts the maximum number of visitors while minimising the number of customers lost to less optimal variations. There are twenty-four different 'arms' or options for our website design. We will consider different algorithms, such as Epsilon-Greedy and the Upper Confidence Bound (UCB-1), and choose the optimal algorithm to achieve the highest reward with the fewest number of pulls.

EXPLORATION - EXPLOITATION DILEMMA

Today, A/B testing is the most common technique in web optimization that compares between two or more designs to determine which performs better in certain metrics such as Click-Through Rate or Conversion Rate. A/B testing works by randomly assigning a proportion of the population to be exposed to a specific design with equal probability for a given period of time. The results are later compared to determine the superior design.

In reality, this is not ideal as losses, often in the form of revenue, are incurred during the testing (or exploration) phase in which data is collected. As such, to minimise losses, we are naturally inclined towards reducing the exploration phase while maximising our revenue by exposing the best design to the target population (as known as exploitation). This is where MAB comes into play.

THE ALGORITHMS

MAB algorithms differ in terms of the aggressiveness in the exploitation of the winning arm. In our scenario of determining the best web design, this means asking the following questions:

1. How quickly does the algorithm assign traffic to a winning design?
2. How quickly does it drop non-performing variations?
3. Does it still explore other arms after identifying the best arm?

We will have a look at two algorithms in this paper namely, Epsilon-Greedy and UCB-1, and pick an algorithm that maximises conversion rate.

EPSILON-GREEDY

The Epsilon-Greedy algorithm attempts to find a balance between exploration and exploitation. In the beginning, ε - greedy starts by choosing an arm uniformly at random with probability ε. Once it gets an understanding of the distribution of rewards of each arm, the algorithm picks the action that yields the highest average return so far, with probability of $1 - \varepsilon$.

The best action is defined as the one with the highest experimental mean, where the experimental mean is calculated as the sum of the rewards from that arm divided by the number of times that arm has been pulled. The experimental mean of arm 'a' after T turns is:

$$\frac{\sum_{t=1}^{T} r_{a,t}}{p_{i,T}}$$

$r_{a,t}$: The reward given by arm 'a' at timestep 't'

$p_{i,T}$: The number of times that arm 'a' has been pulled across T total turns

UCB-1

The Upper Confidence Bound (UCB) algorithms are a class of bandit algorithms that explore efficiently by restricting sampling to the action that yields the highest reward. In particular, we have chosen UCB-1 which chooses to explore/exploit based on the following index:

$$A_t = \widehat{\mu}_a(t) + \sqrt{\frac{2ln(t)}{n_a(t)}}$$

A_t = Action 'a' taken at time period 't'

$\widehat{\mu}_a(t)$ = Estimated mean of arm 'a' at round 't'

$n_a(t)$ = The Number of times action 'a' has been selected before time period 't'

The formula is split into two parts, (1) Exploitation and (2) Exploration. In (1), Exploitation is represented by the first half of the equation ($\widehat{\mu}_a(t)$), signalling the algorithm to pick the arm with the highest expected reward at a given time period 't'. Exploration (2) is represented by the uncertainty term in the second half of the equation ($\sqrt{\frac{2ln(t)}{n_a(t)}}$). The numerator $ln(t)$ captures the number of time periods/pulls (of any arm). The denominator $n_a(t)$ records the number of pulls of a specific arm.

This means that the less a particular arm is pulled, the larger the likelihood of the arm being pulled in a coming time period (measured by the uncertainty term). This is despite another arm having a higher expected reward. Yet, the log term in the numerator ensures that the uncertainty term grows slowly quickly leading to unnecessary exploration. Similarly, the linear term $n_a(t)$ reduces the uncertainty term significantly once an arm has been pulled.

In the long run, when $\widehat{\mu}_a(t)$ gets large, it dominates the index and the algorithm focuses on exploiting the best web design, maximising conversion.

WHY WE CHOSE UCB-1

We chose the UCB-1 because it is (1) consistent and tolerant to noise, (2) does not require hyperparameter tuning and (3) time period-to-arms ratio is high.

For (1), since we have no information on the actual distribution of rewards for each arm, it is conservative to deploy an algorithm that is consistent. UCB-1 is found to be consistent in traffic allocation and outperforms A/B testing while quickly achieving statistical significance.

For (2), algorithms such as Epsilon-Greedy require a hyperparameter to be set in order to achieve peak performance e.g. a high epsilon value would cause the algorithm to explore too much while a low value could result in a sub-optimal result. This would require some trial and error to determine the ideal ε resulting in some losses in a real world scenario.

For (3),it is a known weakness of UCB-1 where it may fail to identify the optimal arm in the given time period in situations where the number of pulls is in close range to the number of arms. This is a result of the exploration phase in which each arm is pulled once. Suppose a situation in which we have 5000 pulls to determine the optimal arm out of 5000 arms. Since the maximum number of pulls (1,000,000) far outnumbers the number of arms available (24), we are certain that it will be able to identify the optimal arm within the given number of pulls.

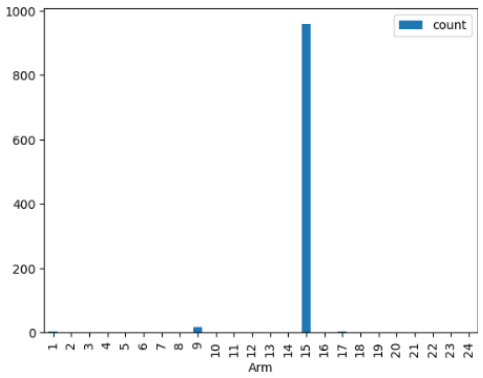
RESULTS

TESTING PHASE

We had to trial pulls at random to test that our algorithm executes as expected and remove any bugs in our code. The results during the testing phase were discarded as they do not reflect the correct execution of the multi-arm bandit. At the end of the testing phase, we trialled 589 pulls, with an accumulated reward of 9894.

OPTIMAL ARM: Arm 15

After the testing phase, we used 1000 pulls to identify the optimal arm. We chose 1000 as we believe it offers the algorithm sufficient pulls to converge on the optimal arm out of the 24 arms and maximise the reward per pull.



As can be seen in the results, the UCB1 algorithm converges on Arm 15 as the best choice and exploits it, with 959 out of a 1000 pulls being performed on Arm 15. However, it still explores a few other arms, such as Arm 9 and Arm 17, but continues to exploit Arm 15 as the one with the highest reward.

REGRET

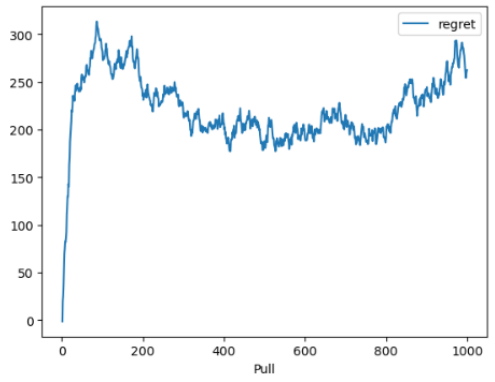
As a term, regret implies that you may have no way to know in advance that you're making a non-optimal choice, only in hindsight do you realise your mistake. Although it may not be possible to tell in advance if you're choosing the best action, you can calculate how your selection policy performs when compared against the optimal policy, in which the best action is selected at every time step. The difference in the total reward obtained by these two policies represents regret.

$$\text{Regret} = T\mu^* - \sum_{t=1}^{T} R_t$$

Here, μ^* gives the the mean of the best hand over time period T, and the Reward function at time t is given as R_t

The regret function is non-negative, since it compares the actual return to the optimal return in hindsight. Within regret theory it is impossible to have negative regret (the optimum is zero regret). This also means that the expected regret is non-negative.

The Epsilon-Greedy algorithm's total regret is theoretically constrained by *O(T)*, while the regret of the UCB-1 is *O(log(T))*, the proof of which is left to the reader as an exercise.



We can see in the chart above that the regret of our algorithm rises sharply in the initial exploration phase but tapers off after reaching a maximum. The deviation from a constantly increasing function can be explained by the fact that rewards are stochastic and may not always be the best for an individual pull, but is optimal when averaged over several pulls.

The shape of the *Regret vs Pull* plot suggests that our algorithm has indeed converged on the optimal arm, allowing us to confidently make the choice of selecting option 15 for the design of the website.

CONCLUSION

In this paper, we have demonstrated that the MAB offers a more rigorous and efficient approach relative to A/B testing as it takes into account the concepts of exploration, exploitation, and regret. The end result is that the system is able to optimise and self correct in real-time without any need for intervention to analyse the results.

Furthermore, the UCB1 algorithm has been discussed more extensively than the Epsilon-Greedy approach for its consistency against noise and lack of need for hyperparameter tuning when the number of arms is small in comparison to the number of ‘pulls’ or trials. In our case, the optimal arm selected by the UCB1 was arm 15 suggesting that the Obama campaign can benefit most by choosing the design corresponding to arm 15. Although only two algorithms are considered in this study, there are numerous MAB algorithms available, with performance and complexity varying on a case-to-case basis. Depending on the use cases that one faces, s/he may explore further which MAB algorithm to adopt, as it offers the optimal option for website design optimization.

In [1]:
```
import os
import pandas as pd
import numpy as np
import scipy
import math
import random
import requests
import matplotlib.pyplot as plt
```

In [2]:
```
class ADozenPairs():

  def __init__(self, credentials):
      self.user_group = credentials['user_group']
      self.secret_key = credentials['secret_key']
      self.arms = list(range(1,25))
      self.pull_no = 0
      self.reward_offset = 0
      self.pull_offset = 0
      self.rewards = pd.DataFrame(columns = ['Arm', 'NetReward', 'Pull', 'Reward', 'Phase'])


      self.mock_total_reward = 0
      self.mock_pull_no = 0


#   def pull(self, arm):

#       reward = random.random()
#       self.mock_total_reward += reward
#       self.mock_pull_no += 1

#       output = {'Arm': str(arm), 'NetReward': self.mock_total_reward, 'Pull': self.mock_pull_no, 'Reward': reward}

#       return output

  def pull(self, arm):
      url = ('https://appiora.nus.edu.sg/pull_arm/%s/%s/%s' % (self.user_group,self.secret_key,str(arm)))

      while True:
          r = requests.get(url)
          if r.ok:
              output = r.json()['result']
              return(output)

      'Succesfully Executed Pull No: {}'.format(self.pull_no)
      #Sample output = {'Arm': '21', 'NetReward': 116, 'Pull': 4, 'Reward': 28}

  def calc_regret(self):

      max_mean_reward = max(self.rewards.groupby('Arm').agg(mean = pd.NamedAgg('Reward', 'mean'))['mean'])

      self.rewards = self.rewards.assign(
          regret = max_mean_reward * self.rewards['Pull'] - self.rewards['NetReward'])
      print(max_mean_reward)
      return True

  def EpsilonGreedyOnce(self, epsilon):

      #Decide to explore or exploit
      explore_flag = np.random.binomial(1, epsilon)


      if (explore_flag) or (self.pull_no == 0):
          #Choose arm to explore
          phase = 'Explore'
          arm = np.random.randint(1, 25)
      else:
          #Choose arm to exploit
          phase = 'Exploit'
          arm = int(self.rewards.groupby('Arm') \
                    .agg(mean_reward = pd.NamedAgg('Reward', 'mean')) \
                    .sort_values('mean_reward', ascending = False) \
                    .reset_index().\
                    iloc[0,0])

      #Pull the arm and get the reward
      reward = self.pull(arm)

      #Modify the NetReward to account for previous pulls not part of this instance/run
      if self.pull_no == 0:
          self.reward_offset = reward['NetReward'] - reward['Reward']
          self.pull_offset = reward['Pull']
          print(reward['Pull'], reward['NetReward'])
      reward['NetReward'] = reward['NetReward'] - self.reward_offset
      reward['Pull'] = reward['Pull'] - self.pull_offset
      reward['Phase'] = phase

      #Append reward to our dataframe
      self.rewards = pd.concat([self.rewards,pd.Series(reward).to_frame().T], ignore_index=True)

      return reward

  def EpsilonGreedy(self, pulls, epsilon):

      if epsilon < 0 or epsilon > 1:
          print('Instructions unclear, arm got stuck in washing machine')
          return True

      for i in range(pulls):
        self.EpsilonGreedyOnce(epsilon)

      self.calc_regret()

      return True

  def UCB1Explore(self):
    for i in range(24):
          print('Executing Pull No: {}'.format(self.pull_no))
          reward = self.pull(i)

          if self.pull_no == 0:
              self.reward_offset = reward['NetReward'] - reward['Reward']
              self.pull_offset = reward['Pull']
              print(reward['Pull'], reward['NetReward'])

          reward['NetReward'] = reward['NetReward'] - self.reward_offset
          reward['Pull'] = reward['Pull'] - self.pull_offset + 1

          self.pull_no += 1

          reward['Phase'] = 'Exploration'
          self.rewards = pd.concat([self.rewards,pd.Series(reward).to_frame().T], ignore_index=True)

    return True

  def UCB1Calculate(self, pull):
      current_pull = self.rewards.groupby('Arm').agg(mean = pd.NamedAgg('Reward', 'mean'), count = pd.NamedAgg('Reward', 'count'))
      current_pull['UCBIndex'] = current_pull['mean'] + np.sqrt(np.log(self.pull_no + 1)/current_pull['count'])
      optimal_arm = int(current_pull.sort_values('UCBIndex', ascending = False).reset_index().iloc[0,0])
      current_pull.to_csv('./UCBindexUCB1ndex/{}.csv'.format(pull))

      return optimal_arm

  def UCB1Exploit(self, pull):
      arm = self.UCB1Calculate(pull)                                          #Get optimal arm based on results so far
      print(arm)
      reward = self.pull(arm)                                                 #Get reward for new pull

      reward['NetReward'] = reward['NetReward'] - self.reward_offset          #Modify the NetReward to account for previous pulls not part of this instance/run
      reward['Pull'] = reward['Pull'] - self.pull_offset + 1
      reward['Phase'] = 'Exploitation'
      self.rewards = pd.concat([self.rewards,pd.Series(reward).to_frame().T], ignore_index=True)
      self.pull_no += 1
      return True

  def UCB1(self, pulls):
      if pulls < 24:
          print('Instructions unclear, arm got stuck in washing machine')
          return True

      self.UCB1Explore()                                                      #Initial Exploration Phase:
      pulls = pulls - 24
      print('Finished Exploration...\nBeginning Exploitation')
      for i in range(pulls):
        print('Executing Pull No: {}'.format(self.pull_no))                   #Exploitation Phase
        self.UCB1Exploit(i + 25)

      self.calc_regret()
      return True
```

In [3]:
```
#Define necessary variables
num_pulls = 1000
epsilon = 0.2

train_credentials = {
    'user_group': 'test_user',
    'secret_key': 'aaaaaaaa'
}

test_credentials = {'user_group':'user15',
                    'secret_key':'n7OPy3lt'}

#Initialize Class
MAB = ADozenPairs(test_credentials)

# Run UCB1
MAB.UCB1(num_pulls)
MAB.rewards.to_csv('Actual_Pull.csv')
# Run epsilon greedy
# MAB.EpsilonGreedy(num_pulls, epsilon)
```

In [4]:
```
MAB.rewards
```

Out[4]:

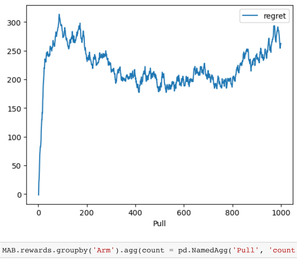
	Arm	NetReward	Pull	Reward	Phase	regret
0	0	26	1	26	Exploration	-1.576642
1	1	29	2	3	Exploration	19.846715
2	2	48	3	19	Exploration	25.270073
3	3	61	4	13	Exploration	36.693431
4	4	68	5	7	Exploration	54.116788
...	...	...	...	...	...	...
995	14	24067	996	31	Exploitation	258.664234
996	14	24096	997	29	Exploitation	254.087591
997	14	24117	998	21	Exploitation	257.510949
998	14	24138	999	21	Exploitation	260.934307
999	14	24161	1000	23	Exploitation	262.357664

1000 rows × 6 columns

In [5]:
```
MAB.rewards.plot.line(x = 'Pull', y = 'regret')
```

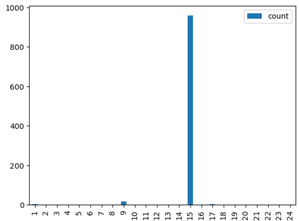
```
/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.py:6999: FutureWarning: In a future version, the Index constructor will not infer numeric dtypes when passed object-dtype sequences (matching Series behavior)
  return Index(sequences[0], name=names)
```

Out[5]:
```
<AxesSubplot:xlabel='Pull'>
```



In [10]:
```
MAB.rewards.groupby('Arm').agg(count = pd.NamedAgg('Pull', 'count')).plot.bar()
```

Out[10]:
```
<AxesSubplot:xlabel='Arm'>
```



Link to Results of 1000 pulls: https://docs.google.com/spreadsheets/d/1Y17uzRs5zKoGR43l2lxxOFdzLySuLeQ56K9-k3rqwcY/edit?usp=share_link