Report Monotonicity Extraction

Yichen Han

2025-01-20

```
library(AER)
library(haven)
library(tidyverse)
library(ddml)
library(ranger)
```

Card (1995)

Adapted analysis plan.

- $N = 3010$ individuals
- Y: continuous, log hourly wage
- T: binary, received education > 12 yrs
- Z: binary, presence of college nearby
- X: 4 binary covariates: work experience > 8 yrs, 3 indicators for race and location of residence.

```
card <- read_dta("https://raw.github.com/scunning1975/mixtape/master/card.dta")
suppressMessages(attach(card))
Y <- lwage
D <- as.numeric(educ > 13)
expbin <- as.numeric(exper > 8)
X <- cbind(expbin, black, south, smsa)
Z <- nearc4
idx_complete <- complete.cases(Y, D, X, Z)
Y <- Y[idx_complete]
D <- D[idx_complete]
X <- X[idx_complete, ]
Z <- Z[idx_complete]
df <- data.frame(Y, D, Z, X)
```

If saturated, the data is:

```
X1 <- X[, 1]
X2 <- X[, 2]
X3 <- X[, 3]
X4 <- X[, 4]
X_full <- cbind(X, X1 * X2, X1 * X3, X1 * X4, X2 * X3, X2 * X4, X3 * X4,
                X1 * X2 * X3, X1 * X2 * X4, X1 * X3 * X4, X2 * X3 * X4,
                X1 * X2 * X3 * X4)
Z_int <- cbind(Z, Z * X_full)
```

DDML

For simplicity, we omit `ivreg`, and only report outputs from DDML.

```
### Estimation DDML
# -------------- estimating beta_rich using DDML
set.seed(123)
# Estimate the local average treatment effect using short-stacking with base
#     learners ols, rlasso, and xgboost.
learners_multiple <- list(list(fun = ols),
                          list(fun = mdl_glmnet),
                          list(fun = mdl_ranger),
                          list(fun = mdl_xgboost))
pliv_rich <- ddml_pliv(Y, D, Z, X,
                           learners = learners_multiple,
                           ensemble_type = c('nnls', 'nnls1', 'ols','average'),
                           shortstack = TRUE,
                           sample_folds = 10,
                           silent = TRUE)
summary(pliv_rich)
```

```
## PLIV estimation results:
##
## , , nnls
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.00112    0.00928   0.121   0.9040
## D_r          0.91380    0.43327   2.109   0.0349
##
## , , nnls1
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 5.12e-05    0.00926 0.00553    0.996
## D_r         9.14e-01    0.43338 2.10889    0.035
##
## , , ols
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.000977    0.00926   0.105   0.9160
## D_r         0.912894    0.43388   2.104   0.0354
##
## , , average
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 7.99e-05    0.00924 0.00865   0.9931
## D_r         9.08e-01    0.43141 2.10440   0.0353
```

```
# -------------- estimating LATE using DDML
late_fit_short <- ddml_late(Y, D, Z, X,
                           learners = learners_multiple,
                           ensemble_type = c('nnls', 'nnls1', 'ols','average'),
                           shortstack = TRUE,
                           sample_folds = 10,
```

```
                              silent = TRUE)
summary(late_fit_short)
```

```
## LATE estimation results:
##
##         Estimate Std. Error t value Pr(>|t|)
## nnls       0.670      0.323    2.08   0.0378
## nnls1      0.671      0.322    2.08   0.0374
## ols        0.696      0.330    2.11   0.0349
## average    0.668      0.320    2.09   0.0367
```

Weight Estimation

Note that DDML-PLIV outputs β_{rich}, and for the binary case

$$\beta_{\text{rich}} = \mathbb{E}\left[\beta_{\text{acr}}(X)\frac{P[T(1) > T(0)|X]\mathbb{V}[Z|X]}{\mathbb{E}[P[T(1) > T(0)|X]\mathbb{V}[Z|X]]}\right]$$

where $\beta_{\text{acr}}(X) := \mathbb{E}[Y(T(1)) - Y(T(0))|T(1) > T(0)]$.

We regard $\frac{P[T(1)>T(0)|X]\mathbb{V}[Z|X]}{\mathbb{E}[P[T(1)>T(0)|X]\mathbb{V}[Z|X]]}$ as the weights.

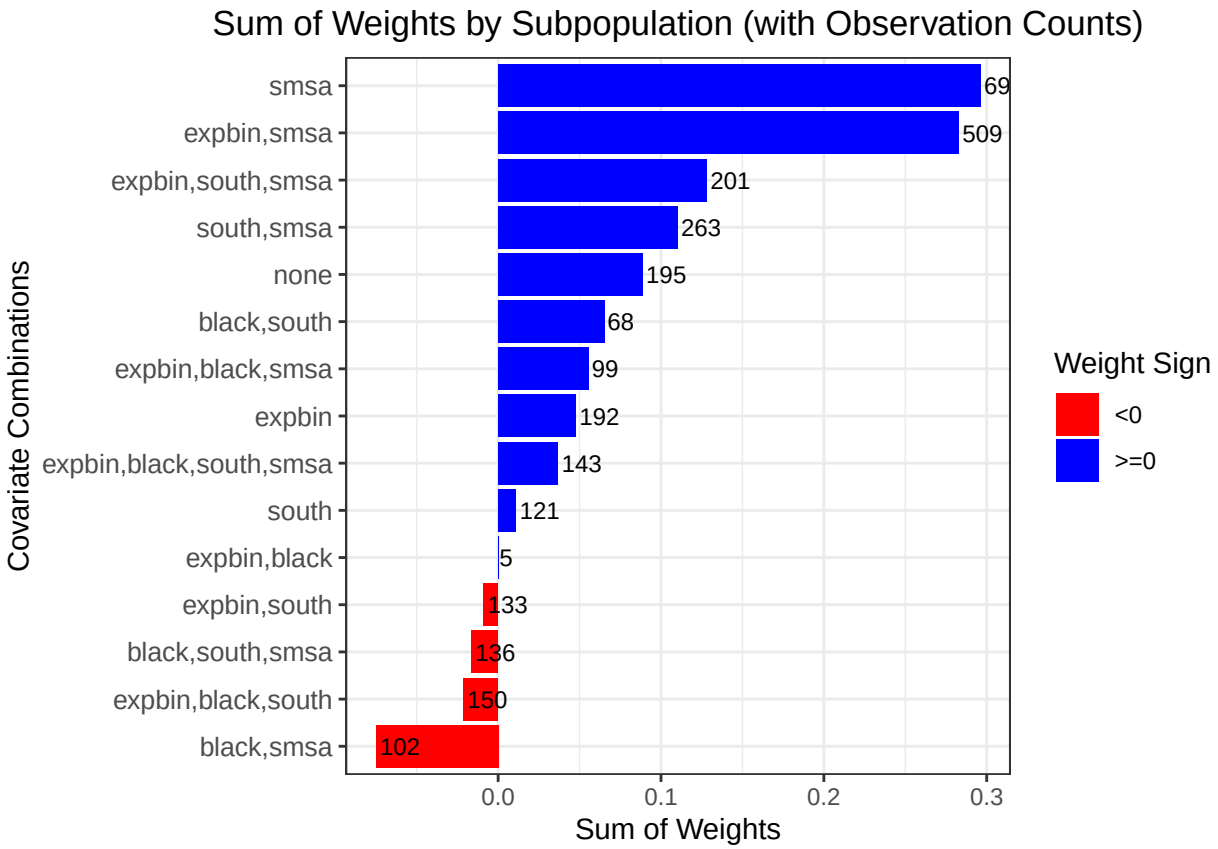
We estimate $P[T(1) > T(0)|X]$ and $\mathbb{V}[Z|X]$ using two random forests.

First, define the function to estimate weights.

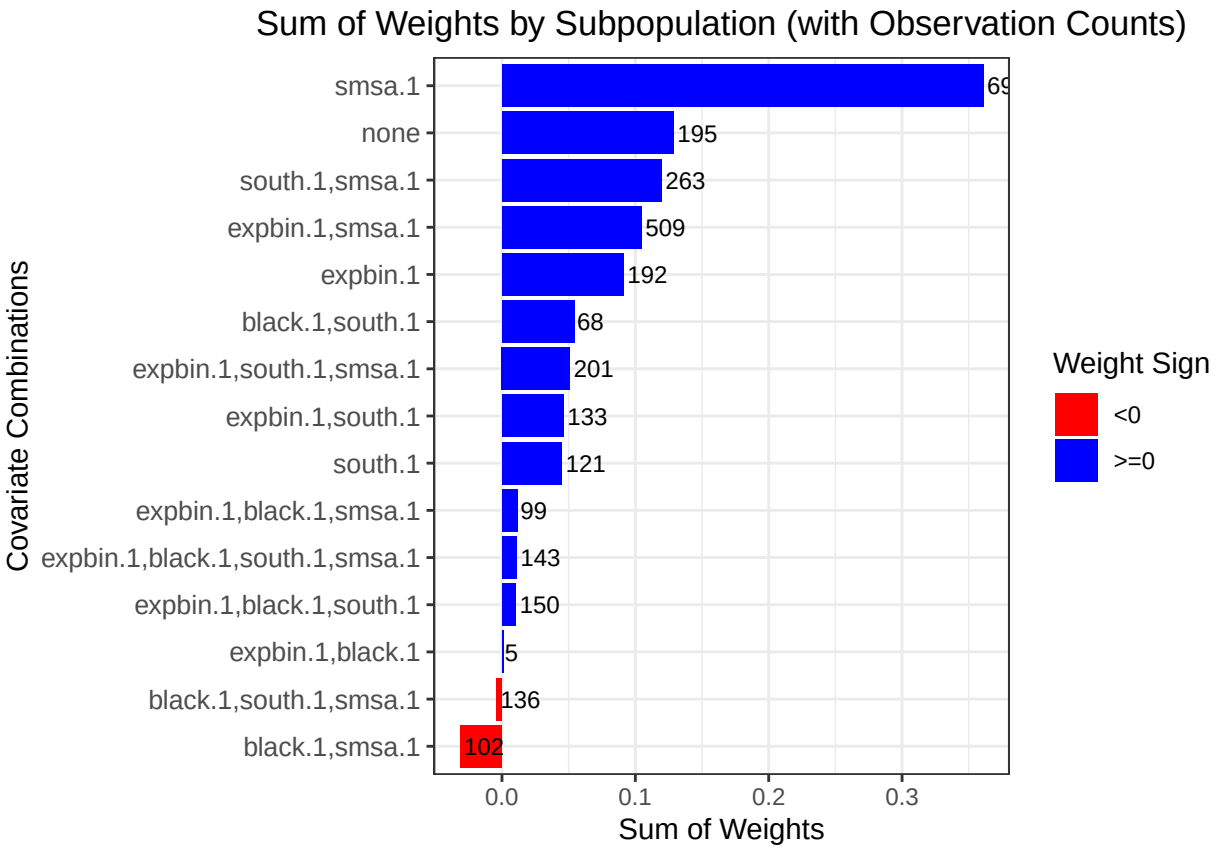
```
estimate_rich_weights <- function(df, z_vars = "Z", plot=TRUE) {
  set.seed(123)
  # 1) Identify covariates: all columns except Y, D, Z
  covariates <- setdiff(names(df), c("Y", "D", z_vars))

  # 2) Ensure D, Z are factors (binary for this example)
  df$D <- as.factor(df$D)
  df$Z <- as.factor(df$Z)

  # 3) Fit model for D ~ Z + X
  formula_D <- as.formula(
    paste("D ~ ", paste(c(z_vars, covariates), collapse = " + "))
  )
  model_D <- ranger(formula_D, data = df, probability = TRUE)

  # 4) Predict P(D=1) for Z=1 vs Z=0 to get p_diff = P[T(1)>T(0)|X]
  p1_hat <- predict(model_D, data = transform(df, Z=1), type = "response")$predictions[,2]
  p0_hat <- predict(model_D, data = transform(df, Z=0), type = "response")$predictions[,2]
  p_diff <- p1_hat - p0_hat  # estimated compliance propensity

  # 5) Fit model for Z ~ X to get pZ_hat, then var(Z|X) = pZ_hat*(1 - pZ_hat)
  formula_Z <- as.formula(
    paste("Z ~", paste(covariates, collapse = " + "))
  )
  model_Z <- ranger(formula_Z, data = df, probability = TRUE)

  pZ_hat <- predict(model_Z, data = df, type = "response")$predictions[,2]
```

```r
  vZ_hat <- pZ_hat * (1 - pZ_hat)

  # 6) Final weights = p_diff * var(Z|X), normalized to sum to 1
  w_raw <- p_diff * vZ_hat
  w_final <- w_raw / sum(w_raw)
  df$w_final <- w_final

  # 7) Create a 'group' label for every unique combination of X
  df$group <- apply(df[, covariates, drop = FALSE], 1, function(xx) {
    # Exclude variables matching the pattern "V<number>"
    vars_to_include <- names(xx)[xx == 1 & !grepl("^V[0-9]+.*$", names(xx))]
    paste(vars_to_include, collapse = ",")
  })
  df$group[df$group == ""] <- "none"  # Label groups with no variables set to 1

  # 8) Sum weights by group
  group_summary <- aggregate(w_final ~ group, data = df, sum)
  group_summary$obs_count <- aggregate(w_final ~ group, data = df, length)$w_final  # Count observation
  group_summary$sign <- ifelse(group_summary$w_final >= 0, ">=0", "<0")  # Positive/negative sign


  if (plot) {
    pt <- ggplot(group_summary, aes(x = reorder(group, w_final), y = w_final, fill = sign)) +
      geom_bar(stat = "identity") +
      geom_text(
        aes(label = obs_count),
        hjust = -0.1,  # Position labels slightly to the right of the bars
        size = 3
      ) +
      coord_flip() +  # Flip axes for better readability
      scale_fill_manual(values = c(">=0" = "blue", "<0" = "red")) +
      labs(
        title = "Sum of Weights by Subpopulation (with Observation Counts)",
        x = "Covariate Combinations",
        y = "Sum of Weights",
        fill = "Weight Sign"
      ) +
      theme_bw() +
      theme(
        axis.text.y = element_text(size = 10),
        plot.title = element_text(hjust = 0.5)
      )
  }
  print(pt)
  # Return a list with the augmented df and group-weight table
  return(list(
    df_with_weights = df,
    group_summary = group_summary
  ))
}
```

Estimate weights under saturated X and simple Z.



Estimate weights under saturated X and saturated Z (Saturate and Weight).



Monotonicity Extraction

Looking at the graphs, we suspect, under rich covariates,

- black, smsa, not south, not expbin
- expbin, black, south, not smsa
- expbin, south, not smsa, not black
- black, south, smsa, not expbin

are not correct in monotonicity.

Further, under Saturation and Weight,

- black, south, smsa, not expbin
- black, smsa, not south, not expbin

are with some confidence, defiers.

Hypothesis: removing them from the data and doing PLIV again will yield LATE.

First remove all four groups.

```
df_comp <- df %>%
  filter(!(black == 1 & smsa == 1 & expbin == 0 & south == 0),
         !(expbin == 1 & black == 1 & south == 1 & smsa == 0),
         !(expbin == 1 & south == 1 & black == 0 & smsa == 0),
```

```
         !(black == 1 & south == 1 & smsa == 1 & expbin == 0))
Y <- df_comp$Y
D <- df_comp$D
Z <- df_comp$Z
X_comp <- df_comp %>% select(-Y, -D, -Z) %>% as.matrix()
# redo PLIV
set.seed(123)
pliv_comp <- ddml_pliv(Y, D, Z, X_comp,
                       learners = learners_multiple,
                       ensemble_type = c('nnls', 'nnls1', 'ols','average'),
                       shortstack = TRUE,
                       sample_folds = 10,
                       silent = TRUE)
summary(pliv_comp)
```

```
## PLIV estimation results:
##
## , , nnls
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.000626    0.00903 -0.0693   0.9448
## D_r          0.651820    0.28291  2.3040   0.0212
##
## , , nnls1
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.000218    0.00903 -0.0241   0.9808
## D_r          0.651804    0.28291  2.3039   0.0212
##
## , , ols
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.000752    0.00902  0.0834   0.9335
## D_r         0.646825    0.27962  2.3133   0.0207
##
## , , average
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.000223    0.00899 -0.0249   0.9802
## D_r          0.639203    0.28068  2.2773   0.0228
```

Clearly, the hypothesis is supported by the data.

Then remove the two groups with some confidence.

```
df_comp <- df %>%
  filter(!(black == 1 & smsa == 1 & expbin == 0 & south == 0),
         !(black == 1 & south == 1 & smsa == 1 & expbin == 0))
Y <- df_comp$Y
D <- df_comp$D
Z <- df_comp$Z
X_comp <- df_comp %>% select(-Y, -D, -Z) %>% as.matrix()
# redo PLIV
```

```r
set.seed(123)
pliv_comp <- ddml_pliv(Y, D, Z, X_comp,
                       learners = learners_multiple,
                       ensemble_type = c('nnls', 'nnls1', 'ols','average'),
                       shortstack = TRUE,
                       sample_folds = 10,
                       silent = TRUE)
summary(pliv_comp)
```

```
## PLIV estimation results:
##
## , , nnls
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.000466    0.00908 -0.0513   0.9591
## D_r          0.800774    0.33954  2.3584   0.0184
##
## , , nnls1
##
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -1.63e-05    0.00908 -0.00179   0.9986
## D_r          8.01e-01    0.33956  2.35838   0.0184
##
## , , ols
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.000279    0.00906  0.0308    0.975
## D_r         0.795931    0.33632  2.3666    0.018
##
## , , average
##
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -4.57e-05    0.00904 -0.00506   0.9960
## D_r          7.89e-01    0.33797  2.33546   0.0195
```

The result is not entirely the same, so removing only the best-suspected defiers is not enough.