École Polytechnique Fédérale de Lausanne

Bandit Algorithms For Recommendation Systems

Author: Thösam Norlha-Tsang
Supervisor: Dr. Suryanarayana Sankagiri
Supervisor: Prof. Matthias Grossglauser
Date: June 2023

INDY Lab, Information and Network Dynamics
School of Computer and Communication Sciences (IC)
BC building
Station 14
CH-1015 Lausanne

Abstract

This project explores and analyzes different types of bandit algorithms in the context of recommendation systems. The goal is to investigate their effectiveness and compare their performance in various scenarios. The report provides insights into the strengths and limitations of each algorithm in the context of recommendation systems.

Contents

1 Acknowledgments

I wish to convey my sincere appreciation to Dr. Suryanarayana Sankagiri for his unwavering assistance, valuable counsel, and the generous time he devoted to this research project. His expertise, elucidations, and mentorship have been very important in enhancing my comprehension of the subject and surmounting obstacles. I am deeply thankful for his patience and dedication to my academic development.

I would like to express my gratitude to Professor Matthias Grossglauser, the head of the Information and Network Dynamics Laboratory, for the invaluable knowledge I have accrued throughout our interactions during the Internet Analytics course and this project.

Finally, I would like to express my gratitude to the entire project team for their participation, cooperation, and support. The opportunity to work alongside such gifted individuals has been an honor, and I appreciate the chance to learn from and develop with them.

2 Introduction

The goal of this bachelor's project was to explore and analyze different types of bandit algorithms within the context of recommendation systems.

The project started with an immersive study of multi-armed bandits, focusing on understanding their underlying principles and significance in data science. Various algorithms, including Explore-Then-Commit, Epsilon-Greedy, and Upper Confidence Bound (UCB), were implemented. These algorithms serve as vital components in the decision-making process by enabling efficient exploration and optimal exploitation when each arm has a different probability of success and a reward associated with it. This initial phase aimed to gain familiarity with bandit algorithms and evaluate their performance and behavior in controlled environments, revealing their respective strengths and weaknesses.

Building upon this foundation, the project's second phase focused on linear bandits, which extends the traditional multi-armed bandit problem to scenarios where each arm has a linear relationship with its corresponding rewards. Implementing the Linear UCB algorithm provided valuable insights into its effectiveness in recommendation systems.

The project's third phase involved utilizing actual datasets from Amazon Music, a rich set of users' reviews and musics information. The integration of real-world data presented exciting challenges, necessitating adaptation of the datasets to effectively work with the Linear UCB algorithm. The primary objective was to evaluate the algorithm's performance in a practical context and assess its potential for real-world deployment.

In summary, this research project navigates through the domain of multi-armed bandits, and linear bandits with both simulated and real-world scenarios. By evaluating the performance and effectiveness of bandit algorithms, particularly in the context of recommendation systems, this project seeks to deepen our understanding of recommendation systems using bandit algorithms.

3 Related Work

In this section, the discussion revolves around the key references that have contributed to the development of the theoretical foundations, algorithms, and evaluation methods relevant to the project. Each reference is described by emphasizing its significance and relevance to the work at hand.

1. "Bandit Algorithms" by Tor Lattimore and Csaba Szepesvári This book provides an in-depth treatment of bandit algorithms, covering various aspects such as stochastic linear bandits, pure exploration... The specific chapters relevant to our project are:
   (a) Chapter 1: Introduction This chapter introduces the concepts of bandit algortihms with various real-world examples/applications and also the concept of regret.
   (b) Chapter 6: The Explore-Then-Commit Algorithm This chapter introduces and explain the explore-then-commit algorithm.
   (c) Chapter 7: The Upper Confidence Bound Algorithm This chapter introduces and explain the upper confidence bound algorithm.
   (d) Chapter 19: Stochastic Linear Bandits This chapter focuses on stochastic linear bandits, providing a detailed treatment of their theoretical properties, including regret analysis. This material is important for understanding the performance guarantees of linear bandit algorithms but also the practical implications of using linear UCB in our project.
2. "Introduction to Linear Bandits" by Yoan Russac This reference introduces the concept of linear bandits, a generalization of the multi-armed bandit problem that models the reward function as a linear function of item features. The text covers the fundamentals of stochastic multi-armed bandits and extends the discussion to linear bandits. Linear bandits serve as the basis for contextual dueling bandits, which incorporate contextual information to improve the quality of recommendations. This reference is essential for understanding the theoretical foundations of linear bandits and their relevance to our project.
3. "Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms" by Li et al. This paper discuss how to evaluate bandit policy using offline data. They work with bandit algorithms: ε-greey, UCB and LinUCB. This reference helps us understand how to evaluate algorithms in offline mode.

4 Description of Dataset

During the first part of the project, simulated data was primarily used for both multi-armed bandits and linear bandits. However, a transition was made from simulated to real-world data in order to conduct experiments. An appropriate dataset on Amazon music was obtained from the 'Datasets-for-Recommender-Systems' GitHub repository maintained by the *caserec* project. The dataset consists of two files. The first file, 'amazon_music_metadata.csv', is in CSV format and contains music IDs (asin) in the first column, song titles in the second column, and approximately 400 columns representing music genres. Each genre is represented by a 0 or 1 associated with the title, allowing for multiple genres per music entry. This dataset comprises 6,932 music entries. The second file, 'Digital_music_5.json', is in JSON format and contains reviews associated with each asin/song. The columns include reviewer ID, asin, helpful votes, review text, overall rating out of 5, summary, and review time. The dataset comprises 64,706 review entries.

The file 'amazon_music_metadata.csv' is used in order to get the feature vectors of all musics. To reduce the dimensionality of the first file, PCA (Principal Component Analysis) and Isomap techniques were employed. As a result, 6,932 distinct music IDs were obtained along with their respective feature vectors (2D, 3D, 5D, 10D). A bias term of 1 was also added to all the feature vectors.

The file 'Digital_music_5.json' is used to train the linear bandits because it already contains the true ratings given by each user to the musics they reviewed. However, it was observed that there are reviews for songs that are not present in 'amazon_music_metadata'. Therefore, those reviews had to be removed from the JSON file since in order to train the linear bandits, we need songs that have a feature vector. This removal results in a reduction from 64,706 to 51,796 review entries in 'Digital_music_5.json'.

In order to train the linear bandits algorithm on this dataset, it was necessary to extract users along with their respective reviews. Users with a minimum of 50 reviews and at least 1.0 variance were selected. Consequently, out of a total of 5,148 reviewers, only 16 reviewers remained. The data for these reviewers, along with their corresponding ratings for the reviewed songs, were exported to a CSV file.

5 Algorithms

5.1 Multi-armed bandits

The multi-armed bandits problem is a fundamental concept in decision theory and reinforcement learning that involves a trade-off between exploration and exploitation. To illustrate this concept, the analogy of a casino with multiple slot machines can be considered, each referred to as an "arm" in this context. The goal is to maximize the earnings by strategically choosing which arm to pull.

In the beginning, the player has no information about the payout probabilities of each machine. An exploration-exploitation dilemma arises when deciding between trying out new machines to gather information about their payout rates (exploration) and focusing on the machine that appears to have the highest payout based on the current knowledge (exploitation).

As the player continues to play, more information is gathered and the player can learn from previous experiences and aim to find the best strategy that maximizes the accumulated profit over time.

This trade-off between exploration and exploitation is at the core of the multi-armed bandits problem and has many important implications in various domains such as online advertising and clinical trials.

5.1.1 Explore-Then-Commit

The Explore-Then-Commit algorithm is a type of multi-armed bandit algorithm that starts by exploring each option or arm to gather information about their potential rewards. The algorithm evenly plays each arm during the exploration phase for a fixed number of times and stores the reward obtained for each arm. After the exploration phase, the algorithm selects the best arm based on the collected data and commits to playing only that arm for the remaining duration of the experiment. The commit phase ensures that the algorithm does not waste time exploring sub-optimal arms and instead exploits the best arm to maximize rewards.

5.1.2 Explore-Then-Commit with elimination rounds

The efficiency of the Explore-Then-Commit algorithm can be improved by eliminating the worst-performing arm after a predetermined number of plays. This approach ensures that further resources are not wasted on an underperforming arm. The algorithm follows a process where the number of elimination rounds is determined, with each round involving a specific number of pulls for each arm. At the conclusion of each elimination round, the arm with the poorest performance is eliminated, avoiding unnecessary money/resources spent on it.

5.1.3 Epsilon-Greedy (ε-greedy)

The Epsilon-Greedy algorithm is a popular, simple multi-armed bandit algorithm that starts by initializing the action-value array to zero in order to store the estimated rewards for each arm (the average of all rewards received for each arm). During each play, the algorithm selects an arm to play using the following strategy: with probability ε, it selects an arm uniformly at random (exploration), while with probability 1-ε, the algorithm selects the arm with the highest estimated reward (exploitation). The epsilon value is typically set to a small value, such as 0.1, to ensure a balance between exploration and exploitation. After each play or round, the algorithm updates the estimated reward for the played arm based on the received reward. This update is done using a simple formula that considers the previous estimated and received rewards.

The formula for updating the action-value estimate is:

$$Q(a) \leftarrow \frac{1}{N(a)}(Q(a) * (N(a) - 1) + R)$$

Alternatively, this formula can be simplified to:

$$Q(a) \leftarrow Q(a) + \frac{1}{N(a)}(R - Q(a))$$

- $Q(a)$ represents the current action-value estimate for arm a.

- $N(a)$ is the number of times arm a has been selected so far.
- R is the observed reward obtained after selecting arm a.

5.1.4 Upper Confidence Bound (UCB)

The UCB algorithm is a popular algorithm that effectively balances exploration and exploitation, making it well-suited for scenarios with limited resources. Unlike the Explore-Then-Commit algorithm, which requires a separate exploration phase and additional resources to perform optimally, UCB achieves this balance without the need for such dedicated exploration. The UCB algorithm works as follows: the algorithm initializes upper and lower bounds to $+\infty$ and $-\infty$ respectively. At each step, the algorithm selects the arm with the highest upper bound, which is expected to yield the highest reward. It then communicates this arm choice to the environment which answers with the true reward. Based on the observed rewards, the algorithm updates the upper and lower bounds for the selected arm. As the algorithm continues to play arms, the confidence bounds gradually shrink. The upper bounds and lower bounds for an arm a are calculated as follows:

$$\text{Upper Bound}(a) = \text{Estimated Reward}(a) + c \times \sqrt{\frac{1}{N(a)}}$$

- Estimated Reward(a) is the estimated reward for arm a
- c is a constant that determines the level of exploration,
- $N(a)$ is the number of times arm a has been played.

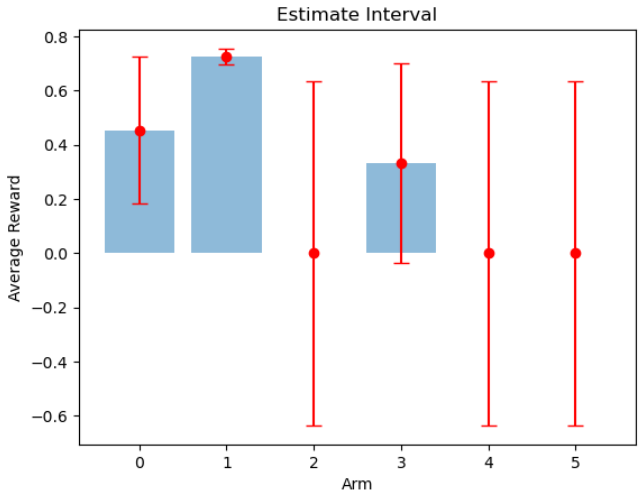


Figure 1: Average reward of each arm estimated by UCB after 1000 steps, probability distribution from (Figure 2, b)

5.1.5 Upper Confidence Bound variation with elimination rounds

In the Explore-Then-Commit algorithm, an optimization has been identified to save resources. However, to utilize this optimization, it is necessary to specify the number of elimination rounds and the number of pulls per arm for each round in advance. To address this limitation, an alternative approach can be employed using upper and lower confidence bounds. This modified algorithm resembles the upper confidence bound method with a slight modification.

During each round, all arms that have not been eliminated are pulled once, then the upper and lower bounds are calculated. An arm is then eliminated if its upper bound falls below the lower bound of the best arm (which is determined as the one with the highest expected reward). This continuous elimination process allows for more resource conservation.

5.2 Linear bandits

Linear bandits are a class of algorithms that extend the concept of multi-armed bandits to make informed recommendations in scenarios where each arm is associated with a linear relationship between its features and the expected rewards.

5.2.1 Example

To illustrate this, the example of a music recommendation system can be considered. In this context, the arms represent different musics and the goal is to recommend musics to users that they are likely to enjoy. Each arm is characterized by a feature vector representing its genre or style such as classical, jazz, blues, country and rock. The feature vector could in this scenario be a binary representation indicating the presence or absence of a particular genre for a given arm.

The feature vector of an arm can be represented as:

$$\text{Feature vector of an arm} = [\text{classical}, \text{jazz}, \text{blues}, \text{country}, \text{rock}]$$

For example, if arm a corresponds to jazz music, its feature vector could be:

$$\text{Feature vector of arm } a = [0, 1, 0, 0, 0]$$

In addition to the feature vectors, each user has an underlying true theta vector that represents their preferences for different music genres. The true theta vector captures the user's intrinsic preferences, which are not directly observable. The linear bandits algorithm aims to estimate the user's true theta vector by leveraging the observed ratings and the corresponding feature vectors of the recommended musics.

Initially, the linear bandit has limited information about the preferences of the user. The user first calculates the predicted rating (reward) of each arm and recommend the arm with the highest expected rating to the user.

$$\text{arm recommended} = \arg\max_{a \in A_t} \left(\hat{\theta} \cdot \text{Feature vector of arm } a \right)$$

As the user listens to music and rates them, the linear bandits algorithm continually updates and refines its estimated theta vector. This estimation is based on the user's ratings and the feature vectors of the recommended arms. By incorporating this feedback loop, the algorithm aims to improve the accuracy of its predictions and provide personalized recommendations to the user.

5.2.2 Formulas

In the context of recommendation systems, the objective is to maximize the expected reward obtained from the system. This is achieved by selecting the best actions that result in the highest expected reward. However, since the true parameter θ^* is unknown, the algorithm must estimate it based on the observed rewards.

In each round t, a set of K actions A_t is available. By selecting the context $A_t = \{A_{t,1}, \ldots, A_{t,K}\}$, the algorithm observes the reward X_t, which is the inner product of the chosen action feature vector and the true parameter vector θ^*, with the addition of random noise η_t.

$$X_t = \langle A_t, \theta^* \rangle + \eta_t$$

- Assumption on the noise: η_t are supposed to be i.i.d and normally distributed: $\eta_t \sim \mathcal{N}(0, 1)$

The best action at time t is determined by finding the action that maximizes the inner product with θ^*, as denoted by A_t^*.

The best action at time t:

$$A_t^* = \arg\max_{a \in A_t} \left(\langle \theta^*, a \rangle \right)$$

5.2.3 Regret

To evaluate the performance of the recommendation system, we introduce the concept of regret. The regret measures the difference between the maximum expected reward and the observed reward obtained using the chosen actions. Mathematically, this can be expressed as:

$$R_t = \mathbb{E}\left[\sum_{s=1}^{T} \max_{a \in A_t} \langle a, \theta^* \rangle - \sum_{t=1}^{T} X_t\right]$$

The goal is to minimize the regret, indicating that the recommendation system is selecting actions that are closer to the optimal ones. By rearranging and manipulating the formulas, we can express the objective as minimizing the expected regret:

$$\max \mathbb{E}\left[\sum_{t=1}^{T} X_t\right] \iff \min \mathbb{E}\left[\sum_{s=1}^{T} \max_{a \in A_t} \langle a, \theta^* \rangle - \sum_{t=1}^{T} X_t\right] \iff \min \mathbb{E}\left[\sum_{t=1}^{T} \max_{a \in A_t} \langle a - A_t, \theta^* \rangle\right]$$

5.2.4 Estimating θ^*

Now the question is: how to choose an action A_t at time t to minimize the regret?

The algorithm has played t-1 rounds where the actions A_1, ..., $A_t - 1$ have been selected and the rewards X_1, ... $X_t - 1$ have been collected.

The linear bandits can estimate θ^*, based on those observations, using Regularized Least-Squares Estimator :

$$\hat{\theta}_t = \arg\min_{\theta \in \mathbb{R}^d} \sum_{s=1}^{t-1} (X_s - A_s^T \theta)^2 + \frac{\lambda}{2} \|\theta\|_2^2$$

By manipulating the formulas, we can derive a closed-form solution for $\hat{\theta}_t$:

$$\hat{\theta}_t = (V_{t-1})^{-1} \sum_{s=1}^{t-1} A_s X_s$$

where

$$V_{t-1} = V_0 + \sum_{s=1}^{t-1} A_s A_s^T$$

$$V_0 = \lambda I_d$$

with I_d being the $d \times d$ identity matrix.

The matrix V_{t-1} serves as a measure of the cumulative uncertainty in the estimation of θ^* up to round $t-1$. As more observations are made, the matrix V_{t-1} becomes a more accurate representation of the true covariance of the parameter estimates. The inverse of V_{t-1} allows us to obtain the estimate $\hat{\theta}_t$ by combining the weighted sum of the feature vectors A_s and the observed rewards X_s from previous rounds.

5.2.5 Linear Upper Confidence Bound (LinUCB)

The Linear Upper Confidence Bound algorithm is a variant of the UCB algorithm designed for the linear bandits problem. It follows a similar logic as UCB algorithm in the multi-armed bandits scenario but it incorporates the linear relationship between the features and expected rewards of each arm.

Given the previous formulas, the algorithm aims to choose the action A_t that maximizes:

$$A_t = \arg\max_{a \in A_t} \left(\langle \hat{\theta}_t, a \rangle\right)$$

However, this greedy approach alone may not yield optimal results due to the lack of exploration. To address this, the Linear Upper Confidence Bound algorithm (LinUCB) incorporates a confidence ellipsoid C_t belonging to $\mathbb{R}^d$ that contains the unknown parameter θ^* with high probability, given the observations available up to time $t-1$. The revised formula becomes:

$$A_t = \arg\max_{a \in A_t} \max_{\theta \in C_t} (\langle \theta, a \rangle)$$

where C_t is a particular confidence ellipsoid that captures the uncertainty about θ^* based on the available information.

The confidence ellipsoids that contain the θ^* vector can be defined as:

$$C_t = \left\{ \theta \in \mathbb{R}^d : \|\theta - \hat{\theta}_t\|_{V_{t-1}} \leq \beta_{t-1} \right\}$$

$$\beta_t = \log(t + 1.0) + 1.0$$

With this choice of confidence ellipsoid, the previous optimization problem is equivalent to :

$$A_t = \arg\max_{a \in A_t} \left(\langle \hat{\theta}_t, a \rangle + \beta_{t-1} \|a\|_{V_{t-1}^{-1}} \right)$$

where $\hat{\theta}_t$ is the estimated θ vector at time t and $\| \cdot \|_{V_{t-1}}$ denotes the norm induced by the matrix V_{t-1}.

5.2.6 Algorithm in the codebase

- $\mathbf{V}_t$ of shape (n_features, n_features) representing the covariance matrix initialized as λ_{param} multiplied by the identity matrix of size n_{features}.
- **sum_A_s_X_s** a vector of shape (n_features,) is the sum of outer products between the chosen action's feature vector $\mathbf{x}_a$ and the observed reward r_a of arm a, initialized as zeros.

$$\textbf{sum\_A\_s\_X\_s} = \sum_{\text{rounds}} \mathbf{x}_a \cdot r_a$$

For each round, the algorithm selects the action (arm) with the highest upper bound value, computed based on the estimated parameter vector and the uncertainty associated with it. The upper bound for arm a is given by:

$$\text{UCB}(a) = \hat{\theta}^T \mathbf{x}_a + \sqrt{\beta_t} \sqrt{\mathbf{x}_a^T \mathbf{V}_t^{-1} \mathbf{x}_a}$$

- $\hat{\theta}^T$ is the transpose of the estimated $\hat{\theta}$ vector
- $\mathbf{x}_a$ is the feature vector of arm a
- $\mathbf{V}_t^{-1}$ is the inverse of the covariance matrix $\mathbf{V}_t$
- β_t is the UCB parameter that determines the level of exploration

After recommending an arm, the algorithm updates its variables based on the observed reward from the user and the feature vector associated with that arm. It updates the covariance matrix, V_t, by adding the outer product of the chosen arm's feature vector with itself. The algorithm also updates the sum of outer products, sum_{AsXs}, by adding the outer product of the chosen action's feature vector with the observed reward of the current round. Finally, it computes a new estimate of the true parameter vector, $\hat{\theta}$, using the inverse of the updated covariance matrix, $(\mathbf{V}_t)^{-1}$ and the and the sum of outer products $sum\_A\_s\_X\_s$.

The covariance matrix update:

$$\mathbf{V}_t \leftarrow \mathbf{V}_t + \mathbf{x}_a \mathbf{x}_a^T$$

The sum of outer products update:

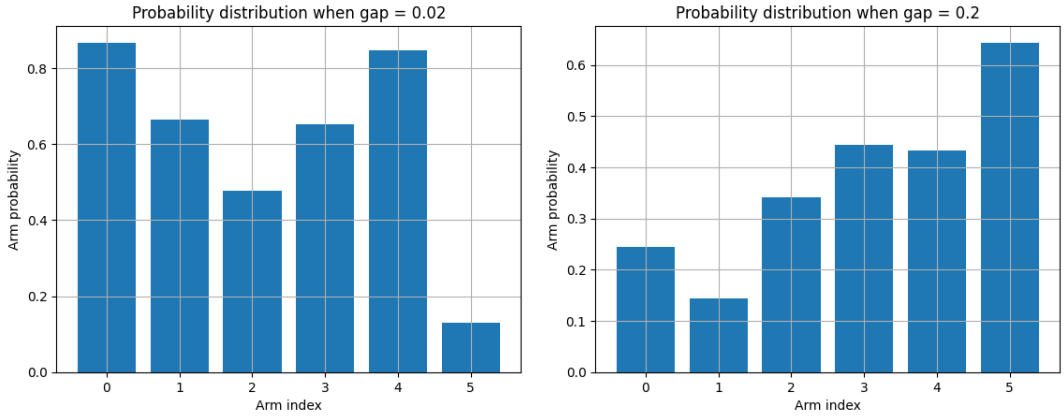
$$\text{sum\_A\_s\_X\_s} \leftarrow \text{sum\_A\_s\_X\_s} + \mathbf{x}_a \cdot \text{reward}$$

The estimate of the true parameter vector update:

$$\hat{\theta} \leftarrow (\mathbf{V}_t)^{-1} \cdot \text{sum\_A\_s\_X\_s}$$

6 Results

6.1 Multi-armed bandits



(a) Probability of success of each arm when gap between best arm and second best arm is 0.02

(b) Probability of success of each arm when gap between best arm and second best arm is 0.2

Figure 2: Probability of success of each arm with a small gap and a big gap

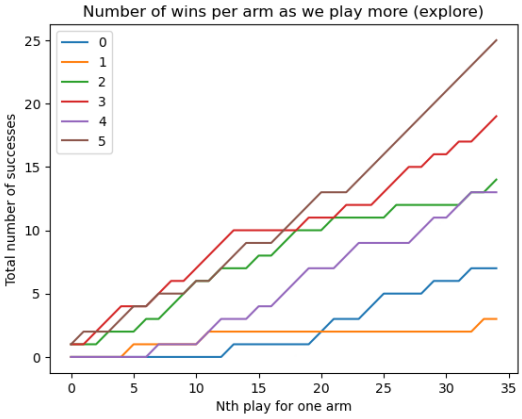
6.1.1 Explore-Then-Commit

The Explore-Then-Commit algorithm was executed for a duration of 700 time steps, with 30% of the allocated time dedicated to exploration and the remaining 70% dedicated to exploitation.
The experiment involved a set of 6 arms that followed the identical probability distribution as depicted in Figure 2(b), they all yield a reward of 1.0.

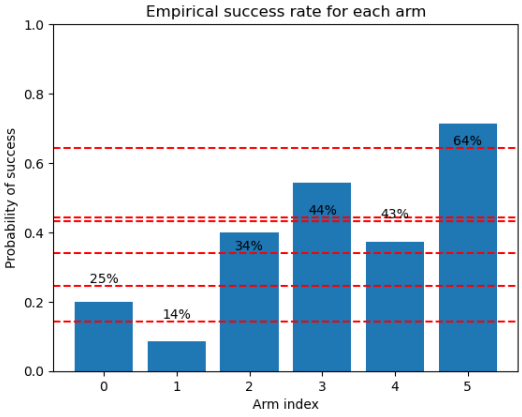
Subfigure (a) illustrates the number of wins per play of each arm. It becomes evident that certain arms exhibit higher success rates compared to others.

Subfigure (b) displays the estimated probability of success at the end of the exploration phase. It can be observed that by exploring for a longer duration, the gap between the empirical values (depicted by vertical blue bars) and the theoretical/true values (represented by horizontal red lines) diminishes.

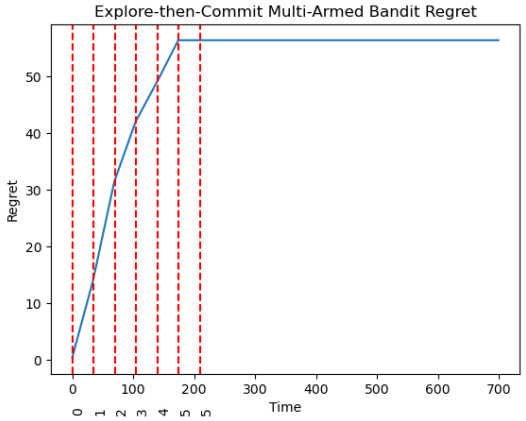
Subfigure (c) presents the regret plot. It contains red vertical lines denoting the exploration of alternative arms. Each arm manifests a distinct slope. After the exploration phase, the slope becomes equal to zero.



(a) Total number of wins of each arm per play



(b) Empirical success rate of each arm (red lines represent the theoretical/true probability)



(c) Cumulative regret

Figure 3: Results of Explore-Then-Commit for multi-armed bandits

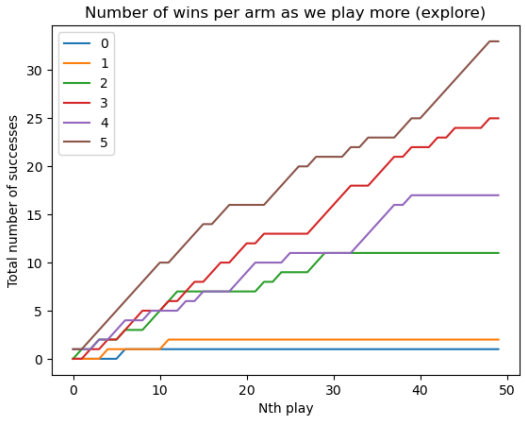
6.1.2 Explore-Then-Commit with elimination rounds

The Explore-Then-Commit with elimination rounds algorithm was executed for a duration of 700 time steps, with 5 elimination rounds (theoretically leaving the optimal arm at the end) and 10 pulls per arm every round.

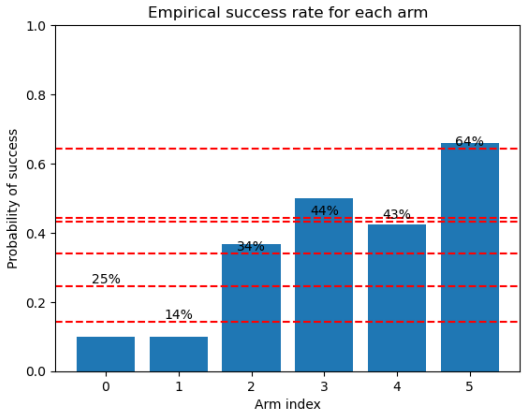
Subfigure (a) illustrates the number of wins per play of each arm. After each elimination round, the arm with lowest number of successes is correctly eliminated (e.g.: the total number of successes for arm 4 does not increase after round 40, after its elimination).

Subfigure (b) displays the estimated probability of success at the end of the exploration phase. It can be observed that by exploring for a longer duration, the gap between the empirical values (depicted by vertical blue bars) and the theoretical/true values (represented by horizontal red lines) diminishes. The empirical successes are more accurate for arms that have been played more by the algorithm (arms that have been eliminated later during the exploration phase).

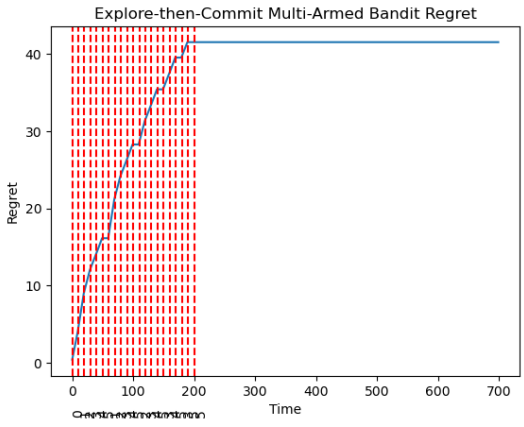
Subfigure (c) presents the regret plot. It contains red vertical lines denoting the exploration of alternative arms. Each arm manifests a distinct slope. After the exploration phase, the slope becomes equal to zero. Compared to the basic Explore-Then-Commit version, this one performs better.



(a) Total number of wins of each arm per play



(b) Empirical success rate of each arm (red lines represent the theoretical/true probability)



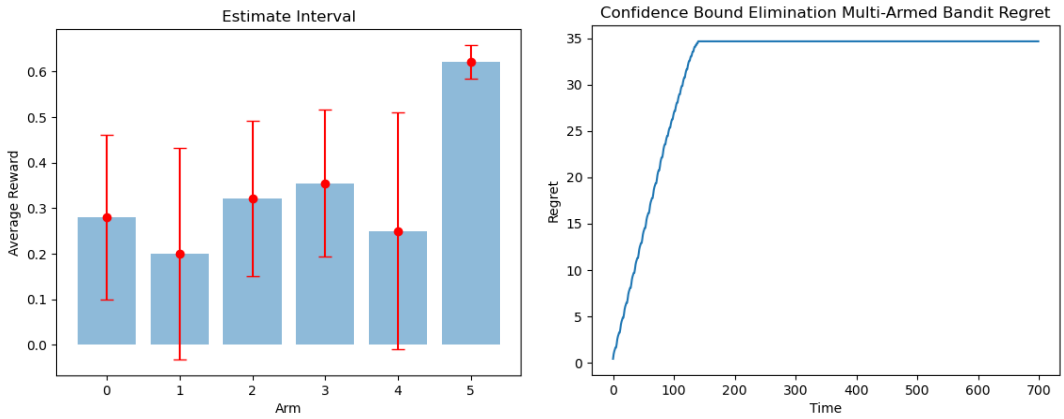
(c) Cumulative regret

Figure 4: Results of Explore-Then-Commit with elimination rounds for multi-armed bandits

6.1.3 UCB variation with elimination rounds

The UCB variation with elimination rounds algorithm was executed for a duration of 700 time steps, with the c constant initialized to 0.9.

Due to the sub-optimality gap being equal to 0.2, the arms are eliminated quickly:
- arm 4 is eliminated at round 12, at time step 72
- arm 1 is eliminated at round 15, at time step 87
- arm 0 is eliminated at round 25, at time step 127
- arm 2 is eliminated at round 28, at time step 136
- arm 3 is eliminated at round 31, at time step 142



(a) Confidence bounds at the end

(b) Cumulative regret

Figure 5: Results of UCB variation with elimination rounds for multi-armed bandits

Subfigure (a) displays the confidence bounds at the end, it can be observed that all sub-optimal arms have their upper bound estimates lower than the lower bound of the optimal one.

Subfigure (b) presents the regret plot. Compared to the previous 2 methods, this one performs better (reaches a slope of zero faster, and the maximum cumulative regret is lower).

6.1.4 Conclusion for ETC, ETC variation and UCB variation

It is imperative to note that the the selection of the optimal arm is predominantly influenced by the sub-optimality gap. A small sub-optimality gap often does not lead to the optimal arm being selected.

6.1.5 Epsilon-Greedy

In this section, the results and conclusions derived from the Epsilon-Greedy algorithm will be subjected to further analysis. The algorithm was executed for 500 time steps, utilizing various values of ε, with 6 arms possessing different sub-optimality gaps. The outcomes were averaged over 100 simulations, while maintaining the same probability distributions. To facilitate clarity and coherence, the reward for all arms was set to 1, ensuring smoothness in the generated plots.

In the first experiment (Figure 6, a), a sub-optimality gap of 0.02 was used between the optimal arm and the second-best arm, as shown in Figure 2, a. The probabilities, listed in descending order, are as follows: 0.86, 0.84, 0.66, 0.65, 0.47, and 0.13. Notably, the Epsilon-Greedy algorithm encountered significant difficulties, as evidenced by the regret being linear for all epsilon values. A noticeable trend emerges: as the epsilon value increases, the slope also increases. When epsilon equals 0.5, it signifies that, on average, the algorithm randomly selects an arm with a 50% probability. Consequently, the algorithm underutilizes the gathered information related to the most rewarding arm, resulting in the observed slope.

Another noteworthy observation is that as the epsilon value decreases, the algorithm increasingly relies on the available information, resulting in less random arm selection. In the given scenario, where four out of the six arms have probabilities exceeding 50%, with two arms having probabilities of 86% and 84%, it is natural for the algorithm to predominantly select either of these two arms based on the

history of received rewards. Consequently, the regret exhibits a linear growth pattern, alternating between the two best arms with the highest probabilities. The only factor influencing the slope of the regret curve is the epsilon value, as it introduces a level of randomness by reducing the frequency at which the algorithm chooses the two best arms when epsilon is larger.

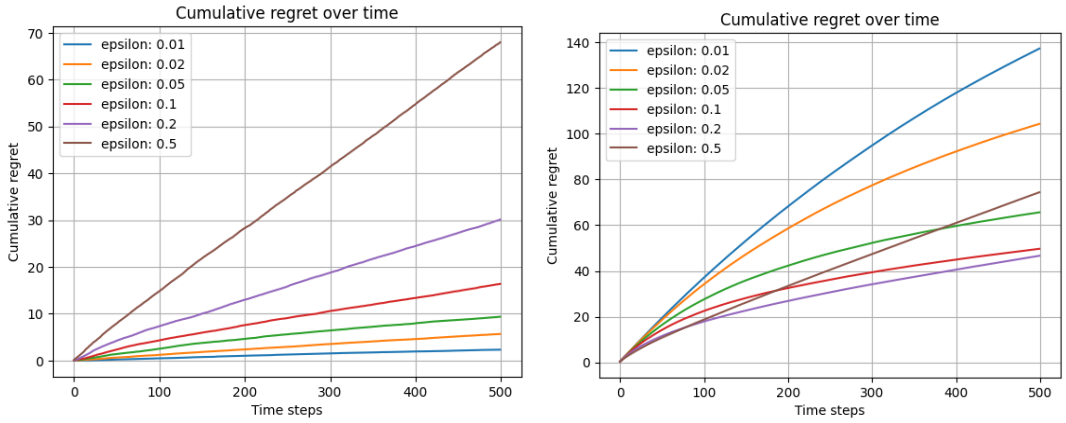
In the second experiment (Figure 6, b), a larger sub-optimality gap of 0.2 was used between the optimal arm and the second-best arm, as shown in Figure 2, b. The probabilities, in decreasing order, are as follows: 0.64, 0.44, 0.43, 0.34, 0.24, 0.14.

Interestingly, this plot reveals an optimal value for epsilon that strikes a balance between exploration and exploitation. The best cumulative regret is observed when epsilon is set to 0.2. This finding contradicts the trend observed in the previous plot, where lower values of epsilon performed better. However, the explanation is straightforward: among the six arms, only the best arm has a probability exceeding 50%.

Consider the scenario where the algorithm initially chooses any of the remaining five arms and obtains a reward. Subsequently, the algorithm will continue selecting these other arms until the best arm is eventually chosen and produces a sufficient number of successful outcomes. When epsilon is set to a very low value, such as 1%, there is only a 1% chance of randomly selecting an arm, and then a 1 in 6 chance of selecting the best arm. The odds of choosing the best arm become significantly low, resulting in the Epsilon-Greedy algorithm struggling to perform well and explore alternative options adequately.

Conversely, when epsilon is set too high, such as 0.5, the algorithm fails to effectively utilize the accumulated information from previous arm pulls. Therefore, there exists a "sweet spot" for epsilon that minimizes the cumulative regret over time, striking a balance between exploration and exploitation.

In conclusion, ε-greedy is an interesting algorithm to study but in the context of multi-armed bandits, it doesn't perform very well, especially when the sub-optimality gap between the 2 best arms is very low. The optimal value of epsilon depends on the probability distribution makes ε-greedy not a reliable algorithm in the context of multi-armed bandits.



(a) ε-greedy for different values of epsilon, when gap between best arm and second best arm is 0.02

(b) ε-greedy for different values of epsilon, when gap between best arm and second best arm is 0.2

Figure 6: ε-greedy with 500 runs and 6 arms, plotting the cumulative regret for different values of ε

6.1.6 Upper Confidence Bound

In this section, the results and conclusions derived from the Upper Confidence Bound algorithm will be subjected to further analysis. The algorithm was executed for 500 time steps, utilizing various values of c, with 6 arms possessing different sub-optimality gaps. The outcomes were averaged over 100 simulations, while maintaining the same probability distributions. To facilitate clarity and coherence, the reward for all arms was set to 1, ensuring smoothness in the generated plots.

In the first experiment (Figure 6, a), a sub-optimality gap of 0.02 was used between the optimal arm and the second-best arm, as shown in Figure 2, a. The probabilities, listed in descending order, are as follows: 0.86, 0.84, 0.66, 0.65, 0.47, and 0.13. Notably, the UCB performs well, even when the

sub-optimality gap is very small.

It is observed that as the value of the constant c decreases, the line representing the cumulative regret also decreases until it reaches the optimal value of $c = 0.8$. Beyond this optimal point, the regret line starts to increase again.

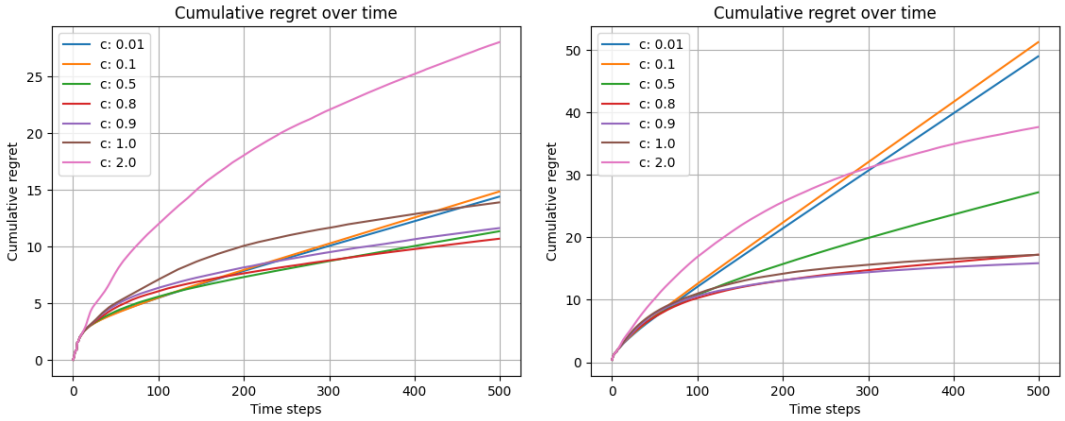
The constant c plays a critical role in controlling the exploration-exploitation trade-off of the algorithm. It determines the width of the confidence interval for each arm's expected value. As c decreases, the confidence interval becomes narrower, leading to a greater emphasis on exploiting the currently best-performing arm. This results in a lower cumulative regret as the algorithm focuses more on maximizing rewards from the known high-reward arms.

$$\text{upper bound of arm i} = \text{estimated reward of arm\_i} + c/\sqrt{\text{total plays of arm i}}.$$

Therefore, the larger the constant c is, the more time UCB will be spending on exploring and finding the optimal arm. This explains why the line with $c = 2.0$ is the highest.

In the second experiment (Figure 7, b), a larger sub-optimality gap of 0.2 was used between the optimal arm and the second-best arm, as shown in Figure 2, b. The probabilities, in decreasing order, are as follows: 0.64, 0.44, 0.43, 0.34, 0.24, 0.14. We observe that UCB also performs well when the sub-optimality gap is larger. There is an optimal value for c, which is neither too big nor too little. The optimal value for c is 0.9 but 0.8 is very close to it. In addition to the explanations of (Figure 2, a), it is interesting to observe that for $c = 0.01$ and $c = 0.1$, the lines are linear. Because the value of c is too little: the algorithm will explore less and potentially cause the algorithm to overlook arms with higher rewards.

In conclusion, UCB is an algorithm that performs very well in the context of multi-armed bandits, when the sub-optimality gap between the 2 best arms is very low as well as when it is larger. Depending on the dataset, there is only the c parameter to tune. This ability to perform well independently of the probability distributions makes the Upper Confidence Bound a robust and reliable algorithm to solve the multi-armed bandits problem.



(a) UCB for different values of c, when gap between best arm and second best arm is 0.02

(b) UCB for different values of c, when gap between best arm and second best arm is 0.2

Figure 7: UCB with 500 runs and 6 arms, plotting the cumulative regret for different values of c

6.2 Linear bandits (LinUCB) on simulated data

In this section, the results and conclusions drawn from the Linear Upper Confidence Bound (UCB) algorithm are further analyzed based on the obtained plots. The algorithm was run for 700 time steps, and the results were averaged over 50 simulations to ensure robustness and reliability.

It is important to note that simulated data is used, allowing for control over the feature vectors of the arms and the true theta vector for the user (new feature vectors for the arms and a new true theta vector is generated every simulation). This random generation process ensures variability and realism in the experiments. Additionally, noise is added to the reward received from the user, introducing an element of randomness.

Four experiments were conducted to investigate the impact of the number of arms and the number of features on the overall performance of the Linear UCB algorithm. These experiments aim to provide insights into the algorithm's behavior and effectiveness under different settings and configurations, allowing for a comprehensive understanding of its capabilities and limitations.

6.2.1 Metrics used to analyse performance of Linear UCB

In order to recommend efficiently, the algorithm must find the best estimate of the true theta vector. This means, that the algorithm must minimize the dissimilarity between the true theta of the user and the theta hat vector which is estimated by the algorithm. Where the theta is the true theta and theta hat is the estimate:

Absolute Difference: The absolute difference is a metric that measures the total difference between corresponding elements of two vectors. In the context of vector comparison, the absolute difference between two vectors is calculated by taking the absolute value of the difference between each element of the first vector (θ_i) and the corresponding element of the second vector ($\hat{\theta}_i$), and summing up these absolute differences:

$$\text{absolute difference} = \sum_{i=1}^{n} |\theta_i - \hat{\theta}_i|$$

Mean Squared Error: The mean squared error (MSE) is a measure of the average squared difference between corresponding elements of two vectors. In the context of vector comparison, the MSE between two vectors is computed by taking the square of the difference between each element of the first vector (θ_i) and the corresponding element of the second vector ($\hat{\theta}_i$), summing up these squared differences, and dividing the result by the number of elements (n):

$$\text{mean squared error} = \frac{1}{n} \sum_{i=1}^{n} (\theta_i - \hat{\theta}_i)^2$$

Euclidean Distance: The Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. When comparing two vectors, the Euclidean distance between them is calculated by taking the square root of the sum of the squared differences between corresponding elements. In other words, for each element (i), the difference between the corresponding elements of the two vectors (θ_i and $\hat{\theta}_i$) is squared, and these squared differences are summed. Finally, the square root of the sum is taken to obtain the Euclidean distance:

$$\text{euclidean distance} = \sqrt{\sum_{i=1}^{n} (\theta_i - \hat{\theta}_i)^2}$$

All 3 metrics: absolute difference, mean squared error and the Euclidean distance provide measures of dissimilarity between two vectors. The smaller these values, the more similar the vectors are considered to be. However, in the experiments, the absolute difference makes more sense to be considered given the fact that the feature vectors have values between -1 and 1, which may lead to thinking that an algorithm is performing well because the mean squared error is very close to zero. By using the absolute difference metric (considering the total difference between the estimated and true values for each element), valuable insights are gained into the algorithm's ability to efficiently recommend and minimize dissimilarity between vectors. The lower the absolute difference, the closer the estimated vector is to the true vector, the more accurate will the predictions be and the better will the algorithm performance be.

6.2.2 Varying the number of arms

In the first 2 experiments, (Figure 8, 9) the algorithm were ran for 700 time steps, with 10 features and a noise value of 0.1. The lambda parameter was fixed to 0.5 and the everything was averaged over 100 simulations.

In the first experiment (Figure 8), the cumulative regret curves were analyzed for four variations in the number of arms: 10, 20, 50, and 100 arms, with a fixed beta value of 1.0. It was observed that as the number of arms increased, the cumulative regret curve flattened out at a higher level. This behavior can be attributed to the algorithm spending more time exploring to identify the best arm among the increased number of options. Also, as the number of arms increased, the gap between the flat parts of the cumulative regret curves reduced.

To assess the accuracy of the algorithm's estimation, the absolute difference between the true theta vector and the estimated theta vector was plotted over time (Figure 8, b). Interestingly, it was observed that as the number of arms increased, the mean squared errors decreased. This trend can be attributed to the presence of more arms, increasing the likelihood of having an optimal arm closer to the true theta vector. For example, in a buffet scenario, a larger variety of food options increases the chances of finding something more preferable. Moreover, when the beta parameter was set to a constant value, the algorithm consistently selected the optimal arm, indicating convergence of the estimate vector to the one associated with the optimal arm.

In the subsequent analysis, the impact of logarithmically increasing beta values was examined (Figure 9). Similar to the fixed beta case, the cumulative regret curves demonstrated that as the number of arms increased, the algorithm spent more time exploring to identify the best arm. The trend of increasing regrets with a larger number of arms persisted. Comparatively, when beta was set as a logarithmic function, the algorithm achieved a better balance between exploration and exploitation over an extended period. This adaptive behavior enabled the algorithm to transition gradually from exploration to exploitation as more data was collected, leading to improved long-term performance. The cumulative regret curves (Figure 9, a) exhibited less flatness compared to the fixed beta scenario, indicating a higher degree of exploration.

Examining the MSE plot (Figure 9, b), it was observed that the MSE values were significantly closer to zero compared to the fixed beta case. This outcome can be attributed to the algorithm spending more time exploring and alternating between different optimal arms, resulting in continuous adjustments of the estimated theta vector.

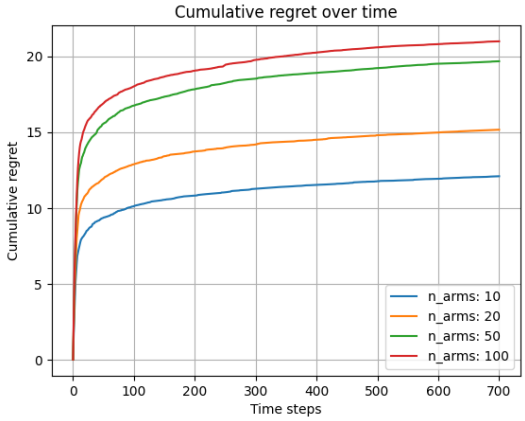
6.2.3 Varying the number of features

In the last 2 experiments, (Figure 10, 11) the algorithm were ran for 700 time steps, with 10 arms and a noise value of 0.1. The lambda parameter was fixed to 0.5 and the everything was averaged over 100 simulations.

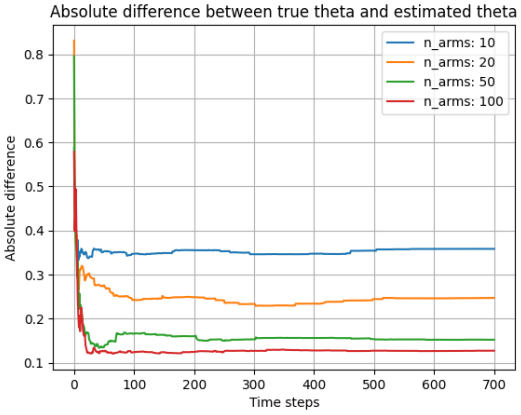
In the first experiment (Figure 10), the cumulative regret curves were analyzed for four variations in the features: 2, 5, 10, and 50 features, with a fixed beta value of 1.0.
It was observed that as the number of features increased, the cumulative regret curve flattened out at a higher level. As the number of features increases, the dimensionality of the feature space expands, resulting in a larger number of possible linear combinations. This increase in potential combinations necessitates a more extended exploration phase for the algorithm to identify the optimal arm accurately. Consequently, the algorithm may take longer to converge to the best arm and, as a result, experience larger regrets.

By looking at the plot of the absolute difference between the true theta vector and the estimated theta vector (Figure 10, b), it was observed that theta is better estimated when the dimensions are low. This trend can be attributed to the likelihood of having an optimal arm closer to the true theta vector. With fewer dimensions, there are fewer elements to match between the 2 vectors.

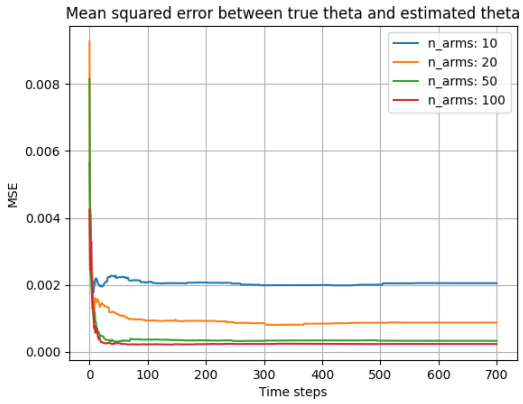
In the subsequent analysis, the impact of logarithmically increasing beta values was examined (Figure 11). Similar to the fixed beta case, the cumulative regret curves demonstrated that as the number of arms increased, the algorithm spent more time exploring to identify the best arm. The trend of increasing regrets with a larger number of features persisted. Comparatively, when the value of beta was set as a logarithmic function, the cumulative regret curves (Figure 11, a) exhibited less flatness compared to the fixed beta scenario, indicating a higher degree of exploration. However, with respect to the similarity between the true theta and the estimated theta, increasing the beta values logarithmically only improve the results for 2, 5 and 10 features compared to when the value of beta is fixed (when looking at the absolute difference and Euclidian distance plots).



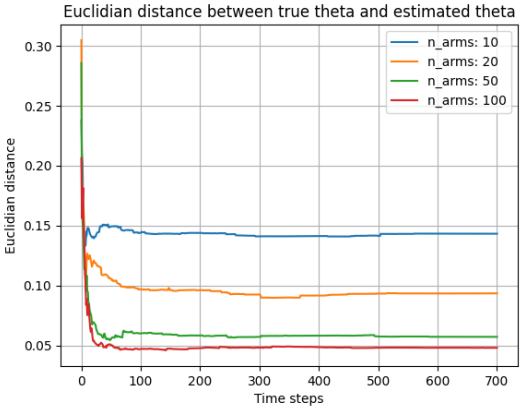
(a) Cumulative regret plot for different number of arms



(b) Absolute difference

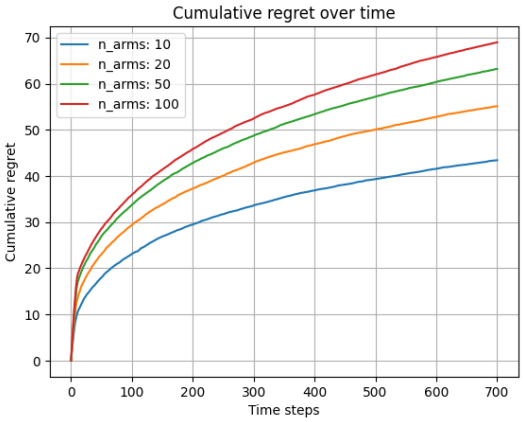


(c) MSE

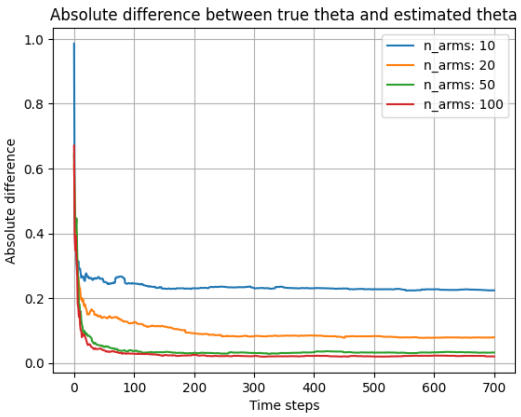


(d) Euclidian distance

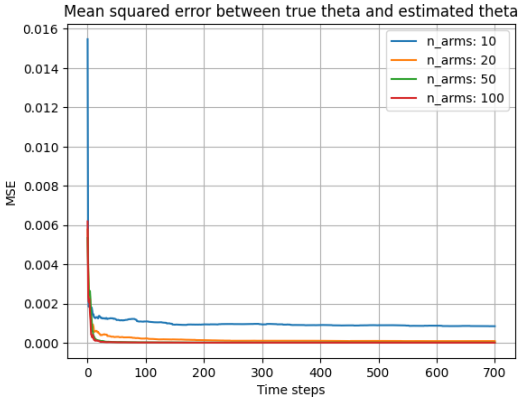
Figure 8: Results found by running Linear UCB for different number of arms when beta is fixed : 1.0



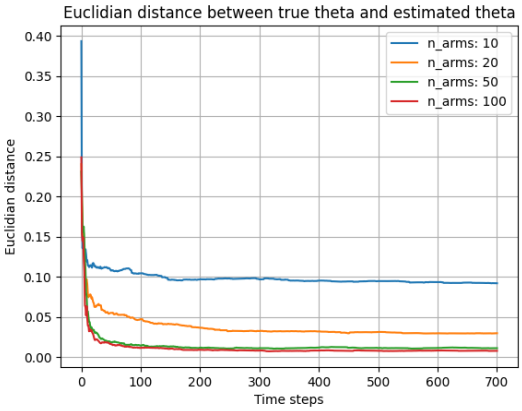
(a) Cumulative regret plot for different number of arms



(b) Absolute difference

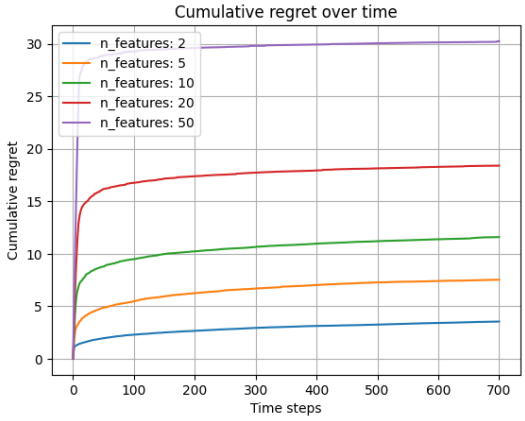


(c) MSE

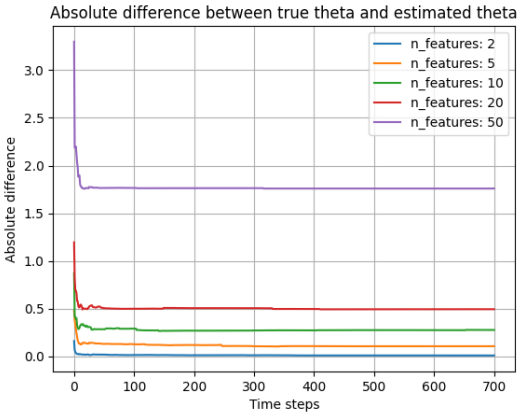


(d) Euclidian distance)

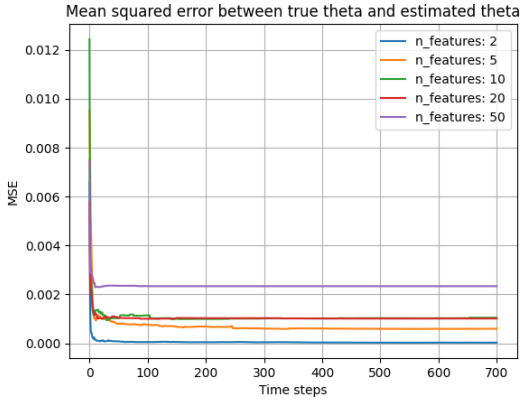
Figure 9: Results found by running Linear UCB for different number of arms when beta increases logarithmically



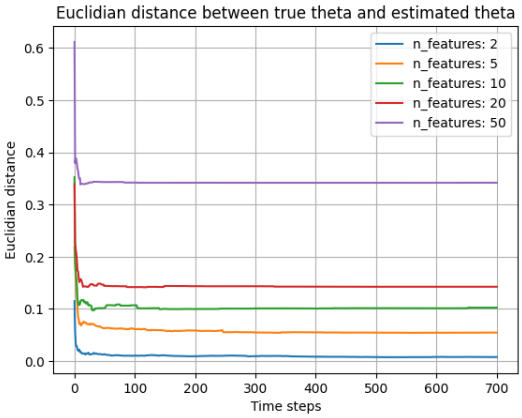
(a) Cumulative regret plot for different number of features



(b) Absolute difference

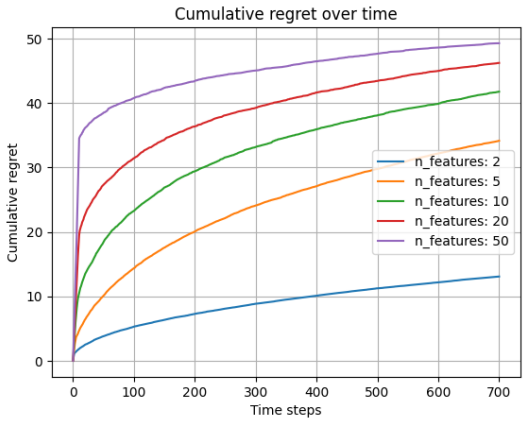


(c) MSE

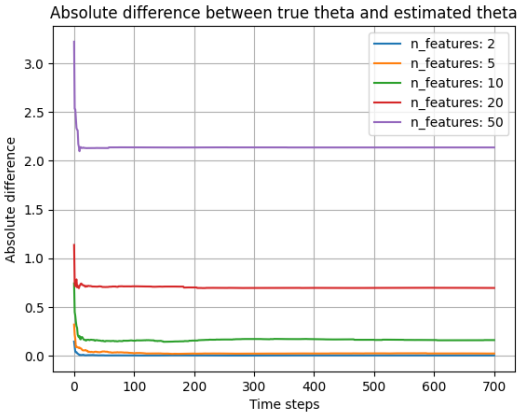


(d) Euclidian distance

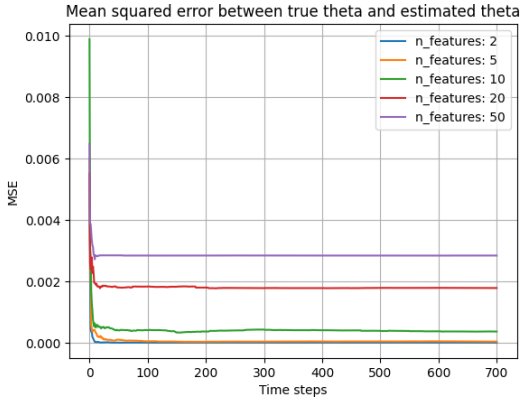
Figure 10: Results found by running Linear UCB for different number of features when beta is fixed : 1.0



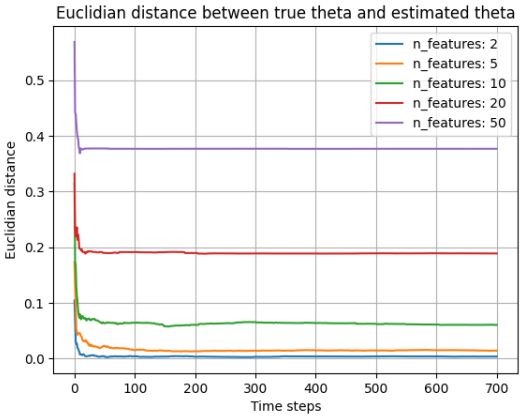
(a) Cumulative regret plot for different number of features



(b) Absolute difference



(c) MSE



(d) Euclidian distance

Figure 11: Results found by running Linear UCB for different number of features when beta increases logarithmically

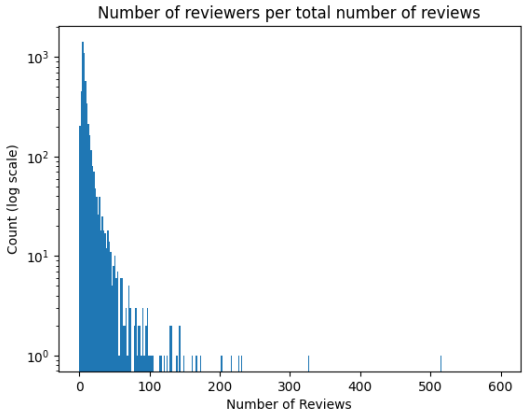
6.3 Linear bandits (LinUCB) on real data

6.3.1 Preprocessing the data

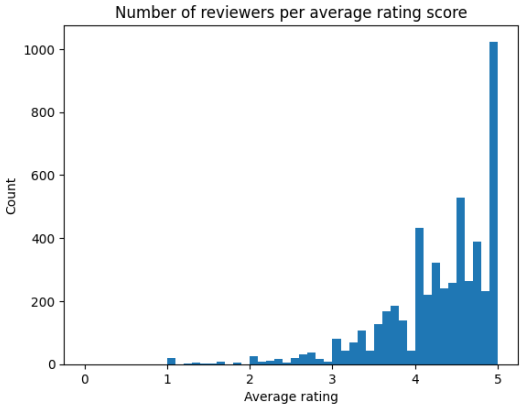
In order to ensure the effectiveness of linear ucb, a selection process was carried out to extract users and their corresponding reviews from the dataset. Specifically, reviewers with a minimum of 50 reviews and a variance of at least 1.0 in their ratings were chosen to evaluate the linear bandits algorithm. This decision was motivated by two key factors. Firstly, the inclusion of users with a substantial number of reviews offers more data for the algorithm to learn from and also to test the effectiveness of the algorithm. Secondly, the requirement of a minimum variance aims to capture the diversity in the ratings of each user, leading to a more realistic scenario in the context of recommendation systems where users are not always rating in a constant pattern. These criteria resulted in a significant reduction in the number of reviewers, with only 16 individuals remaining out of the initial pool of 5,148.

Figure 8 provides valuable insights into the characteristics of the 'Digital_music_5.json' dataset. Subfigure (a) illustrates the distribution of reviewers based on the total number of reviews they each have, showing that the majority of reviewers have less than 50 reviews. Subfigure (b) depicts the distribution of reviewers in relation to their average rating score, indicating that most of them have an average rating between 4 and 5, suggesting a preference to give a rating to music they genuinely enjoy. Subfigure (c) represents the distribution of reviewers with respect to the standard deviation of their ratings, demonstrating that a significant proportion of reviewers exhibit a standard deviation of 0. This indicates that those reviewers give ratings in a consistent pattern. Combining all 3 subfigures, there is a tendency for most users to give very few reviews and to usually rate a few 4 or 5 stars ratings.

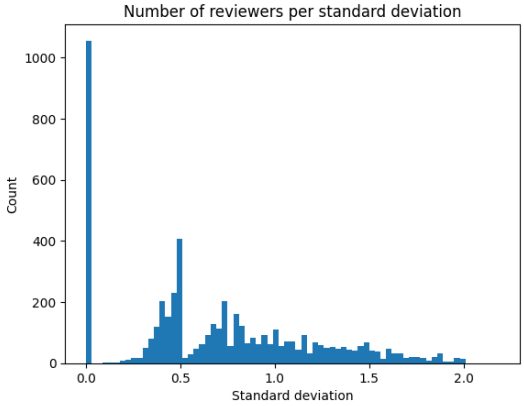
Moving on to Figure 9, which focuses specifically on the 16 users who met the criteria of at least 50 ratings and a variance of 1.0, subfigure (a) reveals that all 16 users have met the minimum threshold of 50 reviews each. Subfigure (b) illustrates the distribution of reviewers in terms of their average rating score, while subfigure (c) examines the distribution of reviewers with respect to the standard deviation of their ratings. It is noteworthy that all 16 users exhibit a standard deviation of at least 1.0, indicating a greater diversity in their rating patterns compared to the rest of the users in the dataset.



(a) Distribution of reviewers with respect to their total number of reviews

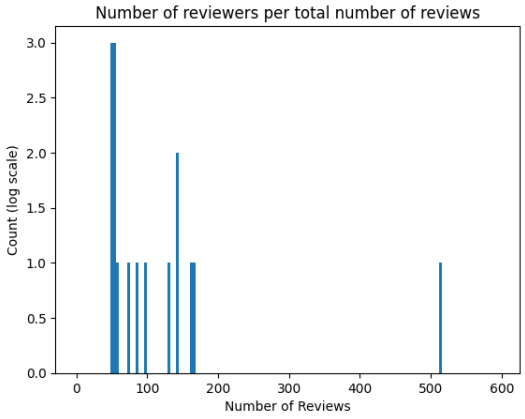


(b) Distribution of reviewers with respect to their average rating score

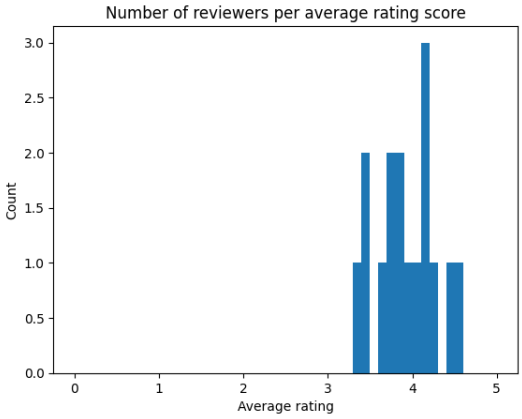


(c) Distribution of reviewers with respect to their standard deviation

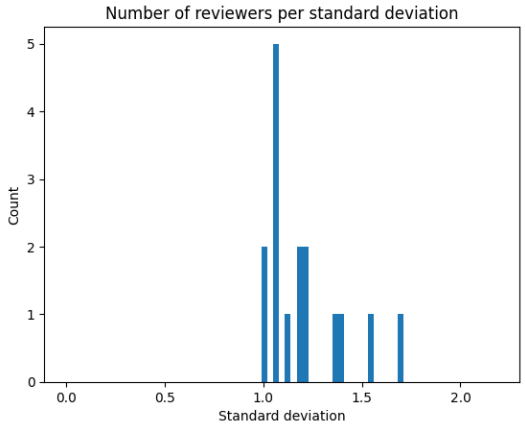
Figure 12: Visualization of dataset 'Digital_music_5.json'



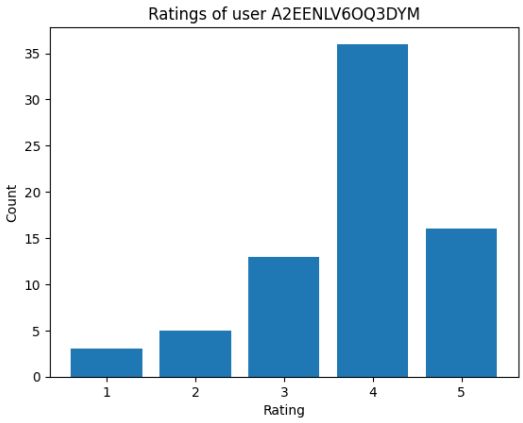
(a) Distribution of reviewers with respect to their total number of reviews



(b) Distribution of reviewers with respect to their average rating score



(c) Distribution of reviewers with respect to their standard deviation



(d) Distribution of ratings for a user

Figure 13: Visualization of filtered reviewers with minimum of 50 reviews and variance 1.0

6.3.2 Metric used to analyse performance of Linear UCB

In order to ensure efficient recommendation, the algorithm aims to minimize the dissimilarity between the estimated theta hat vector and the true theta vector of the user. However, in practical scenarios where real data is involved, the true theta vector is unknown. Instead, only the ratings provided by users for the purchased music items are available. To address this, a data allocation approach was adopted, where 90% of the available data was utilized for training purposes, while the remaining 10% was reserved for testing. This division allowed the linear UCB algorithm to be trained for each user using their respective ratings from the training set.

To evaluate the effectiveness of the algorithm, two metrics could be employed. The first metric is the cumulative regret, which provides insights into the performance of the training phase. However, it does not serve as a reliable indicator of the algorithm's performance on unseen data. Therefore, an additional metric called the "absolute difference" was introduced to assess the quality of recommendations generated by the algorithm on the test data.

Absolute Difference:

$$\text{absolute difference} = \frac{1}{n} \sum_{a \in A_t} \left| \langle \hat{\theta}, a \rangle - r \right|$$

- A_t represents the set of arms in the test set
- n is the number of arms in the test set
- $\hat{\theta}$ is the estimated theta vector
- r is the true rating for that arm.

6.3.3 Discussion of results

In the conducted experiments, Linear UCB algorithm was evaluated across multiple values of beta, with the intention of comparing the effectiveness of ISOMAP and PCA techniques. The experiments involved running the algorithm for 500 rounds and conducting 6 simulations for each user. The maximum reward was set to 5, noise was kept at 0, and the regularization parameter lambda was set to 0.1.

To obtain reliable results, the experiments were repeated 10 times and the averages were computed. Weighted averages for absolute differences were calculated, meaning that users with more reviews have more impact on the overall absolute difference. The main goal was to assess the performance of Linear UCB with different beta values and determine the optimal number of dimensions for feature vectors using ISOMAP and PCA.

In Figure 14, the outcomes of ISOMAP are depicted for various feature vector dimensions. Subfigure (a) indicates that for every beta value, the smallest absolute difference occurs when the dimension is equal to 5. This suggests that, for the given dataset and when utilizing ISOMAP, the most accurate results are achieved with a beta value of 10.0 and feature vectors of dimension 5. In Subfigure (b), it is observed that the optimal number of dimensions is 3 when utilizing ISOMAP for dimensionality reduction in conjunction with the Linear UCB algorithm and increasing beta.

Figure 15 presents the results obtained from running PCA for different feature vector dimensions. Subfigure (a) shows that the dimension yielding the lowest absolute difference when beta is fixed is also dimension 5. In Subfigure (b), it can be observed that as beta increases logarithmically, the best number of dimensions for PCA is 2.

Unfortunately, the conducted experiments did not yield to any insights or reveal discernible trends. Consequently, no conclusive findings can be drawn from the obtained results.

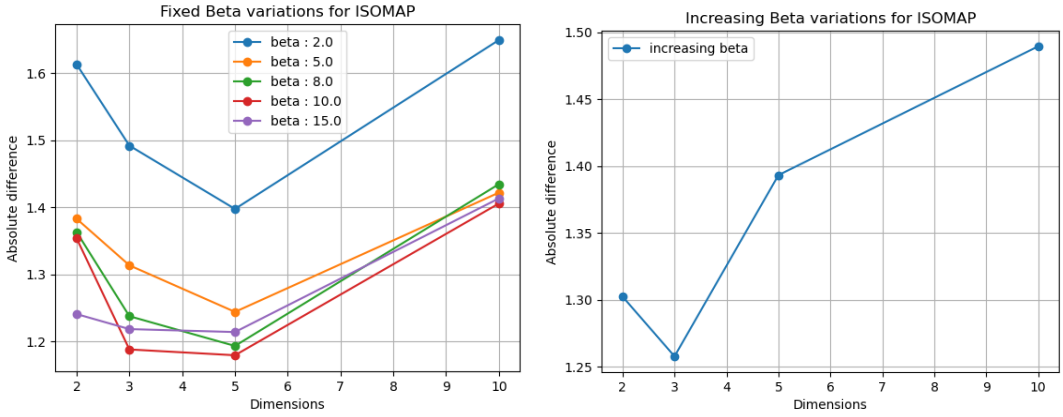
6.3.4 Linear Regression vs Linear UCB

In order to establish a baseline for comparison, the performance of the Linear UCB algorithm was compared against linear regression (Figure 16). The evaluation was conducted to assess the effectiveness and accuracy of predictions made by linear regression.

The results indicate that, linear regression outperforms linear bandits in terms of prediction accuracy for this dataset. Linear regression demonstrates a higher level of accuracy in predicting users' ratings compared to the Linear UCB algorithm.

The pursuit of a true theta vector for a user may not be realistic, as real-world scenarios often involve non-linear relationships between features and rewards.

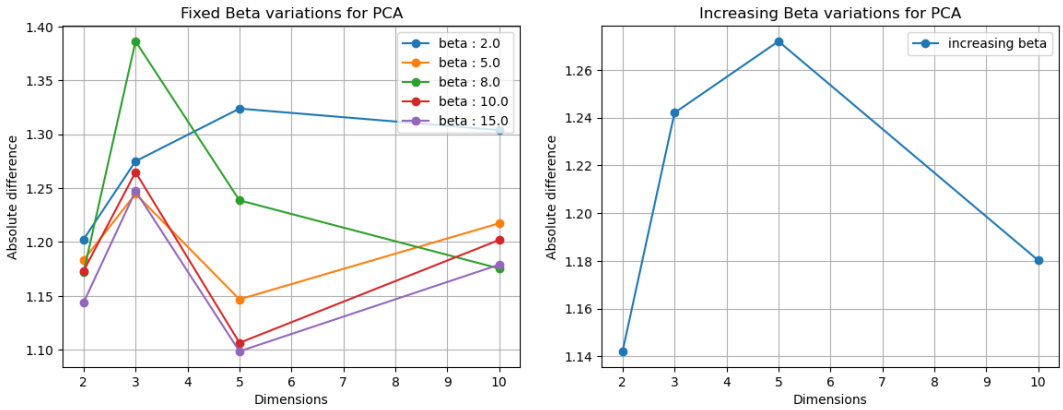
Furthermore, it is important to acknowledge that individual preferences for music can be influenced by multiple factors beyond genre, such as lyrical content or specific instruments. In our experiments, we solely focused on different music styles, potentially overlooking valuable information that could have enhanced the accuracy of the recommendations.



(a) Absolute differences for different values of fixed beta and feature vector dimensions

(b) Absolute differences when beta increases logarithmically

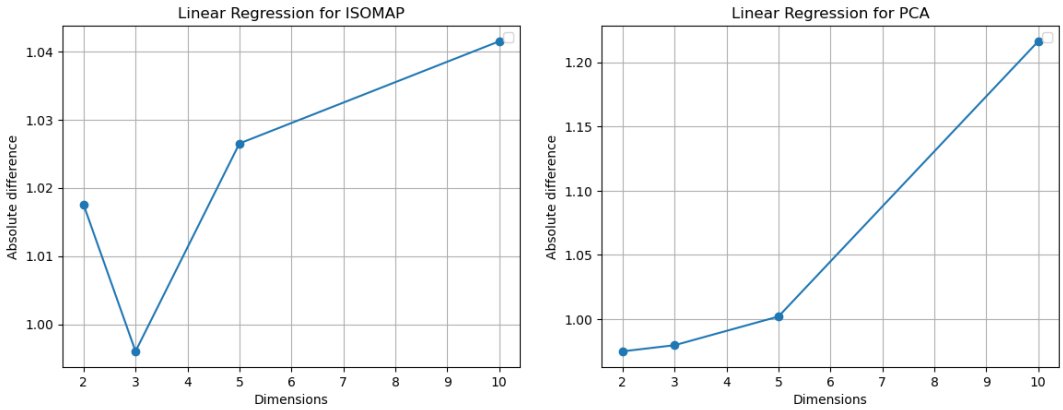
Figure 14: Sum of differences between true ratings and estimated ratings by LinUCB in the test set when using ISOMAP to reduce the dimension of feature vectors, averaged over 10 simulations for 16 users



(a) Absolute differences for different values of fixed beta and feature vector dimensions

(b) Absolute differences for different values of fixed beta and feature vector dimensions

Figure 15: Sum of differences between true ratings and estimated ratings by LinUCB in the test set when using PCA to reduce the dimension of feature vectors, averaged over 10 simulations for 16 users



(a) Absolute differences for different feature vector dimensions with ISOMAP

(b) Absolute differences for different feature vector dimensions with PCA

Figure 16: Sum of differences between true ratings and estimated ratings by Linear Regression in the test set

7 Discussion and Conclusion

The goal of this bachelor's project was to comprehensively examine and analyze various types of bandit algorithms, particularly focusing on their implementation in recommendation systems.

The project initially delved into different types of multi-armed bandits algorithms such as Explore-Then-Commit, Epsilon-Greedy, and Upper Confidence Bound (UCB). The key takeways from this part of the project is that UCB performed well for any sub-optimality gap (both small and large). This ability perform well independently of the probability distributions makes the Upper Confidence Bound a robust and reliable algorithm to solve the multi-armed bandits problem.

A significant portion of the project was dedicated to studying linear bandits. This expanded upon the traditional multi-armed bandit problem, encompassing scenarios where a linear relationship exists between each arm and its corresponding rewards. The implemented Linear UCB algorithm demonstrated notable effectiveness within recommendation systems.

As the number of arms increases, the cumulative regret curve flattens out at a higher level. As the number of arms increased, the mean squared errors decreases. Having logarithmically increasing beta values leads to better the estimated theta being closer to the true theta but larger cumulative regret because the algorithm spends more time exploring, trying to find the optimal arm.

As the number of features increases, the cumulative regret curve flattens out at a higher level. As the number of features increases, the absolute difference between the estimated theta and the true theta increases. Having logarithmically increasing beta values, leads to larger cumulative regret but better estimated thetas.

The project also transitioned from theoretical analysis to practical application by integrating real-world datasets from Amazon Music. This step underlined the inherent complexities when transitioning from simulated to real-world scenarios and demonstrated the flexibility of the Linear UCB algorithm in a real-world setting. However, it was observed that linear regression outperforms linear bandits in terms of prediction accuracy for this dataset. Linear regression demonstrates a higher level of accuracy in predicting users' ratings compared to the Linear UCB algorithm. Therefore, although linear bandits perform very well with simulated data, working with real-world data present many more challenges.

Although linear bandits have shown significant potential in the context of this project (especially with simulated data), it should be noted that in practice, they are not used in large-scale recommender systems like Netflix and Amazon, because other types of algorithms such as Collaborative Filtering perform better. However, compared to this, linear bandits present a compelling advantage in specific contexts. They have been observed to work optimally when there is limited data available, which is a challenging scenario for collaborative filtering, which requires extensive data to deliver precise recommendations. Linear bandits also provide the benefit of continuous learning, as they can be updated with each new data point, allowing them to progressively enhance their performance over time. Furthermore, they enable easier tweaking of parameters to cater to evolving user preferences, a characteristic that can be critical in recommendation systems where user tastes can shift rapidly.

In conclusion, this bachelor's project has shed light on the potential of multi-armed and linear bandit algorithms, in optimizing recommendation systems. By understanding their performance and effectiveness in both theoretical and real-world contexts, the project points to several avenues for future work, such as further exploring their capabilities for continuous learning and adaptation or exploring other more advanced types of bandit algorithms such as collaborative filtering bandits, kernelized bandits, ...

References

[1] Lattimore, Tor, and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[2] Russac, Y. *Introduction to Linear Bandits*.

[3] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. *Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms*.