AppLovin Bid Floor Optimization: A deep dive into the Algorithms by Metica

Pietro Abruzzo

This paper analyzes the algorithms used by AppLovin and Metica to optimize bid floor prices in real-time digital advertising auctions. Understanding these algorithms helps to explain the competitive dynamics and convergence behavior of systems operating in the same advertising environment.

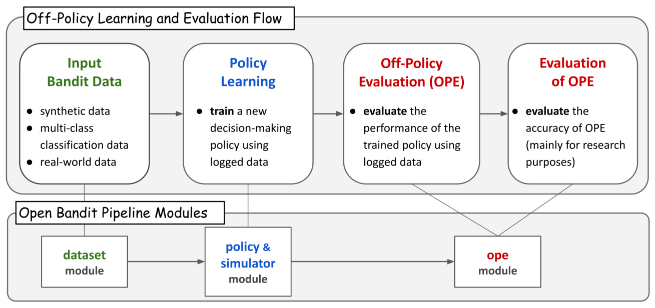
Off-Policy Learning and Evaluation Flow
Input Bandit Data
synthetic data
multi-class classification data
real-world data
Policy Learning
train a new decision-making policy using logged data
Off-Policy Evaluation (OPE)
evaluate the performance of the trained policy using logged data
Evaluation of OPE
evaluate the accuracy of OPE (mainly for research purposes)
Open Bandit Pipeline Modules
dataset module
policy & simulator module
ope module

FIG. 1. OBP Workflow

I. INTRODUCTION

AppLovin recently began a strategic collaboration with Metica[1]. After reviewing the repositories hosted on Metica's GitHub organization, I found that they had forked a source repository containing several CTR modeling algorithms and optimization frameworks. Based on this, I infer that Metica integrates and uses these components in its development and operations. This paper focuses on analyzing the inference and its implications.

II. OPEN BANDIT PIPELINE: A RESEARCH FRAMEWORK FOR OFF-POLICY EVALUATION AND LEARNING

OBP[2] appears to be a pipeline used by Metica for Bid Floor optimization through algorithmic decision-making and real-time bid management.

A. The OBP workflow

Before explaining the algorithms, I will first describe the overall process and workflow used in OBP.

Figure 1 shows the data flow within OBP.

As shown, the dataset feeds into the policy learning module, which will be explained in detail later. Using the logged interaction data, OBP trains a new decision-making policy through offline reinforcement learning. It then evaluates the performance of this trained policy using the same logged data. The final evaluation step is optional, but it can be useful for future system improvements.

1. Input Bandit Data Block

The Input Bandit Data Block represents the raw logged interaction data used in the Open Bandit Pipeline (OBP)[3]. It captures all relevant information about past decisions made by the behavior policy in a structured format. Each entry corresponds to a single interaction and contains the following components:

- **Context (x_t):** A feature vector describing the environment at the time of the decision. Contexts can include user characteristics, device information, ad or campaign attributes, temporal features (e.g., time of day, day of week), and other relevant variables. The context is typically represented as a fixed-size numeric vector or a structured object suitable for input into models.
- **Action (a_t):** The action taken by the behavior policy during the interaction. In advertising applications, this corresponds to a specific bid floor value or other decision made in the auction context. Actions are generally encoded as discrete indices mapping to real-world decisions.
- **Reward (r_t):** The observed outcome after the action was executed, quantifying the success or utility of that action in the given context. In digital advertising, rewards can be binary (e.g., click/no-click), categorical, or continuous (e.g., revenue generated from a click).
- **Propensity Score ($\pi_b(a_t \mid x_t)$):** The probability that the behavior policy would select the observed action given the context. Propensity scores are stored for each interaction to allow for unbiased estimation in later evaluation or learning procedures.
- **Interaction Index / Round (t):** A sequential identifier for each logged interaction. This index allows interactions to be tracked and referenced individually.

[1] https://metica.com/blog/optimize-max-for-30-more
[2] https://github.com/meticalabs/off-policy-evaluation
[3] https://github.com/meticalabs/off-policy-evaluation/tree/main/obp/dataset

- **Additional Metadata:** The block may include auxiliary fields such as user IDs, campaign IDs, timestamp of the interaction, total number of possible actions (K), and other relevant identifiers. These fields assist in data organization, filtering, and ensuring consistency across interactions.

All components are stored in a standardized format, typically as structured arrays, dictionaries, or tensors, to allow efficient access and processing. The Input Bandit Data Block is self-contained and provides a complete snapshot of the environment, actions taken, and outcomes observed during the logged interactions, forming the essential foundation for any subsequent offline analysis or model input.

2. Policy Learning Block

The Policy Learning Block is the central module of OBP responsible for constructing new decision-making policies based on logged bandit feedback. Given the input tuples

$$D = \{(x_i, a_i, r_i, p_i)\}_{i=1}^{N},$$

where $x_i \in \mathcal{X}$ represents the context, $a_i \in \mathcal{A}$ the action taken by the behavior policy π_b, $r_i \in \mathbb{R}$ the observed reward, and $p_i = \pi_b(a_i|x_i)$ the probability of taking that action, the goal of the Policy Learning Block is to derive a new target policy $\pi_\theta(a|x)$ that maximizes the expected reward under the current data distribution.

Formally, the policy is a mapping

$$\pi_\theta(a|x) : \mathcal{X} \to \Delta(\mathcal{A}),$$

where $\Delta(\mathcal{A})$ is the probability simplex over all actions. The parameters θ are optimized according to the learning algorithm chosen by the system.

The OBP implementation provides multiple classes of policy learners, each corresponding to a different optimization strategy. Below, we describe the mathematical structure of these algorithms.

a. (1) ϵ-Greedy Policy A simple exploration–exploitation scheme defined as:

$$\pi(a|x) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}, & \text{if } a = \arg\max_{a'} Q(a'|x), \\ \frac{\epsilon}{|\mathcal{A}|}, & \text{otherwise,} \end{cases}$$

where $Q(a|x)$ is the estimated reward function and $\epsilon \in [0, 1]$ controls the degree of exploration.

b. (2) Softmax (Boltzmann) Policy To obtain a differentiable stochastic policy, the Softmax form is used:

$$\pi(a|x) = \frac{\exp(Q(a|x)/\tau)}{\sum_{a'} \exp(Q(a'|x)/\tau)},$$

where $\tau > 0$ is the temperature parameter. Training is performed by minimizing the negative log-likelihood:

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \pi_\theta(a_i|x_i).$$

c. (3) Logistic Regression Policy For discrete actions, OBP implements a multinomial logistic policy:

$$\pi_\theta(a|x) = \frac{\exp(\theta_a^\top x)}{\sum_{a'} \exp(\theta_{a'}^\top x)}.$$

The optimization objective is to maximize the likelihood of observed actions, equivalently minimizing the cross-entropy loss:

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \pi_\theta(a_i|x_i).$$

d. (4) Neural Network Policy To capture non-linear dependencies, the logistic policy is generalized using a neural function approximator:

$$\pi_\theta(a|x) = \mathrm{Softmax}(f_\theta(x)),$$

where $f_\theta(x)$ is typically a two-layer feed-forward network:

$$f_\theta(x) = W_2 \sigma(W_1 x + b_1) + b_2.$$

The loss is given by

$$L(\theta) = -\sum_i \log \pi_\theta(a_i|x_i) - \lambda \sum_i H(\pi_\theta(\cdot|x_i)),$$

where $H(\pi)$ is the entropy of the policy and λ controls the level of exploration regularization.

e. (5) Inverse Propensity Weighting (IPW) Optimization To evaluate policies off-policy, the importance sampling estimator is used:

$$\hat{V}_{\mathrm{IPW}}(\pi) = \frac{1}{N} \sum_{i=1}^{N} \frac{\pi(a_i|x_i)}{\pi_b(a_i|x_i)} r_i.$$

The gradient with respect to policy parameters is

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{r_i}{\pi_b(a_i|x_i)} \nabla_\theta \pi_\theta(a_i|x_i).$$

To reduce variance, OBP clips the importance ratio:

$$w_i = \min\left(\frac{\pi(a_i|x_i)}{\pi_b(a_i|x_i)}, c \right),$$

where c is a tunable clipping constant.

f. (6) Direct Method (DM) and Doubly Robust (DR) Optimization The Direct Method estimates a reward model $\hat{Q}(x, a)$ via regression and computes:

$$\hat{V}_{\mathrm{DM}}(\pi) = \frac{1}{N} \sum_{i=1}^{N} \sum_{a} \pi(a|x_i) \hat{Q}(x_i, a).$$

The Doubly Robust estimator combines IPW and DM for improved bias–variance tradeoff:

$$\hat{V}_{\mathrm{DR}}(\pi) = \frac{1}{N} \sum_{i=1}^{N} \left[\hat{Q}(x_i, \pi) + \frac{\pi(a_i|x_i)}{\pi_b(a_i|x_i)} (r_i - \hat{Q}(x_i, a_i)) \right].$$

If either π_b or $\hat{Q}$ is well estimated, $\hat{V}_{\mathrm{DR}}$ remains unbiased.

g. (7) Bayesian and Thompson Sampling Policies For settings with uncertain reward estimates, OBP supports posterior sampling. The parameters are drawn from a posterior distribution $p(\theta|D)$, and the policy selects:

$$a = \arg\max_a Q(x, a; \theta), \quad \theta \sim p(\theta|D),$$

thus achieving a Bayesian exploration–exploitation balance.

Overall, the Policy Learning Block integrates these algorithms into a unified interface, enabling systematic comparison and evaluation under a shared data model. Each policy class exposes the methods `fit()`, `predict()`, and `sample_action()`, facilitating modular experimentation and reproducibility in off-policy evaluation settings.

B. Off-Policy Evaluation (OPE) Block

The Off-Policy Evaluation (OPE) block in `obp/ope` is the statistical and mathematical core of the framework. While the Policy Learning Block is concerned with optimizing the parameters of a new policy $\pi_e(a|x)$, the OPE block aims to estimate its performance using only historical, logged data collected by a different policy $\pi_b(a|x)$, without executing π_e in the real environment. Formally, the task is to estimate the value function

$$V^{\pi_e} = \mathbb{E}_{x\sim p(x)}\left[\mathbb{E}_{a\sim\pi_e(\cdot|x)}[r(x,a)]\right], \tag{1}$$

given samples $(x_i, a_i, r_i, p_i)_{i=1}^n$, where $p_i = \pi_b(a_i|x_i)$ denotes the behavior policy's action probability.

The estimators implemented in `obp/ope` represent the main families of OPE approaches: importance weighting, direct modeling, and hybrid estimators. Each algorithm provides a trade-off between bias, variance, and robustness to model misspecification.

a. Inverse Probability Weighting (IPW) The simplest unbiased estimator assumes access to accurate logging propensities p_i. It reweights observed rewards by the ratio between evaluation and behavior policies:

$$\widehat{V}_{\text{IPW}} = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi_e(a_i|x_i)}{\pi_b(a_i|x_i)} r_i. \tag{2}$$

While $\widehat{V}_{\text{IPW}}$ is unbiased under correct propensities, its variance grows exponentially with small $\pi_b(a_i|x_i)$. This issue is mitigated through the **self-normalized** variant (`SNIPW`):

$$\widehat{V}_{\text{SNIPW}} = \frac{\sum_{i=1}^n w_i r_i}{\sum_{i=1}^n w_i}, \quad \text{where } w_i = \frac{\pi_e(a_i|x_i)}{\pi_b(a_i|x_i)}. \tag{3}$$

Normalization stabilizes variance but introduces a small bias.

b. Direct Method (DM) The DM estimator models the conditional reward function $q(x,a) = \mathbb{E}[r|x,a]$ using regression or neural approximators. It computes

$$\widehat{V}_{\text{DM}} = \frac{1}{n}\sum_{i=1}^{n}\sum_{a\in\mathcal{A}} \pi_e(a|x_i)\widehat{q}(x_i, a), \tag{4}$$

which is low-variance but potentially biased when $\widehat{q}$ is misspecified. In practice, DM provides strong baseline estimates in well-modeled environments.

c. Doubly Robust (DR) Estimator The DR estimator combines IPW and DM to achieve robustness under partial misspecification:

$$\widehat{V}_{\text{DR}} = \frac{1}{n}\sum_{i=1}^{n}\left[\widehat{q}(x_i,\pi_e) + \frac{\pi_e(a_i|x_i)}{\pi_b(a_i|x_i)}(r_i - \widehat{q}(x_i,a_i))\right], \tag{5}$$

where $\widehat{q}(x_i,\pi_e) = \sum_a \pi_e(a|x_i)\widehat{q}(x_i,a)$. The first term contributes low-variance prediction, while the second corrects for potential bias. This estimator is *doubly robust* because it is unbiased if either the model $\widehat{q}$ or the logging policy π_b is correctly specified.

d. Switch-DR and DRos The module also includes variants such as the **Switch-DR** and **DRos (Doubly Robust with Optimized Shrinkage)** estimators, which address the instability of large importance weights. Switch-DR introduces a threshold τ:

$$\widehat{V}_{\text{SwitchDR}} = \frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{1}(w_i \le \tau)\,\widehat{V}_{\text{DR},i} + \mathbf{1}(w_i > \tau)\,\widehat{V}_{\text{DM},i}\right], \tag{6}$$

thereby interpolating between DR and DM depending on the reliability of importance weights. DRos further minimizes the mean squared error by adaptively shrinking large weights using optimization techniques derived from the asymptotic variance bound.

e. Model Selection and Confidence Intervals To quantify statistical reliability, the OPE block includes bootstrapped confidence intervals and bias–variance diagnostics. Given an estimator $\widehat{V}$, a bootstrap-based CI can be computed as

$$\text{CI}_{1-\alpha} = \left[\widehat{V}^{(B)}_{(\alpha/2)}, \ \widehat{V}^{(B)}_{(1-\alpha/2)}\right], \tag{7}$$

where $\widehat{V}^{(B)}_{(p)}$ denotes the p-th empirical quantile from B bootstrap replicates. This enables robust statistical testing for policy improvement claims.

f. Summary In essence, the `obp/ope` module transforms static logged bandit feedback into statistically valid policy performance estimates. It bridges the gap between offline learning and online deployment by quantifying the expected gain of a candidate policy π_e while maintaining rigorous control over estimation bias and variance. These estimators form the empirical backbone of safe policy optimization pipelines in industrial-scale recommender and advertising systems.

III. CAVEATS OF OBP

In it's current form, the OBP is more of a research-grade pipeline instead of an AdTech Bid Floor Optimization pipeline. The exact pipeline by Metica isn't public, but I expect that they took this repository and adapted it for their purposes. While the theory and models can stay partially the same, the code in the Github repository is not industry-ready, because of several factors:

- Pytorch is rather slow for large amounts of data. In these cases, it would be better to use libraries like Tensorflow.
- It is adapted for parameters other than those needed in AdTech industries.
- The connection between the Client Side[4] and the ML algorithms is not shown in any of the repositories, so the supply path should be completely redesigned and adapted to the company systems.

IV. CONCLUSION

The OBP is a highly useful and well-structured optimization pipeline. Naturally, it has certain caveats—some of which are discussed in the final section—but these are relatively minor issues that can be addressed with straightforward improvements.[5]

[4] https://github.com/meticalabs/smart-floors-external

[5] For more information about the OBP pipeline please reference to the Github repository(https://github.com/meticalabs/off-policy-evaluation) and to the main paper(https://arxiv.org/pdf/2008.07146)