# DoublyRobust_BoostedStumps

Xujie Ma, Yiqi Lei

11/26/2020

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

## Data Import

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

## 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),
  interaction.depth = c(1, 3, 5),
  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$n.trees[i],
    interaction.depth = hyper_grid_high1$interaction.depth[i],
    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=40, shrinkage=0.1, interaction.depth=5, n.minobsinnode=5, bag.fraction=0.8).

```
set.seed(2020)

tm_highe1 <- system.time(
  boost.high<-gbm(A~., data = train_high[-1],
                  distribution = "bernoulli",
                   n.trees = 40, # the number of trees
                   shrinkage = 0.1, # learning rate
                   interaction.depth = 5, # total split
                  n.minobsinnode = 5,
                  bag.fraction = 0.8
                   )
  )
```

Calculate propensity scores for all entries in high.csv

```
tm_highe2 <- system.time(
  high$e <- predict(boost.high, X.high, n.trees = 40, 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] -2.961551
```

```
#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] -0.03844897
```

```
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= 0.84 s
```

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

```
## Time for getting propensity score= 0.019 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),
  interaction.depth = c(1, 3, 5),
  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$n.trees[i],
    interaction.depth = hyper_grid_low1$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 = 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=40, shrinkage=0.1, interaction.depth=1, n.minobsinnode=15, bag.fraction=0.65).

```
set.seed(2020)
```

```
tm_lowe1 <- system.time(
boost.low <- gbm(A~., data = train_low[-1],
                 distribution = "bernoulli",
                 n.trees = 40, # the number of trees
                 shrinkage = 0.1, # learning rate
                 interaction.depth = 1, # total split
                 n.minobsinnode = 15,
                 bag.fraction = 0.65
            )
)
```

Calculate propensity scores for all entries in high.csv

```
tm_lowe2 <- system.time(
low$e <- predict(boost.low, X.low, n.trees = 40, 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] 2.531308
```

```
#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] -0.03130752
```

```
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= 0.018 s
```

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

```
## Time for getting propensity score= 0.002 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 45 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 0.86
## LowDim   2.531      2.5 -0.031 0.02
```