Evaluating Bandit Policies Across Datasets

Aidan Claffey
New York University
aidan.claffey@nyu.edu

Chris Ick
New York University
chris.ick@nyu.edu

Johnny Ma
New York University
johnnyma@nyu.edu

Dan Turkel
New York University
dan.turkel@nyu.edu

ABSTRACT

Multi-armed bandits are a family of reinforcement learning models that are well-suited for recommendation, particularly in scenarios where user feedback is plentiful, such as e-commerce or streaming platforms. Bandits use a selection policy to choose a finite number of offerings (called actions), and they observe "rewards" based on whether the user interacts with those actions which are used to update policy parameters. The online learning nature of bandits makes offline experimentation difficult as bandits are being trained on interactions coming from whichever selection policy was used to create the logs. Two approaches to solving this problem are the use of off-policy estimators, which attempt to adjust for the counterfactual nature of the offline learning, and simulations which allow for online learning but require a way to generate rewards for all user-action pairs. We compare these two methods by training several multi-armed bandit policies across two recently released datasets: the Open Bandit Dataset from ZOZO and the Deezer playlist dataset. Our experiments are built on the Open Bandit Pipeline, released by ZOZO alongside their dataset. We run several experiments on each dataset, and note the strengths and weaknesses of policies and evaluation approaches. Furthermore, we release our experiment code, enabling future reproducible experimentation with additional policies and datasets not explored in this work.

1 INTRODUCTION

Multi-armed bandits are a family of algorithms for selecting a subset of actions from a larger pool by trading off exploration of untested actions with exploitation of those known to be successful. Bandits are defined by a parametrized selection policy which picks the actions to test for a given round, and those parameters are updated when they observe the "rewards" of their selections. Bandits are attractive for recommendation problems as the problem formulation is analogous to recommendation: the actions are items and the rewards are clicks, streams, purchases, views, etc. A/B tests offer a way to compare bandit algorithms in the form of a hypothesis test and have the advantage of collecting real data in a production scenario on real users. However, A/B tests may not be feasible during the research phase, when there is a larger space of options, some of which are only minor variants. As a result, it is useful to be able to evaluate bandit policies offline by comparing estimated or simulated performance.

The sequential and online nature of bandit models makes offline evaluation difficult. In this paper, we explore two datasets which require separate methods for comparing bandit performance. In the Open Bandit Dataset (OBP) from ZOZO [1], we have a dataset of logged interaction data from two separate policies, and new policies can be trained using those logs and their performance estimated using off-policy estimators. In the Deezer carousel dataset [2], we have user-item click probabilities we can use to simulate online learning and evaluate the performance by simply looking at the simulated rewards.

There have been several recent approaches to solve the problem of offline bandit evaluation. A simple approach was proposed by researchers at Yahoo in [3] [4], where a policy is learned offline on logged data by only using logged interactions which "match" the selected actions of the counterfactual policy, which grossly reduces the number of data points the policy can learn on and works best if the logs were generated using a uniformly random policy.

More complex approaches like Inverse Probability Weighting [5] and Self-Normalized Inverse Probability Weighting [6], the Direct Method [7], and double robust estimation [8] have been proposed, and the task remains an area of active research.

The recently released Open Bandit Pipeline (OBP) [1] provides a framework for studying both policies and estimation methods. Our work extends ZOZO's off-policy estimation and Deezer's simulation work to provide a side-by-side comparison of how policies perform on the two datasets and how the different offline evaluation methods impact the results.

2 BANDIT METHODS

Multi-armed bandit methods are deployed in scenarios where we have a selection of items ("*arms*"), each associated with an unknown distribution of *rewards*, $r \in [0, R_{\max}]$. The reward distribution is typically observed as dependent to some action $a \in \mathcal{A}$ and some context $x \in \mathcal{X}$, so that $p(r) \sim p(r|a, x)$. Given a context, the action is selected some distribution $\pi(a|x)$, which is known as the *policy*. *Context-free* policies do not rely on context x, so those policies are just written as $\pi(a)$.

Due to the iterative nature of bandit recommenders, logged data take the form of a series of observed contexts, actions, and rewards generated from some behavior policy π_b over T rounds:

$$\{(x_t, a_t, r_t)\}_{t=1}^{T} \sim \prod_{t=1}^{T} p(x_t)\pi_b(a_t|x_t)p(r_t|x_t, a_t) \tag{1}$$

We can define the join distribution of the logged data of a given context as $\pi(x, a, t) := p(x)\pi(a|x)p(r|a, x)$. Ideally, we would like to leverage the existing data generated from the behavior policy π_b, $\mathcal{D}$ to estimate a target evaluation policy π_e. This task objective is defined by the *policy value* of the evaluation policy:

$$V(\pi_e) := \mathbb{E}_{\pi_e(x,a,r)}[r] \tag{2}$$

Because the evaluation policy distribution π_e cannot be directly computed or measured in an offline setting, approaches using estimators for the policy value leveraging $\mathcal{D}$, s.t. $\hat{V}_{\text{est}}(\pi_e; \mathcal{D}) \approx V(\pi_e)$, have been proposed and implemented in [1].

An alternative approach is to design an iterative generative process for data synthesis, where data is iteratively sampled from a distribution from logged data, $\pi(x_t, a_t) \sim \pi(x_{t-1}, a_{t-1}|\mathcal{D})$. This allows us to explicitly compute the cumulative reward of a simulated dataset.

These two approaches are our comparative approaches to the task of *off-policy estimation* (OPE).

3 DATASETS

This project focuses on comparing two methods of policy evaluation on two publicly available datasets of historical data generated from online bandit systems: the ZOZO Open Bandit Dataset [1] and Deezer playlist carousel dataset [2].

3.1 ZOZOTOWN Open Bandit Dataset

The ZOZOTOWN dataset[1] contains user browsing activity on ZOZOTOWN, the largest fashion e-commerce platform in Japan. In November of 2019 the platform ran a series of "campaign" experiments on clothing item recommendations. The data from this experiment contain 26 million user impressions and click through logs over a 7 day period, spread across "Men's", "Women's" and "All" shopping categories. This dataset was released as the Open Bandit Dataset (OBD) for the purpose of establishing a public benchmark for bandit evaluation. For each user impression session, we observe the items recommended, their position in the recommendation interface, a probability of each item being assigned to said position, and if the recommended items are clicked on. The dataset also contains user and item embeddings learned by their model. Using the logged bandit feedback data and ground truth from two baseline policies (uniform random and pre-trained Bernoulli Thompson), we can run other bandit policy experiments and evaluate their offline bandit metrics on real browsing data.

This work evaluated the results on a small set of policies and a broad range of estimators using the methodology released with the Open Bandit Pipeline (OBP) described in Section 4. Prior work has focused on the performance methods of different estimators for approximating the performance of a small set of policies. Contained in the OBP are many policies that had not been previously evaluated on the OBD, as prior work focused on evaluating the quality of the estimators themselves, rather than that of the policies.

3.2 Deezer Carousel Dataset

The Deezer dataset[2] contains user and playlist interaction probability data from French music streaming service Deezer. The dataset task is playlist recommendation, filling 12 slots in a "carousel" module on a user's homepage with one of 862 professionally curated playlists centered around genres, geography, or mood. Success for this task is measured by playlist selecting the playlist and streaming one full song (display-to-stream). While the dataset doesn't provide data from online A/B testing, it does offer an environment built from ~1 million user preferences to simulate responses to various bandit policies. This dataset is not real browsing dataset like the ZOZOTOWN dataset, but rather a simulated environment of probabilities to test various bandit algorithms. Latent user-item preferences are encoded as a 97-dimensional vector from the factorization of the interaction matrix between users and songs, with users also being placed into one of 100 clusters through k-means for "semi-personalization." Playlists are described as the weights on the 97-dimensional feature vectors, fit using logistic regression on click data from Janurary 2020 (data that generated the embeddings not available). The authors construct a "ground-truth" probability of a user streaming a playlist using a sigmoid activation on the dot product between user and playlist vectors. In this simulated environment, we can replicate their simulated bandit policy evaluations, as well as test new policies relative to their baselines.

4 IMPLEMENTATION

Our work builds upon the Open Bandit Pipeline [1], a Python library which offers an an environment for loading datasets, learning bandit policies, and performing off-policy estimation. The library was released by ZOZO Research alongside their Open Bandit Dataset, containing code to load and process their logged feedback data. The library also comes with several prebuilt bandit policies and off-policy estimators.

Our contribution includes the `DeezerDataset` class to load the Deezer data and learn policies, several new policies implemented in Deezer's paper that were not in OBP, an `Experiment` class to easily run multiple policies and off-policy estimators on a dataset, a script to run experiments defined entirely in configuration files, and several new visualizations of the results. All of our code is available at https://github.com/daturkel/sd_bandits.

4.1 Deezer Dataset Loader and Simulation

The OBP only has a builtin dataset loader for ZOZO's Open Bandit Dataset. Using the supplied base class for datasets, we added functionality to load the Deezer data.

However, OBP's policies require feedback logs in order to learn. For the Open Bandit Dataset, the dataset loader simply passes the logs to the policy. For the Deezer data, since we only have click probabilities to work with, the dataset loader simulates *online* policy learning using a provided policy.

We implemented this online learning to match the simulation done in Deezer's paper. For each "batch," we pick a random set of 20,000 users with replacement, and provide actions (playlists) for each user based on the provided policy. Using Deezer's provided features, we can calculate the click probabilities for each playlist presented to each user and then generate Bernoulli random variables to determine whether or not the user "clicked" each playlist. These clicks are then used to update the policy's parameters, and the process is repeated over 100 batches.[3]

We remained truthful to several aspect's of Deezer's simulation designed to make it realistic. In particular, the "cascade" model of reward observation when using the Deezer app, whose UI only

[1] https://research.zozo.com/data.html
[2] https://zenodo.org/record/4048678

[3] The Open Bandit Pipeline includes a `SyntheticBanditDataset` for performing the same type of simulated online learning, but we did not discover it until late into the project. However, we still would have had to heavily modify it to implement the Deezer-specific behavior described in this section.

displays the first three playlists without deliberate user scrolling. If the user doesn't click any of the playlists, we assume that they only saw the first three and thus only "observe" the (negative) feedback for the first three playlists, not learning anything about the other nine. If the user *did* click any of the playlists, we observe all rewards up until the *first* playlist they clicked, to simulate the fact that they likely didn't continue scrolling once they found a playlist they liked.

In our implementation, the cascade effect can be modified to observe all rewards until the *last* clicked playlist (rather than the first), or can be disabled altogether. The number of batches and users per batch are also configurable.

As in the Deezer experiment, our simulation does not update the bandit policy parameters until the end of each batch in order to mirror the nightly retraining cycle of bandit systems in production.[4]

The end result is that for any policy, we can simulate multiple days of users interacting with the bandit, collect rewards, and retrain once per batch. Because our policy is learning on simulated data, rather than logged data, we do not need to use an off-policy estimator to evaluate the performance and can instead interpret the "online" rewards directly.

4.2 Bandit Policies

The Open Bandit Pipeline only implements some of the policies we wanted to evaluate. To expand the pipeline, we built a number of policies that would run on top of the OBP. These policies were *explore-then-commit* and *KL-UCB*. Additionally, we built a wrapper class for any context-free policy to be turned into a segmented policy (brief descriptions of policies can be found in Section 5).

4.3 Experimentation Framework

Much of the OBP is designed around evaluating the performance of off-policy estimators, with the examples in the OBP paper, codebase, and documentation reflecting this focus. While there is code supplied for estimator evaluation, we built out new functionality for running policy-oriented experiments to shift the focus to comparing policy performance across and within datasets.

We built a generic `Experiment` class and implemented it as `OBDExperiment` and `DeezerExperiment`.

For an `OBDExperiment`, we pass the experiment the dataset, a list of policies, a list of off-policy estimators, and optionally a regression model if any of the estimators require one. Running the experiment then performs several steps:

(1) Get logged feedback from the OBD Dataset.
(2) Use logged feedback to run *off*-policy learning for each bandit policy.
(3) Estimate each policy's rewards using each off-policy estimator and generate bootstrapped confidence interval of estimated mean rewards.

For a `DeezerExperiment`, we pass the experiment the dataset, a list of policies, and optional simulation parameters (number of users per batch, cascade mode, etc.) which can be set differently for each policy. Running the experiment then performs a similar procedure to `OBDExperiment`, but without the need for off-policy estimation:

(1) Simulate online learning for each policy.
(2) Generate a bootstrapped confidence interval of mean rewards for each policy.

Both experiment classes save all relevant output, including action choices and counts, feedback, and estimated or observed rewards.

The `Experiment` classes can be used interactively in a notebook, but we also wrote scripts to automate building and running experiments. The end-user only needs to point the script to a directory with YAML files specifying the dataset, policies, and optionally estimators to use. This allowed us to easily run multiple longer experiments on NYU's Prince cluster without the need for an interactive environment or detailed setup script. This method provides opportunity for parallelized experiments of increased scale, even with the addition of future datsets and policies, should further work necessitate it.

4.4 Visualizations

We evaluate the overall success of the bandit policies by plotting their cumulative reward gain as each round progresses. To capture the evolution of bandit performance over time, we also show each policy's running average and cumulative rewards. Note that for the Deezer experiment, rewards are indexed by batch rather than round, since each batch can contain a variable number of rounds due to the cascade model of observation.

Finally, for every proposed policy we calculate its relative estimated policy value by normalizing its estimated reward structure with the "ground-truth" policy's expected rewards, with policy value calculated as per Equation (2). The ground-truth policy for the `OBDExperiment` uses the logged data they observed from their experiments (either a uniform random or *Bernoulli Thompson* policy), while the `Deezer` "ground-truth" comes from the rewards of simulating a uniform random policy. The `OBDExperiment` must use an *off*-policy estimator over the logged to calculate $\hat{V}$, whereas `DeezerExperiment` can directly use online policies assuming their simulated environment can represent real browsing data. As $\hat{V}(\pi_e; \mathcal{D})$ can only estimate of the true policy value $\hat{V}$, we use the aforementioned bootstrapping methodology to generate 95% confidence intervals for our estimated relative policy values.

5 EXPERIMENTS

Since the two data sources vary in their approach to calculating rewards given an action, we treated the evaluation of policies for each source differently, coming up with different experiment pipeline and evaluation criteria for each. The differing methods we used are illustrated in Figure 1.

We divide the policies we tested into three categories: context-free, contextual, and segmented policies.

Context-free policies only take into account past rewards to inform the actions chosen. In particular, they are not personalized, so the items offered are not dependent on the user they are being presented to. Examples include random, ϵ-greedy, explore-then-commit, Bernoulli Thompson sampling [9], and KL upper confidence bound [10] [11].

[4] Deezer's experiment included one feature we did not replicate due to lack of time: For any policy, the first three actions are shuffled before performing the observation cascade, to account for the fact that the user theoretically sees the first three playlists simultaneously, not in stricly left-to-right order.

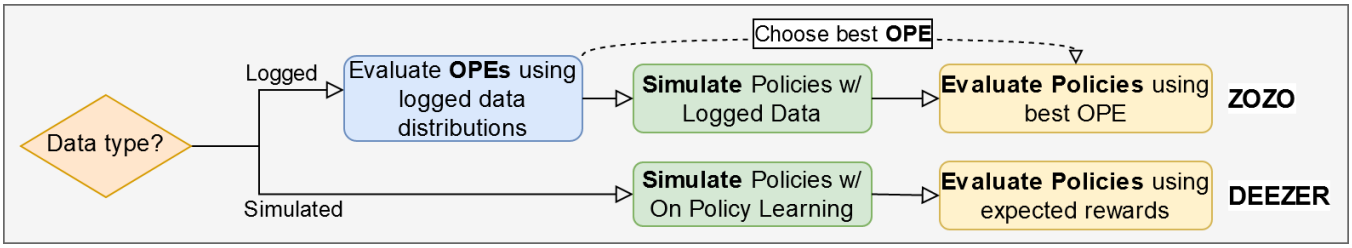


Figure 1: Experimentation and evaluation procedure for logged and simulated data

Contextual policies allow the bandit to personalize the actions selected by observing a user-specific and potentially time-dependent *context vector*. The bandit typically has access to action-specific contexts, and a goodness-of-fit between a user and each item is calculated as a function of the user context and item context.

The only contextual policy we included in our experiments was *Linear ϵ-greedy*, although the Open Bandit Pipeline also implements other contextual bandit policies that rely on both linear and logistic regression to handle user-item contexts.

Segmented policies leverage pre-calculated user clusters to learn a separate context-free bandit for each cluster. We only used these policies on the Deezer data, since user clusters were provided. Future work could explore generating clusters for users in the ZOZO dataset in order to run segmented policies on that data.

We ran multiple policies on each dataset. For all policies but random and KL-UCB, we experiment with two different hyperparameter settings; one biased toward exploitation and one toward exploration. As an example, `egreedy_exploit` will randomly choose an action only 1% of the time, whereas `egreedy_explore` will randomly choose an action 10% of the time. All hyperparameter settings and abbreviated policy names are listed in Table 1. As shorthand, `egreedy` is ϵ-greedy, `etc` is explore-then-commit, `ts` is Thompson sampling, `seg` indicates a segmented variant, and `lin` indicates a linear variant.

5.1 Off-Policy Learning and Estimation on OBD

As explained in section Section 3.1, the Open Bandit Dataset contains logged interactions for several campaigns using both the uniform random and Bernoulli Thompson policies. Since we have logged data *and* known distributions for the logged data, the best way to evaluate other policies is to use off-policy estimation. Choosing the right off-policy estimator is not a trivial matter. Luckily, [1] completed a benchmark on different estimators which we extend, explained in Section 5.1.1.

For now, assume we have chosen a few off-policy estimators to use for policy evaluation. Then, given a policy and the logged data, we simulate the policy offline. This is done by (i) choosing an action a_e given the evaluation policy/user, (ii) iterating through the logs until we find an action a_b that matches a_e, (iii) adjusting the policy parameters and overall reward counts based on the reward paired with a_b, and repeating (i)-(iii) until the logged data is fully iterated through. Finally, we evaluate the results of the offline simulation using the chosen off-policy estimator(s) and compare to the baseline logged results.

For our policy experiments, we used the "All" users campaign with the "Random" baseline behavior.

5.1.1 *Off-Policy Estimator Benchmarking.*

A difficulty with evaluating off-policy estimators is that we need a known baseline policy *and* a known evaluation policy so we can get the "ground truth" estimation. Luckily, the OBD contains logs from two baseline policies, uniform random and Bernoulli Thompson, which they used to benchmark. The protocol for evaluating off-policy estimators can be found in [1].

In their paper, Saito et. al evaluated in- and out-sample for both Uniform Random → Bernoulli TS and Bernoulli TS → Uniform Random cases for each of their campaigns. When extending their benchmarking and selecting the best estimator to use for policy evaluation, we decided to stick to the in-sample, Uniform Random → Bernoulli TS case. "In-sample" means that the ground truth for the evaluation policy is calculated on the same data that the estimator will use to estimate policy value, and Uniform Random → Bernoulli TS means that the Uniform Random policy logged data is used to evaluate on the Bernoulli TS policy. We chose this formulation because it seemed the most realistic— one should use the full sample to evaluate a policy (in-sample) and the best estimator should be able to estimate a more complicated policy from a simple one (Uniform Random → BTS) rather than the other way around.

To extend the benchmarking work done in [1], we use a `lightgbm` [12] model for estimators that require a regression model (such as Direct Method, Doubly Robust, and all of the Doubly Robust extensions). Saito et al. only used a logistic regression model but provided functionality for `lightgbm` in the OBP.

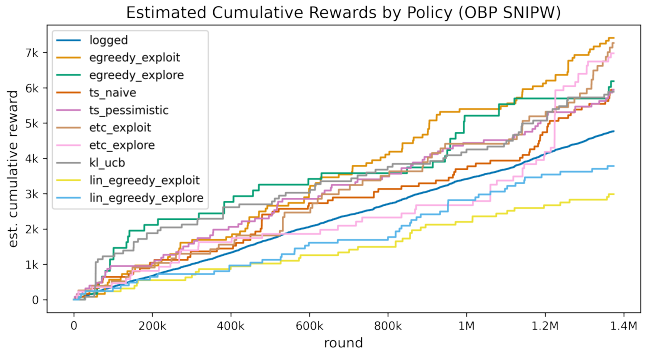
5.2 On-Policy Learning and Estimation on Deezer

Since we can directly calculate the probability of a playlist stream for each user-playlist pair in the Deezer dataset, we can learn bandit policies online without the use of ground-truth log data, as described in Section 4.1.

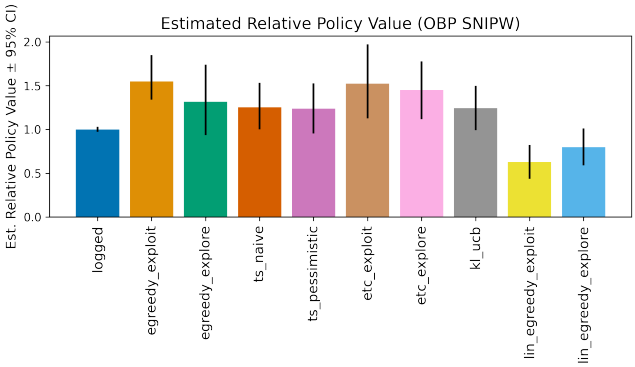
While our simulation code offers several configuration options, for our experiments we simply reproduced the primary settings[5] from Deezer's own experimentation: rewards are observed up to the first clicked playlist (or for the first three playlist if none were clicked), and each simulation contained 100 batches of 20,000 users each. Due to long computation times, we used 100 batches of 1000 users for Linear policies.

Once we have obtain our simulated log of rewards, we generate a 95% confidence interval of the mean from 100 bootstrapped samples.

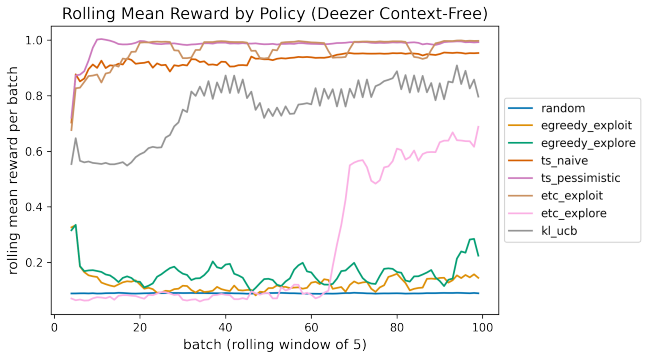
[5]Deezer found that disabling the cascade made all policies perform slightly worse, and that delaying the updates until the end of each batch tended to favor policies with randomization, since deterministic policies would be completely static for a full batch.



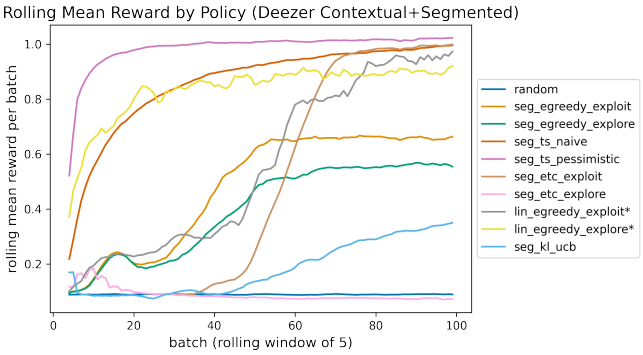
(a) SNIPW estimated cumulative rewards on OBP. Simple policies like ϵ-greedy and explore-then-commit performed better than contextual policies like linear ϵ-greedy.



(b) SNIPW estimated relative policy value on OBP. While there is clear difference in policy performance, the variance of the estimator muddles the results



(c) Deezer rolling mean rewards for context-free policies. The delayed ascents of `etc_explore` and KL-UCB iareapparent, but the dominant policies emerge almost immediately.



(d) Deezer rolling mean rewards, contextual and segmented policies. The segmented variant of `etc_exploit` learns much slower than its unsegmented variant, since the exploration component now applies at the subpolicy level.

Figure 2: Visualizations for policies evaluated on OBP and Deezer datasets.

Additionally, we compare the time-series of simulated rewards across policies to see how some policies learn faster than others.

6 RESULTS

6.1 ZOZO Results

We show only results from the Self-Normalized Inverse Probability Weighting (SNIPW) estimator in Figure 2a and Figure 2b. The policies that leaned toward exploitation tended to perform slightly better than their counterparts that leaned towards exploration. Both Explore-Then-Commit policies greatly surpassed the baseline policy, outperforming it by around 50%. Looking at Figure 2b, it seems that `etc_explore` ended up performing only slightly worse than `etc_exploit`. However, its large gains in cumulative rewards towards the end signal that that policy could potentially end up performing the best given more data and time. The relative policy value bar charts in Figure 2b show that although there is a distinguishable difference in policy performance, the off-policy estimation suffers from high variance. As an example, the `e_greedy_exploit` policy is estimated to perform anywhere from slightly above the baseline to almost double the baseline.

The most surprising result is that the linear contextual policies (`lin_egreedy_exploit` and `lin_egreedy_explore`) did not beat the random baseline. This is likely due to the low power of the regression techniques' ability to predict rewards, and it may need more data or more complex models to robustly learn actions from user contexts.

6.1.1 Benchmarking Results. Our results showed that both Inverse Probability Weighting (IPW) and SNIPW had the lowest errors, matching the results demonstrated in literature [1]. While outside the scope of our project, there remains ample research opportunity in policy regression model selection for providing the strongest estimator performance, and the tools developed in the course of this project provide ample support and framework for exploring the benefits of a well-tuned estimator.

6.2 Deezer Results

At a high level, our results show that Thompson Sampling methods consistently performed best, trailed closely by explore-then-commit

with a low exploration requirement. These results successfully reproduce the results of the Deezer authors' own experiments. KL-UCB also performed well, with relative policy value nearly eight times the baseline. The rolling mean rewards of all policies are illustrated in Figures 2c and 2d. The plot shows how, for example explore-then-commit with low exploration (`etc_exploit`) very quickly achieves strong performance whereas ETC with high required exploration (`etc_explore`) doesn't start performing well until it hits its exploration threshold more than halfway through the simulation.

Full context-free policies also typically outperformed their segmented variants, with the exception of the ϵ-greedy policies where segmentation improved performance. As the data had 100 user segments, each subpolicy observes substantially fewer rewards than the single unsegmented policy does. This is particularly damaging for policies like `seg_etc_explore`, which must select each arm at least 100 times before it can commit to using the most successful one. At the segmented scale, this policy is stuck exploring and never enters the exploitation stage, and thus acts and scores similarly to the Uniform Random baseline. This result is in line with Deezer's observation that segmented policies outperformed fully personalized ones, suggesting that leveraging commonalities between users helps policies learn faster. This might also hint to a a strong popularity bias in Deezer's dataset, where a small subset of policies perform very well for most users.

The contextual methods also performed relatively well, with `lin_egreedy_exploit` achieving six times the baseline and `lin_egreedy_explore` over eight times the baseline. This result is somewhat unremarkable: the simulation itself relies on a linear (logistic) model of click probabilities, so the linear models simply need to learn an approximation of this ground truth (in fact, a logistic ϵ-greedy policy could potentially learn the supplied action contexts as its weights, effectively reproducing the process by which the action contexts were originally learned).

It is worth noting that an important underlying assumption of Deezer's dataset drives these results: the restriction that user-playlist click probabilities are static and do not drift over time or are influenced by user history. This can give an upper-hand to exploitative or deterministic policies like `etc_exploit` and KL-UCB, which ignore user preference change and may simply find popular actions that will do well on average from that point on. The simulation is then not able to test the degree to which the policies can adapt to time-varying user preferences and is slightly biased toward exploitation.

7 DISCUSSION/CONCLUSION

By extending the experimental methods and dataset capabilities of the OBP, we ran additional experiments to shed light on the benefits and disadvantages of two policy evaluation methods.

The experimental results from the ZOZO dataset aligned with our expectations in terms of policy performance, but the results from Figure 2b demonstrate significant variance in policy value estimation, despite running on the entire dataset. Future work with datasets of even larger magnitude may improve the consistency of these estimators.

While Deezer's simulated approach appears to be free of the issue of high estimator variance, it suffers from weaker generalizability to practical use case scenarios. The synthetic data provided by this model comes from a finite set of existing user data, so modeling user's evolving interests and changes is harder to predict and update and make the results of our experiments less interpretable. The restriction that user preferences are fixed over time is a large limitation and benefits exploitative models. However, the simulated data can be scaled up to provide massive amounts of data with many actions, providing an experimental framework that can greatly benefit from exploration of segmented policies. This is supported by strong policy performance when using Deezer's released users clusters.

In addition to the results demonstrated by our experiments, new datasets and policy designs can be easily integrated into the experimental framework built in this project. This project successfully compared offline policy learning using both off-policy evaluation and simulation. In extending OBP's robust and powerful experimental framework and adding bindings for a very different dataset to the library, we have further developed the ability to evaluate bandit methods using consistent procedures and standardized, publicly-available data.

CONTRIBUTION BREAKDOWN

- A. Claffey - Experiment design and policy selection, policy implementation, and estimator benchmarking.
- C. Ick - Experiment scripting and configuration, HPC setup and deployment.
- J. Ma - Dataset research and literature review, visualization research and implementation.
- D. Turkel - Deezer dataset loader, online learning simulation, and experiment class implementation.

ACKNOWLEDGEMENTS

We would like to thank Guillaume Salha from Deezer for his help clarifying implementation and rationale behind aspects of his paper.

REFERENCES

[1] Yuta Saito, Shunsuke Aihara, Megumi Matsutani, and Yusuke Narita. 2020. Large-scale open dataset, pipeline, and benchmark for bandit algorithms. *arXiv:2008.07146 [cs, stat]*, (November 15, 2020). arXiv: 2008.07146. Retrieved 11/20/2020 from http://arxiv.org/abs/2008.07146.

[2] Walid Bendada, Guillaume Salha, and Théo Bontempelli. 2020. Carousel personalization in music streaming apps with contextual bandits. In *Fourteenth ACM Conference on Recommender Systems*. ACM, (September 22, 2020), 420–425. DOI: 10.1145/3383313.3412217. arXiv: 2009.06546. Retrieved 10/23/2020 from http://arxiv.org/abs/2009.06546.

[3] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th international conference on World wide web - WWW '10*, 661. DOI: 10.1145/1772690.1772758. arXiv: 1003.0146. Retrieved 10/22/2020 from http://arxiv.org/abs/1003.0146.

[4] Lihong Li, Wei Chu, and John Langford. 2010. An unbiased, data-driven, offline evaluation method of contextual bandit algorithms. *CoRR*, abs/1003.5956. arXiv: 1003.5956. http://arxiv.org/abs/1003.5956.

[5] James M. Robins, Andrea Rotnitzky, and Lue Ping Zhao. 1994. Estimation of regression coefficients when some regressors are not always observed. *Journal of the American Statistical Association*, 89, 427, 846–866. ISSN: 01621459. http://www.jstor.org/stable/2290910.

[6] Adith Swaminathan and Thorsten Joachims. 2015. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*. C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors. Volume 28. Curran Associates, Inc., 3231–3239.

[7] Alina Beygelzimer and John Langford. 2008. The offset tree for learning with partial labels. *CoRR*, abs/0812.4044. arXiv: 0812.4044. http://arxiv.org/abs/0812.4044.

[8] Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. 2014. Doubly robust policy evaluation and optimization. *Statistical Science*, 29, 4, (November 2014), 485–511. ISSN: 0883-4237. DOI: 10.1214/14-sts500. http://dx.doi.org/10.1214/14-STS500.

[9] William R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25, 3-4, (December 1933), 285–294. DOI: 10.1093/biomet/25.3-4.285. https://doi.org/10.1093/biomet/25.3-4.285.

[10] D. Bouneffouf. 2016. Finite-time analysis of the multi-armed bandit problem with known trend. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, 2543–2549. DOI: 10.1109/CEC.2016.7744106.

[11] Olivier Cappé, Aurélien Garivier, Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. 2013. Kullback–leibler upper confidence bounds for optimal sequential allocation. *The Annals of Statistics*, 41, 3, (June 2013), 1516–1541. DOI: 10.1214/13-aos1119. http://dx.doi.org/10.1214/13-AOS1119.

[12] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: a highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors. Volume 30. Curran Associates, Inc., 3146–3154. https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

TABLES

Table 1: Bandit Policy Hyperparameters

Policy name	Base Policy	Parameters
`random`	Random	(none)
`egreedy_exploit`	ϵ-greedy	ϵ = 0.01
`egreedy_explore`	ϵ-greedy	ϵ = 0.1
`ts_naive`	Bernoulli Thompson Sampling	α = 1, β = 1
`ts_pessimistic`	Bernoulli Thompson Sampling	α = 1, β = 100
`etc_exploit`	Explore-Then-Commit	`min_n` = 20
`etc_explore`	Explore-Then-Commit	`min_n` = 100
`kl_ucb`	KL-UCB	(none)
`seg_egreedy_exploit`	Segmented ϵ-greedy	ϵ = 0.01
`seg_egreedy_explore`	Segmented ϵ-greedy	ϵ = 0.1
`seg_ts_naive`	Segmented Bernoulli Thompson Sampling	α = 1, β = 1
`seg_ts_pessimistic`	Segmented Bernoulli Thompson Sampling	α = 1, β = 100
`seg_etc_exploit`	Segmented Explore-Then-Commit	`min_n` = 20
`seg_etc_explore`	Segmented Explore-Then-Commit	`min_n` = 100
`seg_kl_ucb`	Segmented KL-UCB	(none)
`lin_egreedy_exploit`	Linear ϵ-greedy	ϵ = 0.01
`lin_egreedy_explore`	Linear ϵ-greedy	ϵ = 0.1

Table 2: Estimator Benchmarking, Random → Bernoulli TS, in-sample.

Estimator	Error
dm	0.190±0.031
ipw	0.055±0.041
snipw	0.056±0.040
dr	0.058±0.041
sndr	0.058±0.041
switch-dr (τ=5)	0.136±0.031
switch-dr (τ=10)	0.103±0.033
switch-dr (τ=50)	0.058±0.041
switch-dr (τ=100)	0.058±0.041
switch-dr (τ=500)	0.058±0.041
switch-dr (τ=1000)	0.058±0.041
dr-os (λ=5)	0.162±0.028
dr-os (λ=10)	0.151±0.028
dr-os (λ=50)	0.120±0.028
dr-os (λ=100)	0.106±0.029
dr-os (λ=500)	0.079±0.033
dr-os (λ=1000)	0.071±0.036
mrdr	0.058±0.040

Estimators in **bold** performed the best (IPW and SNIPW). Run on 30 bootstrapped samples with a Random baseline policy on the *all* campaign. Direct method ("dm") and double robust ("dr") policies use `lightgbm` as prediction models.