# Offline Evaluation of LinUCB

COMS 6998-02: Advanced Machine Learning for Personalization
Homework 2

Kevin J. Wu - kjw2157

March 24, 2018

# 1 Introduction

## 1.1 Contextual Bandit Setting

The contextual bandit is an extension of the multi-armed bandit problem in which at each time step $t$, we observe a $d$-dimensional feature vector $x_t \in \mathbb{R}^d$. We are given a fixed set of arms $\mathcal{A}$, and a policy $\pi$ mapping contexts to a probability density over discrete actions $a \in \mathcal{A}$. At each round $t = 1, 2, ...T$ we have the following sequence of events:

- The environment samples $x_t \sim D$. (In this case, we assume $x_t$ are sampled i.i.d.).
- The algorithm chooses an action $a_t \in \mathcal{A}$ according to $\pi(x_t)$.
- The environment samples a reward $r_t \sim D(r|x_t, a_t)$.

We use $D$ to denote nature's distribution over contexts as well as the conditional distribution over rewards given a context-action pair. For this particular dataset, rewards are binary random variables, so $r_t \in [0, 1]$.

Because at each round $t$, the algorithm only observes the reward conditional on the action $a_t$ and not on any of the other arms in $\mathcal{A}$, the contextual bandit (along with the general multi-armed bandit setting) requires a tradeoff between *exploration* of actions and *exploitation* of those actions which are most likely to lead to high reward.

## 1.2 LinUCB

The contextual bandit learning algorithm used in this assignment is a variation on the LinUCB algorithm proposed by Li et al. 2012a [1]. Instead of using disjoint linear models, however, in which the context vector $x_t$ is partitioned into subsets $x_{t,a}$ for all arms $a$, we assume a joint context vector $x_t$. The expected reward is estimated to be linear in this joint context vector, while the estimated parameters are specific to each arm. This linear reward model is defined as follows:

$$\mathbb{E}[r_t|x_t, a_t] = x_t^T \theta_{a_t} \tag{1}$$

The parameter vector $\theta_a$ for all $a \in \mathcal{A}$ can be estimated using ridge regression. In the online setting, given $m$ rounds in which arm $a$ was chosen, we define $\boldsymbol{D_a}$ to be a $t \times d$ matrix whose rows consist of the context vectors $x_i$ where $a_i = a$. Let $\boldsymbol{c_a}$ be a $m$-dimensional vector consisting of rewards $r_i$ for all $i$ where $a_i = a$. The ridge regression estimate for $\theta_a$ is:

$$\hat{\theta}_a = (\boldsymbol{D_a^T D_a} + \boldsymbol{I})^{-1} \boldsymbol{D_a^T c_a} \tag{2}$$

where $I$ is the $d \times d$ identity matrix.

For all actions $a$, define matrix $\boldsymbol{A_a} := \boldsymbol{D_a^T D_a} + \boldsymbol{I}$. The LinUCB update rule selects actions at each round according to the following:

$$a_t = \arg\max_{a \in \mathcal{A}} \quad \left( x_t^T \hat{\theta}_a + \alpha \sqrt{x_t^T \boldsymbol{A_a^{-1}} x_t} \right) \tag{3}$$

Thus, at each time step LinUCB selects the arm with the highest upper confidence bound as defined by the expected payoff $x_t^T \hat{\theta}_a$, and the standard deviation of the regression estimate, $\sqrt{x_t^T \boldsymbol{A_a^{-1}} x_t}$, with the hyperparameter $\alpha$ determining the width of the confidence interval around the estimated mean payoff.

## 1.3 Offline Policy Evaluation

While LinUCB describes a policy for choosing actions in an online setting, one can evaluate the effectiveness of LinUCB (and other learning algorithms) on offline, or logged, data in the following manner:

Given a stream of data $S$ consisting of context-action-reward triplets, we step through each event $t$, and if the agent samples an action $a_\pi \sim \pi(x_t)$ where $a_\pi = a_t$, we add the $t$-th event to our history and update the policy accordingly; otherwise, we discard the event and proceed to $t + 1$.

This offline evaluation method will yield an unbiased estimate of the true expected value of a policy in an online setting [2].

### 1.3.1 Evaluation Metric

The evaluation metric used in this homework is the cumulative take-rate, or CTR, which is the average reward obtained over events in which the policy's chosen arm coincides with the arm chosen in the logged data.

$$CTR_t = \frac{\sum_{i=1}^{T} y_i \times \mathbb{1}[\pi_{i-1}(x_i) = a_i]}{\sum_{i=1}^{T} \mathbb{1}[\pi_{i-1}(x_i) = a_i]} \tag{4}$$

Table 1: Observed frequencies of arms in logged data.

| **arm** | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| **frequency** | 1020 | 982 | 974 | 1047 | 1005 | 963 | 1035 | 999 | 988 | 987 |

# 2 Experiment

## 2.1 Dataset

The dataset for this assignment consists of 10,000 logged events, each consisting of a binary reward $r \in [0, 1]$, an arm $a \in \{1, 2, ..., 10\}$, and a 100-dimensional feature vector $x \in \mathbb{R}^{100}$.

The dataset was collected under a random exploration policy, in which all arms are sampled uniformly. Table 1 shows the observed frequency of each arm in the data.

## 2.2 Code

The code used to evaluate LinUCB policies on the given data can be found in *COMS6998-HW2-Code.ipynb*.

## 2.3 Results

The only tunable parameter in the LinUCB is $\alpha$, which controls the agent's degree of exploration in the environment. As discussed in Section 1, $\alpha$ controls the upper confidence interval bound around the estimated reward. Higher $\alpha$ will cause the LinUCB algorithm to choose arms with greater uncertainty over those with slightly higher estimated reward. To examine the effect of this parameter, I explore three strategies for choosing $\alpha$:

1. *Constant* $\alpha$: In these simulations, $\alpha$ is fixed over the course of the simulation, effectively maintaining the same degree of exploration over all rounds. Simulations were run over a set of $\alpha$ values ranging from 0.001 to 1.0. (Figure 1)

2. *Annealed* $\alpha$: Here $\alpha$ is decayed over time according to the following schedule: $\alpha_t = 1/t^\gamma$. By decaying $\alpha$ over time in this way, the exploration is gradually decreased over time in favor of exploitation. The parameter $\gamma$ controls the speed of this decay; simulations were run using a set of $\gamma$ values ranging from 0.1 to 2.0. (Figure 1)

3. *Hybrid* $\alpha$: This is a combination of strategies (1) and (2). $\alpha$ is initially decayed according to the update rule in (2), with the additional constraint that $\alpha$ be greater than some $\alpha_{min}$. Effectively this decays exploration over time while ensuring a minimum level of exploration as $t$ increases. A combination of the best-performing parameters from strategies (1) and (2) were chosen for this experiment ($\alpha_{min} \in \{0.001, 0.005, 0.1\}$ and $\gamma \in \{0.5, 1.0\}$). (Figure 2)

On this particular dataset, for fixed $\alpha$, we can see that CTR increases as we lower $\alpha$ up to $\alpha = 0.01$, after which further decreasing $\alpha$ no longer results in significantly better accuracy. With $\alpha \geq 0.5$, the LinUCB agent is particularly slow to learn, with CTR barely

increasing above the random baseline (10% CTR) after 5000+ iterations. It appears that with high enough $\alpha$, the argmax arm choice becomes dominated by the standard deviation component of the upper confidence bound, rather than the mean expected reward, resulting in consistently suboptimal arm choices.

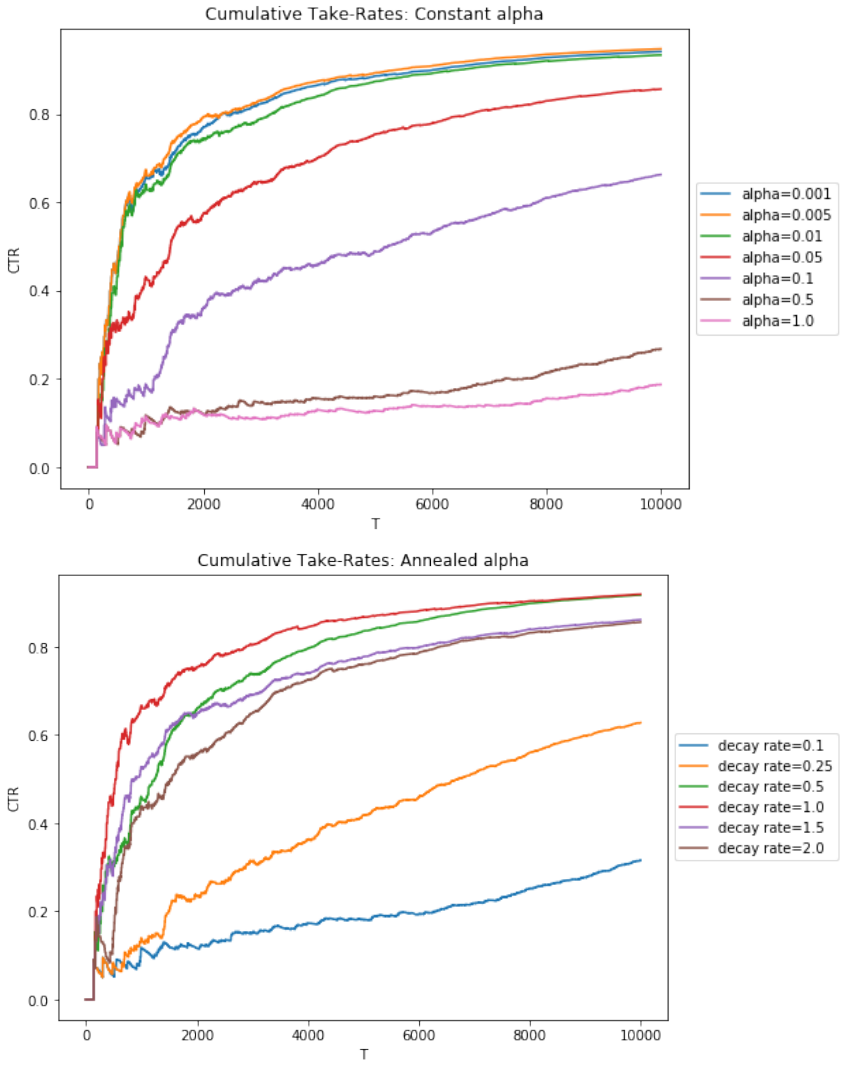


Figure 1: Constant vs. Decaying Exploration

For a decaying $\alpha$, the optimal exploration schedule appears to be either $\alpha_t = 1/\sqrt{t}$ or $\alpha_t = 1/t$. Slower rates of decay (i.e. $\alpha_t = 1/t^{0.1}$ or $\alpha_t = 1/t^{0.25}$ maintain excessively high rates of exploration throughout the training period, resulting in lower CTRs similarly to fixed exploration schedules with $\alpha \geq 0.5$.

Finally, we can see that a hybrid approach in which we maintain a sufficiently small level of exploration through all rounds (Figure 2), results in comparable performance to the optimal fixed and decayed $\alpha$ schedules.

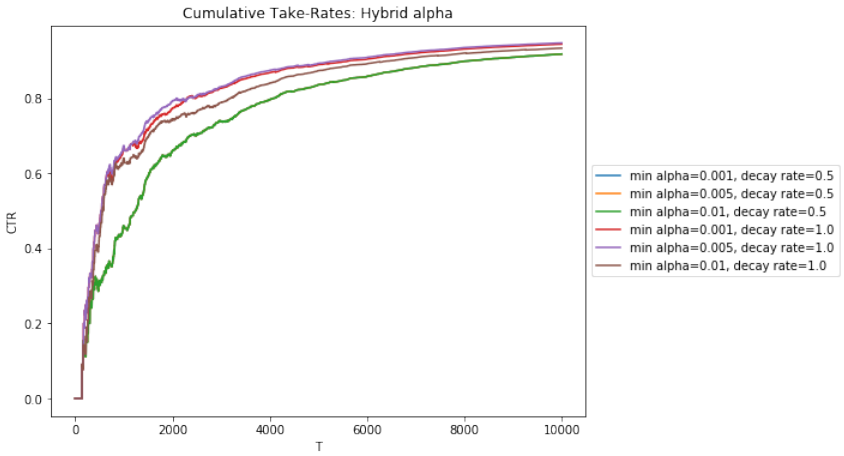


Figure 2: Hybrid Exploration Strategy

All three strategies for determining $\alpha$ result in comparable CTR rates for their optimal hyperparameter settings. Given the relatively small size of the dataset and the high CTRs across training runs, it appears that a "naive" implementation of LinUCB with constant $\alpha$ is able to sufficiently balance exploration and exploitation in this environment. It is important to note that a decaying $\alpha$ rate is not necessary for decreased exploration over time. In fact, the LinUCB algorithm, just like the regular UCB algorithm, results in a natural decrease of exploration over time as the standard deviation of the predicted reward decreases over time.

### 2.3.1 LinUCB under Non-Uniform Logging Policy

Offline policy evaluation of LinUCB under replay method is unbiased if the logging policy used to gather the offline data picked arms uniformly at random. Given the original classification dataset, I performed additional experiments to analyze the efficacy of LinUCB under a non-uniform logging policy in this same environment.

To convert the multi-class classification dataset into the bandit setting, for each event in the original dataset, I picked an arm (corresponding to a class label in the classification setting) according to a probability distribution, $P(x)$. The probability distribution over the arms $\mathcal{A}$ for this experiment is defined as follows:

$$P(x = k) = \begin{cases} 0.19, & 1 \leq k \leq 5 \\ 0.01, & 6 \leq k \leq 10 \end{cases}$$

For each event, if the randomly sampled arm/class matched up with the true class, it was assigned a reward of 1 in the bandit dataset, and 0 otherwise. The sampled arm, its corresponding reward, and the feature vector were concatenated for each event to form the "banditized" dataset.

Table 2 shows the performance of LinUCB on off-policy logged data obtained under uniform and non-uniform sampling of actions (columns $(i)$ and $(ii)$) in terms of the final

CTR after the simulation. For identical settings of $\alpha$, the performance of LinUCB on the non-uniform dataset is consistently lower. However, for the optimal settings of $\alpha$ in the uniform case, the final CTR obtained on dataset (ii) is very close to the final CTR under dataset (i). In this particular case, uniformity of the logged data is not necessary to "solve" the bandit learning problem.

| | *CTR* | |
|---|---|---|
| *alpha* | *(i)* | *(ii)* |
| 0.001 | 0.94 | 0.90 |
| 0.005 | 0.95 | 0.94 |
| 0.01 | 0.93 | 0.92 |
| 0.05 | 0.86 | 0.80 |
| 0.1 | 0.66 | 0.42 |
| 0.5 | 0.27 | 0.15 |

| | *CTR* | |
|---|---|---|
| *gamma* | *(i)* | *(ii)* |
| 0.1 | 0.32 | 0.16 |
| 0.25 | 0.63 | 0.44 |
| 0.5 | 0.92 | 0.89 |
| 1.0 | 0.92 | 0.88 |
| 1.5 | 0.86 | 0.69 |
| 2.0 | 0.86 | 0.79 |

| | *CTR* | |
|---|---|---|
| *(gamma, alpha)* | *(i)* | *(ii)* |
| 0.5, 0.001 | 0.92 | 0.89 |
| 0.5, 0.005 | 0.92 | 0.89 |
| 0.5, 0.01 | 0.92 | 0.89 |
| 1.0, 0.001 | 0.94 | 0.92 |
| 1.0, 0.005 | 0.95 | 0.94 |
| 1.0, 0.01 | 0.93 | 0.92 |

Table 2: Final CTR of LinUCB algorithm using uniformly logged data ($i$) and non-uniformly logged data ($ii$). From left to right: constant alpha, exponentially decayed alpha, hybrid alpha.

### 2.3.2 Rejection sampling

The authors of the LinUCB paper propose rejection sampling as a way to obtain an unbiased estimate on logged data from a non-uniform policy over actions [1,2]. Given the propensities of each arm (see previous section), we accept each event in the logged data in inverse proportion to its propensity in order to obtain a uniform distribution of arms. The distribution of arms in the logged data and in the accepted sample after rejection sampling is shown in Figure 3.

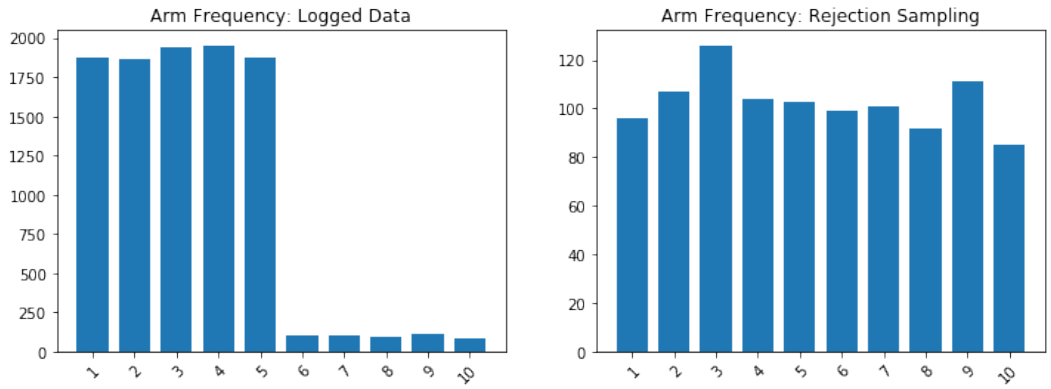


Figure 3: Distribution of arms before and after rejection sampling. Note the decrease in overall data size after rejection sampling.

While offline policy evaluation maintains its unbiasedness guarantees after rejection sampling, the main drawback of this approach is the decreased data efficiency. Since only ∼1000

events remain after rejection sampling (from a total of 10,000 logged events), LinUCB with fixed $\alpha$ is only able to obtain a maximum CTR of 0.65. However, given more data, we would expect this algorithm to be consistent and to eventually approach the true CTR as $T \to \infty$.

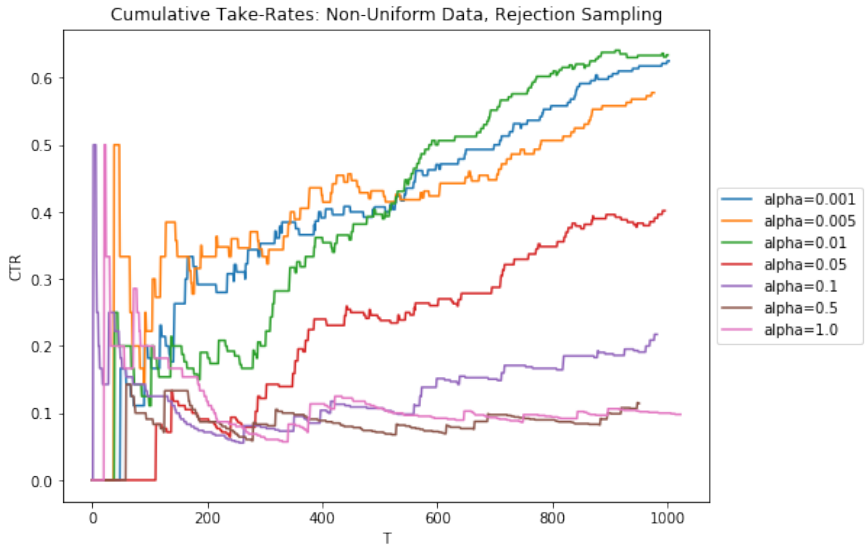


Figure 4: Off-policy evaluation of LinUCB on non-uniform logged data, with rejection sampling.

# 3 Conclusion

In this experiment I explore the effectiveness of the LinUCB method as a contextual bandit learning algorithm, using logged data. I choose three different strategies for selecting the confidence interval width determining the arm choice at each step. Using cumulative take-rate (CTR) as the evaluation metric, I compare the performance of these three strategies under various sets of hyperparameters. Finally, I explore the effect of a non-uniform logging policy on LinUCB performance under this same environment, and show a proof-of-concept of a rejection sampling method that maintains the unbiasedness guarantees of offline LinUCB evaluation on uniformly sampled arms.

# References

[1] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," *CoRR*, vol. abs/1003.0146, 2010.

[2] L. Li, W. Chu, and J. Langford, "An unbiased, data-driven, offline evaluation method of contextual bandit algorithms," *CoRR*, vol. abs/1003.5956, 2010.