Homework: Ch 02

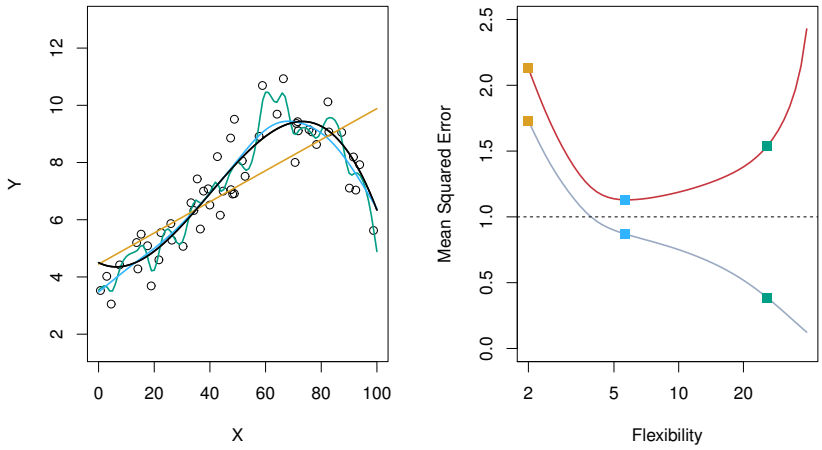
STAT 4510/7510

Due Monday, February 7, 11:59 pm

Instructions: Please list your name and student number clearly. In order to receive credit for a problem, your solution must show sufficient detail so that the grader can determine how you obtained your answer.

Submit a single pdf file for your final outcome. All `R` code should be included, as well as all output produced. Upload your work to the Canvas course site.

Problem 1



The plot in the left-hand side shows the simulated data, shown as empty circles, generated by the underlying true function f, shown as the black curve. Three estimates of f are also displayed:

- The linear regression fit (orange curve)
- The smoothing spline fit with low degrees of freedom (skyblue curve)
- The smoothing spline fit with high degrees of freedom (green curve)

The plot in the right-hand side shows the training MSE (grey curve), test MSE (red curve), and $Var(\epsilon)$, that is the minimum possible test MSE or irreducible error (dashed horizontal line). Squares represent the training and test MSEs for the three fits mentioned above. Answer the following questions.

a) Explain the difference between training MSE and test MSE.

b) Explain why two curves in the right-hand panel show different trend.

c) What causes the large test MSE for the liner regression fit?

d) What causes the large test MSE for the smoothing spline fit with high degrees of freedom?

e) How is it possible that the training MSEs for the second and third models are lower than the irreducible error?

Problem 2

Complete Chapter 2, problem 8 (p. 54).

- You should be able to find `College.csv` file in Canvas course website.
- The question in part (a) asks to read the data file using `read.csv()` function. When you do this, try to add the option `stringAsFactors=TRUE` as follows.

```
college <- read.csv('College.csv', stringsAsFactors=TRUE)
```

This additional option makes `R` recognize data fields with strings as factor variables so that you don't need to transform variable types from `char` to `factor` after you load the data.

- In part (b), you don't need to show the outcome of `fix()` function.
- You can skip part (c) vi.

Problem 3

In this problem, data points with two class labels will be simulated. There are two predictors X_1 and X_2. Using this data, you will learn how to use `R` for the KNN classifier.

Data simulation

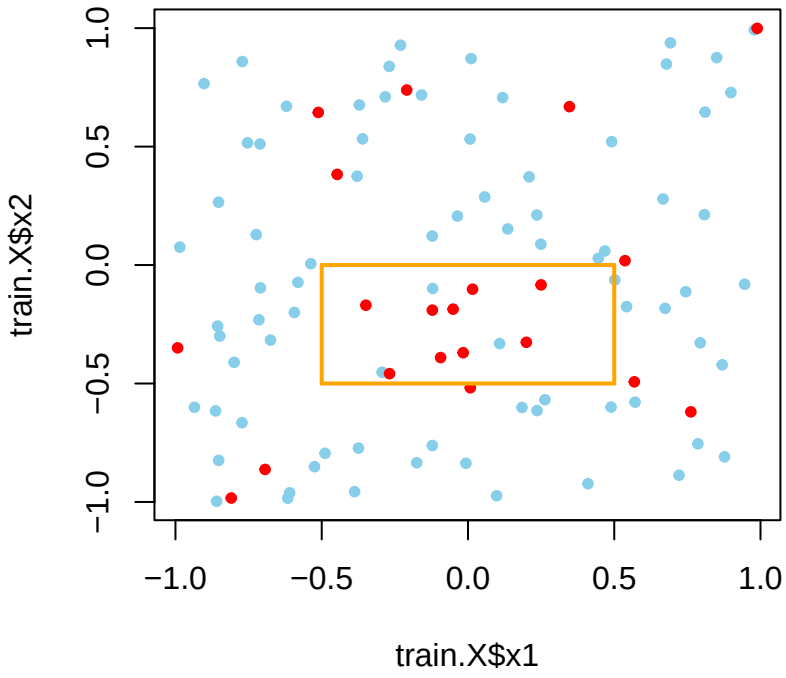
- There are 100 training data points here. The data frame `train.X` includes the locations of each data point.
- Each data point is associated with a class label, either `TRUE` or `FALSE`, which is recorded in `train.Y`. The data points with class label `TRUE` are plotted in red dot, and those with `FALSE` are plotted in bluish color.
- In the plot, you can see the rectangle in orange lines. The underlying probabilities of $Y =$ `TRUE` inside the rectangle is 0.9 and those outside the rectangle is 0.1. That is, there is 90% of chance that an observed data point inside the rectangle belongs to `TRUE`, and 10% of chance that a data point outside of the rectangle belongs to `TRUE`. The training data shown in the plot below is simulated from these underlying probabilities.
- Try to follow each line of codes below. To do that, try to run each line in `Console`, and observe the outcome. By doing that, you will be able to understand what the corresponding code is doing. In addition, investigate unfamiliar functions by `?functionName`.

```
set.seed(80)
train.X <- data.frame(x1 = runif(n=100,-1,1), x2 = runif(n=100,-1,1))
# Randomly select 100 locations of (x1, x2)

prob <- ifelse( (-0.5 < train.X$x1) & (train.X$x1 < 0.5)
                & (-0.5 < train.X$x2) & (train.X$x2 < 0), 0.9, 0.1)
# If (x1, x2) is inside the rectangle, prob is 0.9, otherwise, prob is 0.1

train.Y <- as.factor(runif(n=100) < prob)
# Simulate class labels according to prob

colors <- c("skyblue","red")
plot(train.X$x1, train.X$x2, pch=20, col=colors[factor(train.Y)])
segments(-0.5, -0.5, 0.5, -0.5, col="orange", lwd=2)
segments(-0.5, -0.5, -0.5, 0, col="orange", lwd=2)
segments(-0.5, 0, 0.5, 0, col="orange", lwd=2)
segments(0.5, -0.5, 0.5, 0, col="orange", lwd=2)
```



Answer the following questions.

a) What is the Bayes error rate involved in this problem? In other words, when you generate very large number of data points, with which proportion, in average, would you expect to predict the class labels incorrectly by the Bayes classifier?

b) In the middle of the codes above, the function `runif(n=100)` is used to draw 100 random numbers from the *Uniform(0,1)* distribution. Those numbers have been compared to the values of `prob` (the vector of length 100 which includes $Pr(Y_i = TRUE|X_i)$ for $i = 1, 2, \cdots, 100$) to produce class labels of each point. Applying similar ideas, write `R` codes to produce outcomes of 100 tossed coins (head = `TRUE`, tail = `FALSE`). Assume that those coins are all fair (the same probabilities of head and tail) and independent. Show the outcome (the number of heads and tails) and see if it is a reasonable outcome for this experiment.

KNN classifier in `R`

- The following code prepares grid points in (X_1, X_2) plane. We will predict the class label of each of these grid points using the KNN classifier with the given training data.

```
xGrid.1d <- seq(-1,1,0.02)
nx <- length(xGrid.1d)
xGrid.2d <- data.frame(x1=rep(xGrid.1d, each=nx), x2=rep(xGrid.1d,times=nx))
```

- In what follows, we use the `knn` function to run the KNN classifier. To do this, `class` package is necessary. I believe `class` package is provided by default when you install `R`, so you may not need to install it separately. But, if you see an error message saying that there is no such library, you will have to install it by `install.packages("class")` before you run the following code.

- `knn` function can have following input arguments.

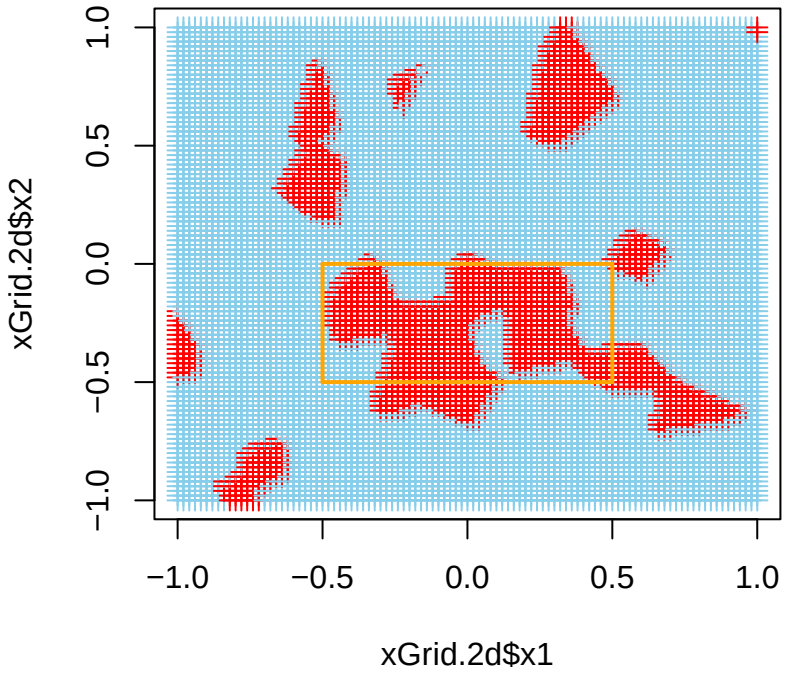
  - `train`: X of a training data set
  - `test`: X of a test data set (the new observations of X for which we want to predict Y)
  - `cl`: Y (class labels) of a training data set
  - `k`: the number of nearest neighbors considered (default is set as 1)
  - Other arguments: please refer to function description by `?knn`.

- We first try KNN with $k = 1$.

```
library(class)
set.seed(1)
# Set a random seed because if several observations are tied as nearest neighbors,
# then R will randomly break the tie. Setting a seed number makes us obtain
# the same prediction outcomes when we repeatedly run this code.

knn.pred <- knn(train=train.X, test=xGrid.2d, cl=train.Y, k=1)
# The predicted class labels of each data points in xGrid.2d will be stored
# in knn.pred object

plot(xGrid.2d$x1, xGrid.2d$x2, pch=3, col=colors[factor(knn.pred)])
segments(-0.5, -0.5, 0.5, -0.5, col="orange", lwd=2)
segments(-0.5, -0.5, -0.5, 0, col="orange", lwd=2)
segments(-0.5, 0, 0.5, 0, col="orange", lwd=2)
segments(0.5, -0.5, 0.5, 0, col="orange", lwd=2)
```



- The ideal prediction outcome would be assigning `TRUE` (red dots) for all grid points inside the orange rectangle and assigning `FALSE` (blue dots) for all those outside of the orange rectangle.

Answer the following questions.

c) What do you observe from this outcome from KNN with $k = 1$? Do you satisfy this prediction outcome?

d) Run the last chunk of codes with several increasing k values, and observe how the decision boundary and corresponding prediction outcome changes.

e) Can you tell briefly which k value seems to produce the best prediction outcome? (I don't expect you to compute exact test performance. You can just describe what you observe from plots generated.)