# Supplementary Materials: Open Dataset, Pipeline, and Benchmark for Off-Policy Evaluation

## A. Examples

Our setup allows for many popular multi-armed bandit algorithms, as the following examples illustrate.

**Example 1** (Random A/B testing)**.** *We always choose each action uniformly at random: $\pi_b(\cdot|X) = \frac{1}{m+1}$ always holds for any $a \in \mathcal{A}$ and $X \in \mathcal{X}$.*

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

$$\pi(a|X_t) = \Pr\{a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \tilde{Y}(a')\}.$$

## B. Definitions of Off-Policy Estimators

Here, we summarize several standard OPE methods.

**Direct Method (DM).** A widely-used method, DM (Beygelzimer & Langford, 2009), first learns a supervised machine learning model, such as random forest, ridge regression, and gradient boosting, to estimate the mean reward function. DM then uses it to estimate the policy value as

$$\hat{V}_{DM}(\pi_e; \mathcal{D}, \hat{\mu}) = \frac{1}{T} \sum_{t=1}^{T} \sum_{a=0}^{m} \hat{\mu}(X_t, a) \pi_e(a|X_t).$$

where $\hat{\mu}(x, a)$ is the estimated reward function. If $\hat{\mu}(x, a)$ is a good approximation to the mean reward function, this estimator accurately estimates the policy value of the evaluation policy $V(\pi_e)$. If $\hat{\mu}(x, a)$ fails to approximate the mean reward function well, 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 (Farajtabar et al., 2018).

**Inverse Probability Weighting (IPW).** To alleviate the issue with DM, researchers often use another estimator called IPW (Precup et al., 2000; Strehl et al., 2010). IPW re-weights the rewards by the ratio of the evaluation policy and behavior policy as

$$\hat{V}_{IPW}(\pi_e; \mathcal{D}) = \frac{1}{T} \sum_{t=1}^{T} Y_t \frac{\pi_e(A_t|X_t)}{\pi_b(A_t|X_t)}.$$

When the behavior policy is known, the IPW estimator is unbiased and consistent for the policy value. However, it can have a large variance, especially when the evaluation policy significantly deviates from the behavior policy.

**Doubly Robust (DR).** DR (Dudík et al., 2014) combines DM and IPW as

$$\hat{V}_{DR}(\pi_e; \mathcal{D}, \hat{\mu}) = \hat{V}_{DM}(\pi_e; \mathcal{D}, \hat{\mu}) + \frac{1}{T} \sum_{t=1}^{T} (Y_t - \hat{\mu}(X_t, A_t)) \frac{\pi_e(A_t|X_t)}{\pi_b(A_t|X_t)}.$$

DR mimics IPW to use a weighted version of rewards, but DR also uses the estimated mean reward function as a control variate to decrease the variance. It preserves the consistency of IPW if either the importance weight or the mean reward estimator is accurate (a property called *double robustness*). Moreover, DR is *semiparametric efficient* (Narita et al., 2019) when the mean reward estimator is correctly specified. On the other hand, when it is wrong, this estimator can have larger asymptotic mean-squared-error than IPW (Kallus & Uehara, 2019) and perform poorly in practice (Kang et al., 2007).

**Self-Normalized Inverse Probability Weighting (SNIPW)** . 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:

$$\hat{V}_{SNIPW}(\pi_e; \mathcal{D}) = \frac{1}{\sum_{t=1}^{T} \frac{\pi_e(A_t|X_t)}{\pi_b(A_t|X_t)}} \sum_{t=1}^{T} Y_t \frac{\pi_e(A_t|X_t)}{\pi_b(A_t|X_t)}.$$

SNIPW is more stable than IPW, because estimated policy value 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 (Kallus & Uehara, 2019). IPW does not have these properties.

**Switch Doubly Robust (Switch-DR)** . The DR estimator can still be subject to the variance issue, particularly when the importance weights are large due to low overlap. Switch-DR aims to reduce the effect of the variance issue by using DM where importance weights are large as:

$$\hat{V}_{\textit{Switch-DR}}(\pi_e; \mathcal{D}, \hat{\mu}) = \frac{1}{T} \sum_{t=1}^{T} \sum_{a=0}^{m} \pi_e(a|X_t) \hat{\mu}(X_t, a) + \frac{1}{T} \sum_{t=1}^{T} (Y_t - \hat{\mu}(X_t, A_t)) \frac{\pi_e(A_t|X_t)}{\pi_b(A_t|X_t)} \mathbb{I}\{\frac{\pi_e(A_t|X_t)}{\pi_b(A_t|X_t)} \leq \tau\}.$$

where $\mathbb{I}\{\cdot\}$ is the indicator function and $\tau \geq 0$ is a hyperparameter. Switch-DR interpolates between DM and DR. When $\tau = 0$, it coincides with DM, while $\tau \to \infty$ yields DR. This estimator is minimax optimal when $\tau$ is appropriately chosen (Wang et al., 2017).

**Algorithm 1** Experimental Protocol for Evaluating Off-Policy Estimators

**Require:** a policy $\pi^{(1)}$; two different logged bandit feedback datasets $\mathcal{D}^{(1)} = \{(X_t^{(1)}, A_t^{(1)}, Y_t^{(1)})\}_{t=1}^T$ and $\mathcal{D}^{(2)} = \{(X_t^{(2)}, A_t^{(2)}, Y_t^{(2)})\}_{t=1}^T$ where $\mathcal{D}^{(1)}$ is collected by $\pi^{(1)}$ and $\mathcal{D}^{(2)}$ is collected by a different one; an off-policy estimator to be evaluated $\hat{V}$; *split-point* $\tilde{t}$

**Ensure:** the mean and standard deviations of *relative-EE* $(\hat{V})$

1: $\mathcal{S} \leftarrow \emptyset$
2: Define the evaluation set: (*in*-sample case) $\mathcal{D}_{eval} \coloneqq \mathcal{D}_{1:T}^{(2)}$, (*out*-sample case) $\mathcal{D}_{eval} \coloneqq \mathcal{D}_{1:\tilde{t}}^{(2)}$
3: Define the test set: (*in*-sample case) $\mathcal{D}_{test} \coloneqq \mathcal{D}_{1:T}^{(1)}$, (*out*-sample case) $\mathcal{D}_{test} \coloneqq \mathcal{D}_{\tilde{t}+1:T}^{(1)}$
4: Approximate $V(\pi^{(1)})$ by its on-policy estimation using $\mathcal{D}_{test}$
5: **for** $k = 1, \dots, K$ **do**
6: Sample data from $\mathcal{D}_{eval}$ with *replacement* and construct $k$-th bootstrapped samples $\mathcal{D}_{eval}^{(k)}$
7: Estimate the policy value of $\pi^{(1)}$ by $\hat{V}(\pi^{(1)}; \mathcal{D}_{eval}^{(k)})$
8: $\mathcal{S} \leftarrow \mathcal{S} \cup \{$*relative-EE* $(\hat{V}; \mathcal{D}_{eval}^{(k)})\}$
9: **end for**
10: Estimate the mean and standard deviations of *relative-EE* $(\hat{V})$ by using $\mathcal{S}$

*Table 5.* Estimation Performances of the Regression Model ($\hat{\mu}$)

| Metric | Behavior Policies | Campaigns: All | Men's | Women's |
|---|---|---|---|---|
| AUC | RANDOM | 0.56097 [0.55625, 0.56507] | 0.58256 [0.57395, 0.58976] | 0.55797 [0.55552, 0.56035] |
| | BERNOULLI TS | 0.57073 [0.57012, 0.57144] | 0.57576 [0.57296, 0.57855] | 0.54737 [0.54574, 0.54903] |
| RCE | RANDOM | 0.00221 [0.00127, 0.00300] | 0.00409 [0.00140, 0.00650] | -0.00022 [-0.00319, 0.00195] |
| | BERNOULLI TS | 0.00573 [0.00560, 0.00583] | 0.00593 [0.00553, 0.00632] | 0.00314 [0.00305, 0.00331] |

*Notes*: This table presents the and relative cross-rntropy (RCE) of the regression model on a validation set. The accuracy measures averaged over 5 different bootstrapped samples are reported.

# C. Additional Experimental Settings and Results

## C.1. Detailed Experimental Protocol

We describe detailed protocols for evaluating OPE estimators in Algorithm 1.

## C.2. Prediction Accuracy of the Regression Model

We evaluate the performance of the regression model by using the following two evaluation metrics in classification.

**Relative Cross Entropy (RCE).** RCE is defined as the improvement of a prediction relative to the naive prediction, which predicts the mean CTR for every data. We calculate this metric using a size $n$ of validation samples $\{(x_t, y_t)\}_{t=1}^n$ as:

$$\textit{RCE of } \hat{\mu} = 1 - \frac{\sum_{t=1}^n y_t \log(\hat{\mu}(x_t)) + (1-y_t)\log(1-\hat{\mu}(x_t))}{\sum_{t=1}^n y_t \log(\hat{\mu}_{naive}) + (1-y_t)\log(1-\hat{\mu}_{naive})}$$

where $\hat{\mu}_{naive} = n^{-1}\sum_{t=1}^n y_t$ is the naive prediction. A larger value of RCE means better performance of a predictor.

**Area Under the ROC Curve (AUC).** AUC is defined as the probability that positive samples are ranked higher than negative items by a classifier under consideration.

$$\textit{AUC of } \hat{\mu} = \frac{1}{n^{\mathrm{pos}} n^{\mathrm{neg}}} \sum_{t=1}^{n^{\mathrm{pos}}} \sum_{j=1}^{n^{\mathrm{neg}}} \mathbb{I}\{\hat{\mu}(x_t^{\mathrm{pos}}) > \hat{\mu}(x_j^{\mathrm{neg}})\}$$

*Table 6.* Comparing Relative-Estimation Errors of OPE Estimators (**Men's Campaign**)

| | **OPE Situations and *in*- or *out*-sample** | | | |
|---|---|---|---|---|
| | **Random → Bernoulli TS** | | **Bernoulli TS → Random** | |
| **Estimators** | *in*-sample | *out*-sample | *in*-sample | *out*-sample |
| DM | 0.24151 ±0.01914 | 0.28127 ±0.03001 | 0.24275 ±0.00961 | 0.11324 ±0.01370 |
| IPW | 0.09806 ±0.03203 | 0.20723 ±0.05773 | 0.03682 ±0.01626 | 0.06987 ±0.02281 |
| SNIPW | 0.08153 ±0.03409 | 0.18744 ±0.05693 | 0.04165 ±0.02909 | 0.14973 ±0.03938 |
| DR | 0.08530 ±0.03336 | 0.19176 ±0.05559 | 0.10129 ±0.06458 | 0.21363 ±0.08588 |
| Switch-DR ($\tau = 0.1$) | 0.25981 ±0.03032 | 0.25981 ±0.03032 | 0.24373 ±0.00954 | 0.11409 ±0.01370 |
| Switch-DR ($\tau = 1.0$) | 0.26697 ±0.03223 | 0.26697 ±0.03223 | 0.24653 ±0.01041 | 0.11636 ±0.01400 |
| Switch-DR ($\tau = 10$) | 0.19176 ±0.05559 | 0.10983 ±0.01913 | 0.17704 ±0.00482 | 0.05485 ±0.01064 |

*Notes*: The averaged relative-estimation errors of estimators and their unbiased standard deviations are reported. $\pi^{(2)} \rightarrow \pi^{(1)}$ represents the OPE situation where the estimators aim to estimate the policy value of $\pi^{(1)}$ using logged bandit data collected by $\pi^{(2)}$.

*Table 7.* Comparing Relative-Estimation Errors of OPE Estimators (**Women's Campaign**)

| | **OPE Situations and *in*- or *out*-sample** | | | |
|---|---|---|---|---|
| | **Random → Bernoulli TS** | | **Bernoulli TS → Random** | |
| **Estimators** | *in*-sample | *out*-sample | *in*-sample | *out*-sample |
| DM | 0.22886 ±0.00758 | 0.24886 ±0.01825 | 0.31892 ±0.00678 | 0.22413 ±0.00737 |
| IPW | 0.03252 ±0.02158 | 0.02831 ±0.01974 | 0.04635 ±0.01498 | 0.10508 ±0.01520 |
| SNIPW | 0.03179 ±0.02225 | 0.03073 ±0.02519 | 0.07551 ±0.01308 | 0.12611 ±0.02006 |
| DR | 0.03224 ±0.02233 | 0.03006 ±0.02446 | 0.08971 ±0.01580 | 0.13877 ±0.02285 |
| Switch-DR ($\tau = 0.1$) | 0.23109 ±0.00716 | 0.25084 ±0.01815 | 0.32055 ±0.00676 | 0.22585 ±0.00730 |
| Switch-DR ($\tau = 1.0$) | 0.21877 ±0.00797 | 0.24223 ±0.01777 | 0.31825 ±0.00734 | 0.22611 ±0.00820 |
| Switch-DR ($\tau = 10$) | 0.05674 ±0.02379 | 0.08696 ±0.05231 | 0.21710 ±0.01148 | 0.12253 ±0.00825 |

*Notes*: The averaged relative-estimation errors of estimators and their unbiased standard deviations are reported. $\pi^{(2)} \rightarrow \pi^{(1)}$ represents the OPE situation where the estimators aim to estimate the policy value of $\pi^{(1)}$ using logged bandit data collected by $\pi^{(2)}$.

where $\mathbb{I}\{\cdot\}$ is the indicator function. $\{x_t^{\mathrm{pos}}\}_{t=1}^{n^{\mathrm{pos}}}$ and $\{x_j^{\mathrm{neg}}\}_{j=1}^{n^{\mathrm{neg}}}$ are sets of positive and negative samples in the validation set, respectively. A larger value of AUC means better performance of a predictor.

## D. Open Bandit Pipeline (OBP) Package

As described in Section 3, *Open Bandit Pipeline* contains implementations of dataset preprocessing, offline bandit simulator, several bandit policies, and OPE estimators.

Below, we show an example of conducting an offline evaluation of the performance of BernoulliTS using Replay Method (Li et al., 2011) as an OPE estimator and the Random policy as a behavior policy. We see that only ten lines of code are sufficient to complete OPE from scratch (Code Snippet 1).

```
# a case for implementing OPE of BernoulliTS using log data generated by Random
>>> from obp.dataset import OpenBanditDataset
>>> from obp.policy import BernoulliTS
>>> from obp.simulator import run_bandit_simulation
>>> from obp.ope import OffPolicyEvaluation, ReplayMethod

# (1) Data loading and preprocessing
>>> data = OpenBanditDataset(behavior_policy='random', campaign='women')
>>> bandit_feedback = data.obtain_batch_bandit_feedback()

# (2) Offline Bandit Simulation
>>> new_policy = BernoulliTS(n_actions=data.n_actions, len_list=data.len_list)
>>> action_dist = run_bandit_simulation(bandit_feedback, policy=new_policy)

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

# estimated performance of BernoulliTS relative to the ground-truth performance of Random
>>> ground_truth_random = bandit_feedback['reward'].mean()
>>> relative_policy_value = estimated_policy_value['rm'] / ground_truth_random
>>> print(relative_policy_value) # 1.120574...
```

Code Snippet 1: **Overall Flow of using OBP**

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

### D.1. Data Loading and Preprocessing

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

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

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

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

### D.2. Offline Bandit Simulation

After preparing our data, we now run offline bandit simulation on the logged bandit feedback as follows.

```
# define a counterfacutal policy (Bernoulli TS)
>>> new_policy = BernoulliTS(n_actions=data.n_actions, len_list=data.len_list)
# `action_dist` is an array representing the distribution over action by the evaluation
    policy
>>> action_dist = run_bandit_simulation(bandit_feedback, policy=new_policy)
```

Code Snippet 3: **Offline Bandit Simulation**

`run_bandit_simulation` takes a bandit policy and `bandit_feedback` (a dictionary storing logged bandit feedback) as inputs and runs offline bandit simulation of a given evaluation bandit policy. `selected_actions` is an array of selected actions during the offline bandit simulation by the evaluation policy. Users can implement their own bandit algorithms by following the interface of `obp.policy.BanditPolicy`.

### D.3. Off-Policy Evaluation

Our final step is OPE, which attempts to estimate the performance of bandit algorithms using log data generated by offline bandit simulations. Our pipeline also provides an easy procedure for doing OPE as follows.

```
# estimate the policy value of BernoulliTS based on actions
# selected by that policy in offline bandit simulation
# it is possible to set multiple OPE estimators to the `ope_estimators` argument
>>> ope = OffPolicyEvaluation(bandit_feedback, ope_estimators=[ReplayMethod()])
>>> estimated_policy_value = ope.estimate_policy_values(action_dist=action_dist)
>>> print(estimated_policy_value)
{'rm': 0.005155..}

# compare the estimated performance of BernoulliTS (evaluation policy)
# with the ground-truth performance of Random (on-policy estimation of behavior policy)
>>> ground_truth_random = bandit_feedback['reward'].mean()
>>> relative_policy_value = estimated_policy_value['rm'] / ground_truth_random
# our OPE procedure suggests that BernoulliTS improves Random by 12.05%
>>> print(relative_policy_value)
1.120574...
```

Code Snippet 3: **Off-Policy Evaluation**

Users can implement their own OPE estimator by following the interface of `BaseOffPolicyEstimator` class. `OffPolicyEvaluation` class summarizes and compares the estimated policy values by several off-policy 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 ground-truth performance of the behavior policy (the Random policy).