BIOST 546 HW 4

My-Anh Doan

2023-02-23

```r
# set global options for code chunks
knitr::opts_chunk$set(message = FALSE, warning = FALSE, collapse = TRUE)
knitr::opts_knit$set(root.dir = rprojroot::find_rstudio_root_file())

library(dplyr)
library(ggplot2)
library(knitr)
library(tree)
library(caret)
library(randomForest)
library(gbm)
```

In this homework assignment, we will perform binary classification on the Heart Disease Data Set in the file `heart.RData`. The data set contains multiple attributes. The goal is to predict heart disease events (field `Disease`). A description of the attributes can be found at this link.

1. Tree models for Heart Disease prediction

- 1a. Describe the data: sample size n, number of predictors p, and number of observations in each class. Divide the data into a training set of 200 observations and a test set. Set the seed with `set.seed(2)` before you sample the training set.

```r
# load data
load("./dataset/heart.RData")

n <- nrow(full)
p <- ncol(full[ , names(full) != "Disease"])
class_count <- full %>%
  count(Disease)
kable(class_count, caption = "Total observations by diagnosis class")
```

Table 1: Total observations by diagnosis class

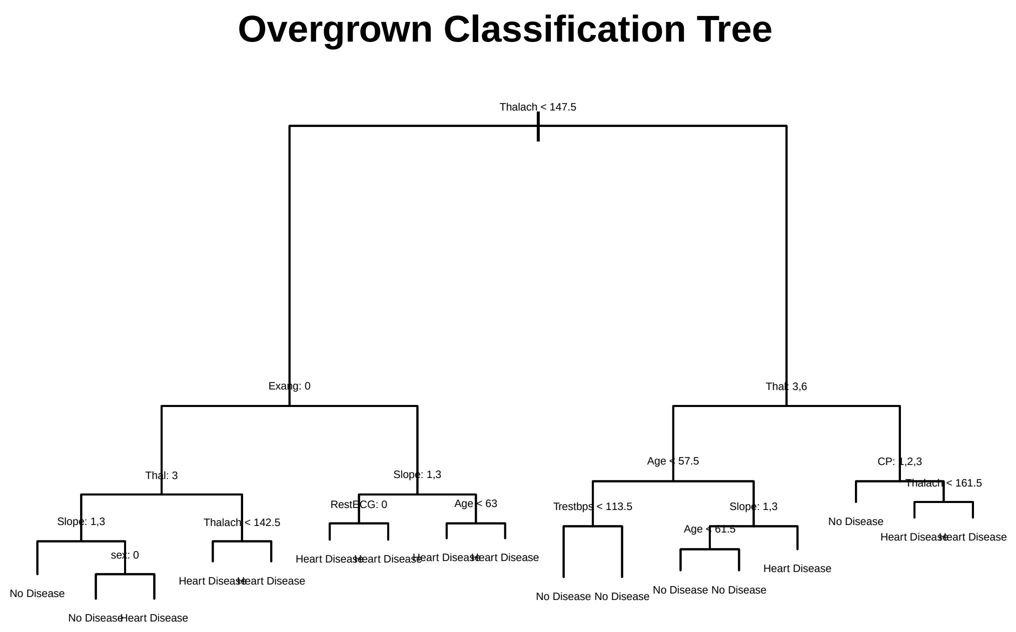
Disease	n
No Disease	171
Heart Disease	200

```r
# split data into train/test set as above
set.seed(2)
index <- sample(1:n, size = 200, replace = FALSE)

train_set <- full[index, ]
test_set <- full[-index, ]
```

The full `heart.RData` data set has a sample size of $n = 371$ and $p = 12$ predictors. There are 171 observations in the No Disease class and 200 observations in the Heart Disease class.

- 1b. Fit a simple classification tree to your data and display the estimated tree, which here we will call the overgrown tree. Note: you can use the function `tree`, just make sure the outcome is encoded as a `factor`.

```r
# fit a simple classification tree
overgrown_tree <- tree(Disease ~ ., train_set)
summary(overgrown_tree)
##
## Classification tree:
## tree(formula = Disease ~ ., data = train_set)
## Variables actually used in tree construction:
## [1] "Thalach"  "Exang"    "Thal"     "Slope"    "sex"      "RestECG"  "Age"
## [8] "Trestbps" "CP"
## Number of terminal nodes:  17
## Residual mean deviance:  0.471 = 86.2 / 183
## Misclassification error rate: 0.11 = 22 / 200

plot(overgrown_tree)
text(overgrown_tree, all = FALSE, pretty = 0, cex = 0.3)
title(main = "Overgrown Classification Tree")
```



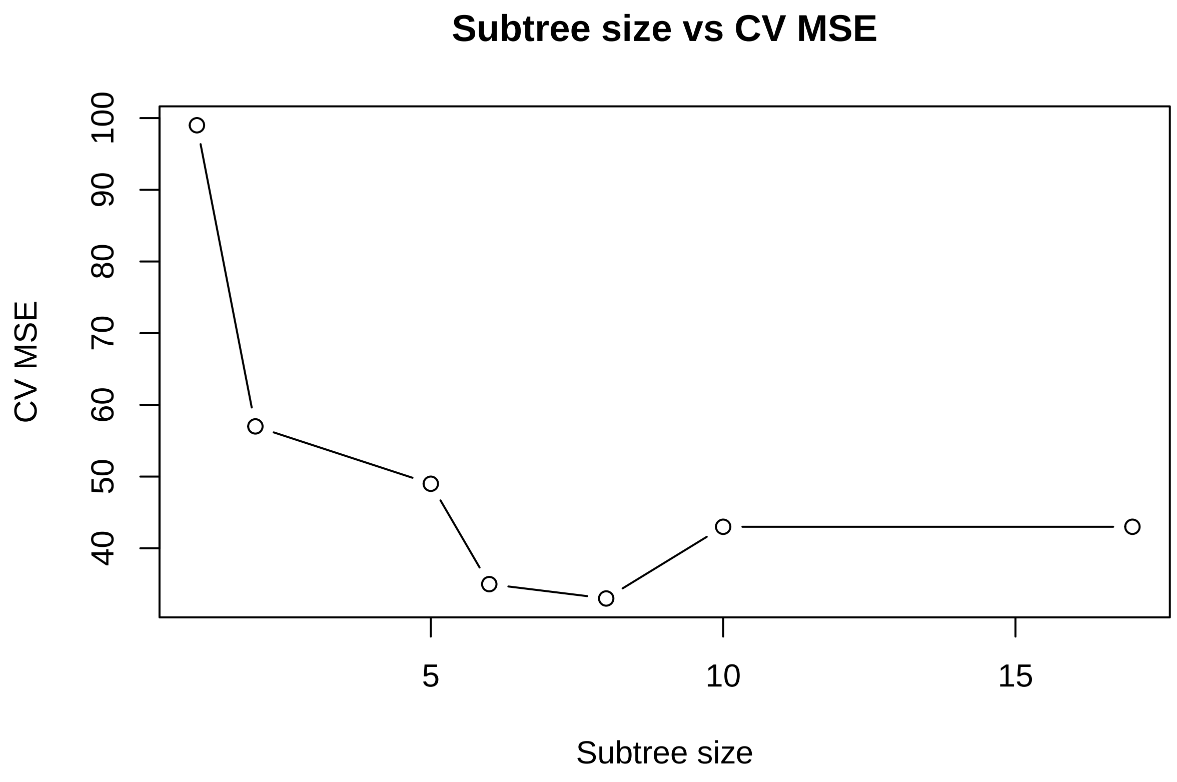
- 1c. Compute the training and test misclassification error. Comment on the output. To compute the predicted output, use the function `predict()` as in the regression setting, but with the additional argument `type = "class"`.

```
# training misclassification error of overgrown tree model
tree_train_y <- predict(overgrown_tree, newdata = train_set, type = "class")
tree_train_MSE <- mean((tree_train_y != train_set$Disease)^2)

# test misclassification error of overgrown tree model
tree_test_y <- predict(overgrown_tree, newdata = test_set, type = "class")
tree_test_MSE <- mean((tree_test_y != test_set$Disease)^2)
```

The training misclassification error of the overgrown classification tree model is 0.12 while the test misclassification error is 0.2280702. As expected, the training misclassification error is smaller than the test misclassification error (due to fitting the model to the training set).

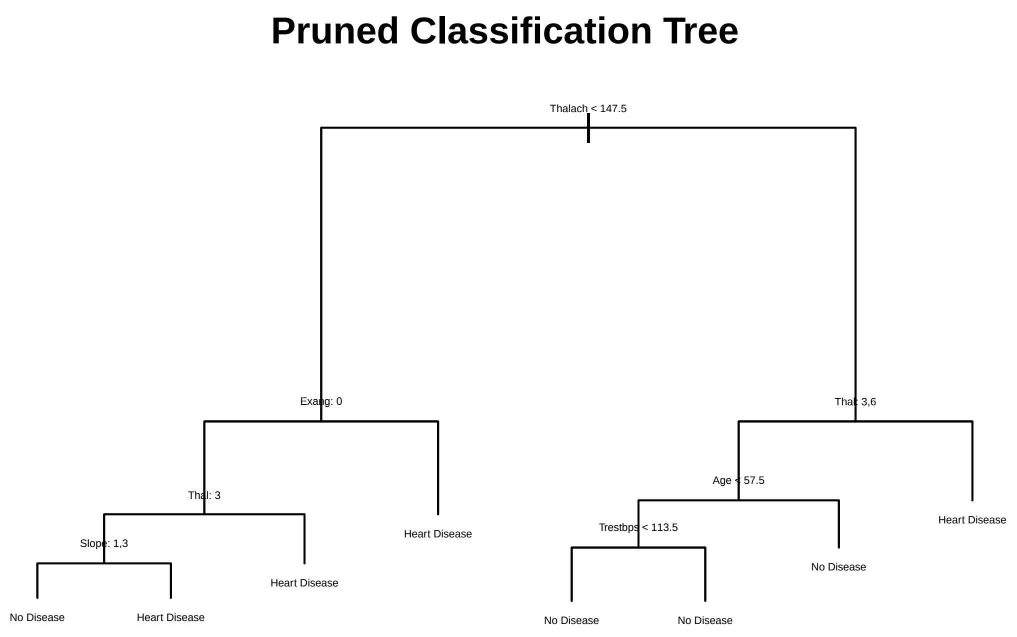
- 1d. Prune the overgrown tree. Make a plot that displays the subtree size against the cross-validated misclassification error that you get from the pruning procedure. Select the subtree that minimizes the cross-validated misclassification error. Plot the **overgrown tree** and highlight the branches of the selected subtree.
  Note: Use `set.seed(2)` before calling `cv.tree` in order to get reproducible results. In `cv.tree()`, use the additional argument `FUN = prune.misclass` to indicate that you want the pruning to be driven by the misclassification error. The *number of misclassified observations* is stored in the variable `dev` of the output of `cv.tree`.

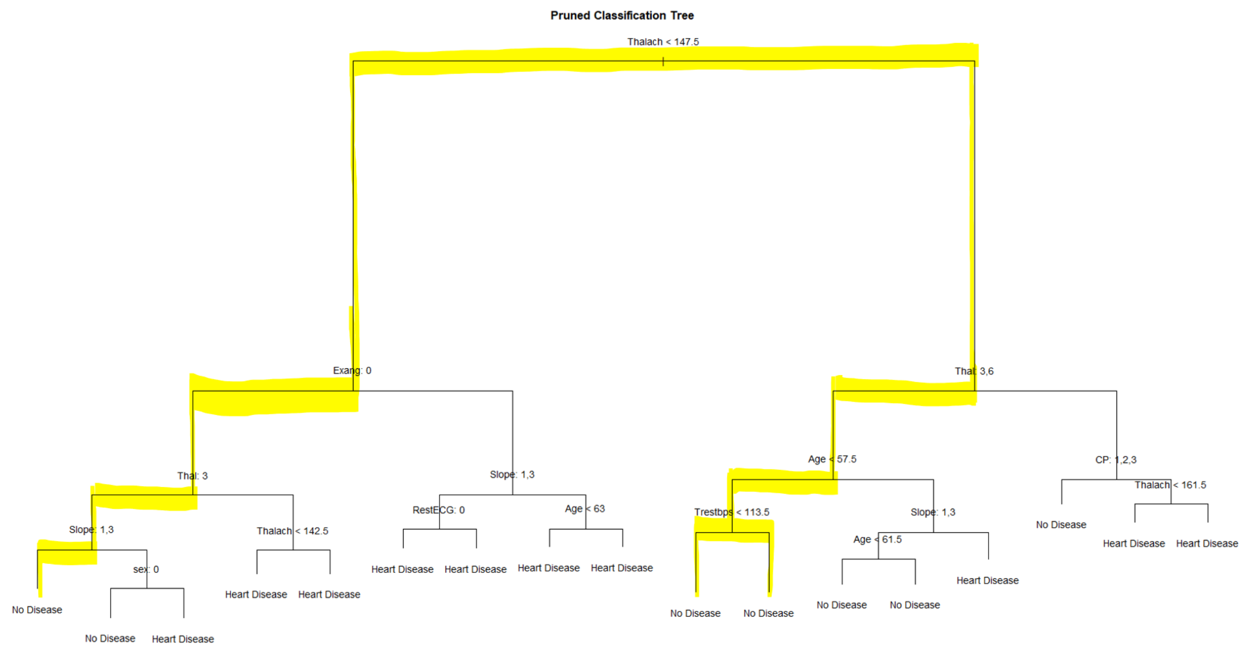
```
# prune overgrown tree
set.seed(2)
cv_tree <- cv.tree(overgrown_tree, FUN = prune.misclass)
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "", ylab = "")
title(main = "Subtree size vs CV MSE",
      xlab = "Subtree size",
      ylab = "CV MSE")
```



```
# select subtree that minimizes CV misclassification error
subtree_size <- cv_tree$size[which(cv_tree$dev == min(cv_tree$dev))]

pruned_tree <- prune.tree(overgrown_tree, best = subtree_size)
summary(pruned_tree)
##
## Classification tree:
## snip.tree(tree = overgrown_tree, nodes = c(9L, 7L, 13L, 17L,
## 5L))
## Variables actually used in tree construction:
## [1] "Thalach"  "Exang"    "Thal"     "Slope"    "Age"      "Trestbps"
## Number of terminal nodes:  8
## Residual mean deviance:  0.6775 = 130.1 / 192
## Misclassification error rate: 0.16 = 32 / 200

plot(pruned_tree)
text(pruned_tree, all = FALSE, pretty = 0, cex = 0.3)
```

```r
title(main = "Pruned Classification Tree")
```





In the plot above, the original overgrown tree model is plotted and the selected subtree branches are highlighted in yellow.

- 1e. Compute the training and test misclassification error of the selected subtree and comment on the results. Relate the results to the notion of bias and variance.

```r
# training misclassification error of pruned tree model
pruned_train_y <- predict(pruned_tree, newdata = train_set, type = "class")
pruned_train_MSE <- mean((pruned_train_y != train_set$Disease)^2)

# test misclassification error of pruned tree model
pruned_test_y <- predict(pruned_tree, newdata = test_set, type = "class")
pruned_test_MSE <- mean((pruned_test_y != test_set$Disease)^2)

tree_MSE <- data.frame(Model = c("Overgrown", "Pruned"),
                       `Training MSE` = c(tree_train_MSE, pruned_train_MSE),
                       `Test MSE` = c(tree_test_MSE, pruned_test_MSE),
                       check.names = FALSE)
kable(tree_MSE, caption = "Pre- vs. Post-Pruning Tree Model Performance")
```

Table 2: Pre- vs. Post-Pruning Tree Model Performance

Model	Training MSE	Test MSE
Overgrown	0.120	0.2280702
Pruned	0.155	0.2631579

In the table above, we see that the training and test misclassification errors of the selected subtree are 0.155 and 0.2631579, respectively, and that the MSE values (for both training and test MSEs) are greater after pruning compared to the original overgrown tree model. This is not a surprising result given the notion of the bias-variance trade-off. Looking at only the training MSE values between the overgrown and pruned tree models, the overgrown model has the lower training MSE value which makes sense because training MSE decreases as model complexity increases. Looking at only the test MSE values, we also see that the overgrown model has the lower test MSE value. The bias-variance trade-off is that increasing model complexity will only result in decreased test MSE values up to a certain level of model complexity, after which, test MSE will begin to increase again as model complexity continues to increase.

Overall, very little accuracy is lost in the test set when comparing the overgrown model with the pruned model despite the reduction in model variance.

- 1f. Now you decide to apply bagging to improve the performance of your tree model. Fit a Bagged Trees model to your training data and compute the training and test misclassification error. Briefly comment on the results. Note: Set the seed with `set.seed(2)` before you run the bagged model.

```r
# fit a bagged tree model
set.seed(2)
bagged_model <- randomForest(Disease ~ ., data = train_set,
                             mtry = p, importance = TRUE)

# training misclassification error of bagged tree model
bagged_train_y <- predict(bagged_model, newdata = train_set)
bagged_train_MSE <- mean((bagged_train_y != train_set$Disease)^2)

# test misclassification error of bagged tree model
bagged_test_y <- predict(bagged_model, newdata = test_set)
bagged_test_MSE <- mean((bagged_test_y != test_set$Disease)^2)
```

```r
tree_MSE[nrow(tree_MSE) + 1, ] <- c("Bagged", bagged_train_MSE, bagged_test_MSE)
kable(tree_MSE, caption = "Tree model performances")
```

Table 3: Tree model performances

Model	Training MSE	Test MSE
Overgrown	0.12	0.228070175438596
Pruned	0.155	0.263157894736842
Bagged	0	0.228070175438596

The training MSE for the bagged trees model decreased to 0 while the test MSE of the bagged trees model is on par with the overgrown tree model (overgrown tree test MSE: 0.228070175438596, bagged trees test MSE: 0.228070175438596), which is less than the test MSE value of the pruned tree model.

- 1g. Fit a random forest model (with the number of randomly selected features $m = p/3$) to your training data and compute the training and test misclassification error. Briefly comment on the results. Note: Set the seed with `set.seed(2)` before you run the random forest model.

```r
set.seed(2)
rf_model <- randomForest(Disease ~ ., data = train_set,
                         mtry = p/3, importance = TRUE)

# training misclassification error of bagged tree model
rf_train_y <- predict(rf_model, newdata = train_set)
rf_train_MSE <- mean((rf_train_y != train_set$Disease)^2)

# test misclassification error of bagged tree model
rf_test_y <- predict(rf_model, newdata = test_set)
rf_test_MSE <- mean((rf_test_y != test_set$Disease)^2)

tree_MSE[nrow(tree_MSE) + 1, ] <- c("Random Forest", rf_train_MSE, rf_test_MSE)
kable(tree_MSE, caption = "Tree model performances")
```

Table 4: Tree model performances

Model	Training MSE	Test MSE
Overgrown	0.12	0.228070175438596
Pruned	0.155	0.263157894736842
Bagged	0	0.228070175438596
Random Forest	0	0.210526315789474

The training MSE for the random forest model is 0 while the test MSE has decreased to 0.210526315789474. Comparing the four models, it would appear that the random forest model performs the best for the training and test sets.

- 1h. If you run the bagged and the random forest models multiple times on the same data (without fixing the seed) you obtain different results. Explain what are the sources of randomness in the two models.
  Both bagged trees and random forests builds upon bootstrapped data for model fitting. Bootstrapping

is a resampling method that randomly samples the training data (with replacement) to generate new training data sets. This is one source of randomness to the two modeling algorithms as the resampling method may generate completely different data sets each time we run the models without fixing the seed.

The main difference between random forests and bagged trees is that random forests do not look at the full set of p predictors but instead, randomly select a subset of m predictors from the full set of p predictors. This is the main source of randomness in the random forest models if run multiple times on the same data without fixing the seed. The trees constructed will have different subset of predictors from each other. With bagged trees, because the full set of p predictors is used each time, the top nodes of the generated trees will be similar to each other. This is not the case with random forests since a subset of predictors will be randomly selected to generate the trees, thus making the top nodes potentially different each time.

- 1i. Fit a boosted tree model to your training data. Use the following parameters: 500 trees, 2 splits for each tree, and a 0.1 shrinkage factor. Compute the training and test misclassification error. Comment on the results.
  Note 1: Use the function `gbm` with the argument `distribution = "bernoulli"` to indicate that this is a classification problem. The function `gbm` requires that the outcome variable be a number in {0,1} - you will need to generate such a variable from the categorical outcome.
  Note 2: Use the function `predict()` with the argument `type = "response"`. This will return the estimated probability of the outcome being 1. Use the Bayes rule to get the estimated labels from the estimated probabilities.
  Note 3: Set the seed with `set.seed(2)` before you run the boosted model. Although we haven't seen this in class, there is a source of randomness in a boosted model. If you are curious, check out the documentation of the function `gbm`.

```r
levels(train_set$Disease) <- c(0, 1)
train_set_num <- train_set %>%
  mutate(Disease = as.integer(as.character(Disease)))

levels(test_set$Disease) <- c(0, 1)
test_set_num <- test_set %>%
  mutate(Disease = as.integer(as.character(Disease)))

set.seed(2)
boosted_model <- gbm(Disease ~ ., train_set_num, distribution = "bernoulli",
                     n.trees = 500, interaction.depth = 2, shrinkage = 0.1)

# training misclassification error of boosted tree model
boosted_train_y <- predict(boosted_model, newdata = train_set_num,
                           type = "response")
boosted_train_MSE <- mean((boosted_train_y - train_set_num$Disease)^2)

# test misclassification error of boosted tree model
boosted_test_y <- predict(boosted_model, newdata = test_set_num,
                          type = "response")
boosted_test_MSE <- mean((boosted_test_y - test_set_num$Disease)^2)

tree_MSE[nrow(tree_MSE) + 1, ] <- c("Boosted", boosted_train_MSE, boosted_test_MSE)

tree_MSE <- tree_MSE %>%
  mutate_at(c("Training MSE", "Test MSE"), as.numeric)
kable(tree_MSE, caption = "Tree model performances", digits = 3)
```

Table 5: Tree model performances

Model	Training MSE	Test MSE
Overgrown	0.120	0.228
Pruned	0.155	0.263
Bagged	0.000	0.228
Random Forest	0.000	0.211
Boosted	0.008	0.164

The training MSE for the boosted tree model increased to 0.0075511 while the test MSE has decreased to 0.1643323. Comparing the five models, the boosted tree model had the best performance with the lowest training and test MSE values.

2. Non-linear Regression

- 2a. Use the `runif()` function to generate a predictor X of length $n = 50$ with values in the interval [-1, 1], and use the `rnorm()` function to generate a noise vector ϵ of length $n = 50$, mean = 0, and standard deviation = 0.1.
- 2b. Generate a response vector Y of length $n = 50$ according to the model $Y = f(X) + \epsilon$, with $f(X) = 3 - 2X + 3 * X^3$.

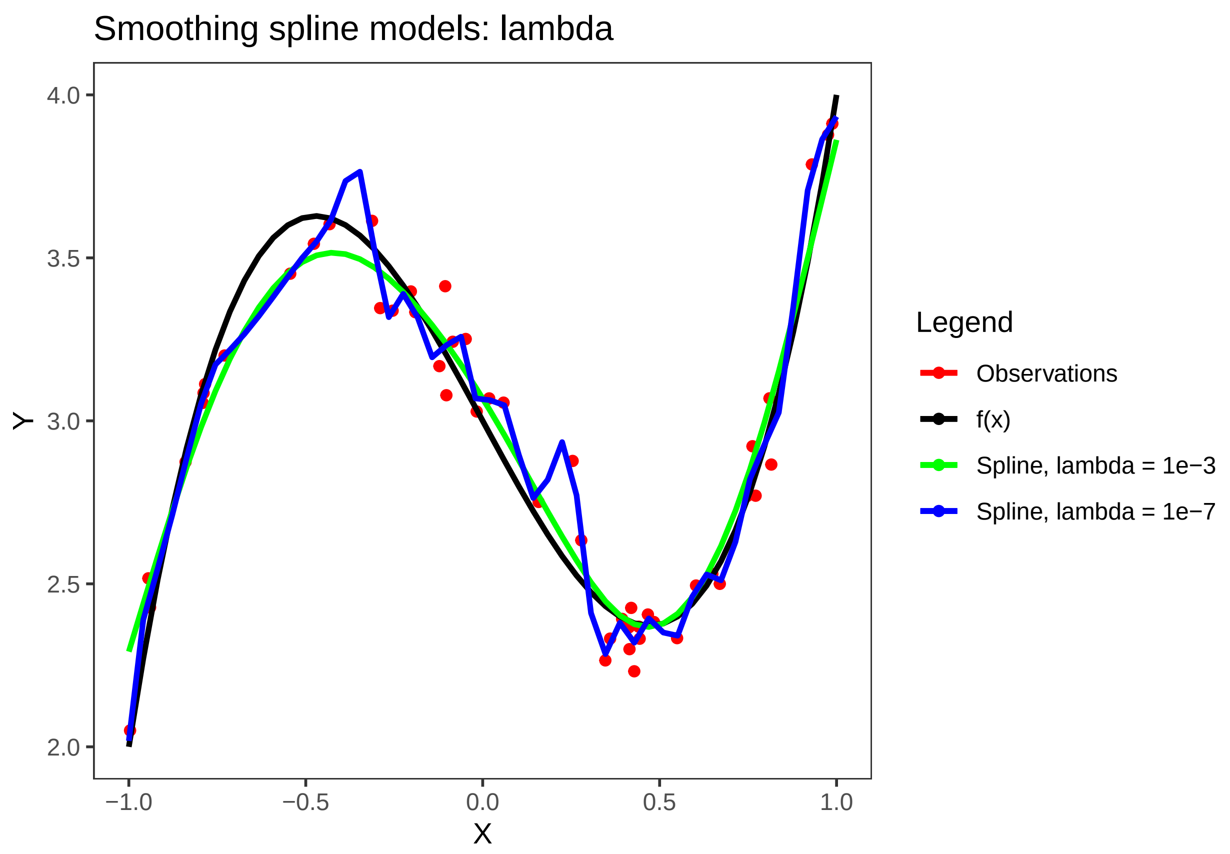
```
X <- runif(n = 50, min = -1, max = 1)
e <- rnorm(n = 50, mean = 0, sd = 0.1)
f <- 3 - (2 * X) + (3 * (X^3))
Y <- f + e

df <- data.frame(X = X, f = f, Y = Y)
```

- 2c. Fit two smoothing spline models, $\hat{f}_{\lambda=1e-3}$ and $\hat{f}_{\lambda=1e-7}$, with $\lambda = 1e-3$ and $\lambda = 1e-7$, respectively. Use the function `smooth.spline` introduced in class.

```
spline3_model <- smooth.spline(X, Y, lambda = 1e-3)
spline7_model <- smooth.spline(X, Y, lambda = 1e-7)
```

- 2d. Make one plot that displays:
  - A scatter plot of the generated observations with X on the x-axis and Y on the y-axis;
  - The true function f evaluated on a dense grid of values in the interval [-1, 1];
  - The predicted functions $\hat{f}_{\lambda=1e-3}$ and $\hat{f}_{\lambda=1e-7}$ evaluated on a dense grid of values in the interval [-1, 1] (use the function `predict` to produce the predictions).
    Comment on the estimated functions $\hat{f}_{\lambda=1e-3}$ and $\hat{f}_{\lambda=1e-7}$ and the role of λ.

```
colors <- c("Observations" = "red",
            "f(x)" = "black",
            "Spline, lambda = 1e-3" = "green",
            "Spline, lambda = 1e-7" = "blue")

scatterplot <- ggplot(df) +
  geom_point(aes(x = X, y = Y, col = "Observations")) +
  theme_bw() +
  theme(panel.grid.minor = element_blank(),
        panel.grid.major = element_blank())

dense_grid <- seq(-1, 1, length.out = 50)
f_grid <- 3 - (2 * dense_grid) + (3 * (dense_grid^3))
fit_spline3 <- predict(spline3_model, dense_grid)
fit_spline7 <- predict(spline7_model, dense_grid)

scatterplot +
  geom_line(aes(x = dense_grid, y = f_grid, col = "f(x)"), size = 1) +
  geom_line(aes(x = fit_spline3[[1]], y = fit_spline3[[2]],
                col = "Spline, lambda = 1e-3"), size = 1) +
  geom_line(aes(x = fit_spline7[[1]], y = fit_spline7[[2]],
                col = "Spline, lambda = 1e-7"), size = 1) +
  scale_color_manual(name = "Legend",
                     values = colors,
                     labels = c("Observations", "f(x)",
                                "Spline, lambda = 1e-3",
                                "Spline, lambda = 1e-7")) +
  labs(title = "Smoothing spline models: lambda")
```



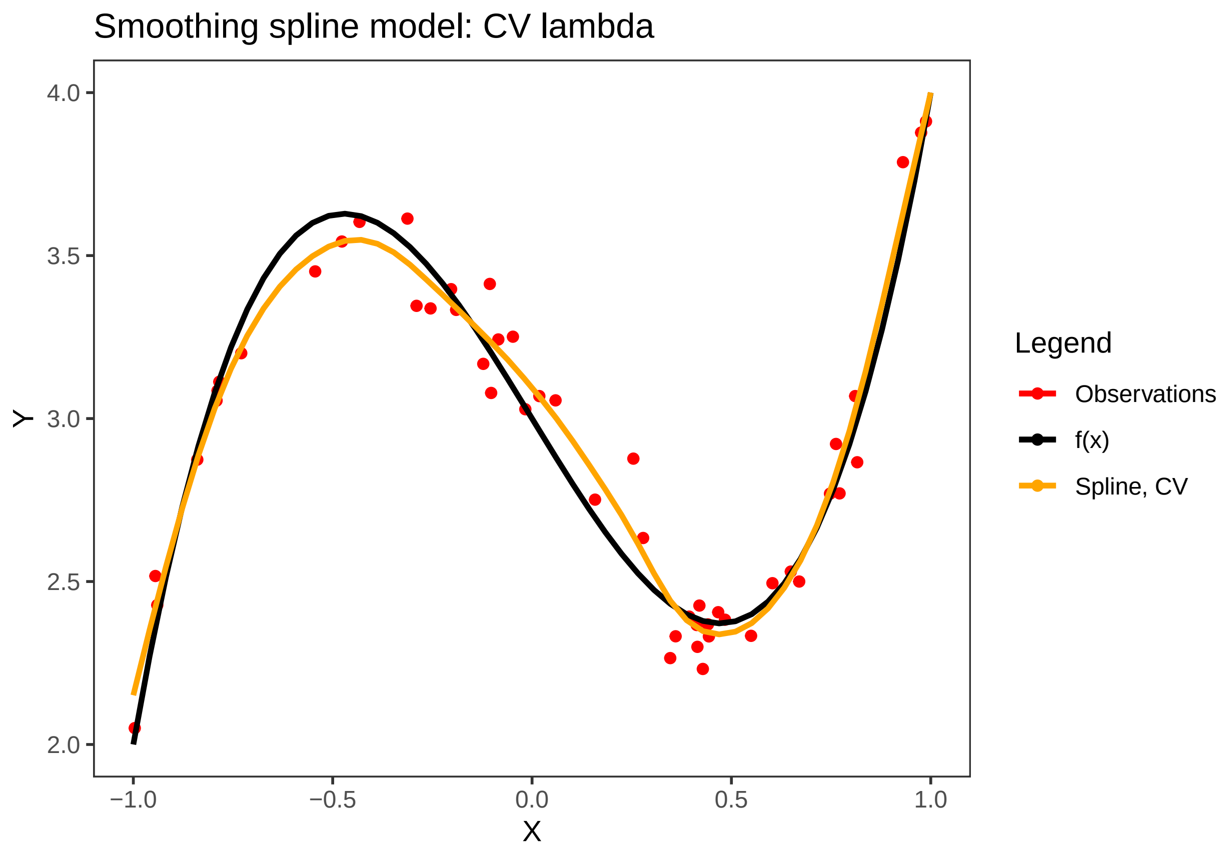
The fitted smoothing spline model $\hat{f}_{\lambda=1e-3}$ has a smoother fit to the generated observations compared to $\hat{f}_{\lambda=1e-7}$, which has a more jagged fit to the generated observations. This suggests that smaller λ values will result in model overfitting while larger λ will result in model underfitting.

- 2e. Fit a smoothing spline model $\hat{f}_{CV}$ with a cross-validated choice of lambda. Use `smooth.spline` with argument `cv = TRUE`.

```
spline_cv_model <- smooth.spline(X, Y, cv = TRUE)
fit_splineCV <- predict(spline_cv_model, dense_grid)
```

- 2f. Make one plot that displays:
  - A scatter plot of the generated observations with X on the x-axis and Y on the y-axis;
  - The true function $f(x)$ evaluated on a dense grid of values in the interval [-1, 1];
  - The predicted function $\hat{f}_{CV}$ evaluated on a dense grid of values in the interval [-1, 1] (use the function `predict` to produce the predictions).

```
colors <- c("Observations" = "red",
            "f(x)" = "black",
            "Spline, CV" = "orange")

scatterplot +
  geom_line(aes(x = dense_grid, y = f_grid, col = "f(x)"), size = 1) +
  geom_line(aes(x = fit_splineCV[[1]], y = fit_splineCV[[2]],
                col = "Spline, CV"), size = 1) +
  scale_color_manual(name = "Legend",
                     values = colors,
                     labels = c("Observations", "f(x)", "Spline, CV")) +
  labs(title = "Smoothing spline model: CV lambda")
```


Smoothing spline model: CV lambda
Y
X
Legend
Observations
f(x)
Spline, CV