

# Causal Machine Learning: Homework 2

*Due date: October 6, 11:59PM*

Your name:

```
library(hdm)
library(keras3)
library(dplyr)
library(sandwich)
```

## Potential Outcome Framework

1. In lecture, we showed that the observed difference in means (Average Predictive Effect, APE)

$$\pi = E[Y_i | D_i = 1] - E[Y_i | D_i = 0]$$

can be decomposed into the Average Treatment Effect on the Treated (ATT) plus a selection bias term. Show that  $\pi$  can equivalently be decomposed into the Average Treatment Effect on the Controls (ATC) plus a selection bias term. In words, explain the meaning of this alternative selection bias.

2. Suppose treatment status  $D_i$  is independent of potential outcomes  $Y_i(d)$ ,  $d \in \{0, 1\}$  conditional on a set of covariates  $X_i$ . Define the Conditional ATE (CATE) as

$$\delta(X_i) = E[Y_i(1) | X_i] - E[Y_i(0) | X_i]$$

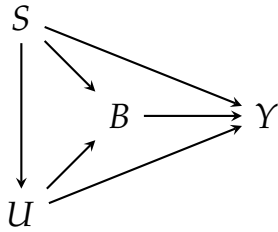
Show that

$$\delta(X_i) = Y_i \left[ \frac{\mathbb{1}(D_i = 1)}{P(D_i = 1 | X_i)} - \frac{\mathbb{1}(D_i = 0)}{1 - P(D_i = 1 | X_i)} \right]$$

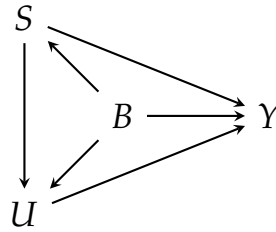
where  $P(D_i = 1 | X_i)$  is the probability of having  $D_i = 1$  given  $X_i$ .

## Directed Acyclic Graphs (DAG)

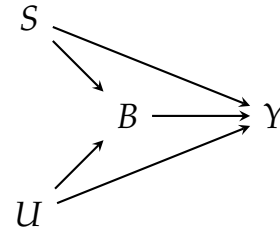
1. The DAGs below graphically describe 3 hypothetical Structural Equation Models (SEMs). state which set of covariates need to be included to identify the causal effect of  $B$  on  $Y$ .



Case 1



Case 2



Case 3

2. Now focus on Case 3. Suppose the SEM has the following data-generating process

$$Y := S + B + \kappa U + \epsilon_Y$$

$$B := S + U + \epsilon_B$$

$$S := \epsilon_S$$

$$U := \epsilon_U$$

where  $\epsilon_U$ ,  $\epsilon_Y$ ,  $\epsilon_S$ , and  $\epsilon_B$  are independent  $\mathcal{N}(0, 1)$  shocks. Show the following equation holds, and explain the implication of the equation.

$$E[Y | S, B] = S + B + (1 - \kappa/2)S + (1 + \kappa/2)B$$

## Test of Convergence Hypothesis

In lecture, we provided an empirical example of partialling-out with Double LASSO to estimate the regression coefficient  $\beta_1$  in the high-dimensional linear regression model. Specifically, we estimated how the rates at which economies of different countries grow ( $Y_i$ ) are related to the initial wealth levels in each country ( $D_i$ ) controlling for country's institutional, educational, and other similar characteristics ( $X_i$ ). The relationship is captured by  $\beta_1$ , the *speed of convergence/divergence*, which measures the speed at which poor countries catch up ( $\beta_1 < 0$ ) or fall behind ( $\beta_1 > 0$ ) rich countries, after controlling for  $X_i$ .

In this assignment, I want you to use the formal procedure of Double Machine Learning to re-estimate  $\beta_1$ . Specifically, I want you fine-tune a shallow one-layer Neural Network to estimate the nuisance function for  $Y_i$  and  $D_i$  using  $X_i$ , and residualize  $Y_i$  and  $D_i$  using cross-fitting.

```
library(hdm)
## function to get data
getdata <- function(...) {
  e <- new.env()
  name <- data(..., envir = e)[1]
  e[[name]]
}

## now load your data calling getdata()
growth <- getdata(GrowthData)

## create the outcome variable y, treatment d, and control covariates x
y <- growth$Outcome
x <- growth[-which(colnames(growth) %in% c("Outcome", "intercept", "gdpsh465"))]
d <- growth$gdpsh465
```

I have already implemented a Neural Network model using keras3 and performed a manual grid search to fine-tune its hyperparameters. The tuning procedure evaluated each candidate model on a 20% validation split, and the best configuration was chosen based on the lowest validation loss. The default hyperparameter grid includes:

- Number of hidden neurons (units): 10, 50, 100
- Learning rate (lr): 0.001, 0.02, 0.05
- Batch size: 15 (no tuning)
- Number of epochs: 50, 100, 200, 400

This setup allowed the model to be optimized over different network sizes, learning speeds, mini-batch sizes, and training durations. I also used early stopping based on 10 epoch's validation loss.

```

## build and fit a keras model
build_and_fit <- function(x, y, units, lr, batch_size, epochs) {
  model <- keras_model_sequential() |>
    layer_dense(units = units, activation = "relu", input_shape = c(ncol(x))) |> ## hidden
    layer_dense(units = 1) |>
    compile(
      optimizer = optimizer_adam(learning_rate = lr), ## default optimizer is Adam
      loss = "mse"
    )

  ## fit the model
  history <- model %>% fit(
    x = as.matrix(x),
    y = y,
    batch_size = batch_size,
    epochs = epochs,
    validation_split = 0.2,
    callbacks = list(
      callback_early_stopping(
        monitor = "val_loss",
        patience = 10, ## early stopping criterion
        restore_best_weights = TRUE
      )
    ),
    verbose = 0
  )

  ## return the loss of the last validation set
  val_loss <- tail(history$metrics$val_loss, 1)

  list(model = model, val_loss = val_loss)
}

## grid search for hyperparameters
tune_keras <- function(x, y,
                      units_grid = c(10, 50, 100),

```

```

        lr_grid = c(0.001, 0.02, 0.05),
        batch_grid = c(15),
        epoch_grid = c(50, 100, 200, 400)) {
results <- expand.grid(units = units_grid,
                      lr = lr_grid,
                      batch_size = batch_grid,
                      epochs = epoch_grid)
results$val_loss <- NA

for (i in 1:nrow(results)) {
  fit <- build_and_fit(x, y,
                      units = results$units[i],
                      lr = results$lr[i],
                      batch_size = results$batch_size[i],
                      epochs = results$epochs[i])
  results$val_loss[i] <- fit$val_loss
}

## find the best combination of hyperparameter
best <- results[which.min(results$val_loss), ]
return(best)
}

```

Now I want you to fine-tune the NN model *before* cross-fitting. That is, do not fine-tune the model within each  $K - 1$  fold, but only fine-tune model once based on the whole sample. Use the default grid search for  $Y$ , but use the following grid search for  $D$ :

- Number of hidden neurons (units): 10, 50, 100
- Learning rate (lr): 0.001, 0.02, 0.05
- Batch size: 15 (no tuning)
- Number of epochs: 500, 1000, 2000

Then use the NN model with the combination of hyperparameters that reports the lowest validation loss to estimate the nuisance function. You may find such model by calling

```

ymodel <- build_and_fit(
  x[-I[[b]], ], y[-I[[b]]], ## suppose you are retrieving the nuisance function for
  units = best_y$units,
  lr = best_y$lr,
  batch_size = best_y$batch_size,
  epochs = best_y$epochs
)$model

```

Residualize your  $Y$  and  $D$  and report the point estimate of  $\beta_1$  and its standard error using the standard DML pipeline. What can you conclude from this practice? Note this may take a while for R to complete.

```

## DML using NN - your code here
dmlNN_for_plm <- function(x, d, y, nfold = 5) {

}

## execute your DML NN function
set.seed(1)
dmlNN_for_plm(x, d, y, nfold = 5)

```