Machine Learning (2021) 110:2577–2602
https://doi.org/10.1007/s10994-021-05978-9

Lessons on off-policy methods from a notification component of a chatbot

Scott Rome[1] · **Tianwen Chen**[2] · **Michael Kreisel**[2] · **Ding Zhou**[3]

Received: 24 February 2020 / Revised: 6 March 2021 / Accepted: 12 April 2021 /
Published online: 4 May 2021

Abstract
This work serves as a review of our experience applying off-policy techniques to train and evaluate a contextual bandit model powering a troubleshooting notification in a chatbot. First, we demonstrate the effectiveness of off-policy evaluation when data volume is orders of magnitude less than typically found in the literature. We present our reward function and choices behind its design, as well as how we construct our logging policy to balance exploration and performance on key metrics. Next, we present a guided framework to update a model post-training called Post-Hoc Reward Distribution Hacking, which we employed to improve model performance and correct deficiencies in trained models stemming from the existence of a null action and a noisy reward signal. Throughout the work, we include discussions of various practical pitfalls encountered while using off-policy methods in hopes to expedite other applications of these techniques.

Keywords Contextual bandits · Off-policy training · Off-policy evaluation · Limited data · Small data

1 Introduction

In this paper, we present our experience developing a contextual bandit model to display notifications, known as predictive cards, in a customer support application. When the customer logs into the application, they can be presented with a notification card for troubleshooting either their internet or television service with the goal of streamlining the troubleshooting process. Our training data consists of logs from the production system, and our methodology is taken from the off-policy learning and evaluation literature. We successfully used off-policy methods to continually develop models which

Editors: Yuxi Li, Alborz Geramifard, Lihong Li, Csaba Szepesvari, Tao Wang.

✉ Scott Rome
scott_rome@comcast.com

[1] Applied AI Research, Comcast, Philadelphia, PA, USA

[2] Applied AI Research, Comcast, Washington, D.C., USA

[3] Statistics Department, Columbia University, New York, NY, USA

improved upon our key business metrics tracking customer engagement; however, while applying off-policy methods, we encountered two major practical problems that were not fully covered in the existing literature. Therefore, in this work we aim to provide case studies of our empirical findings. In particular, we aim to

(a) demonstrate how the choice of logging policy affects standard off-policy estimates while highlighting insights we found to not be immediately accessible in the literature and
(b) present a series of practical tricks and hacks that we employed post-training to improve bandit model performance when there exists a null action and a noisy reward signal.

To motivate our first goal, we pose the following question: when data resources are limited, what kind of logging policy should one use for accurate off-policy estimates? This is of key practical concern when applying off-policy evaluation. Although there is a rich literature discussing off-policy methods (Strehl et al. 2010; Vlassis et al. 2019; Joachims and Swaminathan 2016; Li et al. 2015; Swaminathan and Joachims 2015; Jeunen et al. 2019; Swaminathan and Joachims 2015), most practical examples in the literature involve applications to search engines (Li et al. 2015; Swaminathan et al. 2016; Chen et al. 2018), advertising (Bottou et al. 2013), or news recommendation (Li et al. 2012, 2010, 2011), where millions of examples are generated per day with hundreds to thousands of bandit arms to suggest. In examples of contextual bandit problems for customer support applications such as Karampatziakis et al. (2019), a policy returns a slate of help options selected from a large action space, which is starkly different than the use case herein. When compared to the existing literature, this work is differentiated by an atypically small data regime and action space size.

In the nascent phase of the project, there were only 30,000 data points per month and three bandit arms available. Within a year, our system generated approximately 200,000 data points per month, and after two years, the project generated over 1,000,000 unique data points per month. Given the smaller volume of data, one must be strategic about which policies to deploy in production, considering the policy with the highest business value may not generate useful data for offline evaluation and training. In Sect. 3.2, we present a series of experiments comparing the accuracy of different off-policy estimators when using different logging policies. We also demonstrate empirically the convergence and variance of these estimators. Our results underscore the importance of carefully selecting a logging policy tailored to a given use case in order to balance off-policy estimation accuracy and business value in a practical setting.

We further submit that several insights from our study which we present in this work are not thoroughly discussed in the literature from the perspective of practical applications. For example:

- the empirical observation of the importance of stochasticity in the logging function (Sect. 3.5.1),
- best practices for the choice of train/test splits with multiple logging policies and limited data for off-policy evaluation (Sect. 3.5.2), and
- detailed examples of and strategies for constructing an exploration policy (Sect. 3.1.1).

These topics are of practical concern and required the authors to complete multiple A/B tests over the course of several months to converge on a working off-policy system. We document the case studies and discussions in this work to help streamline more applications of off-policy methods.

Our second case study was born of practical necessity. Our goal for the notification system is to decrease the rate at which customers abandon the application before typing an utterance or clicking a notification; however, as there could be many reasons driving a customer not to interact with the bot, the "abandon rate" of customers proved to be a noisy reward for training models. Causally, a model could influence the abandon rate by displaying a notification and the customer clicking on it, or an incorrect notification could increase the abandon rate. On the other hand, customers use the chatbot for more than troubleshooting, which implies there are other factors that could influence the abandon rate beyond the scope of our model. These external factors add noise to the reward signal, and thus, crafting a reward function solely based on the abandon rate led to practical problems during training. Many candidate reward functions tested led to models which virtually only recommended the null action of "no card". Furthermore, nontrivial candidate models failed to improve upon our business metrics during testing. In order to correct for these apparent issues, we applied a series of practical hacks and procedures, the collection of which we call Post-Hoc Reward Distribution Hacking (PRDH) and describe in Sect. 4. PRDH creates a one parameter family of models from a single trained model by shifting the reward value for the null action and moreover gives a visual framework for evaluating this family's performance on key business metrics. Our approach can be considered a form of reward hacking, as it can be interpreted as adjusting the posterior reward distribution of a trained model. We note that this approach is not a novel theoretical framework for model selection but rather a novel procedure for updating a model post-training as a form of model selection, which to the best of our knowledge is not well-represented in the literature but relevant for practical applications.

1.1 Problem setting

For Comcast customers, there is an automated chatbot called the Xfinity Assistant which offers a self-service customer support experience. The core dialogue component is handled through a natural language processing system, but the user interface also contains a predictive troubleshooting module. In this module, we use telemetry coming from devices (e.g., a modem or television box) to identify customers who may be experiencing issues and prompt the customer with a notification to troubleshoot their device. This notification component is the focus of this work. When a customer opens the app, we may opt to show the customer a troubleshooting predictive card from a set of eligible cards. These cards are eligible based on business rules related to the customer account. We also have the option to not show a card, which is called the "null action" in this work. In Fig. 1, we provide a screen shot of the Xfinity Assistant showing an early version of the internet troubleshooting card available to the production policy.

As discussed in the introduction, our goal is to reduce the customer abandon rate. An "abandon" occurs if a customer exits the application before clicking a notification or entering a question (called an "utterance") for the chatbot. To understand the success of our interventions, we also tracked two metrics with a stronger causal relationship to our bandit arms: the click-through rate (CTR) and the percent of sessions that are shown a card (%S). To decrease the abandon rate, we train a contextual bandit model to decide which card, if any, to show to

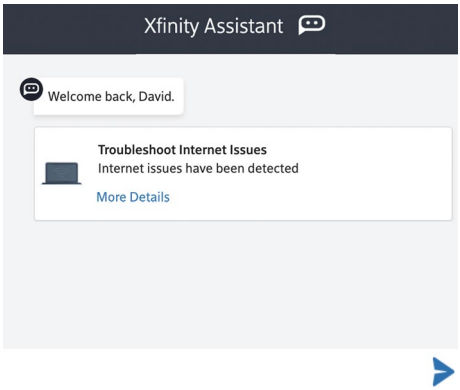

Fig. 1 Demonstration of the "internet" predictive card in the Xfinity Assistant

a customer. A reward function (detailed in Sect. 2.2) was designed to serve as a proxy for the abandon rate using the click-through rate of troubleshooting notifications and to balance competing concerns: to show as many notifications as possible without showing incorrect notifications. Showing no notification has a neutral, if not slightly negative, effect. Namely, if the customer did need troubleshooting, then it is a missed opportunity. Because of this trade-off, we consider CTR and %S as analogues to precision and recall in the classification setting, and we demonstrate in Sect. 4 how we use these metrics in a similar fashion to precision and recall during model selection for contextual bandit applications with a null action.

1.2 Notation

We will now formalize the problem described above in a more abstract setting. The following definitions are informed by the approaches of Swaminathan and Joachims (2015), Strehl et al. (2010), Swaminathan and Joachims (2015), and in particular, our notation and naming conventions follow (Swaminathan and Joachims 2015; Joachims and Swaminathan 2016) closely.

We denote our customer's telemetry data $x \in \mathcal{X}$ which are sampled i.i.d. from a distribution $Pr(\mathcal{X})$. Our logging policy is denoted h_0, and actions a are sampled conditional on x, $a \sim h_0(\cdot|x)$, which we will write $a \sim h_0(x)$ in an abuse of notation. For a given policy h, we will write $h(x, a)$ to be the propensity score assigned to x by h for the action a. When the action is undefined (like in an expectation), we will denote this $h(\cdot|x)$. For the rest of the work, we reserve h to represent a policy and h_0 will always represent the logging policy which has generated the data for training and evaluation. We assume we have a reward function $r := r(x, a)$ which is deterministic and defined concretely in a later section, and we will suppress the x and a for convenience. Our data takes the form of triplets $(x_i, a_i, r_i) \sim \mathcal{D}$ where $\mathcal{D}$ implicitly depends on the logging policy h_0 with $|\mathcal{D}| = n$.

In order to correct the data dependence on h_0, the standard inverse propensity score is applied, and the (offline) empirical reward estimate of a policy h is given via

$$V_{OP}(h) := \sum_{(x,a,r)\in\mathcal{D}} \left[r(x, a) \frac{h(x, a)}{h_0(x, a)} \right]. \tag{1}$$

It has been shown that (1) is an unbiased estimate of the true reward of h,

$$V(h) := E_{x\sim Pr(\mathcal{X})} E_{a\sim h(\cdot|x)} r(x, a) \tag{2}$$

and has been used in practice in a variety of applications (Li et al. 2010; Strehl et al. 2010; Chen et al. 2018; Li et al. 2015).

The formula we use for the inverse propensity score (IPS) estimate of the reward, also called importance weighting, is

$$V_{IPS}(h) := \frac{1}{n}\sum_{i=1}^{n} r_i \frac{h(x_i, a_i)}{h_0(x_i, a_i)}. \tag{3}$$

Although it is an unbiased estimator, the IPS estimator can have unbounded variance (Swaminathan and Joachims 2015), and so V_{IPS} can be improved via variance reduction techniques to form a biased but strongly consistent estimator called the self-normalized estimator (snIPS). We use the definition of snIPS from Eq. 7 of Sect. 5 in Swaminathan and Joachims (2015),

$$V_{snIPS}(h) := \frac{\sum_{i=1}^{n} r_i \frac{h(x_i,a_i)}{h_0(x_i,a_i)}}{\sum_{i=1}^{n} \frac{h(x_i,a_i)}{h_0(x_i,a_i)}}. \tag{4}$$

We note that Eq. 4 is also called weighted importance sampling in the literature. In this work, we will use both estimators in the results section to demonstrate the utility of each estimate on our problem.

Finally, it is common to define the effective sample size (as opposed to the sample size, n) for our data $(x_i, a_i, r_i) \in \mathcal{D}$ via

$$n_e := \frac{(\sum_{i=1}^{n} w_i)^2}{\sum_{i=1}^{n} w_i^2} \quad \text{where } w_i = \frac{h(x_i, a_i)}{h_0(x_i, a_i)}. \tag{5}$$

One important note is as follows: in the experiments of later sections, we calculate the sample size and effective sample size using only data points with non-null actions. As our metric of interest requires a card to be shown (e.g., the click-through rate), sample sizes and effective sample sizes calculated including null actions are misleading. Finally, we will also refer to the relative difference (abbreviated "% diff") between two quantities x and y defined as

$$100 \cdot \frac{x - y}{y}, \tag{6}$$

where the denominator is a "true" or non-estimated value.

2 Data and modeling

In this section, we present an overview of the data and the specifics of our modeling approach. The features engineered to solve this problem are detailed in Sect. 2.1 along with references to our production serving architecture. Our reward function and model architecture are found in Sects. 2.2 and 2.3 respectively.

2.1 Features and action spaces

In addition to the null action, there are two notifications available in the notification component: troubleshoot internet and troubleshoot TV. Thus, the bandit arms we refer to are "internet card", "TV card", and "no card". The "no card" action corresponds to the null action. Because of this, our features are based on telemetry related to a customer's internet and TV connections.

The features x are a mix of continuous, discrete, and binary signals from devices. A majority of features come from telemetry data, specifically application logs from internet gateways and television boxes themselves. Our gateways and television boxes are powered by an open source framework called Reference Design Kit (RDK). These logs are sent in near realtime back to our monitoring systems, which we then consume to create features. We also have features pertaining to the customer's subscriptions, as well as what application the customer is using to access the Xfinity Assistant, which we refer to as the "channel". Possible channels include mobile, web, and Facebook chat.

RDK logs a variety of internet-related errors and system messages which are aggregated and sent every 15 minutes. (On some older firmwares, this aggregation is every hour.) These error and system codes have varying degrees of importance: some relate to Linux processes restarting while others catalog connectivity timeouts and gateway reboots. From these events, we extract unique messages and errors to use as "words" for a term-frequency inverse document-frequency embedding that is passed to the model as a feature. Rather than using individual events as inputs to the model, we roll up the counts to a larger time window of either 3 or 24 hours. Today, we utilize approximately 750 error keys in our production policy.

RDK also provides in-home wireless network (WiFi) related messages summarizing telemetry markers from the gateway. Those messages include measures between connected client devices and the gateway like data transmission rates and WiFi signal strength (relative signal strength, RSSI). Overall measures on the gateway itself are also reported, including a channel utilization score indicating the extent of WiFi channel congestion. These measures have been used to reliably estimate in-home wireless experience (Patro et al. 2013). To generate features for our model, we use a rolling window of several hours (e.g., 6 hours) to collect these WiFi related messages and derive approximately 80 aggregate features such as the minimum, maximum, mean, and standard deviation of each WiFi measure.

For television, the Xfinity X1 application surfaces a number of errors to the customer on their TV screen. We capture these errors and use them as a signal to the model. As opposed to errors from the gateway, these TV errors indicate a variety of service disruptions; however, these events are also rare. During the initial launch and when our experiments were conducted, such errors were our sole source of features for the television domain, causing the "TV card" to appear much less frequently than the internet card.

In-depth technical details of our production feature store and model deployment architecture are available in Pinckernell and Rome (2020). Up-to-date features were stored in a cache, and the most recent values of the features were collected and passed to our production policies when a customer accessed the Xfinity Assistant.

2.2 Reward function and quasi-advantage

As discussed in the introduction, the abandon rate is impacted by external factors and thus not completely controlled by our possible bandit arms, leading to a noisy reward signal. Instead, we chose to model our reward function to closely track the click-through rate. We found that using the click-through rate directly also led to poor results as the model had no incentive to choose the null action. Instead, we used the following reward which utilized a blend of the click-through rate and the abandon rate:

$$r_{\alpha,\lambda}(x) = -\alpha \text{ if the customer abandoned the session on the launch page,}$$
$$\text{otherwise} \begin{cases} 1 & \text{card shown and was clicked} \\ -\lambda & \text{card shown and was not clicked} \\ -1 & \text{no card and troubleshooting action initiated} \\ 0 & \text{no card and no troubleshooting action initiated} \end{cases} \tag{7}$$

where $\alpha, \lambda \in [0, 1]$ are configurable hyperparameters. The logic behind the reward is to encourage the model to show a card to anyone who needs to perform a troubleshooting action. Because the application has other purposes, we do not expect to show a card to every customer. The parameter λ is one lever used to control the trade-off between higher click-through rates and a higher occurrence of cards in the application. The parameter α is meant to encourage the model to intervene with customers who would otherwise abandon the application. For our experiments we set $\lambda = .9$ and $\alpha = .25$, and these values were determined by trial and error throughout our model development.

We observed an improvement in performance when we took steps to normalize the reward during training. In particular, let $\bar{r}_a$ represent the average reward for all cases of action a in our training set. During training, we replaced r_i with $\hat{r}_i$,

$$\hat{r}_i := r_i - \bar{r}_a, \tag{8}$$

where we subtract the average reward from the training set for the action corresponding to r_i. We call this approach quasi-advantage because it is a quick, rough version of traditional advantage terms in reinforcement learning. Our deployed production policy consisted of the individual reward predictors and an argmax was used to select the action. Therefore, we would store $\bar{r}_a$ for each action and add each respective average reward to the output of the model at runtime to maintain the original reward distributions.

2.3 Model architecture

Here, we will detail the model architecture for our original reward predictor model. In production, we used a linear model (Fig. 2) implemented in Keras (Chollet et al. 2015) where each troubleshooting card's features were independent, but all features were used for the "no card" arm. One outcome of this design was it allowed easy addition of new features which only affect a specific arm. For example, there is a single input layer for our gateway telemetry features. We then pass those features through a dense layer with a linear activation to an output of size 1. Similarly, there is a single input layer for our TV-related features, and we again pass them through a dense layer with linear activation to an output of size 1. For "no card", we concatenate all inputs and pass them through a dense layer with a linear activation of output size 1. As each output corresponds to a single arm, we can concatenate the three outputs and pass them through a tanh activation to use as our reward

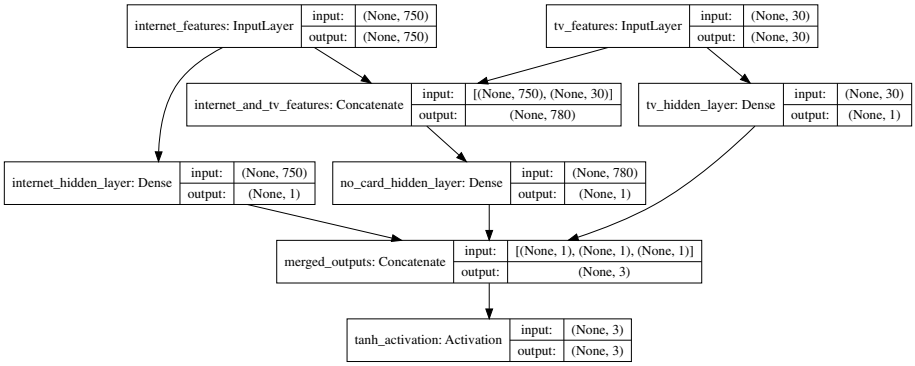

Fig. 2 A graphviz representation of the model architecture for our original production policy using the Keras model_to_dot function

predictor's output. Later in Sect. 5.3, we describe how we updated this architecture when adding the channel feature to the model.

3 On the choice of logging policies for off-policy evaluation

In order to calculate accurate off-policy estimates while maintaining performance, we tested a variety of candidate exploration policies. In addition to our production policy, we used two main logging policies for training and evaluation: a uniformly random policy and a simplified exploration policy. The uniformly random policy would sample from available actions uniformly, and the simplified exploration policy was designed to have a better experience for the user while still producing accurate off-policy estimates. We also tested an ϵ -greedy version of our production policy which we present in Sect. 3.3.1.

Our final production setup split traffic between three policies, typically giving a small portion of traffic to the uniformly random policy, a moderate amount of traffic to the simplified exploration policy, and the remainder of traffic to our production model. These policy choices were settled upon after many iterations of trial and error where we aimed to balance performance on the key metrics and exploration.

In this section, we will describe the design of the simplified exploration policy and provide experimental results for the accuracy of different off-policy estimators using different logging policies. For the experiments, we present the relative difference between the off-policy estimate of the click-through rate and the true performance of the policy in production for business privacy purposes. Moreover, we include an experiment detailing the convergence of these estimators with respect to the sample size and effective sample size plus an estimate of their variance. Lastly, we present a brief investigation of the business performance of the different logging policies.

We note that in all cases of stochastic policies herein a seed was set based on the customer's account number when sampling from the distribution of actions. Similarly, when multiple policies were deployed, the logging policy was selected uniformly at random using a seed also based on the customer's account number.

3.1 Designing a logging policy

Our first inclination was not to design any additional logging policies. We initially tested a deterministic version of our production policy as the logging policy. This led to exceedingly biased and inaccurate off-policy estimates. Our experiment in Sect. 3.2.1 and results in Table 1 demonstrate this bias. We then deployed a stochastic modification of our production policy using an ϵ-greedy scheme, but that approach also did not lead to accurate estimates either, which we present in Sect. 3.3.1. In Joachims and Swaminathan (2016), the off-policy estimates were not accurate until the data volume was orders of magnitude larger than ours, and so we decided to take special care to reduce the variance of our off-policy estimator.

Our approach is based on the idea of stratified sampling: we segment the feature space to well understood segments on which we can reduce the variance of future off-policy estimates. To accomplish this, Sect. 3.2 of Li et al. (2015) suggests that if a new policy h is "near" the logging policy h_0, the variance of off-policy estimates will be reduced. Thereby to reduce the overall variance, we craft meaningful segments with action distributions that are directionally in line with future trained policies. We note that because our experiments with an ϵ-greedy modification of production yielded poor estimates, we did not test other candidates for policies "near" our production policies, e.g., a softmax version of our argmax-based production policy. We also did not investigate optimization schemes for constructing logging policies such as Hanna et al. (2017). We instead opted to use organic user behavior data from the application and the variance reduction strategy described below to create a new logging policy.

Let $\{X_i\}_{i=1}^m$ be a partition of $x \in \mathcal{X}$ with $X_i \cap X_j = \{\}$ for all $i, j = 1, \ldots, m$ with $i \neq j$. Let D_i be the data points of $\mathcal{D}$ where $x \in X_i$ and $n_i = |D_i|$. For each $x \in \mathcal{D}$, we will also write x_{jk} to denote the k-th sample in D_j. We will also add D into the notation of V_{IPS} from (4), $V_{IPS}(h, D)$, to clarify the summation domain. Then, we can decompose the offline estimator V_{IPS} into

$$V_{IPS}(h, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} r(x_i, a_i) \frac{h(x_i, a_i)}{h_0(x_i, a_i)} \tag{9}$$

$$= \sum_{j=1}^{m} \frac{n_j}{n} \frac{1}{n_j} \sum_{k=1}^{n_j} r(x_{jk}, a_{jk}) \frac{h(x_{jk}, a_{jk})}{h_0(x_{jk}, a_{jk})} \tag{10}$$

Table 1 Table of relative difference (% Diff) between the offline estimate of performance calculated on data generated before the A/B test and the model's performance in an A/B test

Estimates using data generated prior to the test					
Model	Logging policy	V_{IPS} % Diff. (%)	V_{snIPS} % Diff. (%)	n	n_e
A	Random	−91.1	−0.890	65,582	1996
A	SCX	−41.8	1.48	40,992	1013
A	Production	−54.6	4.15	8486	3694
B	Random	−80.1	0.320	65,582	4345
B	SCX	−45.3	3.20	40,922	1596
B	Production	−50.3	15.0	8486	3662

Sample size (n) and effective sample size (n_e) are listed

$$= \sum_{j=1}^{m} \frac{n_j}{n} V_{IPS}(h, D_j) \tag{11}$$

where the second line is done by rearranging terms and multiplying by $n_j/n_j = 1$ and the final line follows from the definition of V_{IPS}. From this formula, reducing the variance on each segment D_i is sufficient to reduce the overall variance.

3.1.1 Our simplified exploration policy

In defining our exploration policy, we aimed to create meaningful segments of data where we could assign action distributions which would be in line with future policies. The resulting policy we refer to as the simplified exploration policy (SCX). To form our segments, we trained a collection of binary classification models using subsets of the features outlined in Sect.2.1 on alternative tasks. Labels were generated by organic customer interactions with the chatbot. We concatenated the outputs from these models as an identifier for each example and manually assigned an action distribution to each combination.

The motivation behind the approach is as follows: if we could classify customers as seeking internet or television troubleshooting using only telemetry data, then the output of those models may be sufficient to form segments. Additionally, by knowing the customer's predicted intent for interacting with the chatbot, we could select an appropriate exploratory action distribution for that customer. The natural choice for labels was to use annotated customer interactions with the chatbot, and we used the natural language processing system powering the Xfinity Assistant as our annotator.

The NLP system classifies every utterance input by the customer into one of approximately 200 relevant classes referred to as "customer intents". The system is trained on patterns of previous queries and their known intents. Any new query that follows a known pattern is recognized via a syntactic parser (Earley 1970). Queries that do not match to a known pattern are run through a neural model to predict one of the intents with high confidence. The model encodes words as vectors using a multilingual word embedding (Lample et al. 2018) trained on in-domain text. The vectors are fed into an ensemble of encoder layers: a fully bidirectional LSTM model and a pre-trained universal sentence encoder (Cer et al. 2018). The output of the encoder finally goes through a softmax layer to return a probability distribution over the intent classes.

We constructed labels by selecting the first utterance of every session and mapping the customer intent from the NLP system to a new label for training. In particular, our labels were "relating to internet troubleshooting", "relating to television troubleshooting" or "other". If a customer entered the chatbot and stated "my internet is not working", then this interaction would be classified by the NLP system as "internet. troubleshooting", which we would then label as "relating to internet troubleshooting". (Note, the data for the SCX component models were taken from production before the introduction of model-driven notifications.)

Our training regiment was inspired by a "one-vs-all" scheme for multiclass classification tasks (Aly 2005). We trained models to distinguish a single label from the other labels using a subset of features, and we enforced that a feature could only be used in at most one model. The original SCX consisted of three models. Using the RDK telemetry features, we trained a model to predict the label "relating to internet troubleshooting". A separate model was trained using WiFi features for the internet-related

label, and we trained the last model using the X1 errors to predict "relating to television troubleshooting". The binary outputs of these models were combined in an array, forming a bucket in which to bin examples. We then manually defined a fixed probability distribution over the arms for each bucket. At runtime, SCX called each model to determine the proper bin and then sampled from the specified probability distribution.

3.2 Comparison of multiple off-policy estimators using different logging policies

This section contains three experiments to demonstrate the performance of different logging policies for off-policy evaluations. Each experiment was run on live production traffic during October 2019. In the first experiment, we compare the off-policy estimates calculated on data *before* the A/B test with the true performance during the A/B test. As a second experiment, we compare off-policy estimates with the A/B test results from logging policy data on the days which constituted the A/B test. The third experiment creates a fair comparison by fixing the sample sizes of each logging policy via bootstrapping. In all cases, we observed the IPS estimate V_{IPS} defined in (3) to have high variance across tests and logging policies which led us to only use the V_{snIPS} from (4) in practice. Moreover, we saw larger bias and variance from deterministic policies which led us to only use stochastic policies in our off-policy estimates, and we discuss this phenomenon further in Sect. 3.5.1.

We include experiments using data generated before and during the A/B test because there can be variation in the estimates due to the use of different days when calculating the estimate (see Tables 1, 2). Our data contains non-stationary trends in user behavior (e.g., seasonal effects), feature distribution shifts from frequent firmware rollouts, and rare daily events such as outages or promotions. Because of this, the off-policy estimates can be influenced by how recently the logging policy data are generated and if data conditions are similar. Moreover, we note that the experiment in Sect. 3.2.1 provides a typical example of the data available to us when determining which models to release into A/B tests, and thus, we are interested in if the off-policy estimates are robust to practical data drift experienced in our setting.

In all experiments, we compare the performance of different off-policy estimators and logging policies against the online performance for two models. Each model was run in an A/B test concurrently for two weeks, and the metric used for the comparison is

Table 2 Table of relative difference (% Diff) between the offline estimate of performance calculated on data generated during the A/B test and the model's performance in an A/B test

Estimates using data generated during the test					
Model	Logging policy	V_{IPS} % Diff. (%)	V_{snIPS} % Diff. (%)	n	n_e
A	Random	−90.2	4.75	54,033	1693
A	SCX	−42.1	−0.297	35,475	886
A	Model B	−59.9	4.15	21,934	8456
B	Random	−79.5	1.60	54,033	3651
B	SCX	−25.6	0.641	35,475	1372
B	Model A	−3.50	12.8	14,941	12,764

Sample size (n) and effective sample size (n_e) are calculated for non-null actions only

the click-through rate. For this test, we use the off-policy estimators V_{IPS} and V_{snIPS}. The relative difference was calculated as the difference from the true online performance, i.e., via Eq. 6 with the denominator y equal to the online CTR of the model.

3.2.1 Experiment 1: using logging data generated before the A/B test

In this test, we compared performance estimates of the two candidate models using data from three logging policies generated prior to the A/B test. In practice, this is the type of test that a researcher would use to choose a model to deploy. The three logging policies for this experiment were a deterministic production policy ("Production"), a uniformly random policy ("Random"), and the stochastic simplified exploration policy ("SCX") from Sect. 3.1.1, and the models used for the comparisons are denoted "A" and "B". In this case, we saw the uniformly random policy produce better results than the simplified context policy. However, both logging policies were able to give a useful prediction when using the snIPS estimator (4) as can be seen in Table 1. It is clear that the V_{IPS} estimator was not useful for our data regime, and we relied on the V_{snIPS} for decisions on which model to release into an A/B test.

It is worth noting that the effective sample size n_e is on the order of 10^3. Of course, as shown in the table, the sample sizes n and effective sample sizes n_e are different for every model, and that is an issue one sometimes deals with in practice. As our metric in question is the click-through rate, n and n_e are only calculated on non-null actions, i.e., examples where we show a card.

3.2.2 Experiment 2: using logging data generated during the A/B test

In this experiment, data generated during an A/B test is used to produce offline estimates of model performance using different logging policies and then the estimates are compared to the observed performance during the A/B test. During this A/B test, we generated logging data from Model A, Model B, a uniformly random policy ("Random") and our simplified exploration policy ("SCX"). Model A and B were deterministic while SCX and the random policy are stochastic. From Table 2, the uniformly random policy and the simplified exploration policy with the V_{snIPS} estimator perform the most consistently. It is interesting to note that in this case the simplified exploration policy provides a better estimate, whereas in Experiment 1 the uniformly random logging policy performs better.

Figure 3 plots daily off-policy estimates for the two models during the A/B test. Each plot point only uses data from a single day. We also note that the variances appear high at first glance, but there is an explanation: the effective sample sizes for each day are small, so one expects higher variances. We also highlight that the V_{IPS} estimator has a much higher bias when being used to estimate Model A's performance for all logging policies.

3.2.3 Experiment 3: fair comparison using bootstrapping and fixed sample sizes

Finally, we aim to provide a fair comparison between the different logging policies by fixing the sample sizes. The previous experiments used the exact data available to us when performing off-policy evaluations, but it is also of interest to observe which logging policy is most performant under a fixed sample size. In order to do this, we

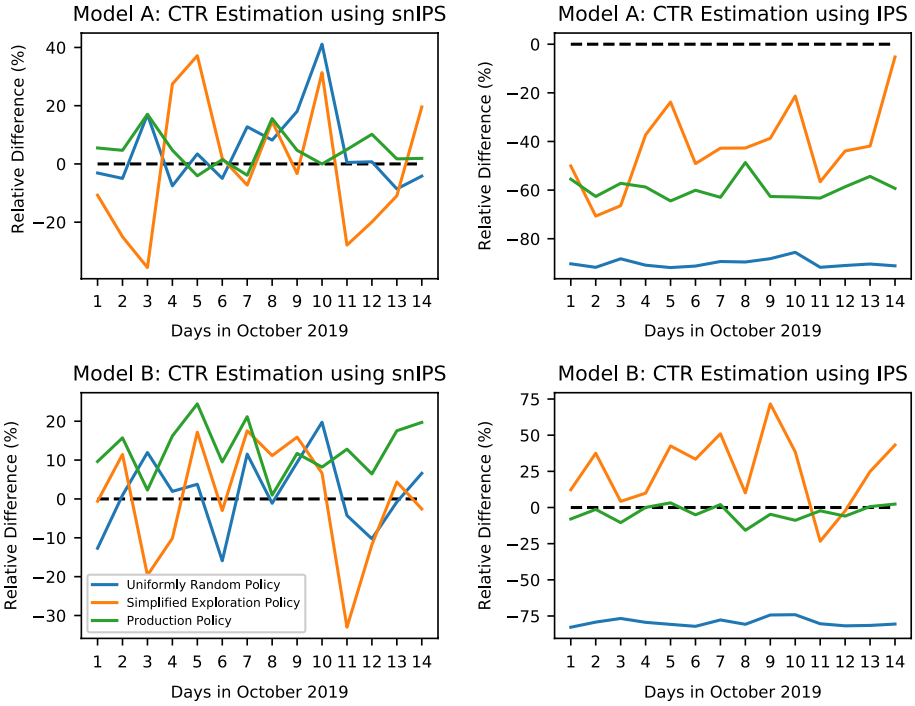

Fig. 3 Plot of daily off-policy estimates for model A and B using various logging policies. Each point represents an off-policy estimate using a single day of data

bootstrapped on the data from the previous two experiments with a sample size of $n = 10,000$ and $m = 1000$ bootstrap iterations. For each metric, the mean of the metric over the bootstrap iterations is reported as the "bootstrap metric". To denote that these estimates were bootstrap estimates, we append the superscript B. The bootstrapped fair versions of Experiment 1 and 2 are contained in Tables 3 and 4 respectively. One striking result is that the V_{snIPS} estimator performs relatively well even with a small effective

Table 3 Results of the bootstrapped relative difference (% Diff) between the offline estimate of performance calculated on data generated before the A/B test and the model's performance in an A/B test

Estimates using data generated prior to the test					
Model	Logging policy	V_{IPS} % Diff.^{B} (%)	V_{snIPS} % Diff.^{B} (%)	n	$n_e{}^{\mathrm{B}}$
A	Random	−91.0	−0.62	10,000	304
A	SCX	−42.0	1.62	10,000	248
A	Production	−54.7	4.14	10,000	4352
B	Random	−80.0	0.519	10,000	664
B	SCX	30.8	3.39	10,000	390
B	Production	−50.4	14.9	10,000	4317

Sample size (n) and the bootstrap estimate of the effective sample size (n_e) are listed

Table 4 Results of the bootstrapped relative difference (% Diff) between the offline estimate of performance calculated on data generated during the A/B test and the model's performance in an A/B test

Estimates using data generated during the test					
Model	Logging policy	V_{IPS} % Diff.^{B} (%)	V_{snIPS} % Diff.^{B} (%)	n	$n_e{}^B$
A	Random	−90.2	4.75	10,000	312
A	SCX	−42.0	0.14	10,000	250
A	Model B	−59.9	4.06	10,000	3856
B	Random	−79.5	1.25	10,000	676
B	SCX	−25.0	0.109	10,000	386
B	Model A	−3.69	12.76	10,000	8541

Sample size (n) and the bootstrap estimate of the effective sample size (n_e) are calculated for non-null actions only

sample size (< 1000). It is interesting to note that the results do not change significantly from the previous experiments.

3.3 Empirical investigation of the convergence and variance of the off-policy estimators

3.3.1 Convergence

Now, we fix a trained model and examine the convergence of offline estimators with respect to the sample size and logging policy. We compare the offline estimates to ground truth performance obtained from a production deployment. The policies are a uniformly random logging policy, a simplified exploration exploration policy as defined in Sect. 3.1.1, and a prior production policy with ϵ−greedy exploration added. We note that the model tested was trained using data from a previous period, and its training set consisted solely of data from the ϵ-greedy modification of the production policy. We suspect that the overestimate observed in the figure while using the ϵ-greedy exploration policy is related to the training set composition. In particular, our hypothesis is that using an ϵ-greedy exploration policy to calculate the off-policy estimate, even when using data from a different period, is similar to estimating performance of a model using the training set rather than a held-out test set.

In Fig. 4, we plot the snIPS estimator $V_{snIPS}(h)$ defined in (4) using different choices of $\mathcal{D}$, i.e., using different logging policies and various sample sizes. It is clear that using the ϵ-greedy version of the production policy for off-policy estimates led to a severe overestimate of performance. This agrees with findings of other studies which recommend avoiding logging policies that utilize ϵ-greedy exploration (Karampatziakis et al. 2019). The uniformly random policy tended to underestimate the online result: the error converges to approximately 10%. The simplified exploration policy estimated the online result the most accurately: the error is below 5% for larger sample sizes. The three logging policies collected data at different time periods due to our continuous model deployment process as well as different online traffic volumes allocated to each model. However, it is unlikely that the difference in performance of the estimators is due to different effective sample sizes. For example, when the effective sample size is around 1000, both of the estimates for the uniformly random and the simplified exploration policies have already stabilized,

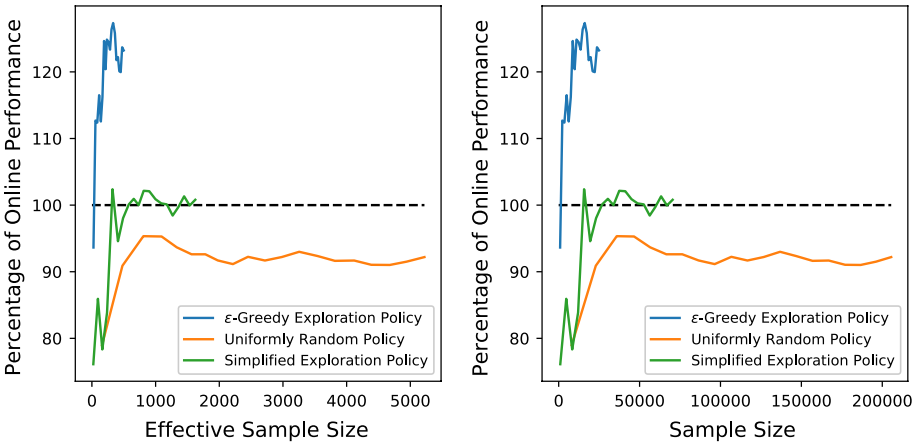

Fig. 4 Off-policy evaluation convergence comparison between three logging policies at various sample sizes and effective sample sizes for non-null actions only. The plotted percent is given via $V_{snIPS}/(\text{Online CTR})$

apparently converging to their corresponding estimate calculated with higher effective sample sizes, whereas the convergence of the ϵ-greedy policy is ambiguous.

One surprising observation from Fig. 4 is that the ϵ-greedy policy and uniformly random policy do not converge to the online value. This is surprising because the literature suggests they should. However, there are some practical reasons why that is not the case. For the ϵ -greedy policy, we "clip" the IPS weights (Ionides 2008) during evaluation via the standard formula $\min\{M, h/h_0\}$, which can add bias to the estimate. However, this clipping would not explain the uniformly random policy's estimation error as propensities were 1/3 for each action. There are two likely factors impacting the uniformly random policy's convergence. First, the application's business logic overrides the predictions in some scenarios, and the propensity weights are therefore affected in a way that we do not account for. Second, in order to get enough data for the uniformly random policy, we were forced to use a larger time window containing older dates compared to the simplified exploration model and ϵ-greedy policy, which generated more data in production. All estimates were compared to the performance of a single model while it was running in an A/B test, and so the older data could have also added a bias to the estimate.

3.3.2 Variance

For our last investigation, we provide empirical variance estimates for the IPS and snIPS estimators calculated on our data. In order to create the best estimate, we use data from both of our stochastic policies (Random and SCX) to create a dataset of size $n = 20,519$ containing non-null actions only. (The model used for the evaluation was our production policy.) This data constitutes an approximately two month period in 2019 and is compiled into Table 5. We used bootstrapping to calculate the variance of the IPS and snIPS estimates of the click-through rate for a fixed model. We performed $m = 10,000$ iterations for the calculation and

Table 5 Bootstrap estimates of the standard deviation and variance of V_{IPS} and V_{snIPS} rounded to two significant digits. The number of bootstrap iterations was $m = 10,000$ and the base dataset contained $n = 20,519$ examples

$V_{IPS}\ \sigma$	$V_{IPS}\ \sigma^2$	$V_{snIPS}\ \sigma$	$V_{snIPS}\ \sigma^2$	$\overline{n}_e$
0.027	0.00072	0.011	0.00012	1295

provide both the standard deviation σ and variance σ^2, plus the mean effective sample size $\overline{n}_e$, for the experiment.

3.4 Comparison of the production performance of various logging policies

Our last result of the study is a comparison of the relative performance of the various logging policies on our key business metric: the abandon rate. As discussed, the abandon rate depends on both CTR and %S and is the best reflection of a model's impact to the business. For business privacy purposes, we will compare each policy to the baseline performance of the production policy and calculate the relative difference (6) with the production policy performance as the denominator. This data set is an aggregation of production logs over a two week span near the end of 2019. During this period, the approximate breakdowns are as follows: the random policy served 5% of traffic, the SCX model served 10% of traffic, ϵ -greedy version of the production model ($\epsilon = .05$) served 25%, and the production model served 60% of traffic. The allocation of the logging policy for each user was sticky during the period and the application saw approximately 700,000 users. The results are compiled in Table 6. As expected, the performance of the ϵ-greedy model is the closest to production, followed by the SCX model, and the uniformly random model performed the worst on the business metric.

3.5 Considerations and conclusions

In this section, we collect our conclusions from the previous case study. From the experiments, it is clear that the SCX logging policy is the best choice to balance accurate off-policy estimates with business value. Another clear takeaway is the value of A/B tests in validating an off-policy methodology. In all cases, we recommend running an experiment to determine if a logging policy and the data volume is sufficient: estimate the performance of the current production model via your proposed logging policy. If the estimates were to agree over the same period, one should be more confident in the offline estimates for new models using that logging policy. Over the course of the project, we began to use the V_{snIPS} estimator exclusively as it was the most reliable, which was also demonstrated in the study.

3.5.1 The importance stochasticity

In practice, there is a question when using off-policy methods whether it is more important to have more data or more stochasticity. We show empirically in Table 1 that a deterministic policy yielded a severely inconsistent and biased estimate when using both V_{snIPS} and V_{IPS} across multiple tests. We also note that by using a deterministic logging policy during

Table 6 The abandon rate rate of each logging policy relative to the abandon rate of the production policy. A positive number implies a higher (and worse) abandon rate than our production policy

Logging policy	Relative difference from production policy performance (%)
Random	56.2
SCX	31.2
ϵ-greedy	5.19

training with an inverse propensity score-based loss (Eq. 14 in Sect. 4.1), one would be introducing that bias into the new model. Thus, we do not use deterministic data in the evaluation or learning phases. Moreover, in the convergence experiment of Sect. 3.3.1, we show that an ϵ-greedy scheme to randomize our production policy was not sufficient for off-policy estimates, which is unfortunate as that exploration policy maximized business value of the policies tested (Table 6). On the other hand, we found generally in our experience and have shown in the experiments herein that sufficiently stochastic logging policies can yield more reliable estimates, providing unique business value by aiding in the development of new production policies.

In the past, we attempted to train models using estimated propensities that arise from multiple deterministic logging policies which were sampled randomly at runtime to add stochasticity. We employed the "Balanced IPS Estimator" (Definition 5.1 in Agarwal et al. (2017)) as our off-policy estimator. The initial off-policy estimates calculated via this method were not as accurate as the off-policy estimates obtained using a single stochastic policy, so we quickly abandoned this approach. In light of this discussion, our hypothesis to explain the poor performance is that there were an insufficient number of deterministic policies to make the aggregation of those policies sufficiently stochastic for accurate estimates; however, it could also be that the Balanced IPS Estimator is a multiple logger analogue to V_{IPS}, which also performed poorly on our data compared to our expectation from the literature.

3.5.2 Train/test split and our choice of policies

We typically had two explorations policies deployed into production: the uniformly random policy and the simple exploration model defined in Sect. 3.1.1. Both of these models constituted a small portion of traffic, and because of this, we had to decide how best to utilize the data for training and testing.

Typically, we trained on the simplified exploration policy data and evaluated on the uniformly random policy data. The off-policy estimator using the uniformly random policy was sufficient for our offline evaluations, but the addition of data from the uniformly random policy to the training set did not improve model performance. We suspect the low prevalence of troubleshooting led to insufficient signal for learning when using the uniformly random policy data. To maximize our data volume, we would allow the train and test set to overlap in time, but we split the sets by policy as mentioned above. This approach had the added benefit of also splitting the data by customer because user assignment to a policy was sticky over the data collection period. When data were particularly scarce, we would include the deterministic production policy as part of the training set, but as the stochastic data volume grew, we saw performance gains by removing the deterministic policy data from training.

4 Model selection through post-hoc reward distribution hacking

Post-Hoc Reward Distribution Hacking (PRDH) provides a guided framework to update a model post-training. In practice, this approach can be useful when direct optimization of a business metric may be impossible due to a weak causal link between the model and the metric. The weak link can manifest itself in a seemingly noisy reward signal where models optimized to maximize the expected reward fail to improve the true business metric. One

situation this can occur is when the metric can be impacted by various external factors. We found that such noisy rewards could still be used to train models which improve the business metric in question by modifying the trained model's reward output. We have codified our approach in the PRDH procedure which creates a visualization of possible business outcomes from a parametrized modification of a trained bandit model and was inspired by the model selection process in supervised learning using precision-recall curves. PRDH had two significant impacts on our training pipeline and model performance:

(a) PRDH allowed us to adjust model performance on key business metrics in an attempt to correct for noise in the reward signal without retraining or editing the reward function, and
(b) when some loss functions caused a model to favor the null action at the expense of performance, we could increase the propensity of the model to perform non-null actions.

At a high level, the central idea of PRDH is to add a constant vector to the reward output of a contextual bandit model to adjust for model shortcomings without retraining. We internally referred to the vector added to the reward output as the "fudge factor" because we were "fudging" the output to make the model perform better. We create a sequence of candidate models with different "fudge factors", calculate the business metrics for each "fudge", and graph them in order to identify possible outcomes in production for various versions. This procedure purposely mirrors the process to construct a precision-recall curve as one can see in the formalized steps below. We then perform an approximate grid search near the optimal "fudge" to identify the best model to release to production.

We will now formally document the steps of PRDH as follows:

1. Train a contextual bandit model, denoted f.
2. For the null action denoted a_n, we define a family of functions f_γ parametrized by $\gamma \in (-\eta, \eta)$ to be

$$f_\gamma(x,a) := \begin{cases} f(x,a) & a \neq a_n \\ f(x,a) + \gamma & a = a_n \end{cases} \tag{12}$$

3. Choose the two most important business metrics, m_1 and m_2. For each γ, calculate the off-policy estimate of the business metrics using the reward output from f_γ. Denote their values $m_1(\gamma)$ and $m_2(\gamma)$.
4. Construct the parametrized curve $(m_1(\gamma), m_2(\gamma)) \subset \mathbb{R}^2$ and visually identify the optimal γ which satisfies the best business value for $m_1(\gamma)$ and $m_2(\gamma)$, denoted γ^*.
5. Perform a grid search, possibly by inspection or heuristics, to find the optimal model defined via $f_{\gamma^*} + y$ where y is in an $y \in (-\epsilon, \epsilon)^n \subset \mathbb{R}^n$ such that $f^* := f_{\gamma^*} + y$ maximizes the potential business value.

We note that this approach was only necessary as models trained using the noisy reward signal did not generalize well or produce reasonable action distributions. Using this procedure, we were able to consistently train new versions of the contextual bandit model while improving on the abandon rate of the application; however, without the manual steps of PRDH, our training regiment frequently would have resulted in models with unacceptable performance, which we illustrate in Sect. 4.1.

Next, we outline the concrete details of how we applied this approach. Recall V_{snIPS} is defined in (4). We further define $\%S(f)$ to be the estimated percentage of sessions in

which f will display a card. When we graph the curve in Step 4, we let $m_1(\gamma) := \%S(f_\gamma)$ and $m_2(\gamma) := V_{snIPS}(f_\gamma)$. This generates a curve similar to a precision/recall curve with percent of sessions (%S) on the x-axis and estimated click-through rate (CTR) on the y. (When training on insufficient data, either in size or in stochasticity, we observed that this curve would be jagged, but in the presence of enough data and stochasticity, we observed a smooth curve resulting from the algorithm.) As stated, our ultimate goal was to improve the abandon rate, and we viewed the off-policy estimate of CTR and %S as our levers to adjust how our notifications drove business value. This curve allowed us to identify an optimal trade-off between the two metrics. Once an optimal point was selected, we performed a grid search around this point to find the "fudge" that gave the optimal local abandon rate.

4.1 Experiment: performance of training regiments before and after PRDH

In this experiment, we will detail model performance before and after PRDH for various popular loss functions for this setting. We used the architecture defined in Sect. 2.3 for all tests. For the policy gradient case, we made the appropriate adjustment by swapping the final tanh activation of the reward predictor $f(x, a)$ in Fig. 2 for a softmax, leading to the output defined as $h(x, a)$. Each candidate model was trained on approximately 10^5 data points from our simplified exploration policy defined in Sect. 3.1.1 and tested on 10^4 data points from the uniformly random logging policy. All performance estimates were calculated using V_{snIPS} defined in (4) using the click-through rate as the reward. The final estimate displayed below is the relative percent difference (Eq. 6) over the logging policy's click-through rate r^*: $100 \cdot (V_{snIPS} - r^*)/r^*$.

For our experiment, we train with three typical loss functions found in the off-policy literature: the direct method (DM), the inverse propensity score loss (IPS), and the policy gradient loss (PG). Recall, our data $\mathcal{D}$ depends on the choice of a logging policy h_0. We now define the direct method loss function (DM) to be

$$\ell_{DM}(h) := \frac{1}{n}\sum_{i=1}^{n}(r_i - f(x_i, a_i))^2, \tag{13}$$

and the inverse propensity score loss (IPS) from Swaminathan and Joachims (2015), Bottou et al. (2013) via

$$\ell_{IPS}(h) := \frac{1}{n}\sum_{i=1}^{n}\frac{(r_i - f(x_i, a_i))^2}{h_0(x_i, a_i)}. \tag{14}$$

Lastly, for our policy gradient implementation (PG), we follow (Chen et al. 2018) and define the off-policy corrected gradient update as

$$\nabla_h \ell_{PG}(h) := \sum_i \frac{h(x_i, a_i)}{h_0(x_i, a_i)} r_i \nabla_h \log(h(x_i, a_i)), \tag{15}$$

using the softmax output $h(x, a)$ rather than the reward prediction $f(x, a)$. We note, that in practice, it is common to "clip" the propensity weights by replacing the term h/h_0 with $\min\{M, h/h_0\}$ for $M \geq 1$. We utilized this clipping for all loss functions.

As one can see in Table 7, there are not many performant choices among the models tested. The policy gradient model effectively showed no cards in the experiment, although it had the highest click-through rate (which may be suspect due to the extremely low

Table 7 Table of relative difference (% Diff in CTR) between the offline estimate of the CTR and the logging policy's CTR

Loss function	% Diff. in CTR (%)	Internet card %S (%)	TV card %S (%)	n_e
DM	100.82	2.36	3.07	667
IPS	59.63	11.12	2.71	555
PG	71.96	0.394	0	14

(The true CTR is obscured for business privacy.) The percent of sessions (%S) for the internet and TV card plus the effective sample size (n_e) are listed for each model. For all models, the off-policy logging data has size $n = 8,320$. Note, the sample size n and effective sample size n_e represent only data points with non-null actions

effective sample size). As we would not release a model that shows so few cards, we are left with two choices: a slightly higher percent of sessions models with half the click-through rate (IPS) or the higher CTR but lower percent of sessions model (DM). The abandon rates are withheld for business privacy.

On the other hand, from Fig. 5, the application of the PRDH procedure shows that the families constructed from the DM and PG models greatly outperform the IPS family for all practical purposes. At higher values of percent of sessions, the PG and DM families performed similarly, but at lower percent of sessions, PG or DM may be more appropriate depending on other considerations. Regardless, it is clear that the PRDH method expands the possible options for a final model and makes the model trained by policy gradient viable in production, whereas before PRDH the model did not show many cards as seen in Table 7.

4.2 Conclusions

In our practice, we found the employment of PRDH to be one of the most beneficial components of our training pipeline. It allowed us to easily release new iterations of our production policy while keeping performance consistently improving on all metrics. Before we developing PRDH, we experienced wild swings in performance from policy to policy with no levers to control the trade-offs discussed above. Many algorithms we initially avoided (like policy gradient) became performant when the proper γ was selected.

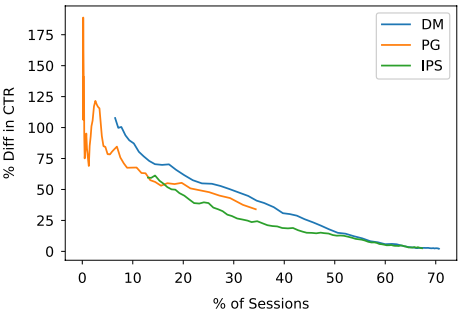

Fig. 5 Plot of curve detailed in step 4 of PRDH to visualize the performance of each family of models. Note, we set $m_1 = \%S$ and $m_2 = \%$ Diff between V_{snIPS} and the logging policy's CTR. For each γ, we calculated V_{snIPS} using the argmax of its output, as we required our non-exploration production policies to be deterministic. The formula for m_2 can be found in Eq. 6

5 Lessons

In the previous sections, we have identified multiple lessons which were validated empirically through our experiments. We observed the importance of stochasticity in the logging function (Sect. 3.5.1), discussed our approach to train/test splitting when using multiple logging policies (Sect. 3.5.2), and explained how we constructed a logging policy (Sect. 3.1.1). Our experiments in Sect. 3.2 demonstrated the value of the snIPS estimator when data are limited. Moreover, we identified a post-training model updating procedure in Sect. 4 which has practical business value. On the other hand, some of our learnings are not tied to any one experiment. In this section, we collect a series of anecdotes from our experience applying off-policy techniques to the notification component of a chatbot. We identify a few practical steps a practitioner can take around model definition and validation, and in addition, we discuss when to apply these learnings.

5.1 Importance of cannel awareness

An important variable we identified to control for is the channel that the customer is using. Our customers can access the Xfinity Assistant through a variety of channels including the web, multiple phone applications, SMS, or by using their television box. Our data show that each channel has different priors in regards to the likelihood to accept a notification. Without controlling for which channel the customer used, we struggled to improve our models. We found that troubleshooting was rare in certain channels, but due to the use of channel-agnostic exploration policies, these rare, negative cases were over-represented in our training data. By controlling for the channel, we were able to adjust the prior distribution for each action, leading to improvements in model performance.

In Fig. 6, we demonstrate how we added the channel feature to the model. The basic approach is similar to a fixed effects model from statistics where each population has its own bias term in a regression formula. The channel information was input into the model via a one-hot encoded vector. The vector is passed through a linear layer to add a channel-related term to each of the three outputs in order to control for the behavior of different populations. Here is a quantitative example of its impact: We observed a higher number of internet troubleshooting instances through our web-based chat than other channels. Thus,

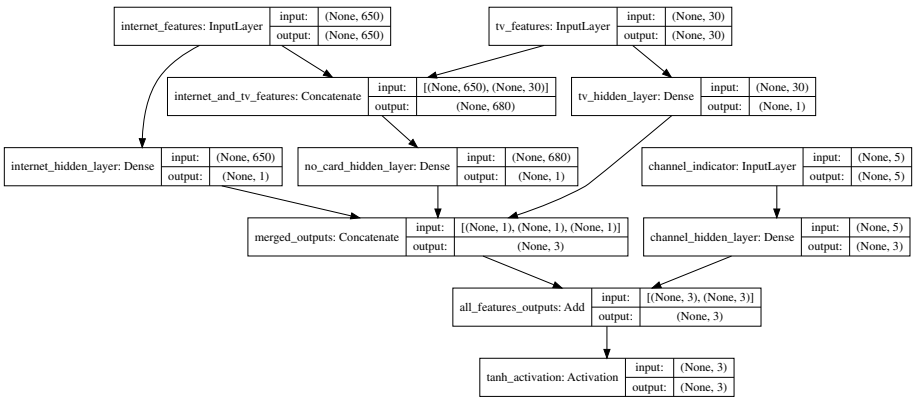

Fig. 6 A plot of the model architecture after adding the channel feature

the term associated with the web-based chat for the internet troubleshooting action had a positive value, whereas other channels with a lower prior rate of internet troubleshooting had negative values.

5.2 Data gathering validation

Ad-hoc sanity checks are important for validating the overall validity of the data gathering mechanism. Section 3.3 of Li et al. (2015) outlines statistical tests for identifying errors in propensity scores, but in our case, we ran into more obvious propensity score errors long before we performed statistical tests. For example, upon release of our uniformly random action policy, we found that one action occurred at three times the rate of the other actions. This was clearly wrong, and it identified an error in the logging of customer sessions. Later, we noticed a smaller but still obvious discrepancy: in select cases the application was overriding predictions from the production policy, and those records were ultimately excluded from our data processing for training.

We note that sanity checks surrounding the randomization and expected proportions of actions are not sufficient. There was a nontrivial number of instances when our notifications were combined with other notifications, which was not immediately apparent from our existing logging pipeline. (In fact, these dual notifications were only discovered when a researcher accessed the Xfinity Assistant while troubleshooting their service.) Data points with multiple notifications disrupted our off-policy estimates and had to be excluded from training and evaluation until proper features were available to control for this situation.

5.3 How we added new features to the production policy

Much of the literature is focused on contextual bandit models in larger action spaces and approaches to add new bandit arms to existing policies (i.e., news recommendation). In our case, we consider improving the results of a policy on a small action space by adding new features while the action space remains fixed. For models which output probabilities, it is common to train a new policy close to the existing policy, and varying techniques are used to enforce this constraint (Schulman et al. 2015, 2017) when iterating on a model. Approaches to keep the result of retraining a reward predictor model "close" to an existing model are less common, and so we present ours here.

Anecdotally, we found our results varied widely when training on different data sets, likely due to the small size of our training data. Another complication was that data sets varied in terms of feature inclusion as well as size. When new features were added, their data were missing on older, historical examples. Thus, new models trained only on data with all features were at a disadvantage. To remedy this, we chose to keep our new candidate policy close to the existing production policy through a combination of freezing model weights for longstanding features and strong regularization of new features. This approach implicitly utilizes continuity to keep the two models' outputs "close" by limiting how far new features could alter the output of the new model.

The architecture we have defined in Sect. 2.3 is flexible in order to add a new action or a new source of features while still maintaining the old weights or freezing the other feature weights during training. One of the first features we added was a channel indicator feature. When doing so, we froze other feature weights and trained the channel weights to improve performance. The channel feature should impact all actions, and so its output is of size 3 and added to the last step before the activation. This can be seen in Fig. 6.

A key strength of our strategy is that it easily fits into most training procedures for neural networks without significant additional development. Other approaches to keep a reward predictor-style model "close" to another could require solving optimization problems or enforcing constraints during model updates, which are more complicated to implement and scale poorly as the data size grows.

6 Summary

When it comes to practical examples in the off-policy evaluation literature, the action space in a typical contextual bandit application is large (Li et al. 2010; Chen et al. 2018; Li et al. 2011, 2015). Many papers have written about news recommendation and search engine personalization. In such use cases, a contextual bandit model always returns a recommendation. That is to say, there is no benefit or option for the model to return a null operation. A major difference between this work and previous works is that our use case has a null action and a limited number of bandit arms. When combined with our noisy reward signal, these two aspects of our problem led to unique challenges during training, particularly trained models which only select the null action or generalize poorly. We have made several modifications to our model selection procedure to address the issues observed while training bandit models on our data. In this work, we detailed the PRDH procedure which modifies a contextual bandit model post-training in order to optimize performance. To our knowledge, we have not seen such an approach employed before to mitigate these specific complications.

Moreover, we have presented a detailed case study of using different logging policies for off-policy evaluation on our use case. We found propensity weights and off-policy techniques broadly to be useful for both model optimization and evaluation, but only when the stochasticity is properly incorporated. Our experiments utilized two off-policy estimators: both the IPS estimator of Strehl et al. (2010), Swaminathan and Joachims (2015) and snIPS estimator found in Swaminathan and Joachims (2015). In theory, both estimate the true reward for a policy, but unfortunately, we found the variance of the IPS estimator to be too high to be useful in our data regime. On the other hand, snIPS would generalize to production even if there was not a large effective sample size. Our results also show that our approach to defining a simplified exploration policy can be utilized for accurate off-policy estimates while improving the performance of the logging policy on key business metrics.

Acknowledgements We would like to thank Ferhan Ture for detailing the methods used in the production NLP system powering the Xfinity Assistant. In addition, we thank our colleagues on the AI for CX team at Comcast for giving feedback on early drafts of this work, particularly Hongcheng Wang, Yonatan Vaizman, and Rama Mahajanam. We finally thank Dave Monnerat for providing the screenshot of the Xfinity Assistant.

Author Contributions Scott Rome wrote and revised the manuscript, designed and implemented the research, and analyzed the results. Tianwen Chen contributed to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript. Michael Kreisel contributed to design of the research and the writing of the manuscript. Ding Zhou contributed to the analysis of the results.

Funding Not applicable

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Agarwal, A., Basu, S., Schnabel, T., & Joachims, T. (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*, pages 687–696.

Aly, M. (2005). Survey on multiclass classification methods. *Neural Networks, 19,* 1–9.

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Ed Snelson, et al. (2013). Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research 14*(65), 3207–3260.

Cer, D., Yang, Y., Kong, S. Y., Hua, N., Limtiaco, N. L. U., St. John, R., Constant, N., Guajardo-Céspedes, M., Yuan, S., Tar, C., Hsuan Sung, Y., Strope, B., & Kurzweil, R. (2018). Universal sentence encoder. In *EMNLP Demonstration*. Brussels, Belgium. https://www.aclweb.org/anthology/D18-2.pdf.

Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., & Ed H. Chi. (2018). Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, pages 456–464, New York, NY, USA: ACM.

Chollet, F. et al. (2015). Keras. https://keras.io.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communication of the ACM, 13*(2), 94–102.

Hanna, J. P., Thomas, P. S., Stone, P., & Niekum, S. (2017). Data-efficient policy evaluation through behavior policy search. In D. Precup, Y. W. Teh (Eds.), In*Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1394–1403, International Convention Centre, Sydney, Australia, 06–11. PMLR.

Ionides, E. L. (2008). Truncated importance sampling. *Journal of Computational and Graphical Statistics, 17*(2), 295–311.

Jeunen, O., Mykhaylov, D., Rohde, D., Vasile, F., Gilotte, A., & Martin, B. (2019). *Learning from bandit feedback: An overview of the state-of-the-art*.

Joachims, T., & Swaminathan, A. (2016). Sigir tutorial on counterfactual evaluation and learning for search, recommendation and ad placement. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1199–1201. ACM.

Karampatziakis, N., Kochman, S., Huang, J., Mineiro, P., Osborne, K., & Chen, W. (2019). Lessons from contextual bandit learning in a customer support bot. arXiv preprint arXiv:1905.02219.

Lample, G., Conneau, A., Denoyer, L., & Ranzato, M. (2018). Unsupervised machine translation using monolingual corpora only. *International Conference on Learning Representations*.

Li, L., Chen, S., Kleban, J., & Gupta, A. (2015). Counterfactual estimation and optimization of click metrics in search engines: A case study. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 929–934, New York, NY, USA: ACM.

Li, L., Chu, W., Langford, J., Moon, T., & Wang, X. (2012) An unbiased offline evaluation of contextual bandit algorithms with generalized linear models. In D. Glowacka, L. Dorard, & J. Shawe-Taylor (Eds.), *Proceedings of the Workshop on On-line Trading of Exploration and Exploitation 2*, volume 26 of *Proceedings of Machine Learning Research*, pages 19–36, Bellevue, Washington, USA, 02: PMLR.

Li, L., Chu, W., Langford, J., & Schapire, R.E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA: ACM.

Li, L., Chu, W., Langford, J., & Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 297–306, New York, NY, USA: ACM.

Li, L., Munos, R., & Szepesvári, C. (2015). Toward minimax off-policy value estimation. 01, pages 608–616.

Patro, A., Govindan, S., & Banerjee, S. (2013). Observing home wireless experience through wifi aps. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, page 339–350, New York, NY, USA: Association for Computing Machinery.

Pinckernell, N., & Rome, S. (2020). Operationalizing streaming telemetry and machine learning model serving: Customer experience automation. In *Proceedings of SCTE•ISBE Cable-Tec Expo, Fall Technical Forum*.

Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P. (2015). Trust region policy optimization. *CoRR*, abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Strehl, A., Langford, J., Li, L., & Kakade, S.M. (2010). Learning from logged implicit exploration data. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 23*, pages 2217–2225. Curran Associates, Inc.

Swaminathan, A., & Joachims, T. (2015). Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research, 16*(52), 1731–1755.

Swaminathan, A., & Joachims, T. (2015). The self-normalized estimator for counterfactual learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28*, pages 3231–3239. Curran Associates, Inc.

Swaminathan, A., Krishnamurthy, A., Agarwal, A., Dudík, M., Langford, J., Jose, D., & Zitouni, I. (2016). Off-policy evaluation for slate recommendation. *CoRR*, abs/1605.04812.

Vlassis, N., Bibaut, A., Dimakopoulou, M., & Jebara, T. (2019). On the design of estimators for bandit off-policy evaluation. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6468–6476, Long Beach, California, USA, 09–15: PMLR.