MATH50002 - GROUP RESEARCH PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Multi-armed Bandits

Authors:
Shivam Patel, Adhavan Sashikumar, Diandian Chen, Xinqi Yao, Rui Tang

Contents

1 Introduction

1.1 The Problem

The multi-armed bandit problem is best expressed as a gambler with a row of slot machines, all with unknown expected returns. The problem is to determine the optimal method to play each machine to maximise returns (Vermorel and Mohri, 2005). The difficulty of the problem lies in the fact that we only get to know more information about each machine by playing that machine. This will be highlighted when we come onto the concept of regret later on.

This problem could be simplified down to 2 coins of unknown bias. Suppose we are allowed to make n tosses, with the promise of getting $1 for each head but nothing for tails (Robbins, 1952). What is the best way to choose which coin to toss to maximise earnings? The multi-armed bandit problem has many variations, with known or unknown distributions and/or means.

In our report we will use arms to describe the slot machines and pulling an arm is equivalent to playing a slot machine. We will call each pull a round such that in the coin example above, we can say there are n rounds.

1.2 Applications

The applications of the multi armed bandit problem are extensive. In healthcare, statisticians collect data for assessing treatment effectiveness (Bouneffouf and Rish, 2019). Even in the current COVID-19 situation one could call the vaccine rollout a multi-armed bandit problem. Determining which vaccine will give the best protection by trialing a proportion of population on each vaccine, to then eventually figure out which is the best to then administer to more people. The world of financial services uses the solutions to this problem for portfolio selection to maximise returns on investments. For example if one would like to invest in stocks, one could see each stock as an arm, and pulling an arm is like buying such a stock.

A more modern application is in social media (Chen et al., 2013). Influence maximisation is a huge problem in today's world. This particular problem exposes seeds of users to some information and then tracks the spread of said information, with the aim being to find which seed of users will result in the greatest spread of information. For example, on one particular website they may show different advertisements but for the same product to determine which advertisement results in the most people clicking on such advertisement.

1.3 Forms of the Problem

There are many forms of the problem. The two we will focus on this problem are the stationary and non-stationary problem. The stationary problem is where the actual expectation of each arm does not change and in the non-stationary problem, the actual expectation of each arm can change. This could be periodically or continuous. Another form of the problem is the contextual bandit problem, where one has information on each arm before playing. In the stocks example this could be the financial situation of the company. Or in the social media example, it is information on the user.

1.4 Regret

Before analysing strategies, there is one very important measure that we must define. The **regret** (ρ) after T rounds of the multi-armed bandit is defined as

$$\rho = T\mu^* - \sum_{t=1}^{T} \hat{r}_t$$

where μ^* is the **maximal reward mean** and $\hat{r}_t$ is the reward at time t (Vermorel and Mohri, 2005). The maximal reward mean is the mean of all the maximums across arms for each round. We can rewrite this as

$$\rho = \sum_{t=1}^{T} \hat{r}_t$$

The reason this is of utmost importance is because minimizing it will give us our most optimal strategies. The 'holy grail' we are searching for is known as a **zero regret strategy**, which is a strategy whose average regret per round tends to zero with probability 1 as the number of rounds T tends to infinity (Vermorel and Mohri, 2005).

2 Stationary Strategies

Here, we define **stationary** distributions for arms as those where the underlying distribution of each arm stays the same regardless of the round we are in - some strategies for this type of multi-armed bandit problem are described below.

2.1 ϵ-Greedy

The ϵ-Greedy strategy is where arms are 'explored randomly ϵ percentage of rounds and exploited $1 - \epsilon$ percentage of the time' (Collier and Llorens, 2018). It follows that the ϵ we choose must be in the interval [0,1], and we would like to choose ϵ so that we get the perfect balance between exploration (gathering data from all arms) and exploitation (playing the arm with the highest realised expectation). This theme is key to all solution strategies of the multi-armed bandit, but later strategies, rather than having such a clear divide between exploration and exploitation, have more of a dynamic approach.

2.1.1 Random strategy

The first (and most basic) of the ϵ-Greedy strategies is the random strategy, which will provide us a benchmark against which to compare all other strategies. As the name implies, for each round we choose an arm at random. This means $\epsilon = 1$ and there will be pure exploration and no exploitation.

2.1.2 Epsilon-first Strategy

The ϵ-first strategy consists of doing the exploration all at once at the beginning (Vermorel and Mohri, 2005). Assume we arbitrarily take k rounds in total, where $k \in \mathbb{N}$, then ϵk rounds of them will be selected in the exploration phase, where ϵ is in the interval (0,1), during which an arm is randomly selected. This is followed by (1-ϵ)k rounds in the exploitation phase, where the arm with the highest realised expectation is always selected.

Pseudo Code for Epsilon-first Strategy

```
Data: number of rounds, number of arms, epsilon
Result: regret after using epsilon strategy;
Initialise environment with number of arms;
for number of rounds*epsilon do
    Pick random arm;
end
Calculate realised expectation of each arm;
for number of rounds*(1-epsilon) do
    Pick arm with highest realised expectation;
end
Calculate and return regret
```

Algorithm 1: Epsilon first strategy

2.2 Upper Confidence Bound

In the Upper Confidence Bound (UCB) strategy, we use the realised values to construct upper confidence bounds on the expectation of each arm at some fixed confidence level. We then choose the arm which has the highest upper confidence bound to pull next (Bubeck and Cesa-Bianchi, 2012). As an arm is pulled more its confidence region gets smaller and our estimate is assumed to be more accurate.

The algorithm measures potential by an upper confidence bound of the reward value, $\widehat{U_t(a)}$. Then the true value becomes $Q(a) \leq \widehat{Q_t(a)} + \widehat{U_t(a)}$, where $\widehat{Q_t(a)}$ is the sample mean and $Q(a)$ is the true mean.

2.2.1 Hoeffding's Inequality

Hoeffding's inequality is applicable in this strategy. Let $X_1, \ldots, X_n$ be independent and identically distributed random variables, then $P(E[X] > \overline{X}_t + u) \leq e^{-2tu^2}$ with u being the upper confidence bound and $\overline{X}_t$ the sample mean respectively (Hoeffding, 1994).

Applied in our case,

$$P[Q(a) > \widehat{Q_t(a)} + \widehat{U_t(a)}] \leq e^{-2tU_t(a)^2},$$

where a represents the arm index and t is the time. We take $e^{-2tU_t(a)^2}$ to be the threshold here and represented by p. So we can get $U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$, where $N_t(a)$ is the number of times arm a has been played at time t.

The threshold is used as the upper bound for probability. We can use Hoeffding's inequality to get the confidence bounds of reward variance.

2.2.2 UCB1

UCB1 uses Hoeffding's inequality as the upper limit of the average reward distribution of the arm, where the true average is likely to be lower than the UCB allocated by the algorithm. UCB1, achieving logarithmic regret uniformly over n and without any preliminary knowledge about the reward distributions. The index of this policy is the sum of two terms. The strategy starts by playing each arm once, we do the loop that play arm t which maximises $\overline{X}_t + \sqrt{\frac{2logt}{N_t(a)}}$ (Auer et al., 2002).
Under UCB1, logarithmic regret $O(\log N)$ is achieved when the reward distributions are supported on [0,1] (Chan, 2020). We can conclude that regret of UCB1 grows at a rate of $\log N$.

2.2.3 KL-UCB

In KL-UCB strategy, we first pull each arm once and choose the next pull according to regret bounds rely on deviations results of independent interest (Garivier and Cappé, 2011).
The regret of KL-UCB algorithm is

$$\lim_{n\to\infty} \frac{E[R_n]}{\log n} \leq \sum_{a:u_a<u_{a*}} \frac{u_{a*} - u_a}{d(u_a, u_{a*})}$$

where $d(p,q) = p\log(p/q) + (1-p) * \log((1-p)/(1-q))$ denotes the Kullback-Leibler divergence between Bernoulli distributions of parameters p and q, respectively. Here K is the number of arms, a* is the optimal arm and independent rewards bounded in [0,1] (Garivier and Cappé, 2011).

Pseudocode for Upper Confidence Bound Strategy

```
Data: number of rounds, number of arms, confidence level
Result: regret after using UCB strategy;
Initialise environment with number of arms;
for number of rounds do
    Pick arm with highest upper confidence bound;
    arm uncertainty = confidence level $\times\sqrt{\frac{\ln N+1}{N}}$;
    arm upper confidence bound = arm realised expectation + arm
      uncertainty;
end
Calculate and return regret
```

Algorithm 2: UCB Strategy

2.3 Thompson Sampling (Bayesian)

The goal of Thompson sampling is to construct an accurate estimate of rewards by building a **probability distribution** based on the rewards we have already received - this is an example of **Bayesian inference**, where we update our models based on new knowledge. More specifically, we construct the **conjugate prior** for the distributions of the arms allowing us to approximately predict our posterior reward distribution.

Using these prior distributions, we choose the next arm to play by sampling from the prior distribution of each arm, selecting the highest value from the samples, and playing the arm which gives this value; we repeat this method after each round, changing our conjugate prior distributions for the following round based on new information we gain. These conjugate prior distributions will continue to change as we gain more information (and therefore confidence) about each arm's distribution. In this section, we shall go through the Thompson sampling methods where the distribution of the arms is different in each case (Agrawal and Goyal, 2012).

2.3.1 Bernoulli Thompson Sampling

As the name suggests, in this case, all the arms have a Bernoulli distribution: therefore we can model our conjugate prior distribution as a Beta(α,β) distribution, where α corresponds to the number of successes (rounds where the arm gives one), and β corresponds to the number of failures (rounds where the arm gives zero). As we have no previous successes or failures in the first round, we shall set both α and β to 1, and after this, we sample using the constructed Beta distribution for each arm, with α and β being updated for each arm the more we play. As we go on, this results in the distributions with higher α values and lower β values being selected more often, maximising our reward.

2.3.2 Gaussian Thompson Sampling

In this case, the arms have a Gaussian (Normal) distribution **with known variance**: this means we can model our conjugate prior distribution as a Normal distribution. However, the parameters are now changed. We shall use the **precision** (τ), which is 1/variance and therefore known, to make the expression to update the parameters after each round easier to manipulate. We name our modelling parameters for the arm before the round μ_0 and τ_0 for the mean and precision respectively. Therefore, the updated precision for the arm after the round is given by

$$\tau_0 \longleftarrow \tau_0 + n\tau,$$

and the updated mean for the arm after the round is given by

$$\mu_0 \leftarrow \frac{\tau_0\mu_0 + \tau\sum_{t=1}^{n}\widehat{r_t}}{\tau_0 + n\tau},$$

where $\widehat{r_t}$ is the reward for each round the arm has been tested in (Agrawal and Goyal, 2013) (which was defined in section 1.3), and n is the number of times the arm has been played. This means the posterior distribution in this case is a normal distribution using the updated parameters above for the precision and mean respectively; this will function as our prior distribution for the next round. Similar to our Bernoulli Thompson sampling model above, as the rounds progress, the precision for each arm increases, and the variance for each arm therefore decreases, meaning that the arms with the highest estimated mean are chosen more often.

2.3.3 Regret for Thompson Sampling

The expected regret for both our Thompson sampling models above has order

$$O\left(\sqrt{NT\ln T}\right)$$

where T is the number of rounds and N is the number of arms (Agrawal and Goyal, 2013). This means that Thompson sampling is a **zero regret** strategy, as the order of the expected regret is such that its gradient becomes flatter as time progresses. This is because as $T \to \infty$, $NT\ln T < NT^2$ (with N being a constant), and the gradient of the square root of NT^2 is constant, meaning the gradient of the expected regret decreases, given that $\sqrt{T\ln T}$ has a decreasing gradient which tends to zero as $T \to \infty$:

$$\frac{\mathrm{d}}{\mathrm{d}T}\left(\sqrt{T\ln T}\right) = \frac{1}{2}T^{-\frac{1}{2}}(\ln T)^{\frac{1}{2}} + \frac{1}{2}T^{\frac{1}{2}}(\ln T)^{-\frac{1}{2}}\frac{1}{T}$$

which then becomes

$$\frac{1}{2}\left(\frac{\ln T}{T}\right)^{\frac{1}{2}} + \frac{1}{2}(T\ln T)^{-\frac{1}{2}}$$

which tends to zero as $T \to \infty$, demonstrating that the algorithm is a zero regret strategy.

Pseudocode for Bernoulli Thompson Sampling Strategy

```
Data: number of rounds, number of arms
Result: regret after using Thompson Sampling strategy;
Initialise environment with number of arms;
Set all arms with same prior distribution Beta(1,1);
for number of rounds do
    Sample distributions for all arms;
    Select arm with highest arg max of Beta(successes+1,failures+1)
      sample;
    Update success and failure numbers of arm
end
Calculate and return regret
```

Algorithm 3: Thompson Strategy

The pseudocode for Gaussian Thompson Sampling is similar except the prior and posterior distributions and parameters are different.

3 Simulations for Stationary Strategies

In our simulations we ran them with many iterations and then took an average as an example. This was to avoid skewed results from anomalies where all arms had similar actual expectations.

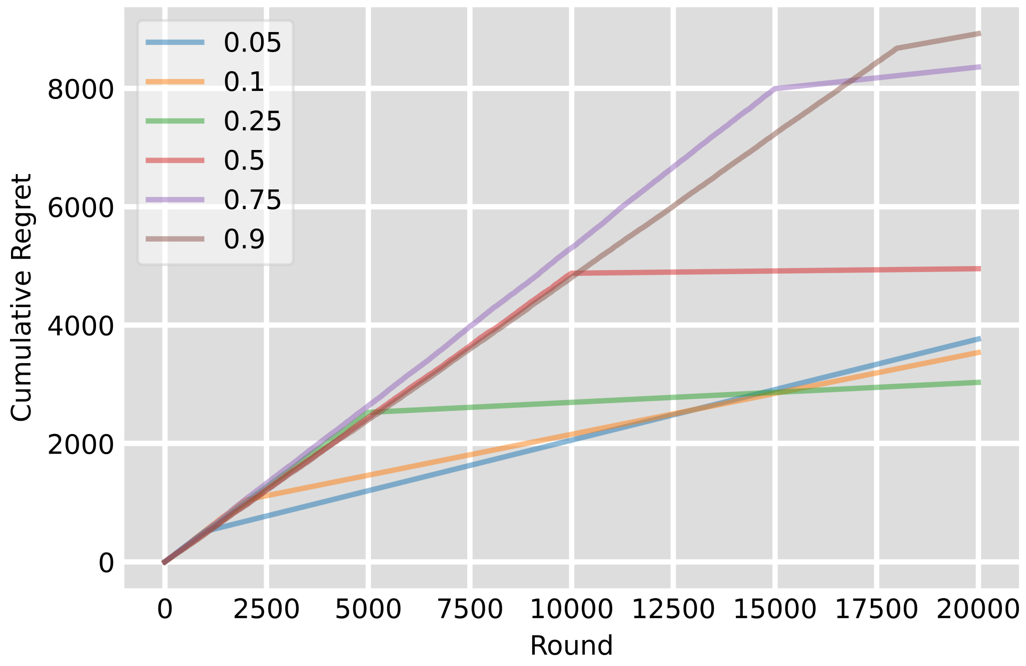

Figure 1: Epsilon-first strategy with varying values of epsilon. 100 arms. 20000 rounds per iteration. Average of 50 iterations

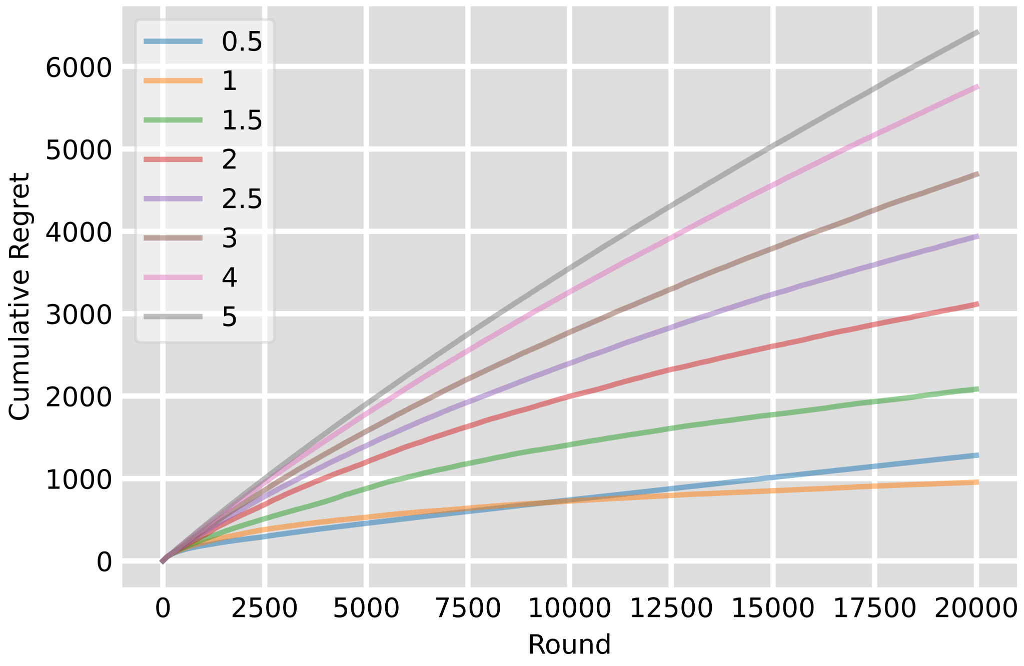

Figure 2: UCB strategy with varying values of confidence level. 100 arms. 20000 rounds per iteration. Average of 50 iterations

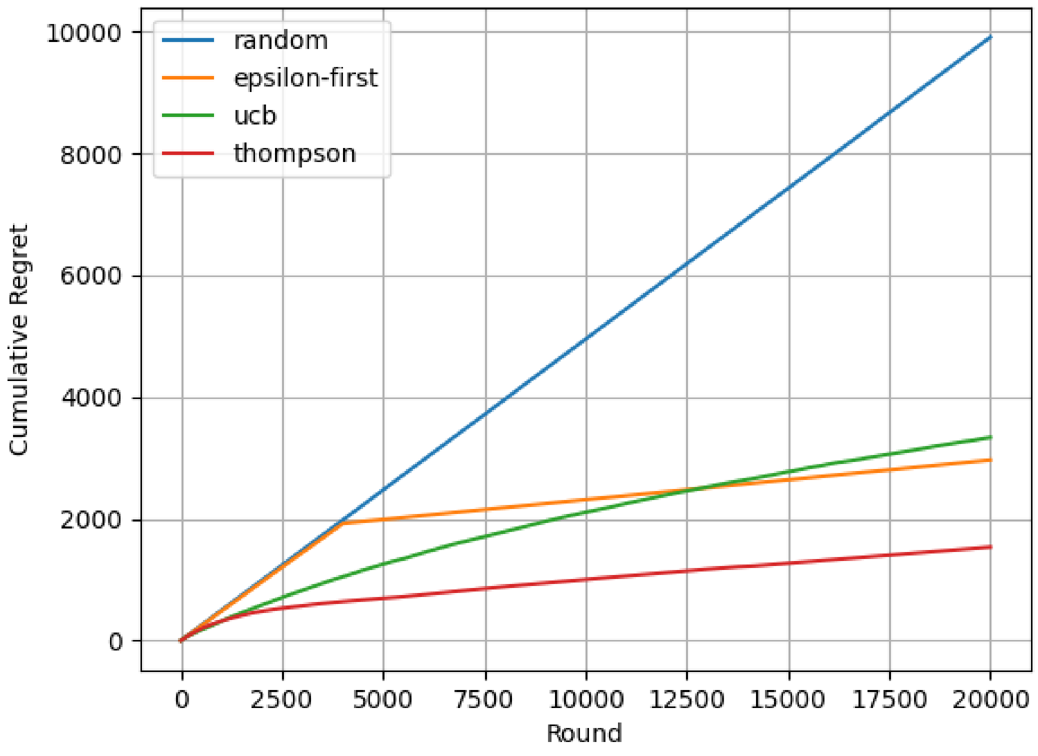

Figure 3: Comparison of Stationary Strategies. Epsilon =0.2, confidence level=1. 100 arms. 20000 rounds per iteration. 20 iterations

4 Analysing Simulations

We see from Figure 1 that the ϵ-first strategy is not necessarily a zero-regret strategy. We see this as the graph for $\epsilon = 0.5$ doesn't increase in cumulative regret after it finishes its exploration phase, however this is not the case for the two ϵ values (0.75 and 0.9) which are greater in Figure 1. This demonstrates to us that a higher ϵ value doesn't mean we are guaranteed to find the arm with the highest actual mean, even with more exploration, due to the fact that we have either had the arm with the highest mean underestimated, or an arm with a lower mean overestimated.
We also see that for our 100 arm plot (Figure 3), Thompson sampling works the best and is a zero-regret strategy, whilst the random strategy is the worst, with its cumulative regret scaling linearly. Here, UCB also scales to a zero-regret strategy, and ϵ-first does not. However, ϵ-first does better than UCB in the number of rounds given, due to a fairly low amount of exploration followed by exploitation (ϵ-value 0.2).
Our figure showing the UCB strategy's cumulative regret with varying confidence levels also demonstrates that there is almost a 'sweet spot' region for confidence levels where the algorithm performs its best (Figure 2); the cumulative regret decreases as the confidence level goes from 0.5 to 1, but after this it continuously increases through to confidence value 5. This highlights how the UCB algorithms with very low confidence intervals (less than 0.5) are more likely to pick the same arm repeatedly and algorithms with very high confidence intervals (>1.5) are more likely to be too broad with the confidence intervals for each arm.
Given more time, we would have plotted more graphs and confidence intervals for each algorithm, whilst exploring more values of ϵ and the confidence level for the ϵ-first and UCB strategies respectively. In addition, in our plots we have only considered the multi-armed bandit problem with Gaussian distributions for each arm where the means have been randomly generated between 0 and 1; we would have also plotted graphs comparing the Gaussian and Bernoulli arm variants for each algorithm, whilst also producing more plots for a greater range of means and comparing them.

5 Strengths and Weaknesses

In this section, we explore which strategies work best and worst in different scenarios, whilst also comparing them to each other.

5.1 ϵ-Greedy

5.1.1 Strengths

The largest strength of epsilon-greedy is it is quite easy to understand and it is not computationally expensive at all, especially compared to UCB and Thompson Sampling - we have found that it has been the easiest strategy to code in our Python environment. After the initial process of exploration (e.g., 10,000 rounds), the algorithm greedily exploits the best option k, ϵ percent of the time. For example, if we set ϵ=0.01, the epsilon-greedy algorithm will exploit the best variant 99% of the time and also will explore random alternatives 1% of the time. This would be rather effective in practice (Mc., 2018).

5.1.2 Weaknesses

Epsilon-Greedy is a pretty powerful algorithm capable of exploring the arms and making close to optimal decisions within it. However, the algorithm does not always identify the absolute best arm. Certainly we can increase the rounds for exploration, but then we are wasting more rounds in searching randomly. This would then further hurt our total reward. Moreover, there is inherent randomness during this process, which means if we were to run the algorithm for one more time it is possible that epsilon-greedy will identify and exploit the best arm. This indicates that epsilon-greedy can get stuck exploiting a sub-optimal arm (Mc., 2018).
In addition, the Epsilon-Greedy strategy's regret will scale linearly, due to a random arm being chosen every so often (this will probably not be the optimal arm in later rounds). Therefore, this strategy will never be a zero-regret strategy, unlike the UCB and Thompson sampling strategies.

5.1.3 Best conditions for ϵ-Greedy algorithm

It is clear to see from the previous sections that the ϵ-Greedy algorithm is far less computationally expensive than the UCB and Thompson Sampling algorithms. Therefore, we can conclude that when we don't have very powerful computers to process algorithms, ϵ-Greedy will be the optimal choice. Otherwise, given that the ϵ-Greedy strategy's regret scales linearly, the UCB and Thompson sampling algorithms are a better choice, given they are zero-regret strategies and will exploit the best arms in later rounds.

5.2 Upper Confidence Bound

5.2.1 Strengths

The UCB method quickly finds the optimal arm and continuously exploits it, while the ϵ-Greedy algorithm, although also finding the optimal action relatively quickly, has too much randomness for larger numbers of rounds. The UCB algorithm does not perform exploration by simply choosing any action selected with a constant probability, but changes its exploration-exploitation balance when collecting more environmental knowledge.
The UCB-1 results from a single simulation are very stable and similar to the average results. This demonstrates that the algorithm can balance exploration and exploitation in a single experiment, can optimize contact allocation to discover the best arm, and then exploit it.
For arbitrary bounded rewards, the KL-UCB algorithm satisfies a uniformly better regret bound than UCB and its variants. Any maximizer can be chosen in this strategy (**?**)

5.2.2 Weaknesses

Many variants need to be considered, for example, the non-stochastic bandit, which makes no assumptions about the reward distribution, cannot be estimated under UCB.
UCB methods can handle bandits with a small number of arms and high reward differences well, but when K becomes larger, their performance deteriorates faster than other algorithms (Kuleshov and Precup, 2014).
The choice of confidence interval also affects the precision. If we choose 95 percent rather than 99 percent, it will generate less specific results.
Although the final solution appears very good, UCB1 converges to a much slower solution than other algorithms (Auer et al., 2002). For a small number of arms (K = 2,5), the reinforcement comparison produces a good average regret per round at the end of 1000 rounds, but starts lag behind other algorithms at larger K values (K = 10, 50). Its total regret is relatively high because it is slower in the beginning, due to the need to estimate the average reward. In general, these results show that there is no advantage to using UCB in practice (Kuleshov and Precup, 2014).

5.2.3 Best conditions for UCB algorithm

Provided that we find the best value for our confidence level, the UCB algorithm works about as well as the Thompson sampling algorithm (which is even more computationally expensive), and this is shown by the plots from our simulations. Therefore, we can conclude that the UCB algorithm is very useful in practice, although this very much depends on whether we choose the right confidence level, as illustrated by the plots above 2; this algorithm works best with a large number of rounds, so the size of its confidence intervals for each arm decrease enough and converge towards the mean for each arm.

5.3 Thompson Sampling

Similar to the Upper Confidence Bound (UCB) algorithm above, Thompson Sampling chooses an arm to play based on the highest estimated reward. Unlike the UCB algorithm, however, the 'highest estimates' in this case depend on the best **sample** from the conjugate prior distributions (explained in section 2.3) for each arm, rather than the upper bound of the confidence interval around the estimated mean for each arm, as the UCB does.

5.3.1 Strengths

By sampling from the prior distributions for each arm, rather than picking the best estimate like ϵ-greedy or the best potential estimate like UCB, we get a better proportion of draws, with which we can model our new prior distributions - this gives us a very good exploration-exploitation balance. In addition, unlike the UCB (which sets a confidence value to control the level of exploration) and ϵ-greedy (which sets an exploration value ϵ) algorithms, a parameter doesn't need to be set for Thompson sampling, meaning that Thompson sampling doesn't depend on how well we choose our parameters.
Furthermore, (although beyond the scope of this project), Gaussian Thompson sampling does not necessarily need to have known precision - if both the mean and precision are unknown, we construct our conjugate priors using the **Normal-Gamma** distribution, with four parameters. This demonstrates that even with very little information about the underlying distribution, Thompson sampling still manages to find optimal arms and a zero regret strategy; this is the case with all distributions which have (known) conjugate priors.

5.3.2 Weaknesses

By trying to estimate the underlying distribution for each arm, and continuously updating prior distributions using posterior results, Thompson sampling ends up being very computationally expensive compared to algorithms like ϵ-Greedy (Mazumdar et al., 2020). Although the computational cost can be decreased using algorithms such as Markov Chain Monte Carlo methods, these will be less accurate as they are numerical approximations (Mazumdar et al., 2020). In addition, for a smaller number of arms with means close to each other, we see that a strategy such as ϵ-first would work better, as Thompson sampling could still choose to play arms using samples from inferior prior distributions, whereas the ϵ-first strategy will locate the arm with the highest expectation far more quickly. It should also be noted that the arm which actually has the highest expectation could also give a really low reward for the first round, meaning that the algorithm may not exploit it for many of the following rounds, which is especially bad with a low number of arms and rounds. Furthermore, in this project, we have only considered arm distributions where the prior and posterior are **conjugate distributions** - when this is not the case, Thompson sampling becomes even more computationally expensive, as we have to consider different prior and posterior distributions with changing parameters (Zhou et al., 2018). Finally, Thompson sampling requires us to know the form of the underlying distribution of each arm so we can construct prior distributions effectively; in practice, however, it is unlikely that we will know these distributions, meaning that other methods (such as UCB or ϵ-first) would work better without this information.

5.3.3 Best conditions for Thompson sampling algorithm

From this, we can say that Thompson sampling works best when there are a large number of rounds and the prior distributions for each arm are known, as this would result in more accurate prior distributions for each round over the long run. A smaller number of arms will also maximize the efficiency of Thompson sampling, as this means there is less time spent exploring each arm's distributions and more time exploiting the arm with the highest estimated mean. As Thompson sampling is a zero-regret strategy, we can also say that after a very large number of rounds, the cumulative regret will almost cease to increase.

6 Non-stationary Strategies

The strategies we have explained above work well when the underlying distributions for each arm don't change, but what happens when they do? In this section, we go through strategies for different non-stationary distribution scenarios, and explain their strengths and weaknesses.

6.1 Discounted UCB

Discounted UCB (D-UCB), which was proposed by (Kocsis and Szepesvári, 2006), is a way specifically implying in local channels due to the varying time of local environment. The principle is adding a discount factor, which is a geometric moving average over the samples, to the UCB Tuned (UCBT) algorithm, and the average reward is given by

$$\bar{X}_t(\gamma,i) = \frac{\sum_{k=1}^{t} \gamma^{t-k} X_k(i)}{N_t(\gamma,i)}, \quad N_t(\gamma,i) = \sum_{k=1}^{t} \gamma^{t-k}, \quad I_k = i$$

where the $\gamma_i \in [0,1]$ represents the change rate of i and is also the discount factor, based on the construction of an UCB ($\bar{X}_t(\gamma,i) + c_t(\gamma,i)$). Then define the padding function as

$$c_t(\gamma,i) = 2B\sqrt{\frac{\delta \log n_t(\gamma)}{N_t(\gamma,i)}},$$

where δ is an appropriate parameter.
The algorithm is also proved to achieve the upper-bounded regret by $O\left(\sqrt{\gamma_i T}\log T\right)$ as the lower bound is $\Omega\left(\sqrt{T}\right)$(Liu et al., 2018). As well, the algorithm is shown to be more weighted for new data rather than old data.

6.2 Sliding-Window UCB

The Sliding-Window UCB (SW-UCB) algorithm is similar to the D-UCB, but it updates observation in UCB1 algorithm and consider average reward only within a window with fixed length of l, where l decreases when the change rate of environment increases. The maximum regret of this algorithm is proven to be $O\left(\sqrt{\gamma_i T \log T}\right)$(Garivier and Moulines, 2008).
Compared with D-UCB, the computational complexity of SW-UCB is linear in time and does not involve j. However, SW-UCB requires to store the last j actions and rewards at each time t in order to efficiently update $\hat{n}_i$ as well as $\hat{z}_i$ (Garivier and Moulines, 2008).

6.3 GLR-klUCB

GLR-klUCB can be considered as a klUCB algorithm allowing for some restarts on the different arms. This algorithm combines the efficient bandit algorithm klUCB, with a parameter- free, change-point detector, the Bernoulli Generalized Likelihood Ratio Test.
Regret is upper-bounded by $O\left(\Upsilon_T \sqrt{T \log(T)}\right)$ if the number of changed points Υ_T is unknown and $O\left(\sqrt{\Upsilon_T T \log(T)}\right)$ if Υ_T is known (Besson et al., 2020).
For a simple model with global changes, choosing parameter α and δ as $\sqrt{\frac{\log T}{T}}$ and $1/\sqrt{T}$ respectively, the regret becomes $O\left(\frac{K}{(\delta^{change})^2}\Upsilon_T\sqrt{T\log(T)} + \frac{(K-1)}{\delta^{opt}}\Upsilon_T \log(T)\right)$. Here δ^{opt} denote the smallest value of a sub-optimality gap on one of the stationary segments, δ^{change} be the smallest magnitude of any change point on any arm and K is the number of arms (Besson et al., 2020).
This strategy is efficient for the piece-wise stationary bandit problem. This actively adaptive method attains state-of-the-art regret upper-bounds when tuned with a prior knowledge of the number of changes Υ_T , but without any other prior knowledge on the problem (Besson et al., 2020).

6.4 Limited-Memory DSEE

Limited-Memory DSEE is also an algorithm which works for a non-stationary bandit, including both exploration and exploitation phases. In the n-th exploration phase, each arm is sampled L(k)=$\lceil \gamma \ln(n^{\rho} l b \rceil$ times, and in the n-th exploitation phase, the arm is getting to have highest sample mean in the n-th exploration phase sampled $an^{\rho}l - NL(n)$ times, where ρ, γ, a, b, l are parameters tuning on the characteristics of environment.
This is a similar algorithm to DSEE algorithm, wherein the length of exploitation phase is exponentially increased with the epoch number and we estimate the mean rewards referring to all the data collected in the previous phase of exploration (Wei and Srivatsva, 2018).

7 Implementation

We created an environment to test out our strategies and produce the regret plots for the simulations in section 3. This was done using Python and Jupyter Notebooks. To create the environment, we created a class of all the arms, and each lever arm was a class. We then defined the strategies as functions, so they could be called by the Jupyter notebook files to run the simulations.
To run our simulations we used Amazon Web Services servers and set up an email feature such that when the simulation was complete it would email us with the regret plot and raw data of the simulation.

7.1 Project Flow

Initially we focused on the stationary form of the problem. We started by creating our testing environment and implementing a few basic strategies we had thought of, one of which turning out to be the ϵ-first strategy. We then researched through a host of strategies and chose to implement UCB and Thompson Sampling as well. Along with this, other members of the group would research more into theoretical results of these strategies to confirm our results from the simulations.

8 Contributors

8.1 Adhavan Sashikumar

Most of my work in this project has come in the report, and the rest with coding algorithms into our Python environments and producing plots. I have written all the sections on Thompson Sampling (other than its pseudocode), including the explanation of the algorithm, its regret, its strengths and weaknesses and the situations in which it can be best utilised. I also wrote the section on regret in the introduction, to explain how we measure the strength of an algorithm. I have also researched and written on the other strategies; I wrote the sections explaining the random strategy and epsilon greedy algorithms as well as the 'Analysis of Simulations' segments, whilst adding to the sections on Strengths and Weaknesses for the other strategies. In addition, I have helped to write the algorithms for Thompson sampling, and produced the plot for Figure 7 in our Python environment. Furthermore, I added a section to the UCB algorithm titled 'Best conditions for UCB algorithm', which I then wrote on.

8.2 Diandian Chen

My key contribution in this project has come in the report. I have written all the UCB strategies under the stationary strategies part including Hoeffding's Inequality, UCB1 and KL-UCB as well as their strengths and weaknesses in the report. I also wrote the section on GLR-klUCB as a non-stationary strategy with its strengths and weaknesses. Furthermore, I advised to use latex beamer to generate the slides for presentation then we can get the slides together conveniently and use latex mathematical labels directly.

8.3 Rui Tang

My contribution in this project is in this report.
I have mainly written the epsilon-first strategy under the stationary strategies part and also discovered some strategies content under the non-stationary strategies part, including sections of the Discounted-UCB, Sliding-window UCB, as well as some contents of the Limited-memory DSEE algorithm, and put them on the report with its principles, strengths and weaknesses.

8.4 Shivam Patel

My key contribution in this project was programming the environment and simulations. Coming into the project, I had completed various other projects and hackathons which required programming skills so was excited when this opportunity presented itself. I was also familiar with git and implemented the use of this both for our code-base, and our report within our group to ensure an effective workflow. Furthermore, from my experience of latex, I set up the report, referencing methods and taught our group how to use a .bib file and cite to accurately reference work and ensure consistency within the group. I also took the role of 'group leader' on and delegated tasks to each member of the group and ensured we were working at an efficient pace, which was important in such a short timeline.

8.5 Xinqi Yao

My key contribution in this project has come in the report. I have written part of the epsilon-greedy strategy, especially epsilon-first strategy, as well as its strengths and weaknesses in the report. Moreover, I wrote the sliding-window UCB algorithm under the non-stationary strategy section and compared it with Discounted UCB to summarise its characteristics.

9 Acknowledgements

We would firstly like to thank our project supervisor Ciara M Pike-Burke for her fantastic and continued support throughout this project. We would also like to thank Liz Kinsey and the mathematics department for organising the module.
We are also grateful for the following organisations and people that have enabled us to accomplish this:

- Python (Van Rossum and Drake, 2009)
- Jupyter Notebooks (Kluyver et al., 2016)
- Python modules
  - numpy (Harris et al., 2020)
  - scipy (Virtanen et al., 2020)
  - matplotlib (Hunter, 2007)

10 Appendix

To see our codebase, head to our GitHub repository

List of Algorithms

List of Figures

References

Agrawal, S. and Goyal, N. (2012). Analysis of thompson sampling for the multi-armed bandit problem. In Mannor, S., Srebro, N., and Williamson, R. C., editors, *Proceedings of the 25th Annual Conference on Learning Theory*, volume 23 of *Proceedings of Machine Learning Research*, pages 39.1–39.26, Edinburgh, Scotland. JMLR Workshop and Conference Proceedings. pages 9

Agrawal, S. and Goyal, N. (2013). Further optimal regret bounds for thompson sampling. In Carvalho, C. M. and Ravikumar, P., editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 99–107, Scottsdale, Arizona, USA. PMLR. pages 10

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2):235–256. pages 8, 17

Besson, L., Kaufmann, E., Maillard, O.-A., and Seznec, J. (2020). Efficient change-point detection for tackling piecewise-stationary bandits. pages 21

Bouneffouf, D. and Rish, I. (2019). A survey on practical applications of multi-armed and contextual bandits. *CoRR*, abs/1904.10040. pages 4

Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and non-stochastic multi-armed bandit problems. *CoRR*, abs/1204.5721. pages 7

Chan, H. P. (2020). The multi-armed bandit problem: An efficient nonparametric solution. *The Annals of Statistics*, 48(1):346 – 373. pages 8

Chen, W., Wang, Y., and Yuan, Y. (2013). Combinatorial multi-armed bandit: General framework and applications. In *International Conference on Machine Learning*, pages 151–159. PMLR. pages 4

Collier, M. and Llorens, H. U. (2018). Deep contextual multi-armed bandits. *CoRR*, abs/1807.09809. pages 6

Garivier, A. and Cappé, O. (2011). The kl-ucb algorithm for bounded stochastic bandits and beyond. In *24th annual conference on learning theory*. JMLR Workshop and Conference Proceedings. pages 8

Garivier, A. and Moulines, E. (2008). On upper-confidence bound policies for non-stationary bandit problems. pages 20

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585:357–362. pages 25

Hoeffding, W. (1994). *Probability Inequalities for sums of Bounded Random Variables*, pages 409–426. Springer New York, New York, NY. pages 7

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95. pages 25

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press. pages 25

Kocsis, L. and Szepesvári, C. (2006). Discounted ucb. In *2nd PASCAL Challenges Workshop*, volume 2. pages 20

Kuleshov, V. and Precup, D. (2014). Algorithms for multi-armed bandit problems. *CoRR*, abs/1402.6028. pages 17

Liu, F., Lee, J., and Shroff, N. (2018). A change-detection based framework for piecewise-stationary multi-armed bandit problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1). pages 20

Mazumdar, E., Pacchiano, A., Ma, Y., Bartlett, P. L., and Jordan, M. I. (2020). On thompson sampling with langevin algorithms. *CoRR*, abs/2002.10002. pages 18

Mc., C. (2018). Solving multiarmed bandits: A comparison of epsilon-greedy and thompson sampling. pages 16

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535. pages 4

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA. pages 25

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In Gama, J., Camacho, R., Brazdil, P. B., Jorge, A. M., and Torgo, L., editors, *Machine Learning: ECML 2005*, pages 437–448, Berlin, Heidelberg. Springer Berlin Heidelberg. pages 4, 5, 6

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272. pages 25

Wei, L. and Srivatsva, V. (2018). On abruptly-changing and slowly-varying multiarmed bandit problems. In *2018 Annual American Control Conference (ACC)*, pages 6291–6296. pages 21

Zhou, Y., Zhu, J., and Zhuo, J. (2018). Racing thompson: an efficient algorithm for thompson sampling with non-conjugate priors. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 6000–6008. PMLR. pages 19