Comparative Analysis of High-Dimensional Propensity Scores & Machine Learning Extensions

Evaluating Methods for Residual Confounding Control using NHANES Data

Ehsan Karim

2026-02-05

Contents

Step 0: Load Data

We start by loading the cleaned NHANES data.

```r
# Ensure the path points to your specific file location
load("data/analytic3cycles.RData")
ls()
```

```
## [1] "dat.proxy.long" "data.complete"  "data.merged"
```

1. `dat.proxy.long`: This is the proxy data. It contains the longitudinal or claims data (e.g., ICD-10 diagnosis codes) for each individual. It's usually in a "long" format, meaning there are multiple rows per person (one row for each diagnosis code they have). The columns are typically id (patient identifier) and icd10 (the code itself). This is the source from which the hdPS algorithm generates candidate covariates.

2. `data.complete`: This is your analytic dataset. It contains the main study variables for each individual, such as the exposure (e.g., obesity), the outcome (e.g., diabetes), and the investigator-specified covariates (variables you explicitly chose to adjust for, e.g., age, sex, race, etc.). It's in a "wide" format (one row per person). na.omit(data.merged) is called to create data.complete.

Step 1-3: Generate All Recurrence Covariates

We first generate the full set of proxy variables (out2).

```r
# 1. Clean Proxy Data
dat.proxy.long <- subset(dat.proxy.long, icd10 != "E66")
dat.proxy.long <- subset(dat.proxy.long, icd10 != "O24")
dat.proxy.long <- subset(dat.proxy.long, icd10 != "E10")
dat.proxy.long <- subset(dat.proxy.long, icd10 != "E11")

# 2. Bridge Data (Create dfx)
analytic <- data.complete
idx <- analytic$id
outcome_vec <- as.numeric(analytic$diabetes == "Yes")
exposure_vec <- as.numeric(analytic$obese == "Yes")
domain <- "dx"

analytic.dfx <- as.data.frame(cbind(idx,
                                    exposure = exposure_vec,
                                    outcome = outcome_vec, domain))
proxy.var.long <- dat.proxy.long
proxy.var.long$idx <- proxy.var.long$id
proxy.var.long$id <- NULL
dfx <- merge(analytic.dfx, proxy.var.long, by = "idx")
basetable <- dfx %>% select(idx, exposure, outcome) %>% distinct()
patientIds <- basetable$idx

# 3. Generate Candidate Covariates
step1 <- get_candidate_covariates(df = dfx,
                                  domainVarname = "domain",
                                  eventCodeVarname = "icd10",
                                  patientIdVarname = "idx",
                                  patientIdVector = patientIds,
                                  n = 200,
                                  min_num_patients = 20)
```

```
## Joining with `by = join_by(eventCodeVarname, domainVarname)`
```

```r
out1 <- step1[["covars_data"]]

step2 <- get_recurrence_covariates(df = out1,
                                   patientIdVarname = "idx",
                                   eventCodeVarname = "icd10",
                                   patientIdVector = patientIds)
out2 <- step2[["recurrence_data"]]
```

Step 4: Prepare Datasets

We define investigator covariates and prepare two datasets: one with ALL proxies (for ML/SL/TMLE) and one with prioritized proxies (for standard hdPS).

```r
investigator.specified.covariates <- c(
  "age.cat", "sex", "education", "race",
  "marital", "income", "born", "year",
  "diabetes.family.history", "medical.access",
  "smoking", "diet.healthy", "physical.activity", "sleep",
  "uric.acid", "protein.total", "bilirubin.total", "phosphorus",
  "sodium", "potassium", "globulin", "calcium.total",
  "systolicBP", "diastolicBP", "high.cholesterol"
)

# --- Dataset 1: Full Proxy Set (for Pure ML, SL, TMLE) ---
proxy.dim <- out2
proxy.dim$id <- proxy.dim$idx
proxy.dim$idx <- NULL

fullcovproxy.data <- merge(data.complete[,c("id", "diabetes", "obese",
                                            investigator.specified.covariates)],
                           proxy.dim, by = "id")

fullcovproxy.data$outcome <- as.numeric(I(fullcovproxy.data$diabetes=='Yes'))
fullcovproxy.data$exposure <- as.numeric(I(fullcovproxy.data$obese=='Yes'))

# --- Dataset 2: Prioritized Proxy Set (for Standard hdPS & Hybrid) ---
# We prioritize covariates here (k=50)
out3 <- get_prioritised_covariates(df = out2,
                                   patientIdVarname = "idx",
                                   exposureVector = basetable$exposure,
                                   outcomeVector = basetable$outcome,
                                   patientIdVector = patientIds,
                                   k = 50)

hdps.dim <- out3[["autoselected_covariate_df"]]
hdps.dim$id <- hdps.dim$idx
hdps.dim$idx <- NULL

hdps.data.prioritized <- merge(data.complete[,c("id", "diabetes", "obese",
                                                investigator.specified.covariates)],
                               hdps.dim, by = "id")
hdps.data.prioritized$outcome <- as.numeric(I(hdps.data.prioritized$diabetes=='Yes'))
hdps.data.prioritized$exposure <- as.numeric(I(hdps.data.prioritized$obese=='Yes'))
```

Analysis 0: Crude (Unadjusted)

Simple logistic regression of outcome on exposure.

```r
out.formula <- as.formula("outcome ~ exposure")
fit_crude <- glm(out.formula,
                 data = fullcovproxy.data,
                 family = binomial(link = "logit"))

fit.summary <- summary(fit_crude)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit_crude)[2,2]) # Robust SE
conf.int <- confint(fit_crude, "exposure", level = 0.95, method = "hc1")
```

```
## Waiting for profiling to be done...
```

```
fit.summary_with_ci.crude <- c(fit.summary, conf.int)

cat("Crude Odds Ratio:", exp(fit.summary_with_ci.crude[1]), "\n")
```

```
## Crude Odds Ratio: 1.937889
```

Analysis 1: Conventional Propensity Score (PS)

Standard PS using only investigator-specified covariates (no proxies).

```
# 1. Formula
covform <- paste(investigator.specified.covariates, collapse = "+")
ps.formula <- as.formula(paste("exposure", "~", covform))

# 2. Weighting
W.out.conv <- weightit(ps.formula,
                       data = fullcovproxy.data,
                       estimand = "ATE",
                       method = "ps")

# 3. Estimation
out.formula <- as.formula("outcome ~ exposure")
fit <- glm(out.formula,
           data = fullcovproxy.data,
           weights = W.out.conv$weights,
           family = binomial(link = "logit"))

# 4. Results
fit.summary <- summary(fit)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit)[2,2]) # Robust SE
conf.int <- confint(fit, "exposure", level = 0.95, method = "hc1")
fit.summary_with_ci.conv <- c(fit.summary, conf.int)

cat("Conventional PS Odds Ratio:", exp(fit.summary[1]), "\n")
```

```
## Conventional PS Odds Ratio: 1.894328
```

Analysis 2: Standard hdPS

Using the top 50 proxies prioritized by the hdPS algorithm (no LASSO).

```r
# 1. Formula (Investigator + Top 50 Proxies)
# Note: out3 was generated in Step 4
proxy.list <- names(out3$autoselected_covariate_df[,-1])
proxyform <- paste(proxy.list, collapse = "+")
covform <- paste(investigator.specified.covariates, collapse = "+")
rhsformula <- paste(c(covform, proxyform), collapse = "+")
ps.formula <- as.formula(paste("exposure", "~", rhsformula))

# 2. Weighting
W.out.hdps <- weightit(ps.formula,
                       data = hdps.data.prioritized,
                       estimand = "ATE",
                       method = "ps")

# 3. Estimation
fit <- glm(out.formula,
           data = hdps.data.prioritized,
           weights = W.out.hdps$weights,
           family = binomial(link = "logit"))

# 4. Results
fit.summary <- summary(fit)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit)[2,2])
conf.int <- confint(fit, "exposure", level = 0.95, method = "hc1")
fit.summary_with_ci.hdps <- c(fit.summary, conf.int)

cat("Standard hdPS Odds Ratio:", exp(fit.summary[1]), "\n")
```

```
## Standard hdPS Odds Ratio: 1.42027
```

Analysis 3: Pure ML Approach (LASSO)

Uses all recurrence variables (from out2) to select proxies based on outcome association.

```r
# 1. Setup LASSO Data
proxy.list <- names(out2[-1])
covarsTfull <- c(investigator.specified.covariates, proxy.list)
Y.form <- as.formula(paste0(c("outcome~ exposure", covarsTfull), collapse = "+"))
covar.mat <- model.matrix(Y.form, data = fullcovproxy.data)[,-1]

# 2. Run LASSO
lasso.fit <- glmnet::cv.glmnet(y = fullcovproxy.data$outcome,
                               x = covar.mat,
                               type.measure='mse',
                               family="binomial",
                               alpha = 1,
                               nfolds = 5)

# 3. Extract Selected Variables
coef.fit <- coef(lasso.fit, s='lambda.min', exact=TRUE)
sel.variables <- row.names(coef.fit)[which(as.numeric(coef.fit)!=0)]
proxy.list.sel.ml <- proxy.list[proxy.list %in% sel.variables]
```

```
# 4. Fit PS Model
proxyform <- paste0(proxy.list.sel.ml, collapse = "+")
rhsformula <- paste0(c(covform, proxyform), collapse = "+")
ps.formula <- as.formula(paste0("exposure", "~", rhsformula))

W.out.ml <- weightit(ps.formula,
                     data = fullcovproxy.data,
                     estimand = "ATE",
                     method = "ps")

# 5. Estimation
fit <- glm(out.formula,
           data = fullcovproxy.data,
           weights = W.out.ml$weights,
           family = binomial(link = "logit"))

fit.summary <- summary(fit)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit)[2,2])
conf.int <- confint(fit, "exposure", level = 0.95, method = "hc1")
fit.summary_with_ci.lasso <- c(fit.summary, conf.int)

cat("Pure ML (LASSO) Odds Ratio:", exp(fit.summary[1]), "\n")
```

```
## Pure ML (LASSO) Odds Ratio: 1.455584
```

Analysis 4: Hybrid ML Approach

Starts with hdPS variables (Top 50 from out3), then refines with LASSO.

```
# 1. Run LASSO on Prioritized Data (hdps.data.prioritized)
proxy.list <- names(out3$autoselected_covariate_df[,-1])
covarsTfull <- c(investigator.specified.covariates, proxy.list)
Y.form <- as.formula(paste0(c("outcome~ exposure", covarsTfull), collapse = "+"))
covar.mat <- model.matrix(Y.form, data = hdps.data.prioritized)[,-1]

lasso.fit <- glmnet::cv.glmnet(y = hdps.data.prioritized$outcome,
                               x = covar.mat,
                               type.measure='mse',
                               family="binomial",
                               alpha = 1,
                               nfolds = 5)

coef.fit <- coef(lasso.fit, s='lambda.min', exact=TRUE)
sel.variables <- row.names(coef.fit)[which(as.numeric(coef.fit)!=0)]
proxy.list.sel.hybrid <- proxy.list[proxy.list %in% sel.variables]

# 2. Fit PS Model
proxyform <- paste0(proxy.list.sel.hybrid, collapse = "+")
rhsformula <- paste0(c(covform, proxyform), collapse = "+")
ps.formula <- as.formula(paste0("exposure", "~", rhsformula))

W.out.hybrid <- weightit(ps.formula,
                          data = hdps.data.prioritized,
                          estimand = "ATE",
                          method = "ps")

# 3. Estimation
fit <- glm(out.formula,
           data = hdps.data.prioritized,
           weights = W.out.hybrid$weights,
           family = binomial(link = "logit"))

fit.summary <- summary(fit)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit)[2,2])
conf.int <- confint(fit, "exposure", level = 0.95, method = "hc1")
fit.summary_with_ci.hybrid <- c(fit.summary, conf.int)

cat("Hybrid ML Odds Ratio:", exp(fit.summary[1]), "\n")
```

```
## Hybrid ML Odds Ratio: 1.468922
```

Analysis 5: Ensemble / Super Learner

Uses **all** proxies with multiple learners.

```
# 1. Define Formula
proxy.list <- names(out2[-1])
proxyform <- paste0(proxy.list, collapse = "+")
rhsformula <- paste0(c(covform, proxyform), collapse = "+")
ps.formula <- as.formula(paste0("exposure", "~", rhsformula))

# 2. Fit Super Learner
# Warning: Computationally intensive
W.out.sl <- weightit(ps.formula,
                     data = fullcovproxy.data,
                     estimand = "ATE",
                     method = "super",
                     #SL.library = c("SL.glm", "SL.mean")) # shorter compute time
                     SL.library = c("SL.glm", "SL.glmnet", "SL.earth"))

# 3. Estimation
fit <- glm(out.formula,
           data = fullcovproxy.data,
           weights = W.out.sl$weights,
           family = binomial(link = "logit"))

fit.summary <- summary(fit)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit)[2,2])
conf.int <- confint(fit, "exposure", level = 0.95, method = "hc1")
fit.summary_with_ci.sl <- c(fit.summary, conf.int)

cat("Super Learner Odds Ratio:", exp(fit.summary[1]), "\n")
```

```
## Super Learner Odds Ratio: 1.555131
```

Analysis 6: TMLE

Uses the Super Learner results and full proxy set.

```r
# 1. Setup Data
# SL.library <- c("SL.glm", "SL.mean") # shorter compute time
SL.library <- c("SL.glm", "SL.glmnet", "SL.earth")
proxy.list <- names(out2[-1])
ObsData.noYA <- fullcovproxy.data[, c(investigator.specified.covariates, proxy.list)]

# 2. Run TMLE (With Super Learner)
tmle.fit <- tmle::tmle(Y = fullcovproxy.data$outcome,
                       A = fullcovproxy.data$exposure,
                       W = ObsData.noYA,
                       family = "binomial",
                       V.Q = 3,
                       V.g = 3,
                       Q.SL.library = SL.library,
                       g1W = W.out.sl$ps)

estOR.tmle <- tmle.fit$estimates$OR

# 3. Run TMLE (GLM Only - "TMLE (only GLM in SL)")
SL.library.glm <- c("SL.glm")
tmle.fit0 <- tmle::tmle(Y = fullcovproxy.data$outcome,
                       A = fullcovproxy.data$exposure,
                       W = ObsData.noYA,
                       family = "binomial",
                       V.Q = 3,
                       V.g = 3,
                       Q.SL.library = SL.library.glm,
                       g1W = W.out.sl$ps)

estOR.tmle0 <- tmle.fit0$estimates$OR

cat("TMLE (SL) Odds Ratio:", estOR.tmle$psi, "\n")
```

```
## TMLE (SL) Odds Ratio: 1.485521
```

```r
cat("TMLE (GLM) Odds Ratio:", estOR.tmle0$psi, "\n")
```

```
## TMLE (GLM) Odds Ratio: 1.473184
```

The above code calculates Super Learner propensity scores externally via `WeightIt` and passes them to `tmle` using the `g1W` argument. However, `tmle` can estimate the propensity score model (g) internally by specifying `g.SL.library`. To do that, you can modify your TMLE code chunk to remove the `g1W` argument and instead set `g.SL.library = c("SL.glm", "SL.glmnet", "SL.earth")`, allowing the `tmle` function to perform its own internal estimation and cross-validation.

Analysis 7: Kitchen Sink

Directly adjust for all proxies in the outcome model (outcome ~ exposure + covariates + all proxies). Warning: This often fails to converge or overfits if $n < p$.

```r
# Define formula with ALL covariates and ALL proxies
proxy.list <- names(out2[-1])
covarsTfull <- c(investigator.specified.covariates, proxy.list)
kitchen.formula <- as.formula(paste0(c("outcome ~ exposure", covarsTfull), collapse = "+"))

# Fit GLM
fit_ks <- glm(kitchen.formula,
              data = fullcovproxy.data,
              family = binomial(link = "logit"))

fit.summary <- summary(fit_ks)$coef["exposure", c("Estimate", "Std. Error", "Pr(>|z|)")]
fit.summary[2] <- sqrt(sandwich::sandwich(fit_ks)[2,2])
conf.int <- confint(fit_ks, "exposure", level = 0.95, method = "hc1")
```

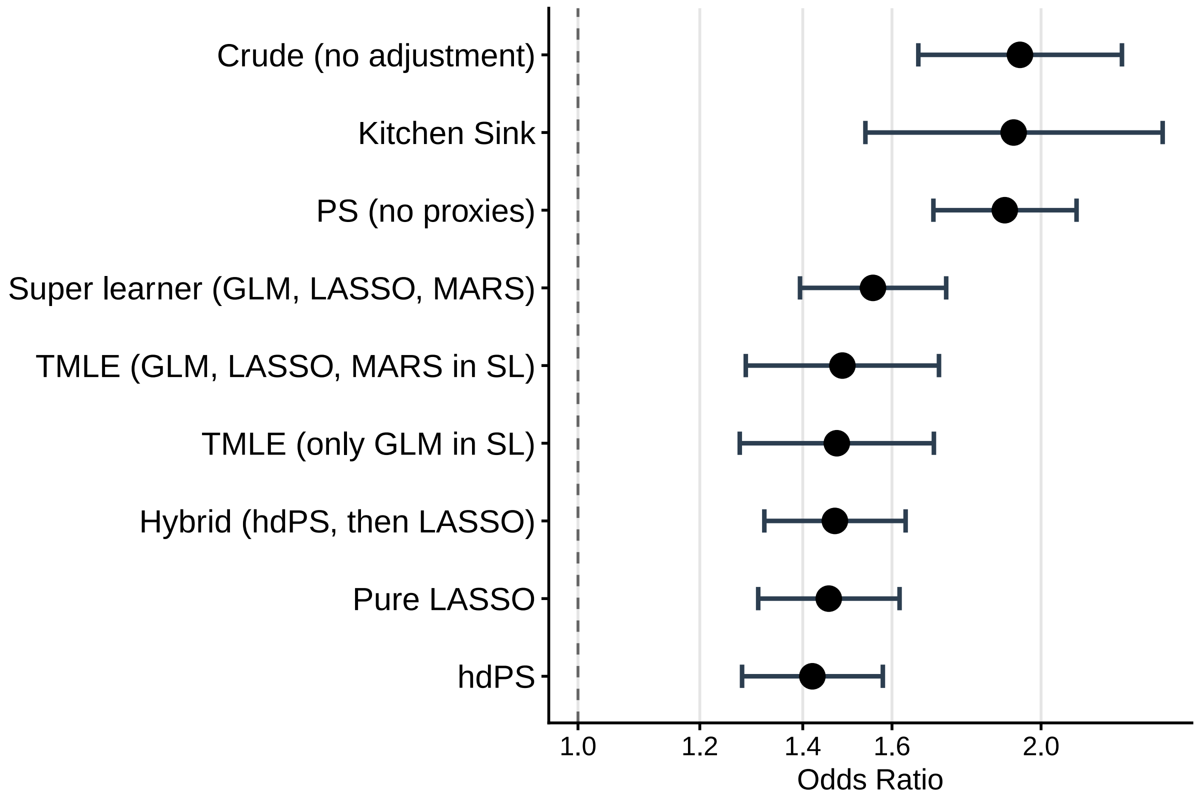
```
## Waiting for profiling to be done...
```

```r
fit.summary_with_ci.ks <- c(fit.summary, conf.int)

cat("Kitchen Sink Odds Ratio:", exp(fit.summary_with_ci.ks[1]), "\n")
```

```
## Kitchen Sink Odds Ratio: 1.919621
```

Summarize

```r
# Construct Dataframe
results_df <- rbind(
  # Crude
  data.frame(model = "Crude (no adjustment)",
             or = exp(fit.summary_with_ci.crude[1]),
             lower = exp(fit.summary_with_ci.crude[4]),
             upper = exp(fit.summary_with_ci.crude[5])),

  # Conventional PS
  data.frame(model = "PS (no proxies)",
             or = exp(fit.summary_with_ci.conv[1]),
             lower = exp(fit.summary_with_ci.conv[4]),
             upper = exp(fit.summary_with_ci.conv[5])),

  # hdPS
  data.frame(model = "hdPS",
             or = exp(fit.summary_with_ci.hdps[1]),
             lower = exp(fit.summary_with_ci.hdps[4]),
             upper = exp(fit.summary_with_ci.hdps[5])),

  # Pure LASSO
  data.frame(model = "Pure LASSO",
             or = exp(fit.summary_with_ci.lasso[1]),
             lower = exp(fit.summary_with_ci.lasso[4]),
             upper = exp(fit.summary_with_ci.lasso[5])),
  # Hybrid
  data.frame(model = "Hybrid (hdPS, then LASSO)",
             or = exp(fit.summary_with_ci.hybrid[1]),
             lower = exp(fit.summary_with_ci.hybrid[4]),
             upper = exp(fit.summary_with_ci.hybrid[5])),

  # Kitchen Sink
  data.frame(model = "Kitchen Sink",
             or = exp(fit.summary_with_ci.ks[1]),
             lower = exp(fit.summary_with_ci.ks[4]),
             upper = exp(fit.summary_with_ci.ks[5])),

  # Super Learner
  data.frame(model = "Super learner (GLM, LASSO, MARS)",
             or = exp(fit.summary_with_ci.sl[1]),
             lower = exp(fit.summary_with_ci.sl[4]),
             upper = exp(fit.summary_with_ci.sl[5])),

  # TMLE (SL)
  data.frame(model = "TMLE (GLM, LASSO, MARS in SL)",
             or = estOR.tmle$psi,
             lower = estOR.tmle$CI[1],
             upper = estOR.tmle$CI[2]),

  # TMLE (GLM)
  data.frame(model = "TMLE (only GLM in SL)",
             or = estOR.tmle0$psi,
             lower = estOR.tmle0$CI[1],
             upper = estOR.tmle0$CI[2])
)

# Plot
ggplot(results_df, aes(x = or, y = reorder(model, or))) +
  geom_vline(xintercept = 1, linetype = "dashed", color = "gray40") +
  geom_errorbar(aes(xmin = lower, xmax = upper),
                width = 0.3, linewidth = 0.8, color = "#2C3E50") +
  geom_point(size = 4, shape = 19, color = "black") +
  scale_x_log10(breaks = c(1.0, 1.2, 1.4, 1.6, 2.0)) +
  labs(
    title = "",
    x = "Odds Ratio",
    y = ""
  ) +
  theme_classic() +
  theme(
    axis.text.y = element_text(size = 12, color = "black"),
    axis.text.x = element_text(size = 10, color = "black"),
    panel.grid.major.x = element_line(color = "gray90")
  )
```



Interactive Plot (only for HTML outputs)

```r
# Interactive Plot using Plotly
p <- ggplot(results_df, aes(x = or, y = reorder(model, or),
                            text = paste("Model:", model,
                                         "<br>OR:", round(or, 2),
                                         "<br>CI:", round(lower, 2),
                                         "-", round(upper, 2)))) +
  geom_vline(xintercept = 1, linetype = "dashed", color = "gray40") +
  geom_errorbar(aes(xmin = lower, xmax = upper),
                width = 0.3, linewidth = 0.8, color = "#2C3E50") +
  geom_point(size = 3, shape = 19, color = "#E74C3C") +
  scale_x_log10(breaks = c(1.0, 1.2, 1.4, 1.6, 2.0)) +
  labs(title = "", x = "Odds Ratio (log scale)", y = "") +
  theme_minimal() +
  theme(panel.grid.major.x = element_line(color = "gray90"))

ggplotly(p, tooltip = "text")
```

Interactive Table (only for HTML outputs)

```r
display_df <- results_df %>%
  mutate(
    OR = round(or, 3),
    `95% CI` = paste0("(", round(lower, 3), ", ", round(upper, 3), ")")
  ) %>%
  select(Model = model, OR, `95% CI`)

datatable(display_df, rownames = FALSE,
          options = list(pageLength = 15,
                         dom = 't',
                         columnDefs = list(list(className = 'dt-center',
                                                targets = 0:1))),
          caption = "Comparison of Effect Estimates")
```