Testing Report

Group 5: 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

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

Methodology and Implementation

1. Reference: [1] Chan D , Ge R , Gershony O , et al. Evaluating online ad campaigns in a pipeline: causal models at scale[C]// Acm Sigkdd International Conference on Knowledge Discovery & Data Mining. ACM, 2010. [2] Lunceford, Jared K . Stratification and weighting via the propensity score in estimation of causal treatment effects: a comparative study[J]. Statistics in Medicine, 2017.
2. Doubly Robust Estimation

- What is it: It is a method to estimate treatment effect. Doubly robust estimator has the smallest asymptotic variance. "Doubly robust" in the sense that the estimator remains consistent if either (i) if the propensity score model is correctly specified but the two regression models m0 and m1 are not or (ii) the two regression models are correctly specified but the propensity score model is not, although under these conditions it might not be the most efficient.
- Implementation:

$$\begin{aligned} ATE =& E[E(Y|T=1,X) \\ & - E(Y|T=0,X)] \\ & + E[(\frac{I[T=1]}{propensity\,score} - \frac{I[T=0]}{(1-propensity\,score)})(Y - E(Y|T,X))] \end{aligned}$$

  in which $E(Y|T=t,X)$ is usually obtained by regressing the observed response Y on X in group t (where t = 0,1).

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),
  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).

```
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
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] -2.961573
```

```
#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] -0.03842743
```

```
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= 0.248 s
```

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

```
## Time for getting propensity score= 0.009 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).

```
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
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.
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
```

```
#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
```

```
# True ATE:
true_ATE_low <- 2.5
# Comparison:
true_ATE_low - ATE.low
```

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

```
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= 0.014 s
```

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

```
## Time for getting propensity score= 0.001 s
```

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

```
## Time for calculating ATE= 0 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 -2.962     -3.0 -0.038 0.258
## LowDim   2.547      2.5 -0.047 0.015
```

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

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)
```

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 -
```

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

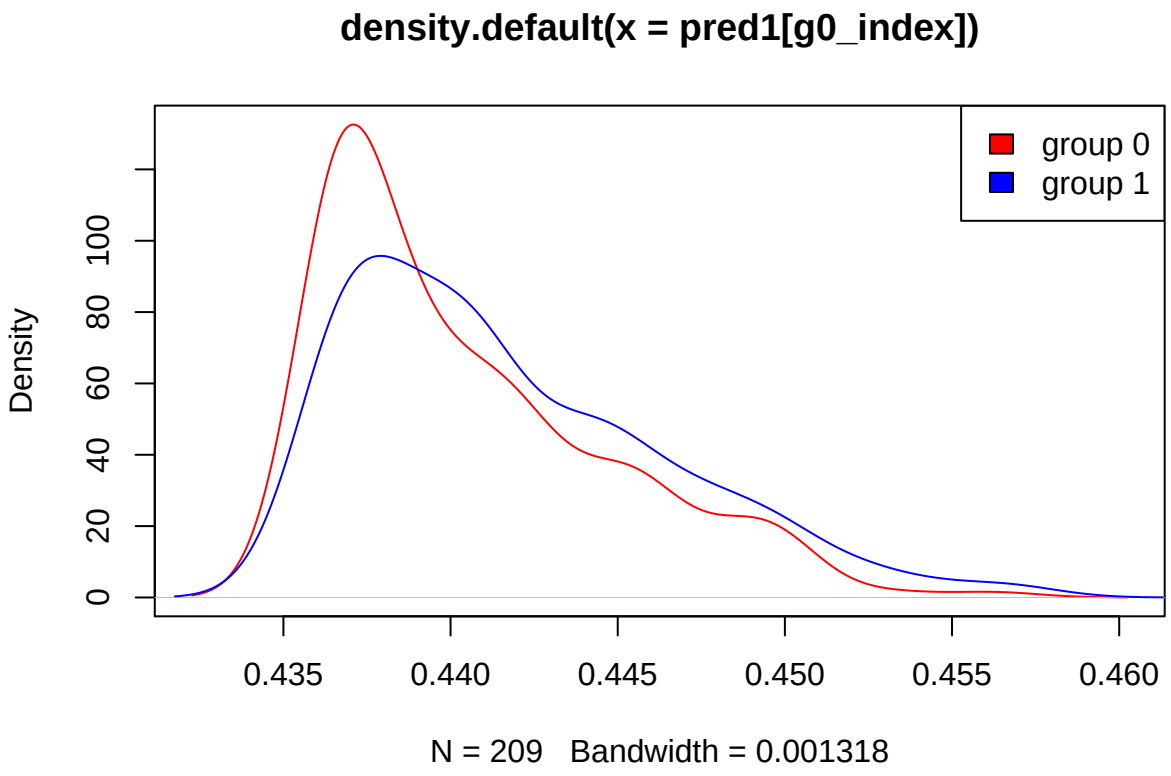
```
               #see the bottom of the file for detail

#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'))
```




```
## 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()
```

```
#summary(boost1)
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 - start1,'s')
```

```
## The total time using boosted stumps and regression adjustment with high dimension
##     data is: 0.479815 s
```

Low Dimension

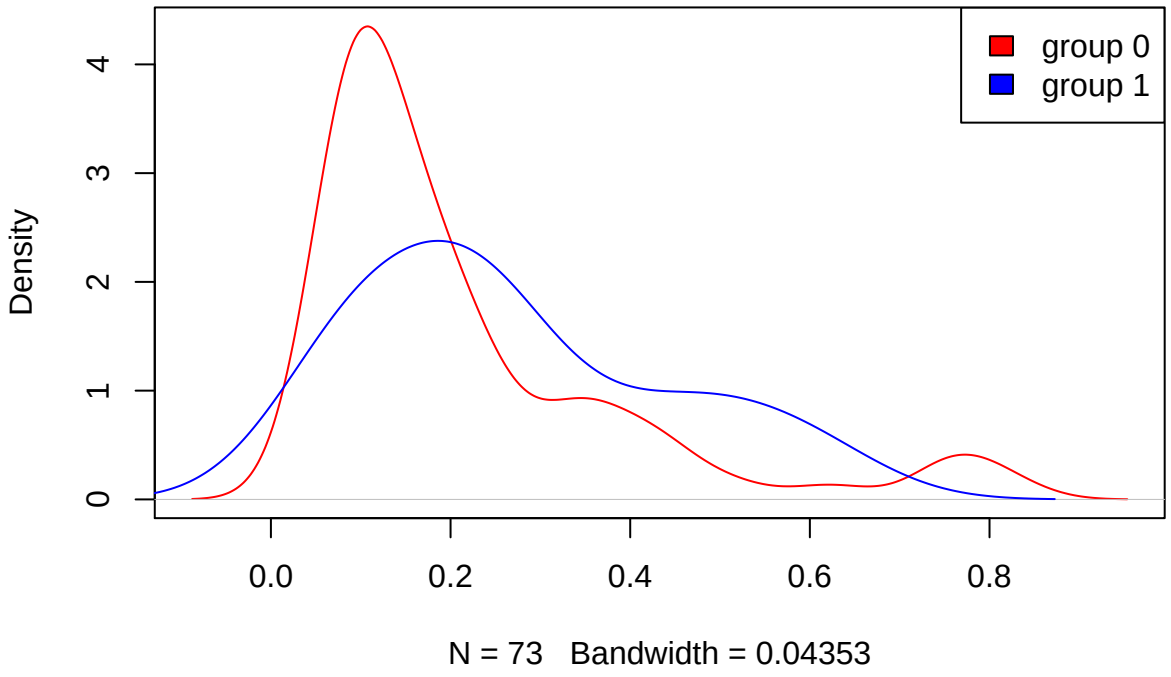
```
# trani-test split
set.seed(2021)
n <- nrow(low)
n_train <- round(n*(4/5),0)
train_idx <- sample(1:n,n_train)
# test_idx <- setdiff(1:2000, train)
train_low <- low[train_idx,]
test_low <- low[-train_idx,]


## 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 -
```

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

```
               #see the bottom of the file for detail

# 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.
```

```
# 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)
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 - start0,'s')
```

```
## The total time using boosted stumps and regression adjustment with low dimension
##     data is: 0.08484101 s
```

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)
```

Conclusion

```
data.frame(ATE=c(round(model1$coefficients[2],4),
                 round(model0$coefficients[2],3)), True.ATE=c(true_ATE_high,true_ATE_low),
                  diff=c(round(true_ATE_high - model1$coefficients[2],3),
                         round(true_ATE_low - model0$coefficients[2],3)),
                  time=c(end1-start1,end0-start0),row.names = c('HighDim', 'LowDim'))
```

```
##            ATE True.ATE   diff            time
## HighDim -3.083     -3.0  0.083 0.47981501 secs
## LowDim   2.527      2.5 -0.027 0.08484101 secs
```

The density plots show the matching of propensity scores between the two groups. The boosted stumps performs well on both of the datasets as they show the density plots between two groups overlap each other on a large scale. For the parameters of the regression model, higher dimension data requires smaller learning rate and less number of trees. I would suggest to use the combination on higher dimension datasets, as the boosting stumps identifying variable relative influence effectively and produces well-matched propensity scores between the experiment group and the control group.

Algo 3: 15 A4 Regression Estimate

Methodology

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]
```

```r
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]

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

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

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

# Calculate the ATE

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
#reg_est_ATE_low
```

Conclustions

```r
data.frame(ATE=c(round(reg_est_ATE_high,4),
                 round(reg_est_ATE_low,4)), True.ATE=c(true_ATE_high,true_ATE_low),
                  diff=c(round(true_ATE_high - model1$coefficients[2],3),
                         round(true_ATE_low - model0$coefficients[2],3)),
                  time=c(train_time_high,train_time_low),row.names = c('HighDim', 'LowDim'))
```

```
##             ATE True.ATE   diff  time
## HighDim -2.9598     -3.0  0.083 0.104
## LowDim   2.5269      2.5 -0.027 0.008
```

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%).

Summary

Comparision among the three models

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

Algorithm	**ATE High**	difference	**ATE Low**	difference
True ATE	-3	-	2.5	-
Doubly Robust Estimation + Boosted Stumps	-2.9626	2.5187	0.0374	0.0187
Regression Adjustment + Boosted Stumps	-3.0830	2.5271	0.0830	0.0271
Regression Estimate	-2.9598	2.5269	0.0402	0.0269

All of the models are well-performed after tuning. From the table above, we can see that the Doubly Robust Estimation + Boosted Stumps model has the smallest difference with the true ATE on both of the datasets. Results of Regression Estimate and Regression adjustment are close on the high dimension datasets while the former model outperformed on low dimension.

Algorithm	Training Time High	Training Time Low
Doubly Robust Estimation + Boosted Stumps	1.2180	0.0230
Regression Estimate	0.2270	0.0190
Regression Adjustment + Boosted Stumps	0.5060	0.1287

Comparing the training time on each of the methods and the datasets, the regression estimate has the shortest training time on both of the datasets. Regression adjustment + boosted stumps have a shorter training time than doubly robust estimation on the high dimension dataset, but six times longer on the low dimension dataset. We conclude that the regression estimate is more computationally efficient than the Doubly Robust Estimation + Boosted Stumps model.

For model flexibility, we would recommend Doubly Robust Estimation + Boosted Stumps, as it can be customized with grid search and achieve higher accuracy for high and low dimension data. For efficiency, the Regression Estimate would give an informative idea of the ATE between the experiment and the control group in a productive manner.