MKTG 596: Adaptive Experimental Design Methods,Homework 2

Lalit Jain

May 23, 2023

In this assignment, we will get our hands dirty with off-policy evaluation. A ipynb file has been included to help guide your way through this assignment. If you don't have a local Jupyter installation ,you can upload it to Google Colab. You should feel free to modify any code that is included (except for the logging functions) to help with the assignment.

1 Off-Policy Evaluation: Warm-up

In the method `logging_data_tabular(n, num_contexts, num_actions, seed)`, I have provided code to generate logged data where

- The context distribution is uniform over `num_contexts`
- The reward matrix `r` is `num_contexts x num_actions` and is chosen from a uniform distribution on [0,1].
- The action distribution of the logging policy $\pi(\cdot|x)$ is chosen arbitrarily.
- The returning logged data of size n is generated by sampling a random context, and then a random action from $a \sim \pi_0(\cdot|x)$ and then observing a noisy reward $\max(r(x,a) + .1\epsilon, 0)$ where $\epsilon \sim N(0,1)$

Note that I have set a seed of 10 to ensure the same reward function and logging distribution each time `logging_data_tabular` is called, but an arbitrary seed can be set to generate the actual logged data. This allows variation between runs.

I've also implemented a policy class *UniformPolicy*. This class provides the following methods that can return a policy π_{unif}.

- `pi_a_x(ctx, action)` returns $\pi(a|x)$ which for the uniform policy is $1/|\mathcal{A}|$.
- `V(r)` returns $V(\pi)$ which for a deterministic policy is $V(\pi) = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}}\pi(a|x)r(x,a)$

For this warm-up exercise we are going to estimate $\hat{V}(\pi_{unif})$ using three different methods. Before you start the exercise, please read all the provided code carefully.

1. Implement for a general policy:
   - IPS: $\hat{V}_{IPS}(\pi) = \frac{1}{n}\sum_{t=1}^{n}\frac{\pi(a_t|x_t)}{\pi_0(a_t|x_t)}r_t$

- DM: $\hat{V}_{DM}(\pi) = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}}\pi(a|x)\hat{r}(x,a)$; where

$$\hat{r}(a,x) = \frac{1}{\sum_{t=1}^{n}\mathbf{1}\{x_t = x, a_t = a\}}\sum_{t=1}^{n}\mathbf{1}\{x_t = x, a_t = a\}\, r_t,$$

  that is the average of rewards observed when $x_t = x, a_t = z$.
- DR: $\hat{V}_{DR}(\pi) = V_{DM}(\pi) + \frac{1}{n}\sum_{t=1}^{n}\frac{\pi(a_t|x_t)}{\pi_0(a_t|x_t)}(r_t - \hat{r}(x_t,a_t))$

You can assume that your policy has the same methods as the uniform policy (make this code as general as possible, you will need it in the next exercise).

2. Run a simulation. For each $n \in [100 \cdots, 1000]$ draw a logged data set from `logging_data_tabular` with 50 contexts and 2 actions. Then using IPS, DM, and DR, estimate the value of the uniform policy and compute squared error between your estimate and the true value $V(\pi_{unif})$. Repeat this 20 times, and compute the average and standard error of your results for each estimator. Then plot your results including error bars. This should result in three different curves with error bars.

3. What do you notice? Which estimator has the most bias? Which has the most variance? Which has the lowest mean squared error? Which has the highest?

2 Off-Policy Evaluation: Learning

In this exercise, we will focus on learning policies from logged data. We will focus on deterministic policies, i.e. we are trying to learn a map $\pi : \mathcal{X} \to \mathcal{A}$ with the highest value of $V(\pi)$ (with the exception of the logging policy π_0 which is still assumed to be randomized). In the method `logging_data_logistic(n, num_contexts, num_actions, dim, seed)`, I have provided code to generate logged data where

- The context distribution is uniform over `num_contexts`
- The dimension d of the context vectors and actions is `dim`.
- The context vectors are chosen from a $N(0, I) \subset \mathbb{R}^d$.
- The action vectors are chosen from a $N(0, I) \subset \mathbb{R}^d$.
- The reward matrix `r` is `num_contexts x num_actions` and is computed as $r(x, a) = \max(x^\top\theta, 0)$ where $\theta \sim N(0, I) \subset \mathbb{R}^{d\times d}$
- The logging policy is chosen from a softmax policy based on the reward function. Namely $\pi_0(a|x) \propto \exp(\eta r(x, a) + .05\epsilon)$ where $\epsilon \sim N(0, 1)$. This ensures that the logging policy is close the optimal deterministic policy - but perhaps not too close.
- The returning logged data of size n is generated by sampling a random context, and then a random action from $a \sim \pi_0(\cdot|x)$ and then observing a noisy reward $\max(r(x, a) + .1\epsilon, 0)$ where $\epsilon \sim N(0, 1)$

Note that I have set a seed of 10 to ensure the same reward function and logging distribution each time `logging_data_logistic` is called, but an arbitrary seed can be set to generate the actual logged data. This allows variation between runs.

I've also implemented a policy class `LogisticDeterministicPolicy`. We are going to take an abstraction which allows us to take a multi-class classifier $f : \mathcal{X} \sim \mathcal{A}$ implemented with a `predict` function and allow us to treat it as a policy. This class provides the following:

- `pi_a_x(ctx, action)` returns $\pi(a|x)$ which for a deterministic policy is 1 when $a = \pi(x)$ and 0 otherwise.
- `V(pi)` returns $V(\pi)$ which for a deterministic policy is $\frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} r(x, \pi(x))$

1. First we will implement the function `ipw_learner`. Recall the following reduction that allows us to do policy learning using access if we have access to learn a weighted classifier:

$$\begin{aligned}
\hat{\pi} &= \arg\max_{\pi \in \Pi} \hat{V}_{IPS}(\pi) \\
&= \arg\max_{\pi \in \Pi} \sum_{t=1}^{T} \frac{\mathbf{1}\{\pi(x_t) = a_t\}}{\pi_0(a_t|x_t)} r_t \\
&= \arg\min_{\pi \in \Pi} \sum_{t=1}^{T} \frac{r_t}{\pi_0(a_t|x_t)} \mathbf{1}\{\pi(x_t) \neq a_t\}
\end{aligned}$$

   Given a training procedure to learn a classifier given training data $X = \{x_t\}_{t=1}^n$, $Y = \{a_t\}_{t=1}^n$ and sample weights $\{r_t/\pi_0(a_t|x_t)\}_{t=1}^n$, this allows us to learn a policy from the underlying training procedure.

   For the underlying classifier class, feel free to play around with different options, though it may be easiest to get `LogisticRegression` in `sklearn` working first. The resulting model you generate should be used to create a `LogisticDeterministicPolicy` which is subsequently returned by the `ipw_learner` method.

2. Run a simulation. For each $n \in [100 \cdots, 1000]$ draw a logged data set from `logging_data_logistic` with 50 contexts and 2 actions. Then learn a policy using the output of `ipw_learner` , namely, $\hat{\pi}$. Finally, using all three estimators IPS, DM, and DR (as implemented in the previous exercise), estimate the value of your estimated policy and compute squared error between your estimate and the true value $V(\hat{\pi})$. Repeat this 20 times, and compute the average and standard error of your results for each estimator. Then plot your results including error bars. This should result in three different curves with error bars.

   I highly recommend taking a value of $n = 500$ and debugging your implementation of `ipw_learner` using a logged dataset carefully before running the above simulation (or you may get very wrong results). I've provided some code. To debug, compare $V(\hat{\pi})$ as computed by the `V` function in the `LogisticDeterministicPolicy` class.

   Note that if we were being more careful, we would take our logged dataset and provide a train and test set.

3. What do you notice? Which estimator has the most bias? Which has the most variance? Which has the lowest mean squared error? Which has the highest? How do your estimates compare to the best-optimal policy if you had the reward function?