Homework #4

Spencer Pease

3/08/2021

Question 1

Part (a, b)

For this exercise, 50 observations are generated from the function

$$Y = f(X) + \epsilon$$

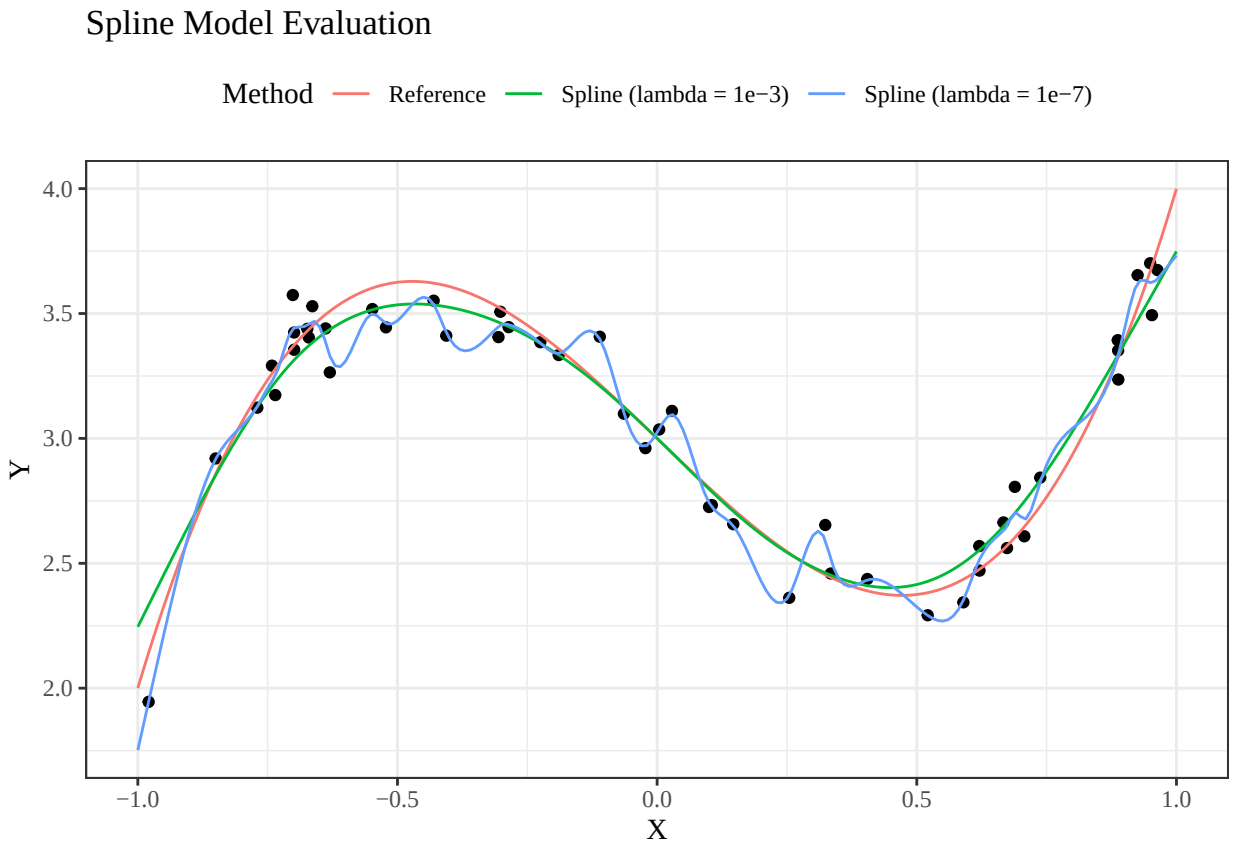
where

$$\begin{aligned} X &\sim \text{Unifrom}(-1, 1) \\ \epsilon &\sim \text{Normal}(0, 0.1) \\ f(X) &= 3 - 2X + 3X^3 \end{aligned}$$

Part (c)

Two smoothing spline models, with $\lambda = 10^{-3}$ and $\lambda = 10^{-7}$, are then fit to the generated dataset.

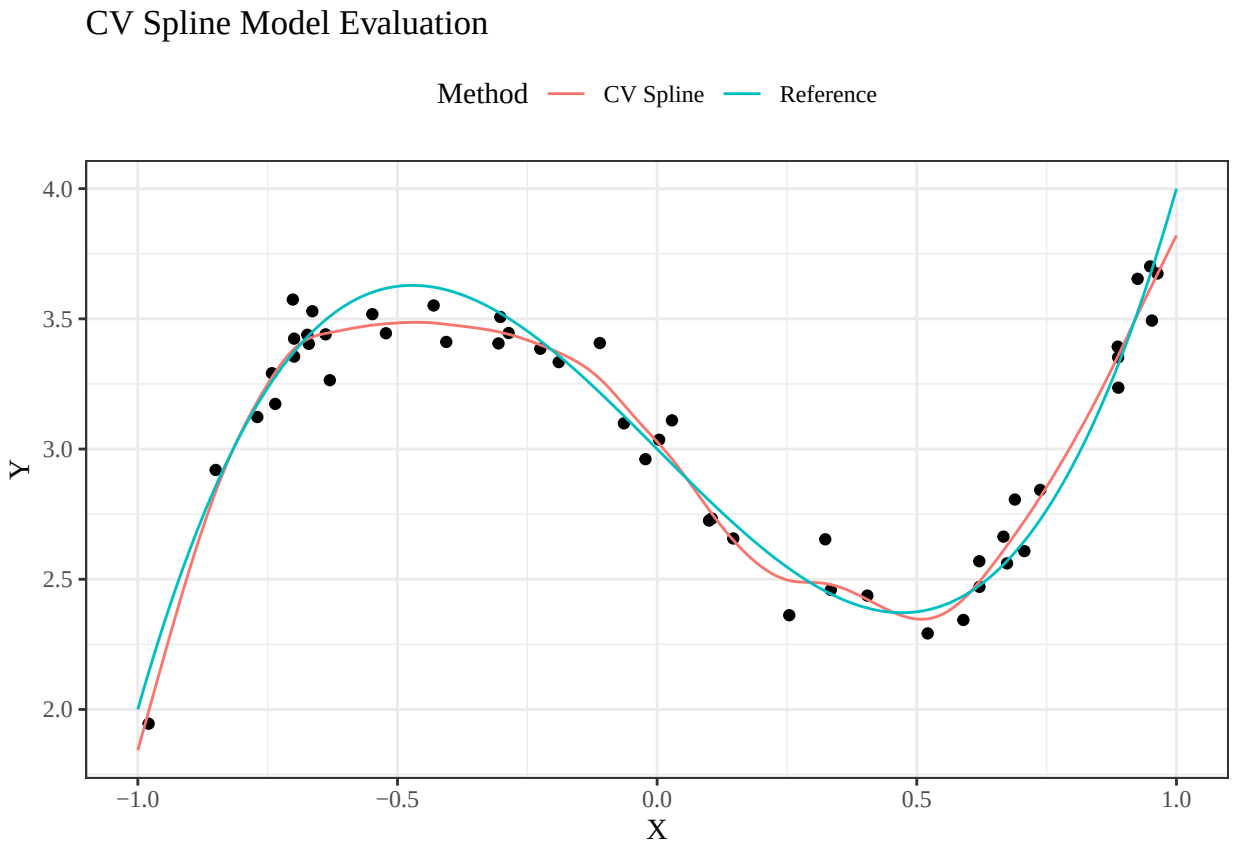
Part (d)



Part (e)

Fitting a cross-validated smoothing spline model to the generated dataset shows that the optimal value of λ is 2.879e-05.

Part (f)



Part (g)

Table 1: Bootstraped variance at X = 0

$\hat{f}_{\lambda=10^{-3}}$	$\hat{f}_{\lambda=10^{-7}}$
1.037e-03	4.321e-03

Looking at the estimated variance of $X = 0$ from 1000 bootstrapped datasets for each spline model, we see that the model with greater smoothing had a lower variance. This is consistent with our understanding of how a less smooth model will overfit to the data more, leading to greater variation in estimates when fit to many bootstrapped datasets.

Question 2

Part (a)

Table 2: Observations by disease class

Disease	observations
No Disease	171
Heart Disease	200

The *heart* dataset has a sample size n of 371 and 12 predictors p. *Table 2* shows a summary of the number of observations in each class.

For model training, the *heart* dataset is split into a training set of 200 observations and test set of 171 observations.

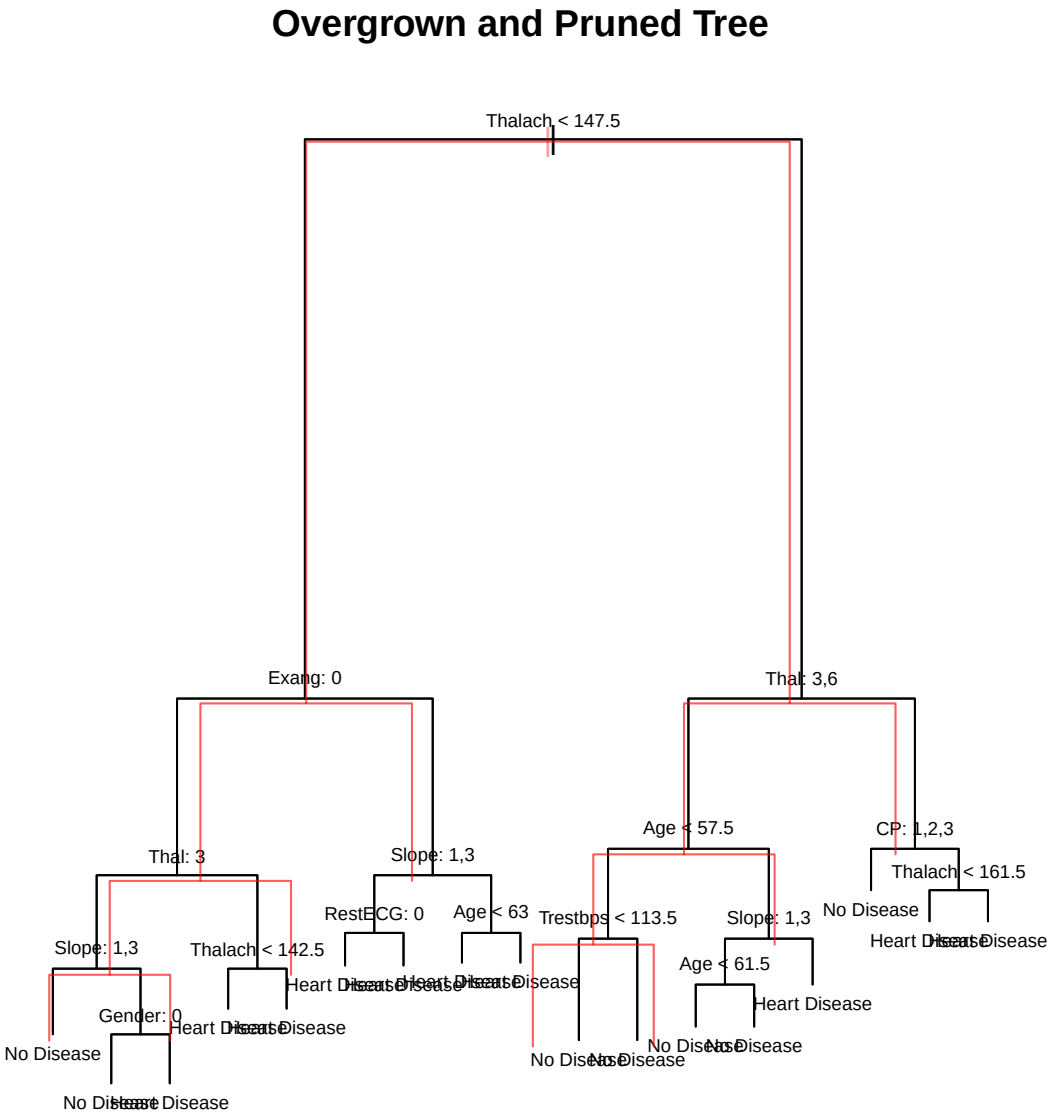
Part (b, c)

Table 3: Misclassification error rates for the overgrown model

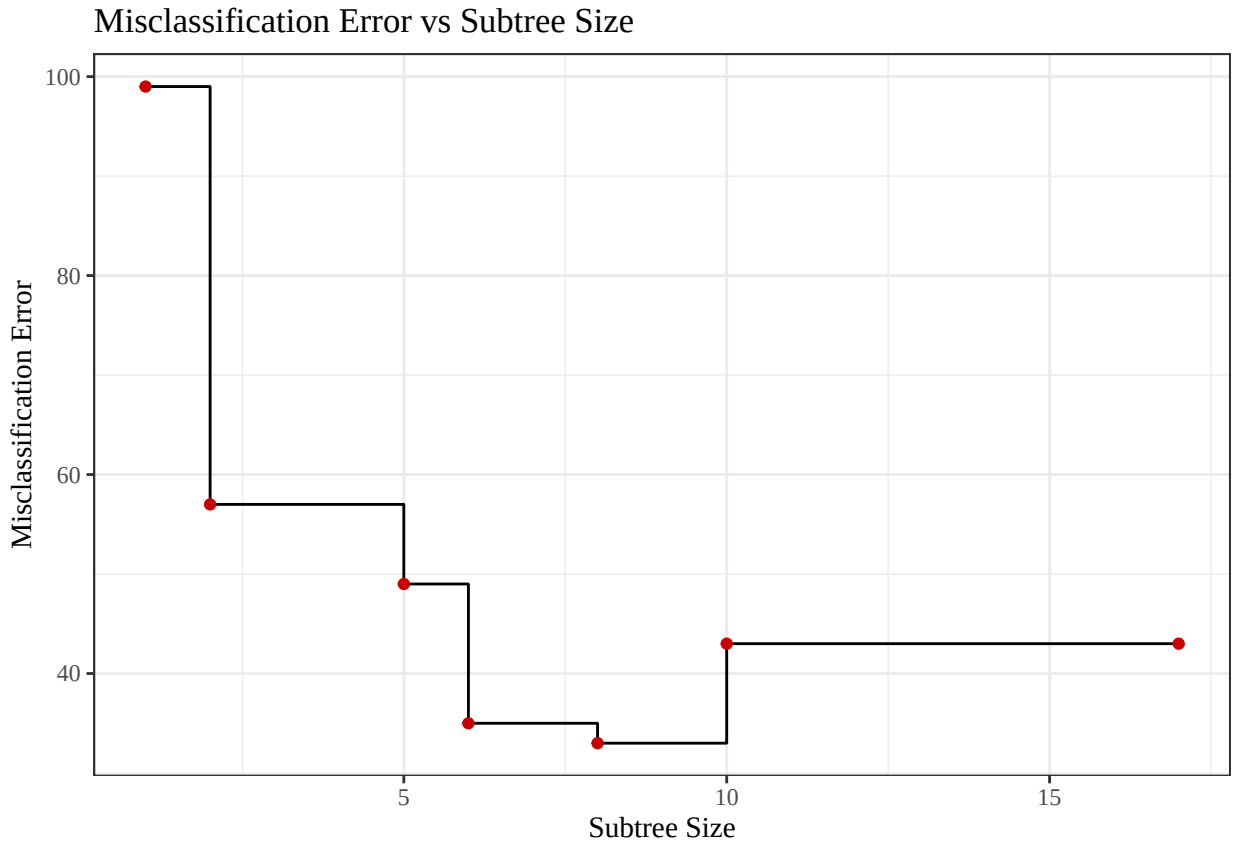
train	test
0.125	0.234

The overgrown tree model has a lower misclassification rate when predicting the training data class than when predicting the test data, which is expected since the model is overfit to the training data.

Part (d)



Key: Black = overgrown tree; Red = pruned tree branches



Part (e)

Table 4: Misclassification error rates for the optimal subtree model

train	test
0.155	0.251

The training misclassification rate of the optimal subtree model is higher than that of the overgrown tree, which is expected as there are fewer splits to overfit on. The test misclassification rate is higher than its training counterpart (as expected) and the that of the overgrown model. This may be an artifact of sampling, but it also shows how fitting the optimal subtree will only sacrifice a little accuracy, if any, for a corresponding reduction in variance.

Part (f)

Table 5: Misclassification error rates for the bagged tree model

train	test
0	0.228

The bagging model is able to reduce the training misclassification to zero, as well as reduce the test misclassification compared to the optimal pruned tree model. This shows the power bootstrapping data has for reducing variance in a tree model.

Part (g)

Table 6: Misclassification error rates for the random forest model

train	test
0	0.211

The random forest model maintains the zero misclassification error for the training set, and reduces the test misclassification further. This shows that sampling a subset of predictors when bootstrapping is another way to create a model more robust to new observations.

Part (h)

Both bagged trees and random forests use bootstrapped data to fit their models. Since bootstrapping involves randomly sampling training data, there is no innate guarantee that a model will be fit using the same data sample each time, leading to randomness in the final results.

In addition, random forests select a subset of predictors for each bagged tree. The selection process also has elements of randomness, so again fitting the same model will produce different results.

Part (i)

Table 7: Misclassification error rates for the boosted tree model

train	test
0	0.193

The boost tree model has the best performance out of all the tree models, with zero training classification error and the lowest test misclassification error. By sequentially building off of the residuals of previous splits, boosted trees target difficult classification cases for improvement and perform better than other methods.

Appendix

Analysis

```
# Prep work ---------------------------------------------------------------

library(dplyr)
library(ggplot2)
library(splines)
library(boot)
library(tree)
library(randomForest)
library(gbm)

source("functions/evaluate_grid.R")
source("functions/misclass_table.R")

load("data/heart.RData")
data_heart <- as_tibble(full)
rm(full)


# Question 1a,b -----------------------------------------------------------

gen_sim_data <- function(n) {

  X <- runif(n = n, min = -1, max = 1)
  noise <- rnorm(n = n, mean = 0, sd = 0.1)
  f <- function(x) 3 - 2 * x + 3 * x^3

  Y <- f(X) + noise

  return(list(X = X, Y = Y, f = f))

}

set.seed(2)
data_sim <- gen_sim_data(n = 50)


# Question 1c -------------------------------------------------------------

model_spline3 <- with(data_sim, smooth.spline(X, Y, lambda = 1e-3))
model_spline7 <- with(data_sim, smooth.spline(X, Y, lambda = 1e-7))


# Question 1d -------------------------------------------------------------

pred_sim_spline <- evaluate_grid(
  data_sim$X, data_sim$Y,
  pred_list = list(
    "Reference" = data_sim$f,
    "Spline (lambda = 1e-3)" = function(x) predict(model_spline3, x)$y,
```

```
    "Spline (lambda = 1e-7)" = function(x) predict(model_spline7, x)$y
  )
)


# Question 1e ------------------------------------------------------------

model_spline_cv <- with(data_sim, smooth.spline(X, Y, cv = TRUE))


# Question 1f ------------------------------------------------------------

pred_sim_spline_cv <- evaluate_grid(
  data_sim$X, data_sim$Y,
  pred_list = list(
    "Reference" = data_sim$f,
    "CV Spline" = function(x) predict(model_spline_cv, x)$y
  )
)


# Question 1g ------------------------------------------------------------

estimate_spline <- function(data, indices, pred_point) {

  data_boot <- data[indices, ]
  spline3 <- with(data_boot, smooth.spline(X, Y, lambda = 1e-3))
  spline7 <- with(data_boot, smooth.spline(X, Y, lambda = 1e-7))

  return(c(
    "spline3" = predict(spline3, pred_point)$y,
    "spline7" = predict(spline7, pred_point)$y
  ))

}

model_spline_boot <-
  with(data_sim, tibble(X = X, Y = Y)) %>%
  boot(estimate_spline, R = 1000, pred_point = 0)

summary_spline_boot <- model_spline_boot$t %>%
  as_tibble(.name_repair = "unique") %>%
  rename(spline3 = 1, spline7 = 2) %>%
  summarise(across(.fns = var, .names =  "{.col}_var"))


# Question 2a ------------------------------------------------------------

obs_by_class <- data_heart %>%
  group_by(Disease) %>%
  summarise(observations = n())

train_obs <- 200
```

```r
set.seed(2)
df_shuffled <- data_heart[sample(nrow(data_heart)), ]
df_train <- df_shuffled[1:train_obs, ]
df_test <- df_shuffled[-(1:train_obs), ]


# Question 2b ------------------------------------------------------------------

model_tree_overgrown <- tree(Disease ~ ., data = df_train)


# Question 2c ------------------------------------------------------------------

misclass_tree_overgrown <- misclass_table(
  model_tree_overgrown,
  df_train,
  df_test,
  type = "class"
)


# Question 2d ------------------------------------------------------------------

set.seed(2)

model_tree_cv <- cv.tree(model_tree_overgrown, FUN = prune.misclass)

optimal_tree_size <- with(model_tree_cv, size[which.min(dev)])
model_tree_prune <- prune.tree(model_tree_overgrown, best = optimal_tree_size)

plot_tree_prune <-
  with(model_tree_cv, tibble(size = size, misclass_err = dev)) %>%
  ggplot(aes(x = size, y = misclass_err)) +
  geom_step() +
  geom_point(color = "red3") +
  theme_bw(base_family = "serif") +
  labs(
    title = "Misclassification Error vs Subtree Size",
    x = "Subtree Size",
    y = "Misclassification Error"
  )


# Question 2e ------------------------------------------------------------------

misclass_tree_opt <- misclass_table(
  model_tree_prune,
  df_train,
  df_test,
  type = "class"
)
```

```r
# Question 2f ------------------------------------------------------------

set.seed(2)

model_tree_bag <- randomForest(
  Disease ~ .,
  data = df_train,
  mtry = ncol(df_train) - 1,
  importance = TRUE
)

# misclass_tree_bag <- list(
#   "train" = mean(model_tree_bag$predicted != df_train$Disease),
#   "test" = mean(predict(model_tree_bag, df_test) != df_test$Disease)
# )

misclass_tree_bag <- misclass_table(model_tree_bag, df_train, df_test)


# Question 2g ------------------------------------------------------------

set.seed(2)

model_tree_rf <- randomForest(
  Disease ~ .,
  data = df_train,
  mtry = floor((ncol(df_train) - 1) / 3),
  importance = TRUE
)

misclass_tree_rf2 <- list(
  "train" = mean(model_tree_rf$predicted != df_train$Disease),
  "test" = mean(predict(model_tree_rf, df_test) != df_test$Disease)
)

misclass_tree_rf <- misclass_table(model_tree_rf, df_train, df_test)


# Question 2i ------------------------------------------------------------

df_train_int <- df_train %>% mutate(Disease = as.integer(Disease) - 1L)
df_test_int <- df_test %>% mutate(Disease = as.integer(Disease) - 1L)

set.seed(2)

model_tree_boost <- gbm(
  Disease ~ .,
  data = df_train_int,
  distribution = "bernoulli",
  n.trees = 500,
  shrinkage = 0.1,
  interaction.depth = 2
)
```

```r
misclass_tree_boost <- misclass_table(
  model_tree_boost,
  df_train_int,
  df_test_int,
  bayes_thresh = 0.5,
  type = "response"
)
```

Helper Functions

```r
evaluate_grid <- function(X, Y, pred_list, grid = seq(-1, 1, .01)) {

  data_point <- tibble(x = X, y = Y)

  data_grid <-
    tibble(x = grid) %>%
    mutate(across(x, .fns = pred_list, .names = "{.fn}")) %>%
    tidyr::pivot_longer(-x, names_to = "method", values_to = "y")

  plot <-
    ggplot(data_point, aes(x, y)) +
    geom_point() +
    geom_line(aes(color = method), data = data_grid) +
    theme_bw(base_family = "serif") +
    theme(legend.position = "top") +
    labs(
      x = "X",
      y = "Y",
      color = "Method"
    )

  return(list(data = data, plot = plot))

}
```

```r
misclass_table <- function(model, df_train, df_test, bayes_thresh = NULL, ...) {

  train_preds <- predict(model, df_train, ...)
  test_preds <- predict(model, df_test, ...)

  if (is.numeric(bayes_thresh)) {
    train_preds <- if_else(train_preds > bayes_thresh, 1L, 0L)
    test_preds <- if_else(test_preds > bayes_thresh, 1L, 0L)
  }

  tibble(
    train = mean(train_preds != df_train$Disease),
    test = mean(test_preds != df_test$Disease)
  )

}
```