Stat 432 HW 05

Name: Ahmadreza Eslaminia, netID: Ae15

Summer 2024

Include the R code for this HW.

```
knitr::opts_chunk$set(echo = TRUE)
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.3
```

```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.3.3
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
library(tibble)
```

```
## Warning: package 'tibble' was built under R version 4.3.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.3.3
```

```
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 4.3.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
#add more libraries as needed.
```

Question 1 (k-NN, tree for classification)

Use `hw5data1.Rdata` to answer this question.

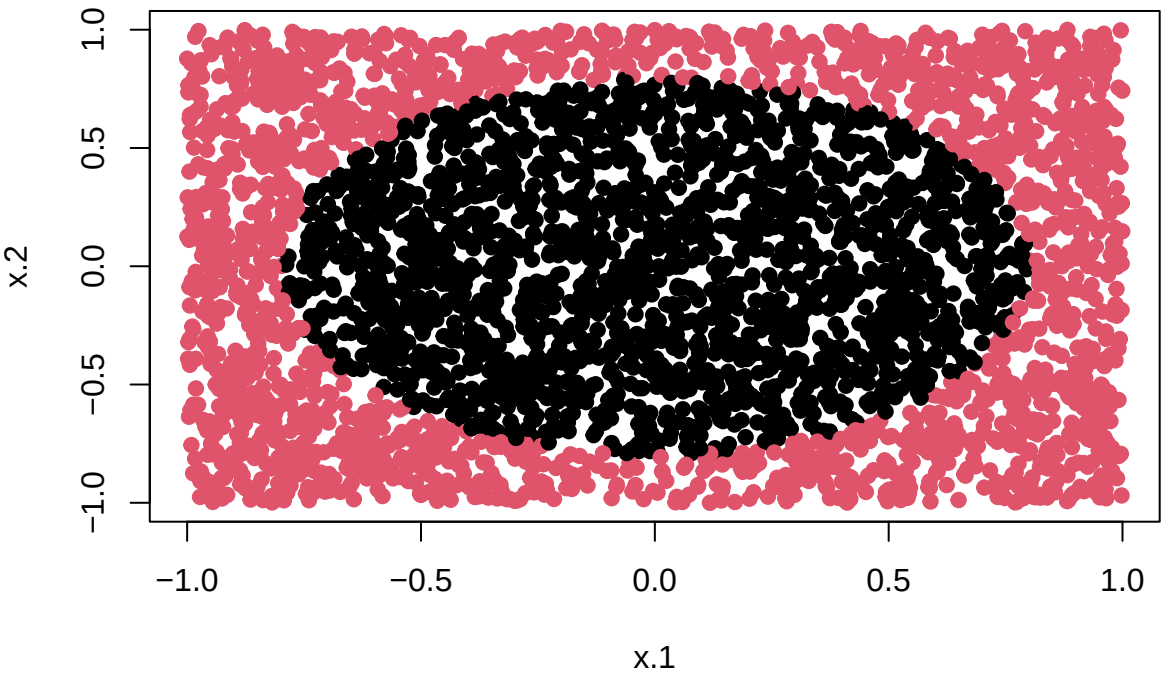
```
load("hw5data1.Rdata") #put this file in your working directory or in the same folder as your HW Rmd fi
str(circle.trn)#your training data
```

```
## tibble [4,000 x 3] (S3: tbl_df/tbl/data.frame)
##  $ x.1    : num [1:4000] -0.479 0.2 0.32 0.888 0.333 ...
##  $ x.2    : num [1:4000] 0.36 0.784 0.581 -0.271 0.532 ...
##  $ classes: Factor w/ 2 levels "1","2": 1 2 1 2 1 1 1 2 2 2 ...
```

```
str(circle.tst)# your test data
```

```
## tibble [1,000 x 3] (S3: tbl_df/tbl/data.frame)
##  $ x.1    : num [1:1000] 0.9886 -0.0187 -0.4419 0.7234 0.1634 ...
##  $ x.2    : num [1:1000] 0.352 -0.234 0.278 0.209 -0.259 ...
##  $ classes: Factor w/ 2 levels "1","2": 2 1 1 1 1 2 1 1 2 1 ...
```

```
plot(x.2~x.1,data=circle.trn,col=circle.trn$classes,pch=19)
```



- `classes` variable: categorical response variable
- `x.1`, `x.2`: feature variables

```
circle.trn$classes <- as.factor(circle.trn$classes)
circle.tst$classes <- as.factor(circle.tst$classes)
```

(a) The given graphic is the plot of data, using different color for different classes. Based on given information, would decision tree work well? Explain why or why not.

As the plots shows the two classes are not linearly seperable and would habe complex decision boundries , the decision tree would not be a good choice here and probably would not work well. Decision trees tend to create axis-alignd splits, which might not captur the curved boundary well, This can lead to overfiting with deep trees or underfitting with shallow trees.

(b) Conduct k-NN classification using `train()` function of `caret` package, with 10-fold cross validation. Use the grid of odd numbers, from 1 to 101 for `k`. Choose best k. (For this problem, do not need to consider scaling.)

```
set.seed(1)
knn_grid <- expand.grid(k = seq(1, 101, by = 2))
knn_model <- train(classes ~ ., data = circle.trn, method = "knn", trControl = trainControl(method = "c
```

```
# Best k
best_k <- knn_model$bestTune$k
print(paste("Best k value:", best_k))
```

```
## [1] "Best k value: 1"
```

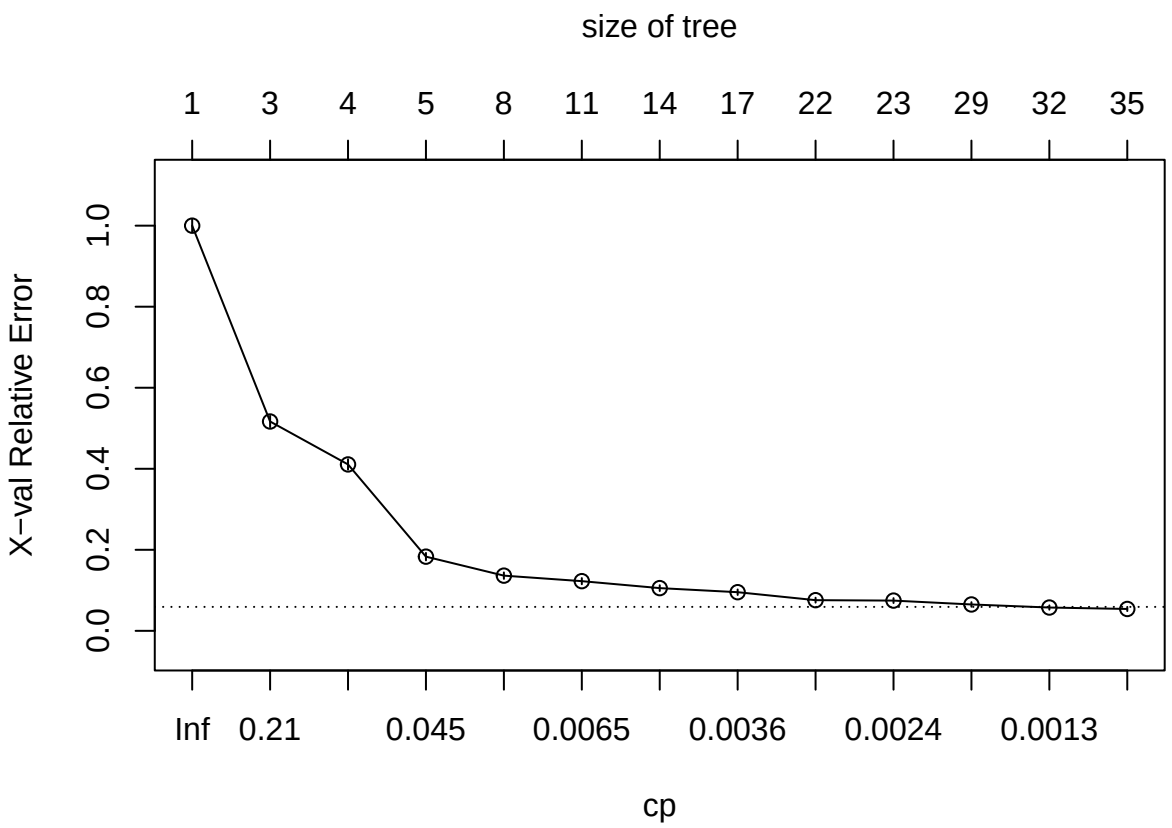
(c) Conduct tree classification using `rpart()` function of `rpart` package. Use `cp=0` to grow a big tree. Then create the cp-table and cp vs size of plot. (By default, this function use 10-fold cross validation. This number can be controlled using `xval=` option if necessary. No need to change for this HW.) Based on the result, choose the optimal cp value.

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

```
## Warning: package 'rpart.plot' was built under R version 4.3.3
```

```
#  with cp=0
tree_model <- rpart(classes ~ ., data = circle.trn, cp = 0)


plotcp(tree_model)
```




```
printcp(tree_model)
```

```
##
## Classification tree:
## rpart(formula = classes ~ ., data = circle.trn, cp = 0)
##
## Variables actually used in tree construction:
## [1] x.1 x.2
##
## Root node error: 1982/4000 = 0.4955
##
## n= 4000
##
##           CP nsplit rel error   xerror      xstd
## 1  0.2479818      0  1.000000 1.000000 0.0159543
## 2  0.1755802      2  0.504036 0.516650 0.0139262
## 3  0.1659939      3  0.328456 0.410696 0.0128470
## 4  0.0123613      4  0.162462 0.183148 0.0091662
## 5  0.0080727      7  0.122099 0.136226 0.0080058
## 6  0.0052136     10  0.093340 0.122603 0.0076224
## 7  0.0045409     13  0.077699 0.105449 0.0071009
## 8  0.0029011     16  0.064077 0.095358 0.0067704
## 9  0.0025227     21  0.048436 0.075681 0.0060624
## 10 0.0023545     22  0.045913 0.074672 0.0060234
## 11 0.0016818     28  0.031786 0.065086 0.0056373
## 12 0.0010091     31  0.026741 0.057518 0.0053097
## 13 0.0000000     34  0.023713 0.053986 0.0051487
```

```r
optimal_cp <- 0.01
print(paste("Optimal cp value:", optimal_cp))
```

```
## [1] "Optimal cp value: 0.01"
```

```r
#  optimal cp
pruned_tree <- prune(tree_model, cp = optimal_cp)
```

According to the plot we can see that after cp = 0.01 the error doe snot significantly drops so we can chose this number as the complexity param.

(d) Using the models chosen from (b) and (c), refit the models to the whole training data and report test accuracy. Which method is performing better on our test data?

```r
knn_final_model <- train(classes ~ ., data = circle.trn, method = "knn", tuneGrid = data.frame(k = best

# k-NN model
knn_final_predictions <- predict(knn_final_model, newdata = circle.tst)
knn_final_confusion <- confusionMatrix(knn_final_predictions, circle.tst$classes)
knn_final_accuracy <- knn_final_confusion$overall['Accuracy']
print(paste("k-NN Test Accuracy:", knn_final_accuracy))
```

```
## [1] "k-NN Test Accuracy: 0.986"
```

```r
# pruned tree
tree_final_predictions <- predict(pruned_tree, newdata = circle.tst, type = "class")
tree_final_confusion <- confusionMatrix(tree_final_predictions, circle.tst$classes)
tree_final_accuracy <- tree_final_confusion$overall['Accuracy']
print(paste("Decision Tree Test Accuracy:", tree_final_accuracy))
```

```
## [1] "Decision Tree Test Accuracy: 0.917"
```

According to the results the test accuracy for the K-NN model is way better than the Desion tree as we discussed in previous part a.

Question 2

This question relates to the `Boston` data set of `ISLR2` package.

```
set.seed(42)
trn.idx=sample(1:nrow(ISLR2::Boston),450)
tst.boston=ISLR2::Boston[-trn.idx,]
trn.boston=ISLR2::Boston[trn.idx,]
```

We are splitting the data into two parts: a testing data that contains 56 observations, and the rest 450 observations as training data.

- The goal is to model `crim` (our response variable) with all the other variables in the data.
- Use `train` function of `caret` package for this question.

(a) Conduct linear regression with 10-fold CV. Report CV error for the chosen parameter. (RMSE or MSE, either way is ok. Just need to be consistent throughout this problem.) In this HW, use:

```
control=trainControl(method = "cv",number=10)

lm.boston<-train(formula,data=data,
                 method = 'lm',
                 trControl=control
               )
```

```
control <- trainControl(method = "cv", number = 10)

# liner reg
lm_model <- train(crim ~ ., data = trn.boston, method = "lm", trControl = control)

# RMSE
lm_cv_error <- lm_model$results$RMSE
print(paste("Linear Regresion CV RMSE:", lm_cv_error))
```

```
## [1] "Linear Regresion CV RMSE: 5.9919830429278"
```

(b) Conduct k-NN regression with 10-fold CV. Choose optimal tuning parameter. Report CV error for the chosen parameter. Use train function of caret package.

Consider two different pre-processing setups.

- Setup 1: Numeric variables not scaled.
- Setup 2: *Numeric variables are scaled* to have mean 0 and standard deviation 1. You need to add `preProcess = c("center","scale")` option inside the train function.

Which setup and `k` gives the lowest error?

```
set.seed(42)
# not scaled
knn_model_1 <- train(crim ~ ., data = trn.boston, method = "knn", trControl = control, tuneLength = 10)
best_k_1 <- knn_model_1$bestTune$k
knn_cv_error_1 <- knn_model_1$results[knn_model_1$results$k == best_k_1, "RMSE"]
print(paste("k-NN (not scled) Best k:", best_k_1))
```

```
## [1] "k-NN (not scled) Best k: 21"
```

```
print(paste("k-NN (not scled) CV RMSE:", knn_cv_error_1))
```

```
## [1] "k-NN (not scled) CV RMSE: 5.60273106625454"
```

```
#  scaled
set.seed(42)
knn_model_2 <- train(crim ~ ., data = trn.boston, method = "knn", trControl = control, preProcess = c("
best_k_2 <- knn_model_2$bestTune$k
knn_cv_error_2 <- knn_model_2$results[knn_model_2$results$k == best_k_2, "RMSE"]
print(paste("k-NN (scled) Best k:", best_k_2))
```

```
## [1] "k-NN (scled) Best k: 19"
```

```
print(paste("k-NN (scled) CV RMSE:", knn_cv_error_2))
```

```
## [1] "k-NN (scled) CV RMSE: 5.55730608543818"
```

AS we can see here model considering scaling can enhance the performance a little bit. So the best model is k = 19 with considering scaling.

(c) Conduct ridge regression with 10-fold CV. In this HW, use the `train()` function of the `caret` package:

```
control=trainControl(method = "cv",number=10)

lasso<-train(formula,data=data,
                 method = 'glmnet',
                 trControl=control,preProc = c("center","scale"),
                 tuneGrid = expand.grid(alpha = 1, lambda =seq(from=0,to=1,by=0.01))
                 #alpha=1 indicates lasso method. You can choose your own grid of lambda.
               )

ridge<-train(formula,data=data,
                 method = 'glmnet',
                 trControl=control,preProc = c("center","scale"),
                 tuneGrid = expand.grid(alpha = 0, lambda =seq(from=0,to=1,by=0.01))
  #alpha=0 indicates ridge regression method. You can choose your own grid of lambda.

               )
```

```
#  lasso
lasso <- train(crim ~ ., data = trn.boston, method = 'glmnet',
               trControl = control, preProc = c("center", "scale"),
               tuneGrid = expand.grid(alpha = 1, lambda = seq(0, 1, by = 0.01)))

# Lsso (RMSE)
lasso_cv_error <- min(lasso$results$RMSE)
best_lasso_lambda <- lasso$bestTune$lambda
print(paste("Lasso CV RMSE:", lasso_cv_error))
```

```
## [1] "Lasso CV RMSE: 5.89443538861217"
```

```r
print(paste("Best lamnda for Lasso:", best_lasso_lambda))
```

```
## [1] "Best lamnda for Lasso: 0.1"
```

```r
#  ridge
ridge <- train(crim ~ ., data = trn.boston, method = 'glmnet',
               trControl = control, preProc = c("center", "scale"),
               tuneGrid = expand.grid(alpha = 0, lambda = seq(0, 1, by = 0.01)))

# ridge (RMSE)
ridge_cv_error <- min(ridge$results$RMSE)
best_ridge_lambda <- ridge$bestTune$lambda
print(paste("Ridg Regresion CV RMSE:", ridge_cv_error))
```

```
## [1] "Ridg Regresion CV RMSE: 5.99793353751411"
```

```r
print(paste("Best lammda for Ridg Regression:", best_ridge_lambda))
```

```
## [1] "Best lammda for Ridg Regression: 0.53"
```

Find best tuning parameter for each lasso and ridge regression, and report CV error. It can be seen that the Ridge outperform the lasso here. As the best labmda are not on the boundries used grid should be fine.

(d) Conduct Bagging, Random Forest, and Boosting with 10-fold CV. Use the `train()` function of the `caret` package. Find best tuning parameter for each methods.

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# Bagging
bagging <- train(crim ~ ., data = trn.boston, method = 'treebag',
                 trControl = control)
bagging_cv_error <- min(bagging$results$RMSE)
print(paste("Baging CV RMSE:", bagging_cv_error))
```

```
## [1] "Baging CV RMSE: 6.22420682605003"
```

```
# Random Forest
rf <- train(crim ~ ., data = trn.boston, method = 'rf',
            trControl = control, tuneLength = 10)
rf_cv_error <- min(rf$results$RMSE)
best_rf_mtry <- rf$bestTune$mtry
print(paste("Random forest CV RMSE:", rf_cv_error))
```

```
## [1] "Random forest CV RMSE: 5.17548473159112"
```

```
print(paste("Best mtry for Random Forest:", best_rf_mtry))
```

```
## [1] "Best mtry for Random Forest: 2"
```

```
# Boosting
boosting <- train(crim ~ ., data = trn.boston, method = 'gbm',
                  trControl = control, tuneLength = 10, verbose = FALSE)
boosting_cv_error <- min(boosting$results$RMSE)
best_boosting_n_trees <- boosting$bestTune$n.trees
best_boosting_interaction_depth <- boosting$bestTune$interaction.depth
best_boosting_shrinkage <- boosting$bestTune$shrinkage
best_boosting_n_minobsinnode <- boosting$bestTune$n.minobsinnode
print(paste("Boosting CV RMSE:", boosting_cv_error))
```

```
## [1] "Boosting CV RMSE: 5.44480383195644"
```

```
print(paste("Best n.tres for Boosting:", best_boosting_n_trees))
```

```
## [1] "Best n.tres for Boosting: 50"
```

```
print(paste("Best interction.depth for Boosting:", best_boosting_interaction_depth))
```

```
## [1] "Best interction.depth for Boosting: 7"
```

```
print(paste("Best shrikage for Boosting:", best_boosting_shrinkage))
```

```
## [1] "Best shrikage for Boosting: 0.1"
```

```
print(paste("Best n.mino for Boosting:", best_boosting_n_minobsinnode))
```

```
## [1] "Best n.mino for Boosting: 10"
```

(e) Based on (a)-(d), pick the best method and train your whole training data set using the chosen method and tuning parameter(s). Report the test MSE.

According to the results in previous section we can see that the random forest model has the lower error. So we use this model for traing, and testing.

```r
best_rf_model <- train(crim ~ ., data = trn.boston, method = 'rf',
                       trControl = control, tuneGrid = data.frame(mtry = 2))

test_predictions <- predict(best_rf_model, tst.boston)

#(MSE) on the test
test_mse <- mean((tst.boston$crim - test_predictions)^2)
print(paste("Test MSE for best model(random forests):", test_mse))
```

```
## [1] "Test MSE for best model(random forests): 4.60202399262098"
```