Causal Inference Final Project

CBPS and Entropy Balancing - A simulation Study

Chansoo Song

November 2018

Introduction:

Matching:

In an observational study setting where the confounding covariates are known and measured, we may use mathching methods to ensure that there is sufficient overlap and balance on these covariates. Then, we can estimate the treatement effect using a simple difference in means or regression methods.

Overlap is important because we want to make sure that for each treated or control subject in the study, there exists an empirical counterfactual (this criteria varies depending on the estimand of interest, i.e. to estimate the ATT it is sufficient to have empirical counterfactuals for just the treated subjects in the study). Imbalance on the covariates forces us to rely more on the correct functional form of the model.

There are many different matching methods, but the driving principle is to identify observations that are "most similar", based on some distance metric. Methods include K-nearest-neighbor, caliper-matching, kernel-matching, Mahalanobis matching, Genetic Matching, Optimal Matching.

Propensity Scores:

A propensity score is a one-number summary of the covariates. Rosenbaum and Rubin (1983) define the propensity score for participant i as the conditional probability of treatment assignment ($Z_i = 1$) given a vector of observed covariates: $e(X_i) = Pr(Z_i = 1|X)$. The most common traditional approaches to estimating the propensity score are logistic regression and probit regression.

If strong ignorability holds after conditioning on the propensity score, that is:

$$y_0, y_1 \perp Z_i | e(X_i),\ 0 < e(X_i) < 1$$

Then we may obtain an unbiased estimate of the treatment effect by either matching or weighting using just the propensity score instead of the vector of covariates.

In practice, a key challenge in the application of propensity scores for matching is that the propensity score is unknown and must be estimated. To make matters worse, slight misspecification of the propensity score model can lead to substantial biases in treatmenet effects. As such, researchers may iteratively re-estimate the propensity score model and subsequently check the resulting covariate balance. Imai et al (2008) calls this the 'propensity score tautology': the estimated propensity score is appropriate if it balances covariates.

In this simulation study, I analyze two approaches that seek to bypass this 'propensity score tautology': Covariate Balancing Propensity Score and Entropy Balancing. Each method obviates the need for iteratively re-estimating the propensity score model and checking balance on the covariate moments. That is, a single model is used to estimate both the treatment assignment mechanism and the covariate balancing weights.

Covariate Balancing Propensity Score (CBPS):

The CBPS exploits the dual characteristics of the propensity score as a covariate balancing score and the conditional probability of treatment assignment (Imai and Ratkovic (2012)).

First, consider a commonly used model for estimating propensity scores: logistic regression:

$$e_B(X_i) = \frac{exp(X_i^T \beta)}{1 + exp(X_i^T \beta)}$$

We typically estimate the unknown parameters by maximum likelihood, i.e. maximizing the log-likelihood function:

$$\hat{\beta}_{MLE} = \sum_{i=1}^{N} Z_i \ log\{e_B(X_i) \ + (1 - Z_i) \ log\{1 - e_B(X_i)\}$$

Differentiating with respect to β, we get:

$$\frac{1}{N} \sum_{i=1}^{N} \frac{Z_i \ e'_B(X_i)}{e_B(X_i)} - \frac{(1 - Z_i) \ e'_B(X_i)}{1 - e_B(X_i)}$$

Imai and Ratkovic emphasize that the equation above can also be interpreted as the condition that balances a particular function of covariates, i.e. the first derivative of $e_B(X_i)$. Then, they operationalize the covariate balancing property by using inverse propensity score weighting:

$$E\left\{ \frac{Z_i \ e'_B(\tilde{X}_i)}{e_B(X_i)} - \frac{(1 - Z_i) \ e'_B(\tilde{X}_i)}{1 - e_B(X_i)} \right\} = 0$$

where $\tilde{X}_i = f(X_i)$, a function of X_i specified by the researcher. Setting $tilde X_i = e'_B(X_i)$ gives more weights to covariates that are predictive of treatmeent assignment according to the logistic regression propensity score model. But so long as the expectaion exists, the equation must hold for any choice of f(.). For example, setting $\tilde{X}_i = X_i$ ensures the first moment of each covariate is balanced. Setting $\tilde{X}_i = (X_i^T X_i^{2T})^T$ ensures the first and second moment of each covariate is balanced.

Entropy Balancing:

Entropy balancing similarly involves a reweighting scheme that directly incorporates covariate balance into the weight function that is applied to the sample units (Heinmueller 2015). To do this, entropy balancing searches for a set of weights that satisfies the balance constraints, while trying to keep the distribution of weights as uniform as possible (i.e. minimizing the divergence of distribution of weights from a uniform distribution). Thus, entropy balancing (1) allows us to obtain a high degree of covariate balance (using balance constraints that can involve the first, second, and possibly higher moments of the covariate distributions as well as interactions). And (2) allows for a more flexible reweighting scheme that seeks to retain as much information as possible. For example, nearest neighbor matching may discard subjects that are not matched (i.e. set weight equal to 0).

Consider the reweighting scheme to estimate the Average Treatement Effect on the Treated (ATT). We would want to estimate the counterfactual mean by:

$$E[\widehat{Y(0)|Z} = 1] = \frac{\sum_{i|Z=0} Y_i w_i}{\sum_{i|Z=0} w_i}$$

where w_i is a weight for each control unit.

The weights are chosen by the following reweighting scheme:

$$H(w) = \sum_{i|D=0} h(w_i)$$

where $h(.)$ is a distance metric and $c_{ri}(X_i) = m_r$ describes a set of R balance constraints imposed on the covariate moments of the reweighted control group.

Minimize $H(w)$ subject to the balance and normalizing constraints:

$$\sum_{i|D=0} w_i c_{ri}(X_i) = m_r$$

with $r \in 1, ..., r$

$$\sum_{i|Z=0} w_i = 1$$

and $w_i \geq 0$ for all i such that $T = 0$.

Simulation and DGP Method:

In this section, I examine whether the CBPS or Entropy Balancing methods improve upon the performance of a baseline approach to both (1) achieving balanced covariates and (2) estimating the treatment effect. The baseline approach estimates include (1) propensity scores using logistic regression and matches using 1-1 matching with replacement and (2) mahalanobis matching.

I consider four data generating processes: (1) The pre-treatment covariates $X_i = (X_{i1}, X_{i2}, X_{i3}, X_{i4})$ are four independent and identically distributed random variables following a standard normal distribution. The true propensity score model is a logistic regression whose linear predictor is a linear transform of the pre-treatment covariates.
(2) The pre-treatment covariates are the same as Simulation #1; however, the true propensity score model is a logistic regression whose linear predictor are non-linear transforms of the pre-treatment covariates: $X_i^* = (0.25 * exp(\frac{1}{X_{i1}}), X_{i2}, X_{i3}, X_{i3} * X_{i4}, X_{i4})$ (3) The pre-treatment covariates $X_i = (X_{i1}, X_{i2}, X_{i3}, X_{i4}, X_{i5}, X_{i6})$ consist of three independent and identically distributed random variables following a normal distribution with means $(-2, -1, -1)$ and standard deviations $(1, 1, 1)$, a random variable following a poisson distribution with $\lambda = 5$, the negative values of a random variable following a binomial distribution with $n = 5$ and $p = 0.75$, and a random variable following a chi-squared distribution with $df = 1$.
(4) The pre-treatment covariates are the same as Simulation #3; however, the true propensity score model is a logistic regression whose linear predictor are non-linear transforms of the pre-treatment covariates: $X_i^* = (0.25 * exp(\frac{1}{X_{i1}}), X_{i2}, X_{i3}, X_{i3} * X_{i4}, X_{i4})$

In all four simulations, the true outcome model is a linear regression with the pre-treatment covariates as predictors.

Simulations:

Set Up and Initialize:

```
# Samle size
n = 500
n_sims = 100

# Models
model_names = c("Baseline Logistic", "Baseline Mahalanobis",
    "CBPS - Over", "CBPS - Just", "Entropy Balancing")

# Inverse Logit Function
inv.logit = function(x) {
    y = 1/(1 + exp(-x))
```

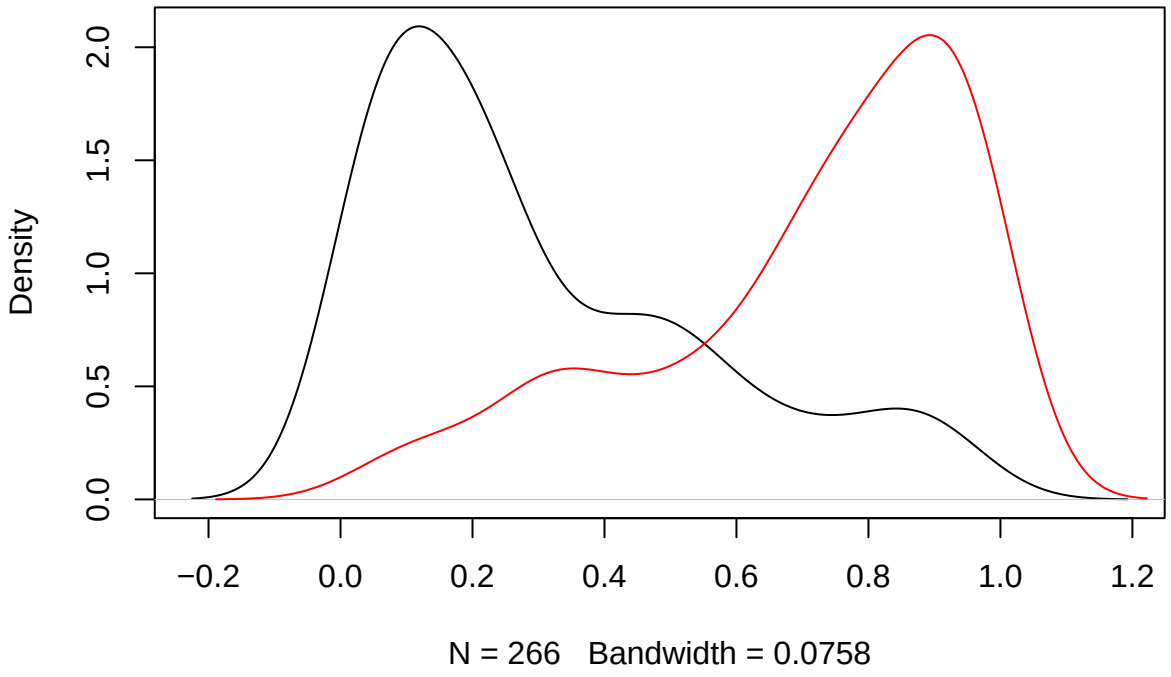
```
    return(y)
}

mod.inv.logit = function(x) {
    y = 0.9/(1 + exp(-x)) + 0.05
    return(y)
}

df = list()
SATE = list()
confounders = list()
confounders[[1]] = paste("z", seq(1, 4), sep = "")
confounders[[2]] = paste("z", seq(1, 4), sep = "")
confounders[[3]] = paste("z", seq(1, 6), sep = "")
confounders[[4]] = paste("z", seq(1, 6), sep = "")
```

DGP #1:

```
# Randomly generate covariates
z1 = rnorm(n, 0, 1)
z2 = rnorm(n, 0, 1)
z3 = rnorm(n, 0, 1)
z4 = rnorm(n, 0, 1)
Z_ = cbind(z1, z2, z3, z4)

# Estimate 'True Propensity Scores'
e_Z = inv.logit(z1 + z2 + z3 + z4)

# Generate treatment vector
treat = rbinom(n, 1, prob = e_Z)
table(treat)/length(treat)
```

```
## treat
##     0     1
## 0.532 0.468
```

```
# True Propensity Score Density
plot(density(e_Z[treat == 0]))
lines(density(e_Z[treat == 1]), col = 2)
```




```
# Treatment Effect
eff = 4

# Generate Potential Outcomes
y_0 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + rnorm(n, 0,
    1)
y_1 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + eff + rnorm(n,
    0, 1)
y = y_0 * (1 - treat) + y_1 * treat

# Generate Researcher Dataset
df[[1]] = as.data.frame(cbind(y, y_1, y_0, treat, Z_[, confounders[[1]]]))
df[[1]] = df[[1]] %>% arrange(-treat)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
# SATE
SATE[[1]] = mean(y_1) - mean(y_0)
SATE[[1]]
```

```
## [1] 4.012787
```

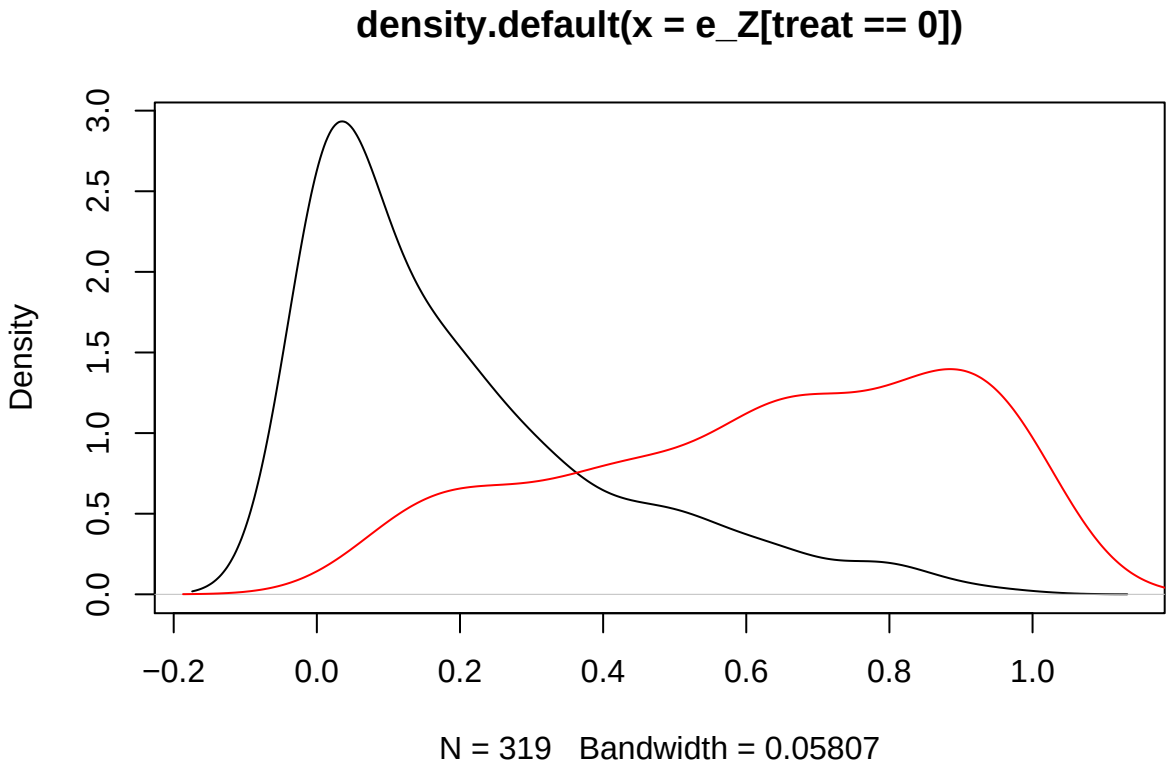
DGP #2:

```
# Randomly generate covariates
z1 = rnorm(n, 0, 1)
z2 = rnorm(n, 0, 1)
z3 = rnorm(n, 0, 1)
z4 = rnorm(n, 0, 1)
Z_ = cbind(z1, z2, z3, z4)
```

```r
# Estimate 'True Propensity Scores'
x1 = -0.25 * exp(1/z1)
x2 = z2
x3 = z3
x4 = z3 * z4
x5 = z4
e_Z = inv.logit(x1 + x2 + x3 + x4 + x5)

# Generate treatment vector
treat = rbinom(n, 1, prob = e_Z)
table(treat)/length(treat)
```

```
## treat
##     0     1
## 0.638 0.362
```

```r
# True Propensity Score Density
plot(density(e_Z[treat == 0]))
lines(density(e_Z[treat == 1]), col = 2)
```




```r
# Treatment Effect
eff = 4

# Generate Potential Outcomes
y_0 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + rnorm(n, 0,
    1)
y_1 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + eff + rnorm(n,
    0, 1)
y = y_0 * (1 - treat) + y_1 * treat
```

```r
# Generate Researcher Dataset
df[[2]] = as.data.frame(cbind(y, y_1, y_0, treat, Z_[, confounders[[2]]]))
df[[2]] = df[[2]] %>% arrange(-treat)


# SATE
SATE[[2]] = mean(y_1) - mean(y_0)
SATE[[2]]
```

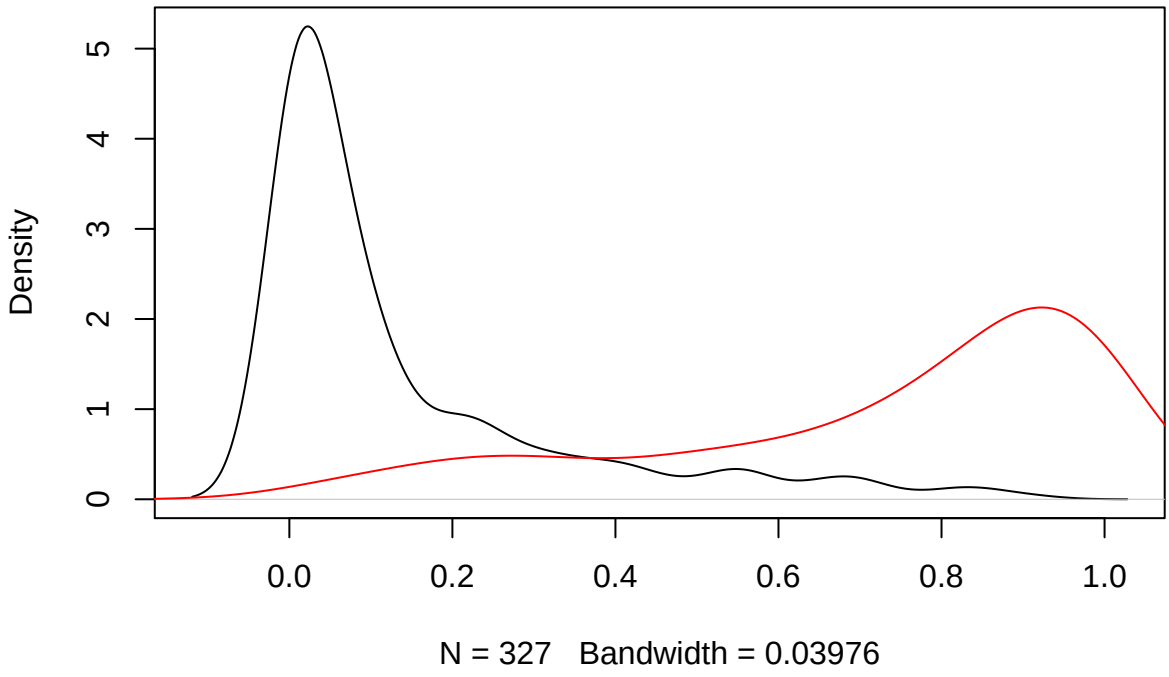
```
## [1] 3.91818
```

DGP #3:

```r
# Randomly generate covariates:
z1 = rnorm(n, -2, 1)
z2 = rnorm(n, -1, 1)
z3 = rnorm(n, -1, 1)
z4 = rpois(n, 5)
z5 = -rbinom(n, 5, 0.75)
z6 = rchisq(n, df = 1)
Z_ = cbind(z1, z2, z3, z4, z5, z6)

# Estimate 'True Propensity Scores'
e_Z = inv.logit(z1 + z2 + z3 + z4 + z5 + z6)

# Generate treatment vector
treat = rbinom(n, 1, prob = e_Z)
table(treat)/length(treat)
```

```
## treat
##     0     1
## 0.654 0.346
```

```r
# True Propensity Score Density
plot(density(e_Z[treat == 0]))
lines(density(e_Z[treat == 1]), col = 2)
```




```
# Treatment Effect
eff = 4

# Generate Potential Outcomes
y_0 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + 0.1 * z5 +
    z6 + rnorm(n, 0, 1)
y_1 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + 0.1 * z5 +
    z6 + eff + rnorm(n, 0, 1)
y = y_0 * (1 - treat) + y_1 * treat

# Generate Researcher Dataset
df[[3]] = as.data.frame(cbind(y, y_1, y_0, treat, Z_[, confounders[[3]]]))
df[[3]] = df[[3]] %>% arrange(-treat)


# SATE
SATE[[3]] = mean(y_1) - mean(y_0)
SATE[[3]]
```

```
## [1] 4.161038
```

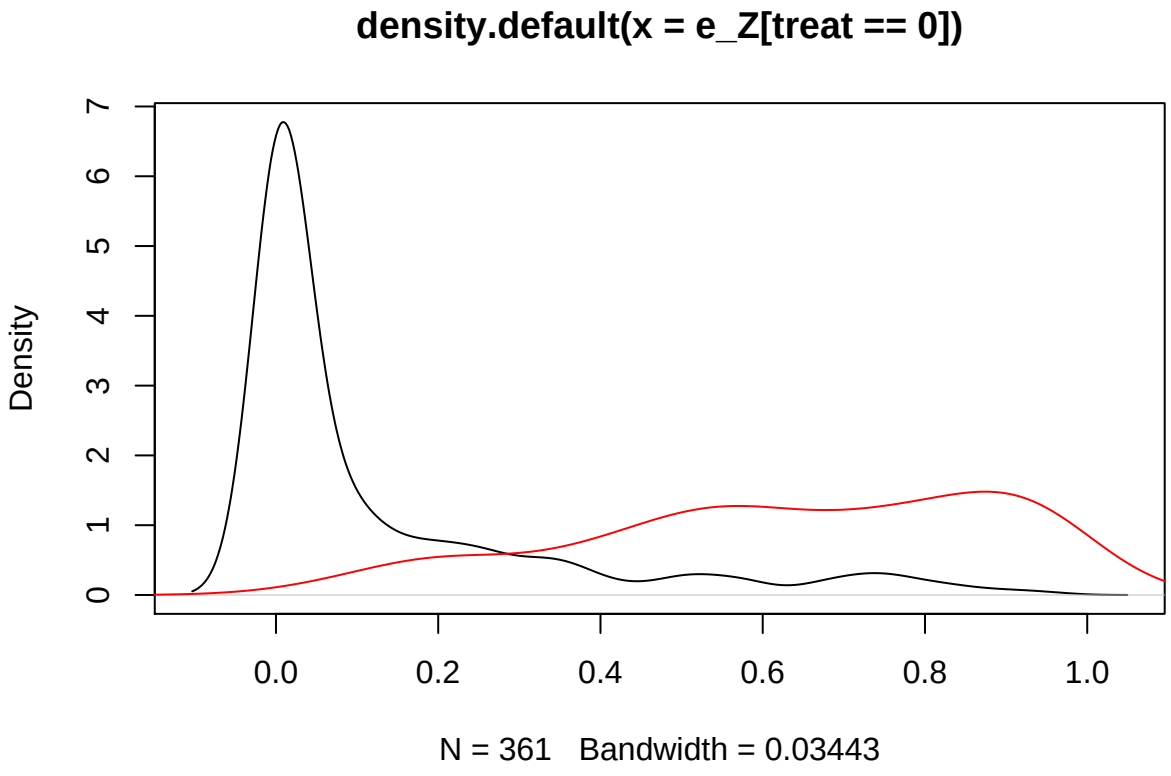
DGP #4:

```
# Randomly generate covariates:
z1 = rnorm(n, 0, 1)
z2 = rnorm(n, 0, 1)
z3 = rbinom(n, 5, 0.75)
z4 = rpois(n, 5)
z5 = rnorm(n, -2, 2)
z6 = rchisq(n, df = 1)
```

```
Z_ = cbind(z1, z2, z3, z4, z5, z6)

# Estimate 'True Propensity Scores'
x1 = exp(z1/2)
x2 = z2/(1 + exp(z1))
x3 = -2 * sqrt(z3)
x4 = sqrt(z3 * z4)
x5 = -0.25 * (z1 + z5)^2
x6 = -2 * z6
e_Z = inv.logit(x1 + x2 + x3 + x4 + x5 + x6)

# Generate treatment vector
treat = rbinom(n, 1, prob = e_Z)
table(treat)/length(treat)
```

```
## treat
##     0     1
## 0.722 0.278
```

```
# True Propensity Score Density
plot(density(e_Z[treat == 0]))
lines(density(e_Z[treat == 1]), col = 2)
```




```
# Treatment Effect
eff = 4

# Generate Potential Outcomes
y_0 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + 0.1 * z5 +
    z6 + rnorm(n, 0, 1)
y_1 = 0.4 * z1 + 0.5 * z2 + 0.3 * z3 + 0.9 * z4 + 0.1 * z5 +
```

```
    z6 + eff + rnorm(n, 0, 1)
y = y_0 * (1 - treat) + y_1 * treat

# Generate Researcher Dataset
df[[4]] = as.data.frame(cbind(y, y_1, y_0, treat, Z_[, confounders[[4]]]))
df[[4]] = df[[4]] %>% arrange(-treat)

# SATE
SATE[[4]] = mean(y_1) - mean(y_0)
SATE[[4]]
```

```
## [1] 4.029213
```

Matching Methods

1. Baseline: Propensity Score using Logistic Regression:
    a. Logistic Regression to estimate propensity scores
    b. 1-1 nearest neighbor matching using propensity score
    c. linear regression using propensity score and covariates
2. Baseline: Propensity Score using Logistic Regression:
3. Covariate Balancing Propensity Score
    a. Over-identified: combines the propensity score and covariate balancing conditions
    b. Just-identified: only contains covariate blancing conditions
4. Entropy Balancing

Estimate Weights

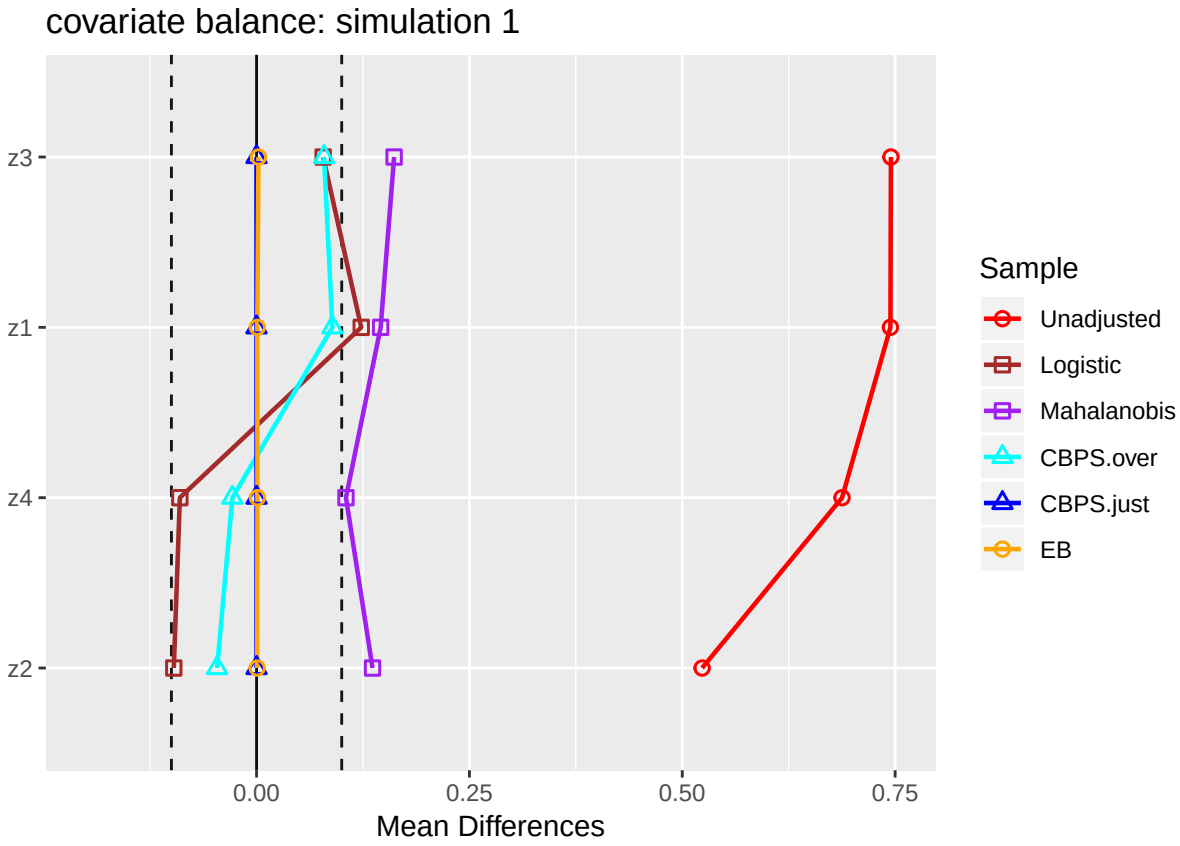
```
mod_match = mod_mahalo = eb.out = cbps.over = cbps.just = list()

for (k in 1:4) {

    formula = as.formula(paste("treat ~ ", paste(confounders[[k]],
        collapse = " + "), sep = ""))

    #### (1)

    # Logistic Regression, 1-1 Matching
    mod_match[[k]] = weightit(formula, data = df[[k]], method = "ps",
        estimand = "ATT")
    df[[k]]$wt = mod_match[[k]]$weights

    #### (2)

    # Mahalanobis Distance Matching
    mod_mahalo[[k]] = matchit(formula, data = df[[k]], method = "nearest",
        distance = "mahalanobis", replace = TRUE)
    df[[k]]$wt_mh = mod_mahalo[[k]]$weights

    #### (3)

    # CBPS Weights (OVER)
    cbps.over[[k]] = weightit(formula, data = df[[k]], method = "cbps",
        estimand = "ATT", over = TRUE)
```

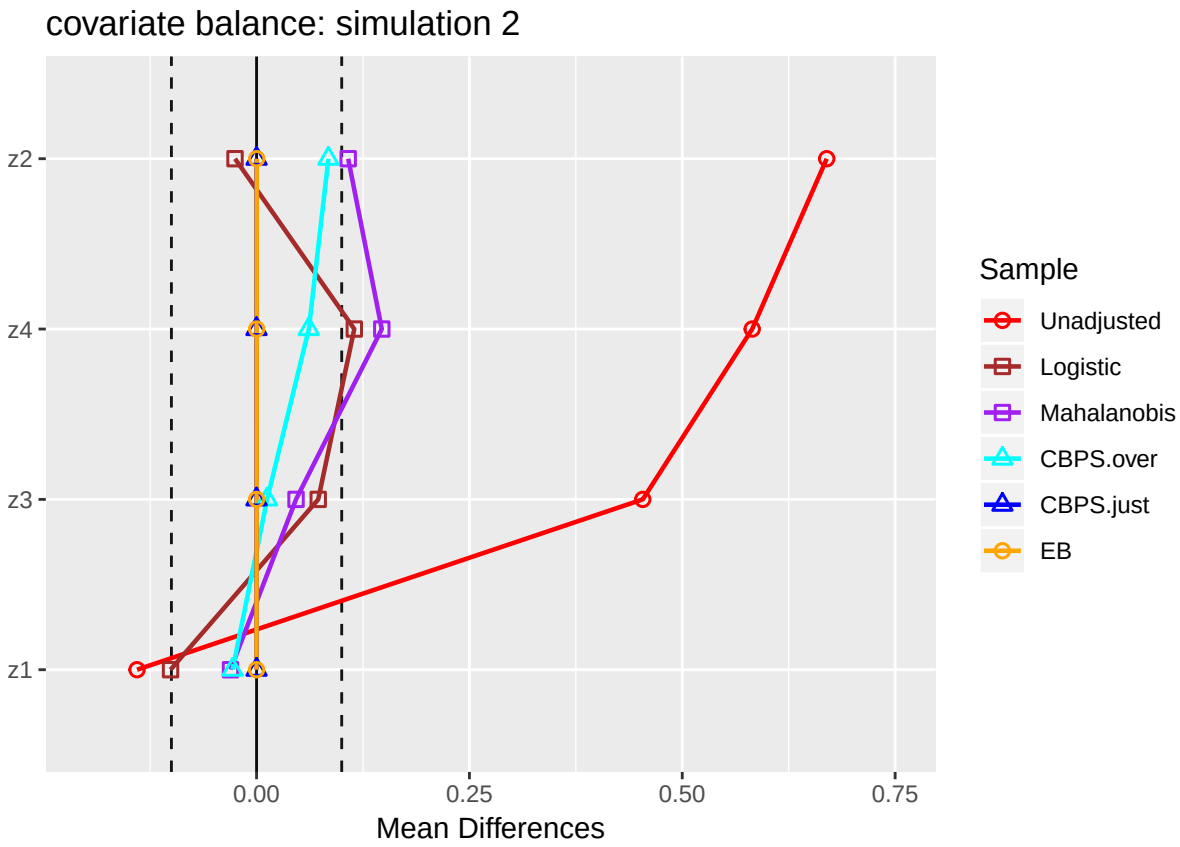
```
    df[[k]]$wt.cbps = cbps.over[[k]]$weights

    #### (4)

    # CBPS Weights (JUST)
    cbps.just[[k]] = weightit(formula, data = df[[k]], method = "cbps",
        estimand = "ATT", over = FALSE)
    df[[k]]$wt.cbps = cbps.just[[k]]$weights

    #### (5)

    # Entropy Balancing Weights
    eb.out[[k]] = ebalance(Treatment = df[[k]]$treat, X = df[[k]][,
        confounders[[k]]])
    df[[k]]$wt.eb = 1
    df[[k]][df[[k]]$treat == 0, "wt.eb"] = eb.out[[k]]$w

}
```

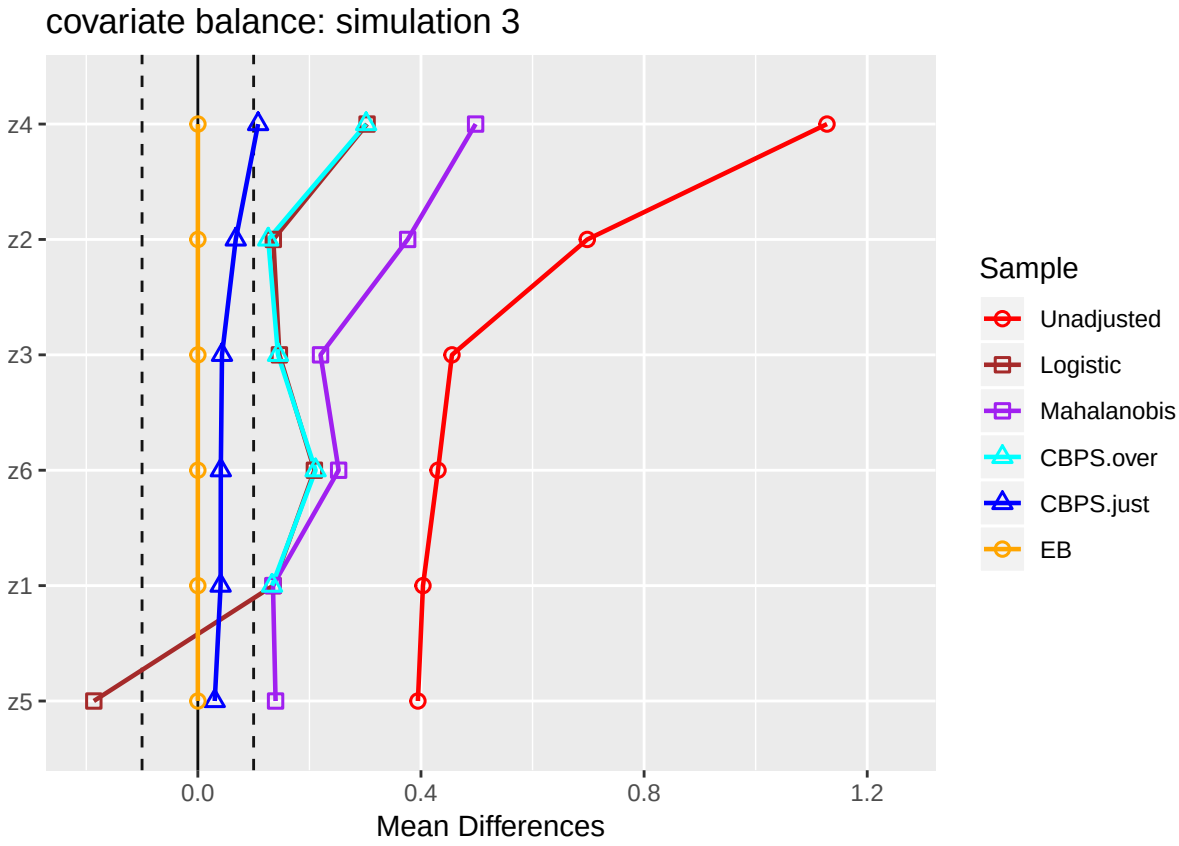
```
## Converged within tolerance
## Converged within tolerance
## Warning: Some extreme weights were generated. Examine them with summary()
## and maybe trim them with trim().
## Converged within tolerance
## Warning: Some extreme weights were generated. Examine them with summary()
## and maybe trim them with trim().
## Converged within tolerance
```

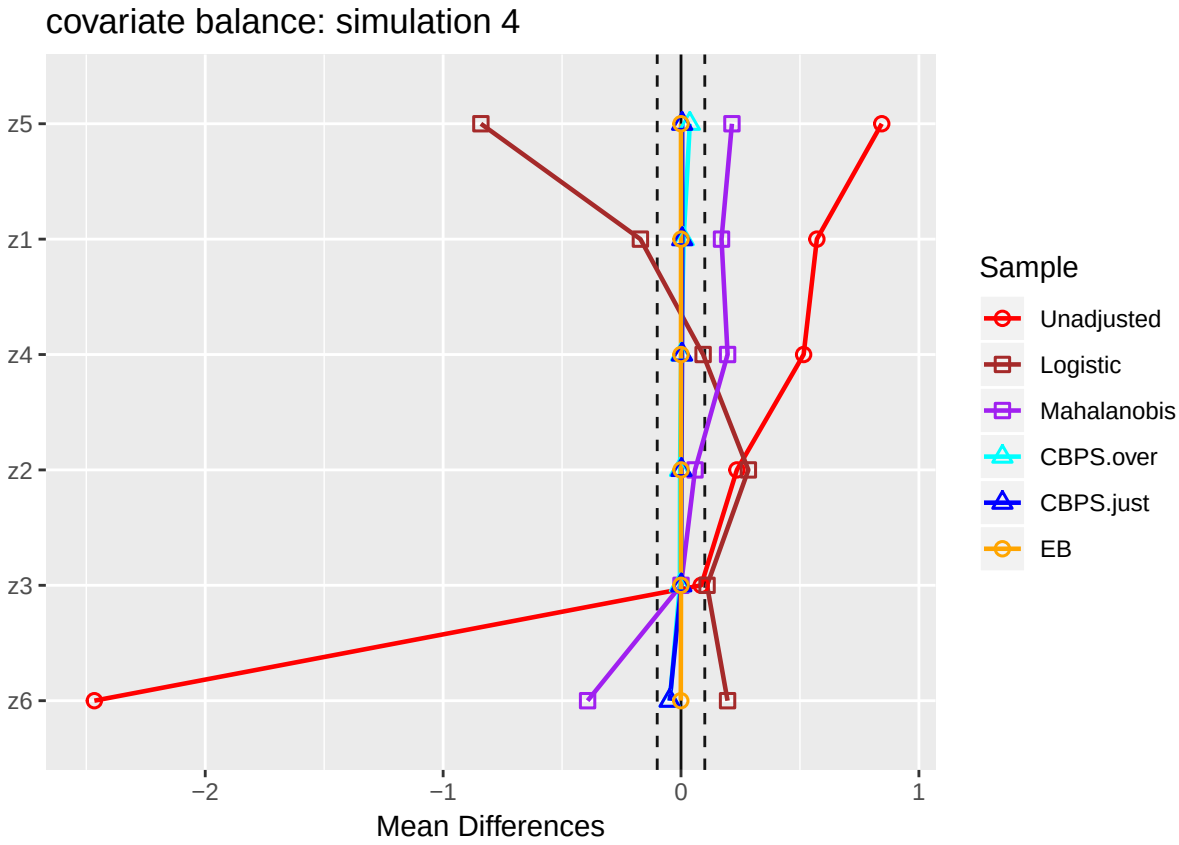
Examine Overlap of Propensity Score and Covariates (Before Matching)

```
xmin = c(-0.2, -0.2, -0.2, -2.5)
xmax = c(0.75, 0.75, 1.25, 0.9)

for (k in 1:4) {
    print(love.plot(bal.tab(covs = df[[k]][, confounders[[k]]],
        treat = df[[k]]$treat, data = df[[k]][, c("treat", confounders[[k]])],
        weights = data.frame(Logistic = get.w(mod_match[[k]]),
            Mahalanobis = get.w(mod_mahalo[[k]]), CBPS.over = get.w(cbps.over[[k]]),
            CBPS.just = get.w(cbps.just[[k]]), EB = df[[k]]$wt.eb),
        m.threshold = 0.1), var.order = "unadjusted", abs = FALSE,
        colors = c("red", "brown", "purple", "cyan", "blue",
            "orange"), shapes = c("circle open", "square open",
            "square open", "triangle open", "triangle open",
            "circle open"), line = TRUE) + theme_gray() + ggtitle(paste("covariate balance: simulation"
        k)) + xlim(xmin[k], xmax[k]))
}
```


covariate balance: simulation 1
z3
z1
z4
z2
0.00
0.25
0.50
0.75
Mean Differences
Sample
Unadjusted
Logistic
Mahalanobis
CBPS.over
CBPS.just
EB


covariate balance: simulation 2
z2
z4
z3
z1
0.00
0.25
0.50
0.75
Mean Differences
Sample
Unadjusted
Logistic
Mahalanobis
CBPS.over
CBPS.just
EB


covariate balance: simulation 3
z4
z2
z3
z6
z1
z5
0.0
0.4
0.8
1.2
Mean Differences
Sample
Unadjusted
Logistic
Mahalanobis
CBPS.over
CBPS.just
EB


covariate balance: simulation 4
z5
z1
z4
z2
z3
z6
−2
−1
0
1
Mean Differences
Sample
Unadjusted
Logistic
Mahalanobis
CBPS.over
CBPS.just
EB

```
# balplots = list() for(k in 1:4){ balplots[[k]] = list()
# covs = length(mod_match[[k]]$covs) model_n = 4 idx =
# seq(0,model_n*covs-1,by=4) i=1 # Unadjusted plots for(i in
# 1:length(idx)){ balplots[[k]][[idx[i]+1]] =
# bal.plot(mod_match[[k]],treat = df[[k]]$treat,covs =
# df[[k]][,confounders[[k]]], var.name =
# confounders[[3]][i],which='unadjusted',type='histogram') +
# guides(fill=FALSE) + theme(plot.title =
# element_text(size=8)) +
# ggtitle(paste('z1',confounders[[3]][i])) } for(i in
# 1:length(idx)){ balplots[[k]][[idx[i]+2]] =
# bal.plot(mod_match[[k]],treat = df[[k]]$treat,covs =
# df[[k]][,confounders[[k]]], var.name =
# confounders[[3]][i],type='histogram') + guides(fill=FALSE)
# + theme(plot.title = element_text(size=8)) +
# ggtitle('Baseline') } for(i in 1:length(idx)){
# balplots[[k]][[idx[i]+3]] = bal.plot(cbps.over[[k]],treat =
# df[[k]]$treat,covs = df[[k]][,confounders[[k]]], var.name =
# confounders[[3]][i],type='histogram') + guides(fill=FALSE)
# + theme(plot.title = element_text(size=8)) +
# ggtitle('CBPS') } for(i in 1:length(idx)){
# balplots[[k]][[idx[i]+4]] = bal.plot(eb.out[[k]],treat =
# df[[k]]$treat,covs = df[[k]][,confounders[[k]]], var.name =
# confounders[[3]][i],type='histogram') + guides(fill=FALSE)
# + theme(plot.title = element_text(size=8)) + ggtitle('EB')
# } } grid.arrange(grobs=balplots[[1]][1:8], ncol=4, nrow =
# 2, top='Simulation 1')
# grid.arrange(grobs=balplots[[1]][9:12], ncol=4, nrow = 2)
# grid.arrange(grobs=balplots[[2]][1:8], ncol=4, nrow = 2,
# top='Simulation 2') grid.arrange(grobs=balplots[[2]][9:12],
# ncol=4, nrow = 2) grid.arrange(grobs=balplots[[3]][1:8],
# ncol=4, nrow = 2, top='Simulation 3')
# grid.arrange(grobs=balplots[[3]][9:16], ncol=4, nrow = 2)
# grid.arrange(grobs=balplots[[3]][17:24], ncol=4, nrow = 2)
# grid.arrange(grobs=balplots[[4]][1:8], ncol=4, nrow = 2,
# top='Simulation 4') grid.arrange(grobs=balplots[[4]][9:16],
# ncol=4, nrow = 2) grid.arrange(grobs=balplots[[4]][17:24],
# ncol=4, nrow = 2)
```

Sampling Distributions of Estimated Treatment Effect

```
# for(k in 1:4){ baseline_coefs = rep(NA,n_sims) baseline_mh
# = rep(1,n_sims) cbps_coefs_over = rep(1,n_sims)
# cbps_coefs_just = rep(1,n_sims) eb_coefs = rep(1,n_sims)
# naive_est = rep(1,n_sims) for(i in 1:n_sims){ # Generate
# treatment vector treat = as.integer(runif(n) <= e_Z) y =
# y_0*(1-treat) + y_1*treat df[[k]] = as.data.frame(cbind(y,
# y_1, y_0, treat, Z_[,confounders[[k]]])) ## NAIVE-estimate
# naive_est[i] = mean(df[[k]]$y[df[[k]]$treat==1]) -
# mean(df[[k]]$y[df[[k]]$treat==0]) ## Baseline: Propensity
# Score using Logistic Regression: # Logistic Regression, 1-1
# Matching mod_match[[k]] = weightit(formula, data = df[[k]],
# method = 'ps', estimand = 'ATT') df[[k]]$wt =
```

```
# mod_match[[k]]$weights # Mahalanobis Distance Matching
# mod_mahalo[[k]] = matchit(formula, data = df[[k]], method =
# 'nearest', distance = 'mahalanobis', replace = TRUE)
# df[[k]]$wt_mh = mod_mahalo[[k]]$weights ## Covariate
# Balancing Propensity Score (CBPS) (Over) # CBPS Weights
# (OVER) cbps.over[[k]]= weightit(formula, data = df[[k]],
# method = 'cbps', estimand = 'ATT', over=TRUE)
# df[[k]]$wt.cbps_over = cbps.over[[k]]$weights # CBPS
# Weights (JUST) cbps.just[[k]]= weightit(formula, data =
# df[[k]], method = 'cbps', estimand = 'ATT', over=FALSE)
# df[[k]]$wt.cbps_just = cbps.just[[k]]$weights ## Entropy
# Balancing # Entropy Balancing Weights eb.out =
# ebalance(Treatment = df[[k]]$treat, X =
# df[[k]][,confounders[[k]]]) df[[k]]$wt.eb = 1
# df[[k]][df[[k]]$treat == 0,'wt.eb'] = eb.out$w # Linear
# Regression # mod.baseline.sim = lm(y ~ ., data =
# df[[k]][,c('y','treat',confounders[[k]])], weights =
# df[[k]]$wt) # mod.cbps = lm(y ~ ., data =
# df[[k]][,c('y','treat',confounders[[k]])], weights =
# df[[k]]$wt.cbps) # mod.eb = lm(y ~ ., data =
# df[[k]][,c('y','treat',confounders[[k]])], weights =
# df[[k]]$wt.eb) # Save Coefficients # baseline_coefs[i] =
# summary(mod.baseline.sim)$coefficients['treat',1] #
# cbps_coefs[i] = summary(mod.cbps)$coefficients['treat',1] #
# eb_coefs[i] = summary(mod.eb)$coefficients['treat',1]
# trt=df[[k]]$treat==1 ctrl=df[[k]]$treat==0 df.trt =
# df[[k]][trt,] df.ctrl = df[[k]][ctrl,] baseline_coefs[i] =
# weighted.mean(df.trt$y,df.trt$wt) -
# weighted.mean(df.ctrl$y,df.ctrl$wt) baseline_mh[i] =
# weighted.mean(df.trt$y,df.ctrl$wt_mh) -
# weighted.mean(df.ctrl$y,df.ctrl$wt_mh) cbps_coefs_over[i] =
# weighted.mean(df.trt$y,df.ctrl$wt.cbps_over) -
# weighted.mean(df.ctrl$y,df.ctrl$wt.cbps_over)
# cbps_coefs_just[i] =
# weighted.mean(df.trt$y,df.ctrl$wt.cbps_just) -
# weighted.mean(df.ctrl$y,df.ctrl$wt.cbps_just) eb_coefs[i] =
# weighted.mean(df.trt$y,df.ctrl$wt.eb) -
# weighted.mean(df.ctrl$y,df.ctrl$wt.eb) }
# save(baseline_coefs, baseline_mh, cbps_coefs_over,
# cbps_coefs_just, eb_coefs, file =
# paste('sim',k,'.RDATA',sep='')) }
```

Analysis of Simulation #1:

```
# k=1 load('sim1.RDATA') par(col='dark grey')
# plot(density(baseline_coefs), ylim=c(0,10),
# xlim=c(2.8,4.2),main='Dists of Treatment Eff Est')
# abline(v=mean(baseline_coefs), col='black', lty=2)
# lines(density(baseline_mh), col = 'purple')
# abline(v=mean(baseline_mh), col='purple', lty=2)
# lines(density(cbps_coefs_over), col = 'orange')
# abline(v=mean(cbps_coefs_over), col='orange', lty=2)
# lines(density(cbps_coefs_just), col = 'red')
```

```
# abline(v=mean(cbps_coefs_just), col='red', lty=2)
# lines(density(eb_coefs), col = 'green')
# abline(v=mean(eb_coefs), col='green', lty=2)
# abline(v=mean(SATE[[k]]), col='red', lty=2)
```

Analysis of Simulation #2:

```
# k=2 load('sim2.RDATA') plot(density(baseline_coefs),
# ylim=c(0,10), xlim=c(2.8,4.2),main='Dists of Treatment Eff
# Est') abline(v=mean(baseline_coefs), col='black', lty=2)
# lines(density(baseline_mh), col = 'purple')
# abline(v=mean(baseline_mh), col='purple', lty=2)
# lines(density(cbps_coefs_over), col = 'orange')
# abline(v=mean(cbps_coefs_over), col='orange', lty=2)
# lines(density(cbps_coefs_just), col = 'red')
# abline(v=mean(cbps_coefs_just), col='red', lty=2)
# lines(density(eb_coefs), col = 'green')
# abline(v=mean(eb_coefs), col='green', lty=2)
# abline(v=mean(SATE[[k]]), col='red', lty=2)
```

Analysis of Simulation #3:

```
# k=3 load('sim3.RDATA') plot(density(baseline_coefs),
# ylim=c(0,10), xlim=c(3,5),main='Dists of Treatment Eff
# Est') abline(v=mean(baseline_coefs), col='black', lty=2)
# lines(density(baseline_mh), col = 'purple')
# abline(v=mean(baseline_mh), col='purple', lty=2)
# lines(density(cbps_coefs_over), col = 'orange')
# abline(v=mean(cbps_coefs_over), col='orange', lty=2)
# lines(density(cbps_coefs_just), col = 'red')
# abline(v=mean(cbps_coefs_just), col='red', lty=2)
# lines(density(eb_coefs), col = 'green')
# abline(v=mean(eb_coefs), col='green', lty=2)
# abline(v=mean(SATE[[k]]), col='red', lty=2)
```

```
# k=4 load('sim4.RDATA') plot(density(baseline_coefs),
# ylim=c(0,10), xlim=c(3,5),main='Dists of Treatment Eff
# Est') abline(v=mean(baseline_coefs), col='black', lty=2)
# lines(density(baseline_mh), col = 'purple')
# abline(v=mean(baseline_mh), col='purple', lty=2)
# lines(density(cbps_coefs_over), col = 'orange')
# abline(v=mean(cbps_coefs_over), col='orange', lty=2)
# lines(density(cbps_coefs_just), col = 'red')
# abline(v=mean(cbps_coefs_just), col='red', lty=2)
# lines(density(eb_coefs), col = 'green')
# abline(v=mean(eb_coefs), col='green', lty=2)
# abline(v=mean(SATE[[k]]), col='red', lty=2)
```

Application to IHDP Data:

```
# load('hw4.Rdata') df.ihdp = hw4 col_names =
# colnames(df.ihdp) cols_to_keep = col_names[!col_names %in%
# c('treat','pputr.36','white','ltcoll','bwg','momed','st99')]
# ihdp.formula = formula(paste('pputr.36 ~
# ',paste(cols_to_keep,collapse = ' + ')))
# summary(lm(ihdp.formula, data = df.ihdp)) # Logistic
# Regression, 1-1 Matching ihdp.log = weightit(ihdp.formula,
# data = df.ihdp, method = 'ps', estimand = 'ATT') df.ihdp$wt
# = ihdp.log$weights # CBPS Weights (JUST) ihdb.cbps_j =
# weightit(ihdp.formula, data = df.ihdp, method = 'cbps',
# estimand = 'ATT', over=FALSE) df.ihdp$wt.cbps =
# ihdb.cbps_j$weights # Entropy Balancing Weights ihdp.eb =
# ebalance(Treatment = df.ihdp$treat, X =
# df.ihdp[,cols_to_keep]) df.ihdp$wt.eb = 1
# df.ihdp[df.ihdp$treat == 0,'wt.eb'] = ihdp.eb$w
# baltab.ihdp=list() baltab.ihdp[[1]] = bal.tab(data =
# df.ihdp, covs = df.ihdp[,cols_to_keep], treat =
# df.ihdp$treat, weights = df.ihdp$wt, method = 'weighting',
# disp.v.ratio = TRUE, un = TRUE, m.threshold = 0.1,
# v.threshold = 1.1, estimand = 'ATT') baltab.ihdp[[2]] =
# bal.tab(ihdb.cbps_j, disp.v.ratio = TRUE, un = TRUE,
# m.threshold = 0.1, v.threshold = 1.1, estimand = 'ATT')
# baltab.ihdp[[3]] = bal.tab(ihdp.eb, treat = df.ihdp$treat,
# covs = df.ihdp[,cols_to_keep], disp.v.ratio = TRUE, un =
# TRUE, m.threshold = 0.1, v.threshold = 1.1, estimand =
# 'ATT') love.plot(bal.tab(covs = df.ihdp[,cols_to_keep],
# treat = df.ihdp$treat, data =
# df.ihdp[,c('treat',cols_to_keep)], weights =
# data.frame(Logistic = get.w(ihdp.log), CBPS =
# get.w(ihdb.cbps_j), EB = df.ihdp$wt.eb)), var.order =
# 'unadjusted', abs = TRUE, colors = c('red', 'blue',
# 'darkgreen','orange'), shapes = c('circle', 'square',
# 'triangle','triangle')) + xlim(0,0.75)
```

Conclusion and Limitations: