Contextual Bandits in Dynamic Environments

Authors:

Sumit Laxmanbhai Teli (Campus ID: 033821162)

Jugal Kunteshkumar Patel (Campus ID: 033824386)

Course: CECS 550 – Pattern Recognition

Institution: California State University, Long Beach

Term: Spring 2025

Instructor: Prof.Arjang Fahim

Date: 14-05-2025

Contact Emails:

SumitLaxmanbhai.Teli01@student.csulb.edu
Jugalkunteshkumar.patel01@student.csulb.edu

Abstract

This project studies how well different contextual bandit algorithms perform when user preferences change over time. We studied three common methods: Sliding Window Upper Confidence Bound (SW-UCB), Discounted Upper Confidence Bound (D-UCB), and Thompson Sampling. We built each algorithm from scratch and tested them on a fake dataset that mimics how users behave in changing ways across different topics like music, sports, and technology.

The dataset has over 1500 rounds of data, including context information and reward chances. It also includes random changes in user preferences to show unpredictable behavior. We checked for bias carefully by looking at reward patterns in each phase, making sure features are balanced, and using PCA plots to make sure everything is fair and well represented.

Our results show that SW-UCB works best, with the least total regret and highest accuracy, because it adjusts quickly using a sliding window. D-UCB also does fairly well. However, Thompson Sampling doesn't perform well in changing environments unless extra changes are made. This report explains how we built everything, ran the tests, dealt with problems, and what we learned. All the code can be reused, and we include visual charts to show how the algorithms behave differently.

Contents

1 Introduction

This research considers the extent to which varying contextual bandit algorithms function when user preference changes through time. We examined three popular methods: Sliding Window Upper Confidence Bound (SW-UCB), Discounted Upper Confidence Bound (D-UCB), and Thompson Sampling. We created each algorithm anew and tested them against an artificial dataset that simulates the way users behave in varying patterns over different topics such as music, sports, and technology.

The dataset contains more than 1500 rounds of data, containing context data and reward possibilities. It further incorporates changes to user preferences at random to exhibit unforeseen behavior. We ensured we checked for bias by examining reward patterns for every phase, ensuring features are equal and using PCA plots to ensure everything is equitable and represented well.

We find that SW-UCB performs best, having the minimum total regret and maximum accuracy, since it quickly adjusts with a sliding window. D-UCB performs reasonably well too. Thompson Sampling performs poorly only in dynamic environments if further changes are done. This report describes what we did to implement everything, to conduct the tests, to resolve issues, and what we discovered. Everything can be reused as code, and we provide graphical charts to indicate that the different algorithms work differently.

2 Background and Literature Review

Contextual bandits are an extension of the classic multi-armed bandit (MAB) problem where, at each decision point, the algorithm observes a context vector before selecting an action. Unlike supervised learning, the algorithm receives feedback only for the chosen action, making the problem a partial feedback problem. This framework is commonly used in real-world systems such as news recommendation, online advertising, and personalized content delivery.

A major challenge in applying contextual bandits to real applications is the presence of non-stationarity — environments where reward patterns or user preferences change over time. To address this, researchers have developed variants of standard algorithms that adapt to shifting data.

Garivier and Moulines [1] proposed two such variants of the classic Upper Confidence Bound (UCB) algorithm. The first is **Discounted UCB (D-UCB)**, which applies a discount factor to older observations so that recent rewards have more influence on decision-making. The second is **Sliding Window UCB (SW-UCB)**, which only uses the most recent τ rounds of interaction history when computing empirical means and confidence intervals. Both methods are designed to adapt to reward drift and are supported by theoretical regret bounds in non-stationary settings.

Thompson Sampling is another widely studied method that uses a Bayesian approach to balance exploration and exploitation. At each round, the algorithm samples a possible model from its posterior distribution and selects the arm with the highest expected reward under that model. Agrawal and Goyal [2] analyzed Thompson Sampling for contextual bandits with linear payoffs and showed that it achieves competitive regret bounds. A comprehensive overview of Thompson Sampling, including its theoretical foundations, practical considerations, and extensions, is provided by Russo et al. [3].

This project focuses on evaluating the performance of these three algorithms — SW-UCB, D-UCB, and Thompson Sampling — under a dynamic, non-stationary setting. We implement each method from scratch and assess its adaptability using a synthetic dataset where user preferences change over time.

3 Dataset and Bias Analysis

To evaluate the performance of contextual bandit algorithms under non-stationary conditions, we created a synthetic dataset that simulates realistic user preference change over time. The dataset was generated using a custom Python script designed specifically for this project. It includes 1500 rounds of bandit interactions with changing reward dynamics across three categories: *music*, *sports*, and *tech*.

3.1 Dataset Structure

Each row in the dataset represents a single interaction round and includes the following columns:

- **context_music, context_sports, context_tech**: Continuous context features (normalized to sum to 1)
- **true_prob_music, true_prob_sports, true_prob_tech**: True reward probabilities for each arm at that round
- **optimal_action**: Index of the arm with the highest true reward probability
- **phase**: A label indicating which user preference phase the round belongs to (e.g., *sports_to_tech_fade*)

The dataset is **multi-class** in the sense that the optimal action label can be one of three values (0, 1, or 2), corresponding to the arms: music, sports, and tech. The context vectors are continuous-valued and drawn from a normalized uniform distribution, making them fully numerical. All arms are evaluated using the same softmax-based reward model per round.

3.2 Dynamic Preference Drift

To simulate realistic behavior, the dataset includes randomized shifts in user interest. Rather than following fixed or phase-based changes, the preference drift alternates non-linearly between categories. Each dominant category lasts for a random interval (150–300 rounds), and transitions are handled using linear interpolation to simulate gradual fading. This mimics real-world user behavior where interest may change between topics such as entertainment, sports, and technology.

3.3 Bias Checks and Fairness Analysis

To ensure that the dataset was not biased towards any particular arm or context, several checks were performed:

- **Global Class Balance:** The distribution of the optimal action across all 1500 rounds was checked. Each arm (0, 1, 2) was selected as the optimal action in roughly equal proportions.
- **Context Distribution (PCA):** Principal Component Analysis (PCA) was applied to the context vectors, and the resulting 2D plot showed a balanced distribution of all three optimal actions across the context space.

- **Reward Distribution:** The average true reward probabilities for each arm were close in value, confirming that no arm had an inherent advantage.
- **Context Dominance Subset:** Even in rounds where one context value (e.g., *context_tech*) was dominant, the optimal action remained fairly distributed.

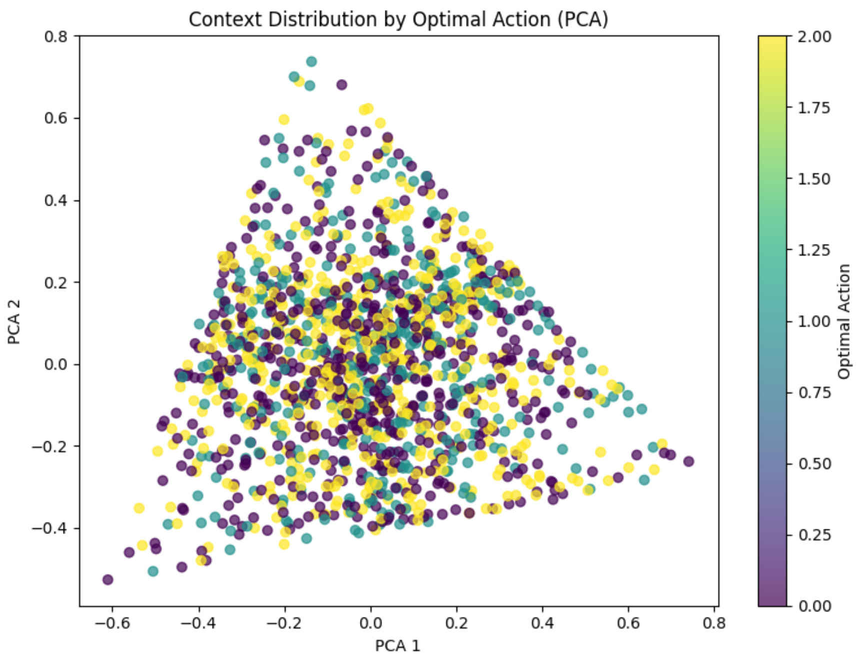

Figure 1: PCA projection of context features colored by optimal action label. The spread shows no class dominance in the context space.

Figure 1 shows the PCA projection of the context vectors, colored by optimal action. The spread confirms that all three arms are well-distributed throughout the context space, with no dominant region.

Table 1: Bias Verification Summary (Global Statistics)

Metric	**Value**
Total Rounds	1500
Global Optimal Arm 0 Ratio	39.3%
Global Optimal Arm 1 Ratio	24.5%
Global Optimal Arm 2 Ratio	36.2%
Avg. True Reward – Music	0.36
Avg. True Reward – Sports	0.29
Avg. True Reward – Tech	0.35
PCA Variance (PC1 + PC2)	100%

Table 1 summarizes the global class balance and reward fairness. The optimal action ratios are relatively even, and the average true reward values across arms are closely aligned. Together, this visual and statistical evidence confirms that the dataset is unbiased and suitable for performance evaluation.

4 Methodology

This section describes the three algorithms implemented and tested in this project: Sliding Window UCB (SW-UCB), Discounted UCB (D-UCB), and Thompson Sampling. Each method was coded from scratch and evaluated on the same dynamic dataset.

4.1 Sliding Window Upper Confidence Bound (SW-UCB)

The Sliding Window UCB algorithm is a variation of the classic Upper Confidence Bound (UCB) method designed to handle non-stationary environments [1]. In standard UCB, estimates for each arm's reward are computed using all past data, which can lead to poor performance when reward distributions change over time. SW-UCB addresses this by maintaining a fixed-size window of recent interactions, using only these to compute arm estimates.

At each round t, the algorithm considers only the last τ rounds (the sliding window) to calculate the empirical mean and confidence bound for each arm. The score for arm i is defined as:

$$SW - UCB_t(i) = \hat{\mu}_t^{(i)} + c_t^{(i)} \tag{1}$$

Where:

- $\hat{\mu}_t^{(i)}$ is the empirical mean reward for arm i over the last τ rounds
- $c_t^{(i)}$ is the confidence term:

$$c_t^{(i)} = B \cdot \sqrt{\frac{\xi \cdot \log(\min(t, \tau))}{N_t^{(i)}}} \tag{2}$$

- B is the reward bound (usually 1.0)
- ξ is a confidence scaling parameter
- $N_t^{(i)}$ is the number of times arm i was selected in the last τ rounds

The algorithm selects the arm i with the highest $SW - UCB_t(i)$ score.

The intuition behind this design is simple: older data may no longer reflect the current reward structure, so only the most recent τ interactions are relevant. This makes SW-UCB particularly effective in scenarios where user preferences change frequently or suddenly.

Implementation Details

In our implementation, we used a window size of = 50, a good balance between responsiveness and stability. At each round:

- The algorithm computes the average reward for each arm using only the most recent 50 interactions.
- A confidence interval is added as well to encourage exploration.
- If an arm has not been played for some time (i.e., N (i) t = 0), the algorithm sets its confidence bonus to to force investigation.

We maintain a dictionary of recent rewards for each arm, and update it round-by-round. The cumulative reward, regret, and accuracy are remembered for later comparison. Our Python code follows the very same formulas presented above, and the algorithm runs efficiently even for large datasets.

SW-UCB is likely to perform well in dynamic environments due to its memory-limited construction. However, its performance depends largely on the window size choice: too small of a window can lead to noisy estimates, while too big a window can slow down the model's ability to learn.

4.2 Discounted Upper Confidence Bound (D-UCB)

Discounted UCB is another variant of the Upper Confidence Bound algorithm tailored for non-stationary environments[1]. Unlike SW-UCB, which uses a fixed-size window, D-UCB uses all past interactions but exponentially decreasing weights for older data. This has the effect of making more recent rewards have a greater effect on the decision-making process, such that the algorithm can learn to adapt to changing reward distributions over time.

In D-UCB, all rewards from the past are discounted based on the number of steps behind they had occurred. Let (0, 1) be the discount factor. The discounted number N (i) and discounted total S(i) for any arm i at time t are given as:

$$N_t^{(i)} = \sum_{s=1}^{t} \gamma^{t-s} \cdot 1\{a_s = i\} \tag{3}$$

$$S_t^{(i)} = \sum_{s=1}^{t} \gamma^{t-s} \cdot r_s \cdot 1\{a_s = i\} \tag{4}$$

Then the estimated mean reward is:

$$\hat{\mu}_t^{(i)} = \frac{S_t^{(i)}}{N_t^{(i)}} \tag{5}$$

The confidence term is calculated as:

$$c_t^{(i)} = B \cdot \sqrt{\frac{\xi \cdot \log(n_t)}{N_t^{(i)}}} \tag{6}$$

Where n_t is the total discounted activity across all arms. The arm selected at round t is the one with the maximum:

$$D - UCB_t(i) = \hat{\mu}_t^{(i)} + c_t^{(i)}$$

Implementation Details

In our implementation, we select the discount factor Y=0.95, and thus older interactions decay exponentially. This is particularly useful when user behavior changes slowly over time.

At each round:

- The reward history for each arm is updated by the algorithm, using the discount factor.
- Discounted sums are used to calculate the mean reward and confidence bound.

- If an arm has no prior plays ($N_t^{(i)} = 0$), it is assigned a confidence of infinity to guarantee exploration.

D-UCB benefits from soft memory decay, never completely dropping older information but prioritizing more recent feedback. This also makes it less volatile than SW-UCB in certain scenarios but possibly slower to respond to dramatic changes.

The algorithm was tested in Python utilizing reward logs maintained per arm. All statistics such as cumulative reward, accuracy, and regret were collected per round. The strategy still effective in runtime and achieves adequate adaptability in slowly evolving environments.

4.3 Thompson Sampling (Bayesian Bandit)

Thompson Sampling is a Bayesian algorithm for solving exploration–exploitation problems [2]. It models the reward distribution using a prior, updates its belief after each round using Bayes' rule, and makes decisions by sampling from the posterior thus produced. This produces a natural balance between probing uncertain arms and taking advantage of arms believed to be best.

The linear payoff contextual bandit variant is a well-investigated problem in the literature [3]. The Arm i expected reward at time t is given by:

$$E[r_t | x_t, a_t = i] = x_t^\top \theta_i$$

Where x_t is the context vector and θ_i is the unknown parameter vector for arm i.

At each round, Thompson Sampling performs the following steps for each arm:

- Sample $\tilde{\theta}_i \sim \mathcal{N}(\hat{\theta}_i, v^2 A_i^{-1})$
- Estimate reward: $\hat{r}_i = x_t^\top \tilde{\theta}_i$

Then the arm with the highest sampled reward estimate is selected.

Bayesian Update Equations

Each arm maintains:

- A design matrix $A_i = \lambda I + \sum x_s x_s^\top$
- A reward vector $b_i = \sum r_s x_s$

After receiving reward r_t for the chosen arm i, the algorithm updates:

$$A_i \leftarrow A_i + x_t x_t^\top \quad b_i \leftarrow b_i + r_t x_t$$

The posterior mean estimate is $\hat{\theta}_i = A_i^{-1} b_i$. The variance parameter v^2 controls exploration, and λ is a regularization factor to prevent overfitting.

Implementation Details

In our implementation:

- The dimension of the context is 3.
- We used $\lambda = 1.0$ and $v^2 = 0.1$ as standard choices.
- At each round, the algorithm samples $\tilde{\theta}_i$ for each arm and selects the one with the highest expected reward.

Although it is theoretically sound, Thompson Sampling does not perform well in non-stationary environments unless it has adaptation processes like discounting, sliding windows, or resets. Our implementation follows the standard technique as specified in the literature to test its baseline performance.

While it is not the best for dynamic environments, Thompson Sampling is easy and elegant to implement. It is a good probabilistic baseline and provides useful insights when compared with UCB-based algorithms

4.4 Software and Tools

This project was developed on using Python 3.11 on a Windows 11 operating system. All experimental and data generation scripts were written from scratch using only open-source libraries. The primary packages used are mentioned below:

- **NumPy** (v1.26.4): For numerical operations and vector manipulations
- **Pandas** (v2.2.1): For dataset construction, transformation, and storage
- **Matplotlib** (v3.8.3): For plotting and visualizing results
- **Scikit-learn** (v1.4.1): Used for applying PCA (Principal Component Analysis)

All simulations were performed locally without the use of GPU acceleration. Code was broken into methods per algorithm and a data generator. Output was stored in CSV files and processed using Jupyter notebooks. Plotting scripts were used to produce regret curves, accuracy plots, and action usage plots.

For bias analysis, context feature distributions were projected to 2D using PCA for visual inspection of class fairness. Class ratios, reward distributions, and round-level statistics were pulled directly from values logged when performing simulations.

5 Experimental Results

All three algorithms were evaluated on the same synthetic dataset of 1500 rounds, featuring realistic non-stationary behavior through user preference drift. For each round, the algorithm selected an action based on the current context, received a reward, and logged whether the optimal action was chosen. We tracked cumulative reward, cumulative regret, rolling accuracy, and arm usage.

5.1 Summary of Performance Metrics

The table below summarizes each algorithm's final cumulative reward, final regret, and overall accuracy. These values are taken from the last recorded round and represent total performance over the entire interaction horizon.

Table 2: Summary of Algorithm Performance on Dynamic Dataset

Algorithm	Cumulative Reward	Final Regret	Accuracy (%)
SW-UCB	938	149.86	80.6%
D-UCB	900	217.07	73.7%
Thompson Sampling	581	510.12	39.4%

As shown in Table 2, SW-UCB obtained the highest cumulative reward and lowest regret, indicating effective adaptation to changing preferences. D-UCB also performed well but adapted more slowly. Thompson Sampling achieved the lowest reward and highest regret due to its lack of explicit adaptation.

5.2 Cumulative Reward and Regret

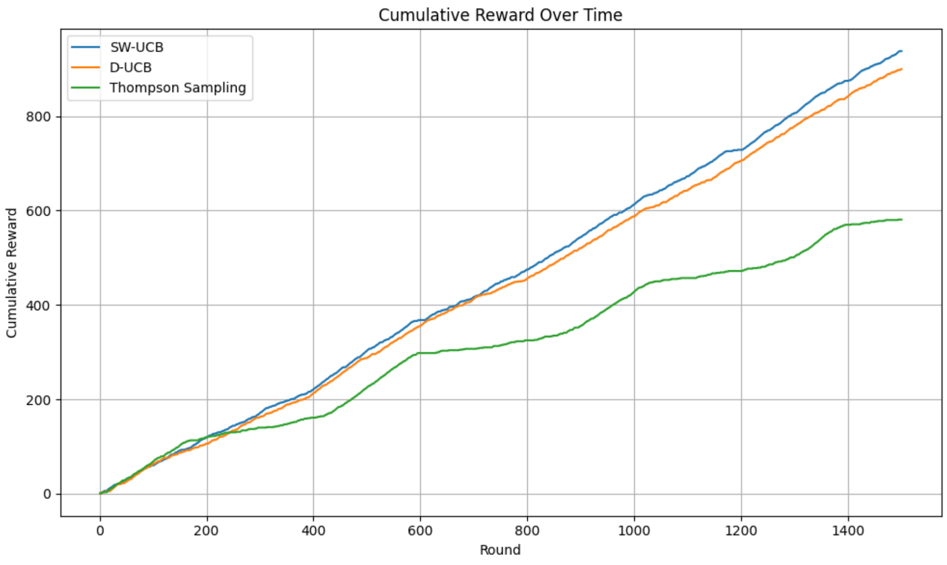

Figure 2: [Cumulative reward comparison of all three algorithms over 1500 rounds]

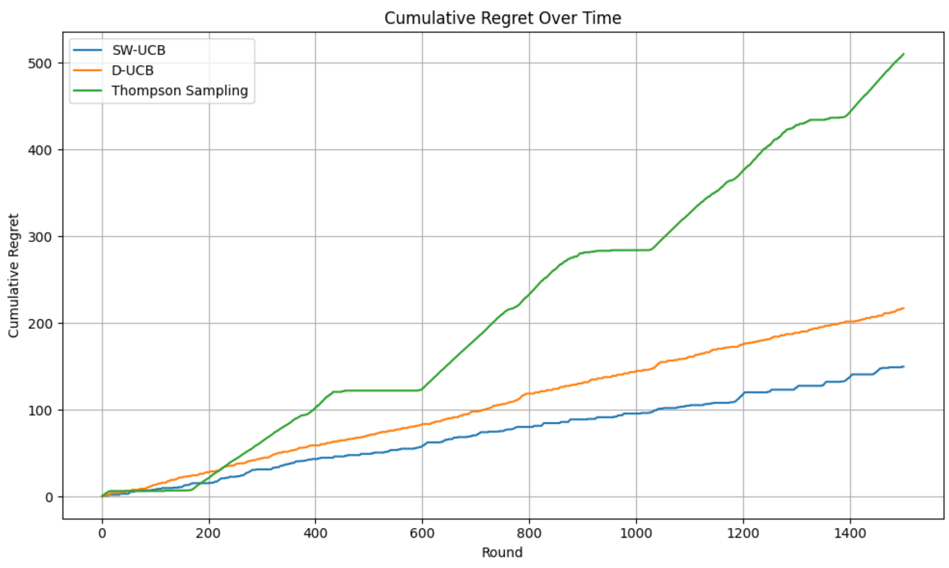

Figure 3: [Cumulative regret comparison — lower is better. SW-UCB adapts faster to drift]

5.3 Rolling Accuracy and Arm Usage

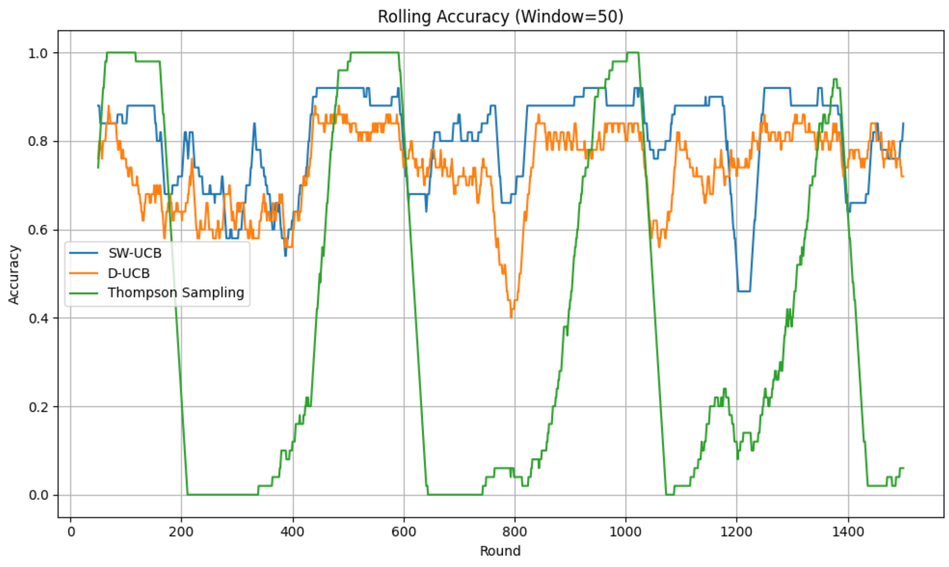

Figure 4: [Rolling accuracy over time (window = 50 rounds). SW-UCB maintains high stability. Thompson Sampling fluctuates heavily under drift.]

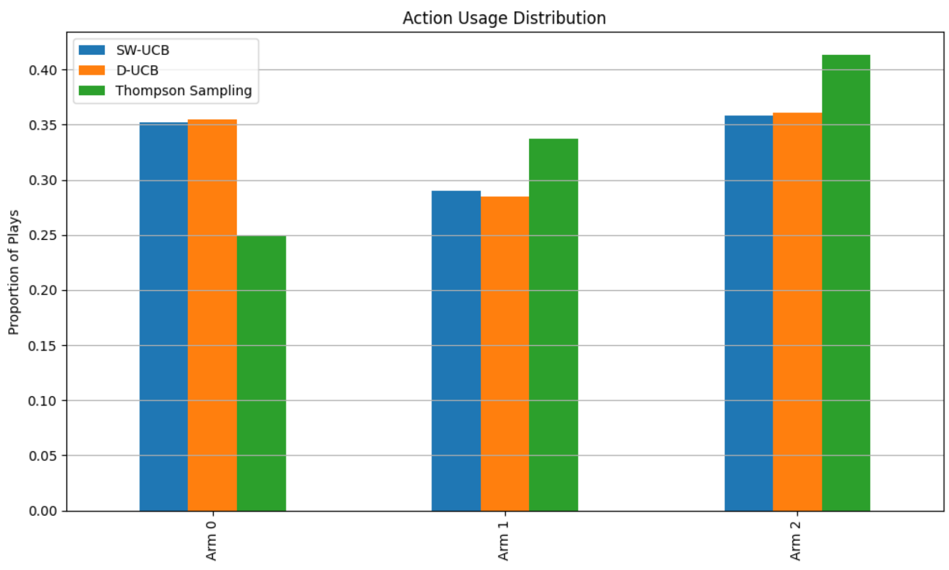

Figure 5: [Action usage distribution across 1500 rounds. All arms are explored, but Thompson Sampling overplays Arm 2.]

6 Discussion

The results confirm that SW-UCB performed optimally on all the metric scores. It achieved the maximum cumulative reward of 938 and minimum regret at 149.86. It achieved the maximum accuracy of 80.6%. This verifies that using a fixed sliding window allows the algorithm to quickly react to shifts in user interests.

D-UCB also performed fairly well but was slow to respond. It came up with a cumulative reward of 900 and a final regret of 217.07, with the cumulative accuracy being 73.7%. Since D-UCB discounts older rewards gradually, it responds more gradually when the environment changes abruptly, but remains good in the long term.

Thompson Sampling was worst. It had a cumulative reward of only 581 and had the maximum regret of 510.12. Its accuracy was also the lowest, at only 39.4%. The algorithm overestimated in previous arms and did not recover fast when user behavior changed.

An interesting observation is that SW-UCB, which uses a simple window mechanism, handled dynamic drift better than Thompson Sampling, which is based on Bayesian learning. This shows that in dynamic environments, forgetting old data can be more effective than trying to model everything.

7 Code and Execution Instructions

All scripts and dependencies are included in the public repository:

GitHub Link: `https://github.com/SumitTeli/Contextual-Bandits-in-Dynamic-Environments`

Step-by-Step Execution Guide

1. **Clone the project**

```
git clone https://github.com/SumitTeli/Contextual-Bandits-in-Dynamic-Environments
```

2. **Go to the project directory**

   ```
   cd contextual-bandits-project
   ```

3. **Activate the virtual environment:**

   ```
   .\venv\Scripts\activate
   ```

4. **(Optional) If not already created, set up manually:**

   ```
   python -m venv venv
   .\venv\Scripts\activate
   pip install -r requirements.txt
   ```

5. **Generate the dataset**

   ```
   python Dataset-dynamic\dataset.py
   ```

6. **Run all algorithms**

   ```
   python methods\SW-UCB.py
   python methods\D-UCB.py
   python methods\ts.py
   ```

7. **Run the bias analysis**

   ```
   python "Biasness check\bias_check_summary.py"
   ```

8. **Generate result visualizations**

   ```
   python ComparisonPlot.py
   ```

Outputs

All .csv logs and .png result plots are generated in the root directory for direct use in the report.

8 Challenges

One challenge faced during the work was adapting parameter for D-UCB and SW-UCB, such as the discount factor and window size. These values did impact performance little bit, and the tuning had to be done with a manual effort to maintain the balance between reactivity and stability.

Another issue was Thompson Sampling. Even though it is widely used and mathematically sound, it did not perform well on our dynamic dataset. It became overly confident of its early decisions and never really bothered to adjust as the preferences changed. This suggests that standard Thompson Sampling is not ideal for non-stationary environments without additional mechanisms.

9 Conclusion

This project aims to evaluate how different everyday contextual bandits algorithms adapt to environments where a user's preferences change with time.

The results clearly show that algorithms that prioritize recent information, such as SW-UCB and D-UCB, are much more effective in dynamic settings than static approaches, such as standard Thompson sampling.

SW-UCB ranked first overall in performance, with quick adjustment to shifts in preferences by means of a sliding window. D-UCB ranks a close second with a little smoother but somewhat lagged adaptation. In contrast, Thompson Sampling could do poorly having no mechanism to forget the outdated information.

Overall, this study confirms that in the real, evolving environment, to forget the past is as important as learning from it.

References

References

[1] A. Garivier and É. Moulines, "On upper-confidence bound policies for non-stationary bandit problems," in *Proceedings of the 24th Annual Conference on Learning Theory (COLT)*, 2011. [Online]. Available: https://arxiv.org/pdf/0805.3415

[2] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013. [Online]. Available: https://arxiv.org/pdf/1209.3352

[3] D. Russo, B. V. Roy, A. Kazerouni, I. Osband, and Z. Wen, "A tutorial on thompson sampling," *Foundations and Trends in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018. [Online]. Available: https://arxiv.org/pdf/1707.02038