ATE Estimation using Full Propensity Matching (implemented from scratch) and Linear Propensity Scoring

Amir Idris

```r
if(!require("dplyr")){
  install.packages("dplyr")
}
```

```
## 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("MatchIt")){
  install.packages("MatchIt")
}
```

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

```r
if(!require("lmtest")){
  install.packages("lmtest")
}
```

```
## 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")
}
```

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

```r
if(!require("boot")){
  install.packages("boot")
}
```

```
## Loading required package: boot
```

```r
if(!require("survival")){
  install.packages("survival")
}
```

```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:boot':
##
##     aml
```

```r
if(!require("optmatch")){
  install.packages("optmatch")
}
```

```
## Loading required package: optmatch

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

```r
if(!require("glmnet")){
  install.packages("glmnet")
}
```

```
## Loading required package: glmnet

## Loading required package: Matrix

## Loaded glmnet 4.1-1
```

```r
if(!require("gbm")){
  install.packages("gbm")
}
```

```
## Loading required package: gbm

## Loaded gbm 2.1.8
```

```r
if(!require("rpart")){
  install.packages("rpart")
}
```

```
## Loading required package: rpart
```

```r
library(dplyr)
library(MatchIt)
library(lmtest)
library(sandwich)
library(boot)
library(survival)
library(optmatch)
library(glmnet)
library(gbm)
library(rpart)
```

First, let's load the data and summarize it.

```r
lowdim_data <- read.csv("../data/lowDim_dataset.csv")
highdim_data <- read.csv("../data/highDim_dataset.csv")
```

```r
print(dim(lowdim_data))
```

```
## [1] 500  24
```

```r
print(dim(highdim_data))
```

```
## [1] 2000  187
```

We can see that the high-dimensional data has an order of magnitude more dimensions, and four times the data as compared to the low dimensional data. Let's view a couple rows

```r
head(lowdim_data)
```

```
##          Y A   V1   V2   V3  V4 V5   V6  V7   V8   V9  V10  V11 V12  V13  V14
## 1 30.48700 0 0.00 0.00 0.00 0.0  0 0.00 0.0 0.00 0.00 0.00 0.00   0 0.00 0.00
## 2 18.20842 0 0.00 0.00 0.00 0.0  0 0.00 0.0 1.40 0.00 0.00 0.00   0 0.70 0.00
## 3 13.48504 0 0.00 0.00 0.00 0.0  0 0.00 0.0 0.00 0.00 0.00 0.00   0 0.00 0.00
## 4 25.69968 1 2.38 0.00 0.00 0.0  0 0.00 0.0 0.00 0.00 0.00 0.00   0 0.00 0.00
## 5 23.75297 0 0.15 0.15 0.05 0.1  0 0.42 0.1 0.95 0.42 0.05 0.05   0 0.00 0.36
## 6 13.63108 0 0.16 0.00 0.00 0.0  0 0.16 0.0 0.16 0.16 0.00 0.00   0 0.16 0.00
##    V15 V16  V17 V18  V19  V20  V21       V22
## 1 0.00   0 0.00   0 0.00 0.00 9.09 1.1496222
## 2 1.40   0 1.40   0 0.00 0.00 0.00 2.8877015
## 3 3.57   0 0.00   0 0.00 0.00 0.00 0.0000000
## 4 2.38   0 2.38   0 0.00 0.00 0.00 0.4054651
## 5 3.16   0 1.58   0 0.52 0.31 0.00 1.5746394
## 6 2.88   0 0.50   0 0.00 0.00 0.00 0.3798054
```

```r
head(highdim_data)
```

```
##           Y A V1 V2 V3 V4 V5 V6 V7    V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18
## 1 41.224513 0  0  1  4 18 17 -1  1  0.75  1  28   2  20  20  15  15   5  25   1
## 2 40.513875 0  0  0  1 10  6 -1 10  0.35  1  30   2  35  15  25  10   5  10   1
## 3 38.495476 0  0  0 16  8  4  4  4  0.40  1  26   2  30  25  20  10  10   5   1
## 4 33.001889 0  1  0  3 10  2 -1  5  0.41  0  24   4   5  15  45  25   0  10   1
## 5 37.043603 0  1  1 11 21 10 10 20  0.43  1  28   2  25  20  20  25   0  10   1
## 6 -2.877098 1  0  1 11 21 14 14 15 -0.20  1  28   1  20  18  20  17  10  15   1
##   V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32   V33 V34 V35 V36 V37
## 1   9  -1   8   8   7   5   7   6   2  13 141  10   1   1     0   2   1   3 205
## 2   7   6   3   5   6   5   6   4   2   7 152   2   1   1     0   2   1   2 234
## 3   8   9   9   7   9   7   7   8   2  12 239  12   9  29 25000   2   0   2  66
## 4   7   7   8   5   5   5   6   3   2  11 189   7   1   1     0   1   1   2 226
## 5   6   8   8   7   5  -1   8   5   2  11 242   4  54  59 25552   2   3   6 262
## 6   6  10  10   8   7   6   7   7   2  10 223   4 240  64 26100   1   9  11  40
##   V38 V39 V40 V41 V42 V43 V44 V45 V46 V47 V48 V49 V50 V51 V52 V53 V54 V55 V56
## 1  17   1 211   7   7   2   8  10   9   9   9   2   9  10   4  10  10   8  10
## 2 103   1 188   1   1   1   6   7   6   7   7   5   7   7   7   9   7   8   7
## 3  50   1 315  11   6   2   5   8   5   6   5   2   9   9   8   9   9   6   6
## 4 198   3 225   6   3   2   2   7   8   8   3   3   7  10   7   5   9   7   7
## 5 195   3 258   2   9   5   3   8   5   5   6   4   4   9   2   6   7   5   7
## 6  11   2 353  10   1   1   1   5   7   7   7   1   3   9   4   7   7   7   8
##   V57 V58 V59 V60 V61 V62 V63 V64 V65 V66 V67 V68 V69 V70 V71 V72 V73 V74 V75
## 1   6   9   5   6  10  20  20  20  20  10  -1  -1  -1  -1  -1  -1  30  10  10
## 2   1   8   1   2  20  20  20  20  10  10  -1  -1  -1  -1  -1  -1  30  10  15
## 3   4   7   5  -1   5  30  20  10   5  30  50   5  10  20   5  10  50  10   5
## 4   7   1   5   2  21  17  22  20   8  13  -1  -1  -1  -1  -1  -1  18  15  17
```

```
## 5   2   2   5  -1  20  20  15  25  10  10  40   5  10  20   5  20  30  20   5
## 6   9   6   1  -1  20  20  20  10  20  10  10  30  30   5  20   5  50  30   5
##   V76 V77 V78 V79 V80 V81 V82 V83 V84 V85 V86 V87 V88 V89 V90 V91 V92 V93 V94
## 1  30  10  10   9  10   9   9   9  -1  -1  -1  -1  -1   6   9   9   8   8   5
## 2  30   5  10   7   8   9   7   8  -1  -1  -1  -1  -1   7  10  10   7  10   5
## 3  10   5  20   9   8   5   9   3   8   8   9   3   8   4   7   6   6   5   8
## 4  14  20  16   6   8   7   8   8  -1  -1  -1  -1  -1   4   7   7   6   5   3
## 5  25  10  10   5   8   7   8   7   5   8   7   6   6   7   6   7   6   5   5
## 6   5   5   5   8   8   8   8   5   8   7  10   9   8  10   8   9  10  10   7
##   V95 V96 V97 V98 V99 V100 V101 V102 V103 V104 V105 V106 V107 V108 V109 V110
## 1   7   8   2   3  -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1    6
## 2   5   1   2   2  -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1    4
## 3   6   8   2   3  20   20   20   10    5   25    8    8    8    4    4   -1
## 4   6   5   2  -1  -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1    4
## 5   6   3   0   1  35   15   20   20    5    5    4    7    5    4    7    4
## 6   9   4   0   2  -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1    5
##   V111 V112 V113 V114 V115 V116 V117 V118  V119  V120  V121  V122  V123  V124
## 1    1    2   -1   -1   -1   -1   -1   -1 20.00 20.00 15.00 20.00 10.00 15.00
## 2    3    2   -1   -1   -1   -1   -1   -1 24.14 13.79 20.69 27.59 10.34  3.45
## 3   -1   -1   -1   -1   -1   -1   -1   -1 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00
## 4    1    2   -1   -1   -1   -1   -1   -1 25.00 10.00 20.00 20.00  5.00 20.00
## 5    1    2   -1   -1   -1   -1   -1   -1 30.00 20.00 15.00 30.00  5.00  0.00
## 6    3    2   -1   -1   -1   -1   -1   -1 20.00 20.00 20.00 20.00 10.00 10.00
##   V125 V126 V127 V128 V129 V130  V131  V132  V133  V134  V135  V136 V137 V138
## 1   -1   -1   -1   -1   -1   -1 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00    7    9
## 2   -1   -1   -1   -1   -1   -1 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00    6    8
## 3   -1   -1   -1   -1   -1   -1 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00   -1   -1
## 4   -1   -1   -1   -1   -1   -1 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00    6    7
## 5   40   15   15   30    0    0 33.33 11.11  5.56 33.33 16.67  0.00    5    7
## 6   10   30   10   10   30   10 36.36 18.18  9.09 18.18  4.55 13.64    8    8
##   V139 V140 V141 V142 V143 V144 V145 V146 V147 V148 V149 V150 V151 V152 V153
## 1    9    9    8   -1   -1   -1   -1   -1    1    4    1   -1   -1   15   20
## 2    7    8    6   -1   -1   -1   -1   -1    0    0    0   -1   -1   20   20
## 3   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 4    8    9    8   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 5    7    5    7    5    7    7    4    7    0    4    1    2   -1   30   20
## 6    8   10    6    8    8    9   10    8   -1   -1   -1   -1   -1   -1   -1
##   V154 V155 V156 V157 V158 V159 V160 V161 V162 V163 V164 V165 V166 V167 V168
## 1   15   20   15   15   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 2   20   20    0   20   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 3   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 4   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 5   20   20    5    5   30   20   20   30    0    0   40   10   20   20    5
## 6   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
##   V169 V170 V171 V172 V173 V174 V175 V176 V177 V178 V179 V180 V181 V182 V183
## 1   -1   -1   -1   -1   -1   -1   -1    8   10    8    9    8   -1   -1   -1
## 2   -1   -1   -1   -1   -1   -1   -1    6    5    6    8    5   -1   -1   -1
## 3   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 4   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
## 5    5   30   10   20   10   20   10    6    8    7    7    7    6    7    6
## 6   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
##   V184 V185
## 1   -1   -1
## 2   -1   -1
```

```
## 3   -1   -1
## 4   -1   -1
## 5    6    7
## 6   -1   -1
```

First, we must train models to estimate the propensity scores. We have five models assigned to us:

P1: Logistic Regression
P2: L1 Penalized Logistic Regression
P3: L2 penalized logistic regression
P4: Regression trees
P5: Boosted stumps

```r
# Model selector method
source("../lib/LR_propensity_est.R")
source("../lib/boosted_stumps_propensity_est.R")
source("../lib/regression_tree_propensity_est.R")
model_selector <- function(data, mode = 1){
  if (mode == 1) {
    return(lr_propensity_model(data))
  } else if (mode == 2 | mode == 6){
    return(lr_propensity_model(data, mode = "lasso"))
  } else if (mode == 3){
    return(lr_propensity_model(data, mode = "ridge"))
  } else if (mode == 4){
    return(regression_tree_propensity_model(data))
  } else{
    return(gbm_propensity_model(data))
  }
}
```

```r
# P1-P5, A7 + P2 included
m <- 6 # number of models
lowdim_models <- list()
highdim_models <- list()
lowdim_model_times <- list()
highdim_model_times <- list()
for(i in 1:m){
  lowdim_model_times[[i]] <- system.time({lowdim_models[[i]] <- model_selector(lowdim_data, mode = i);}
}
for(i in 1:m){
  highdim_model_times[[i]] <- system.time({highdim_models[[i]] <- model_selector(highdim_data, mode = i
}
```

Then, we'll estimate propensity scores, and transform them to use linear propensity distance.

**It's worth noting that we can't have propensity scores that are exactly 0 or exactly 1, since that will give us infinity for our logit transformation. So, the entries from the regression tree that match this criteria are added/subtracted a small dx so that they don't go to infinity.

```r
lowdim_prop_scores <- list()
highdim_prop_scores <- list()
lowdim_score_times <- list()
highdim_score_times <- list()

predict_selector <- function(features, model, mode = 1){
  if (mode == 1) {
```

```
    return(lr_propensity(features, model))
  } else if (mode == 2 | mode == 6){
    return(lr_propensity(features, model, mode = mode))
  } else if (mode == 3){
    return(lr_propensity(features, model, mode = mode))
  } else if (mode == 4){
    return(regression_tree_propensity(features, model))
  } else{
    return(gbm_propensity(features, model))
  }
}

for(i in 1:m){
  lowdim_score_times[[i]] <- system.time({
    if(i < 6){
      lowdim_prop_scores[[i]] <- logit(predict_selector(lowdim_data, lowdim_models[[i]], mode = i))
    }
    else{
      lowdim_prop_scores[[i]] <- predict_selector(lowdim_data, lowdim_models[[i]], mode = i)
    };})
}
for(i in 1:m){
  highdim_score_times[[i]] <- system.time({
    if (i < 6){
      highdim_prop_scores[[i]] <- logit(predict_selector(highdim_data, highdim_models[[i]], mode = i))
    }
    else{
      highdim_prop_scores[[i]] <- predict_selector(highdim_data, highdim_models[[i]], mode = i)
    };})
}
```

Now, we'll proceed with the full matching, using the *MatchIt* package.

This part is the most time intensive, since it must find the optimal matching such that total discrepancy between matched comparisons is minimized, so it can't take a greedy approach. *(Hansen, 2004)*

```
m <- 5 # We'll handle weighted regression separately
lowdim_data_match <- subset(lowdim_data, select = -c(Y))
highdim_data_match <- subset(highdim_data, select = -c(Y))
lowdim_matches <- list()
highdim_matches <- list()
lowdim_match_times <- list()
highdim_match_times <- list()

for(i in 1:m){
  lowdim_match_times[[i]] <- system.time({lowdim_matches[[i]] <- matchit(A ~ ., data = lowdim_data_matc
}
for(i in 1:m){
  highdim_match_times[[i]] <- system.time({highdim_matches[[i]] <- matchit(A ~ ., data = highdim_data_ma
}

# Example matchings
lowdim_matches[[1]]
```

```
## A matchit object
```

```
##  - method: Optimal full matching
##  - distance: User-defined
##  - number of obs.: 500 (original), 500 (matched)
##  - target estimand: ATE
##  - covariates: V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19,
```

```
highdim_matches[[1]]
```

```
## A matchit object
##  - method: Optimal full matching
##  - distance: User-defined
##  - number of obs.: 2000 (original), 2000 (matched)
##  - target estimand: ATE
##  - covariates: too many to name
```

We obtain datasets of the matches, and bind our outcome Y back to them.

```
lowdim_match_sets <- list()
highdim_match_sets <- list()
for(i in 1:m){
  lowdim_match_sets[[i]] <- match.data(lowdim_matches[[i]])
  Y.low <- lowdim_data$Y
  lowdim_match_sets[[i]] <- as.data.frame(cbind(Y.low, lowdim_match_sets[[i]]))
}
for(i in 1:m){
  highdim_match_sets[[i]] <- match.data(highdim_matches[[i]])
  Y.high <- highdim_data$Y
  highdim_match_sets[[i]] <- as.data.frame(cbind(Y.high, highdim_match_sets[[i]]))
}
```

Finally, we estimate our ATEs.

```
lowdim_ATEs <- c()
highdim_ATEs <- c()

for(i in 1:m){
  fit.low <- lm(Y.low ~ A, data = lowdim_match_sets[[i]], weights = weights)
  coeftest(fit.low, vcov. = vcovCL, cluster = ~subclass)
  lowdim_ATEs <- c(lowdim_ATEs, summary(fit.low)$coefficients["A", "Estimate"])
}
for(i in 1:m){
  fit.high <- lm(Y.high ~ A, data = highdim_match_sets[[i]], weights = weights)
  coeftest(fit.high, vcov. = vcovCL, cluster = ~subclass)
  highdim_ATEs <- c(highdim_ATEs, summary(fit.high)$coefficients["A", "Estimate"])
}

# Weighted regression
source("../lib/weighted_regression.R")
lowdim_match_times[[m+1]] <- system.time({lowdim_ATEs[[m+1]] <- weighted_regression(lowdim_data, lowdim
highdim_match_times[[m+1]] <- system.time({highdim_ATEs[[m+1]] <- weighted_regression(highdim_data, high
```

How do our ATE estimates compare to the real ones?

```
m <- 6
# Real ATEs
lowdim_ATE_real <- 2.0901
highdim_ATE_real <- -54.8558
```

```r
#SDs of outcomes, so we can normalize our difference in estimation
sd.low <- sd(Y.low)
sd.high <- sd(Y.high)

ATE_diff.low <- (lowdim_ATEs - lowdim_ATE_real)/sd.low
ATE_diff.high <- (highdim_ATEs- highdim_ATE_real)/sd.high

method_names <- c("A1 + D3 + P1", "A1 + D3 + P2", "A1 + D3 + P3", "A1 + D3 + P4", "A1 + D3 + P5", "A7 +

ATE_table <- as.matrix(cbind(lowdim_ATEs, ATE_diff.low, highdim_ATEs, ATE_diff.high))
rownames(ATE_table) <- method_names
colnames(ATE_table) <- c("Low-Dim ATE Estimate", "Low-Dim Error in SDs", "High-Dim ATE Estimate", "High-
ATE_table
```

```
##              Low-Dim ATE Estimate Low-Dim Error in SDs High-Dim ATE Estimate
## A1 + D3 + P1             1.725128        -2.925137e-02             -60.14118
## A1 + D3 + P2             1.171943        -7.358735e-02             -57.18506
## A1 + D3 + P3             1.666441        -3.395494e-02             -59.20897
## A1 + D3 + P4             4.369954         1.827230e-01             -52.62841
## A1 + D3 + P5             2.371125         2.252324e-02             -53.38666
## A7 + P2                  2.090175         6.028979e-06             -57.95906
##              High-Dim Error in SDs
## A1 + D3 + P1           -0.09748049
## A1 + D3 + P2           -0.04295943
## A1 + D3 + P3           -0.08028740
## A1 + D3 + P4            0.04108073
## A1 + D3 + P5            0.02709601
## A7 + P2                -0.05723466
```

Not bad! Our worst estimate is less than 0.2 standard deviations away from the true ATE.

Now, let's take a look at the time our algorithm took:

```r
lowdim_model_time <- c()
highdim_model_time <- c()
lowdim_score_time <- c()
highdim_score_time <- c()
lowdim_match_time <- c()
highdim_match_time <- c()

for(i in 1:m){
  lowdim_model_time <- c(lowdim_model_time, lowdim_model_times[[i]][3])
  highdim_model_time <- c(highdim_model_time, highdim_model_times[[i]][3])
  lowdim_score_time <- c(lowdim_score_time, lowdim_score_times[[i]][3])
  highdim_score_time <- c(highdim_score_time, highdim_score_times[[i]][3])
  lowdim_match_time <- c(lowdim_match_time, lowdim_match_times[[i]][3])
  highdim_match_time <- c(highdim_match_time, highdim_match_times[[i]][3])
}

time_table <- as.matrix(cbind(lowdim_model_time, highdim_model_time, lowdim_score_time, highdim_score_t
total <- rowSums(time_table)
time_table <- as.matrix(cbind(time_table, total))
rownames(time_table) <- method_names
colnames(time_table) <- c("Low-Dim Model Train Time", "High-Dim Model Train Time", "Low-Dim Scoring Tim
```

```
time_table
```

```
##              Low-Dim Model Train Time High-Dim Model Train Time
## A1 + D3 + P1                    0.011                     0.356
## A1 + D3 + P2                    0.034                     3.453
## A1 + D3 + P3                    0.022                     0.561
## A1 + D3 + P4                    0.043                     0.859
## A1 + D3 + P5                    0.024                     0.360
## A7 + P2                         0.011                     3.382
##              Low-Dim Scoring Time High-Dim Scoring Time Low-Dim Matching Time
## A1 + D3 + P1                0.003                 0.005                 0.160
## A1 + D3 + P2                0.034                 0.018                 0.153
## A1 + D3 + P3                0.004                 0.014                 0.158
## A1 + D3 + P4                0.022                 0.007                 0.136
## A1 + D3 + P5                0.003                 0.027                 0.147
## A7 + P2                     0.004                 0.013                 0.063
##              High-Dim Matching Time Total
## A1 + D3 + P1                  3.208 3.743
## A1 + D3 + P2                  2.661 6.353
## A1 + D3 + P3                  2.643 3.402
## A1 + D3 + P4                  2.317 3.384
## A1 + D3 + P5                  2.622 3.183
## A7 + P2                       0.962 4.435
```

As we can see with quadruple the data, the full matching algorithm took about 16.781 times longer for the high-dimensional dataset, and the other components of the process were negligible in comparison. So, we can hypothesis that we have an O(nˆ2) time complexity.

Weighted Regression seems to be the overall champion, with a low runtime compared to full-matching, and very good estimates, with an almost exact estimate on the low-dimensional dataset.

References:
- https://cran.r-project.org/web/packages/MatchIt/vignettes/estimating-effects.html#after-full-matching
- Hansen BB. Full matching in an observational study of coaching for the SAT. Journal of the American Statistical Association. 2004;99(467):609–618.