Open Bandit Dataset and Pipeline: Towards Realistic and Reproducible Off-Policy Evaluation

Yuta Saito[*]
Cornell University
ys552@cornell.edu

Shunsuke Aihara
ZOZO Research
shunsuke.aihara@zozo.com

Megumi Matsutani
ZOZO Research
megumi.matsutani@zozo.com

Yusuke Narita
Yale University
yusuke.narita@yale.edu

Abstract

Off-policy evaluation (OPE) aims to estimate the performance of hypothetical policies using data generated by a different policy. Because of its huge potential impact in practice, there has been growing research interest in this field. There is, however, no real-world public dataset that enables the evaluation of OPE, making its experimental studies unrealistic and irreproducible. With the goal of enabling realistic and reproducible OPE research, we present *Open Bandit Dataset*, a public logged bandit dataset collected on a large-scale fashion e-commerce platform, ZOZOTOWN. Our dataset is unique in that it contains a set of *multiple* logged bandit datasets collected by running different policies on the same platform. This enables experimental comparisons of different OPE estimators for the first time. We also develop Python software called *Open Bandit Pipeline* to streamline and standardize the implementation of batch bandit algorithms and OPE. Our open data and software will contribute to fair and transparent OPE research and help the community identify fruitful research directions. We provide extensive benchmark experiments of existing OPE estimators using our dataset and software. The results open up essential challenges and new avenues for future OPE research.

1 Introduction

Interactive bandit systems such as personalized medicine and recommendation platforms produce log data valuable for evaluating and redesigning the system. For example, the logs of a news recommendation system records which news article was presented and whether the user read it, giving the system designer a chance to make its recommendations more relevant. Exploiting log bandit data is, however, more difficult than conventional supervised machine learning, as the result is only observed for the action chosen by the system, but not for all the other actions that the system could have taken. The logs are also biased in that they overrepresent the actions favored by the system. A potential solution to this problem is an A/B test, which compares the performance of counterfactual systems in an online environment. However, A/B testing counterfactual systems is often difficult because deploying a new policy is time- and money-consuming and entails the risk of failure. This leads us to the problem of *off-policy evaluation* (OPE), which aims to estimate the performance of a counterfactual (or evaluation) policy using only log data collected by a past (or behavior) policy. OPE allows us to evaluate the performance of candidate policies without implementing A/B tests and contributes to safe policy improvement. Its applications range from

---

contextual bandits [4, 25, 26, 32, 37, 38, 40, 43, 44, 45, 51] and reinforcement learning in the web industry [10, 17, 20, 28, 47, 48, 52] to other social domains such as healthcare [31] and education [29].

Issues with current experimental procedures. Although the research community has produced theoretical breakthroughs over the past decade, the experimental evaluation of OPE remains primitive. Specifically, it lacks a public benchmark dataset for comparing the performance of different estimators. Researchers often validate their estimators using synthetic simulation environments [20, 28, 50, 52]. A version of the synthetic approach is to modify multiclass classification datasets and treat supervised machine learning methods as bandit policies to evaluate the estimation accuracy of OPE estimators [8, 10, 18, 49, 51]. An obvious problem with these studies is that they are ***unrealistic*** because there is no guarantee that their simulation environment is similar to real-world settings. To solve this issue, some previous studies use proprietary real-world datasets [12, 14, 32, 33]. Because these datasets are not public, however, the results are ***irreproducible***, and it remains challenging to compare existing estimators with new ideas in a fair manner. The lack of a public real-world benchmark makes it hard to identify critical research challenges and the bottleneck of the literature. This contrasts with other domains of machine learning, where large-scale open datasets, such as the ImageNet dataset [7], have been pivotal in driving objective progress [9, 13, 15, 16].

Contributions. Our goal is to implement and evaluate OPE in ***realistic and reproducible*** ways. To this end, we release *Open Bandit Dataset*, a set of logged bandit datasets collected on the ZOZOTOWN platform.[2] ZOZOTOWN is the largest fashion e-commerce platform in Japan. When the platform produced the data, it used Bernoulli Thompson Sampling (Bernoulli TS) and uniform random (Random) policies to recommend fashion items to users. The dataset thus includes a set of *two* logged bandit datasets collected during an A/B test of these bandit policies. Having multiple (at least two) log datasets is essential because it enables the evaluation of the estimation accuracy of OPE estimators as we describe in detail in Section 5.

In addition to the dataset, we implement *Open Bandit Pipeline*, an open-source Python software including a series of modules for implementing dataset preprocessing, policy learning methods, and OPE estimators. Our software provides a complete, standardized experimental procedure for OPE research, ensuring that performance comparisons are fair, transparent, and reproducible. It also enables fast and accurate OPE implementation through a single unified interface, simplifying the practical use of OPE.

Using our dataset and software, we perform comprehensive benchmark experiments on existing estimators. We implement this OPE experiment by using the log data of one of the policies (e.g., Bernoulli TS) to estimate the policy value of the other policy (e.g., Random) with each OPE estimator. We then assess the accuracy of the estimator by comparing its estimation with the policy value obtained from the data in an *on-policy* manner. Through the experiments, we showcase the utility of Open Bandit Dataset and Pipeline by using them to analyze the challenges that we face when we try applying OPE to real-world scenarios.

Our key contributions are summarized as follows:

- **Public Dataset**: We build and release *Open Bandit Dataset*, a set of *two* logged bandit data to enable realistic and reproducible research on OPE.
- **Software Implementation**: We implement *Open Bandit Pipeline*, an open-source Python software that helps practitioners utilize OPE to evaluate their bandit systems. It also helps researchers compare different OPE estimators in a standardized manner.
- **Benchmark Experiment**: We perform comprehensive benchmark experiments on existing OPE estimators and indicate critical challenges in future research.

2 Off-Policy Evaluation

2.1 Setup

We consider a general contextual bandit setting. Let $r \in [0, r_{\max}]$ denote a reward variable (e.g., whether a fashion item as an action results in a click). We let $x \in \mathcal{X}$ be a context vector (e.g., the

---

[2]https://corp.zozo.com/en/service/

user's demographic profile) that the decision maker observes when picking an action. We also let $a \in \mathcal{A}$ be a discrete action such as a fashion item in a recommender system. Rewards and contexts are sampled from unknown distributions $p(r \mid x, a)$ and $p(x)$, respectively. We call a function $\pi : \mathcal{X} \to \Delta(\mathcal{A})$ a *policy*. It maps each context $x \in \mathcal{X}$ into a distribution over actions, where $\pi(a \mid x)$ is the probability of taking action a given context x. We describe some examples of such decision making policies in Appendix A.

Let $\mathcal{D} := \{(x_i, a_i, r_i)\}_{i=1}^n$ be logged bandit dataset with n observations. a_i is a discrete variable indicating which action in $\mathcal{A}$ is chosen for i. r_i and x_i denote the reward and the context observed for each data, respectively. We assume that the logged dataset is generated by *behavior policy* π_b as

$$\{(x_i, a_i, r_i)\}_{i=1}^n \; \sim \; \prod_{i=1}^n p(x_i)\pi_b(a_i \mid x_i)p(r_i \mid x_i, a_i),$$

where each triplet is sampled independently from the product distribution. We sometimes use $\mathbb{E}_n[f] := n^{-1} \sum_{(x_i, a_i, r_i) \in \mathcal{D}} f(x_i, a_i, r_i)$ to denote the empirical expectation over n observations in $\mathcal{D}$. We also use $q(x, a) := \mathbb{E}_{r \sim p(r|x,a)}[r \mid x, a]$ and $g(x, \pi) := \mathbb{E}_{a \sim \pi(a|x)}[g(x, a) \mid x]$ to define estimators.

2.2 Estimation Target and Estimators

We are interested in using logged bandit data to estimate the following *policy value* of any given *evaluation policy* π_e, which might be different from π_b:

$$V(\pi_e) := \mathbb{E}_{(x,a,r) \sim p(x)\pi_e(a|x)p(r|x,a)}[r].$$

An OPE estimator $\hat{V}$ estimates $V(\pi_e)$ using only $\mathcal{D}$ as $V(\pi_e) \approx \hat{V}(\pi_e; \mathcal{D})$. We define three standard estimators in the following.[3]

Direct Method (DM). DM [2] first estimates q using a supervised machine learning model, such as random forest or ridge regression. It then plugs it in to estimate the policy value as

$$\hat{V}_{\mathrm{DM}}(\pi_e; \mathcal{D}, \hat{q}) := \mathbb{E}_n[\hat{q}(x_i, \pi_e)],$$

where $\hat{q}(x, a)$ is a reward estimator. If $\hat{q}(x, a)$ is accurate, DM also estimates the policy value accurately. If $\hat{q}(x, a)$ is inaccurate, however, the final estimator is no longer consistent. The model misspecification issue is problematic because the extent of misspecification cannot be easily quantified from data [10, 50].

Inverse Probability Weighting (IPW). To alleviate the issue with DM, IPW is often used [35, 40]. This estimator weighs the observed rewards by the importance weights as

$$\hat{V}_{\mathrm{IPW}}(\pi_e; \mathcal{D}) := \mathbb{E}_n[w(x_i, a_i)r_i],$$

where $w(x, a) := \pi_e(a \mid x)/\pi_b(a \mid x)$. When the behavior policy is known, IPW is unbiased and consistent. However, it can have a large variance, especially when the evaluation policy deviates significantly from the behavior policy.

Doubly Robust (DR). DR [8] combines DM and IPW as follows.

$$\hat{V}_{\mathrm{DR}}(\pi_e; \mathcal{D}, \hat{q}) := \mathbb{E}_n[\hat{q}(x_i, \pi_e) + w(x_i, a_i)(r_i - \hat{q}(x_i, a_i))].$$

DR mimics IPW to use a weighted version of rewards, but it also uses $\hat{q}$ as a control variate to decrease the variance. It preserves the consistency of IPW if either the importance weight or the reward estimator is consistent (a property called *double robustness*). Moreover, DR is *semiparametric efficient* when the reward estimator is correctly specified [32]. However, when it is misspecified, this estimator can have a larger asymptotic mean-squared-error than that of IPW [20].

3 Open-Source Dataset and Software

Motivated by the paucity of real-world datasets and implementations enabling the evaluation of OPE, we release the following open-source dataset and software.

---

[3]We define some other advanced estimators in Appendix B.

Figure 1: Fashion items as actions displayed in ZOZOTOWN recommendation interface.

Table 1: Statistics of Open Bandit Dataset

Campaigns	Data Collection Policies	#Data	#Items	#Dim	CTR ($V(\pi)$) $\pm$95% CI	Relative-CTR
ALL	**Random**	1,374,327	80	84	0.35% $\pm$0.010	1.00
	Bernoulli TS	12,168,084			0.50% $\pm$0.004	1.43
Men's	**Random**	452,949	34	38	0.51% $\pm$0.021	1.48
	Bernoulli TS	4,077,727			0.67% $\pm$0.008	1.94
Women's	**Random**	864,585	46	50	0.48% $\pm$0.014	1.39
	Bernoulli TS	7,765,497			0.64% $\pm$0.056	1.84

Note: Bernoulli TS stands for Bernoulli Thompson Sampling. **#Data** is the total number of user impressions observed during the 7-day experiment. **#Items** is the total number of items having a non-zero probability of being recommended by each policy. **#Dim** is the number of dimensions of the raw context vectors. **CTR** is the percentage of a click being observed in the log data, and this is the performance of the data collection policies in each campaign. 95% confidence interval (CI) of CTR is calculated based on the normal approximation of the Bernoulli sampling. **Relative-CTR** is CTR relative to that of the Random policy for the "ALL" campaign.

Open Bandit Dataset. Our open-source dataset is a set of *two* logged bandit datasets provided by ZOZO, Inc., the largest fashion e-commerce company in Japan. The company uses multi-armed bandit algorithms to recommend fashion items to users in their fashion e-commerce platform called ZOZOTOWN. We present examples of displayed fashion items in Figure 1. We collected the data in a 7-day experiment in late November 2019 on three "campaigns," corresponding to "ALL", "Men's", and "Women's" items, respectively. Each campaign randomly uses either the Random policy or the Bernoulli TS policy for each user impression. These policies select three of the candidate fashion items for each user. Figure 1 shows that there are three *positions* in our dataset. We assume that the reward (click indicator) depends only on the item and its position, which is a general assumption on the click generative process used in the web industry [27]. Under this assumption, we can apply the OPE setup in Section 2 to our dataset. We provide some statistics of the dataset in Table 1. The dataset is large and contains many millions of recommendation instances. Each row of the data has feature vectors such as age, gender, and past click history of the users. These feature vectors are hashed, thus the dataset does not contain any personally identifiable information. Moreover, the dataset includes some item-related features such as price, fashion brand, and item categories. It also includes the probability that item a is displayed at each position by the data collection policies. This probability is used to calculate the importance weights.[4] We share the full version of our dataset at `https://research.zozo.com/data.html`.[5] Small-sized example data are also available at `https://github.com/st-tech/zr-obp/tree/master/obd`.

To our knowledge, our open-source dataset is the first to include logged bandit datasets collected by running *multiple* (at least two) different policies and the exact policy implementations used in real production, enabling ***realistic and reproducible evaluation of OPE*** for the first time. Indeed, Open Bandit Dataset is already actively used by multiple research papers to benchmark new bandit algorithms and OPE estimators [6, 22, 23, 39, 46].

---

[4]We computed the action choice probabilities by Monte Carlo simulations based on the policy parameters (e.g., parameters of the beta distribution used by Bernoulli TS) used during the data collection process.

[5]The dataset is licensed under CC BY 4.0.

4

Open Bandit Pipeline. To facilitate the use of OPE in practice and standardize its experimental procedures, we implement a Python software called *Open Bandit Pipeline*.[6] Our software contains the following main modules:

- The **dataset** module provides a data loader to preprocess Open Bandit Dataset and tools to generate synthetic bandit datasets. It also implements functions to handle multiclass classification datasets as bandit data, which is useful when we conduct OPE experiments in research papers.

- The **policy** module implements several online bandit algorithms and off-policy learning methods such as a neural network-based off-policy learning method. This module also implements interfaces that allow practitioners to easily evaluate their own policies in their business using OPE.

- The **ope** module implements several existing OPE estimators including DM, IPW, DR, and some advanced ones such as Switch [51], More Robust Doubly Robust (MRDR) [10], and DR with Optimistic Shrinkage (DRos) [41]. This module also provides generic abstract interfaces to support custom implementations so that researchers can evaluate their own estimators easily.

Appendix E describes how the software facilitates the evaluation of OPE and bandit algorithms. We also prepare plenty of tutorial contents at `https://github.com/st-tech/zr-obp/tree/master/examples/quickstart` to help users grasp the usage of the software easily. We also provide the thorough documentation at `https://zr-obp.readthedocs.io/en/latest/`.

Every core function of the software is tested and thus are well maintained.[7] It currently has five core contributors.[8] The active development and maintenance will continue over a long period. Users can follow our progress at the following mailing list: `https://groups.google.com/g/open-bandit-project`.

We believe that the software allows researchers to focus on building their OPE estimator and to easily compare it with other methods in a standardized manner. It will also help practitioners implement cutting-edge estimators in their applications and improve their decision making systems.

4 Related Work

Here, we summarize the existing related work and resources, and clarify the advantages of ours.

Related Datasets. Our dataset is closely related to those of [24] and [25]. Lefortier et al. [24] introduce a large-scale logged bandit data (Criteo Data[9]) from a leading company in display advertising, Criteo. The dataset contains context vectors of user impressions, advertisements (ads) as actions, and click indicators as rewards. It also provides the ex-ante probability of each ad being selected by the behavior policy. Therefore, this dataset can be used to compare different off-policy *learning* methods, which aim to learn a new policy using only logged bandit data. In contrast, Li et al. [25] introduce a dataset (Yahoo! R6A&B[10]) collected on a news recommendation interface of the Yahoo! Today Module. The dataset contains context vectors of user impressions, presented news as actions, and click indicators as rewards. The dataset was collected by running a uniform random policy on the news recommendation platform, allowing researchers to evaluate their (online or offline) bandit algorithms.

However, the existing datasets have several limitations, which we overcome as follows:

- The existing datasets include only a single logged bandit dataset collected by running only a single policy. Moreover, the previous datasets do not provide the implementation to replicate the policies used during data collection. As a result, these datasets cannot be used for

---

[6]https://github.com/st-tech/zr-obp

[7]https://github.com/st-tech/zr-obp/tree/master/tests

[8]https://github.com/st-tech/zr-obp/graphs/contributors

[9]https://www.cs.cornell.edu/ādith/Criteo/

[10]`https://webscope.sandbox.yahoo.com/catalog.php?datatype=r`

Table 2: Comparison of currently available bandit datasets

	Criteo Data [24]	Yahoo! R6A&B [25]	Open Bandit Dataset (ours)
Domain	Display Advertising	News Recommendation	Fashion E-Commerce
Dataset Size	≈ 103M	≈ 40M	≈ 26M
#Data Collection Policies	1	1	**2**
Uniform Random Data	✗	✔	✔
Data Collection Policy Code	✗	✗	✔
Evaluation of Bandit Algorithms	✔	✔	✔
Evaluation of OPE	✗	✗	✔
Pipeline Implementation	✗	✗	✔

Note: **Dataset Size** is the total number of samples included in the whole dataset. **#Data Collection Policies** is the number of policies that were used to collect the data. **Uniform Random Data** indicates whether the dataset contains a subset of data generated by the uniform random policy. **Data Collection Policy Code** indicates whether the code to replicate data collection policies is publicized. **Evaluation of Bandit Algorithms** indicates whether it is possible to use the data to evaluate bandit algorithms. **Evaluation of OPE** indicates whether it is possible to use the dataset to evaluate OPE estimators. **Pipeline Implementation** indicates whether a pipeline tool to handle the dataset is available.

Table 3: Comparison of currently available packages of bandit algorithms and OPE

	Vowpal Wabbit [3]	contextualbandits [5]	RecoGym [36]	Open Bandit Pipeline (ours)
Synthetic Data Generator	✗	✗	✔	✔
Classification Data Handler	✗	✗	✗	✔
Support for Real-World Data	✗	✗	✗	✔
Bandit Algorithms	✔	✔	✔	✔
Basic OPE Estimators	✔	✔	✗	✔
Advanced OPE Estimators	✗	✗	✗	✔
Evaluation of OPE	✗	✗	✗	✔

Note: **Synthetic Data Generator** indicates whether it is possible to generate synthetic bandit data with the package. **Classification Data Handler** indicates whether it is possible to transform multiclass classification data to bandit data with the package. **Support for Real-World Data** indicates whether it is possible to handle real-world bandit data with the package. **Bandit Algorithms** indicates whether the package includes implementations of online and offline bandit algorithms. **Basic OPE Estimators** indicates whether the package includes implementations of *basic* OPE estimators such as DM, IPW, and DR described in Section 2. **Advanced OPE Estimators** indicates whether the package includes implementations of *advanced* OPE estimators such as Switch and More Robust Doubly Robust described in Appendix B. **Evaluation of OPE** indicates whether it is possible to evaluate the accuracy of OPE estimators with the package.

the comparison of different OPE estimators, although they can be used to evaluate policy *learning* methods.

→ In contrast, we provide the code to replicate the data collection policies in our software, which allows researchers to rerun the same policies on the log data. Without the code of the exact algorithms, we could not implement the evaluation of OPE experiments. Therefore, the code and algorithm release is an essential component of our open-source. Moreover, our dataset consists of a set of *two* different logged bandit datasets generated by running two different policies on the same platform. It enables the comparison of different OPE estimators, as we show in Section 5.

- The existing datasets do not provide a tool to handle their data. Researchers have to reimplement the experimental environment by themselves before implementing their own OPE estimators. This can lead to inconsistent experimental conditions across different studies, potentially causing reproducibility issues.

  → We implement Open Bandit Pipeline to simplify and standardize the experimental processing of bandit algorithms and OPE. This tool thus contributes to the reproducible and transparent use of our dataset.

Table 2 summarizes the key differences between our dataset and the existing ones.

Related Packages. There are several existing packages related to Open Bandit Pipeline. *Vowpal Wabbit*[11] is a library for fast machine learning, online learning, contextual bandits, and reinforcement learning [3]. It handles learning problems with any number of sparse features, achieving great scaling. The *contextualbandits* package[12] contains implementations of several contextual bandit algorithms [5]. It aims to provide an easy procedure to compare bandit algorithms to reproduce research papers that do not provide easily available implementations. There is also *RecoGym*[13], which focuses on providing simulation bandit environments imitating the e-commerce recommendation setting [36].

However, the following features differentiate our software from the previous ones:

- The previous packages focus on implementing and comparing online bandit algorithms or off-policy learning methods. However, they ***cannot*** be used to implement several advanced OPE estimators. They also do not help conduct benchmark experiments of OPE estimators.

  → Our software implements a wide variety of OPE estimators, including advanced ones such as Switch, MRDR, and DRos. In addition, Open Bandit Pipeline provides estimators that can address continuous actions or combinatorial actions, which cannot be handled by other packages. It also provides flexible interfaces for implementing new OPE estimators, allowing researchers to plug in their own estimators and compare them with existing estimators easily.

- The previous packages accept their own interface and data formats. Thus, they are not user-friendly.

  → Our software follows the prevalent *scikit-learn* style interface and provides sufficient example codes at https://github.com/st-tech/zr-obp/tree/master/examples so that anyone, including practitioners and students, can follow the usage.

- The previous packages cannot handle real-world bandit datasets.

  → Our software comes with Open Bandit Dataset and includes the **dataset module**. This facilitates the evaluation of bandit algorithms and OPE estimators using real-world data.

Table 3 summarizes the key differences between our software and the existing ones.

Related Benchmarks. There are several studies conducting benchmark experiments on OPE estimators. Fu et al. [11] provide the Deep Off-Policy Evaluation (DOPE) benchmark, which is designed to evaluate the performance of OPE estimators on several control tasks. The notable contribution of DOPE is that it evaluates the OPE performance across a range of evaluation policies with different policy values and measures performance on ranking and selection as well as policy evaluation. Evaluating the ranking and policy selection performance is challenging with real-world bandit data due to the difficulty of deploying many policies during data collection. Voloshin et al. [50] provide a benchmark study on a variety of tasks ranging from tabular problems to image-based tasks in Atari. A wide variety of OPE estimators are compared in the benchmark, including recent variants of DR such as MRDR and Self-Normalized DR (SNDR), producing a holistic summary of the challenges one should address in OPE applications.

Our work differentiates itself from these previous benchmark studies in several ways. First, our benchmark and implementation cover relevant methods that are not included in the previous benchmarks. For example, we evaluate some advanced estimators such as Switch Doubly Robust (Switch-DR), and DRos, which are not compared in DOPE. Moreover, we evaluate how a recent hyperparameter tuning method proposed in [41] works with real-world bandit data. Tuning hyperparameters of OPE estimators is a critical component in OPE application, as it can greatly affect the OPE performance. However, the previous benchmarks do not evaluate the performance of the tuning method. Moreover, we release the real-world bandit dataset that allows benchmarking of OPE estimators in a realistic scenario. Our public dataset makes it possible to identify what matters in applying OPE to real-world scenarios. This is in contrast to the previous benchmarks using only synthetic environments. Indeed, we found some critical bottlenecks, which have not yet been pointed out in the literature. Specifically, we found that it is necessary to develop a reliable method to choose and tune OPE estimators in a data-driven manner (discussed in Sections 5 and 6). We would also emphasize that the previous

---

[11]https://github.com/VowpalWabbit/vowpal_wabbit

[12]https://github.com/david-cortes/contextualbandits

[13]https://github.com/criteo-research/reco-gym

benchmarks focus on conducting comprehensive empirical studies. They do not provide implementations that allow researchers to add their own estimators to the benchmark, generate synthetic data, and handling real bandit data.

5 Benchmark Experiments

We perform benchmark experiments of OPE estimators using Open Bandit Dataset and Pipeline. We first describe an experimental protocol to evaluate OPE estimators and use it to compare a wide variety of existing estimators. We then discuss the initial findings from the experiments. We share the code to replicate the benchmark experiments at https://github.com/st-tech/zr-obp/tree/master/benchmark/ope.

5.1 Experimental Protocol

We can empirically evaluate OPE estimators' performance by using two sources of logged bandit data collected by running two different policies. In the protocol, we regard one policy as behavior policy π_b and the other as evaluation policy π_e. We denote log data generated by π_b and π_e as $\mathcal{D}^{(b)} := \{(x_i^{(b)}, a_i^{(b)}, r_i^{(b)})\}_{i=1}^{n^{(b)}}$ and $\mathcal{D}^{(e)} := \{(x_i^{(e)}, a_i^{(e)}, r_i^{(e)})\}_{i=1}^{n^{(e)}}$. Then, by applying the following protocol to several different OPE estimators, we compare their estimation performance:

1. Estimate the policy value of π_e using $\mathcal{D}^{(b)}$ by OPE estimator $\hat{V}$. We represent a policy value estimated by $\hat{V}$ as $\hat{V}(\pi_e; \mathcal{D}^{(b)})$.

2. Evaluate the estimation accuracy of $\hat{V}$ using the following *squared error* (SE):

$$SE(\hat{V}; \mathcal{D}^{(b)}) := \left(\hat{V}(\pi_e; \mathcal{D}^{(b)}) - V_{\mathrm{on}}(\pi_e) \right)^2,$$

where $V_{\mathrm{on}}(\pi_e) := (1/n^{(e)}) \sum_{i=1}^{n^{(e)}} r_i^{(e)}$ is the Monte-Carlo estimate (on-policy estimate) of $V(\pi_e)$ based on $\mathcal{D}^{(e)}$.

3. Repeat the above process T times with different bootstrap samples and calculate the following *root mean-squared-error* (RMSE) as the estimators' performance measure.

$$RMSE(\hat{V}; \mathcal{D}^{(b)}) := \sqrt{\frac{1}{T} \sum_{t=1}^{T} SE(\hat{V}; \mathcal{D}_t^{(b,*)})},$$

where $\mathcal{D}_t^{(b,*)}$ is the t-th bootstrapped sample of $\mathcal{D}^{(b)}$.

Algorithm 1 in Appendix C describes the experimental protocol to evaluate OPE estimators in detail.

5.2 Compared Estimators

We compare the following OPE estimators: DM, IPW, Self-Normalized Inverse Probability Weighting (SNIPW), DR, SNDR, Switch-DR, and DRos. We tune the built-in hyperparameter of Switch-DR and DRos using a data-driven hyperparameter tuning method described in Su et al. [41]. The details of the above estimators and data-driven hyperparameter tuning method are given in Appendix B.

For estimators except for DM, we use the true action choice probability $\pi_b(a|x)$ contained in Open Bandit Dataset. For estimators except for IPW and SNIPW, we need to obtain a reward estimator $\hat{q}$. We do this by using gradient boosting[14] (implemented in *scikit-learn* [34]) and training it on $\mathcal{D}^{(b)}$. We also use cross-fitting [19, 33] to avoid substantial bias from overfitting when obtaining $\hat{q}$.

5.3 Results

The results of the benchmark experiments with $n = 300,000$ are given in Table 4. We describe **Bernoulli TS $\rightarrow$ Random** to represent the OPE situation where we use Bernoulli TS as π_b and Random as π_e. Please see Appendix C for additional results.

---

[14]Specifically, we use 'sklearn.ensemble.HistGradientBoostingClassifier(learning_rate=0.01, max_iter=100, max_depth=5, min_samples_leaf=10, random_state=12345)' to obtain $\hat{q}$.

Table 4: RMSE ($\times 10^3$) of OPE estimators (**Bernoulli TS $\rightarrow$ Random**)

OPE Estimators ($\hat{V}$)	ALL	Men's	Women's
IPW[1]	$0.493\ (1.560)^3$	$0.789\ (1.719)$	$0.776\ (1.382)^3$
SNIPW[2]	$0.507\ (1.602)^3$	$0.644\ (1.403)^{1/3}$	$0.804\ (1.433)^3$
DM[3]	**1.026 (3.244)**	$0.773\ (1.685)$	**0.816 (1.455)**
DR[4]	$0.482\ (1.526)^3$	$0.613\ (1.336)^{1/3}$	$0.803\ (1.430)^3$
SNDR[5]	$0.482\ (1.526)^3$	$0.659\ (1.436)^{1/3}$	$0.791\ (1.408)^3$
Switch-DR[6]	$0.482\ (1.526)^3$	$0.613\ (1.336)^{1/3}$	$0.803\ (1.430)^3$
DRos[7]	**$0.316\ (1.000)^{1/2/3/4/5/6}$**	**$0.459\ (1.000)^{1/2/3/4/5/6}$**	**$0.561\ (1.000)^{1/2/3/4/5/6}$**

Table 5: RMSE ($\times 10^3$) of DRos with different hyperparameter values (**ALL Campaign**)

λ of DRos	Random $\rightarrow$ Bernoulli TS	Bernoulli TS $\rightarrow$ Random
1	1.384 (2.906)	0.963 (3.920)
5	1.247 (2.619)	0.837 (3.407)
10	1.162 (2.439)	0.770 (3.135)
50	0.896 (1.881)	0.589 (2.398)
100	0.778 (1.634)	0.498 (2.029)
500	0.574 (1.206)	0.294 (1.199)
1,000	0.482 (1.106)	**0.245 (1.000)**
5,000	**0.476 (1.000)**	0.270 (1.101)
10,000	**0.476 (1.000)**	0.323 (1.315)
tuning	**0.476 (1.000)**	0.323 (1.315)

Note: Root mean squared errors (RMSEs) estimated with 200 different bootstrapped iterations are reported ($n = 300,000$). RMSEs normalized by the best (lowest) RMSE are reported in parentheses. $^{1/2/3/4/5/6/7}$ denote a significant difference compared to the indicated estimator (Wilcoxon rank-sum test, $p < 0.05$). The **red bold** is used when the best results outperform the second bests in a statistically significant level. The **blue bold** is used when the worst results underperform the second worsts in a statistically significant level. $\pi_b \rightarrow \pi_e$ represents the OPE situation where the estimators aim to estimate the policy value of π_e using logged bandit data collected by π_b.

Performance comparisons. Table 4 shows that DRos (with automatic hyperparameter tuning) performs best for the three campaigns, achieving about 30-60% more accurate OPE than the second-best estimators. We then evaluate several values for the hyperparameter λ of DRos. Table 5 shows the OPE performance (RMSE) of DRos with different values of λ ($\in \{1, 5, 10, 50, \ldots, 10000\}$). This table also includes the OPE performance of DRos with automatic hyperparameter tuning of Su et al. [41]. First, we observe that the choice of λ greatly affects the performance of DRos. Specifically, for **Random $\rightarrow$ Bernoulli TS**, a larger value of λ leads to a better OPE performance. In contrast, for **Bernoulli TS $\rightarrow$ Random**, $\lambda = 1,000$ is the best setting. Second, we observe that the automatic hyperparameter tuning procedure prefers a large value of λ. This means that the tuning procedure puts emphasis on the bias of the estimator, as a large value of λ leads to a low bias, but a high variance estimator. This strategy succeeds for **Random $\rightarrow$ Bernoulli TS**. However, we observe that there is room for improvement for **Bernoulli TS $\rightarrow$ Random** in terms of automatic hyperparameter tuning. This suggests opportunities for further investigations on the quality of automatic hyperparameter tuning for achieving more accurate OPE in practice.

OPE performance with different sample sizes. Next, we compare the *small*-sample setting ($n = 10,000$) and *large*-sample setting ($n = 300,000$) to evaluate how the OPE performance changes with the sample size. We observe in Table 6 that the estimators' performance can change significantly depending on the size of the logged bandit data. In particular, for the Men's and Women's campaigns, the most accurate estimator changes with the sample size. The table shows that DM outperforms the other estimators in the *small*-sample setting, while DRos is the best for the *large*-sample setting. These observations suggest that practitioners have to choose an appropriate OPE estimator carefully for their specific application. It is thus necessary to develop a reliable method to choose and tune OPE estimators in a data-driven manner. Specifically, in real applications, we have to tune the estimators'

Table 6: Comparison of *small*-sample and *large*-sample OPE performance (RMSE $\times 10^3$)

OPE Estimators	ALL		Men's		Women's	
	small-sample	*large*-sample	*small*-sample	*large*-sample	*small*-sample	*large*-sample
IPW[1]	1.899	0.493^{3}	3.683^{5}	0.789	**3.156**	0.776^{3}
SNIPW[2]	1.641	0.507^{3}	$3.661^{4/5/6}$	$0.644^{1/3}$	$3.038^{1/5}$	0.804^{3}
DM[3]	$0.797^{1/2/4/5/6}$	**1.026**	$\textbf{3.041}^{1/2/4/5/6/7}$	0.773	$\textbf{2.665}^{1/2/4/5/6/7}$	$\textbf{0.816}^{3}$
DR[4]	$1.203^{1/2}$	0.482^{3}	3.747	$0.613^{1/3}$	$3.055^{1/2}$	0.803^{3}
SNDR[5]	$1.159^{1/2}$	0.482^{3}	3.757	$0.659^{1/3}$	3.069^{1}	0.791^{3}
Switch-DR[6]	$1.203^{1/2}$	0.482^{3}	3.747	$0.613^{1/3}$	3.055^{5}	0.803^{3}
DRos[7]	$0.765^{1/2/4/5/6}$	$\textbf{0.316}^{1/2/3/4/5/6}$	3.727	$\textbf{0.459}^{1/2/3/4/5/6}$	$3.051^{1/5}$	$\textbf{0.561}^{1/2/3/4/5/6}$

Note: Root mean squared errors (RMSEs) estimated with 200 different bootstrapped iterations are reported (**Bernoulli TS $\rightarrow$ Random**). $n = 10,000$ for the *small*-sample setting, while $n = 300,000$ for the *large*-sample setting. $^{1/2/3/4/5/6/7}$ denote a significant difference compared to the indicated estimator (Wilcoxon rank-sum test, $p < 0.05$). The **red bold** is used when the best results outperform the second bests in a statistically significant level. The **blue bold** is used when the worst results underperform the second worsts in a statistically significant level.

hyperparameters or identify an accurate estimator without the ground-truth or on-policy policy value of the evaluation policy.

6 Conclusion, Limitations, and Future Work

To enable a realistic and reproducible evaluation of OPE, we presented Open Bandit Dataset, a set of logged bandit datasets collected on a fashion e-commerce platform. The dataset comes with Open Bandit Pipeline, Python software that makes it easy to evaluate and compare different OPE estimators. We aim to facilitate understanding of the empirical properties of OPE estimators and address experimental inconsistencies in the literature. We also perform extensive experiments on a variety of OPE estimators and analyze the effects of hyperparameter choice and sample size in a real-world setting. Our experiments highlight that an appropriate estimator can change depending on a problem setting such as the sample size. The results also suggest that there is room to improve the OPE performance in terms of automatic hyperparameter tuning and estimator selection. These observations call for a new estimator selection method and a hyperparameter tuning procedure to be developed.

A limitation is that we assume that the reward of an item at a position does not depend on other simultaneously presented items. This assumption might not hold, as an item's attractiveness can have a significant effect on the expected reward of another item in the same recommendation list [27]. To address more realistic situations, we have implemented some OPE estimators for the slate action setting [30, 45] in Open Bandit Pipeline. Comparing the standard OPE estimators and those for the slate action setting on our dataset is a valuable and interesting research direction.

Open Bandit Dataset is currently the only public dataset allowing OPE experiments. Therefore, it might lead to an overfitting issue. Moreover, Open Bandit Dataset includes only two policies. Here, we emphasize that there has never been any public real-world data that allow realistic and reproducible OPE research before. Our open-source project is an initial step towards this goal. Having many policies would be even more valuable, but releasing data with two different data collection policies is distinguishable enough from the prior work. We believe that our work will inspire other researchers and companies to create follow-up benchmark datasets to advance OPE research further.

References

[1] Aman Agarwal, Soumya Basu, Tobias Schnabel, and Thorsten Joachims. Effective evaluation using logged bandit feedback from multiple loggers. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 687–696, 2017.

[2] Alina Beygelzimer and John Langford. The offset tree for learning with partial labels. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 129–138, 2009.

[3] Alberto Bietti, Alekh Agarwal, and John Langford. A contextual bandit bake-off. *arXiv preprint arXiv:1802.04064*, 2018.

[4] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising. *Journal of Machine Learning Research*, 14(1):3207–3260, 2013.

[5] David Cortes. Adapting multi-armed bandits policies to contextual bandits scenarios. *arXiv preprint arXiv:1811.04383*, 2018.

[6] Jin Dai. *Offline Evaluation of Multi-Armed Bandit Algorithms Using Bootstrapped Replay on Expanded Data*. University of California, Los Angeles, 2021.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[8] Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Doubly Robust Policy Evaluation and Optimization. *Statistical Science*, 29:485–511, 2014.

[9] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[10] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *International Conference on Machine Learning*, volume 80, pages 1447–1456. PMLR, 2018.

[11] Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, et al. Benchmarks for deep off-policy evaluation. *arXiv preprint arXiv:2103.16596*, 2021.

[12] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206, 2018.

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[14] Alois Gruson, Praveen Chandar, Christophe Charbuillet, James McInerney, Samantha Hansen, Damien Tardieu, and Ben Carterette. Offline evaluation to make decisions about playlist recommendation algorithms. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 420–428, 2019.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[17] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, volume 48, pages 652–661. PMLR, 2016.

[18] Nathan Kallus, Yuta Saito, and Masatoshi Uehara. Optimal off-policy evaluation from multiple logging policies. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 5247–5256. PMLR, 2021.

[19] Nathan Kallus and Masatoshi Uehara. Efficiently breaking the curse of horizon: Double reinforcement learning in infinite-horizon processes. *arXiv preprint arXiv:1909.05850*, 2019.

[20] Nathan Kallus and Masatoshi Uehara. Intrinsically efficient, stable, and bounded off-policy evaluation for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 3325–3334, 2019.

[21] Nathan Kallus and Angela Zhou. Policy evaluation and optimization with continuous treatments. In *International Conference on Artificial Intelligence and Statistics*, pages 1243–1251. PMLR, 2018.

[22] Masahiro Kato, Kenshi Abe, Kaito Ariu, and Shota Yasui. A practical guide of off-policy evaluation for bandit problems. *arXiv preprint arXiv:2010.12470*, 2020.

[23] Junpei Komiyama, Edouard Fouché, and Junya Honda. Finite-time analysis of globally nonstationary multi-armed bandits. *arXiv preprint arXiv:2107.11419*, 2021.

[24] Damien Lefortier, Adith Swaminathan, Xiaotao Gu, Thorsten Joachims, and Maarten de Rijke. Large-scale validation of counterfactual learning methods: A test-bed. *arXiv preprint arXiv:1612.00367*, 2016.

[25] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A Contextual-bandit Approach to Personalized News Article Recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670. ACM, 2010.

[26] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, pages 297–306, 2011.

[27] Shuai Li, Yasin Abbasi-Yadkori, Branislav Kveton, S Muthukrishnan, Vishwa Vinay, and Zheng Wen. Offline evaluation of ranking policies with click models. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1685–1694, 2018.

[28] Yao Liu, Omer Gottesman, Aniruddh Raghu, Matthieu Komorowski, Aldo A Faisal, Finale Doshi-Velez, and Emma Brunskill. Representation balancing mdps for off-policy policy evaluation. In *Advances in Neural Information Processing Systems*, volume 31, pages 2644–2653, 2018.

[29] Travis Mandel, Yun-En Liu, Sergey Levine, Emma Brunskill, and Zoran Popovic. Offline policy evaluation across representations with applications to educational games. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1077–1084, 2014.

[30] James McInerney, Brian Brost, Praveen Chandar, Rishabh Mehrotra, and Benjamin Carterette. Counterfactual evaluation of slate recommendations with sequential reward interactions. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1779–1788, 2020.

[31] Susan A Murphy, Mark J van der Laan, James M Robins, and Conduct Problems Prevention Research Group. Marginal mean models for dynamic regimes. *Journal of the American Statistical Association*, 96(456):1410–1423, 2001.

[32] Yusuke Narita, Shota Yasui, and Kohei Yata. Efficient counterfactual learning from bandit feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4634–4641, 2019.

[33] Yusuke Narita, Shota Yasui, and Kohei Yata. Debiased off-policy evaluation for recommendation systems. In *Fifteenth ACM Conference on Recommender Systems*, RecSys '21, page 372–379, New York, NY, USA, 2021. Association for Computing Machinery.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[35] Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 759–766, 2000.

[36] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720*, 2018.

[37] Yuta Saito. Doubly robust estimator for ranking metrics with post-click conversions. In *14th ACM Conference on Recommender Systems*, pages 92–100, 2020.

[38] Yuta Saito and Thorsten Joachims. Counterfactual learning and evaluation for recommender systems: Foundations, implementations, and recent advances. In *Fifteenth ACM Conference on Recommender Systems*, RecSys '21, page 828–830, New York, NY, USA, 2021. Association for Computing Machinery.

[39] Yuta Saito, Takuma Udagawa, Haruka Kiyohara, Kazuki Mogi, Yusuke Narita, and Kei Tateno. Evaluating the robustness of off-policy evaluation. In *Fifteenth ACM Conference on Recommender Systems*, RecSys '21, page 114–123, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. Learning from Logged Implicit Exploration Data. In *Advances in Neural Information Processing Systems*, volume 23, pages 2217–2225, 2010.

[41] Yi Su, Maria Dimakopoulou, Akshay Krishnamurthy, and Miroslav Dudík. Doubly robust off-policy evaluation with shrinkage. In *International Conference on Machine Learning*, volume 119, pages 9167–9176. PMLR, 2020.

[42] Yi Su, Lequn Wang, Michele Santacatterina, and Thorsten Joachims. Cab: Continuous adaptive blending for policy evaluation and learning. In *International Conference on Machine Learning*, pages 6005–6014, 2019.

[43] Adith Swaminathan and Thorsten Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. *The Journal of Machine Learning Research*, 16(1):1731–1755, 2015.

[44] Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*, volume 28, pages 3231–3239, 2015.

[45] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. Off-policy Evaluation for Slate Recommendation. In *Advances in Neural Information Processing Systems*, volume 30, pages 3635–3645, 2017.

[46] Akira Tanimoto, Tomoya Sakai, Takashi Takenouchi, and Hisashi Kashima. Causal combinatorial factorization machines for set-wise recommendation. In *PAKDD (2)*, pages 498–509. Springer, 2021.

[47] Philip Thomas and Emma Brunskill. Data-efficient Off-policy Policy Evaluation for Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 2139–2148, 2016.

[48] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *Proceedings of the 32th International Conference on Machine Learning*, volume 37, pages 2380–2388, 2015.

[49] Nikos Vlassis, Aurelien Bibaut, Maria Dimakopoulou, and Tony Jebara. On the design of estimators for bandit off-policy evaluation. In *International Conference on Machine Learning*, pages 6468–6476, 2019.

[50] Cameron Voloshin, Hoang M Le, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*, 2019.

[51] Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudik. Optimal and Adaptive Off-policy Evaluation in Contextual Bandits. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3589–3597, 2017.

[52] Tengyang Xie, Yifei Ma, and Yu-Xiang Wang. Towards optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. In *Advances in Neural Information Processing Systems*, volume 32, pages 9665–9675, 2019.

Contents

A   Examples of Bandit Algorithms

Our setup allows for many popular multi-armed bandit algorithms and off-policy learning methods, as the following examples illustrate.

Example 1 (Random A/B testing). *We always choose each action uniformly at random, i.e., $\pi_{\text{Uniform}}(a \mid x) = |\mathcal{A}|^{-1}$ always holds for any given $a \in \mathcal{A}$ and $x \in \mathcal{X}$.*

Example 2 (Bernoulli Thompson Sampling). *We sample the potential reward $\tilde{r}(a)$ from the beta distribution $Beta(S_{ta} + \alpha, F_{ta} + \beta)$ for each action in $\mathcal{A}$, where $S_{ta} := \sum_{t'=1}^{t-1} r_{t'}$, $F_{ta} := (t-1) - S_{ta}$. (α, β) are the parameters of the prior Beta distribution. We then choose the action with the highest sampled potential reward, $a :\in \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \tilde{r}(a')$ (ties are broken arbitrarily). As a result, this algorithm chooses actions with the following probabilities:*

$$\pi_{\text{BernoulliTS}}(a \mid x) = \Pr\{a \in \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \tilde{r}(a')\}$$

for any given $a \in \mathcal{A}$ and $x \in \mathcal{X}$. When implementing the data collection experiment on the ZOZOTOWN platform, we modified TS to adjust to our top-3 recommendation setting shown in Figure 1. The modified TS selects three actions with the three highest sampled rewards which create a nonrepetitive set of item recommendations for each coming user.

Example 3 (IPW Learner). *When $\mathcal{D}$ is given, we can train a deterministic policy $\pi_{\text{det}} : \mathcal{X} \to \mathcal{A}$ by maximizing the IPW estimator as follows:*

$$\pi_{\text{det}}(x) \in \underset{\pi \in \Pi}{\operatorname{argmax}} \hat{V}_{\text{IPW}}(\pi; \mathcal{D})$$

$$= \underset{\pi \in \Pi}{\operatorname{argmax}} \mathbb{E}_{\mathcal{D}} \left[\frac{\mathbb{I}\{\pi(x_t) = a_t\}}{\pi_b(a_t \mid x_t)} r_t \right]$$

$$= \underset{\pi \in \Pi}{\operatorname{argmin}} \mathbb{E}_{\mathcal{D}} \left[\frac{r_i}{\pi_b(a_t \mid x_t)} \mathbb{I}\{\pi(x_t) \neq a_t\} \right]$$

, which is equivalent to the cost-sensitive classification problem that can be solved with an arbitrary machine learning classifier.

B OPE Estimators and Related Techniques

Here we define some advanced OPE estimators compared in Section 5.

Self-Normalized Estimators. Self-Normalized Inverse Probability Weighting (SNIPW) is an approach to address the variance issue with the original IPW. It estimates the policy value by dividing the sum of weighted rewards by the sum of importance weights as follows.

$$\hat{V}_{\text{SNIPW}}(\pi_e; \mathcal{D}) := \frac{\mathbb{E}_n[w(x_i, a_i)r_i]}{\mathbb{E}_n[w(x_i, a_i)]}.$$

SNIPW is more stable than IPW, because the policy value estimated by SNIPW is bounded in the support of rewards and its conditional variance given action and context is bounded by the conditional variance of the rewards [20]. IPW does not have these properties. We can define Self-Normalized Doubly Robust (SNDR) in a similar manner as follows.

$$\hat{V}_{\text{SNDR}}(\pi_e; \mathcal{D}) := \mathbb{E}_n \left[\hat{q}(x_i, \pi_e) + \frac{w(x_i, a_i)}{\mathbb{E}_n[w(x_i, a_i)]}(r_i - \hat{q}(x_i, a_i)) \right].$$

Switch Estimator. The original DR estimator can still be subject to the variance issue, particularly when the importance weights are large due to weak overlap. Switch-DR aims to alleviate the variance issue by using DM where the importance weights are large as:

$$\hat{V}_{\text{SwitchDR}}(\pi_e; \mathcal{D}, \hat{q}, \tau) := \mathbb{E}_n \left[\hat{q}(x_i, \pi_e) + w(x_i, a_i)(r_i - \hat{q}(x_i, a_i))\mathbb{I}\{w(x_i, a_i) \le \tau\} \right],$$

where $\mathbb{I}\{\cdot\}$ is the indicator function and $\tau \ge 0$ is a hyperparameter. Switch-DR interpolates between DM and DR. When $\tau = 0$, it is identical to DM, while $\tau \to \infty$ yields DR.

Doubly Robust with Optimistic Shrinkage (DRos). Su et al. [41] propose DRos based on a new weight function $\hat{w} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}_+$, which directly minimizes sharp bounds on the MSE of the resulting estimator. DRos is defined as

$$\hat{V}_{\text{DRos}}(\pi_e; \mathcal{D}, \hat{q}, \lambda) := \mathbb{E}_n[\hat{q}(x_i, \pi_e) + \hat{w}(x_i, a_i; \lambda)(r_i - \hat{q}(x_i, a_i))],$$

where $\lambda \ge 0$ is a pre-defined hyperparameter and the new weight is

$$\hat{w}(x, a; \lambda) := \frac{\lambda}{w^2(x, a) + \lambda} w(x, a).$$

When $\lambda = 0$, $\hat{w}(x, a; \lambda) = 0$ leads to DM. On the other hand, as $\lambda \to \infty$, $\hat{w}(x, a; \lambda) = w(x, a)$ leading to DR.

Cross-Fitting Procedure. We sometimes use *cross-fitting* to avoid the overfitting issue in obtaining $\hat{q}$ from logged bandit data [33]. We describe the cross-fitting procedure below.

1. Take a K-fold random partition $(\mathcal{D}_k)_{k=1}^K$ of size n of $\mathcal{D}$ such that the size of each fold is $n_k = n/K$. We also define $\mathcal{D}_k^c := \mathcal{D} \backslash \mathcal{D}_k$ for $k = 1, 2, \ldots K$.

2. For each $k = 1, 2, \ldots K$, obtain reward estimators $\{\hat{q}_k\}_{k=1}^K$ with $\mathcal{D}_k^c$.

3. Given reward estimators $\{\hat{q}_k\}_{k=1}^K$, estimate $V(\pi_e)$ by $K^{-1} \sum_{k=1}^K \hat{V}(\pi_e; \mathcal{D}_k, \hat{q}_k)$ where $\hat{V}$ is a model-dependent estimator.

$K(\ge 2)$ is the number of folds and we use $K = 2$ in the benchmark experiments.

Automatic Hyperparameter Tuning. As we have summarized, many estimators in the OPE community depend on hyperparameters such as λ and τ. Recently, an automatic hyperparameter tuning procedure is proposed by Su et al. [41], which tunes the hyperparameter values as follows.

$$\hat{\theta} \in \operatorname*{argmin}_{\theta \in \Theta} \overline{\text{Bias}}(\theta; \mathcal{D})^2 + \hat{\mathbb{V}}(\theta; \mathcal{D}),$$

where $\overline{\mathrm{Bias}}(\theta; \mathcal{D})$ is the bias upper bound estimable with $\mathcal{D}$, and $\hat{\mathbb{V}}(\theta; \mathcal{D})$ is the sample variance. Su et al. [41] describe some variants of the bias upper bound including the following direct bias estimation:

$$\overline{\mathrm{Bias}}(\theta; \mathcal{D}) := |\mathbb{E}_n[(\hat{w}(x_i, a_i; \theta) - w(x_i, a_i))(r_i - \hat{q}(x_i, a_i))]|$$

$$+ \sqrt{\frac{2\mathbb{E}[w(x, a)^2]\log(2/\delta)}{n}} + \frac{2w_{\max}\log(2/\delta)}{3n}$$

where $\delta \in (0, 1]$ defines a confidence in deriving a high probability upper bound, and $w_{\max} := \max_{x,a} w(x, a)$ is the maximum importance weight. $\hat{w}(x, a; \theta)$ is the importance weight modified by a hyperparameter. For example, DRos use $\hat{w}(x, a; \lambda) = \frac{\lambda}{w^2(x,a)+\lambda}w(x, a)$, while Switch-DR is based on $\hat{w}(x, a; \tau) = w(x, a)\mathbb{I}\{w(x, a) \leq \tau\}$. In Section 5, we use this direct bias estimation with $\delta = 0.05$ to conduct automatic hyperparameter tuning.

Algorithm 1 Experimental protocol for evaluating OPE estimators

---

Input: policy π_e; two different logged bandit datasets $\mathcal{D}^{(e)} = \{(x_i^{(e)}, a_i^{(e)}, r_i^{(e)})\}_{i=1}^{n^{(e)}}$ and $\mathcal{D}^{(b)} = \{(x_i^{(b)}, a_i^{(b)}, r_i^{(b)})\}_{i=1}^{n^{(b)}}$ where $\mathcal{D}^{(e)}$ is collected by π_e and $\mathcal{D}^{(b)}$ is collected by a different one π_b; an off-policy estimator to be evaluated $\hat{V}$; a number of bootstrap iterations T; sample size n

Output: some statistics of $SE(\hat{V})$ (such as the root mean squared error)

1: $V_{\mathrm{on}}(\pi_e) = (1/n^{(e)}) \sum_{i=1}^{n^{(e)}} r_i^{(e)}$ (on-policy estimation of $V(\pi_e)$)
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad \mathcal{D}_t^{(b,*)} = \mathrm{Bootstrap}(\mathcal{D}^{(b)}, n)$ (sample data of size n from $\mathcal{D}^{(b)}$ with *replacement*)
4: $\quad SE(\hat{V}; \mathcal{D}^{(b)}) := \left(\hat{V}(\pi_e; \mathcal{D}^{(b)}) - V_{\mathrm{on}}(\pi_e)\right)^2$, (calculate the squared error in OPE)
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{SE(\hat{V}; \mathcal{D}_{\mathrm{ev}}^{(b,*)})\}$
6: **end for**

Table 7: Elapsed Time (minutes)

Sample Size (n)	ALL	Men's	Women's
10,000	22.38	10.22	12.96
30,000	48.73	19.23	24.18
50,000	107.84	29.00	39.25
100,000	208.91	57.58	89.70
300,000	750.34	220.53	332.56

Note: This table presents the time (in minutes) needed to run the evaluation of OPE experiment using Algorithm 1 with $T = 200$ and different sample size $n \in \{10000, 30000, 50000, 100000, 300000\}$.

Table 8: Estimation performance of reward estimator $\hat{q}$ with different machine learning methods

Models ($\hat{q}$)	Random → Bernoulli TS			Bernoulli TS → Random		
	ALL	Men's	Women's	ALL	Men's	Women's
GB	0.2954 ± 0.1000	0.1871 ± 0.0756	0.2598 ± 0.0555	0.0854 ± 0.0751	0.1028 ± 0.1337	0.0910 ± 0.0422
LR	0.1091 ± 0.0313	0.0745 ± 0.0194	0.0821 ± 0.0250	0.0338 ± 0.0145	0.0641 ± 0.0202	0.0232 ± 0.0110

Note: This table presents the relative cross-entropy (RCE) of the reward estimator for each campaign. The averaged results and their unbiased standard deviations estimated using 200 different bootstrapped samples are reported. **GB** and **LR** stand for Gradient Boosting and Logistic Regression, respectively. $\pi_b \rightarrow \pi_e$ represents the OPE situation where the estimators aim to estimate the policy value of π_e using logged bandit data collected by π_b, meaning that $\hat{q}$ is trained on data collected by π_b.

C Additional Experimental Setups and Results

In this section, we describe some additional experimental setups and results.

C.1 Computational Resource

All experiments were conducted on MacBook Pro (2.4 GHz Intel Core i9, 64 GB). Table 7 summarizes the time needed to run the evaluation of OPE experiments with $T = 200$ and varying sample size $n \in \{10000, 30000, 50000, 100000, 300000\}$.

C.2 Estimation Performance of Reward Estimator ($\hat{q}$)

We evaluate the performance of the reward estimators by using relative cross entropy (RCE). RCE is defined as the improvement of an estimation performance relative to the naive estimation, which uses the mean CTR for every prediction. We calculate this metric using a size n of validation samples

$\{(x_i, a_i, r_i)\}_{i=1}^{n}$ as:

$$RCE\left(\hat{q}\right) := 1 - \frac{\sum_{i=1}^{n} r_i \log(\hat{q}(x_i, a_i)) + (1 - r_i) \log(1 - \hat{q}(x_i, a_i))}{\sum_{i=1}^{n} r_i \log(\hat{q}_{\text{naive}}) + (1 - r_i) \log(1 - \hat{q}_{\text{naive}})}$$

where $\hat{q}_{\text{naive}} := n^{-1} \sum_{i=1}^{n} r_i$ is the naive estimation using the mean CTR for every estimation. A larger value of RCE means a better performance of a predictor. We use RCE as an evaluation metric to evaluate $\hat{q}$, as in OPE, the estimation accuracy of $\hat{q}$ (not Recall or Precision) matters.

Table 8 reports the estimation accuracy of logistic regression and gradient boosting as a reward estimator $\hat{q}$. Specifically, we use 'sklearn.linear_model.LogisticRegression(C=100, random_state=12345)' as logistic regression and 'sklearn.ensemble.HistGradientBoostingClassifier(learning_rate=0.01, max_iter=100, max_depth=5, min_samples_leaf=10, random_state=12345)' as gradient boosting to obtain $\hat{q}$. Note that we use action-related feature vectors to represent action variables to train reward estimators. The action-related feature vectors used in the benchmark experiments are available at https://github.com/st-tech/zr-obp/blob/master/obd/bts/all/item_context.csv. Note also that, in the evaluation of OPE experiments, we use gradient boosting (with cross-fitting) to obtain $\hat{q}$. Table 8 contains the estimation accuracy of logistic regression as a baseline. The table shows that gradient boosting is more accurate in estimating q than logistic regression.

C.3 Additional Experimental Results

Table 9 shows the benchmark results (RMSE) for the three campaigns when the OPE situation is **Random** $\rightarrow$ **Bernoulli TS**. Table 10 compares *small*-sample and *large*-sample OPE performance on the three campaigns when the OPE situation is **Random** $\rightarrow$ **Bernoulli TS**. Table 11 compares the OPE performance of the model-dependent estimators (i.e., DM, DR, SNDR, Switch-DR, and DRos) with different reward estimators. The results show that the choice of machine learning method to construct $\hat{q}$ greatly affects the OPE performance. Specifically, gradient boosting leads to a better OPE than logistic regression as its reward estimation is relatively accurate (see Table 8). We also observe the similar trend for Men's and Women's campaigns.

Table 9: RMSE ($\times 10^3$) of OPE estimators (**Random $\rightarrow$ Bernoulli TS**)

OPE Estimators ($\hat{V}$)	ALL	Men's	Women's
IPW[1]	0.474 (1.012)[3]	1.298 (1.145)[3]	1.645 (2.817)
SNIPW[2]	0.468 (1.000)[3]	1.133 (1.000)[1/3]	1.585 (2.713)
DM[3]	**1.499 (3.201)**	**1.758 (1.551)**	**0.584 (1.000)**[1/2/4/5/6/7]
DR[4]	0.501 (1.070)[3]	1.169 (1.032)[1/3]	1.539 (2.636)[1]
SNDR[5]	0.500 (1.067)[3]	1.151 (1.032)[1/3]	1.524 (2.609)[1]
Switch-DR[6]	0.501 (1.070)[3]	1.169 (1.032)[1/3]	1.539 (2.036)[1]
DRos[7]	0.508 (1.085)[3]	1.171 (1.033)[1/3]	1.524 (2.609)[1]

Table 10: Comparison of *small*-sample and *large*-sample OPE performance (RMSE $\times 10^3$)

OPE Estimators	ALL		Men's		Women's	
	small-sample	*large*-sample	*small*-sample	*large*-sample	*small*-sample	*large*-sample
IPW[1]	1.104	0.474[3]	3.156	1.298[3]	**6.789**	1.645
SNIPW[2]	1.100[3/5]	0.468[3]	3.038	**1.133**[1/3]	5.685[1]	1.585
DM[3]	1.128	**1.499**	**2.665**[1/2/4/5/6/7]	**1.758**	**2.986**[1/2/4/5/6/7]	**0.584**[1/2/4/5/6/7]
DR[4]	1.133	0.501[3]	3.055	1.169[1/3]	5.724[1]	1.539[1]
SNDR[5]	1.148	0.500[3]	3.069	1.151[1/3]	5.476[1]	1.524[1]
Switch-DR[6]	1.133	0.501[3]	3.055	1.169[1/3]	5.724[1]	1.539[1]
DRos[7]	1.125	0.500[3]	3.051	1.171[1/3]	5.694[1]	1.524[1]

Note: Root mean squared errors (RMSEs) estimated with 200 different bootstrapped iterations are reported (**Random $\rightarrow$ Bernoulli TS**). RMSEs normalized by the best (lowest) RMSE are reported in parentheses. n is the sample size of $\mathcal{D}^{(b,*)}$, which is used in OPE. $n = 10,000$ for the *small*-sample setting, while $n = 300,000$ for the *large*-sample setting. [1/2/3/4/5/6/7] denote a significant difference compared to the indicated estimator (Wilcoxon rank-sum test, $p < 0.05$). The **red bold** is used when the best results outperform the second bests in a statistically significant level. The **blue bold** is used when the worst results underperform the second worsts in a statistically significant level.

Table 11: Comparison of OPE performance (RMSE $\times 10^3$) with different reward estimators

OPE Estimators	LR	GB	Improvement of GB over LR
DM	1.597	1.026	64.2%
DR	2.250	0.482	78.5%
SNDR	2.255	0.482	78.7%
Switch-DR	2.250	0.482	78.5%
DRos	2.065	0.316	84.7%

Note: Root mean squared errors (RMSEs) estimated with 200 different bootstrapped iterations are reported ($n = 300,000$, **ALL** campaign, **Bernoulli TS $\rightarrow$ Random**). **GB** and **LR** stand for Gradient Boosting and Logistic Regression, respectively.

timestamp	item_id	position	click indicator	action_prob	user_features		user_item_affinity	
2019-11-xx	25	1	0	0.0125	e2500f3f	faiweurg	0	1
2019-11-xx	32	2	1	0.0871	7c414ef7	juqj2qfd	1	0
2019-11-xx	11	3	0	0.0613	60bd4df9	fji23qhrf	0	3
2019-11-xx	40	1	0	0.1889	7c20d9b5	slafhas2	2	0
...	...	...	...	...	...	...	...	...

Figure 2: Schema of Open Bandit Dataset.

D   Open Bandit Dataset

We publish *Open Bandit Dataset* at: `https://research.zozo.com/data.html`. It is our priority to protect the privacy of third parties. We bear all responsibility in case of violation of rights, etc., and confirmation of the data license.

Overview   The dataset currently consists of a total of about 26M rows, each representing a user impression with some feature values, selected items as actions, true action choice probabilities by the data collection policies, and click indicators as reward variables. Specifically, Figure 2 describes the schema of Open Bandit Dataset, where

- timestamp: timestamp of impressions.
- item_id: index of items as arms (index ranges from 0-79 in "ALL" campaign, 0-33 for "Men" campaign, and 0-45 "Women" campaign).
- position: the position of an item being recommended (1, 2, or 3 correspond to the left, center, and right positions of the ZOZOTOWN recommendation interface in Figure 1, respectively).
- click_indicator: a reward variable that indicates if an item was clicked (1) or not (0).
- action_prob: the probability of an item being recommended at the given position by a data collection policy.
- user features (categorical): user-related feature values such as age and gender. User features are anonymized using a hash function.
- user-item affinity (numerical): user-item affinity scores induced by the number of past clicks observed between each user-item pair.

Dataset Documentation   We include the dataset documentation and intended uses in the dataset website.

Terms of use and License.   Open Bandit Dataset is published under CC BY 4.0. license, which means everyone can use this dataset for non-commercial research purposes.
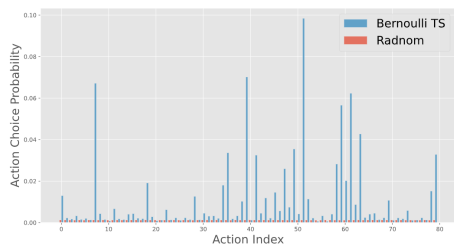
Dataset maintenance.   We will maintain the dataset for a long time at the same website and check the data accessibility in a regular basis. We will also maintain the small size example data at the Open Bandit Pipeline repository: `https://github.com/st-tech/zr-obp/tree/master/obd`. This will help potential users grasp how the dataset works with Open Bandit Pipeline. We also maintain the data loader class implemented in Open Bandit Pipeline.[15] When a function of the package is updated, we will let the users know the changes via the project mailing list.

Benchmark and code.   `https://github.com/st-tech/zr-obp/tree/master/benchmark/ope` provides scripts and instructions to reproduce the benchmark results using Open Bandit Dataset. We utilize tools such as *Hydra*[16] and *Poetry*[17] to ensure the reproducibility of the benchmark results.

---

[15]https://github.com/st-tech/zr-obp/blob/master/obp/dataset/real.py
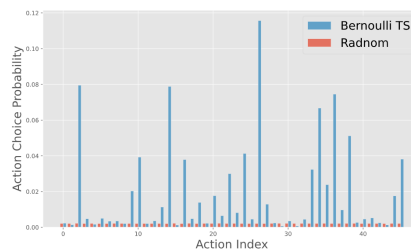[16]https://hydra.cc/
[17]https://python-poetry.org/

ALL

Men's

Women's

Table 12: Empirical Action Distribution (**Bernoulli TS vs Random**)

ALL

Men's

Women's

Table 13: Empirical Action Distribution (**Random only**)

Dataset statistics. Figure 12 shows the empirical action distributions in Open Bandit Dataset. We see that the action distribution is uniform when the dataset is collected under the Random policy (see also Figure 13). In contrast, we can see that some actions are much more frequently observed than others when the dataset is collected under the Bernoulli TS policy, producing the bias in OPE.

Potential Negative Societal Impacts and General Ethical Conduct

Our open data and pipeline contribute to fair and transparent machine learning research, especially bandit algorithms and off-policy evaluation. By setting up a common ground for credibly evaluating the performance of bandit and off-policy evaluation methods, our work is expected to foster their real-world applications. A limitation is that it is difficult to generalize the experimental results and conclusions based on our data to other important domains, such as education, healthcare, and the social sciences. To enable generalizable comparison and evaluation of bandit algorithms and off-policy evaluation, it is desired to construct public benchmark datasets from a broader range of domains.

As we have touched on in Section 3, we hashed the feature vectors related to the users included in the dataset. Therefore, our dataset does not contain any personally identifiable information or sensitive personally identifiable information.
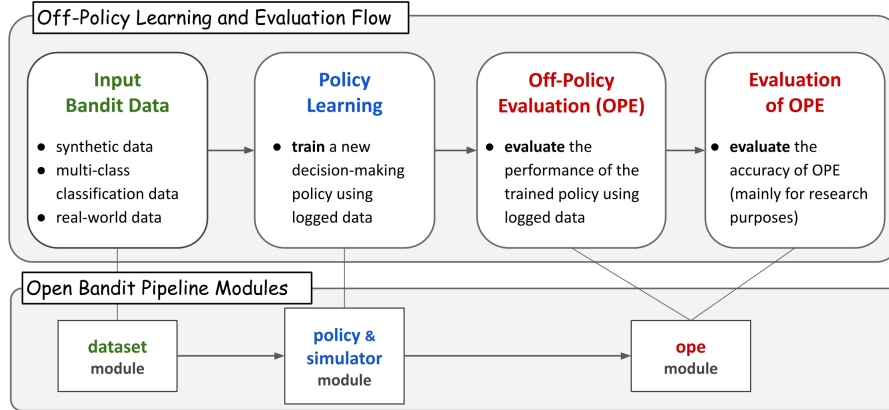
Figure 3: Open Bandit Pipeline Overview.

E Open Bandit Pipeline (OBP) Package

As described in Section 3, Open Bandit Pipeline contains implementations of dataset preprocessing, several policy learning methods, and a variety of OPE estimators including several advanced methods.

E.1 Core Functions

Modules   Open Bandit Pipeline consists of the following three main modules.

- **dataset module**[18]: This module provides a data loader for Open Bandit Dataset and a flexible interface for handling logged bandit data. It also provides tools to generate synthetic bandit data and transform multiclass classification data to bandit data.
- **policy module**[19]: This module provides interfaces for implementing new online and offline bandit policies. It also implements several standard policy learning methods.
- **ope module**[20]: This module provides interfaces for implementing OPE estimators. It also implements several standard and advanced OPE estimators.

Supported Estimators and Algorithms   Open Bandit Pipeline implements the following bandit algorithms and OPE estimators.

Bandit Algorithms

- Online Bandit Algorithms
  - Non-contextual
    * Uniform Random
    * Epsilon Greedy
    * Bernoulli Thompson Sampling
  - Contextual
    * Linear/Logistic Epsilon Greedy
    * Linear/Logistic Thompson Sampling
    * Linear/Logistic Upper Confidence Bound
- Offline (Batch) Bandit Algorithms
- IPW Learner (used with a *scikit-learn* classifier)
- Neural Network Policy Learner (implemented with *PyTorch*)

---

[18]https://github.com/st-tech/zr-obp/tree/master/obp/dataset
[19]https://github.com/st-tech/zr-obp/tree/master/obp/policy
[20]https://github.com/st-tech/zr-obp/tree/master/obp/ope

OPE Estimators

- Standard OPE (formulated in Section 2)

    - Direct Method

    - Inverse Probability Weighting [35]

    - Self-Normalized Inverse Probability Weighting [44]

    - Doubly Robust [8]

    - Self-Normalized Doubly Robust

    - Switch Doubly Robust [51]

    - Doubly Robust with Optimistic Shrinkage [41]

- OPE for Slate Recommendation

    - Independent Inverse Propensity Score [27]

    - Reward Interaction Inverse Propensity Score [30]

- OPE for Continuous Action

    - Kernelized Inverse Probability Weighting [21]

    - Kernelized Self-Normalized Inverse Probability Weighting [21]

    - Kernelized Doubly Robust [21]

Please refer to the package documentation for the basic formulation of OPE and the definitions of the estimators.[21] Note that, in addition to the above algorithms and estimators, the pipeline also provides flexible interfaces. Therefore, researchers can easily implement their own algorithms or estimators and evaluate them with our data and pipeline. Moreover, the pipeline provides an interface for handling real-world logged bandit data. Thus, practitioners can combine their own data with the pipeline and easily evaluate bandit algorithms' performance in their settings with OPE.

E.2 Example Code (OPE Experiment with Open Bandit Dataset)

Below, we show an example of conducting an OPE of the performance of BernoulliTS using IPW as an OPE estimator and the Random policy as a behavior policy. We see that only ten lines of code are sufficient to complete the standard OPE procedure from scratch (Code Snippet 1).

---

[21]https://zr-obp.readthedocs.io/en/latest/

```
# implementing OPE of BernoulliTS using log data generated by Random
>>> from obp.dataset import OpenBanditDataset
>>> from obp.policy import BernoulliTS
>>> from obp.ope import OffPolicyEvaluation, InverseProbabilityWeighting as IPW

# (1) Data Loading and Preprocessing
>>> dataset = OpenBanditDataset(behavior_policy="random", campaign="all")
>>> bandit_feedback = dataset.obtain_batch_bandit_feedback()

# (2) Production Policy Replication
>>> evaluation_policy = BernoulliTS(
        n_actions=dataset.n_actions,
        len_list=dataset.len_list,
        is_zozotown_prior=True, # replicate policy used in the ZOZOTOWN production
        campaign="all",
        random_state=0
    )
>>> action_dist = evaluation_policy.compute_batch_action_dist(
        n_sim=100000, n_rounds=bandit_feedback["n_rounds"]
    )

# (3) Off-Policy Evaluation
>>> ope = OffPolicyEvaluation(
        bandit_feedback=bandit_feedback,
        ope_estimators=[IPW()]
    )
>>> estimated_policy_value = ope.estimate_policy_values(action_dist=action_dist)

# estimate the performance improvement of BernoulliTS over Random
>>> ground_truth_random = bandit_feedback["reward"].mean()
>>> print(estimated_policy_value["ipw"] / ground_truth_random)
1.198126...
```

Code Snippet 1: **Overall Flow of Off-Policy Evaluation using Open Bandit Pipeline**

In the following subsections, we explain some important features in the example flow.

E.2.1 Data Loading and Preprocessing

We prepare easy-to-use data loader for Open Bandit Dataset. `obp.dataset.OpenBanditDataset` will download and preprocess the original data.

```
# load and preprocess raw data in the "ALL" campaign collected by the Random policy
>>> dataset = OpenBanditDataset(behavior_policy="random", campaign="all")
# obtain logged bandit feedback generated by the behavior policy
>>> bandit_feedback = dataset.obtain_batch_bandit_feedback()
```

Code Snippet 2: **Data Loading and Preprcessing**

Users can implement their own feature engineering in the `pre_process` method of `obp.dataset.OpenBanditDataset`. Moreover, by following the interface of `BaseBanditDataset` in the dataset module, one can handle future open datasets for bandit algorithms and OPE. The dataset module also provides a class to generate synthetic bandit datasets and to modify multiclass classification data to bandit data.

E.2.2 Production Policy Replication

After preparing the logged bandit data, we now replicate BernoulliTS used during the data collection in the ZOZOTOWN production. Then, we can use it as the evaluation policy.

```
# define evaluation policy (the Bernoulli TS policy here)
>>> evaluation_policy = BernoulliTS(
        n_actions=dataset.n_actions,
        len_list=dataset.len_list,
        is_zozotown_prior=True, # replicate BernoulliTS in the ZOZOTOWN production
        campaign="all",
        random_state=0
    )
# compute action choice probabilities of the evaluation policy by running simulation
# action_dist is an array of shape (n_rounds, n_actions, len_list)
# representing action choice probabilities of the evaluation policy
>>> action_dist = evaluation_policy.compute_batch_action_dist(
        n_sim=100000, n_rounds=bandit_feedback["n_rounds"]
    )
```

Code Snippet 3: **Production Policy Replication**

The `compute_batch_action_dist` method of BernoulliTS computes action choice probabilities based on given hyperparameters of the beta distribution. By activating the `is_zozotown_prior` argument, one can replicate BernoulliTS used in the ZOZOTOWN production. `action_dist` is an array representing the distribution over actions (i.e., action choice probabilities) made by the evaluation policy.

E.2.3 Off-Policy Evaluation

Our final step is OPE, which attempts to estimate the performance of bandit policies using only the log data generated by a behavior policy. Our pipeline provides an easy procedure to implement OPE as follows.

```
# estimate the policy value of BernoulliTS based on its action choice probabilities
# it is possible to set multiple OPE estimators to the `ope_estimators` argument
>>> ope = OffPolicyEvaluation(
        bandit_feedback=bandit_feedback,
        ope_estimators=[IPW()]
    )
>>> estimated_policy_value = ope.estimate_policy_values(action_dist=action_dist)
>>> print(estimated_policy_value)
{"ipw": 0.004553...} # dictionary containing policy values estimated by each
    estimator

# compare the estimated performance of BernoulliTS with the performance of Random
# our OPE procedure suggests that BernoulliTS improves Random by 19.81%
>>> ground_truth_random = bandit_feedback["reward"].mean()
>>> print(estimated_policy_value["ipw"] / ground_truth_random)
1.198126...
```

Code Snippet 4: **Off-Policy Evaluation**

Users can implement their own OPE estimator by following the interface of `BaseOffPolicyEstimator` class. `OffPolicyEvaluation` class summarizes and compares the policy values estimated by several OPE estimators. `bandit_feedback["reward"].mean()` is the empirical mean of factual rewards (on-policy estimate of the policy value) in the log and thus is the performance of the Random policy during the data collection period.

E.3 Example Code (Bandit Experiment with Open Bandit Dataset)

By using Open Bandit Dataset, we can evaluate the performance of (online and offline) bandit algorithms. Here is an example of using Open Bandit Dataset to evaluate the performance of IPWLearner, which is an offline bandit learning method described in Appendix A.

```
# implementing the evaluation of IPWLearner using log data generated by Random
>>> from sklearn.linear_model import LogisticRegression
# import open bandit pipeline (obp)
>>> from obp.dataset import OpenBanditDataset
>>> from obp.policy import IPWLearner
>>> from obp.ope import OffPolicyEvaluation, InverseProbabilityWeighting as IPW

# (1) Data Loading and Preprocessing
>>> dataset = OpenBanditDataset(behavior_policy="random", campaign="all")
>>> bandit_feedback_train, bandit_feedback_test =
    dataset.obtain_batch_bandit_feedback(is_timeseries_split=True)

# (2) Off-Policy Learning
>>> eval_policy = IPWLearner(
        n_actions=dataset.n_actions, base_classifier=LogisticRegression()
    )
>>> eval_policy.fit(
        context=bandit_feedback_train["context"],
        action=bandit_feedback_train["action"],
        reward=bandit_feedback_train["reward"],
        pscore=bandit_feedback_train["pscore"]
    )
>>> action_dist = eval_policy.predict(context=bandit_feedback_test["context"])

# (3) Off-Policy Evaluation
>>> ope = OffPolicyEvaluation(
        bandit_feedback=bandit_feedback,
        ope_estimators=[IPW()]
    )
>>> estimated_policy_value = ope.estimate_policy_values(action_dist=action_dist)

# performance improvement of IPWLearner over Random (baseline)
>>> ground_truth_random = bandit_feedback["reward"].mean()
>>> print(estimated_policy_value["ipw"] / ground_truth_random)
1.198126...
```

Code Snippet 5: **Bandit Experiment with Open Bandit Dataset**

By following the above flow, researchers can evaluate the performance of their (online or offline) bandit algorithms with Open Bandit Dataset and Pipeline.

A formal quickstart example with synthetic classification data is available at https://github.com/st-tech/zr-obp/blob/master/examples/quickstart/opl.ipynb. We also prepare a script to conduct the evaluation of bandit algorithms with synthetic data in https://github.com/st-tech/zr-obp/tree/master/examples/opl (evaluation of offline bandit algorithms) and https://github.com/st-tech/zr-obp/tree/master/examples/online (evaluation of online bandit algorithms).

E.4 Example Code (OPE Experiment with Synthetic Data)

With Open Bandit Pipeline, we can use synthetic data to evaluate the performance of OPE estimators. Here is an example of conducting OPE of the performance of IPWLearner using Inverse Probability Weighting (IPW) as an OPE estimator. We then evaluate the OPE performance of IPW based on the squared error.

```
# implementing OPE of IPWLearner using synthetic bandit data
>>> from sklearn.linear_model import LogisticRegression
# import open bandit pipeline (obp)
>>> from obp.dataset import SyntheticBanditDataset
>>> from obp.policy import IPWLearner
>>> from obp.ope import (
        OffPolicyEvaluation,
        RegressionModel,
        InverseProbabilityWeighting as IPW,
    )

# (1) Generate Synthetic Bandit Data
dataset = SyntheticBanditDataset(n_actions=10, reward_type="binary")
>>> bandit_feedback_train = dataset.obtain_batch_bandit_feedback(n_rounds=1000)
>>> bandit_feedback_test = dataset.obtain_batch_bandit_feedback(n_rounds=1000)

# (2) Off-Policy Learning
>>> eval_policy = IPWLearner(
        n_actions=dataset.n_actions, base_classifier=LogisticRegression()
    )
>>> eval_policy.fit(
        context=bandit_feedback_train["context"],
        action=bandit_feedback_train["action"],
        reward=bandit_feedback_train["reward"],
        pscore=bandit_feedback_train["pscore"]
    )
>>> action_dist = eval_policy.predict(context=bandit_feedback_test["context"])

# (3) Off-Policy Evaluation and Evaluation of OPE
>>> ope = OffPolicyEvaluation(
        bandit_feedback=bandit_feedback,
        ope_estimators=[IPW()]
    )
# evaluate the estimation performance (accuracy) of IPW by the squared error ("se")
>>> squared_errors = ope.evaluate_performance_of_estimators(
            ground_truth_policy_value=ground_truth,
            action_dist=action_dist,
            metric="se",
    )
>>> print(squared_errors)
{'ipw': 0.00267235896316153} # the accuracy of IPW in OPE
```

Code Snippet 6: **OPE Experiment with Synthetic Data**

By following the above flow, researchers can evaluate the performance of their OPE estimators on synthetic data. A formal quickstart example with synthetic classification data is available at https://github.com/st-tech/zr-obp/blob/master/examples/quickstart/synthetic.ipynb. We also prepare a script to conduct the evaluation of OPE experiments with synthetic data in https://github.com/st-tech/zr-obp/tree/master/examples/synthetic.

E.5 Example Code (OPE Experiment with Multiclass Classification Data)

Researchers often use multiclass classification data to evaluate the estimation accuracy of OPE estimators [1, 8, 10, 20, 41, 42, 51]. Appendix G of Farajtabar et al. [10] describes how to modify classification datasets into bandit data in detail. Open Bandit Pipeline facilitates this kind of OPE experiments with multiclass classification data as follows.

```
# implementing evaluation of OPE using classification data
>>> from sklearn.datasets import load_digits
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.linear_model import LogisticRegression
# import open bandit pipeline (obp)
>>> from obp.dataset import MultiClassToBanditReduction
>>> from obp.ope import OffPolicyEvaluation, InverseProbabilityWeighting as IPW


# (1) Data Loading and Bandit Reduction
>>> X, y = load_digits(return_X_y=True)
>>> dataset = MultiClassToBanditReduction(
        X=X,
        y=y,
        base_classifier_b=LogisticRegression(random_state=0)
    )
>>> dataset.split_train_eval(eval_size=0.7, random_state=0)
>>> bandit_feedback = dataset.obtain_batch_bandit_feedback(random_state=0)

# (2) Evaluation Policy Derivation
# obtain action choice probabilities of an evaluation policy
>>> action_dist = dataset.obtain_action_dist_by_eval_policy(
        base_classifier_e=RandomForestClassifier(random_state=0)
    )
# calculate the ground-truth performance of the evaluation policy
>>> ground_truth = dataset.calc_ground_truth_policy_value(action_dist=action_dist)
>>> print(ground_truth)
0.9634340222575517

# (3) Off-Policy Evaluation and Evaluation of OPE
>>> ope = OffPolicyEvaluation(
        bandit_feedback=bandit_feedback,
        ope_estimators=[IPW()]
    )
# evaluate the estimation performance (accuracy) of IPW by the squared error ("se")
>>> squared_errors = ope.evaluate_performance_of_estimators(
            ground_truth_policy_value=ground_truth,
            action_dist=action_dist,
            metric="se",
    )
>>> print(squared_errors)
{'ipw': 0.01827255896321327} # the accuracy of IPW in OPE
```

Code Snippet 7: **OPE Experiment with Classification Data**

By following the above flow, researchers can evaluate the performance of their OPE estimators by transforming classification data into bandit data.

A formal quickstart example with multiclass classification data is available at https://github.com/st-tech/zr-obp/blob/master/examples/quickstart/multiclass.ipynb. We also prepare a script to conduct the evaluation of OPE experiments with multiclass classification data in https://github.com/st-tech/zr-obp/tree/master/examples/multiclass.