Evaluating Off-Policy Evaluation: Sensitivity and Robustness

Yuta Saito*
Hanjuku-Kaso Co., Ltd.
saito@hanjuku-kaso.com

Takuma Udagawa*
Sony Group Corporation
Takuma.Udagawa@sony.com

Haruka Kiyohara†
Tokyo Institute of Technology
kiyohara.h.aa@m.titech.ac.jp

Kazuki Mogi†
Stanford University
kmogi@stanford.edu

Yusuke Narita
Yale University
yusuke.narita@yale.edu

Kei Tateno
Sony Group Corporation
Kei.Tateno@sony.com

ABSTRACT

Off-policy Evaluation (OPE), or offline evaluation in general, evaluates the performance of hypothetical policies leveraging only offline log data. It is particularly useful in applications where the online interaction involves high stakes and expensive setting such as precision medicine and recommender systems. Since many OPE estimators have been proposed and some of them have hyperparameters to be tuned, there is an emerging challenge for practitioners to select and tune OPE estimators for their specific application. Unfortunately, identifying a reliable estimator from results reported in research papers is often difficult because the current experimental procedure evaluates and compares the estimators' accuracy on a narrow set of hyperparameters and evaluation policies. Therefore, we cannot know which estimator is safe and reliable to use in general practice. In this work, we develop *Interpretable Evaluation for Offline Evaluation* (IEOE), an experimental procedure to evaluate OPE estimators' sensitivity to the choice of hyperparameters and possible changes in evaluation policies in an interpretable manner. We also build open-source Python software, *pyIEOE*, to streamline the evaluation with the IEOE protocol. With this software, researchers can use IEOE to compare different OPE estimators in their research, and practitioners can select an appropriate estimator for the given practical situation. Then, using the IEOE procedure, we perform extensive re-evaluation of a wide variety of existing estimators on public datasets. We show that, surprisingly, simple estimators that have fewer hyperparameters are more reliable than other advanced estimators because advanced estimators need environment specific hyperparameter tuning to perform well. Finally, we apply IEOE to real-world e-commerce platform data and demonstrate how to use our protocol in practice.

ACM Reference Format:
Yuta Saito, Takuma Udagawa, Haruka Kiyohara, Kazuki Mogi, Yusuke Narita, and Kei Tateno. 2021. Evaluating Off-Policy Evaluation: Sensitivity and Robustness . In *Proceedings of BCIRWIS '21: Bayesian Causal Inference for Real World Interactive Systems Workshop at ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), 2021 (BCIRWIS '21).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Both authors contributed equally to this work.
†This work was done during their internship at Hanjuku-Kaso Co., Ltd.

BCIRWIS '21, August 14–15, 2021, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Interactive bandit and reinforcement learning algorithms have been used to optimize decision making in many real-life scenarios such as precision medicine, recommender systems, advertising, etc. We often use these algorithms to maximize the expected reward, but they also produce log data valuable for evaluating and redesigning future decision making. For example, the logs of a news recommender system record which news article was presented and whether the user read it, giving the decision maker a chance to make its recommendation more relevant. Exploiting log data is, however, more difficult than conventional supervised machine learning as the result is only observed for the action chosen by the algorithm but not for all the other actions the system could have taken. The logs are also biased, as the logs over-represent the actions favored by the past algorithm. Online experiment or A/B test is a potential solution to this issue. It compares the performance of counterfactual algorithms in an online environment, enabling unbiased evaluations and comparisons. However, A/B testing counterfactual algorithms is often difficult, since deploying a new policy to a real environment is time-consuming and may damage user satisfaction [6].

This motivates us to study *Off-policy Evaluation* (OPE), which aims to estimate the performance of an evaluation policy using only log data collected by a behavior policy. Such an evaluation allows us to compare the performance of candidate policies safely and helps us decide which policy should be deployed in the field. This alternative offline evaluation approach thus has the potential to overcome the above issues with the online A/B test approach.

With growing interest in offline evaluation, the research community has produced a number of estimators, including Direct Method (DM) [2], Inverse Probability Weighting (IPW) [14, 17], Self-Normalized IPW (SNIPW) [21], Doubly Robust (DR) [4], Switch-DR [25], and Doubly Robust with Optimistic Shrinkage (DRos) [18].

One emerging challenge with this trend is that there is a need for practitioners to select and tune appropriate hyperparameters for OPE estimators for their specific application [19, 24]. For example, DM first estimates the expected reward function using an arbitrary machine learning method, then uses its estimate for OPE. Therefore, one has to identify a good machine learning method to estimate the expected reward before the offline evaluation phase.

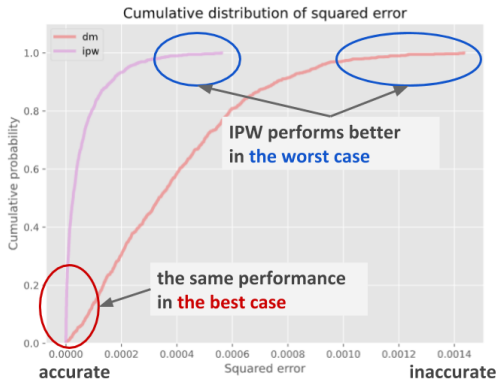


Figure 1: An example output of the proposed evaluation procedure for offline evaluation

Identifying the appropriate machine learning method for DM is difficult, because its accuracy cannot be easily quantified from bandit data [7]. Sophisticated estimators such as Switch-DR [25] and DRos [18] show improved offline evaluation performances in some experiments. However, these estimators have a larger number of hyperparameters to be tuned compared to the baseline estimators. A difficulty here is that the estimation accuracy of offline evaluation methods is highly sensitive to the choice of hyperparameters, as suggested in empirical studies [16, 24]. When we rely on offline evaluation in real-world applications, it is desirable to use an estimator that achieves accurate evaluations without requiring significant hyperparameter tuning. An estimator of this type is preferable, because tuning hyperparameters of OPE estimators (or offline evaluation metrics) with only logged bandit data is challenging in nature. Therefore, we often want to know *which estimator provides an accurate and reliable offline evaluation without any environment specific hyperparameter tuning.*

Current dominant evaluation procedures. Unfortunately, current evaluation procedures for offline evaluation used in OPE research cannot address the above question. Almost all OPE papers evaluate the estimator's performance for a single given hyperparameter and an arbitrary evaluation policy [4, 5, 10–12, 16, 18, 20, 23, 25]. Even though it is common to iterate some trials with different random seeds to provide an estimate of the performance, it cannot evaluate the estimators' sensitivity to hyperparameter choices or the changes in evaluation policies, which is critical in real-world scenarios. The estimator's performance from this common procedure does not properly account for the uncertainty in offline evaluation performance, as the reported performance measure is a single random variable drawn from the distribution over the estimator's performance. Consequently, choosing an appropriate OPE method is difficult, as their sensitivity to hyperparameter choices or the changes in evaluation policies are not quantified in existing experiments.

Contributions. Motivated towards the reliable use of offline evaluation in practice, we develop an interpretable and scalable evaluation procedure for offline evaluation methods that quantifies their sensitivity to the choice of hyperparameters and possible changes in evaluation policies. Our evaluation procedure compares several offline evaluation methods as depicted in Figure 1. This figure compares the offline evaluation performance of IPW and DM by illustrating their accuracy distributions as we vary their hyperparameters, evaluation policies, and random seeds. The x-axis is the squared error in offline evaluation; a lower value indicates that an estimator is more accurate. The figure is visually interpretable, and in this case, we are confident that IPW is better, having lower squared errors with high probability, being robust to the changes in configurations, and being more accurate even in the worst case. In addition to developing the evaluation procedure, we have implemented open-source Python software, ***pyIEOE***, so that researchers can easily implement our procedure in their experiments, and practitioners can identify the best estimator for their specific environment.

Using our procedure and software, we re-evaluate a wide variety of existing OPE estimators on several classification datasets and Open Bandit Dataset [16]. Through the extensive experiment, we derive the following surprising result:

> **Empirical Result 1 (Benchmark Experiment).** Our procedure suggests that IPW is the best in achieving reliable offline evaluation in the sense that it performs stably across a range of different experimental configurations. In contrast, popular estimators such as DR and Switch-DR are difficult to use in practice, as their performances depend heavily on hyperparameter choices.

Empirical Results 1 shows that advanced estimators (such as DR, Switch-DR, and DRos) outperform simple estimators (such as IPW and SNIPW) in only a narrow set of experimental conditions.

Finally, as a proof of concept, we use our procedure to select the best estimator for the offline evaluation of coupon treatment policies on a real-world e-commerce platform. The platform uses OPE to improve its coupon optimization policy safely without implementing A/B tests. However, the platform's data scientists do not know which offline evaluation method is appropriate for their setting. We apply our procedure to provide an appropriate estimator choice for the platform and obtain the following conclusion:

> **Empirical Result 2 (Real-World Application).** Our procedure suggests that SNIPW is the best choice for the platform because of its stable evaluation performance on the platform data.

After reliable empirical verification, the platform now uses SNIPW for offline evaluation to continuously improve the performance of its coupon optimization policy. This real-world application demonstrates how to use our procedure to reduce uncertainty and risk that we face in real-world offline evaluation.

2 OFF-POLICY EVALUATION

2.1 Setup

We consider a general contextual bandit setting. Let $r \in [0, r_{\max}]$ denote a reward or outcome variable (e.g., whether a coupon assignment as an action results in an increase in revenue). We let

$x \in \mathcal{X}$ be a context vector (e.g., the user's demographic profile) that the decision maker observes when picking an action. Rewards and contexts are sampled from unknown probability distributions $p(r \mid x, a)$ and $p(x)$, respectively. Let $\mathcal{A}$ be a finite set of discrete actions. 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 vector x.

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

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

where each context-action-reward triplet is sampled independently from the identical product distribution. Then, for a function $f(x, a, r)$, we 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 its empirical expectation over n observations in $\mathcal{D}$. We also use $q(x, a) := \mathbb{E}_{r\sim p(r|x,a)}[r \mid x, a]$ to denote the mean reward function for a given context and action.

In OPE, we are interested in using historical logged bandit data to estimate the following *policy value* of a given *evaluation policy* π_e which might be different from π_b:

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

Estimating $V(\pi_e)$ before deploying π_e in an online environment is useful in practice, because π_e may perform poorly. Additionally, this makes it possible to select an evaluation policy that maximizes the policy value by comparing their estimated performances without incurring additional implementation cost.

2.2 Existing OPE Estimators

Given the policy value as the estimand, the goal of researchers is to propose an accurate estimator. OPE estimator $\hat{V}$ estimates the policy value of an arbitrary evaluation policy as $V(\pi_e) \approx \hat{V}(\pi_e; \mathcal{D}, \theta)$, where $\mathcal{D}$ is an available logged bandit feedback dataset, and θ is a set of pre-defined *hyperparameters* of $\hat{V}$.

Below, we summarize the definitions and properties of several existing OPE methods. We also summarize their built-in hyperparameters in Table 1.

Direct Method (DM). DM [2] first learns a supervised machine learning method, such as random forest, ridge regression, and gradient boosting, to estimate the mean reward function q. DM then estimates the policy value as

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

where $\hat{q}(x, a)$ is the estimated mean reward function. If $\hat{q}(x, a)$ is a good approximation to the mean reward function, this estimator accurately estimates the policy value of the evaluation policy. If $\hat{q}(x, a)$ fails to approximate the mean reward function well, however, the final estimator is no longer consistent.

Inverse Probability Weighting (IPW). To alleviate the issue with DM, researchers often use IPW [14, 17]. IPW re-weights the rewards by the ratio of the evaluation policy and behavior policy, as

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

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

Doubly Robust (DR). DR [4] combines DM and IPW as

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

DR uses the estimated mean reward function as a control variate to decrease the variance of IPW. It is also *doubly robust* in that it is consistent to the policy value if either the importance weight or the mean reward estimator is accurate.

Self-Normalized Estimators. 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:

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

SNIPW is more stable than IPW, because 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 [9]. IPW does not have these properties. We can define Self-Normalized Doubly Robust (SNDR) in a similar manner as follows.

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

Switch Estimator. The DR estimator can still be subject to the variance issue, particularly when the importance weights are large due to low overlap between behavior and evaluation policies. Switch-DR [25] aims to further reduce variance by using DM where importance weights are large:

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

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 coincides with DM, while $\tau \to \infty$ yields DR.

Doubly Robust with Optimistic Shrinkage (DRos). Su et al. [18] proposes DRos based on a new weight function $\rho_o : \mathcal{X} \times \mathcal{A} \to \mathbb{R}_+$ that directly minimizes sharp bounds on the *mean-squared-error* (MSE) of the resulting estimator. DRos is defined as

$$\hat{V}_{\mathrm{DRos}}(\pi_e; \mathcal{D}, \hat{q}, \lambda) := \mathbb{E}_n[\mathbb{E}_{a\sim\pi_e(a|x)}[\hat{q}(x_i, a)] + \rho_o(x_i, a_i; \lambda)(r_i - \hat{q}(x_i, a_i))],$$

where $\lambda \ge 0$ is a hyperparameter and ρ_o is defined as $\rho_o(x, a; \lambda) := \frac{\lambda}{\rho^2(x,a)+\lambda}\rho(x, a)$. When $\lambda = 0$, $\rho_o(x, a; \lambda) = 0$ leading to the standard DM. On the other hand, as $\lambda \to \infty$, $\rho_o(x, a; \lambda) = \rho(x, a)$ leading to the original DR.

Table 1: Hyperparameters of the OPE estimators

OPE Estimators	Hyperparameters
Direct Method	$\hat{q}, K$
Inverse Probability Weighting (IPW) [14, 17]	$(\hat{\pi}_b)$
Self-Normalized IPW [21]	$(\hat{\pi}_b)$
Doubly Robust (DR) [4]	$\hat{q}, K, (\hat{\pi}_b)$
Self-Normalized DR	$\hat{q}, K, (\hat{\pi}_b)$
Switch-DR [25]	$\hat{q}, \tau, K, (\hat{\pi}_b)$
DRos [18]	$\hat{q}, \lambda, K, (\hat{\pi}_b)$

Note: $\hat{q}$ is an estimator for the mean reward function, q, constructed by an arbitrary machine learning method. K is the number of folds in the cross-fitting procedure. $\hat{\pi}_b$ is an estimated behavior policy. This is unnecessary when we know the true behavior policy, and thus it is in parentheses. τ and λ are non-negative hyperparameters for defining the corresponding estimators.

Cross-Fitting Procedure. To obtain a reward estimator, $\hat{q}$, we sometimes use the technique called *cross-fitting* to avoid the substantial bias that might arise due to over-fitting [13]. The cross-fitting procedure constructs a model-dependent estimator such as DM and DR as follows:

(1) Take a K-fold random partition $(\mathcal{D}_k)_{k=1}^K$ of size n of logged bandit feedback dataset $\mathcal{D}$ such that the size of each fold is $n_k = n/K$. Also, for each $k = 1, 2, \dots K$, we define $\mathcal{D}_k^c := \{1, \dots, n\} \backslash \mathcal{D}_k$.
(2) For each $k = 1, 2, \dots K$, construct reward estimators $\{\hat{q}_k\}_{k=1}^K$ using the subset of data $\mathcal{D}_k^c$.
(3) Given $\{\hat{q}_k\}_{k=1}^K$ and model-dependent estimator $\hat{V}$, estimate the policy value by $K^{-1} \sum_{k=1}^K \hat{V}(\pi_e; \mathcal{D}_k, \hat{q}_k)$.

3 EVALUATING OFFLINE EVALUATION

We have so far seen that the OPE community has developed a variety of offline evaluation methods. What every OPE research paper should do in their experiments is to compare the performance (estimation accuracy) of the existing estimators and report the results. A typical and dominant method to do so is to use the following *mean-squared-error* (MSE) as the estimator's performance measure:

$$\mathrm{MSE}(\hat{V}; \pi_e, \theta) := \mathbb{E}_{\mathcal{D}} \left[\left(V(\pi_e) - \hat{V}(\pi_e; \mathcal{D}; \theta) \right)^2 \right],$$

where $V(\pi_e)$ is the policy value and $\hat{V}$ is an estimator to be evaluated. MSE measures the squared distance between the policy value and its estimated value, and thus a lower value means a more accurate offline evaluation by $\hat{V}$. Researchers often calculate MSE of each estimator several times with different random seeds and report its mean and standard deviation for performance comparisons.

The issue with this procedure is that most of the estimators have some hyperparameters that should be chosen properly by analysts before the estimation process. Moreover, the estimation performance can vary when evaluating different evaluation policies (especially in finite sample cases). However, the current dominant procedure for evaluating offline evaluation uses only one set of hyperparameters and an arbitrary evaluation policy for each estimator, and then discusses the results derived [1, 5, 20, 23, 25].[1] This type of simplified experimental procedure does not accurately capture the uncertainty in the performance of offline evaluation methods. It cannot evaluate the sensitivity to hyperparameter choices and evaluation policy settings, as the score reported is for a single arbitrary set of hyperparameters and for a single evaluation policy.

What is often critical in offline evaluation practices is to identify an estimator that performs well for a variety of evaluation policies without problem-specific hyperparameter tuning. An estimator robust to the changes in such configurations is usable reliably in uncertain real-life scenarios. In contrast, an estimator which performs well only on a narrow set of hyperparameters and evaluation policies always entails a higher risk of failure in its particular application. Therefore, in the next section, we develop an experimental procedure that can evaluate the estimators' sensitivity to experimental configurations, leading to informative estimator comparisons in OPE research and a reliable estimator selection in practice.

4 INTERPRETABLE EVALUATION FOR OFFLINE EVALUATION

In this section, we outline our experimental protocol, *Interpretable Evaluation for Offline Evaluation* (IEOE).

As we have discussed, the expected value of performance (e.g., MSE) alone is insufficient to properly evaluate an estimator, as it discards information about its sensitivity to hyperparameter choices and changes in evaluation policies. We can conduct a more informative experiment by estimating the *cumulative distribution function* (CDF) of an estimator's performance. The CDF is the function, $F_Z : \mathbb{R} \to [0, 1]$, where Z is a random variable representing the performance metric of an estimator (e.g., the squared error).[2] $F_Z(z)$ maps a performance measure z to the probability that the estimator achieves a performance better or equal to that score, i.e., $F_Z(z) := \mathbb{P}(Z \leq z)$.

When we have size m of realizations of Z, i.e., $\mathcal{Z} := \{z_1, \dots, z_m\}$, we can estimate the CDF by

$$\hat{F}_Z(z) := \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\{z_i \leq z\}, \tag{1}$$

Using the CDF for evaluating offline evaluation methods allows researchers to compare different estimators with respect to their robustness to the varying configurations. Specifically, we can use the CDF to evaluate offline evaluation methods by examining the CDF of the estimators' performance visually or computing some summarization scores of the CDF as the estimators' performance measure. The visual investigation by plotting the CDF allows for a quick interpretation of a range of estimator's performance. Moreover, a numerical evaluation can summarize an estimator's performance. For example, we can score an estimator by the *area under the CDF curve* (AU-CDF):

$$\text{AU-CDF}(z_{\max}) := \int_0^{z_{\max}} F_Z(z) dz.$$

[1] This is why we use $\mathrm{MSE}(\hat{V}; \pi_e, \theta)$ to denote MSE so as to highlight that it depends on the estimator's hyperparameters θ and an evaluation policy π_e.
[2] In the following, without loss of generality, we assume that a lower value of Z means more accurate OPE.

Another possible summarization score is *Conditional Value-at-Risk* (CVaR) which computes the expected value of a random variable above given probability α:

$$\mathrm{CVaR}_\alpha(Z) := \mathbb{E}[Z \mid Z \geq F_Z^{-1}(\alpha)],$$

where $F_Z^{-1}(\alpha) := \operatorname{argmin}_z \{z \mid F_Z(z) \geq \alpha\}$ is the inverse of the CDF. When using CVaR, estimators are evaluated based on the average performance of the bottom $100 \times (1-\alpha)$ percent of trials. For example, $\mathrm{CVaR}_{0.7}(Z)$ is the average performance of the worst 30% of trials. In addition to AU-CDF and CVaR, we can use the standard deviation (Std), $\mathbb{E}[(Z - \mathbb{E}[Z])^2]^{1/2}$, and some other moments such as skewness of $\hat{F}(z)$ as summarization scores.

We present the IEOE procedure in Algorithm 1. To evaluate the estimation performance of $\hat{V}$, we need to specify a candidate set of hyperparameters Θ, a set of evaluation policies Π_e, a hyperparameter sampling function ϕ, and a set of random seeds $\mathcal{S}$. Then, for every seed $s \in \mathcal{S}$, the algorithm samples a set of hyperparameters $\theta \in \Theta$ based on sampler ϕ. We can use a hyperparameter tuning method for OPE estimators such as the one used in [25] as ϕ. When we cannot implement such a hyperparameter tuning method for OPE due to its implementation cost or risk of over-fitting, we can use the uniform distribution as ϕ. Next, the IEOE algorithm samples an evaluation policy $\pi_e \in \Pi_e$ from the discrete uniform distribution. Moreover, it replicates the data generating process using the *bootstrap sampling* from $\mathcal{D}$. A bootstrapped logged bandit feedback dataset is defined as $\mathcal{D}^* := \{(x_i^*, a_i^*, r_i^*)\}_{i=1}^n$ where each tuple (x_i^*, a_i^*, r_i^*) is sampled independently from $\mathcal{D}$ *with replacement*. Finally, for sampled tuple $(\pi_e, \mathcal{D}^*, \theta)$, it computes a performance measure (e.g., the squared error). After applying Algorithm 1 to several estimators and obtaining the empirical CDF of their evaluation performances, we can visualize them or compute some summarization scores (e.g., AU-CDF, CVaR, or Std) to discuss their robustness and to decide which estimator to use for a specific environment.

Algorithm 1 Interpretable Evaluation for Offline Evaluation

Input: logged bandit feedback $\mathcal{D}$, an estimator to be evaluated $\hat{V}$, a candidate set of hyperparameters Θ, a set of evaluation policies Π_e, a hyperparameter sampler ϕ (default: uniform distribution), a set of random seeds $\mathcal{S}$

Output: empirical CDF, $\hat{F}_Z$, of the squared error (SE)

$\mathcal{Z} \leftarrow \emptyset$
for $s \in \mathcal{S}$ **do**
  $\theta \leftarrow \phi(\Theta; s)$
  $\pi_e \leftarrow \mathrm{Unif}(\Pi_e; s)$
  $\mathcal{D}^* \leftarrow \mathrm{Bootstrap}(\mathcal{D}; s)$
  $z' \leftarrow \mathrm{SE}(\hat{V}; \mathcal{D}^*, \pi_e, \theta)$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z'\}$
end for
Estimate F_Z using $\mathcal{Z}$ (by Eq. (1))

5 BENCHMARK EXPERIMENTS

In this section, we perform the re-evaluation of a wide variety of OPE estimators using our protocol and software.[3] We conduct experiments on three classification datasets, OptDigits, PenDigits, and SatImage provided in the UCI repository [3]. Please refer to Appendix A.1 for a summary of the datasets.

5.1 Setup

Following previous studies [4, 5, 8, 25], we transform classification data to contextual bandit feedback data. In a classification dataset $\{(x_i, a_i)\}_{i=1}^n$, we have feature vector $x_i \in \mathcal{X}$ and ground-truth label $a_i \in \mathcal{A}$. Here, we regard a machine learning classifier $\pi_{det} : \mathcal{X} \to \Delta(\mathcal{A})$ as a deterministic policy that chooses class label $a_i \in \mathcal{A}$ as an action from feature vector x_i. We then define reward variable $r_i := \mathbb{I}\{\pi(x_i) = a_i\}$. Since the original classifier is deterministic, we make it stochastic by combining π_{det} and the uniform random policy π_u as $\pi(a \mid x) = \alpha\pi_{det}(x) + (1-\alpha)\pi_u(a)$ where $\alpha \in [0, 1]$ is an additional experimental setting.

To apply IEOE to classification data, we first randomly split each dataset into train $\mathcal{D}_{\mathrm{tr}}$ and test $\mathcal{D}_{\mathrm{te}} := \{(x_i, a_i)\}_{i=1}^{n'}$ sets. Then, we train a classifier on $\mathcal{D}_{\mathrm{tr}}$, and use it to construct a behavior policy π_b and a class of evaluation policies Π_e. By running behavior policy π_b on $\mathcal{D}_{\mathrm{te}}$, we transform $\mathcal{D}_{\mathrm{te}}$ to logged bandit feedback data $\mathcal{D}_{\mathrm{ev}} := \{(x_i, a_i^b, r_i = \mathbb{I}\{a_i^b = a_i\})\}_{i=1}^{n'}$, where $a_i^b \sim \pi_b$ is the action sampled by the behavior policy. By applying the following procedure, we compute the squared error (SE) of $\hat{V}$ for each iteration in Algorithm 1:

(1) Estimate the policy value $\hat{V}(\pi_e; \mathcal{D}^*, \theta)$ for tuple $(\pi_e, \mathcal{D}^*, \theta)$ sampled in the algorithm.
(2) Estimate $V(\pi_e)$ using the fully observed rewards in $\mathcal{D}_{\mathrm{te}}$, i.e., $V(\pi_e; \mathcal{D}_{\mathrm{te}}) := \mathbb{E}_{n'}[\mathbb{E}_{a^e \sim \pi_e(a|x_i)}[\mathbb{I}\{a^e = a_i\}]]$.
(3) Compare the off-policy estimate $\hat{V}(\pi_e; \mathcal{D}^*, \theta)$ with its ground-truth $V(\pi_e; \mathcal{D}_{\mathrm{te}})$ using SE as a performance measure of $\hat{V}$, i.e., $\mathrm{SE}(\hat{V}; \mathcal{D}^*, \pi_e, \theta) := (\hat{V}(\pi_e; \mathcal{D}^*, \theta) - V(\pi_e; \mathcal{D}_{\mathrm{te}}))^2$.

5.2 Estimators and Hyperparameters

We use our protocol and re-evaluate DM, IPW, SNIPW, DR, SNDR, Switch-DR, and DRos in an interpretable manner.

Here, we run the experiments under two different settings. First, we test the case where the true behavior policy π_b is available. In this setting, IPW and SNIPW are hyperparameter-free while the other estimators need to be tested for robustness to the choice of the pre-defined hyperparameters such as $\hat{q}$. Next, we test the OPE estimators with the estimated behavior policy $\hat{\pi}_b$, where we assume that the true behavior policy is unknown. In this case, we additionally test the OPE estimators for robustness to choice of machine learning method to obtain $\hat{\pi}_b$.

Tables 2 and 3 describe hyperparameter spaces Θ for each estimator. Note that we use RandomizedSearchCV implemented in *scikit-learn* with n_iter = 5 to tune hyperparameters of reward estimator $\hat{q}$ and behavior policy estimator $\hat{\pi}_b$. We additionally use CalibratedClassifierCV implemented in *scikit-learn* with cv = 2

[3] We run the experiments using our *pyIEOE* software and will publicize the whole code in the near future. By using it, anyone can replicate the results easily. At this time, we share *pyIEOE* in the supplementary materials. We also provide detailed description of the software in Appendix D.

when estimating the behavior policy, as calibration of the behavior policy estimator matters in OPE [15]. Then, we use the uniform distribution as ϕ for other hyperparameters such as τ in Switch-DR. Table 4 shows the true behavior policy and five different evaluation policies to construct Π_e. Finally, we set $\mathcal{S} = \{0, 1, \ldots, 499\}$.

5.3 Results

Figures 2 and 3 visually compare the CDF of the estimators' squared error for each dataset in true and estimated behavior policy settings. We also confirm the observations in a quantitative manner by computing AU-CDF, $\text{CVaR}_{0.7}$, and Std of the squared error of each OPE estimator. We report these summarization scores in Tables 5 and 6.

First, in the setting where the true behavior policy is available, it is obvious that IPW is the best estimator and achieves the most accurate estimation in almost all regions. SNIPW also performs comparably better than other estimators. In contrast, model-dependent estimators, especially DM and DRos perform poorly compared to the typical estimators such as IPW and SNIPW. [4] We observe here that these model-dependent estimators perform worse, when the reward estimator $\hat{q}$ has a serious bias issue. On the other hand, we do not have to care about the specification of $\hat{q}$ when we use IPW or SNIPW. Therefore, we conclude that simple estimators with fewer hyperparameters tend to perform well for a wide variety of settings when the true behavior policy is recorded.[5]

In the setting where the behavior policy needs to be estimated, we observe similar trends. First, Figure 3 and Table 6 show that IPW achieves the most accurate estimation even when it uses the estimated behavior policy.[6] Second, DR shows considerably large squared errors when the behavior policy is estimated. This is because DR is vulnerable to the over-fitting of $\hat{\pi}_b$. DR produces large squared errors when $\hat{\pi}_b$ over-fits the data and outputs extreme estimations (we observe that the minimum estimated action choice probability is 10^{-7}). With these extreme estimated action choice probabilities, the importance weights used in DR also becomes large, amplifying the estimation error of reward estimator $\hat{q}$. This leads to serious overestimation of the policy value of π_e. Note that other estimators based on DR such as Switch-DR and DRos prevent this overestimation issue, because they cut off extremely large importance weights. However, these advanced estimators underperform IPW in most cases in both true and estimated behavior policy settings, suggesting that the benefits of these advanced estimators are limited and they are relatively sensitive to the configuration changes.

In summary, our procedure suggests that IPW is the best in achieving reliable offline evaluation in the sense that it performs stably across different evaluation policies for both true and estimated behavior policy cases. In contrast, we find that the performance of popular estimators such as DR depend heavily on hyperparameter choices, leading to the serious overestimation issue and making them difficult to use in practice.

We suggest that future OPE research use the IEOE procedure to test the stability and robustness of OPE estimators. This additional experimental effort will produce substantial information about the estimators' usability in practice, as we have seen in this section.

6 REAL-WORLD APPLICATION

In this section, we apply our procedure to a real-world application.

6.1 Setup

To show how to use our evaluation procedure in a real-world application, we conducted a data collection experiment on a real e-commerce platform in September 2020. The platform wants to use OPE to improve the performance of its coupon optimization policy safely without conducting A/B tests. However, it does not know which estimator is appropriate for its specific application and environment. Therefore, we apply the IEOE procedure with the aim of providing suitable estimator choice for the platform.

During the data collection experiment, we constructed $\mathcal{D}_A$, $\mathcal{D}_B$, and $\mathcal{D}_C$ by randomly assigning three different policies (π_A, π_B, and π_C) to users on the platform. In this application, x is a user's context vector, a is a coupon assignment variable (where there are four different types of coupons, i.e., $|\mathcal{A}| = 4$), and r is either a user's content consumption indicator (binary outcome) or the revenue from each user observed within the 7-day period after the coupon assignment (continuous outcome). The total number of users considered in the experiment was 39,687, and each of $\mathcal{D}_A$, $\mathcal{D}_B$, and $\mathcal{D}_C$ has approximately one third of the users.

Note that, in this application, there is a risk of over-fitting due to the intensive hyperparameter tuning of OPE estimators, as the size of the logged bandit feedback data is not large. Moreover, to reduce the implementation cost, we do not want to tune OPE estimators' hyperparameters every time the platform's data scientists use OPE. Therefore, following the IEOE procedure, we aim to find and use an estimator that is stable without hyperparameter tuning.

6.2 Performance Measure

To apply our evaluation procedure, we need to define a performance measure (SE in step 6 of Algorithm 1). We can do this by using our real-world data. We first pick one of the three policies as evaluation policy π_e and regard the others as behavior policies. When we choose π_A as an evaluation policy, we define $\mathcal{D}_{\text{ev}} = \mathcal{D}_B \cup \mathcal{D}_C$ and $\mathcal{D}_{\text{te}} = \mathcal{D}_A$. Then, we apply the following procedure (which is slightly different from that with classification data)[7] to given OPE estimator $\hat{V}$ to compute SE as its performance measure:

(1) Estimate the policy value $\hat{V}(\pi_e; \mathcal{D}^*, \theta)$ with tuple $(\pi_e, \mathcal{D}^*, \theta)$ sampled in the algorithm.
(2) Estimate $V(\pi_e)$ by the *on-policy estimation* (the empirical average of the observed rewards in $\mathcal{D}_{\text{te}}$), i.e., $V_{\text{on}}(\pi_e; \mathcal{D}_{\text{te}}) := \mathbb{E}_n[r_i]$.
(3) Compare the off-policy estimate $\hat{V}(\pi_e; \mathcal{D}_{\text{ev}})$ with its on-policy counterpart $V_{\text{on}}(\pi_e; \mathcal{D}_{\text{te}})$ using SE as a performance measure of $\hat{V}$, i.e., $\text{SE}(\hat{V}; \mathcal{D}^*, \pi_e, \theta) := (\hat{V}(\pi_e; \mathcal{D}^*, \theta) - V_{\text{on}}(\pi_e; \mathcal{D}_{\text{te}}))^2$.

[4] The plot of DM is out of range due to the large squared errors in Optdigits. We share its CDF plot showing wider regions in Appendix A.
[5] We additionally confirm the results with the Open Bandit Dataset [1] in Appendix C.
[6] When the behavior policy estimator is not calibrated, IPW shows extremely large squared errors in some cases as we show in Appendix A. In such cases, SNIPW is safer to use. This additional study suggests that calibration of the behavior policy estimation matters, which is consistent with the previous empirical results provided by Raghu et al. [15].
[7] We describe the modified version of IEOE algorithm applicable to real-world logged bandit data in detail in Appendix B.

Table 2: Hyperparameter spaces for OPE estimators

OPE Estimators	Hyperparameter Spaces
Direct Method	$K \in \{1, 2, \ldots, 5\}$, $\hat{q} \in$ {LR/RR,RF,LightGBM}
Inverse Probability Weighting (IPW)	($\hat{\pi}_b \in$ {LR,RF,LightGBM})
Self-Normalized IPW	($\hat{\pi}_b \in$ {LR,RF,LightGBM})
Doubly Robust (DR)	$K \in \{1, 2, \ldots, 5\}$, $\hat{q} \in$ {LR/RR,RF,LightGBM}, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
Self-Normalized DR	$K \in \{1, 2, \ldots, 5\}$, $\hat{q} \in$ {LR/RR,RF,LightGBM}, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
Switch-DR	$\tau \in [1, 100]$, $K \in \{1, 2, \ldots, 5\}$, $\hat{q} \in$ {LR/RR,RF,LightGBM}, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
DRos	$\lambda \in [1, 100]$, $K \in \{1, 2, \ldots, 5\}$, $\hat{q} \in$ {LR/RR,RF,LightGBM}, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})

Note: LR/RR means that LogisticRegression (LR) is used when Y is binary and RidgeRigression (RR) is used otherwise. RF stands for RandomForest. $\hat{\pi}_b$ is an estimated behavior policy. This is unnecessary when we know the true behavior policy, and thus it is in parentheses. $K = 1$ means that we do not use cross-fitting and train a reward estimator using whole logged bandit feedback data $\mathcal{D}$.

Table 3: Hyperparameter spaces for reward estimator $\hat{q}$ and behavior policy estimator $\hat{\pi}_b$

Reward Estimators	Hyperparameter Spaces
LogisticRegression (binary outcome)	$C \in [10^{-3}, 10^{3}]$
RidgeRegression (continuous outcome)	$\alpha \in [10^{-2}, 10^{2}]$
RandomForest	max_depth $\in \{2, 3, \ldots, 10\}$, min_samples_split $\in \{5, 6, \ldots, 20\}$
LightGBM	learning_rate $\in [10^{-4}, 10^{-1}]$, max_depth $\in \{2, 3, \ldots, 10\}$, min_samples_leaf $\in \{5, 6, \ldots, 20\}$

Note: The names of the hyperparameters correspond to the ones specified by the *scikit-learn* package. For LogisticRegression, we use max_iter = 10000. For RandomForest, we use n_estimators = 100 and for LightGBM, we use max_iter = 100.

Table 4: Behavior and Evaluation Policies in Experiments on Classification Datasets

Behavior and Evaluation Policies	Base Machine Learning Classifier (π_{det})	Alpha (α)
behavior policy	LogisticRegression	0.9
evaluation policy 1	LogisticRegression	0.8
evaluation policy 2	LogisticRegression	0.2
evaluation policy 3	RandomForest	0.8
evaluation policy 4	RandomForest	0.2
evaluation policy 5	None (uniform random)	0.0

Note: For LogisticRegression, we use C = 100, max_iter = 10000. For RandomForest, we use n_estimators = 100, min_samples_split = 5, max_depth = 10. We also set random_state = 12345 for both classifiers. The names of the hyperparameters correspond to the ones specified by the *scikit-learn* package.

6.3 Estimators and Hyperparameters

We use the IEOE protocol to identify the best estimator among DM, IPW, SNIPW, DR, SNDR, Switch-DR, and DRos.

During the data collection experiment, we logged the true action choice probabilities for the three policies, and thus IPW and SNIPW are hyperparameter-free here. We use the hyperparameter spaces defined in Tables 2 and 3 for our real-world application. In addition, we use the uniform distribution as ϕ, and set $\mathcal{S} = \{0, 1, \ldots, 1999\}$ and $\Pi_e = \{\pi_A, \pi_B, \pi_C\}$.

6.4 Results

We applied Algorithm 1 to the above estimators for the binary outcome data and the continuous outcome data, respectively.

Figure 4 compares the CDF of the estimators' squared error for each outcome. First, it is obvious that SNIPW is the best estimator for the binary outcome case, achieving the best accuracy in almost all regions. We can also argue that SNIPW is preferable for the continuous outcome case, because it reveals the most accurate estimation in the worst case and is hyperparameter-free, although it underperforms DM in some cases. On the other hand, IPW performs poorly for both outcomes, because our dataset is not large and some behavior policies are near deterministic, making IPW an unstable estimator. Thus, it is unsafe to use IPW in our application, even though it is also hyperparameter-free and tractable.

We additionally confirm the above observations in a quantitative manner. For both binary and continuous outcomes, we compute AUCDF, $\text{CVaR}_{0.7}$, and Std of the squared error of each OPE estimator. We report these summarization scores in Table 7, and the results demonstrate that SNIPW clearly outperforms other estimators in almost all situations. Through this evaluation of offline evaluation practice, we concluded that ***the e-commerce platform should use SNIPW for its offline evaluation.*** After comprehensive accuracy

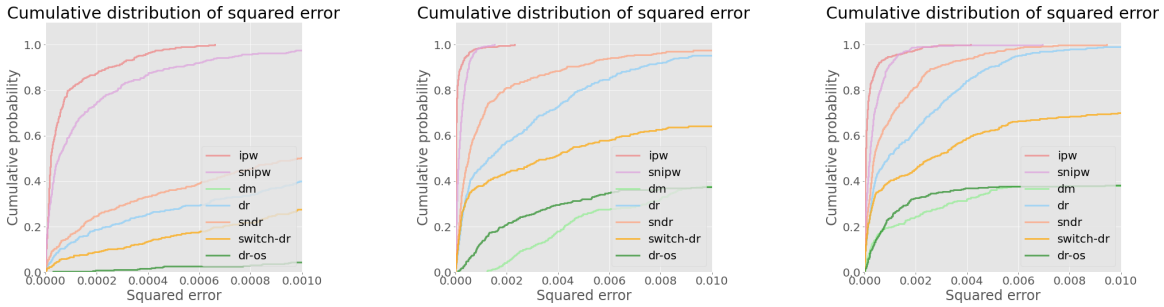


Figure 2: Comparison of the CDF of OPE estimators' squared error in the benchmark experiment with true behavior policy

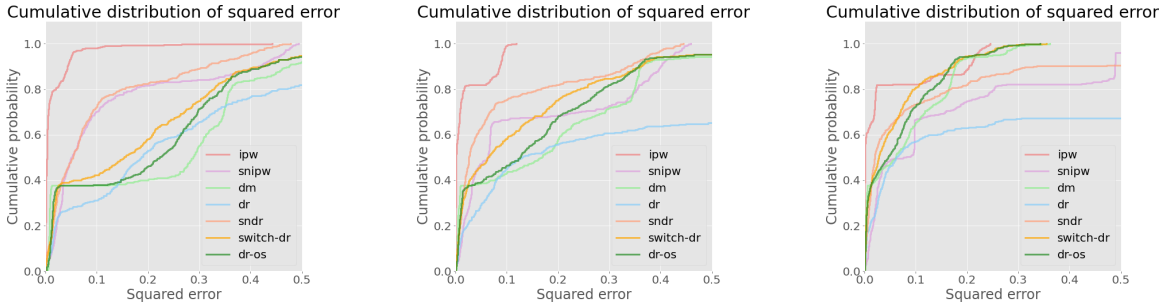


Figure 3: Comparison of the CDF of OPE estimators' squared error in the benchmark experiment with estimated behavior policy with calibration

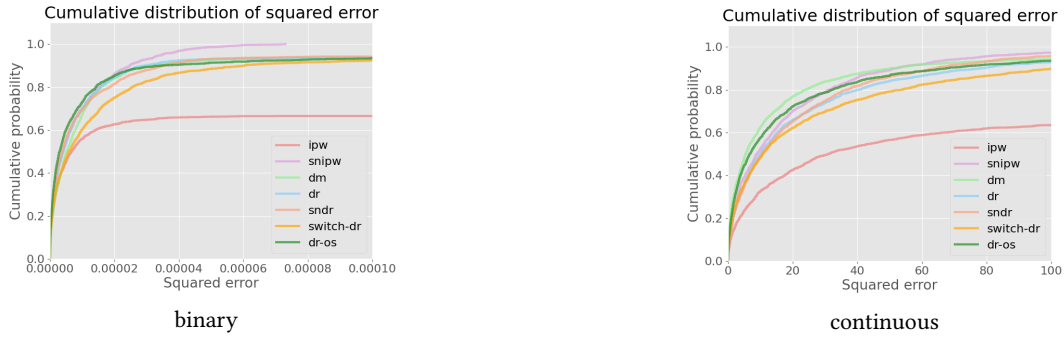


Figure 4: Comparison of the CDF of OPE estimators' squared error in real-world application

and stability verifications, the platform is now using SNIPW to improve its coupon optimization policy safely.

7 CONCLUSION AND FUTURE WORK

In this paper, we argued that the current dominant evaluation procedure for offline evaluation cannot detect the volatility of the estimators' performances. Instead, the evaluation procedure we develop for offline evaluation provides an interpretable way to not only decide which estimator is more likely to be accurate than the other, but how robust each estimator is to the choice of hyperparameters or changes in evaluation policies. We have also developed open-source software to streamline our interpretable evaluation procedure. It enables rapid benchmarking and validation of offline evaluation methods so that practitioners can spend more time on the real decision making problems, and OPE researchers can focus more on tackling advanced technical questions in future research. We perform an extensive re-evaluation of a wide variety of OPE estimators and found that IPW is more stable than other advanced estimators, being robust to varying experimental configurations.

Table 5: Summarization scores of the OPE estimators in the benchmark experiment with true behavior policy

OPE Estimators	OptDigits			PenDigits			SatImage		
	AU-CDF	$\text{CVaR}_{0.7}$	Std	AU-CDF	$\text{CVaR}_{0.7}$	Std	AU-CDF	$\text{CVaR}_{0.7}$	Std
Direct Method	0.000†	2135.07†	1591.12†	0.205†	1604.21†	765.46†	0.321†	315.98†	179.12†
Inverse Probability Weighting (IPW)	1.000*	1.00*	1.00*	1.000*	1.00*	1.02◇	1.000*	1.00*	1.00*
Self-Normalized IPW	0.906◇	2.56◇	3.22◇	0.988◇	1.92◇	1.00*	0.983◇	1.56◇	1.16◇
Doubly Robust (DR)	0.283	142.84	157.48	0.749	27.97	15.00	0.830	7.91	4.74
Self-Normalized DR	0.371	110.18	137.76	0.867	16.73	11.44	0.915	4.69	3.12
Switch-DR	0.167	1829.12	1524.48	0.531	1217.33	718.87	0.589	232.31	168.14
DRos	0.020	1856.13	1385.51	0.287	1337.07	667.24	0.346	265.40	158.69

Note: Larger value is better for **AU-CDF** and lower value is better for **CVaR** and **Std**. Note that we normalize the summarization scores by dividing them by the best score among all estimators. We use $z_{\max} = 1.0 \times 10^{-3}$ for OptDigits and $z_{\max} = 1.0 \times 10^{-2}$ for Pendigits and SatImage to calculate AU-CDF. The **red*** and **green**◇ fonts represent the best and second-best estimators, respectively. The **blue**† fonts represent the worst estimator.

Table 6: Summarization scores of the OPE estimators in the benchmark experiment with estimated behavior policy with calibration

OPE Estimators	OptDigits			PenDigits			SatImage		
	AU-CDF	$\text{CVaR}_{0.7}$	Std	AU-CDF	$\text{CVaR}_{0.7}$	Std	AU-CDF	$\text{CVaR}_{0.7}$	Std
Direct Method	0.569	11.11	5.98	0.680	7.01	5.32	0.914	1.51	1.19
Inverse Probability Weighting (IPW)	1.000*	1.00*	1.00*	1.000*	1.00*	1.00*	1.000*	1.00*	1.07
Self-Normalized IPW	0.789	7.51	4.53	0.744	6.65	4.92	0.773	3.11	2.54
Doubly Robust (DR)	0.567†	69.93†	100.94†	0.549†	1629.74†	2571.15†	0.653†	7.21×10^{7}†	2.24×10^{8}†
Self-Normalized DR	0.826◇	6.24◇	3.76◇	0.853◇	4.42◇	3.88◇	0.872	2.98	3.20
Switch-DR	0.672	9.85	5.25	0.778	5.72	4.63	0.961◇	1.17◇	1.00*
DRos	0.624	10.12	5.34	0.729	6.17	4.73	0.939	1.30	1.04◇

Note: Larger value is better for **AU-CDF** and lower value is better for **CVaR** and **Std**. Note that we normalize the scores by dividing them by the best score among all estimators. We use $z_{\max} = 0.5$ to calculate AU-CDF. The **red*** and **green**◇ fonts represent the best and second-best estimators, respectively. The **blue**† fonts represent the worst estimator.

Table 7: Summarization scores of the OPE estimators in real-world application

OPE Estimators	Binary Outcome			Continuous Outcome		
	AU-CDF	$\text{CVaR}_{0.7}$	Std	AU-CDF	$\text{CVaR}_{0.7}$	Std
Direct Method	0.954	11.88	50.73†	1.000*	1.51	2.33
Inverse Probability Weighting (IPW)	0.721†	32.86†	29.93◇	0.606†	21.03†	15.53†
Self-Normalized IPW	1.000*	1.00*	1.00*	0.962◇	1.00*	1.00*
Doubly Robust (DR)	0.972	11.25	49.62	0.905	1.57	1.71
Self-Normalized DR	0.959	10.38◇	47.76	0.919	1.21◇	1.17◇
Switch-DR	0.882	12.26	49.66	0.861	2.33	2.93
DRos	0.980◇	11.64	50.11	0.956	1.61	2.19

Note: **Binary Outcome** is the results when the outcome is each user's content consumption indicator. **Continuous Outcome** is the results when the outcome is the revenue from each user observed within the 7-day period after the coupon assignment. Larger value is better for **AU-CDF** and lower value is better for **CVaR** and **Std**. Note that we normalize the scores by dividing them by the best score among all estimators. We use $z_{\max} = 5.0 \times 10^{-5}$ for the binary outcome and $z_{\max} = 80$ for the continuous outcome to calculate AU-CDF. The **red*** and **green**◇ fonts represent the best and second-best estimators, respectively. The **blue**† fonts represent the worst estimator.

Finally, we applied our procedure to a real-world application and demonstrated its practical usage.

In future work, we plan to extend our re-evaluation to a more general reinforcement learning setting as the current re-evaluation of offline evaluation is limited to the simple contextual bandit setting. Moreover, investigating how our study could spur the development of offline policy learning methods that are robust to configuration changes would be interesting.

REFERENCES

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

[2] Alina Beygelzimer and John Langford. 2009. The offset tree for learning with partial labels. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 129–138.

[3] Dheeru Dua and Casey Graff. 2017. UCI machine learning repository. (2017).

[4] Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. 2014. Doubly robust policy evaluation and optimization. *Statist. Sci.* 29, 4 (2014), 485–511.

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

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

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

[8] Nathan Kallus, Yuta Saito, and Masatoshi Uehara. 2020. Optimal Off-Policy Evaluation from Multiple Logging Policies. *arXiv preprint arXiv:2010.11002* (2020).

[9] Nathan Kallus and Masatoshi Uehara. 2019. Intrinsically Efficient, Stable, and Bounded Off-Policy Evaluation for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 32. 3325–3334.

[10] Masahiro Kato, Shota Yasui, and Masatoshi Uehara. 2020. Off-Policy Evaluation and Learning for External Validity under a Covariate Shift. In *Advances in Neural Information Processing Systems*, Vol. 33. 49–61.

[11] Anqi Liu, Hao Liu, Anima Anandkumar, and Yisong Yue. 2019. Triply Robust Off-Policy Evaluation. *arXiv preprint arXiv:1911.05811* (2019).

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

[13] Yusuke Narita, Shota Yasui, and Kohei Yata. 2020. Off-policy Bandit and Reinforcement Learning. *arXiv preprint arXiv:2002.08536* (2020).

[14] Doina Precup. 2000. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series* (2000), 80.

[15] Aniruddh Raghu, Omer Gottesman, Yao Liu, Matthieu Komorowski, Aldo Faisal, Finale Doshi-Velez, and Emma Brunskill. 2018. Behaviour policy estimation in off-policy policy evaluation: Calibration matters. *arXiv preprint arXiv:1807.01066* (2018).

[16] Yuta Saito, Shunsuke Aihara, Megumi Matsutani, and Yusuke Narita. 2020. Open Bandit Dataset and Pipeline: Towards Realistic and Reproducible Off-Policy Evaluation. *arXiv preprint arXiv:2008.07146* (2020).

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

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

[19] Yi Su, Pavithra Srinath, and Akshay Krishnamurthy. 2020. Adaptive Estimator Selection for Off-Policy Evaluation. In *Proceedings of the 37th International Conference on Machine Learning*, Vol. 119. PMLR, 9196–9205.

[20] Yi Su, Lequn Wang, Michele Santacatterina, and Thorsten Joachims. 2019. Cab: Continuous adaptive blending for policy evaluation and learning. In *International Conference on Machine Learning*, Vol. 97. PMLR, 6005–6014.

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

[22] William R Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3/4 (1933), 285–294.

[23] Nikos Vlassis, Aurelien Bibaut, Maria Dimakopoulou, and Tony Jebara. 2019. On the design of estimators for bandit off-policy evaluation. In *International Conference on Machine Learning*, Vol. 97. PMLR, 6468–6476.

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

[25] Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudık. 2017. Optimal and adaptive off-policy evaluation in contextual bandits. In *International Conference on Machine Learning*, Vol. 70. PMLR, 3589–3597.

A ADDITIONAL SETUP AND RESULTS ON CLASSIFICATION DATASETS

A.1 Classification Dataset

Table 8 shows some statistics of the datasets used in the benchmark experiments in Section 5.

Table 8: Classification datasets used in the benchmark experiments

Datasets	#Samples	#Actions	#Dimensions
OptDigits	5,620	10	64
PenDigits	10,992	10	16
SatImage	6,435	6	36

Note: **#Samples** is the size of the dataset we use for the benchmark experiment. **#Actions** is the total number of actions (i.e., classes). **#Dimensions** is the number of dimensions of the context (i.e., feature) vector.

A.2 Result with estimated behavior policy without calibration

Here we additionally report the results for the setting where the estimated behavior policy without calibration is used.

Figure 5 and Table 9 show that, when the behavior policy estimator is not calibrated, IPW shows extremely large squared errors in some cases. In such cases, SNIPW is safer to use. This additional study suggests that calibration of the behavior policy estimation matters, which is consistent with the previous empirical results provided by Raghu et al. [15].

A.3 CDF plot in wider range

Figures 6 - 8 show CDF plots in full region of the performance.

B MODIFIED VERSION OF IEOE

Algorithm 2 presents a modified version of the proposed IEOE procedure that can be used in real-world applications. To evaluate the performance of $\hat{V}$ with real-world data, we need to specify a candidate set of hyperparameters Θ, a set of evaluation policies $\Pi_e = \{\pi_j\}_{j=1}^{\ell}$, a hyperparameter sampling function ϕ, and a set of random seeds $\mathcal{S}$. We also need several logged bandit feedback datasets $\{\mathcal{D}_j\}_{j=1}^{\ell}$ where each dataset $\mathcal{D}_j$ is collected by π_j. Then, for every seed $s \in \mathcal{S}$, the algorithm samples a set of hyperparameters $\theta \in \Theta$ based on a sampler ϕ. We can use a hyperparameter tuning method for OPE estimators such as the one used in [25] or the uniform distribution as ϕ. Next, the algorithm samples an evaluation policy $\pi_j \in \Pi_e$ from the discrete uniform distribution. Then, the evaluation and test sets are defined as $\mathcal{D}_{\mathrm{te}} = \mathcal{D}_j$ and $\mathcal{D}_{\mathrm{ev}} = \bigcup_{k=1;k\neq j}^{\ell} \mathcal{D}_j$ where the evaluation set is used in OPE and the test set is used to calculate the ground-truth performance of π_j Then, the algorithm replicates the environment $\mathcal{E}$ under consideration using the *bootstrap sampling* from $\mathcal{D}_{\mathrm{ev}}$. A bootstrapped logged bandit feedback dataset is defined as $\mathcal{D}^*_{\mathrm{ev}} := \{(x_i^*, a_i^*, r_i^*)\}_{i=1}^{n}$ where each tuple (x_i^*, a_i^*, r_i^*) is sampled independently from $\mathcal{D}_{\mathrm{ev}}$ *with replacement*. Finally, for a sampled tuple $(\pi_e, \mathcal{D}^*, \theta)$, it computes the squared error as follows

$$z = \left(V_{\mathrm{on}}(\pi_j; \mathcal{D}_{\mathrm{te}}) - \hat{V}(\pi_j; \theta, \mathcal{D}^*_{\mathrm{ev}})\right)^2$$

where $V_{\mathrm{on}}(\pi_j; \mathcal{D}_{\mathrm{te}}) = \mathbb{E}_n[r_i]$ is the on-policy estimate of the policy value of π_j estimated with the test set. After applying Algorithm 1 to several estimators and obtaining the empirical CDF of their evaluation performances, we can visualize them or compute their summarization scores to discuss their robustness and to decide which estimator to use for a specific environment.

Algorithm 2 Interpretable Evaluation for Offline Evaluation (A Modified Version)

Input: logged bandit feedback datasets $\{\mathcal{D}_j\}_{j=1}^{\ell}$, an estimator to be evaluated $\hat{V}$, a candidate set of hyperparameters Θ, a set of evaluation policies $\Pi_e = \{\pi_j\}_{j=1}^{\ell}$, a hyperparameter sampler ϕ (default: uniform distribution), a set of random seeds $\mathcal{S}$

Output: empirical CDF, $\hat{F}_Z$, of the squared error (SE)

$\mathcal{Z} \leftarrow \emptyset$
for $s \in \mathcal{S}$ **do**
  $\theta \leftarrow \phi(\Theta; s)$
  $\pi_j \leftarrow \mathrm{Unif}(\Pi_e; s)$
  $\mathcal{D}_{\mathrm{te}} = \mathcal{D}_j$ and $\mathcal{D}_{\mathrm{ev}} = \bigcup_{k=1;k\neq j}^{\ell} \mathcal{D}_j$
  $\mathcal{D}^*_{\mathrm{ev}} \leftarrow \mathrm{Bootstrap}(\mathcal{D}_{\mathrm{ev}}; s)$
  $V_{\mathrm{on}}(\pi_j; \mathcal{D}_{\mathrm{te}}) = \mathbb{E}_n[r_i]$
  $z' \leftarrow \left(V_{\mathrm{on}}(\pi_j; \mathcal{D}_{\mathrm{te}}) - \hat{V}(\pi_j; \theta, \mathcal{D}^*_{\mathrm{ev}})\right)^2$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z'\}$
end for
Estimate F_Z using $\mathcal{Z}$ (by Eq. (1))

C EXPERIMENTS WITH OPEN BANDIT DATASET

C.1 Setup

We additionally conducted experiments on Open Bandit Dataset (OBD)[8]. OBD is a set of logged bandit feedback datasets collected on a large-scale fashion e-commerce platform provided by Saito et al. [16]. There are three campaigns, "ALL", "Men", and "Women". In our experiment, we use size 30,000 and 300,000 of randomly sub-sampled data from the "ALL" campaign. The dataset contains user context as feature vector $x \in \mathcal{X}$, fashion item recommendation as action $a \in \mathcal{A}$, and click indicator as reward r. The dimensions of the feature vector x and the number of the actions $|\mathcal{A}|$ are 20 and 80, respectively.

The dataset consists of subsets of data collected by two different policies, i.e., the uniform random policy and the Bernoulli Thompson Sampling policy [22]. We let $\mathcal{D}_A$ denote a dataset collected by uniform random policy π_A and $\mathcal{D}_B$ to denote that collected by Bernoulli Thompson Sampling policy π_B. We let one of the two policies as evaluation policy π_e and regard the other as behavior policy when applying our IEOE procedure. When we choose π_A as an evaluation policy, for example, we define $\mathcal{D}_{\mathrm{ev}} = \mathcal{D}_B$ and

[8] https://research.zozo.com/data.html

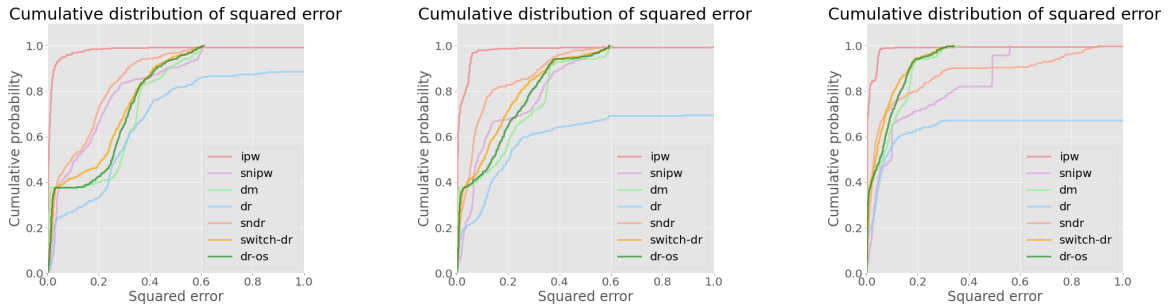


Figure 5: Comparison of the CDF of OPE estimators' squared error in the benchmark experiment with estimated behavior policy without calibration

Table 9: Summarization scores of the OPE estimators in the benchmark experiment with estimated behavior policy without calibration

	OptDigits			PenDigits			SatImage		
OPE Estimators	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**
Direct Method	0.787	2.12	1.39	0.841	1.89	1.37	0.934	3.76	1.18
Inverse Probability Weighting (IPW)	**1.000***	**1.00***	7.31	**1.000***	**1.00***	7.66	**1.000***	**1.00***	1.57
Self-Normalized IPW	0.846	1.87	1.26	0.855	1.86	1.28	0.852	8.31	2.58
Doubly Robust (DR)	**0.687**†	**8.37**†	**10.48**†	**0.604**†	**271.51**†	**407.52**†	**0.642**†	**2.15×10^{11}**†	**2.86×10^{11}**†
Self-Normalized DR	**0.881**$^\diamond$	**1.47**$^\diamond$	**1.00***	**0.918**$^\diamond$	**1.21**$^\diamond$	**1.00***	0.887	7.47	3.17
Switch-DR	0.823	1.93	**1.24**$^\diamond$	0.880	1.55	**1.16**$^\diamond$	**0.953**$^\diamond$	**2.98**$^\diamond$	**1.00***
DRos	0.807	1.97	1.26	0.863	1.67	1.22	0.944	3.29	**1.05**$^\diamond$

Note: Larger value is better for **AU-CDF** and lower value is better for **CVaR** and **Std**. Note that we normalize the scores by dividing them by the best score among all estimators. We use $z_{\max} = 1.0$ to calculate AU-CDF. The **red*** and **green**$^\diamond$ fonts represent the best and second-best estimators, respectively. The **blue**† fonts represent the worst estimator.

$\mathcal{D}_{\text{te}} = \mathcal{D}_A$. By applying the same procedure as Section 6.2, we compute SE for each OPE estimator.

In the experiments with OBD, we use true behavior policy. In addition, we use the uniform distribution as ϕ for all hyperparameters including those in reward regression model $\hat{q}$. We use the same hyperparameter spaces as in Section 5, given in Table 2 and 3. We set $\mathcal{S} = \{0, 1, \ldots, 499\}$.

C.2 Result

Figure 9 visually compares the CDF of the estimators' squared error for each dataset in both ground-truth and estimated behavior policy settings. We also report summarization scores of AU-CDF, CVaR$_{0.7}$, and Std in Table 10.

When we use large size of the dataset (i.e., size = 300,000), we confirm that the same results are observed as in the benchmark experiment with classification datasets in true behavior policy setting; IPW and SNIPW achieves better squared errors than other model-based estimators and DM especially performs poorly. On the other hand, with the small size of the dataset (i.e., size = 30,000, which is extremely small relative to the action space $|\mathcal{A}|$), IPW and SNIPW record relatively large squared errors compared to those under the large sample setting. The result indicates that, under the small sample setting, IPW and SNIPW may suffer from the variance issue, while the performances of the other estimators remain almost the same compared to those under the large sample setting. Therefore, preparing the enough size of dataset is also essential in OPE.

D SOFTWARE IMPLEMENTATION

In addition to developing the evaluation procedure, we have implemented open-source Python software, ***pyIEOE***, to streamline the evaluation of offline evaluation methods with our protocol. This package is built with the intention of being used with ***OpenBanditPipeline*** (obp).[9] We plan to publicize the whole package in a GitHub repository in the near future.

In Code Snippet 1, we show the essential codes to conduct an interpretable evaluation on various OPE estimators with our software so that one can grasp the usage of the software easily. Primarily, only four lines of code are sufficient to complete our IEOE procedure in Algorithm 1 except for some preparations.

In the following subsections, we explain the procedure including preparations in detail, by showing an example of conducting an interpetable evaluation on OPE estimators using synthetic dataset.

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

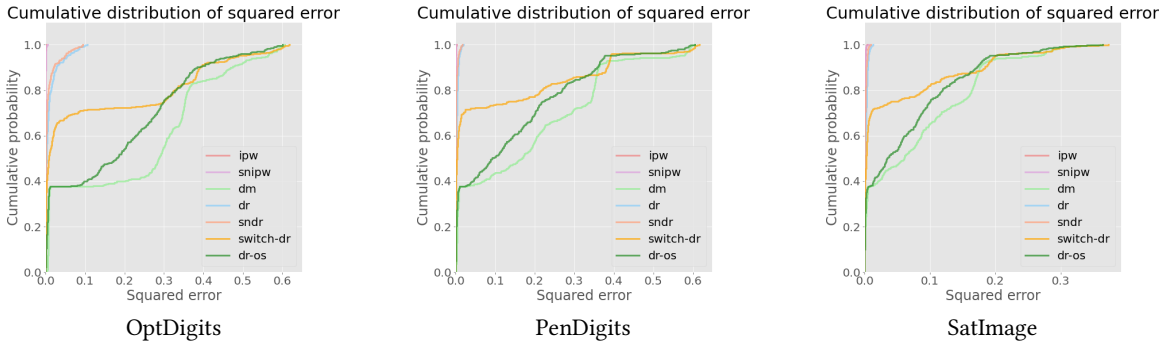


Figure 6: Comparison of the CDF of OPE estimators' squared error for the benchmark experiment with true behavior policy

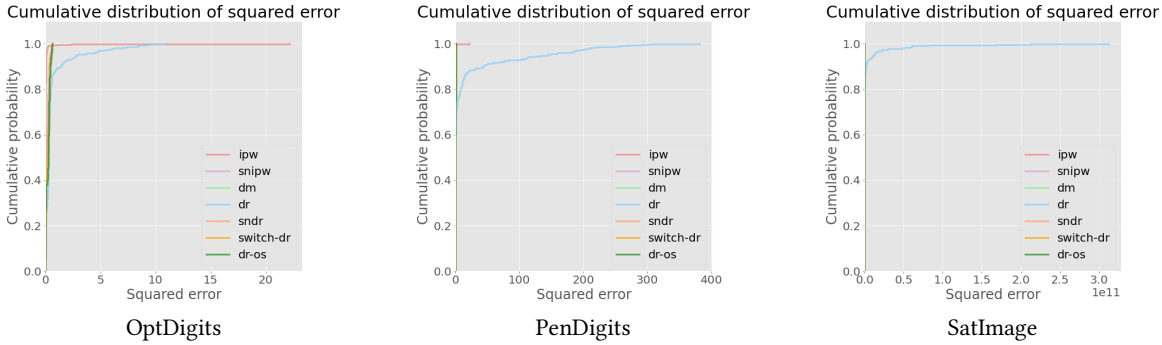


Figure 7: Comparison of the CDF of OPE estimators' squared error for the benchmark experiment with estimated behavior policy without calibration

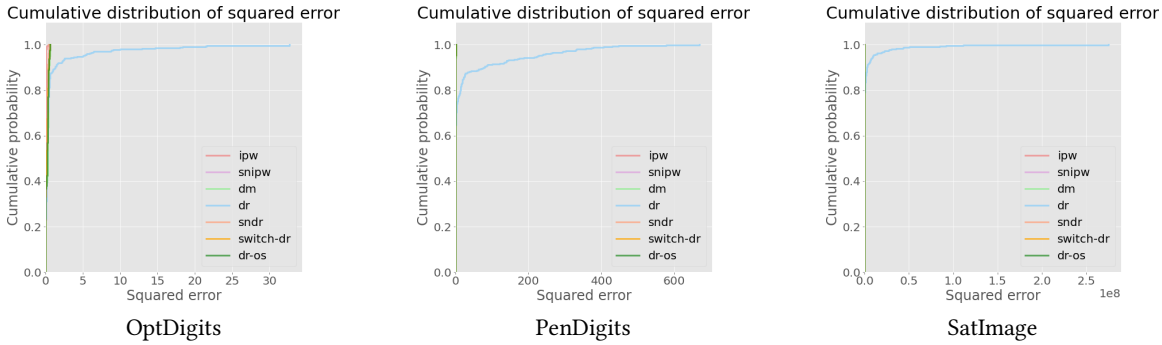


Figure 8: Comparison of the CDF of OPE estimators' squared error for the benchmark experiment with estimated behavior policy with calibration

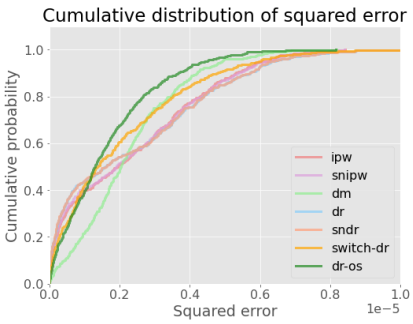
D.1 Preparing Dataset and Evaluation Policies

Before using pyIEOE, we first need to prepare logged bandit feedback data and a set of evaluation policies. Here, each evaluation policy consists of its action distribution and ground-truth policy value. We can conduct this preparation by using the dataset module of obp as shown in Code Snippet 2.

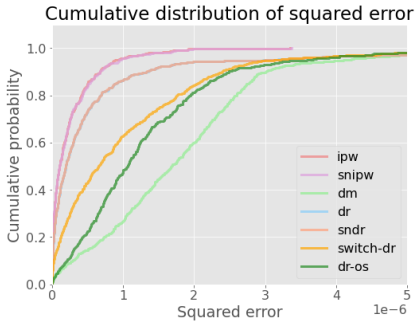
In addition to synthetic dataset, both multi-class classification dataset and real-world dataset such as Open Bandit Dataset [16] and their own environment specific dataset are applicable by following preprocessing procedure of obp. Users can also define a set of evaluation policies by themselves.

D.2 Defining Hyperparameter Spaces

After preparing dataset and a set of evaluation policies, we now define hyperparameter spaces used in OPE estimators as shown in Code Snippet 3.



Small (size 30,000)



Large (size 300,000)

Figure 9: Comparison of the CDF of OPE estimators' squared error in Open Bandit Dataset

Table 10: Summarization scores of the OPE estimators on Open Bandit Dataset with different sample size

	size = 30,000			size = 300,000		
OPE Estimators	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**
Direct Method	0.930	**1.18**$^{\diamond}$	**1.05**$^{\diamond}$	**0.682**†	**5.02**†	3.76
Inverse Probability Weighting (IPW)	**0.920**†	1.46	1.48	**1.000***	**1.00***	**1.00***
Self-Normalized IPW	0.921	1.47	1.50	**0.999**$^{\diamond}$	**1.02**$^{\diamond}$	**1.02**$^{\diamond}$
Doubly Robust (DR)	0.921	**1.54**†	**1.61**†	0.936	2.93	4.58
Self-Normalized DR	0.921	1.53	1.60	0.936	2.93	**4.59**†
Switch-DR	**0.957**$^{\diamond}$	1.31	1.36	0.839	3.97	3.99
DRos	**1.000***	**1.00***	**1.00***	0.781	4.23	3.78

Note: Larger value is better for **AU-CDF** and lower value is better for **CVaR** and **Std**. Note that we normalize the scores by dividing them by the best score among all estimators. We use $z_{\max} = 1.0 \times 10^{-5}$ for size = 30,000 and $z_{\max} = 5.0 \times 10^{-6}$ for size = 300,000 to calculate AU-CDF. The **red*** and **green**$^{\diamond}$ fonts represent the best and second-best estimators, respectively. The **blue**† fonts represent the worst estimator.

Users can define hyperparameter spaces for both OPE estimators and the regression model used in OPE estimators by themselves.

D.3 Interpretable OPE Evaluation

Finally, we evaluate OPE estimators in an interpretable manner. Our software provides an easy step to follow our evaluation procedure as shown in Code Snippet 4.

Users can intuitively find out the most reliable estimator in various OPE estimators by comparing the CDF plot of squared errors. In addition, quantitative comparison is also available by calculating AU-CDF and CVaR as performance measures. In this case, it is easy to figure out that SNDR outperforms DRos.

```
# import InterpretableOPEEvaluator
>>> from pyieoe.evaluator import InterpretableOPEEvaluator

# initialize InterpretableOPEEvaluator class
>>> evaluator = InterpretableOPEEvaluator(
       random_states=np.arange(1000),
       bandit_feedbacks=[bandit_feedback],
       evaluation_policies=[
           (ground_truth_a, action_dist_a),
           (ground_truth_b, action_dist_b),
           ..,
       ],
       ope_estimators=[
           DoublyRobustWithShrinkage(),
           SelfNormalizedDoublyRobust(),
           ..,
       ],
       regression_models=[
           LogisticRegression,
           RandomForest,
           ..,
       ],
       regression_model_hyperparams={
           LogisticRegression: lr_hp,
           RandomForest: rf_hp,
           ..,
       },
       ope_estimator_hyperparams={
           DoublyRobustWithShrinkage.estimator_name: dros_param,
           SelfNormalizedDoublyRobust.estimator_name: sndr_param,
           ..,
       }
   )

# estimate policy values
>>> policy_value = evaluator.estimate_policy_value()

# visualize CDF of squared errors for each OPE estimator
>>> evaluator.visualize_cdf_aggregate() # plot CDF curves
```

Code Snippet 1: Essential Codes for Interpretable OPE Evaluation

```
# import necessary package from obp
>>> from obp.dataset import (
        SyntheticBanditDataset,
        logistic_reward_function,
        linear_behavior_policy
    )
# initialize SyntheticBanditDataset class
>>> dataset = SyntheticBanditDataset(
        n_actions=10,
        dim_context=5,
        reward_type="binary", # "binary" or "continuous"
        reward_function=logistic_reward_function,
        behavior_policy_function=linear_behavior_policy,
        random_state=12345,
    )
# obtain synthetic logged bandit feedback data
>>> bandit_feedback = dataset.obtain_batch_bandit_feedback(n_rounds=10000)
# prepare action distribution and ground truth policy value for each evaluation policy
>>> action_dist_a = #...
>>> ground_truth_a = #...
>>> action_dist_b = #...
>>> ground_truth_b = #...
```

Code Snippet 2: Preparing Dataset and Evaluation Policies

```
# define hyperparameter spaces for ope estimators
>>> lambda_ = {
        "lower": 1e-3,
        "upper": 1e2,
        "log": True,
        "type": float
    }
>>> K = {
    "lower": 1,
    "upper": 5,
    "log": False,
    "type": int
}
>>> dros_param = {"lambda_": lambda_, "K": K}
>>> sndr_param = {"K": K}
# define hyperparameter spaces for regression models
>>> C = {
        "lower": 1e-3,
        "upper": 1e2,
        "log": True,
        "type": float
    }
>>> n_estimators = {
        "lower": 20,
        "upper": 200,
        "log": True,
        "type": int
    }
>>> lr_hp = {"C": C}
>>> rf_hp = {"n_estimators": n_estimators}
```

Code Snippet 3: Defining Hyperparameter Spaces

```
# import InterpretableOPEEvaluator
>>> from pyieoe.evaluator import InterpretableOPEEvaluator
# import other necessary packages
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.ensemble import RandomForestClassifier as RandomForest
>>> from obp.ope import DoublyRobustWithShrinkage, SelfNormalizedDoublyRobust

# initialize InterpretableOPEEvaluator class
# define OPE estimators to evaluate
>>> evaluator = InterpretableOPEEvaluator(
        random_states=np.arange(1000),
        bandit_feedbacks=[bandit_feedback],
        evaluation_policies=[
            (ground_truth_a, action_dist_a),
            (ground_truth_b, action_dist_b)
        ],
        ope_estimators=[
            DoublyRobustWithShrinkage(),
            SelfNormalizedDoublyRobust(),
        ],
        regression_models=[
            LogisticRegression,
            RandomForest,
        ],
        regression_model_hyperparams={
            LogisticRegression: lr_hp,
            RandomForest: rf_hp,
        },
        ope_estimator_hyperparams={
            DoublyRobustWithShrinkage.estimator_name: dros_param,
            SelfNormalizedDoublyRobust.estimator_name: sndr_param
        }
    )
# estimate policy values
>>> policy_value = evaluator.estimate_policy_value()
# compute squared errors
se = evaluator.calculate_squared_error()
# compare OPE estimators in an interpretable manner by visualizing CDF of squared errors
>>> evaluator.visualize_cdf_aggregate() # plot CDF curves

# quantitative analysis by AU-CDF and CVaR
>>> au_cdf = evaluator.calculate_au_cdf_score(threshold=0.004)
>>> print(au_cdf)
{"dr-os": 0.000183.., "sndr": 0.000257..}
>>> cvar = evaluator.calculate_cvar_score(alpha=70)
>>> print(cvar)
{"dr-os": 0.000456.., "sndr": 0.000194..}
```

Code Snippet 4: Interpretable OPE Evaluation