README File for Pytorch Version DML

Introduction. This writeup describes the pytorch version codes of estimating provider profiling. It is implemented fully respects [Liu et al., 2021]. Three parts are contained, first introduces the code structures, hyperparameters and how to run the code for simulations. Second part shows the simulation settings, Third part shows the hyperparameters for sample runs on USRDS data. Tests on real data fully respects the code structures in first part with little modification. For code details, please check `https://github.com/jiayichengcode/provider_dml`.

Code Structure. The main part of the codes is in folder **dml_utils**, which contains three files:

- dgp.py, which generates synthetic data, not used for real data setting.
- nuisance.py, which contains four types of MLP to estimate four nuisance functions, see Simulation Setup for details.
- dml.py, which make uses of the nuisance functions results and estimate the final beta. It also contains bootstrap codes.

dml_simulation.py calls functions above to run the simulation. read_usrds_data.ipynb runs this algorithm on USRDS data. The whole package relies heavily on pytorch, and GPU is the preferred device to run these code.

Simulation Setup. We first address the nuisance functions used in this algorithm, then follows the order in above section to introduce how to run the repo.

As in [Liu et al., 2021], we consider a logistic partially linear model. Let $\{(Y_i, A_i, \mathbf{X}_i) : i = 1, 2, \ldots, n\}$ be independent and identically distributed samples of $Y \in \{0, 1\}$, $A \in \mathbb{R}$ and $\mathbf{X} \in \mathbb{R}^p$. Assume that

$$\mathbb{P}(Y = 1 \mid A, \mathbf{X}) = \text{expit}\{\beta_0 A + r_0(\mathbf{X})\}, \tag{1}$$

Here $r_0(x)$ is one of the outputs in DML function in dml.py with the name rXp. It's used for moment condition to estimate β_0. Another output for DML is named mXp, which is $m_0(x) = \mathbb{E}(A|Y = 0, X = x)$. With these two intermediate variables, we can use the moment condition:

$$\frac{1}{n}\sum_{i=1}^{n} \hat{\phi}(\mathbf{X}_i)\left\{Y_i e^{-\beta A_i - r(\mathbf{X_i})} - (1 - Y_i)\right\}\{A_i - m(\mathbf{X_i})\} = 0 \tag{2}$$

to estimate β. In our implementation, for simplicity, $\hat{\phi}(\mathbf{X}_i) = \text{const}$. However, r and m are not implemented naively, but with the help of many other nuisance functions.

Another thing worth noting is there are two folds split in this algorithm, one outer and one inner. As in the paper, for now we make them both K=5 folds, and call outer fold k and inner fold i.

For each inner fold i in outer fold k, we estimate two nuisance functions: NuisanceModelY (E(Y|A,X)) and NuisanceModelAtemp (E(A|X)) on inner folds except i, and use nuisance model to get estimated values for inner fold i (called Wp_preds and ap_preds in codes) and outer fold k (called aBar).

Then calculate logits of Wp_preds, estimate an intermediate β called betaNi from

$$\beta_0 = \text{argmin}_{\beta\in\mathbb{R}}\mathbb{E}\left[\{\text{logit}\, M_0(A, \mathbf{X}) - \beta\,(A - \mathbb{E}[A \mid \mathbf{X}])\}^2\right] \tag{3}$$

This betaNi is not our final estimate. We then estimate nuisance function NuisanceModeltnk (logits $\sim$ X) for every outer fold, named it tnk. Then we calculate rnk (rXp as well) through $rnk = tnk - betaNi * \mathbb{E}(aBar)$ since logit $M_0 = \mathbb{P}(Y = 1|A, \mathbf{X}) = \beta A + r_0(\mathbf{X})$. Now we finish rXp part.

For mXp part, we use a new nuisance model NuisanceModelAY0 which estimates E(A|Y=0,X), train on all data but outer fold k to calculate outer fold k estimates. This is mXp.

With rXp and mXp, we use 2 with root finding in scipy package to get beta with truncating rXp to the range $(-\infty, 2)$. For bootstrap, as suggested in the paper, we just replace $\hat{\phi}(\mathbf{X}_i)$ with gaussian noise.

Goes back to data generation, we generate r_0 and m_0 (called $r0$, $A0$ in dgp.py) from gaussian noise X with covariance matrix all 0.2 except for the diagonal with highly nonlinear functions, and generate A and Y using respective Bernoulli distribution. For every simulation, we generate 5000 samples, X dimension 20.

To summarize, there are four nuisance models in total: **NuisanceModelY, NuisanceModelAtemp, NuisanceModeltnk, NuisanceModelAY0**. First two will be trained K^2 times for each simulation, while last two will be trained K times.

Nuisance models are all four layer MLP, with each layer input_dim to 128, 128 to 64, 64 to 64 and 64 to 1. Optimizers are all Adam, with learning rate 1e-3. We use batch size 64 and epochs 600 for simulations. We also use early stopping for faster training, with patience step 50 and delta 1e-4. We generate true beta as a $Unif(-2, 2)$, and run simulation 1000 times, and get the following distribution plot:

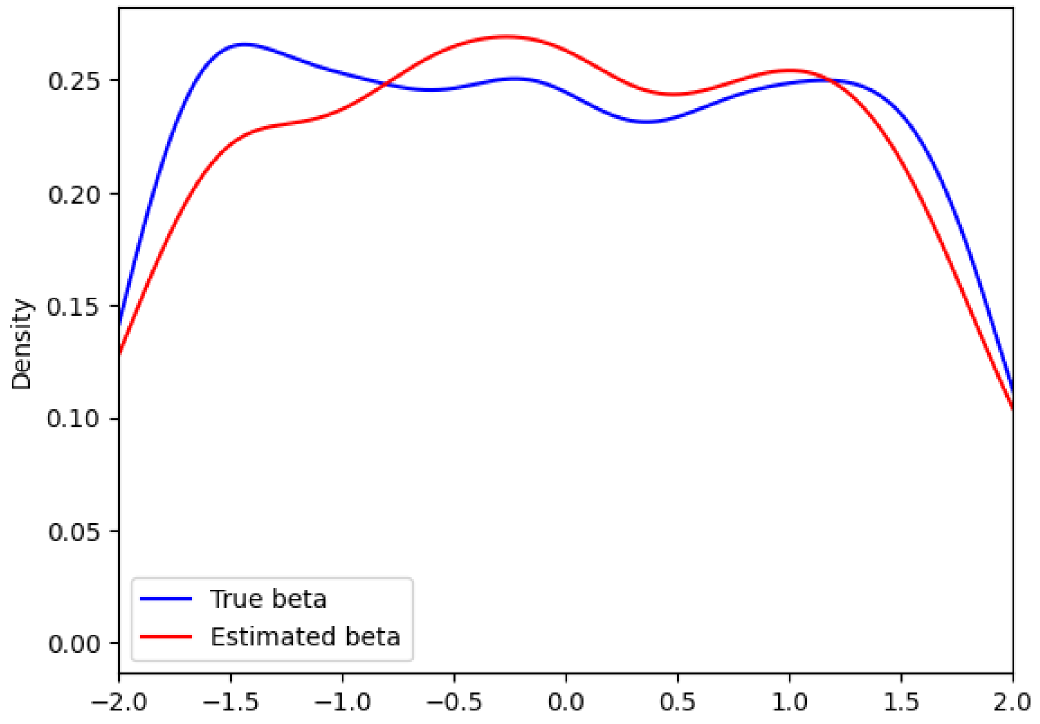

Figure 1: Simulation Results Distribution versus Real Beta Distribution

Real Data Settings. Code structures and function meanings are exactly the same as above. We choose PROVUSRD equals 186 and 6627 two providers to test our codes. The choice is made since they are two providers with the most patients in the data (3301 and 2761). We estimate the treatment effects by running twice, one make A for 3301 all 1 and 2761 0, and vice versa. Only difference here is we make epochs bigger to 2048. We get estimated beta for 3301 to be 0.825, and for 6627 -0.255. However, if I add more A=0 samples from group 2632 (2153 patients), then the beta estimate for

3301 becomes 0.317, meaning the result quite sensitive to data selection. I will try to use all data as 0 examples then estimate betas for groups with sufficient patients later.

References

Molei Liu, Yi Zhang, and Doudou Zhou. Double/debiased machine learning for logistic partially linear model. *The Econometrics Journal*, 24(3):559–588, 2021.