Testing Report

Xujie Ma, Yiqi Lei, Xinyi Zhang, Jiaqi Yuan, Yue Liang

11/26/2020

In this notebook, we are presenting 3 algorithms:

1. 14 A3+P5 Doubly Robust Estimation + boosted stumps
2. 21 A6+P5 Regression Adjustment + boosted stumps
3. 15 A4 Regression Estimate

```
library(gbm)
library(dplyr)
```

Data Import

```
high <- read.csv('../data/highDim_dataset.csv')
low <- read.csv('../data/lowDim_dataset.csv')
```

Algo 1: 14 A3+P5 Doubly Robust Estimation + boosted stumps

highDim_dataset

```
# train-test split
n <- nrow(high)
n_train <- round(n*(4/5),0)
train_idx <- sample(1:n,n_train)
train_high <- high[train_idx,]
test_high <- high[-train_idx,]
```

Split treatment and control group, and complete regression for each group.

```
treatment.group.high<-high[high$A==1,-2]
control.group.high<-high[high$A==0,-2]

treatment.model.high<-lm(Y~.,data=treatment.group.high)
control.model.high<-lm(Y~.,data=control.group.high)
```

Estimate m1(X) and m0(X) for all entries.

```
X.high<-high[-c(1,2)]

high$m1<-predict(treatment.model.high,X.high)
high$m0<-predict(control.model.high,X.high)
```

Get propensity score for all entries using boosted stumps (Gradient Boosting Machine).

Using grid search to get proper parameters for gbm.

```
# grid search
hyper_grid_high1 <- expand.grid(
  n.trees = c(40,50,60),
  shrinkage = c(.01, .05, .1),
```

```r
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(.65, .8, 1),
  optimal_trees = 0,               # a place to dump results
  min_RMSE = 0                     # a place to dump results
)

# randomize data
random_index <- sample(1:nrow(train_high), nrow(train_high))
random_ames_train <- train_high[random_index, ]

# grid search
for(i in 1:nrow(hyper_grid_high1)) {
  # reproducibility
  set.seed(2020)
  # train model
  gbm.tune <- gbm(
    formula = A~.,
    distribution = "bernoulli",
    data = train_high[-1],
    n.trees = hyper_grid_high1$n.trees[i],
    interaction.depth = 1,
    shrinkage = hyper_grid_high1$shrinkage[i],
    n.minobsinnode = hyper_grid_high1$n.minobsinnode[i],
    bag.fraction = hyper_grid_high1$bag.fraction[i],
    train.fraction = .75
  )

  # add min training error and trees to grid
  hyper_grid_high1$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid_high1$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
}

hyper_grid_high1 %>%
  dplyr::arrange(min_RMSE) %>%
  head(10)
```

Apply the parameters with min_RMSE (n.trees=60, shrinkage=0.1, n.minobsinnode=10, bag.fraction=1).

```r
set.seed(2020)

tm_highe1 <- system.time(
  boost.high<-gbm(A~., data = train_high[-1],
                distribution = "bernoulli",
                 n.trees = 60, # the number of trees
                 shrinkage = 0.1, # learning rate
                 interaction.depth = 1, # total split
                n.minobsinnode = 10,
                bag.fraction = 1
                 )
  )
```

Calculate propensity scores for all entries in high.csv

```r
tm_highe2 <- system.time(
  high$e <- predict(boost.high, X.high, n.trees = 60, type = 'response')
)
```

Calculate each part in doubly robust estimation and count out the final result.

```
tm_highATE1 <- system.time(
  {high$p1<-ifelse(high$A==1,(high$Y-high$m1)/high$e,0);
  high$p2<-ifelse(high$A==0,(high$Y-high$m0)/(1-high$e),0);
  high$result<-high$m1-high$m0+high$p1-high$p2;
  ATE.high<-mean(high$result)}
  )

ATE.high
```

```
## [1] -2.959545
```

```
#alternative function, same result

#tm_highATE2 <- system.time(
#  ATE.high<-1/n*(sum((high$A*high$Y-(high$A-high$e)*high$m1)/high$e)
#            -sum(((1-high$A)*high$Y+(high$A-high$e)*high$m0)/(1-high$e))))
#ATE.high
```

```
# True ATE:
true_ATE_high <- -3

# Comparison:
true_ATE_high - ATE.high
```

```
## [1] -0.04045505
```

```
time_high<-tm_highe1[1]+tm_highe2[1]+tm_highATE1[1]
cat("Time for training gbm=", tm_highe1[1], "s \n")
```

```
## Time for training gbm= 0.421 s
```

```
cat("Time for getting propensity score=", tm_highe2[1], "s \n")
```

```
## Time for getting propensity score= 0.012 s
```

```
cat("Time for calculating ATE=", tm_highATE1[1], "s \n")
```

```
## Time for calculating ATE= 0.001 s
```

lowDim_dataset

```
# train-test split
n <- nrow(low)
n_train <- round(n*(4/5),0)
train_idx <- sample(1:n,n_train)
train_low <- low[train_idx,]
test_low <- low[-train_idx,]
```

Split treatment and control group, and complete regression for each group.

```
treatment.group.low<-low[low$A==1,-2]
control.group.low<-low[low$A==0,-2]

treatment.model.low<-lm(Y~.,data=treatment.group.low)
```

```
control.model.low<-lm(Y~.,data=control.group.low)
```

Estimate m1(X) and m0(X) for all entries.

```
X.low<-low[-c(1,2)]

low$m1<-predict(treatment.model.low,X.low)
low$m0<-predict(control.model.low,X.low)
```

Get propensity score for all entries using boosted stumps (Gradient Boosting Machine).

Using grid search to get proper parameters for gbm

```
# grid search
hyper_grid_low1 <- expand.grid(
  n.trees = c(40,50,60),
  shrinkage = c(.01, .05, .1),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(.65, .8, 1),
  optimal_trees = 0,               # a place to dump results
  min_RMSE = 0                     # a place to dump results
)

# randomize data
random_index <- sample(1:nrow(train_low), nrow(train_low))
random_ames_train <- train_low[random_index, ]

# grid search
for(i in 1:nrow(hyper_grid_low1)) {
  # reproducibility
  set.seed(2020)
  # train model
  gbm.tune <- gbm(
    formula = A~.,
    distribution = "bernoulli",
    data = train_low[-1],
    n.trees = hyper_grid_low1$n.trees[i],
    interaction.depth = 1,
    shrinkage = hyper_grid_low1$shrinkage[i],
    n.minobsinnode = hyper_grid_low1$n.minobsinnode[i],
    bag.fraction = hyper_grid_low1$bag.fraction[i],
    train.fraction = 0.75
  )

  # add min training error and trees to grid
  hyper_grid_low1$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid_low1$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
}

hyper_grid_low1 %>%
  dplyr::arrange(min_RMSE) %>%
  head(10)
```

Apply the parameters with min_RMSE (n.trees=60, shrinkage=0.1, n.minobsinnode=15, bag.fraction=0.8).

```r
set.seed(2020)

tm_lowe1 <- system.time(
boost.low <- gbm(A~., data = train_low[-1],
                 distribution = "bernoulli",
                 n.trees = 60, # the number of trees
                 shrinkage = 0.1, # learning rate
                 interaction.depth = 1, # total split
                 n.minobsinnode = 15,
                 bag.fraction = 0.8
            )
)
```

Calculate propensity scores for all entries in high.csv

```r
tm_lowe2 <- system.time(
low$e <- predict(boost.low, X.low, n.trees = 60, type = 'response')
)
```

Calculate each part in doubly robust estimation and count out the final result.

```r
tm_lowATE1 <- system.time(
{low$p1<-ifelse(low$A==1,(low$Y-low$m1)/low$e,0);
low$p2<-ifelse(low$A==0,(low$Y-low$m0)/(1-low$e),0);
low$result<-low$m1-low$m0+low$p1-low$p2;
ATE.low<-mean(low$result)}
)

ATE.low
```

```
## [1] 2.547012
```

```r
#alternative function, same result

#tm_lowATE2 <- system.time(
#ATE.low <- 1/n*(sum((low$A*low$Y-(low$A-low$e)*low$m1)/low$e)
#           -sum(((1-low$A)*low$Y+(low$A-low$e)*low$m0)/(1-low$e)))
#ATE.low
```

```r
# True ATE:
true_ATE_low <- 2.5

# Comparison:
true_ATE_low - ATE.low
```

```
## [1] -0.04701199
```

```r
time_low<-tm_lowe1[1]+tm_lowe2[1]+tm_lowATE1[1]
cat("Time for training gbm=", tm_lowe1[1], "s \n")
```

```
## Time for training gbm= 0.019 s
```

```r
cat("Time for getting propensity score=", tm_lowe2[1], "s \n")
```

```
## Time for getting propensity score= 0.002 s
```

```r
cat("Time for calculating ATE=", tm_lowATE1[1], "s \n")
```

```
## Time for calculating ATE= 0.001 s
```

Conclusion for Doubly Robust Estimation

ATE Estimation precision for HighDim dataset and LowDim dataset is pretty similar. Running time for HighDim dataset is around 43 times that for LowDim dataset.

```
table<-data.frame(ATE=c(round(ATE.high,3),round(ATE.low,3)), True.ATE=c(true_ATE_high,true_ATE_low),
                  diff=c(round(true_ATE_high - ATE.high,3),round(true_ATE_low - ATE.low,3)),
                  time=c(time_high,time_low),row.names = c('HighDim', 'LowDim'))
table
```

```
##            ATE True.ATE   diff  time
## HighDim -2.960     -3.0 -0.040 0.434
## LowDim   2.547      2.5 -0.047 0.022
```

Algo 2: 21 A6+P5 Regression Adjustment + boosted stumps

The ATEs we got from boosted stumps and regression adjustment are 2.5271 and -3.083. We conclude that it is a close estimate where the true ATEs that are 2.5 and -3.

Methodology and Implementation

1. Reference: D'Agostino RB Jr. Propensity score methods for bias reduction in the comparison of a treatment to a non-randomized control group. Stat Med. 1998 Oct 15;17(19):2265-81. doi: 10.1002/(sici)1097-0258(19981015)17:19<2265::aid-sim918>3.0.co;2-b. PMID: 9802183.
2. Boosted stumps:

- What is it: an ensemble of weak learners with boosting algorithms. We combine decision tree stumps (decision tree with depth of 1) to predict the propensity score of each sample to simulate a random sample in an observational setting.
- Implementation: boost_low = gbm(A~., data = train_low[-1], n.trees = 500, # the number of trees shrinkage = 0.03, # learning rate interaction.depth = 1, # depth of each tree, stumps cv.folds=5)

3. Regression adjustment:

- What is it: regress the outcome variable Y on treatment indicator variable A and the estimated propensity score(pred_high); We use the estimated coefficient on the A as indicator variable as an estimate of ATE
- Implementation: ATE_high = lm(Y~A+pred_high,data=high)

Data Preparation

```
library(gbm)
library(caret)
```

```
## Loading required package: lattice
```

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

```
high <- read.csv('../data/highDim_dataset.csv')
low <- read.csv('../data/lowDim_dataset.csv')
#high['A'] <- apply(high['A'],1,as.factor)
#low['A'] <- apply(low['A'],1,as.factor)
```

Low Dimension

```
# trani-test split
set.seed(2021)
```

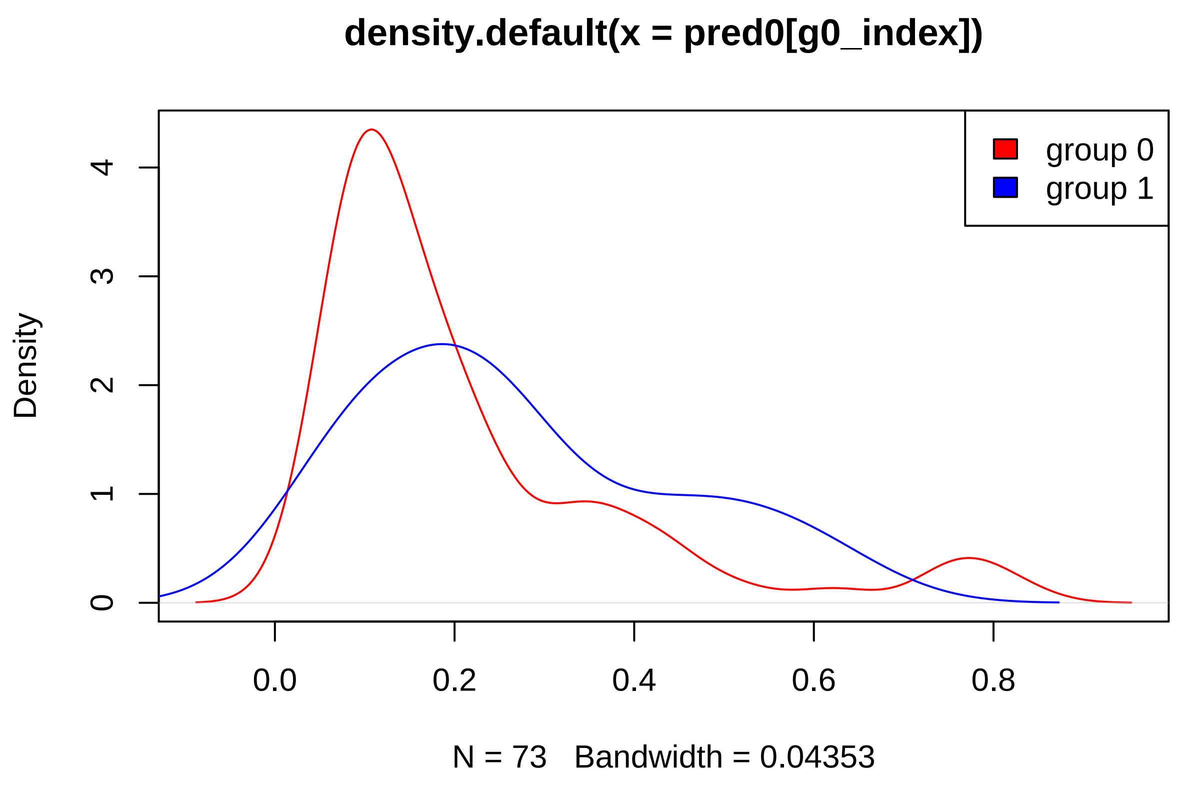
```r
n <- nrow(low)
n_train <- round(n*(4/5),0)
train_idx <- sample(1:n,n_train)
# test_idx <- setdiff(1:2000, train)
train_low <- low[train_idx,]
test_low <- low[-train_idx,]


## Propensity Score
start0 <- Sys.time()

boost0 = gbm(A~., data = train_low[-1],
            n.trees = 500, # the number of trees, 100, 1000. 10000, no big diff
            shrinkage = 0.03, # learning rate, 0.01, 0.03, 0.05, 0.1
            interaction.depth = 1 # depth of each tree, set 1 as stumps
            ) # here, the parameters we get are from grid search results - see the bottom of the file f
```

```
## Distribution not specified, assuming bernoulli ...
```

```r
# n.trees <- seq(from = 100, to = 10000, by = 100)
# n.trees set the number of trees to be built. Here I choose 1000 manually.
pred0 <- predict(boost0, test_low[-c(1,2)],n.trees = 1000, type = 'response')
```

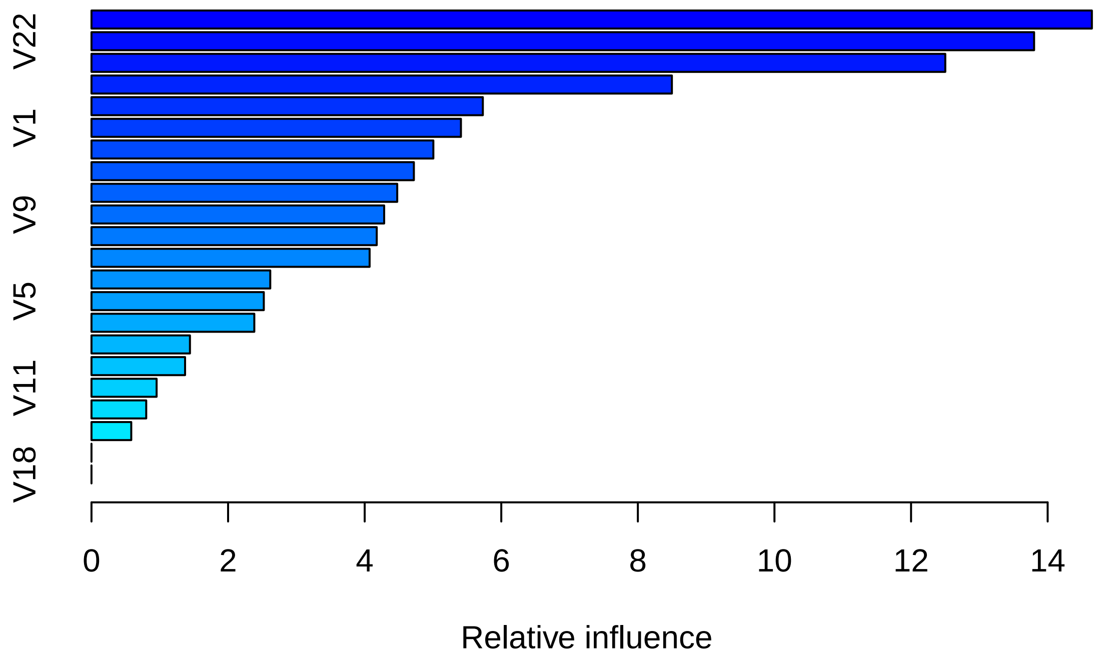
```
## Warning in predict.gbm(boost0, test_low[-c(1, 2)], n.trees = 1000, type =
## "response"): Number of trees not specified or exceeded number fit so far.
## Using 500.
```

```r
# plot by A to see the distribution of the predicted value
g0_index <- test_low$A == 0
g1_index <- test_low$A == 1
plot(density(pred0[g0_index]),col = 'red')
lines(density(pred0[g1_index]),col = 'blue')
legend('topright',legend = c('group 0','group 1'),fill = c('red','blue'))
```



```
## ATE

# build a regression model based on the propensity score
# structure the data frame
ps0 <- predict(boost0, low[-c(1,2)],n.trees = 100, type = 'response')
df0<-data.frame(low$Y,low$A, ps0)
colnames(df0) <- c('Y','A','PS')
model0<-lm(df0$Y~df0$A+df0$PS)

end0 <- Sys.time()
```

```
summary(boost0)
```



```
##     var    rel.inf
## V17 V17 14.6477218
## V22 V22 13.8015173
## V15 V15 12.5012633
## V8   V8  8.4984626
## V12 V12  5.7302940
## V1   V1  5.4095782
## V6   V6  5.0055893
## V3   V3  4.7203187
## V20 V20  4.4765478
## V9   V9  4.2850950
## V2   V2  4.1766691
## V14 V14  4.0724636
## V13 V13  2.6174864
## V5   V5  2.5232861
## V21 V21  2.3838630
## V7   V7  1.4418343
## V4   V4  1.3702618
## V11 V11  0.9535046
## V16 V16  0.8016433
## V19 V19  0.5826000
## V10 V10  0.0000000
## V18 V18  0.0000000
```

```r
summary(model0)
```

```
##
## Call:
## lm(formula = df0$Y ~ df0$A + df0$PS)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.9058  -2.2172  -0.5321   1.0805  27.7107
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.5946     0.3589  32.310  < 2e-16 ***
## df0$A         2.5271     0.4101   6.162 1.54e-09 ***
## df0$PS       22.3931     1.4479  15.466  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.495 on 472 degrees of freedom
## Multiple R-squared:  0.4672, Adjusted R-squared:  0.465
## F-statistic:   207 on 2 and 472 DF,  p-value: < 2.2e-16
```

```
cat('The total time using boosted stumps and regression adjustment with low dimension data is:', end0 -
```

```
## The total time using boosted stumps and regression adjustment with low dimension data is: 0.1375699
```

High Dimension

```
# train-test split
set.seed(2021)
n <- nrow(high)
n_train <- round(n*(4/5),0)
train_idx <- sample(1:n,n_train)
# test_idx <- setdiff(1:2000, train)
train_high <- high[train_idx,]
test_high <- high[-train_idx,]


## Propensity Score
start1 <- Sys.time()

boost1 = gbm(A~., data = train_high[-1],
             n.trees = 100, # the number of trees
             shrinkage = 0.001, # learning rate
             interaction.depth = 1 # stumps
             ) # here, the parameters we get are from grid search results - see the bottom of the file f
```

```
## Distribution not specified, assuming bernoulli ...
```

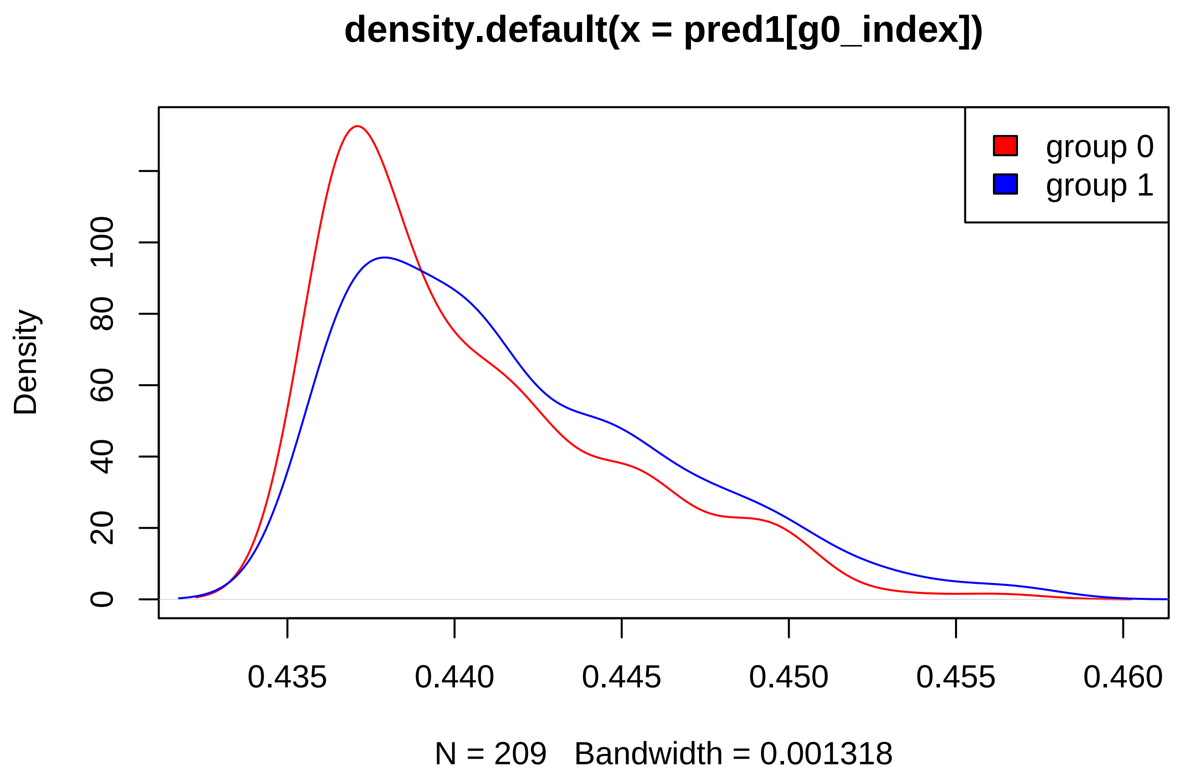
```
#n.trees <- seq(from = 100, to = 10000, by = 100)
# n.trees set the number of trees to be built. Here I choose 1000 manually.
pred1 <- predict(boost1, test_high[-c(1,2)],n.trees = 1000, type = 'response')
```

```
## Warning in predict.gbm(boost1, test_high[-c(1, 2)], n.trees = 1000, type =
## "response"): Number of trees not specified or exceeded number fit so far.
## Using 100.
```

```
length(pred1)
```

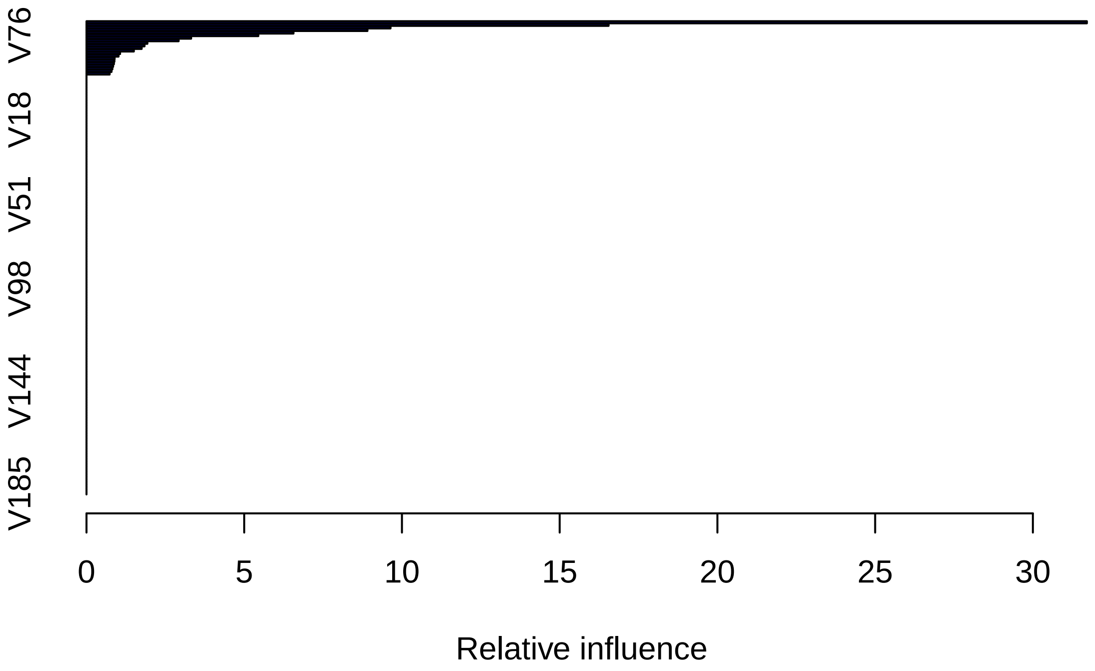
```
## [1] 400
```

```
# plot by A to see the distribution of the predicted value
g0_index <- test_high$A == 0
g1_index <- test_high$A == 1
plot(density(pred1[g0_index]),col = 'red')
lines(density(pred1[g1_index]),col = 'blue')
legend('topright',legend = c('group 0','group 1'),fill = c('red','blue'))
```



```r
## ATE

# build a regression model based on the propensity score
# structure the data frame
ps1 <- predict(boost1, high[-c(1,2)],n.trees = 100, type = 'response')
df1<-data.frame(high$Y,high$A, ps1)
colnames(df1) <- c('Y','A','PS')
model1<-lm(df1$Y~df1$A+df1$PS)

end1 <- Sys.time()
```

```r
summary(boost1)
```



```
##       var    rel.inf
## V95   V95 31.7118710
## V91   V91 16.5506737
## V83   V83  9.6377250
## V92   V92  8.9060494
## V65   V65  6.5617795
## V76   V76  5.4475166
## V63   V63  3.3178531
## V121 V121  2.9209869
## V68   V68  1.9309898
## V67   V67  1.8390144
## V134 V134  1.7500140
## V124 V124  1.5023956
## V73   V73  1.0652995
## V180 V180  1.0160785
## V99   V99  0.8926300
## V90   V90  0.8856194
## V94   V94  0.8699262
## V69   V69  0.8450043
## V131 V131  0.8214455
## V141 V141  0.7930416
## V89   V89  0.7340859
## V1     V1  0.0000000
## V2     V2  0.0000000
## V3     V3  0.0000000
## V4     V4  0.0000000
## V5     V5  0.0000000
## V6     V6  0.0000000
## V7     V7  0.0000000
## V8     V8  0.0000000
## V9     V9  0.0000000
## V10   V10  0.0000000
## V11   V11  0.0000000
## V12   V12  0.0000000
## V13   V13  0.0000000
## V14   V14  0.0000000
## V15   V15  0.0000000
## V16   V16  0.0000000
## V17   V17  0.0000000
## V18   V18  0.0000000
## V19   V19  0.0000000
## V20   V20  0.0000000
## V21   V21  0.0000000
## V22   V22  0.0000000
## V23   V23  0.0000000
## V24   V24  0.0000000
## V25   V25  0.0000000
## V26   V26  0.0000000
## V27   V27  0.0000000
## V28   V28  0.0000000
## V29   V29  0.0000000
## V30   V30  0.0000000
## V31   V31  0.0000000
## V32   V32  0.0000000
## V33   V33  0.0000000
## V34   V34  0.0000000
## V35   V35  0.0000000
## V36   V36  0.0000000
## V37   V37  0.0000000
## V38   V38  0.0000000
## V39   V39  0.0000000
## V40   V40  0.0000000
## V41   V41  0.0000000
## V42   V42  0.0000000
## V43   V43  0.0000000
## V44   V44  0.0000000
## V45   V45  0.0000000
## V46   V46  0.0000000
## V47   V47  0.0000000
## V48   V48  0.0000000
## V49   V49  0.0000000
## V50   V50  0.0000000
## V51   V51  0.0000000
## V52   V52  0.0000000
## V53   V53  0.0000000
## V54   V54  0.0000000
## V55   V55  0.0000000
## V56   V56  0.0000000
## V57   V57  0.0000000
## V58   V58  0.0000000
## V59   V59  0.0000000
## V60   V60  0.0000000
## V61   V61  0.0000000
## V62   V62  0.0000000
## V64   V64  0.0000000
## V66   V66  0.0000000
## V70   V70  0.0000000
## V71   V71  0.0000000
## V72   V72  0.0000000
## V74   V74  0.0000000
## V75   V75  0.0000000
## V77   V77  0.0000000
## V78   V78  0.0000000
## V79   V79  0.0000000
## V80   V80  0.0000000
## V81   V81  0.0000000
## V82   V82  0.0000000
## V84   V84  0.0000000
## V85   V85  0.0000000
## V86   V86  0.0000000
## V87   V87  0.0000000
## V88   V88  0.0000000
## V93   V93  0.0000000
## V96   V96  0.0000000
## V97   V97  0.0000000
## V98   V98  0.0000000
## V100 V100  0.0000000
## V101 V101  0.0000000
## V102 V102  0.0000000
## V103 V103  0.0000000
## V104 V104  0.0000000
## V105 V105  0.0000000
## V106 V106  0.0000000
## V107 V107  0.0000000
## V108 V108  0.0000000
## V109 V109  0.0000000
## V110 V110  0.0000000
## V111 V111  0.0000000
## V112 V112  0.0000000
## V113 V113  0.0000000
## V114 V114  0.0000000
## V115 V115  0.0000000
## V116 V116  0.0000000
## V117 V117  0.0000000
## V118 V118  0.0000000
## V119 V119  0.0000000
## V120 V120  0.0000000
## V122 V122  0.0000000
## V123 V123  0.0000000
## V125 V125  0.0000000
## V126 V126  0.0000000
## V127 V127  0.0000000
## V128 V128  0.0000000
## V129 V129  0.0000000
## V130 V130  0.0000000
## V132 V132  0.0000000
## V133 V133  0.0000000
## V135 V135  0.0000000
## V136 V136  0.0000000
## V137 V137  0.0000000
## V138 V138  0.0000000
## V139 V139  0.0000000
## V140 V140  0.0000000
## V142 V142  0.0000000
## V143 V143  0.0000000
## V144 V144  0.0000000
## V145 V145  0.0000000
## V146 V146  0.0000000
## V147 V147  0.0000000
## V148 V148  0.0000000
## V149 V149  0.0000000
## V150 V150  0.0000000
## V151 V151  0.0000000
## V152 V152  0.0000000
## V153 V153  0.0000000
## V154 V154  0.0000000
## V155 V155  0.0000000
## V156 V156  0.0000000
## V157 V157  0.0000000
## V158 V158  0.0000000
## V159 V159  0.0000000
## V160 V160  0.0000000
## V161 V161  0.0000000
## V162 V162  0.0000000
## V163 V163  0.0000000
## V164 V164  0.0000000
## V165 V165  0.0000000
## V166 V166  0.0000000
## V167 V167  0.0000000
## V168 V168  0.0000000
## V169 V169  0.0000000
## V170 V170  0.0000000
## V171 V171  0.0000000
## V172 V172  0.0000000
## V173 V173  0.0000000
## V174 V174  0.0000000
## V175 V175  0.0000000
## V176 V176  0.0000000
## V177 V177  0.0000000
## V178 V178  0.0000000
## V179 V179  0.0000000
## V181 V181  0.0000000
## V182 V182  0.0000000
## V183 V183  0.0000000
## V184 V184  0.0000000
## V185 V185  0.0000000
```

```r
summary(model1)
```

```
##
## Call:
## lm(formula = df1$Y ~ df1$A + df1$PS)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.4431  -2.6174   0.0114   2.3630  19.8397
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -521.3317     9.6763  -53.88   <2e-16 ***
## df1$A         -3.0830     0.1987  -15.52   <2e-16 ***
## df1$PS      1157.6456    21.9805   52.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.296 on 1997 degrees of freedom
## Multiple R-squared:  0.5824, Adjusted R-squared:  0.5819
## F-statistic:  1392 on 2 and 1997 DF,  p-value: < 2.2e-16
```

```
cat('The total time using boosted stumps and regression adjustment with high dimension data is:', end1
```

```
## The total time using boosted stumps and regression adjustment with high dimension data is: 0.673718
```

The summary of the model gives a feature importance plot. Conduct prediction on the test set so we can have Test Error as an evaluation. The density plot shows the overlap of propensity score between the two groups.

Additional Code: Grid Search

In this part, we included our code for conducting grid search for lowdim. The procedure is similar in highdim. For the readability of the file, we choose to comment out the code.

```
# grid search
# hyper_grid_low1 <- expand.grid(
#  shrinkage = c(.01, 0.03, 0.05),
#  interaction.depth = 1 - since it is boosted stumps
#  n.minobsinnode = c(5, 10, 15),
#  bag.fraction = c(.65, .8, 1),
#  optimal_trees = 0,               # a place to dump results
#  min_RMSE = 0                     # a place to dump results
#)

# randomize data
# random_index <- sample(1:nrow(train_low), nrow(train_low))
# random_ames_train <- train_low[random_index, ]

# grid search
# for(i in 1:nrow(hyper_grid_low1)) {
  # reproducibility
#  set.seed(2020)
  # train model
#  gbm.tune <- gbm(
#    formula = A~.,
#    data = train_low[-1],
#    n.trees = 500,
#    interaction.depth = hyper_grid_low1$interaction.depth[i],
#    shrinkage = hyper_grid_low1$shrinkage[i],
#    n.minobsinnode = hyper_grid_low1$n.minobsinnode[i],
#  bag.fraction = hyper_grid_low1$bag.fraction[i],
#   train.fraction = .75,
#    n.cores = NULL, # will use all cores by default
#    verbose = FALSE
#  )

  # add min training error and trees to grid
#  hyper_grid_low1$optimal_trees[i] <- which.min(gbm.tune$valid.error)
#  hyper_grid_low1$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
# }

# hyper_grid_low1 %>%
# dplyr::arrange(min_RMSE) %>%
#  head(10)
```

Algo 3: 15 A4 Regression Estimate

Understanding

Regression Estimate is a really simple estimation model to the calculate ATE, which do not require Propensity Scores calculation. This makes it a straight forward model and a computational efficient model. By implementing the linear regression on treated groups and untreated groups, we could regress on different groups to get the two different sets of paramaters and then by predicting the models on the whole dataset, substracting the prediction we can get the difference between the two regression models. In the end, we can calculate the ATE(Average Treatment Effect) by taking the average of the difference.

$$ATE = N^{-1}\sum_{i=1}^{N}(\hat{m_1}(X_i) - \hat{m_0}(X_i))$$

Denote that

N is the number of samples in the dataset,

X_i is the datapoint in the dataset,

m_1 is the regression model learned from the treated groups,

m_0 is the regression model learned from the untreated groups,

$\hat{m_1}(X_i)$ is the prediction of the regression model m_1 on the datapoint X_i,

$\hat{m_0}(X_i)$ is the prediction of the regression model m_0 on the datapoint X_i.

Implementation

Read the data and split the data into two groups – Treated Group and Untreated Group

```
high_data <-read.csv('../data/highDim_dataset.csv')
low_data <-read.csv('../data/lowDim_dataset.csv')

N_high <- dim(high_data)[1]
N_low <- dim(low_data)[1]

high_data_X <- high_data[,3:dim(high_data)[2]]
low_data_X <- low_data[,3:dim(low_data)[2]]

high_treated <- high_data[high_data$A==1,-2]
high_untreated <- high_data[high_data$A==0,-2]
```

```r
N_high_treated <- dim(high_treated)[1]
N_high_untreated <- dim(high_untreated)[1]

low_treated <- low_data[low_data$A==1,-2]
low_untreated <- low_data[low_data$A==0,-2]

N_low_treated <- dim(low_treated)[1]
N_low_untreated <- dim(low_untreated)[1]
```

Train the data and record the training time of two datasets

```r
time<- system.time({
  high_treated_lm <- lm(Y~.,data = high_treated);
  high_untreated_lm <- lm(Y~.,data = high_untreated);
  high_treated_predict_all <- predict(high_treated_lm,newdata = high_data_X);
  high_untreated_predict_all <- predict(high_untreated_lm,newdata = high_data_X)})
train_time_high <- time[1]
train_time_high
```

```
## user.self
##     0.133
```

```r
time<- system.time({
  low_treated_lm <- lm(Y~.,data = low_treated);
  low_untreated_lm <- lm(Y~.,data = low_untreated);
  low_treated_predict_all <- predict(low_treated_lm,newdata = low_data_X);
  low_untreated_predict_all <- predict(low_untreated_lm,newdata = low_data_X)})
train_time_low <- time[1]
train_time_low
```

```
## user.self
##     0.008
```

Calculate the ATE

```r
reg_est_ATE_high<-sum(high_treated_predict_all - high_untreated_predict_all)/N_high
reg_est_ATE_low<-sum(low_treated_predict_all - low_untreated_predict_all)/N_low
reg_est_ATE_high
```

```
## [1] -2.95978
```

```r
reg_est_ATE_low
```

```
## [1] 2.526944
```

Compare the ATE with the true ATE

```r
# True ATE:
true_ATE_high <- -3
true_ATE_low <- 2.5

# Comparison:
abs(true_ATE_high - reg_est_ATE_high) /abs(true_ATE_high)
```

```
## [1] 0.01340679
```

```r
abs(true_ATE_low - reg_est_ATE_low) /abs(true_ATE_low)
```

```
## [1] 0.01077759
```

Conclustions

Comparision between the two dataset

We can conclude that the model is more fit to the low dimension dataset. With higher dimension, the ATE has higher bias rate(1.34% vs 1.08%).

Comparision among the three models

The table shows the result of the three algorithm's ATE in the two different datasets.

Algorithm	High ATE	Low ATE	High Train Time	Low Train Time
True ATE	-3	2.5	-	-
Doubly Robust Estimation + Boosted Stumps	-2.9626	2.5187	1.2180	0.0230
Regression Estimate	-2.9598	2.5269	0.2270	0.0190
Regression Adjustment + Boosted Stumps	-3.0830	2.5271	0.5060	0.1287

From the table above, we can clearly conclude that the Regression Estimate's accuracy is relatively high, but slightly lower than the Doubly Robust Estimation + Boosted Stumps model. However, the training time of Doubly Robust Estimation + Boosted Stumps model is higher than the Regression Estimate model for both high dimension dataset and low dimension dataset. We can conclude that the Regression Estimate is more computational efficient but slightly less accuracy than the Doubly Robust Estimation + Boosted Stumps model.