Summary Report of Pairings 1 2 5 12 And 16

Group 6

4/7/2021

In this project, we are implementing 5 algorithms:

1. A1+D1 Propensity Matching with Mahalanobis distance measure
2. A1+D2+P1 Propensity Matching with propensity score distance measure + logistic regression
3. A1+D3+P1 Propensity Matching with linear Propensity score distance measure + logistic regression
4. A3+P1 Doubly Robust Estimation + logistic regression
5. A5+P1 Stratification + logistic regression

Setting up environment

```r
if(!require(MatchIt)) install.packages("MatchIt", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: MatchIt
```

```r
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ggplot2
```

```r
if(!require(lmtest)) install.packages("lmtest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lmtest

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
if(!require(sandwich)) install.packages("sandwich", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: sandwich
```

```r
if(!require(optmatch)) install.packages("optmatch", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: optmatch

## Loading required package: survival

## The optmatch package has an academic license. Enter relaxinfo() for more information.
```

```r
if(!require(broom)) install.packages("broom", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: broom
```

```r
library(MatchIt)
library(dplyr)
library(ggplot2)
library(lmtest)
library(sandwich)
library(optmatch)
library(broom)
```

Pre-Analysis

```r
# Reading in the datasets as R Dataframes
highDim_data = read.csv("../data/highDim_dataset.csv")
lowDim_data = read.csv("../data/lowDim_dataset.csv")

highDim_data %>%
  group_by(A) %>%
  summarise(n=n(),
            mean_outcome=mean(Y),
            std_error = sd(Y) / sqrt(n), .groups = "drop")
```

```
## # A tibble: 2 x 4
##       A     n mean_outcome std_error
## * <int> <int>        <dbl>     <dbl>
## 1     0  1357         29.5     0.588
## 2     1   643        -45.7     2.59
```

```r
with(highDim_data, t.test(Y ~ A))
```

```
##
##  Welch Two Sample t-test
##
## data:  Y by A
## t = 28.278, df = 708.98, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  69.95224 80.39043
## sample estimates:
## mean in group 0 mean in group 1
##        29.45850       -45.71284
```

```r
lowDim_data %>%
  group_by(A) %>%
  summarise(n=n(),
            mean_outcome=mean(Y),
            std_error = sd(Y) / sqrt(n), .groups = "drop")
```

```
## # A tibble: 2 x 4
##       A     n mean_outcome std_error
## * <int> <int>        <dbl>     <dbl>
## 1     0   394         16.6     0.531
## 2     1   106         27.2     1.49
```

```r
with(lowDim_data, t.test(Y ~ A))
```

```
##
##  Welch Two Sample t-test
##
## data:  Y by A
## t = -6.7071, df = 132.81, p-value = 5.201e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -13.728721  -7.475416
## sample estimates:
## mean in group 0 mean in group 1
##        16.56747        27.16953
```

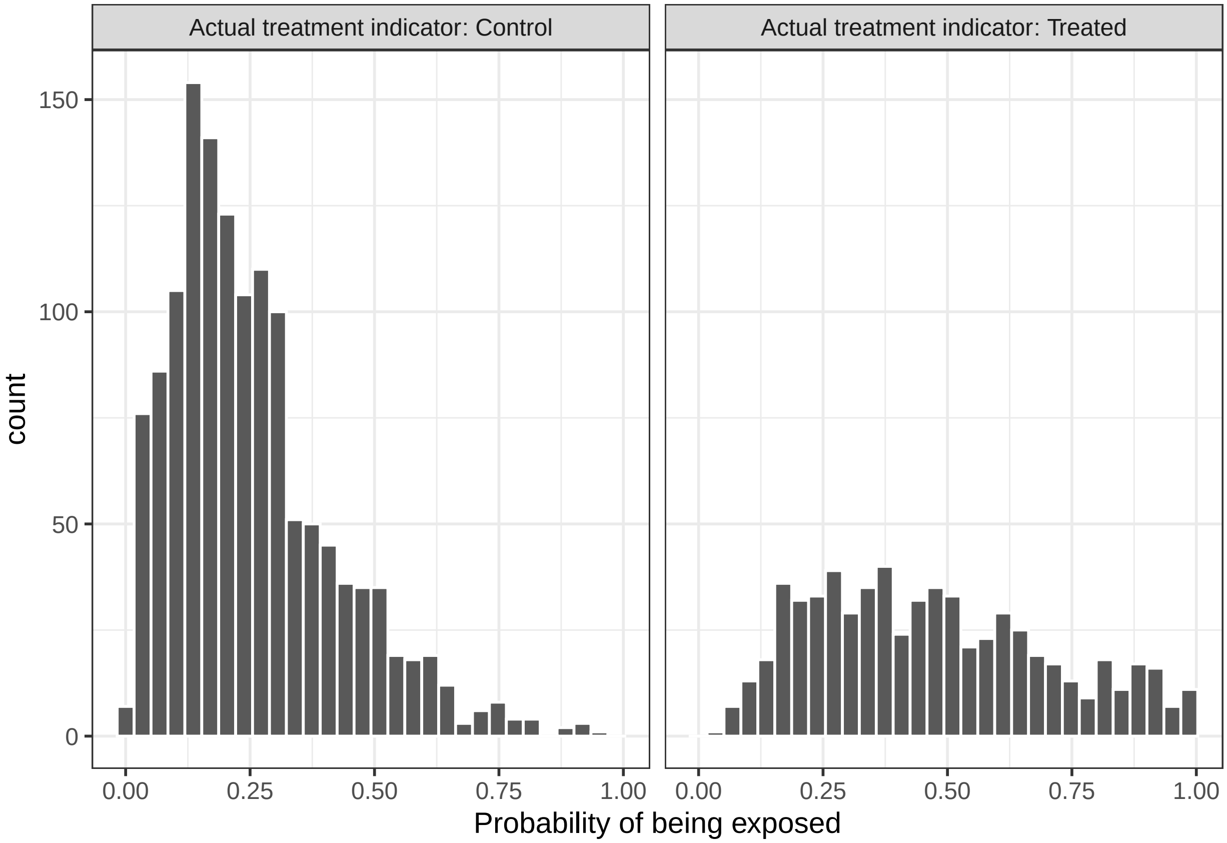
```r
true_low_ATE<-2.0901
true_high_ATE<- -54.8558
```

```r

# High dimension dataset
hd_time = system.time({ps_hd_est <- glm(A ~ . -Y,
                family=binomial(),
                data=highDim_data)

ps_hd_df <- data.frame(prop_score = predict(ps_hd_est, type = "response"),
                       treatment = ps_hd_est$model$A,
                       Y = ps_hd_est$model$Y)
})
ps_hd_time = round(as.numeric(hd_time[3]),3)

ps_hd_df %>%
  group_by(treatment) %>%
  summarise(mean_prop_score=mean(prop_score), .groups = "drop")
```

Use Logistic regression to get the propensity score

```
## # A tibble: 2 x 2
##   treatment mean_prop_score
## *     <int>           <dbl>
## 1         0           0.253
## 2         1           0.466
```

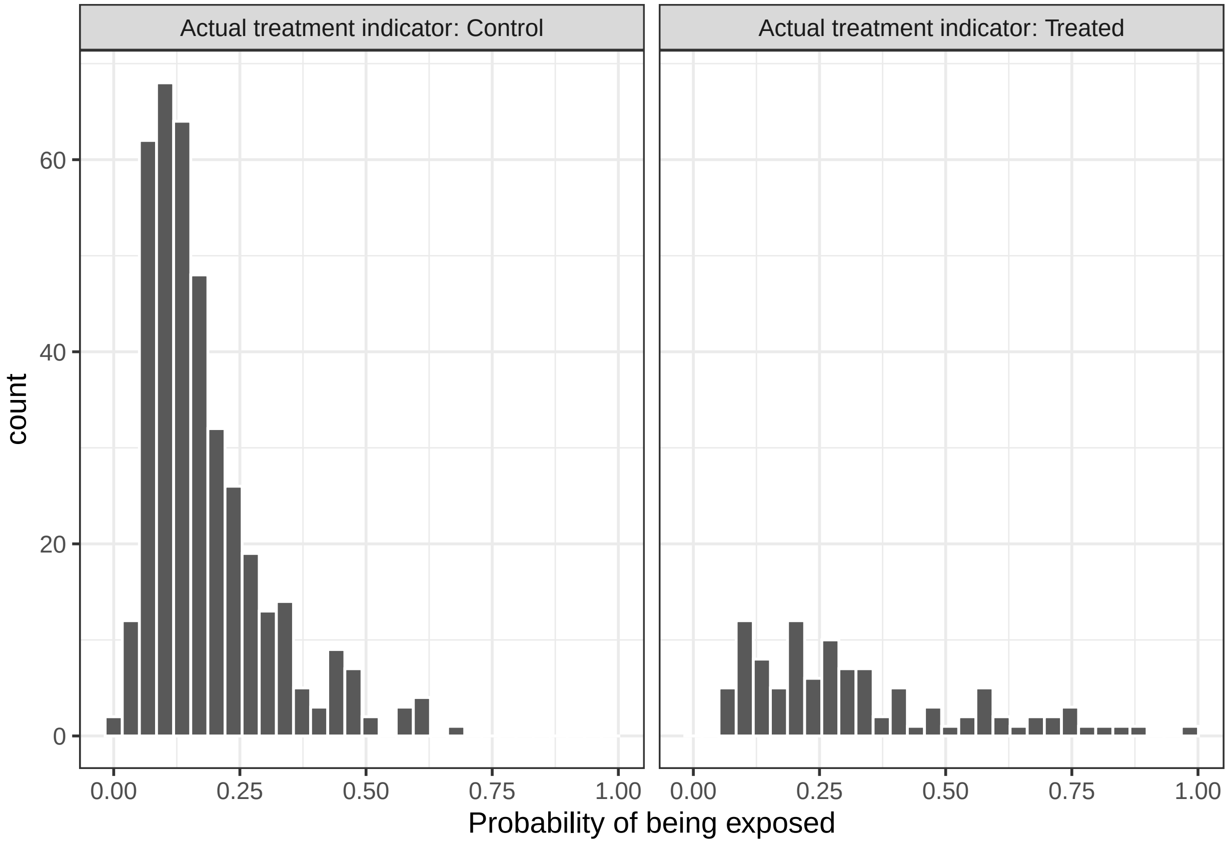
```r
labs <- paste("Actual treatment indicator:", c("Treated", "Control"))
ps_hd_df %>%
  mutate(treatment = ifelse(treatment == 1, labs[1], labs[2])) %>%
  ggplot(aes(x = prop_score), binwidth = 30) +
  geom_histogram(color = "white", bins=30) +
  facet_wrap(~treatment) +
  xlab("Probability of being exposed") +
  theme_bw()
```



```
# Low dimension dataset
ld_time = system.time({ps_ld_est <- glm(A ~ . -Y,
                  family=binomial(),
                  data=lowDim_data)

ps_ld_df <- data.frame(prop_score = predict(ps_ld_est, type = "response"),
                       treatment = ps_ld_est$model$A,
                       Y = ps_ld_est$model$Y)
})
ps_ld_time = round(as.numeric(ld_time[3]),3)

ps_ld_df %>%
  group_by(treatment) %>%
  summarise(mean_prop_score=mean(prop_score), .groups = "drop")
```

```
## # A tibble: 2 x 2
##   treatment mean_prop_score
## *     <int>           <dbl>
## 1         0           0.181
## 2         1           0.329
```

```r
labs <- paste("Actual treatment indicator:", c("Treated", "Control"))
ps_ld_df %>%
  mutate(treatment = ifelse(treatment == 1, labs[1], labs[2])) %>%
  ggplot(aes(x = prop_score)) +
  geom_histogram(color = "white", bins=30) +
  facet_wrap(~treatment) +
  xlab("Probability of being exposed") +
  theme_bw()
```



Causal Inference Methods

Pairing 1

Algorithm: `Propensity Matching`

Distance Measure: `Mahalanobis`

Propensity Score Estimation: `NA`

Introduction of the algorithm

- Propensity Matching motivations:

key assumption: If a control and treated individual are identical before treatment, the probability of the outcome variable is equal where there is no treatment applied (Y_0).

Matching on variables X thus ensures independence between treatment and Y_0 Matching on X is often infeasible especially where X is high dimensional. Instead, we use alternate distance measures to match on in order to circumvent this obstacle.

key parameters:

Distance Measure: Measure of similarity between 2 individuals Matching Method: How matching is conducted between individuals

- When estimating the ATE, either subclassification or full matching can be used. Full matching can be more effective because it optimizes a balance criterion, often leading to better balance. With full matching, it's also possible to exact match on some variables and match using the Mahalanobis distance, eliminating the need to estimate propensity scores. However, for large datasets, full matching may not be possible, in which case subclassification is a faster solution.
- The Mahalanobis distance is defined as:

$$D_ij = (X_i - X_j)^T \Sigma^{-1} (X_i - X_j)$$

Where Σ is the variance covariance matrix of X.

- Mahalanobis Distance does not require propensity score estimation and performs best with continuous variables.

implementation 1 code

```
mahalanobis.ate <- function(data){
  varnum=dim(data)[2]-2
  xnam <- paste0("V", 1:varnum)
  start_time <- Sys.time()
  match_full<-matchit(as.formula(paste("A ~ ", paste(xnam, collapse= "+"))),
                      data=data,method="full", distance="mahalanobis",
                      link = "logit", estimand = "ATE")
  data.fullMatching <- match.data(match_full)
  x = data.fullMatching %>%
    group_by(subclass,A) %>%
    summarise(mean_y = mean(Y), .groups = 'drop')
  group_ate = x %>%
    group_by(subclass) %>%
    summarise(treat_eff = mean_y[A == 1] - mean_y[A == 0], .groups = 'drop')
  group_n = data.fullMatching %>% group_by(subclass) %>% count()
  ate = sum(group_ate$treat_eff*group_n$n/nrow(data))
  end_time <- Sys.time()
  return(list(ATE=ate,running_time = end_time - start_time))
}
```

implementation 2 code

```
# Algorithm: Propensity Matching
# Distance Measure: Mahalanobis
# Propensity Score Estimation: NA
# Matching performed on low dimension dataset
covs = colnames(highDim_data)[-2:-1]
pair1_highdim_time = system.time({
  if (length(na.omit(highDim_data)) != length(highDim_data)) {
    print('There are null values in the dataset')
    break
    } else {
```

```
    pair_1 <- matchit(A ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V12 + V13 + V14 + V1
                      data=highDim_data,
                      method="full",
                      distance="mahalanobis",
                      estimand = "ATE")
  }

# Preparing Data Frame based on Pairing 1 Matches High Dim
mpair_1 = match.data(pair_1)
#Estimating Treatment Effects using Pairing 1 Matches High Dim
mpair1_fit <- lm(Y ~ . -Y -weights -subclass,
                 data = mpair_1,
                 weights = weights)
pair1_highdim_ATE = round(as.numeric(mpair1_fit$coefficients['A']),4)
})
pair1_hd_time <- round(as.numeric(pair1_highdim_time[3]),3)
```

```
# Algorithm: Propensity Matching
# Distance Measure: Mahalanobis
# Propensity Score Estimation: NA
# Matching performed on low dimension dataset
covs = colnames(lowDim_data)[-2:-1]
pair1_lowdim_time = system.time({
  if (length(na.omit(lowDim_data)) != length(lowDim_data)) {
    print('There are null values in the dataset')
    break
    } else {
    pair_1 <- matchit(A ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V12 +
                        V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20 + V21 + V22,
                      data=lowDim_data,
                      method="full",
                      distance="mahalanobis",
                      estimand = "ATE")
  }
})
```

High Dim data

```
## ATE for high dimensional data is: -63.32722 .

## Processing time for high dimensional data is 1.08163 seconds.

## ATE error for high dimensional data is: 8.471422 .
```

Low Dim data

```
## ATE for low dimensional data is: 2.289205 .

## Processing time for low dimensional data is 0.35482 seconds.

## ATE error for low dimensional data is: -0.199105 .
```

Pairing 2

Algorithm: `Propensity Matching`

Distance Measure: `Propensity Score`

Propensity Score Estimation: `Logistic Regression`

Introduction of the algorithm

- The Mahalanobis distance is defined as:

$$D_{ij} = |e_i - e_j|$$

where e_k is the propensity score for individual k.

- Propensity score definition:

$$p(\mathbf{X}) = P(exposed|\mathbf{X})$$

It is the probability of an individual being exposed (treated) given individual-specific characteristics.

implementation 1 code

```
Propensity.Score.ate <- function(data, methods, link){
  start_time <- Sys.time()
  match_full<-matchit(A ~ .-Y,data=data,method=methods,distance="glm",link = link, estimand = "ATE")
  data.fullMatching <- match.data(match_full)
  x = data.fullMatching %>%
    group_by(subclass,A) %>%
    summarise(mean_y = mean(Y), .groups = 'drop')
  group_ate = x %>%
    group_by(subclass) %>%
    summarise(treat_eff = mean_y[A == 1] - mean_y[A == 0], .groups = 'drop')
  group_n = data.fullMatching %>% group_by(subclass) %>% count()
  ate = sum(group_ate$treat_eff*group_n$n/nrow(data))
  end_time <- Sys.time()
  return(list(ATE=ate,running_time = end_time - start_time))
}
```

implementation 2 code

```
# Algorithm: Propensity Matching
# Distance Measure: Propensity Score
# Propensity Score Estimation: Logistic Regression
pair2 <- function(data, method) {
  pair2_time = system.time({
    if (length(na.omit(data)) != length(data)) {
      print('There are null values in the dataset')
      break
      }
    else {
      pair_2 <- matchit(A ~ .-Y,
                        data=data,
                        method=method,
                        distance="glm",
                        link="logit",
                        estimand = "ATE")
    }
    # Preparing Data Frame based on Pairing 2 Matches
    mpair_2 = match.data(pair_2)
    # Estimating Treatment Effects using Pairing 2 Matches
    mpair2_fit <- lm(Y ~ . -Y -weights -subclass -distance,
                     data = mpair_2,
                     weights = weights)
    pair2_ATE = round(as.numeric(mpair2_fit$coefficients['A']),4)
    })
  pair2_time <- round(as.numeric(pair2_time[3]),3)
  return(list(ATE=pair2_ATE, running_time = pair2_time))
}
```

optimal full matching

```
pair2fhigh = Propensity.Score.ate(highDim_data, methods="full", link="logit")
pair2fhigh_alt = pair2(highDim_data, method="full")
```

High Dim data

```
## ATE for implementation 1 is: -59.56389 .

## ATE error for implementation 1 is: 4.708088 .

## Processing time for implementation 1 is 4.031518 seconds.

##
## ATE for implementation 2 is: -60.4208 .

## ATE error for implementation 2 is: 5.565 .

## Processing time for implementation 2 is 3.816 seconds.

## Processing time for calculating the propensity score is 0.369 seconds.
```

```
pair2flow = Propensity.Score.ate(lowDim_data, methods="full", link="logit")
pair2flow_alt = pair2(lowDim_data, method="full")
```

Low Dim data

```
## ATE for implementation 1 is: 1.819887 .

## ATE error for implementation 1 is: 0.2702129 .

## Processing time for implementation 1 is 0.233439 seconds.

##
## ATE for implementation 2 is: 2.4034 .

## ATE error for implementation 2 is: -0.3133 .

## Processing time for implementation 2 is 0.18 seconds.

##
## Processing time for calculating the propensity score is 0.008 seconds.
```

subclassification

```
pair2shigh = Propensity.Score.ate(highDim_data, methods="subclass", link="logit")
pair2shigh_alt = pair2(highDim_data, method="subclass")
```

High Dim data

```
## ATE for implementation 1 is: -60.25466 .

## ATE error for implementation 1 is: 5.398856 .

## Processing time for implementation 1 is 0.4196379 seconds.

##
## ATE for implementation 2 is: -58.5693 .

## ATE error for implementation 2 is: 3.7135 .

## Processing time for implementation 2 is 0.452 seconds.

##
## Processing time for calculating the propensity score is 0.369 seconds.
```

```
pair2slow = Propensity.Score.ate(lowDim_data, methods="subclass", link="logit")
pair2slow_alt = pair2(lowDim_data, method="subclass")
```

Low Dim data

```
## ATE for implementation 1 is: 2.409903 .

## ATE error for implementation 1 is: -0.319803 .

## Processing time for implementation 1 is 0.02247715 seconds.

##
## ATE for implementation 2 is: 2.233 .

## ATE error for implementation 2 is: -0.1429 .

## Processing time for implementation 2 is 0.018 seconds.

##
## Processing time for calculating the propensity score is 0.008 seconds.
```

Pairing 5

> Algorithm: `Propensity Matching`
>
> Distance Measure: `Linear Propensity Score`
>
> Propensity Score Estimation: `Logistic Regression`

Introduction of the algorithm

The distance of Propensity Score is defined as:

$$D_{ij} = |e_i - e_j|$$

Obtained by applying the logit function on the Propensity Scores. Matching on the linear propensity score can be particularly effective in terms of reducing bias.

Linear propensity score matching is same with propensity score to entail forming matched sets of treated and untreated subjects who share a similar value of the propensity score. Once a matched sample has been formed, the treatment effect can be estimated by directly comparing outcomes between treated and untreated subjects in the matched sample. Once the effect of treatment has been estimated in the propensity score matched sample, the variance of the estimated treatment effect and its statistical significance can be estimated.

After the matched sets are obtained, Linear Propensity Score asks to calculate a "subclass effects" for each matched set/subclass, and then estimate overall ATE by an weighted average of the subclass effects where weights would be the number of individuals in each subclass.

Linear Propensity score performed well for low dimensional dataset as explained above for standard propensity score.

Linear Propensity Score didn't perform as well for high dimension as low dimensional dataset for the same reason discussed above for standard Propensity Score.

optimal full matching

```r
pair5fhigh = Propensity.Score.ate(highDim_data, methods="full", link="linear.logit")
pair5flow = Propensity.Score.ate(lowDim_data, methods="full", link="linear.logit")
```

High Dim data

```
## ATE for high dimensional data is: -60.14118 .

## ATE error for high dimensional data is: 5.28538 .

## Processing time for calculating the propensity score is 0.369 seconds.

## Processing time for pairing 5 with full matching is 3.543216 seconds.
```

Low Dim data

```
## ATE for low dimensional data is: 1.725128 .

## ATE error for low dimensional data is: 0.3649724 .

## Processing time for calculating the propensity score is 0.008 seconds.

## Processing time for pairing 5 with full matching is 0.1911891 seconds.
```

```r
pair5shigh = Propensity.Score.ate(highDim_data, methods="subclass", link="linear.logit")
pair5slow = Propensity.Score.ate(lowDim_data, methods="subclass", link="linear.logit")
```

subclassification

High Dim data

```
## ATE for high dimensional data is: -60.25466 .

## ATE error for high dimensional data is: 5.398856 .

## Processing time for calculating the propensity score is 0.369 seconds.
## Processing time for runing pairing 5 is 0.405273 seconds.
```

Low Dim data

```
## ATE for low dimensional data is: 2.409903 .
```

```
## ATE error for low dimensional data is: -0.319803 .
```

```
## Processing time for calculating the propensity score is 0.008 seconds.
```

```
## Processing time for runing pairing 5 is 0.02263904 seconds.
```

Pairing 12

Algorithm: `Doubly Robust Estimation`

Distance Measure: `Propensity Score`

Propensity Score Estimation: `Logistic Regression`

Introduction of the algorithm

- The Doubly Robust Estimation has the smallest asymptotic variance. It remains consistent if the outcome models are wrong but the propensity model is right, or if the propensity model is wrong but the outcome models are right.
- Doubly Robust Estimator formula:

$$\Delta\hat{}_{DR} = N^{-1}\sum_{i=1}^{N}\frac{T_iY_i - (T_i - \hat{e_i})\hat{m_1}(X_i)}{\hat{e_i}} - N^{-1}\sum_{i=1}^{N}\frac{(1-T_i)Y_i - (T_i - \hat{e_i})\hat{m_0}(X_i)}{1-\hat{e_i}}$$

where $\hat{e_i}$ is the estimated propensity score for individual i, $\hat{m_t}(X)$ is a consistent estimate for $E(Y|T=t,X)$ and is usually obtained by regressing the observed response Y on X in group t.

```
DoublyRobustEst <- function(data) {

  X = data %>% select(-Y, -A)
  n <- dim(data)[1]

  start_time1 <- Sys.time()
  #get propensity scores by using logistic regression
  logit_model <- glm(A ~., data=data[,-1], family="binomial")
  propensity <- predict(logit_model, X, type="response")
  data$ps <- propensity
  end_time1 <- Sys.time()

  start_time2 <- Sys.time()
  #split treatment and control group, and do regression for each group
  control <- data[data$A == 0, -2]
  treatment <- data[data$A == 1, -2]

  control_model <- lm(Y ~., data=control)
  treatment_model <- lm(Y ~., data=treatment)
```

```r
  data$m0 <- predict(control_model, data[,-c(1,2)])
  data$m1 <- predict(treatment_model, data[,-c(1,2)])

  #calculate ATE
  ATE <- sum((data$A*data$Y-(data$A-data$ps)*data$m1)/data$ps)/n -
         sum(((1-data$A)*data$Y+(data$A-data$ps)*data$m0)/(1-data$ps))/n

  end_time2 <- Sys.time()

  time1 <- round(end_time1 - start_time1,3)
  time2 <- round(end_time2 - start_time1,3)

  df <- data.frame(cbind(round(ATE,4), time1, time2))

  return(df)
}
```

```r
DRE.high <- DoublyRobustEst(highDim_data)
diff.high <- true_high_ATE-DRE.high[,1]
```

highDim data

```
## ATE for high dimensional data is: -56.422 .

## ATE error for high dimensional data is: 1.5662 .

## Processing time for calculating the propensity score is 0.381 seconds.

## Processing time for running the algorithm is 0.521 seconds.
```

```r
DRE.low <- DoublyRobustEst(lowDim_data)
diff.low <- true_low_ATE-DRE.low[,1]
```

lowDim data

```
## ATE for low dimensional data is: 2.0678 .

## ATE error for low dimensional data is: 0.0223 .

## Processing time for calculating the propensity score is 0.008 seconds.

## Processing time for running the algorithm is 0.016 seconds.
```

Pairing 16

Algorithm: `Stratification`

Distance Measure: `Propensity Score`

Propensity Score Estimation: `Logistic Regression`

Introduction of the algorithm

- Stratification method definition:

$$\hat{\Delta_S} = \sum_{j=1}^{K} \frac{N_j}{N} \{ N_{1j}^{-1} \sum_{i=1} N T_i Y_i I(\hat{e}_i \in \hat{Q}_j) - N_{0j}^{-1} \sum_{i=1}^{N} (1 - T_i) Y_i I(\hat{e}_i \in \hat{Q}_j) \}$$

Where K is the number of strata, N_j is the number of individuals in stratum j, N_{ij} is the number of treated individuals in stratum j, and N_{0j} is the number of controlled individuals in stratum j.

Here we use K=7 as advised by the first article (Chan, Ge, Gershony, Hesterberg & Lambert).

```
n_strat <- 7
#High Dimensional Data
hd_prep <- system.time({

  #Stratify the Data by Propensity score
  hd_strat_divider <- n_strat/nrow(ps_hd_df);
  strat_hd_df <- ps_hd_df %>% arrange(prop_score) %>%
    mutate(stratum =  as.integer((1:n() -.5)*hd_strat_divider ) + 1);

  #Differences in means
  diff_mean_hd <- strat_hd_df %>%
    group_by(treatment, stratum) %>%
    summarise(mean_treat = mean(Y), .groups = "keep") %>%
    mutate(mean_treat= if_else(treatment == 0, -mean_treat, mean_treat)) %>%
    ungroup(treatment) %>%
    mutate(diff_mean = sum(mean_treat)) %>%
    filter(treatment == 0) %>%
    select(stratum, diff_mean);
  #Weights for each stratum
  diff_weight_hd <- strat_hd_df %>%
    group_by(stratum) %>%
    tally() %>%
    mutate(weight = n/nrow(strat_hd_df)) %>%
    left_join(diff_mean_hd, by = "stratum") %>%
    select(weight, diff_mean);

  #Calculate the ATE
  ATE_hd <- sum(diff_weight_hd$diff_mean * diff_weight_hd$weight)
})
time_hd <-sum(hd_prep)
#Low Dimensional Data
ld_prep <- system.time({

  #Stratify the Data by Propensity score
  ld_strat_divider <- n_strat/nrow(ps_ld_df);
  strat_ld_df <- ps_ld_df %>% arrange(prop_score) %>%
    mutate(stratum =  as.integer((1:n() -.5)*ld_strat_divider ) + 1);
  #Differences in means
  diff_mean_ld <- strat_ld_df %>%
    group_by(treatment, stratum) %>%
    summarise(mean_treat = mean(Y), .groups = "keep") %>%
    mutate(mean_treat= if_else(treatment == 0, -mean_treat, mean_treat)) %>%
    ungroup(treatment) %>%
    mutate(diff_mean = sum(mean_treat)) %>%
    filter(treatment == 0) %>%
    select(stratum, diff_mean);
  #Weights for each stratum
  diff_weight_ld <- strat_ld_df %>%
    group_by(stratum) %>%
    tally() %>%
    mutate(weight = n/nrow(strat_ld_df)) %>%
    left_join(diff_mean_ld, by = "stratum") %>%
    select(weight, diff_mean);

  #Calculate the ATE
  ATE_ld <- sum(diff_weight_ld$diff_mean * diff_weight_ld$weight)
})
time_ld <- sum(ld_prep)
```

Perform stratification and find difference of means and weights.

Alternate Method that performs regression on each stratum by Y with A and Vs. Then averages the coefficients

- alternate Method definition:

$$\hat{\Delta^{(j)}} = n_j^{-1}\sum_{i=1}^{n} I(\hat{e}_i \in \hat{Q}_j)\{m^{(j)}(1, X_i, \hat{\alpha^{(j)}}) - m^{(j)}(0, X_i, \hat{\alpha^{(j)}})\}$$

- This is an alternative method for calculating the ATE once the rows are stratified by performing regression on each stratum for (Y ~ A + Covariates) and using the mean of the coefficients among strata as the ATE.
- In larger numbers of strata, there is a risk of getting NAs for coefficients for the last few Covariates, if the number of observations in a stratum is less than the number of covariates + 1

for A in each stratum to get the ATE

```
n_strat <- 7
#High Dimensional Data
hd_prep_alt <- system.time({
  hd_strat_divider <- n_strat/nrow(ps_hd_df);

  #Append propensity score to the initial dataset
  hd_pscore <- matrix(ps_hd_df$prop_score, ncol = 1);
  hd_df_pscore <- cbind(highDim_data, hd_pscore);

  strat_hd_df_alt <- hd_df_pscore %>% arrange(hd_pscore) %>%
    mutate(stratum =  as.integer((1:n() -.5)*hd_strat_divider ) + 1) %>%
    select(-hd_pscore);

  strat_hd_reg <- strat_hd_df_alt %>% group_by(stratum) %>%
    do(strat_reg = lm(Y ~ A + .-stratum, data = .)) %>%
    mutate(coefs = list(strat_reg[["coefficients"]])) %>%
    select(-strat_reg) %>%
    summarize(ajz = coefs[["A"]], .groups = "drop");

  #Calculate the ATE
  ATE_hd_alt <-  mean(strat_hd_reg$ajz)

})
time_hd_alt <- sum(hd_prep_alt)
#Low Dimensional Data
ld_prep_alt <- system.time({

  ld_strat_divider <- n_strat/nrow(ps_ld_df);

  #Append propensity score to the initial dataset
  ld_pscore <- matrix(ps_ld_df$prop_score, ncol = 1);
  ld_df_pscore <- cbind(lowDim_data, ld_pscore);

  strat_ld_df_alt <- ld_df_pscore %>% arrange(ld_pscore) %>%
    mutate(stratum =  as.integer((1:n() -.5)*ld_strat_divider ) + 1) %>%
    select(-ld_pscore);

  strat_ld_reg <- strat_ld_df_alt %>% group_by(stratum) %>%
    do(strat_reg = lm(Y ~ A + .-stratum, data = .)) %>%
    mutate(coefs = list(strat_reg[["coefficients"]])) %>%
    select(-strat_reg) %>%
    summarize(ajz = coefs[["A"]], .groups = "drop")
  #Calculate the ATE
  ATE_ld_alt <-  mean(strat_ld_reg$ajz)

})
time_ld_alt <- sum(ld_prep_alt)
```

Calculates the ATE and process time

High Dim data

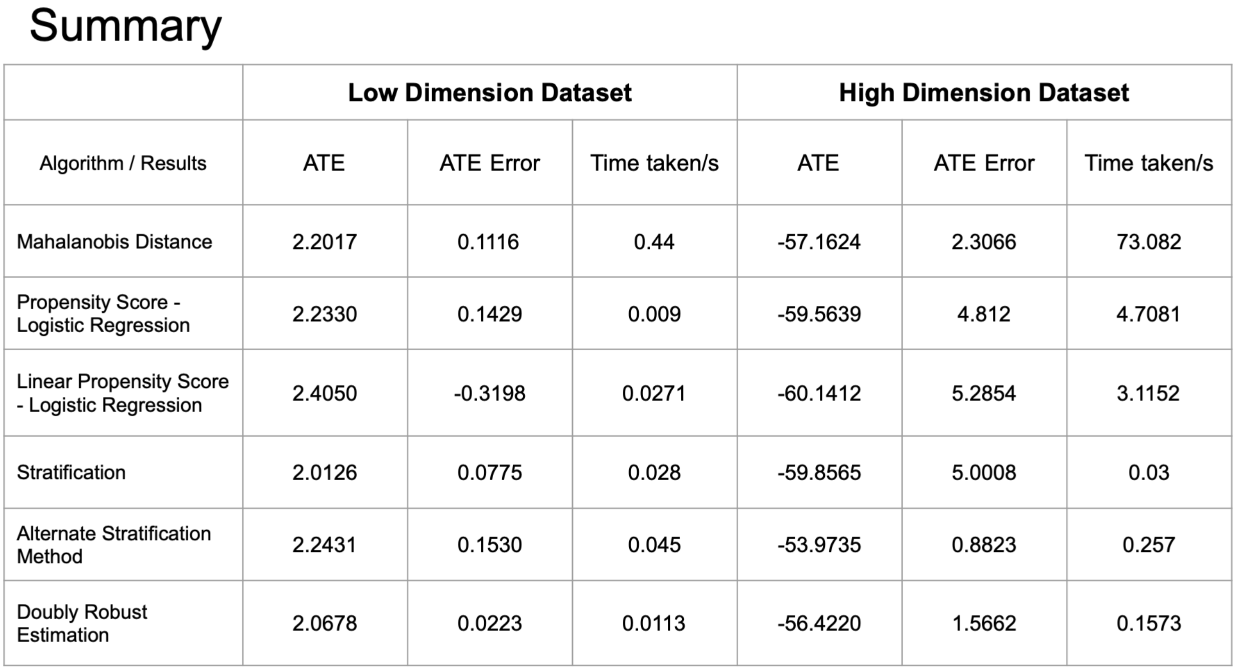
```
## ATE for stratification is: -59.85656 .

## ATE error for stratification is: 5.000763 .

## Processing time for stratification is 0.08 seconds.

##
## ATE for stratification alternative is: -53.97352 .

## ATE error for stratification alternative is: -0.88228 .

## Processing time for stratification is 0.483 seconds.

##
## Processing time for calculating the propensity score is 0.369 seconds.
```

Low Dim data

```
## ATE for stratification is: 2.012627 .
## ATE error for stratification is: 0.07747324 .

## Processing time for stratification is 0.072 seconds.

##
## ATE for stratification alternative is: 2.243051 .

## ATE error for stratification alternative is: -0.1529513 .

## Processing time for stratification is 0.1 seconds.

##
## Processing time for calculating the propensity score is 0.008 seconds.
```

Summary

The table shows the ATEs, ATE errors, and times for all of our algorithms for both low dimensional and high dimensional datasets.



Summary

	Low Dimension Dataset			High Dimension Dataset		
Algorithm / Results	ATE	ATE Error	Time taken/s	ATE	ATE Error	Time taken/s
Mahalanobis Distance	2.2017	0.1116	0.44	-57.1624	2.3066	73.082
Propensity Score - Logistic Regression	2.2330	0.1429	0.009	-59.5639	4.812	4.7081
Linear Propensity Score - Logistic Regression	2.4050	-0.3198	0.0271	-60.1412	5.2854	3.1152
Stratification	2.0126	0.0775	0.028	-59.8565	5.0008	0.03
Alternate Stratification Method	2.2431	0.1530	0.045	-53.9735	0.8823	0.257
Doubly Robust Estimation	2.0678	0.0223	0.0113	-56.4220	1.5662	0.1573

Based on our calculation on ATE and processing time for the five algorithms that we implemented, we conclude that stratification works best on low dimension dataset and the stratification with alternative method is the most efficient on high dimension dataset.

Sources

Chan, David, Rong Ge, Ori Gershony, Tim Hesterberg, and Diane Lambert. 2010. "Evaluating Online Ad Campaigns in a Pipeline: Causal Models at Scale." In Proceedings of the 16th Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 7–16.

Lunceford, Jared K, and Marie Davidian. 2004. "Stratification and Weighting via the Propensity Score in Estimation of Causal Treatment Effects a Comparative Study." Statistics in Medicine 23 (19): 2937–60.

P. Rosenbaum and D. B. Rubin. Reducing bias in observational studies using subclassification on the propensity score. Journal of the American Statistical Association, 79:516–524, 1984.

Stuart, Elizabeth A. 2010. “Matching Methods for Causal Inference: A Review and a Look Forward.” Statistical Science: A Review Journal of the Institute of Mathematical Statistics 25 (1): 1.