Evaluating the Robustness of Off-Policy Evaluation

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

Takuma Udagawa*
Sony Group Corporation
Tokyo, Japan
Takuma.Udagawa@sony.com

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

Kazuki Mogi†
Stanford University
California, United States
kmogi@stanford.edu

Yusuke Narita
Yale University
Connecticut, United States
yusuke.narita@yale.edu

Kei Tateno
Sony Group Corporation
Tokyo, Japan
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' performance on a narrow set of hyperparameters and evaluation policies. Therefore, it is difficult to know which estimator is safe and reliable to use. In this work, we develop *Interpretable Evaluation for Offline Evaluation* (IEOE), an experimental procedure to evaluate OPE estimators' robustness to changes in hyperparameters and/or evaluation policies in an interpretable manner. Then, using the IEOE procedure, we perform extensive evaluation of a wide variety of existing estimators on Open Bandit Dataset, a large-scale public real-world dataset for OPE. We demonstrate that our procedure can evaluate the estimators' robustness to the hyperparamter choice, helping us avoid using unsafe estimators. Finally, we apply IEOE to real-world e-commerce platform data and demonstrate how to use our protocol in practice.

KEYWORDS

off-policy evaluation, recommender systems, counterfactual estimation

ACM Reference Format:
Yuta Saito, Takuma Udagawa, Haruka Kiyohara, Kazuki Mogi, Yusuke Narita, and Kei Tateno. 2021. Evaluating the Robustness of Off-Policy Evaluation. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3460231.3474245

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

RecSys '21, September 27-October 1, 2021, Amsterdam, Netherlands

ACM ISBN 978-1-4503-8458-2/21/09…$15.00
https://doi.org/10.1145/3460231.3474245

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. This is because 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 overrepresent the actions favored by the algorithm used to collect the data. 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 evaluation and comparison. 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 [7, 19].

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 to deploy 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 OPE, the research community has produced a number of estimators, including Direct Method (DM) [2], Inverse Probability Weighting (IPW) [17, 21], Self-Normalized IPW (SNIPW) [25], Doubly Robust (DR) [4], Switch-DR [29], and Doubly Robust with Optimistic Shrinkage (DRos) [22].

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 [23, 28]. 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. Identifying the appropriate machine learning method for DM is difficult, because its accuracy cannot be easily quantified from bandit data [8]. Sophisticated estimators such as Switch-DR [29] and DRos [22] show improved offline evaluation performance 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 OPE estimators is highly sensitive to the choice of hyperparameters, as implied in empirical studies [20, 28]. When we rely on OPE in real-world applications, it is desirable to use an estimator that is robust to the choice of hyperparameters and achieves accurate evaluations without requiring significant hyperparameter tuning. Moreover, we want the estimators to be robust to other possible configuration changes such as evaluation policies. An estimator of this type is preferable, because tuning hyperparameters of OPE estimators with only logged bandit data is challenging in nature, and we often apply an estimator to several different policies to compare the performance of candidate policies offline. The aim of this paper is thus to enable a safer OPE practice by developing a procedure to evaluate the estimators' robustness.

Current dominant evaluation procedures. The current evaluation procedure used in OPE research is not suitable for evaluating the estimators' robustness. Almost all OPE papers evaluate the estimator's performance for a single given set of hyperparameters and an arbitrary evaluation policy [4, 6, 13–15, 20, 22, 24, 27, 29]. Even though it is common to iterate trials with different random seeds to provide an estimate of the performance, this procedure cannot evaluate the estimators' robustness to hyperparameter choices or the changes in evaluation policies, which is critical in real-world scenarios. The estimator's performance derived from this common procedure does not properly account for the uncertainty in offline evaluation performance, as the reported performance metric is a single random variable drawn from the distribution over the estimator's performance. Consequently, choosing an appropriate OPE estimator is difficult, as their robustness to hyperparameter choices or the changes in evaluation policies are not quantified in existing experiments.

Contributions. Motivated towards promoting a reliable use of OPE in practice, we develop an interpretable and scalable evaluation procedure for OPE estimators that quantifies their robustness to the choice of hyperparameters and possible changes in evaluation policies. Our evaluation procedure compares several OPE estimators 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

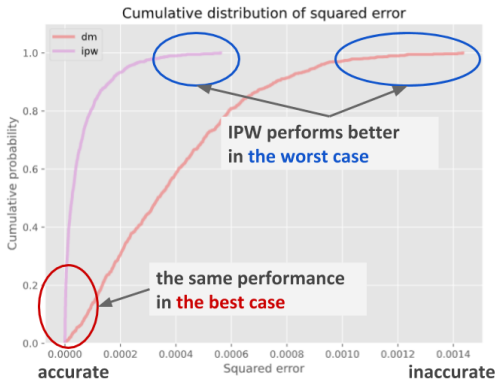


Figure 1: An example output of the proposed evaluation procedure for offline evaluation

software, ***pyIEOE***[1], 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 evaluate a wide variety of existing OPE estimators on Open Bandit Dataset [20] (Section 5) and several classification datasets (Appendix A)[2]. Through these extensive experiments, we demonstrate that IEOE can provide informative results, in particular the estimators' robustness to the hyperparameter settings and evaluation policy changes, which could not be obtained using typical experimental procedure in OPE research.

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 OPE estimator is appropriate for their setting. We apply our procedure to provide an appropriate estimator choice for the platform. This real-world application demonstrates how to use our procedure to reduce uncertainty and risk that we face in real-world offline evaluation.

Our contributions are summarized as follows.

- We develop an experimental procedure called IEOE that is useful for identifying robust estimators and avoid the use of estimators sensitive to configuration changes.
- We have implemented ***pyIEOE***, open-source Python software, that facilitates the use of our experimental procedure both in research and in practice.
- We conduct comprehensive benchmark experiments on public datasets and demonstrate that IEOE is useful for identifying estimators sensitive to configuration changes, and thus can help avoid potential failures in OPE.
- We apply IEOE to a real-world OPE application and demonstrate how this procedure helps us safely conduct OPE in practice.

[1] https://github.com/sony/pyIEOE
[2] The complete version of this paper is available at https://usaito.github.io/files/RecSys2021_IEOE.pdf

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 results in an increase in revenue) and $a \in \mathcal{A}$ be a discrete action. 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. 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. 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 estimators. We also summarize their built-in hyperparameters in Table 1.

Direct Method (DM). DM [2] first trains a supervised machine learning method, such as ridge regression, 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 tends to fail in OPE.

Inverse Probability Weighting (IPW). To alleviate the issue with DM, researchers often use IPW [17, 21]. IPW re-weights the rewards by the ratio of the evaluation policy to the 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. To reduce the variance of IPW, the following weight clipping is often applied.

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

where $\lambda \geq 0$ is a clipping hyperparamter. A lower value of λ greatly reduces the variance while introducing a large bias. Following Su et al. [22], we call IPW with weight clipping as *IPW with Pessimistic Shrinkage (IPWps)*. When $\lambda = \infty$, IPWps is identical to IPW.

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

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

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. The weight clipping can also be applied to DR as follows.

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

where $\lambda \geq 0$ is a clipping hyperparamter. DR with weight clipping is called *DR with Pessimistic Shrinkage (DRps)*. When $\lambda = \infty$, DRps is identical to DR.

Self-Normalized Estimators. SNIPW [25] is an approach to address the variance issue of 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 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 [12]. 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. DR 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 [29] aims to further reduce the variance by using DM where the importance weight is large:

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

Table 1: Hyperparameters of the OPE estimators

OPE Estimators	Hyperparameters
Direct Method (DM)	$\hat{q}$, K
Inverse Probability Weighting with Pessimistic Shrinkage (IPWps) [21, 22]	λ, $(\hat{\pi}_b)$
Self-Normalized Inverse Probability Weighting (SNIPW) [25]	$(\hat{\pi}_b)$
Doubly Robust with Pessimistic Shrinkage (DRps) [4, 22]	$\hat{q}$, K, λ, $(\hat{\pi}_b)$
Self-Normalized Doubly Robust (SNDR)	$\hat{q}$, K, $(\hat{\pi}_b)$
Switch Doubly Robust (Switch-DR) [29]	$\hat{q}$, K, τ, $(\hat{\pi}_b)$
Doubly Robust with Optimistic Shrinkage (DRos) [22]	$\hat{q}$, K, λ, $(\hat{\pi}_b)$

Note: $\hat{q}$ is an estimator for the mean reward function 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.

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 is identical to DM, while $\tau \to \infty$ yields DR.

Doubly Robust with Optimistic Shrinkage (DRos). Su et al. [22] proposes DRos based on a new weight function $\hat{\rho} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}_+$ that directly minimizes sharp bounds on the *mean-squared-error* (MSE) of the resulting estimator. DRos is defined as

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

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

Cross-Fitting Procedure. To obtain a reward estimator, $\hat{q}$, we sometimes use *cross-fitting* to avoid the substantial bias that might arise due to overfitting [16]. 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 $k = 1, 2, \ldots K$, we define $\mathcal{D}_k^c := \mathcal{D} \backslash \mathcal{D}_k$.
(2) For each $k = 1, 2, \ldots K$, 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 $K^{-1}\sum_{k=1}^K \hat{V}(\pi_e; \mathcal{D}_k, \hat{q}_k)$.

Hyperparameter Tuning Procedure. As Table 1 summarizes, most OPE estimators have hyperparameters such as λ, τ, K, and $\hat{q}$ that should appropriately be set. Su et al. [22] proposes to select a set of hyperparameters based on the following criterion.

$$\hat{\theta} \in \operatorname*{argmin}_{\theta\in\Theta} \mathrm{BiasUB}(\theta; \mathcal{D})^2 + \mathbb{V}_n(\theta; \mathcal{D}), \tag{1}$$

where $\mathbb{V}_n(\theta; \mathcal{D})$ is the sample variance in OPE, and $\mathrm{BiasUB}(\theta; \mathcal{D})$ is the upper bound of the bias estimated using $\mathcal{D}$. There are several ways to derive the bias upper bound as stated in Su et al. [22]. One way is the direct bias estimation:

$$\mathrm{BiasUB}(\theta; \mathcal{D}) := |\mathbb{E}_n[(\hat{\rho}(x_i, a_i; \theta) - \rho(x_i, a_i))(r_i - \hat{q}(x_i, a_i))]| + \sqrt{\frac{2\mathbb{E}\left[\rho(x, a)^2\right]\log(2/\delta)}{n}} + \frac{2\rho_{\max}\log(2/\delta)}{3n}$$

where $\delta \in (0, 1]$ is the confidence delta to derive the high probability upper bound, and $\rho_{\max} := \max_{x,a}\rho(x, a)$ is the maximum importance weight. $\hat{\rho}(x_i, a_i; \theta)$ is the importance weight modified by a hyperparameter. For example, for IPWps and DRps, $\hat{\rho}(x_i, a_i; \lambda) = \min\{\rho(x_i, a_i), \lambda\}$, and for Switch-DR, $\hat{\rho}(x_i, a_i; \tau) = \rho(x_i, a_i)\mathbb{I}\{\rho(x_i, a_i) \leq \tau\}$.

3 EVALUATING OFFLINE EVALUATION

So far, we have seen that the OPE community has developed a variety of OPE estimators. 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 estimate the following *mean-squared-error* (MSE) as the estimator's performance metric:

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

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; a lower value means a more accurate OPE by $\hat{V}$. Researchers often calculate the MSE of each estimator several times with different random seeds and report its mean.

The issue with this procedure is that most of the estimators have some hyperparameters that should be chosen properly 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 OPE estimators uses only one set of hyperparameters and an arbitrary evaluation policy for each estimator, and then discusses the derived results [1, 6, 24, 27, 29].[3] This type of simplified experimental procedure does not accurately capture the uncertainty in the performance of OPE estimators. Specifically, it

[3]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.

cannot evaluate the robustness to hyperparameter choices and evaluation policy settings, as the reported score 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 entails a higher risk of failure in its particular application. Therefore, we want to avoid using such sensitive estimators as these estimators are more likely to fail. In the next section, we describe an experimental procedure that can evaluate the estimators' robustness to experimental configurations, leading to informative estimator comparisons in OPE research and a reliable estimator selection in practice.

4 INTERPRETABLE EVALUATION FOR OFFLINE EVALUATION

Here, 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 the real-world applicability of an estimator, as it discards information about its robustness 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, as done in some studies on reinforcement learning [5, 9, 10]. 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).[4] $F_Z(z)$ maps a performance metric 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., $\mathcal{Z} := \{z_1, \ldots, z_m\}$, we can estimate the CDF by

$$\hat{F}_Z(z) := \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\{z_i \leq z\}, \tag{2}$$

Using the CDF for evaluating OPE estimators allows researchers to compare different estimators with respect to their robustness to the varying configurations. Specifically, we can use the CDF to evaluate OPE estimators by examining the CDF of the estimators' performance visually or computing some summary scores of the CDF as the estimators' performance metric. For example, we can score an estimator by the *area under the CDF curve* (AU-CDF): AU-CDF$(z_{\max}) := \int_0^{z_{\max}} F_Z(z)dz$. Another possible summary score is *conditional value-at-risk* (CVaR) which computes the expected value of a random variable above a given probability α: $\mathrm{CVaR}_\alpha(Z) := \mathbb{E}[Z \mid Z \geq F_Z^{-1}(\alpha)]$, where $F_Z^{-1}(\alpha) := \underset{z}{\mathrm{argmin}}\{z \mid F_Z(z) \geq \alpha\}$ is the inverse of the CDF. When using CVaR, the 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, we can use standard deviation (Std), $\mathbb{E}[(Z - \mathbb{E}[Z])^2]^{1/2}$, and some other moments such as the skewness of $\hat{F}(z)$ as summary scores.

IEOE with Synthetic or Classification Data. In research papers, it is common to use synthetic or classification data to evaluate OPE estimators [4, 11, 12, 22, 29]. We first present how to apply the IEOE procedure to synthetic or classification data 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. What kind of ϕ we use can change depending on the purpose of the evaluation of OPE. For example, we can use a hyperparameter tuning method for OPE estimators such as the method described in Section 2.2 as ϕ, assuming practitioners use it in real-world applications. When we cannot implement such a hyperparameter tuning method for OPE due to its implementation cost or risk of overfitting, we can be conservative and use the uniform distribution as ϕ in the evaluation of OPE. Next, the IEOE algorithm samples an evaluation policy $\pi_e \in \Pi_e$ from the discrete uniform distribution. Then, 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 metric (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 summary scores to evaluate and compare the estimators' robustness.

IEOE with Real-World Data. It is also possible to apply IEOE to real-world logged bandit data. Algorithm 2 presents IEOE that can be used in real-world applications. To evaluate the performance of $\hat{V}$ with real-world data, we need to prepare several logged bandit feedback datasets $\{\mathcal{D}_j\}_{j=1}^\ell$ where each dataset $\mathcal{D}_j$ is collected by a policy 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. 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 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.

Following Algorithm 2, researchers can benchmark the robustness of OPE estimators using public real-world data. In addition, practitioners can avoid using unstable estimators by applying Algorithm 2 to their own bandit data.

[4] In the following, without loss of generality, we assume that a lower value of Z means more accurate OPE.

Algorithm 1 Interpretable Evaluation for Offline Evaluation (with Classification Data)

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, a set of random seeds $\mathcal{S}$

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

$\mathcal{Z} \leftarrow \emptyset$ ▷ initialize set of results
for $s \in \mathcal{S}$ **do**
  $\theta \leftarrow \phi(\Theta; s)$ ▷ sample a set of hyperparameters
  $\pi_e \leftarrow \mathrm{Unif}(\Pi_e; s)$ ▷ sample an evaluation policy uniformly
  $\mathcal{D}^* \leftarrow \mathrm{Bootstrap}(\mathcal{D}; s)$ ▷ sample logged bandit data *with replacement*
  $z' \leftarrow \mathrm{SE}(\hat{V}; \mathcal{D}^*, \pi_e, \theta)$ ▷ calculate the squared error of $\hat{V}$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z'\}$
end for
Estimate F_Z using $\mathcal{Z}$ (by Eq. 2)

Algorithm 2 Interpretable Evaluation for Offline Evaluation (with Real-World Data)

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, a set of random seeds $\mathcal{S}$

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

$\mathcal{Z} \leftarrow \emptyset$ (initialize set of results)
for $s \in \mathcal{S}$ **do**
  $\theta \leftarrow \phi(\Theta; s)$ ▷ sample a set of hyperparameters based on a given sampler
  $\pi_j \leftarrow \mathrm{Unif}(\Pi_e; s)$ ▷ sample an evaluation policy uniformly
  $\mathcal{D}_{\mathrm{te}} = \mathcal{D}_j$ and $\mathcal{D}_{\mathrm{ev}} = \bigcup_{k=1;k\neq j}^{\ell} \mathcal{D}_j$ ▷ define evaluation and test sets
  $\mathcal{D}^*_{\mathrm{ev}} \leftarrow \mathrm{Bootstrap}(\mathcal{D}_{\mathrm{ev}}; s)$ ▷ sample data from the evaluation set *with replacement*
  $V_{\mathrm{on}}(\pi_j; \mathcal{D}_{\mathrm{te}}) = \mathbb{E}_n[r_i]$ ▷ calculate an on-policy estimate of the policy value with the test set
  $z' \leftarrow \left(V_{\mathrm{on}}(\pi_j; \mathcal{D}_{\mathrm{te}}) - \hat{V}(\pi_j; \theta, \mathcal{D}^*_{\mathrm{ev}})\right)^2$ ▷ calculate the squared error of the estimator
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z'\}$
end for
Estimate F_Z using $\mathcal{Z}$ (by Eq. 2)

5 EXPERIMENTS WITH OPEN BANDIT DATASET

In this section, we use IEOE and evaluate the robustness of a wide variety of OPE estimators on Open Bandit Dataset (OBD)[5]. We run the experiments using our *pyIEOE* software. By using it, anyone can replicate the results easily.[6]

5.1 Setup

OBD is a set of logged bandit feedback datasets collected on a large-scale fashion e-commerce platform provided by Saito et al. [20]. There are three campaigns, "ALL", "Men", and "Women". 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 $r \in \{0, 1\}$. The dimensions of the feature vector x is 20, and the number of actions is 80.

The dataset consists of subsets of data collected by two different policies, the uniform random policy and the Bernoulli Thompson Sampling policy [26]. We let $\mathcal{D}_A$ denote the dataset collected by uniform random policy π_A and $\mathcal{D}_B$ denote that collected by Bernoulli Thompson Sampling policy π_B. We apply Algorithm 2 to obtain a set of SEs as the performance metric of the estimators.

5.2 Estimators and Hyperparameters

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

In the experiment, we use the true behavior policy contained in the dataset to derive importance weights. In this setting, SNIPW is hyperparameter-free, while the other estimators need to be tested for robustness to the choice of the pre-defined hyperparameters and changes in evaluation policies. In addition, we use the hyperparameter tuning method described in Section 2.2 to tune estimator-specific hyperparameters such as λ and τ. Then, we use RandomizedSearchCV implemented in *scikit-learn* with n_iter = 5 to tune hyperparameters of reward estimator $\hat{q}$. Tables 2 and 3 describe hyperparameter spaces Θ for each estimator. Finally, we set $\mathcal{S} = \{0, 1, \ldots, 499\}$.

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

[6] The code to replicate the results is available at: https://github.com/sony/pyIEOE/benchmark . We also provide detailed description of the software in Appendix B.

Table 2: Hyperparameter spaces for OPE estimators

OPE Estimators	Hyperparameter Spaces
DM	$\hat{q} \in$ {LR/RR,RF,LightGBM}, $K \in \{1, 2, \ldots, 5\}$
IPWps	$\lambda \in \{1, 5, 10, 50, \ldots, 10^5 \infty\}$, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
SNIPW	($\hat{\pi}_b \in$ {LR,RF,LightGBM})
DRps	$\hat{q} \in$ {LR/RR,RF,LightGBM}, $K \in \{1, 2, \ldots, 5\}$, $\lambda \in \{1, 5, 10, 50, \ldots, 10^5 \infty\}$, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
SNDR	$\hat{q} \in$ {LR/RR,RF,LightGBM}, $K \in \{1, 2, \ldots, 5\}$, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
Switch-DR	$\hat{q} \in$ {LR/RR,RF,LightGBM}, $K \in \{1, 2, \ldots, 5\}$, $\tau \in \{1, 5, 10, 50, \ldots, 10^5 \infty\}$, ($\hat{\pi}_b \in$ {LR,RF,LightGBM})
DRos	$\hat{q} \in$ {LR/RR,RF,LightGBM}, $K \in \{1, 2, \ldots, 5\}$, $\lambda \in \{1, 5, 10, 50, \ldots, 10^5 \infty\}$, ($\hat{\pi}_b \in$ {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. We estimate the behavior policy only in the experiments with classification data in Appendix A. Therefore, π_b is in parentheses. $K = 1$ means that we do not use cross-fitting and train $\hat{q}$ on the whole $\mathcal{D}_{\mathrm{ev}}$.

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

Machine Learning Models	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: We follow the *scikit-learn* package as to the names of the hyperparameters As default, we use max_iter = 10,000 for LogisticRegression, n_estimators = 100 for RandomForest, and max_iter = 100 for LightGBM.

Figure 2: Comparison of the CDF of OPE estimators' squared error in Open Bandit Dataset

5.3 Results

Figure 2 visually compares the CDF of the estimators' squared error. Table 4 reports AU-CDF, $\mathrm{CVaR}_{0.7}$, and Std as summary scores.

When the dataset size is small ($n = 30,000$), we see that the typical way of reporting only the mean of the squared error cannot tell which estimator is accurate or robust. However, some other summary scores show that DM has more robust and stable estimation performance than other estimators, having lower $\mathrm{CVaR}_{0.7}$ and Std. Moreover, Figure 2 provides more detailed information about the estimators' performance. Specifically, DM performs better in the worst case while the other estimators show better performance in the region where squared error is lower than 0.2. Thus, when we are conservative and prioritize the worst case performance, DM is the most appropriate choice. Otherwise, other estimators might be a better choice. We cannot obtain this conclusion by comparing only the mean (typical metric) of the squared error.

When the dataset size is large ($n = 300,000$), we confirm that IPWps and SNIPW are more accurate than other model-based estimators. In particular, Figure 2 shows that IPWps performs better than other estimators in all region, meaning that we should use it whether we prioritize the best or the worst case performance.

Overall, the results indicate that an appropriate estimator can drastically change depending on the situation such as the data size.

Table 4: Summary scores of the OPE estimators on Open Bandit Dataset with different sample size

	$n = 30,000$				$n = 300,000$			
OPE Estimators	**Mean** (typical metric)	**AU-CDF**	**CVaR**$_{0.7}$	**Std**	**Mean** (typical metric)	**AU-CDF**	**CVaR**$_{0.7}$	**Std**
DM	**1.00***	**1.000***	**1.00***	**1.00***	**10.77**†	**0.186**†	**7.45**†	**6.94**†
IPWps	**1.02**$^{\diamond}$	**0.994**$^{\diamond}$	**1.19**$^{\diamond}$	**1.31**$^{\diamond}$	**1.00***	**1.000***	**1.00***	**1.00***
SNIPW	**1.02**$^{\diamond}$	**0.994**$^{\diamond}$	1.20	1.33	**1.58**$^{\diamond}$	**0.917**$^{\diamond}$	**1.57**$^{\diamond}$	**1.71**$^{\diamond}$
DRps	**1.04**†	0.989	**1.27**†	1.44	2.48	0.887	2.67	5.02
SNDR	**1.02**$^{\diamond}$	0.995	**1.27**†	1.44	3.27	0.827	3.50	6.01
Switch-DR	**1.02**$^{\diamond}$	**0.994**$^{\diamond}$	**1.27**†	**1.45**†	3.28	0.825	3.50	6.00
DRos	**1.02**$^{\diamond}$	**0.994**$^{\diamond}$	**1.27**†	**1.45**†	3.28	0.825	3.50	6.00

Note: Larger value is better for **AU-CDF** and lower value is better for **Mean**, **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 $n = 30,000$ and $z_{\max} = 1.0 \times 10^{-6}$ for $n = 300,000$ 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.

Therefore, we argue that identifying a reasonable estimator before conducting OPE is essential in practice. Moreover, we demonstrate that the IEOE procedure can provide more informative insight as to the estimators' performance compared to the typical metric.

6 REAL-WORLD APPLICATION

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

6.1 Setup

To show how to use IEOE 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 a 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 overfitting due to the intensive hyperparameter tuning of OPE estimators, as the size of the logged bandit feedback data is not large. Moreover, the data scientists want to use an OPE estimator to evaluate the performance of several candidate policies. Therefore, we aim to find an estimator that performs stably for a wide range of evaluation policies with fewer hyperparameters.

6.2 Performance Metric

To apply our evaluation procedure, we need to define a performance metric (in step 8 of Algorithm 2). 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 the evaluation policy, we define $\mathcal{D}_{\mathrm{ev}} = \mathcal{D}_B \cup \mathcal{D}_C$ and $\mathcal{D}_{\mathrm{te}} = \mathcal{D}_A$. Then, by applying Algorithm 2, we obtain a set of SEs to evaluate the robustness and real-world applicability of the estimators.

6.3 Estimators and Hyperparameters

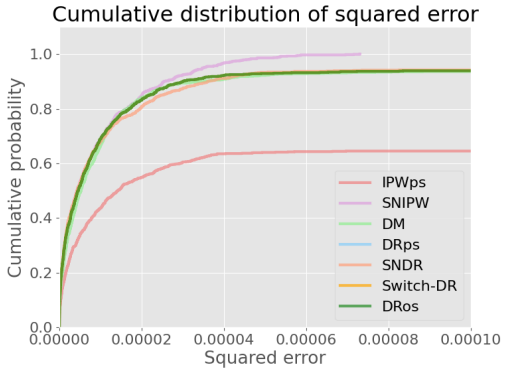
We use the IEOE protocol to evaluate the robustness of DM, IPWps, SNIPW, DRps, SNDR, Switch-DR, and DRos. Then, we utilize the experimental results to help the data scientists of the platform choose an appropriate estimator.

During the data collection experiment, we logged the true action choice probabilities of the three policies, and thus SNIPW is hyperparameter-free. We use the hyperparameter spaces defined in Tables 2 and 3 for our real-world application. In addition, we use the hyperparameter tuning method described in Section 2.2 to tune estimator-specific hyperparameters such as λ and τ. Then, we use the uniform distribution as ϕ to sample hyperparameters of reward regression model $\hat{q}$. Finally, we set $\mathcal{S} = \{0, 1, \ldots, 999\}$ and $\Pi_e = \{\pi_A, \pi_B, \pi_C\}$.

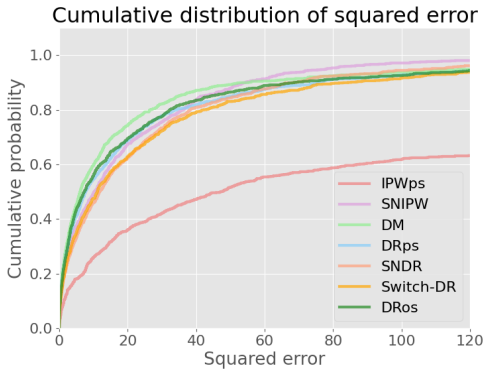
6.4 Results

We applied Algorithm 2 to the above estimators for the binary and continuous outcome data, respectively.

Figure 3 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, IPWps performs poorly for both outcomes, because our dataset is not large and some behavior policies are near deterministic, making IPWps an unstable estimator. Moreover, Switch-DR fails to accurately



binary



continuous

Figure 3: Comparison of the CDF of OPE estimators' squared error in the real-world application

Table 5: Summary scores of the OPE estimators in the real-world application

	Binary Outcome				Continuous Outcome			
OPE Estimators	**Mean** (typical metric)	**AU-CDF**	**CVaR**$_{0.7}$	**Std**	**Mean** (typical metric)	**AU-CDF**	**CVaR**$_{0.7}$	**Std**
DM	8.70	0.946	10.92	**35.94**†	1.29	**1.000***	1.47	2.19
IPWps	**29.45**†	**0.648**†	**31.96**†	**29.84**$^{\diamond}$	**19.00**†	**0.572**†	**19.84**†	**14.67**†
SNIPW	**1.00***	**1.000***	**1.00***	**1.00***	**1.00***	**0.974**$^{\diamond}$	**1.00***	**1.00***
DRps	8.16	**0.953**$^{\diamond}$	10.27	34.54	1.44	0.957	1.60	2.11
SNDR	**7.45**$^{\diamond}$	0.942	**9.35**$^{\diamond}$	32.19	1.21	0.935	**1.22**$^{\diamond}$	**1.17**$^{\diamond}$
Switch-DR	8.16	**0.953**$^{\diamond}$	10.27	34.54	1.48	0.919	1.57	1.68
DRos	8.16	**0.953**$^{\diamond}$	10.27	34.54	1.43	0.968	1.60	2.21

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 **Mean**, **CVaR**, and **Std**. Note that we normalize the scores by dividing them by the best score among all estimators. We use $z_{\max} = 1.0 \times 10^{-4}$ for the binary outcome and $z_{\max} = 1.0 \times 10^{2}$ for the continuous outcome 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.

evaluate the performance of the evaluation policies. Thus, it is unsafe to use these estimators in our application, even though we tune their hyperparameters (λ or τ).

We additionally confirm the above observations in a quantitative manner. For both binary and continuous outcomes, we compute AU-CDF, CVaR$_{0.7}$, and Std of the squared error for each OPE estimator. We report these summary scores in Table 5, and the results demonstrate that SNIPW clearly outperforms other estimators in almost all situations. In particular, SNIPW is the best with respect to CVaR$_{0.7}$ and Std for both binary and continuous outcomes, showing that this estimator is the most stable estimator in our environment. Moreover, SNIPW is hyperparameter-free, and overfitting is less likely to occur compared to other estimators having some hyperparameters to be tuned. Through this evaluation of OPE estimators, we concluded that ***the e-commerce platform should use SNIPW for its offline evaluation.*** After comprehensive accuracy and stability verification, 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 OPE cannot evaluate the robustness of the estimators' performance. Instead, the IEOE procedure can provide an interpretable way to evaluate 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 OPE estimators so that practitioners can spend more time on real decision making problems, and OPE researchers can focus more on tackling advanced technical questions. We perform an extensive evaluation of a wide variety of OPE estimators and demonstrated that our experiments are more informative than a typical procedure, showing which estimators are more sensitive to configuration changes. Finally, we applied our procedure to a real-world application and demonstrated its practical usage.

Although our procedure is useful to evaluate the robustness of estimators, we need to prepare at least two logged bandit feedback datasets collected by different policies to apply it to real-world applications, as described in Algorithm 2. Thus, it would be beneficial

to construct a procedure to enable the evaluation of OPE estimators with only logged bandit data collected by a single policy.

ACKNOWLEDGMENTS

The authors would like to thank Masahiro Nomura, Ryo Kuroiwa, and Richard Liu for their help in reviewing the paper. Additionally, we would like to thank the anonymous reviewers for their constructive reviews and discussions.

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] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. 2020. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. In *International Conference on Learning Representations*.

[6] 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.

[7] 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.

[8] 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.

[9] Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip Thomas. 2020. Evaluating the performance of reinforcement learning algorithms. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 4962–4973.

[10] Scott M Jordan, Daniel Cohen, and Philip S Thomas. 2018. Using cumulative distribution based performance analysis to benchmark models. In *NeurIPS 2018 Workshop on Critiquing and Correcting Trends in Machine Learning*.

[11] Nathan Kallus, Yuta Saito, and Masatoshi Uehara. 2021. Optimal Off-Policy Evaluation from Multiple Logging Policies. In *Proceedings of the 38th International Conference on Machine Learning*, Vol. 139. PMLR, 5247–5256.

[12] 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.

[13] 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.

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

[15] 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.

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

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

[18] 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).

[19] Yuta Saito. 2020. Doubly robust estimator for ranking metrics with post-click conversions. In *Fourteenth ACM Conference on Recommender Systems*. 92–100.

[20] 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).

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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).

[29] 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.

Table 6: Classification datasets used in the benchmark experiment

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. **#Actions** is the total number of actions (or classes). **#Dimensions** is the number of dimensions of the context (or feature) vector.

Table 7: Behavior and evaluation policies used in the benchmark experiment

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.

A BENCHMARK EXPERIMENTS ON CLASSIFICATION DATASETS

Here, we conduct experiments on three classification datasets, OptDigits, PenDigits, and SatImage provided at the UCI repository [3]. Table 6 shows some statistics of the datasets used in the benchmark experiment.

A.1 Setup

Following previous studies [4, 6, 11, 29], 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:

$$\pi(a \mid x) = \alpha \cdot \mathbb{I}\{\pi_{det}(x) = a\} + (1 - \alpha) \cdot \pi_u(a),$$

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. Then, 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 metric of $\hat{V}$, i.e., $\mathrm{SE}(\hat{V}; \mathcal{D}^*, \pi_e, \theta) := (\hat{V}(\pi_e; \mathcal{D}^*, \theta) - V(\pi_e; \mathcal{D}_{\mathrm{te}}))^2$.

A.2 Estimators and Hyperparameters

We use IEOE to evaluate the robustness of DM, IPWps, SNIPW, DRps, SNDR, Switch-DR, and DRos.

Here, we run the experiments under two different settings. First, we test the case where the true behavior policy π_b is available. Next, we investigate the OPE estimators with 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 the choice of machine learning method to obtain $\hat{\pi}_b$.

Tables 2 and 3 (in the main text) describe hyperparameter spaces Θ for each estimator. 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 when estimating the behavior policy, as calibrating the behavior policy estimator matters in OPE [18]. Then, we use the hyperparameter tuning method described in Section 2.2 to tune estimator-specific

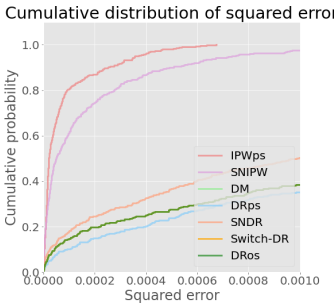
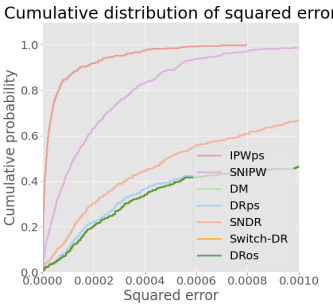
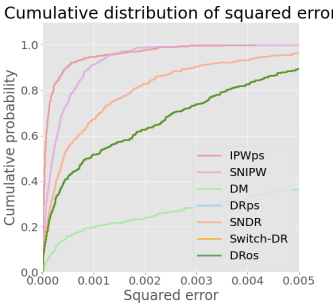
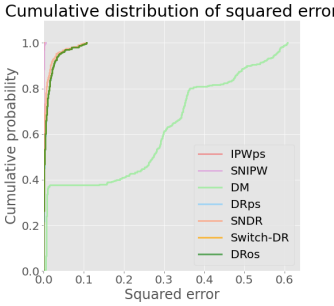
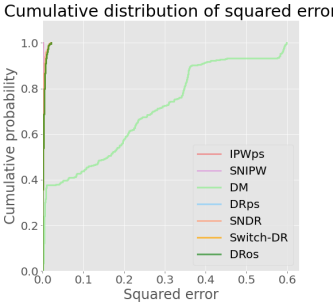
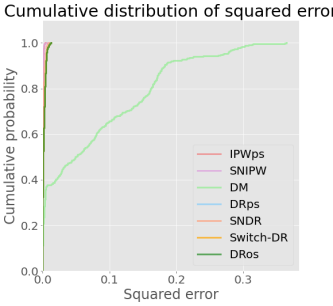
	Datasets		
	Optdigits	Pendigits	Satimage
zoom			
full			

Figure 4: Comparison of the CDF of OPE estimators' squared error (true behavior policy)

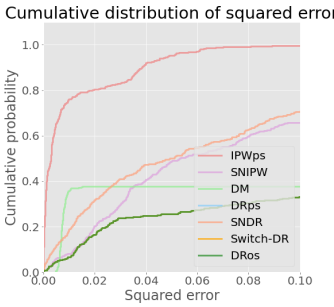
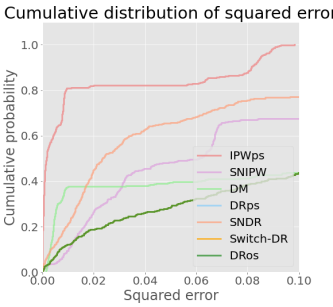
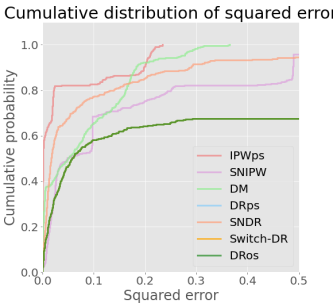
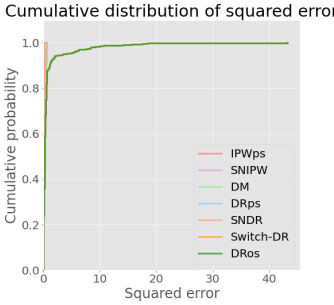
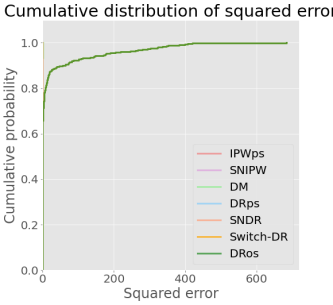
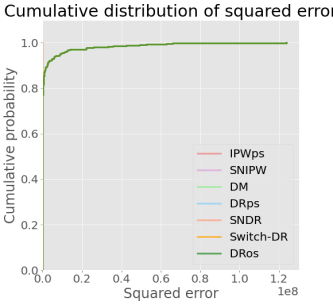
zoom			
full			

Figure 5: Comparison of the CDF of OPE estimators' squared error (estimated behavior policy)

hyperparameters such as λ and τ. Table 7 describes how we construct the true behavior policy and five different evaluation policies in Π_e. Finally, we set $\mathcal{S} = \{0, 1, \ldots, 499\}$.

Table 8: Summary scores of the OPE estimators (true behavior policy)

	OptDigits			PenDigits			SatImage		
OPE Estimators	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**
DM	**0.000**†	**2215.37**†	**1631.15**†	**0.000**†	**2609.92**†	**1674.67**†	**0.266**†	**323.81**†	**184.39**†
IPWps	**1.000***	**1.00***	**1.00***	**1.000***	**1.00***	**1.00***	**1.000***	**1.00***	**1.00***
SNIPW	**0.907**$^{\diamond}$	**2.51**$^{\diamond}$	**3.15**$^{\diamond}$	**0.843**$^{\diamond}$	**3.16**$^{\diamond}$	**2.18**$^{\diamond}$	**0.966**$^{\diamond}$	**1.56**$^{\diamond}$	**1.16**$^{\diamond}$
DRps	0.249	127.61	139.99	0.358	45.49	33.59	0.686	7.99	4.78
SNDR	0.374	96.91	125.92	0.480	27.47	26.15	0.833	4.71	3.17
Switch-DR	0.287	126.21	139.12	0.347	45.46	33.52	0.686	7.99	4.78
DRos	0.287	126.21	139.12	0.347	45.46	33.52	0.686	7.99	4.78

Note: Larger value is better for **AU-CDF** and lower value is better for **CVaR** and **Std**. Note that we normalize the summary scores by dividing them by the best score among all estimators. We use $z_{\max} = 1.0 \times 10^{-3}$ for OptDigits and Pendigits and $z_{\max} = 5.0 \times 10^{-3}$ for SatImage 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.

Table 9: Summary scores of the OPE estimators (estimated behavior policy)

	OptDigits			PenDigits			SatImage		
OPE Estimators	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**	**AU-CDF**	**CVaR$_{0.7}$**	**Std**
DM	0.390	14.18	9.89	0.454	7.84	5.86	**0.911**$^{\diamond}$	**1.62**$^{\diamond}$	**1.20**$^{\diamond}$
IPWps	**1.000***	**1.00***	**1.00***	**1.000***	**1.00***	**1.00***	**1.000***	**1.00***	**1.00***
SNIPW	0.460	**9.06**$^{\diamond}$	7.05	0.538	7.22	5.23	0.778	3.23	2.52
DRps	**0.262**†	**71.18**†	**141.25**†	**0.332**†	**1527.07**†	**2498.85**†	**0.657**†	**5.27×10^{7}**†	**1.25×10^{8}**†
SNDR	**0.525**$^{\diamond}$	7.57	**6.07**$^{\diamond}$	**0.700**$^{\diamond}$	**4.28**$^{\diamond}$	**3.79**$^{\diamond}$	0.904	2.40	2.52
Switch-DR	**0.262**†	**71.18**†	**141.25**†	**0.332**†	**1527.07**†	**2498.85**†	**0.657**†	**5.27×10^{7}**†	**1.25×10^{8}**†
DRos	**0.262**†	**71.18**†	**141.25**†	**0.332**†	**1527.07**†	**2498.85**†	**0.657**†	**5.27×10^{7}**†	**1.25×10^{8}**†

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 $z_{\max} = 0.1$ for OptDigits and Pendigits and $z_{\max} = 0.5$ for SatImage 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.

A.3 Results

Figures 4 and 5 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, CVaR$_{0.7}$, and Std of the squared error of each OPE estimator. We report these summary scores in Tables 8 and 9.

First, in the setting where the true behavior policy is available, it is obvious that IPWps is the best estimator and achieves the most accurate estimation in almost all regions (see Figure 4). SNIPW also performs comparably better than other estimators. In contrast, model-dependent estimators, especially DM, perform poorly compared to the typical estimators such as IPWps and SNIPW. 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 IPWps or SNIPW. Therefore, our experimental procedure poses a possibility that simple estimators with fewer hyperparameters tend to perform well and be robust for a wide variety of settings when the true behavior policy is recorded.

In the setting where the behavior policy needs to be estimated, we observe similar trends. First, Figure 5 and Table 9 show that IPWps achieves the most accurate estimation even when it uses the estimated behavior policy. Second, estimators based on DR such as DRps, Switch-DR, and DRos show considerably large squared errors when the behavior policy is estimated. This is because DR is vulnerable to the overfitting of $\hat{\pi}_b$. DR produces large squared errors when $\hat{\pi}_b$ overfits 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 these estimators also become large, amplifying the estimation error of reward estimator $\hat{q}$. This leads to serious overestimation of the policy value of π_e, even though the cut-off hyperparameters (λ and τ) are properly tuned.

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

B SOFTWARE IMPLEMENTATION

In addition to developing the evaluation procedure, we have implemented open-source Python software, ***pyIEOE***, to streamline the evaluation of OPE with our experimental protocol. This package is built with the intention of being used with ***OpenBanditPipeline*** (obp).[7]

Below, we show the essential codes to conduct an interpretable evaluation of 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 Algorithms 1 and 2 except for some preparations.

```
# 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(), # DRos
           SelfNormalizedDoublyRobust(), # SNDR
           ..,
       ],
       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 of each OPE estimator
>>> evaluator.visualize_cdf_aggregate() # plot CDF curves
```

Code Snippet 1: Essential Codes for Interpretable OPE Evaluation

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

B.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.

In addition to synthetic dataset, users can utilize multi-class classification data, public real-world data (such as Open Bandit Dataset [20]), and their own real-world data to evaluate the robustness of OPE estimators by following preprocessing procedure of obp. Users are also free to define a set of evaluation policies by themselves.

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

```
# 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

B.2 Defining Hyperparameter Spaces

After preparing the synthetic data and a set of evaluation policies, we now define hyperparameter spaces of OPE estimators. Users of the software can define hyperparameter spaces of OPE estimators by themselves as follows.

```
# 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

B.3 Interpretable OPE Evaluation

Finally, we evaluate OPE estimators in an interpretable manner. Our software provides an easy procedure to conduct this evaluation of OPE workflow.

Users can intuitively evaluate the robustness of the estimators by comparing the CDF of the squared error. The quantitative comparison is also possible by calculating some summary scores such as AU-CDF and CVaR. In this case, it is easy to figure out that SNDR is more reliable than DRos.

```
# 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