

test

Levi Lee

11/17/2020

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
```

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

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

```
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

```
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

```
# hyperparameters for trees  
hyper_grid_trees <- expand.grid(  
  cp = c(20, 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, -20)  
)
```

Step 2: Cross-Validate the Hyperparameters

Description

Description

```
# 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*

```
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*

```
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

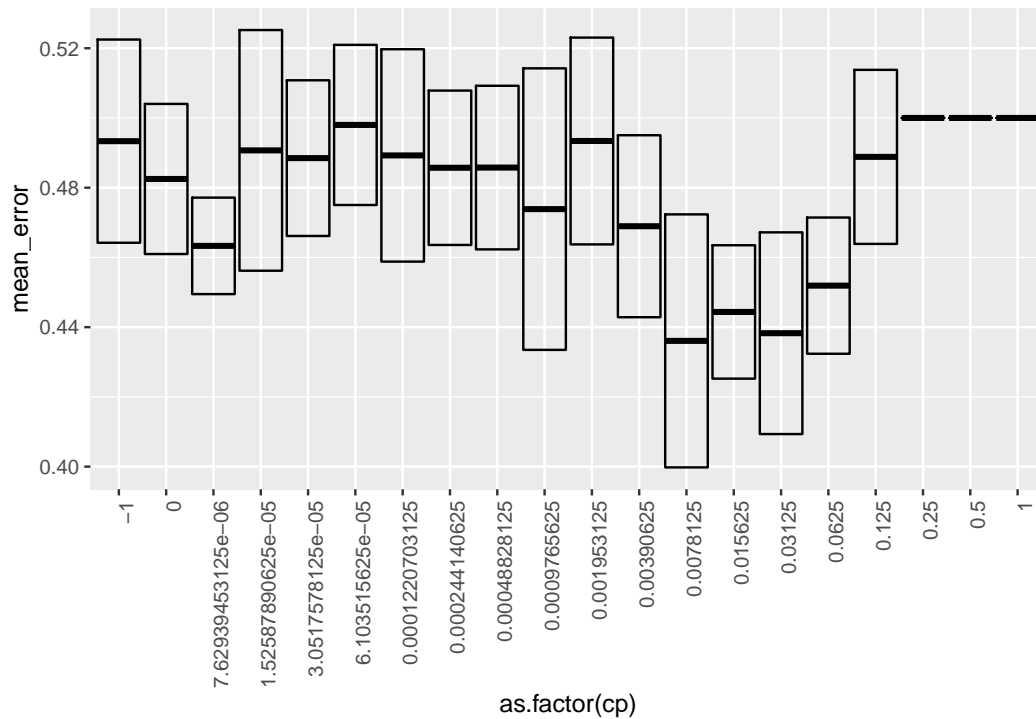
Description

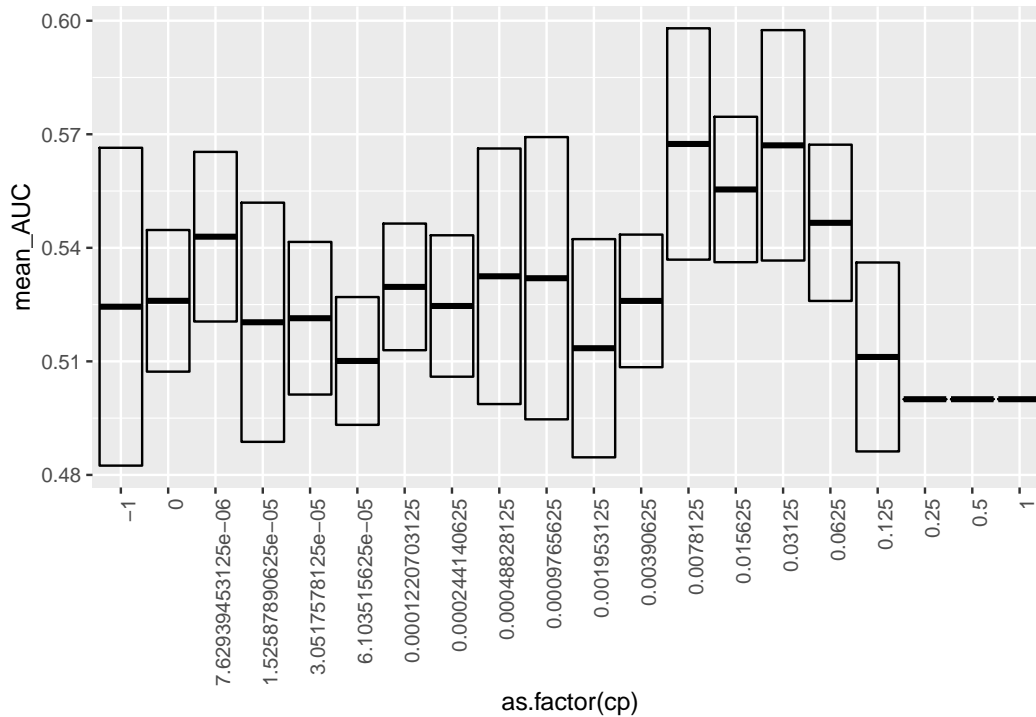
High Dimensional Data *Description*

```
# 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
```

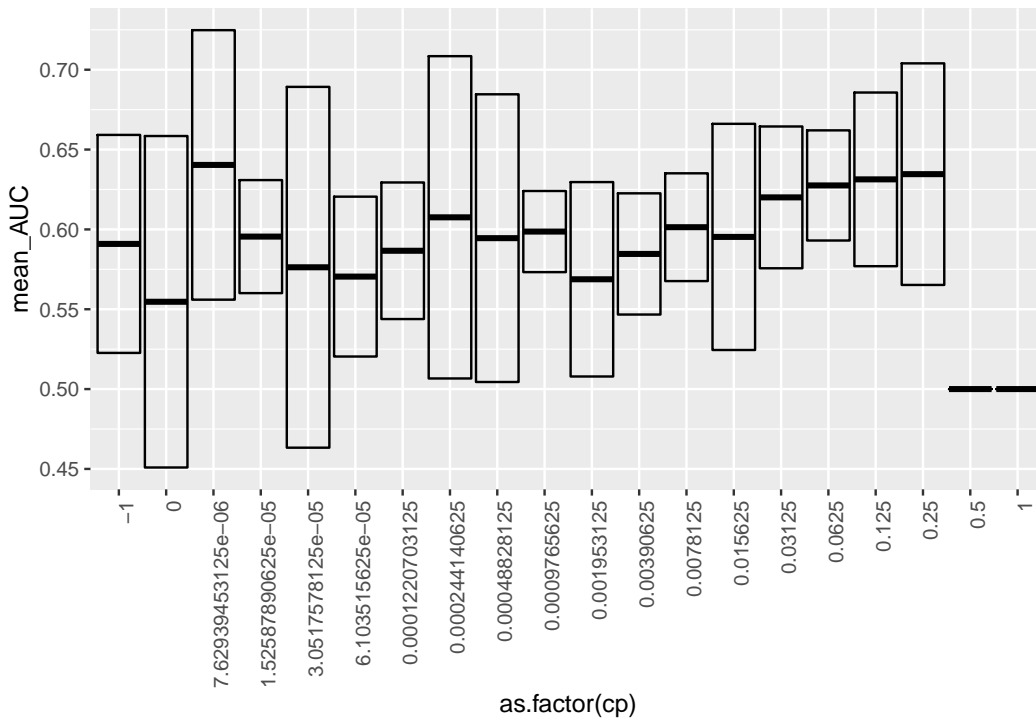
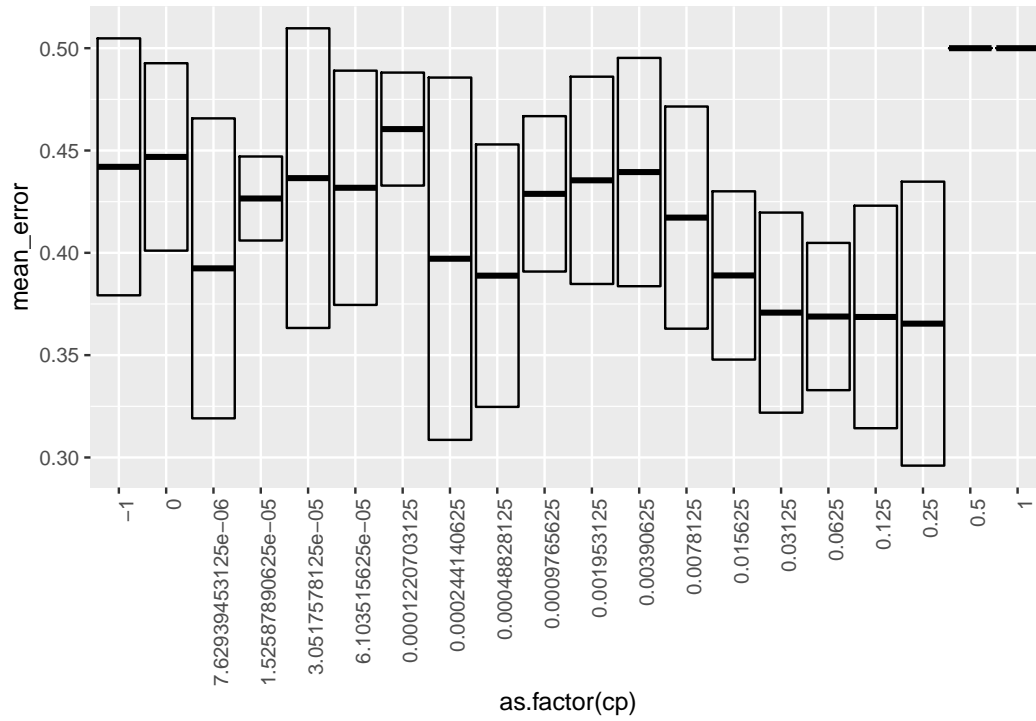
```
## [1] 0.0078125
```

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])
}
```

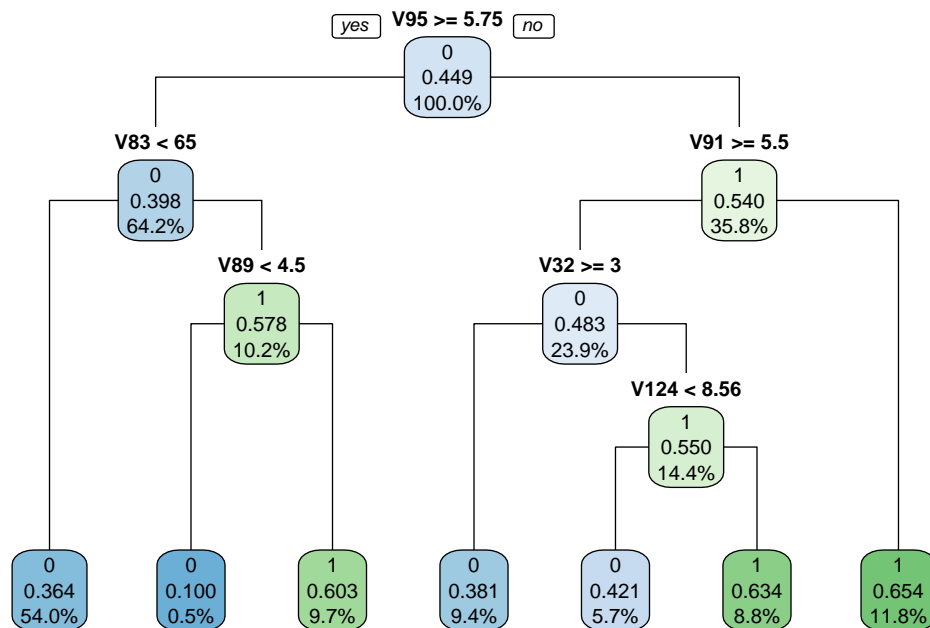
```
##### SET START TIME HERE

# 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

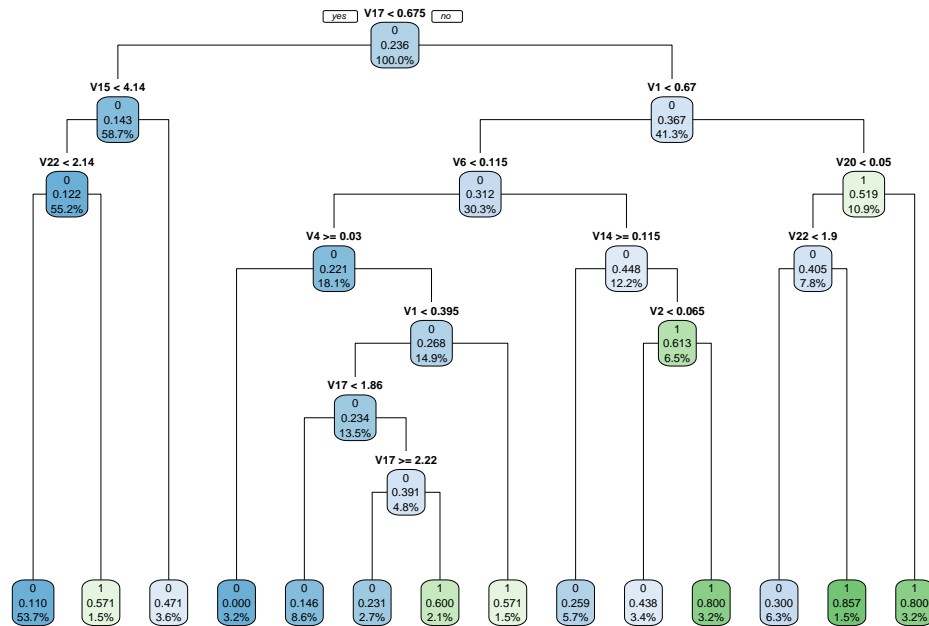
```
tree_low <- rpart(A ~ . - Y, method = "class", data = df_low, cp = best_cp_low)
```

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 Also need run times here*

Low Dimensional Data *Description Also need run times here*

Regression Adjustment

Description

High Dimensional Data *Description Also need run times here*

Low Dimensional Data *Description Also need run times here*

Stratification and Regression Adjustment

Description

High Dimensional Data *Description Also need run times here*

Low Dimensional Data *Description Also need run times here*

Results

Insert Comparison of ATE and all Runtimes Here

Conclusion

Description

References

Description