Project 4: Causal Inference Algorithms Evaluation

Group 4: Zhenglei Chen, Jaival Desai, Qinzhe Hu, Levi Lee, Luyao Sun, Xinyi Wei

12/02/2020

Setup

First, we set working directories, install required libraries and import the data.

```
setwd("~/GitHub/Fall2020-Project4-group-4/doc")
```

```
packages.used <- c("dplyr", "ggplot2", "WeightedROC", "rpart", "rpart.plot")

# check packages that need to be installed.
packages.needed <- setdiff(packages.used, intersect(installed.packages()[,1], packages.used))

# install additional packages
if(length(packages.needed) > 0){
  install.packages(packages.needed, dependencies = TRUE)
}

library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.3

##
## 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
```

```
library(ggplot2)
library(WeightedROC)
```

```
## Warning: package 'WeightedROC' was built under R version 4.0.3
```

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.0.3
```

```r
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.0.3
```

```r
df_high <- read.csv("../data/highDim_dataset.csv")
df_low <- read.csv("../data/lowDim_dataset.csv")
```

Introduction

Description

About the Data

Description

Background: Trees

Description

Cross-Validation

Description

Step 1: Set Controls and Establish Hyperparameters

Description

```r
K <- 5  # number of CV folds
sample.reweight <- TRUE # run sample reweighting in model training

# setting the following to false loads data generated from a previous run
# this data is the same in each run due to a set seed

run.cv.trees_high <- FALSE # run cross-validation on the training set for trees on high dim data

run.cv.trees_low <- FALSE # run cross-validation on the training set for trees on low dim data
```

Description

```r
# hyperparameters for trees
hyper_grid_trees <- expand.grid(
  cp = c(2^(0), 2^(-1), 2^(-2), 2^(-3), 2^(-4),
         2^(-5), 2^(-6), 2^(-7), 2^(-8), 2^(-9),
         2^(-10), 2^(-11), 2^(-12), 2^(-13), 2^(-14),
         2^(-15), 2^(-16), 2^(-17), 0, -2^(0))
)
```

Step 2: Cross-Validate the Hyperparameters

Description

Description

```r
# features are the predictors: V1 - Vp
# column 1 is the response Y
# column 2 is the treatment A

feature_train_high = df_high[, -1:-2]
label_train_high = df_high[, 2]

feature_train_low = df_low[, -1:-2]
label_train_low = df_low[, 2]
```

High Dimensional Data *Description*

```r
set.seed(5243)

if(run.cv.trees_high){
  res_cv_trees_high <- matrix(0, nrow = nrow(hyper_grid_trees), ncol = 4)
  for(i in 1:nrow(hyper_grid_trees)){
    cat("complexity = ", hyper_grid_trees$cp[i], "\n", sep = "")
    res_cv_trees_high[i,] <- cv.function(features = feature_train_high, labels = label_train_high,
                                         cp = hyper_grid_trees$cp[i],
                                         K, reweight = sample.reweight)
  save(res_cv_trees_high, file = "../output/res_cv_trees_high.RData")
  }
}else{
  load("../output/res_cv_trees_high.RData")
}
```

Low Dimensional Data *Description*

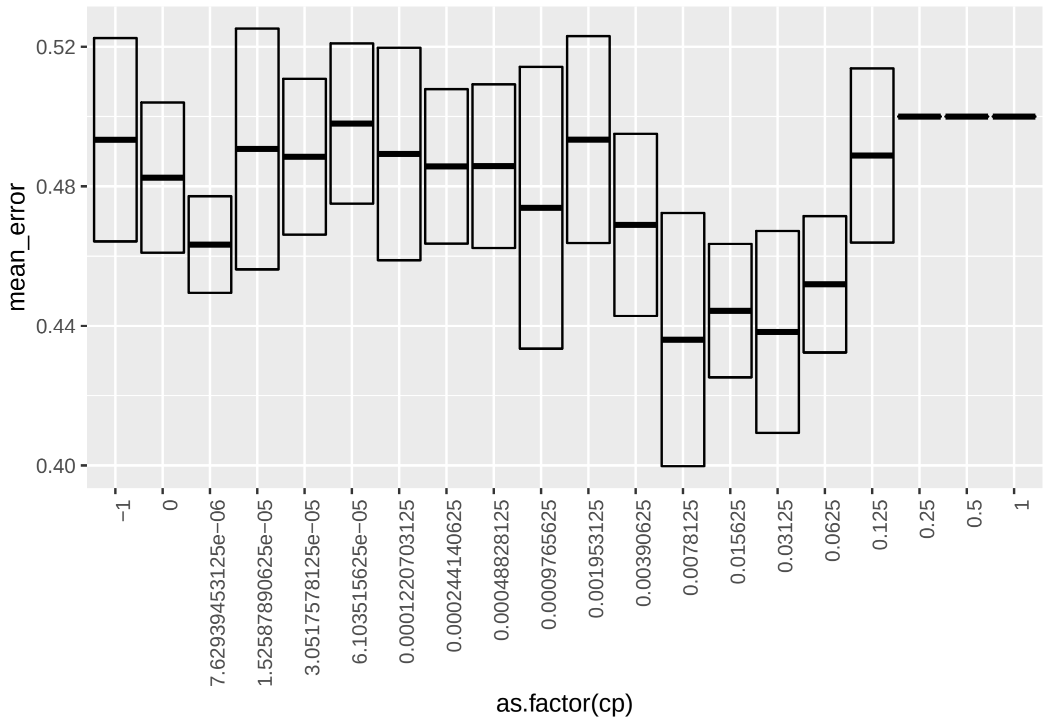
```r
set.seed(5243)

if(run.cv.trees_low){
  res_cv_trees_low <- matrix(0, nrow = nrow(hyper_grid_trees), ncol = 4)
  for(i in 1:nrow(hyper_grid_trees)){
    cat("complexity = ", hyper_grid_trees$cp[i], "\n", sep = "")
    res_cv_trees_low[i,] <- cv.function(features = feature_train_low, labels = label_train_low,
                                        cp = hyper_grid_trees$cp[i],
                                        K, reweight = sample.reweight)
  save(res_cv_trees_low, file="../output/res_cv_trees_low.RData")
  }
}else{
  load("../output/res_cv_trees_low.RData")
}
```

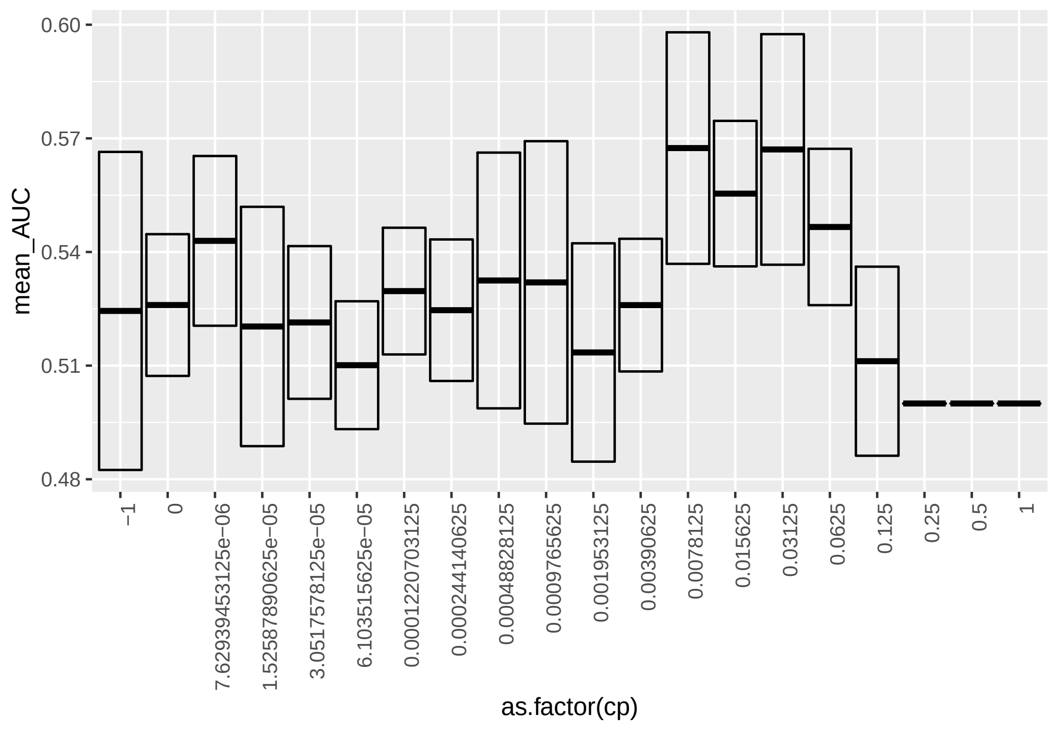
Step 3: Visualize CV Error and AUC

Description

```r
# create data frame to organize results
res_cv_trees_high <- as.data.frame(res_cv_trees_high)
colnames(res_cv_trees_high) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")
cv_results_trees_high = data.frame(hyper_grid_trees, res_cv_trees_high)

# look at top 5 models with highest AUC
cv_results_trees_high[order(cv_results_trees_high$mean_AUC, decreasing = TRUE), ][1:5, ]
```

```
##               cp mean_error   sd_error  mean_AUC     sd_AUC
## 8  7.812500e-03  0.4360668 0.03628091 0.5674343 0.03056252
## 6  3.125000e-02  0.4382665 0.02892494 0.5670732 0.03043522
## 7  1.562500e-02  0.4443627 0.01912652 0.5554088 0.01920727
## 5  6.250000e-02  0.4519064 0.01953579 0.5466031 0.02064263
## 18 7.629395e-06  0.4633120 0.01383644 0.5429351 0.02240466
```





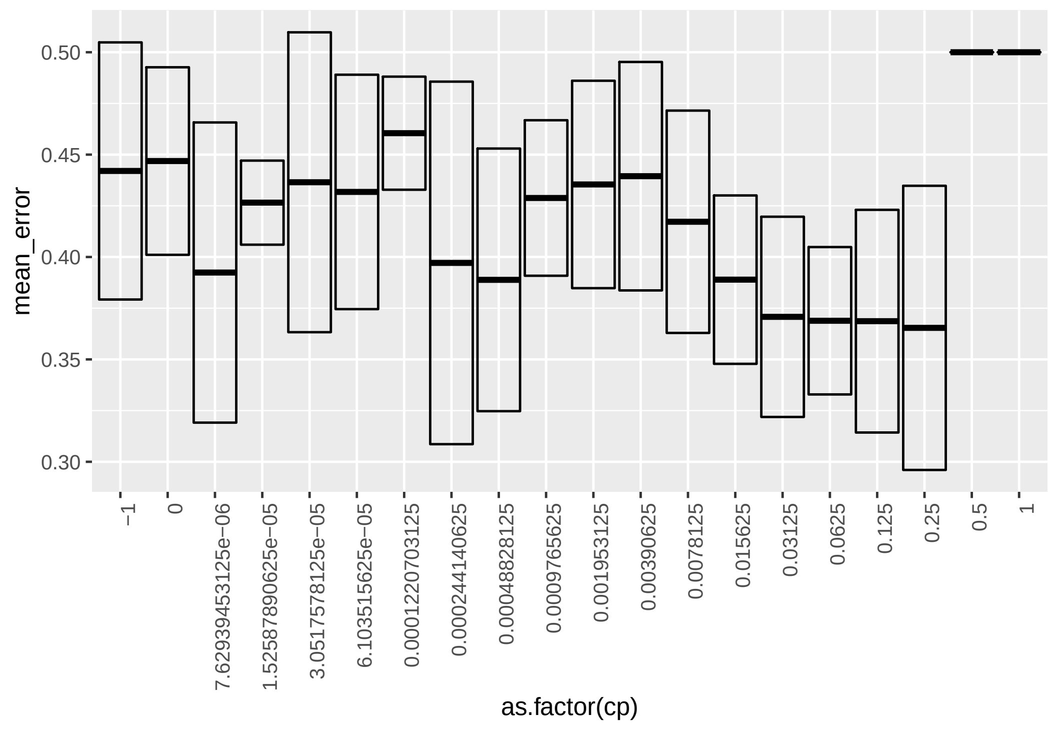
```
best_cp_high <- cv_results_trees_high$cp[cv_results_trees_high$mean_AUC ==
                                           max(cv_results_trees_high$mean_AUC)]

best_cp_high
```

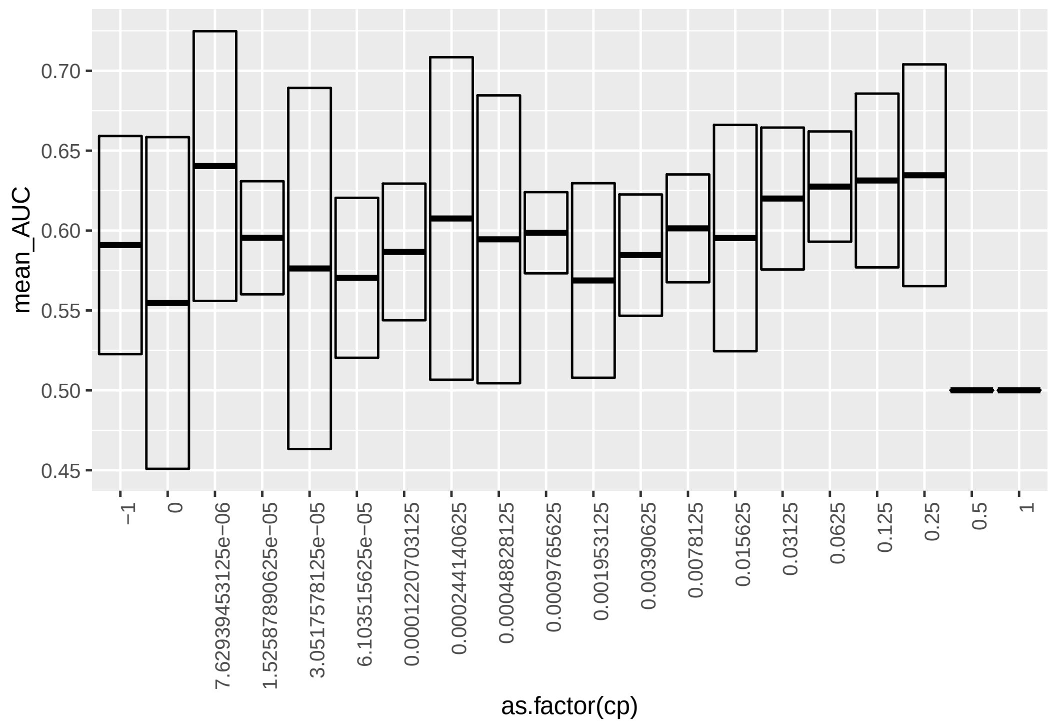
```
## [1] 0.0078125
```

Low Dimensional Data *Description*

```
# create data frame to organize results
res_cv_trees_low <- as.data.frame(res_cv_trees_low)
colnames(res_cv_trees_low) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")
cv_results_trees_low = data.frame(hyper_grid_trees, res_cv_trees_low)

# look at top 5 models with lowest AUC
cv_results_trees_low[order(cv_results_trees_low$mean_AUC, decreasing = TRUE), ][1:5, ]
```

```
##              cp mean_error   sd_error  mean_AUC     sd_AUC
## 18 7.629395e-06  0.3924126 0.07329778 0.6403787 0.08438962
## 3  2.500000e-01  0.3653965 0.06938198 0.6346035 0.06938198
## 4  1.250000e-01  0.3686625 0.05437925 0.6313375 0.05437925
## 5  6.250000e-02  0.3688654 0.03598451 0.6275346 0.03449488
## 6  3.125000e-02  0.3707763 0.04889355 0.6200507 0.04439815
```





```
best_cp_low <- cv_results_trees_low$cp[cv_results_trees_low$mean_AUC ==
                                          max(cv_results_trees_low$mean_AUC)]

best_cp_low
```

```
## [1] 7.629395e-06
```

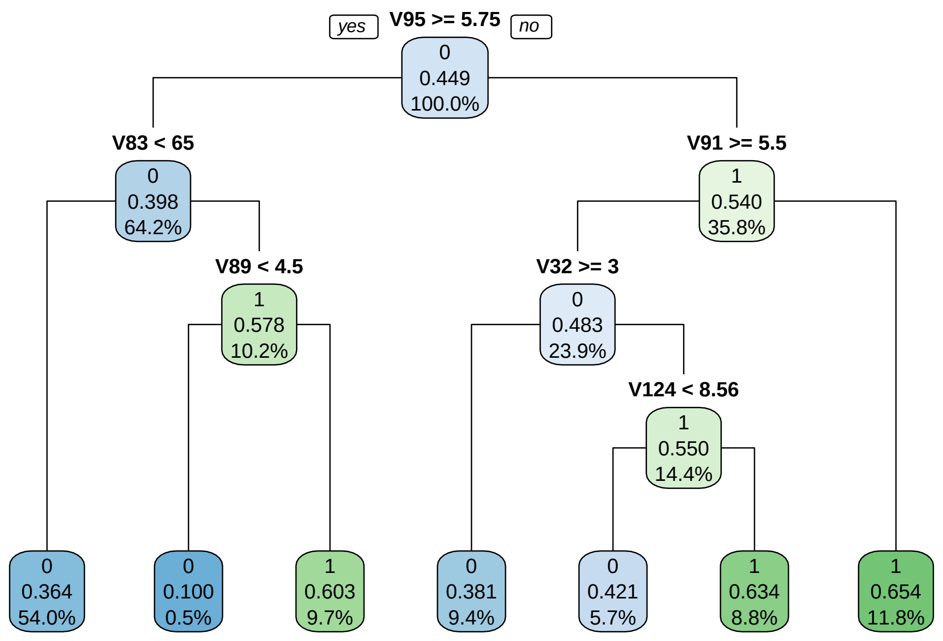
Propensity Score Estimation

Description

```
# imbalanced dataset requires weights
# to be used in the trained model

weights_high <- rep(NA, length(df_high$A))
for (v in unique(df_high$A)){
  weights_high[df_high$A == v] = 0.5 * length(df_high$A) / length(df_high$A[df_high$A == v])
}


weights_low <- rep(NA, length(df_low$A))
for (v in unique(df_low$A)){
  weights_low[df_low$A == v] = 0.5 * length(df_low$A) / length(df_low$A[df_low$A == v])
}
```

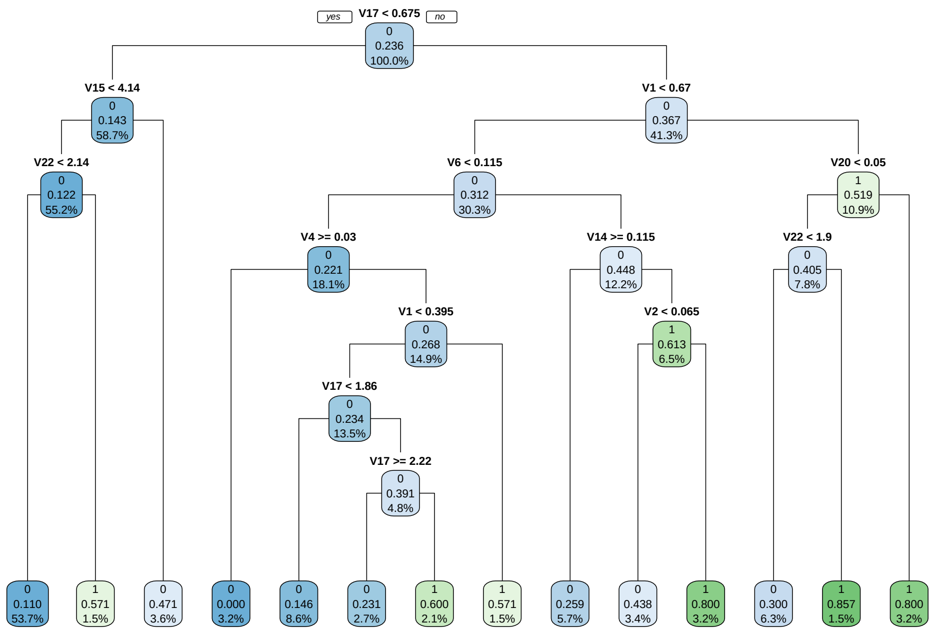
```


############### SET START TIME HERE

# create tree model for high dimensional data with best cp parameter
tree_high <- rpart(A ~ . - Y, method = "class", data = df_high, cp = best_cp_high)

# calculate propensity scores
prop_score_high <- predict(tree_high, newdata = df_high[, -2], type = "prob")[, 2]

############### SET END TIME HERE
```

High Dimensional Data



```
############### SET START TIME HERE

# create tree model for low dimensional data with best cp parameter
tree_low <- rpart(A ~ . - Y, method = "class", data = df_low, cp = best_cp_low)

# calculate propensity scores
prop_score_low <- predict(tree_low, newdata = df_low[, -2], type = "prob")[, 2]

############### SET END TIME HERE
```

Low Dimensional Data



ATE Estimation

Description

Stratification

Description

```
K = 5
quintiles <- seq(0, 1, by = 1/K)
```

High Dimensional Data *Description*

```
############### SET START TIME HERE

df_high <- cbind(df_high, prop_score_high)
quintile_values_high <- rep(NA, length(quintiles))

for (i in 1:length(quintiles)){
  quintile_values_high[i] <- quantile(prop_score_high, quintiles[i])
}

# values of quintiles for high data
quintile_values_high
```

```
## [1] 0.1000000 0.3635523 0.3635523 0.3809524 0.6342857 0.6540084
```

```
df_high$quintile_class_high <- rep(NA, nrow(df_high))

# assign quintile class to each observation
for (i in 1:nrow(df_high)){
  if ((quintile_values_high[1] <= df_high$prop_score_high[i]) &
      (df_high$prop_score_high[i] < quintile_values_high[2])) {
    df_high$quintile_class_high[i] <- 1
  } else if ((quintile_values_high[2] <= df_high$prop_score_high[i]) &
             (df_high$prop_score_high[i] < quintile_values_high[3])) {
    df_high$quintile_class_high[i] <- 2
  } else if ((quintile_values_high[3] <= df_high$prop_score_high[i]) &
             (df_high$prop_score_high[i] < quintile_values_high[4])) {
    df_high$quintile_class_high[i] <- 3
  } else if ((quintile_values_high[4] <= df_high$prop_score_high[i]) &
             (df_high$prop_score_high[i] < quintile_values_high[5])) {
    df_high$quintile_class_high[i] <- 4
  } else if ((quintile_values_high[5] <= df_high$prop_score_high[i]) &
             (df_high$prop_score_high[i] <= quintile_values_high[6])) {
    df_high$quintile_class_high[i] <- 5
  }
}

summary_high = expand.grid(
  A = c(0, 1),
  quintile = c(1, 2, 3, 4, 5),
  n = NA,
  prop = NA,
  avg_y = NA
)

for (i in 1:nrow(summary_high)) {
  subset <- df_high[(df_high$A == summary_high$A[i]) & (df_high$quintile_class_high == summary_high$quin
  summary_high$n[i] = nrow(subset)
  summary_high$prop[i] = summary_high$n[i]/nrow(df_high)
  summary_high$avg_y[i] = mean(subset$Y)
}


for (i in 1:nrow(summary_high)) {
  if (is.nan(summary_high$avg_y[i]) == TRUE) {
    summary_high$avg_y[i] <- 0
  }
}

# this table records the mean response in each quintile; needed for stratification
summary_high
```

```
##   A quintile   n   prop      avg_y
## 1 0        1   9 0.0045  -1.556754
## 2 1        1   1 0.0005   3.448809
## 3 0        2   0 0.0000   0.000000
## 4 1        2   0 0.0000   0.000000
## 5 0        3 688 0.3440 -13.637227
## 6  1        3 393 0.1965 -16.140803
## 7  0        4 260 0.1300 -10.267325
## 8  1        4 237 0.1185 -10.349496
## 9  0        5 146 0.0730  -6.152075
## 10 1        5 266 0.1330  -8.714239
```

```
quntile_prop_high <- summary_high %>% group_by(quintile) %>% summarise(sum = sum(n)/nrow(df_high))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# this table records the proportions for each quintile; also needed for stratification
quntile_prop_high
```

```
## # A tibble: 5 x 2
##   quintile   sum
##      <dbl> <dbl>
## 1        1 0.005
## 2        2 0
## 3        3 0.540
## 4        4 0.248
## 5        5 0.206
```

```
ATE_stratification_high = quntile_prop_high$sum[1]*(summary_high$avg_y[2] - summary_high$avg_y[1]) +
  quntile_prop_high$sum[2]*(summary_high$avg_y[4] - summary_high$avg_y[3]) +
  quntile_prop_high$sum[3]*(summary_high$avg_y[6] - summary_high$avg_y[5]) +
  quntile_prop_high$sum[4]*(summary_high$avg_y[8] - summary_high$avg_y[7]) +
  quntile_prop_high$sum[5]*(summary_high$avg_y[10] - summary_high$avg_y[9])


ATE_stratification_high
```

```
## [1] -1.87638
```

```
############### SET END TIME HERE
```

Low Dimensional Data *Description*

```
############### SET START TIME HERE

df_low <- cbind(df_low, prop_score_low)
quintile_values_low <- rep(NA, length(quintiles))

for (i in 1:length(quintiles)){
  quintile_values_low[i] <- quantile(prop_score_low, quintiles[i])
}

# values of quintiles for low data
quintile_values_low
```

```
## [1] 0.0000000 0.1098039 0.1098039 0.1463415 0.3000000 0.8571429
```

```r
df_low$quintile_class_low <- rep(NA, nrow(df_low))

# assign quintile class to each observation
for (i in 1:nrow(df_low)){
  if ((quintile_values_low[1] <= df_low$prop_score_low[i]) &
      (df_low$prop_score_low[i] < quintile_values_low[2])) {
    df_low$quintile_class_low[i] <- 1
  } else if ((quintile_values_low[2] <= df_low$prop_score_low[i]) &
             (df_low$prop_score_low[i] < quintile_values_low[3])) {
    df_low$quintile_class_low[i] <- 2
  } else if ((quintile_values_low[3] <= df_low$prop_score_low[i]) &
             (df_low$prop_score_low[i] < quintile_values_low[4])) {
    df_low$quintile_class_low[i] <- 3
  } else if ((quintile_values_low[4] <= df_low$prop_score_low[i]) &
             (df_low$prop_score_low[i] < quintile_values_low[5])) {
    df_low$quintile_class_low[i] <- 4
  } else if ((quintile_values_low[5] <= df_low$prop_score_low[i]) &
             (df_low$prop_score_low[i] <= quintile_values_low[6])) {
    df_low$quintile_class_low[i] <- 5
  }
}


summary_low = expand.grid(
  A = c(0, 1),
  quintile = c(1, 2, 3, 4, 5),
  n = NA,
  prop = NA,
  avg_y = NA
)

for (i in 1:nrow(summary_low)) {
  subset <- df_low[(df_low$A == summary_low$A[i]) &
                     (df_low$quintile_class_low == summary_low$quintile[i]), ]
  summary_low$n[i] = nrow(subset)
  summary_low$prop[i] = summary_low$n[i]/nrow(df_low)
  summary_low$avg_y[i] = mean(subset$Y)
}


for (i in 1:nrow(summary_low)) {
  if (is.nan(summary_low$avg_y[i]) == TRUE) {
    summary_low$avg_y[i] <- 0
  }
}

# this table records the mean response in each quintile; needed for stratification
summary_low
```

```
##    A quintile  n       prop    avg_y
## 1  0        1 15 0.03157895 18.25654
## 2  1        1  0 0.00000000  0.00000
## 3  0        2  0 0.00000000  0.00000
## 4  1         2   0 0.00000000  0.00000
## 5  0         3 227 0.47789474 15.06273
## 6  1         3  28 0.05894737 18.56302
## 7  0         4  65 0.13684211 17.99996
## 8  1         4  16 0.03368421 20.74083
## 9  0         5  56 0.11789474 19.23768
## 10 1         5  68 0.14315789 22.65577
```

```
quntile_prop_low <- summary_low %>% group_by(quintile) %>% summarise(sum = sum(n)/nrow(df_low))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# this table records the proportions for each quintile; also needed for stratification
quntile_prop_low
```

```
## # A tibble: 5 x 2
##   quintile    sum
##      <dbl>  <dbl>
## 1        1 0.0316
## 2        2 0
## 3        3 0.537
## 4        4 0.171
## 5        5 0.261
```

```
ATE_stratification_low = quntile_prop_low$sum[1]*(summary_low$avg_y[2] - summary_low$avg_y[1]) +
  quntile_prop_low$sum[2]*(summary_low$avg_y[4] - summary_low$avg_y[3]) +
  quntile_prop_low$sum[3]*(summary_low$avg_y[6] - summary_low$avg_y[5]) +
  quntile_prop_low$sum[4]*(summary_low$avg_y[8] - summary_low$avg_y[7]) +
  quntile_prop_low$sum[5]*(summary_low$avg_y[10] - summary_low$avg_y[9])

ATE_stratification_low
```

```
## [1] 2.662275
```

```
############### SET END TIME HERE
```

Regression Adjustment

Description

High Dimensional Data

Description

```
############### SET START TIME HERE

ps_RA_high <- predict(tree_high, df_high, type = "prob")
high_data_ps <- cbind(ps_RA_high, df_high)
pred_high <- lm(Y ~ A + ps_RA_high, data = high_data_ps)
summary(pred_high)
```

```
##
## Call:
## lm(formula = Y ~ A + ps_RA_high, data = high_data_ps)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.4713  -3.4878  -0.6694   2.7522  30.0897
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.9564     0.6207   9.596   <2e-16 ***
## A            -2.5271     0.2569  -9.836   <2e-16 ***
## ps_RA_high0 -30.5718     1.0322 -29.617   <2e-16 ***
## ps_RA_high1       NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.535 on 1997 degrees of freedom
## Multiple R-squared:  0.3068, Adjusted R-squared:  0.3061
## F-statistic: 441.8 on 2 and 1997 DF,  p-value: < 2.2e-16
```

```
ATE_regression_adjustment_high = pred_high$coefficients[2]
ATE_regression_adjustment_high
```

```
##         A
## -2.527116
```

```
############### SET END TIME HERE
```

Low Dimensional Data *Description*

```
############### SET START TIME HERE

ps_RA_low <- predict(tree_low, df_low, type = "prob")
low_data_ps <- cbind(ps_RA_low, df_low)
pred_low <- lm(Y ~ A + ps_RA_low, data = low_data_ps)
summary(pred_low)
```

```
##
## Call:
## lm(formula = Y ~ A + ps_RA_low, data = low_data_ps)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5153  -2.6807  -0.5655   1.8184  26.9282
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  22.5693     0.8535  26.445  < 2e-16 ***
## A             3.0532     0.5089   5.999 3.95e-09 ***
## ps_RA_low0   -7.5200     1.0017  -7.507 3.04e-13 ***
## ps_RA_low1        NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.055 on 472 degrees of freedom
## Multiple R-squared:  0.2829, Adjusted R-squared:  0.2798
## F-statistic: 93.08 on 2 and 472 DF,  p-value: < 2.2e-16
```

```
ATE_regression_adjustment_low = pred_low$coefficients[2]
ATE_regression_adjustment_low
```

```
##       A
## 3.05324
```

```
############### SET END TIME HERE
```

Stratification and Regression Adjustment

Description

High Dimensional Data *Description*

```
############### SET START TIME HERE

lm_beta_high <- rep(NA, K)

for (i in 1:K){
  subset <- df_high[df_high$quintile_class_high == i, ]

  if (nrow(subset) == 0) {
    # if the quintile is empty, let the coefficient for A automatically be 0
    lm_beta_high[i] <- 0
  } else if (sum(subset$prop_score_high) == 0) {
    # if the propensity scores in the quintile are all 0, let the coefficient for A automatically be 0
    lm_beta_low[i] <- 0
  } else {
    # otherwise, run a linear model on the subset
    lm <- lm(Y ~ A + prop_score_high, data = subset)
    lm_beta_high[i] <- as.numeric(lm$coefficients[2])
  }
}

lm_beta_high
```

```
## [1]  5.005563  0.000000 -2.503576 -2.516297 -2.675199
```

```
ATE_stratification_regression_adjustment_high <- quntile_prop_high$sum[1]*lm_beta_high[1] +
  quntile_prop_high$sum[2]*lm_beta_high[2] +
  quntile_prop_high$sum[3]*lm_beta_high[3] +
  quntile_prop_high$sum[4]*lm_beta_high[4] +
  quntile_prop_high$sum[5]*lm_beta_high[5]
```

```
ATE_stratification_regression_adjustment_high
```

```
## [1] -2.504545
```

```
############### SET END TIME HERE
```

Low Dimensional Data *Description*

```
############### SET START TIME HERE

lm_beta_low <- rep(NA, 5)

for (i in 1:K){
  subset <- df_low[df_low$quintile_class_low == i, ]

  if (nrow(subset) == 0) {
    # if the quintile is empty, let the coefficient for A automatically be 0
    lm_beta_low[i] <- 0
  } else if (sum(subset$prop_score_low) == 0) {
    # if the propensity scores in the quintile are all 0, let the coefficient for A automatically be 0
    lm_beta_low[i] <- 0
  } else {
    # otherwise, run a linear model on the subset
    lm <- lm(Y ~ A + prop_score_low, data = subset)
    lm_beta_low[i] <- as.numeric(lm$coefficients[2])
  }
}

lm_beta_low
```

```
## [1] 0.000000 0.000000 3.500291 2.452154 2.916085
```

```
ATE_stratification_regression_adjustment_low <- quntile_prop_low$sum[1]*lm_beta_low[1] +
  quntile_prop_low$sum[2]*lm_beta_low[2] +
  quntile_prop_low$sum[3]*lm_beta_low[3] +
  quntile_prop_low$sum[4]*lm_beta_low[4] +
  quntile_prop_low$sum[5]*lm_beta_low[5]

ATE_stratification_regression_adjustment_low
```

```
## [1] 3.058512
```

```
############### SET END TIME HERE
```

Results

Insert Comparison of ATE and all Runtimes Here 8 Runtime values 6 ATE estimations + 2 true ATE create table and analyze results

Conclusion

Description

References

Description