Homework #2: Resampling

Hyunsuk Ko

Due: Wed Sept 14 | 11:45am

DS 6030 | Fall 2022 | University of Virginia

Required R packages and Directories

```
data.dir = 'https://mdporter.github.io/DS6030/data/' # data directory
library(R6030)     # functions for DS-6030
library(tidyverse) # functions for data manipulation
library(splines)
library(FNN)
rm(list = ls())
```

Problem 1: Bootstrapping

Bootstrap resampling can be used to quantify the uncertainty in a fitted curve.

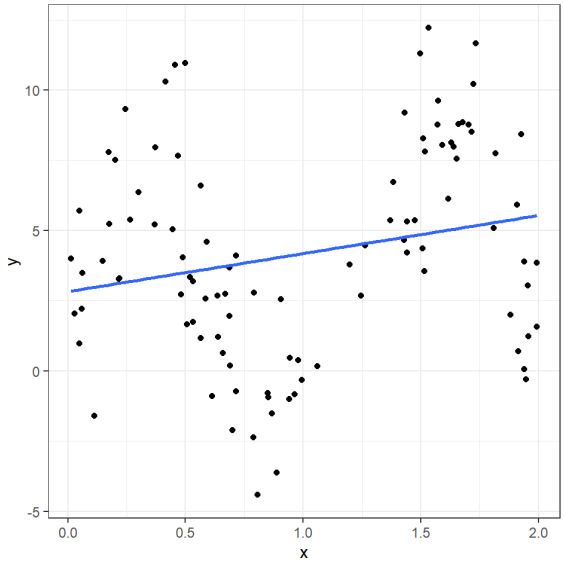
a. Create a set of functions to generate data from the following distributions:

$$
\begin{aligned}
X &\sim \mathcal{U}(0, 2) \qquad \text{Uniform in } [0, 2] \\
Y &= 1 + 2x + 5\sin(5x) + \epsilon \\
\epsilon &\sim \mathcal{N}(0,\, \sigma = 2.5)
\end{aligned}
$$

```
sim_x <- function(n) runif(n, min = 0, max = 2)
sim_y <- function(x){               # generate Y|X from N{f(x),sd}
  n = length(x)
  f <- function(x) 1 + 2 * x + 5 * sin(5 * x)
  f(x) + rnorm(n, mean = 0 , sd = 2.5)  # error
}
```

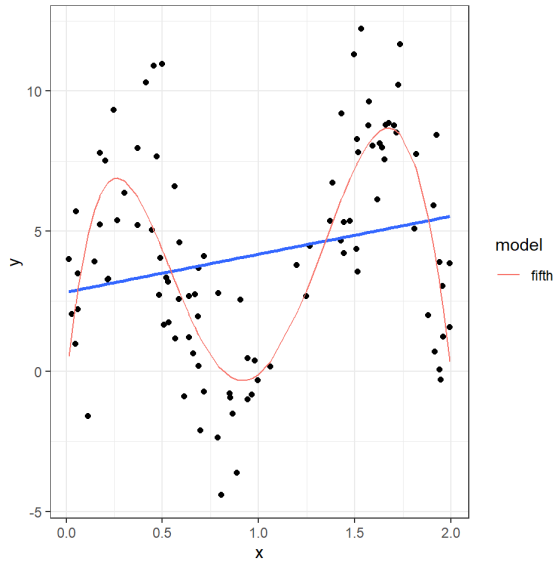
b. Simulate $n = 100$ realizations from these distributions. Produce a scatterplot and draw the true regression line $f(x) = E[Y \mid X = x]$. Use `set.seed(211)` prior to generating the data.

```
set.seed(211)

n <- 100

x <- sim_x(n)    # get x values
y <- sim_y(x)

data_train <- tibble(x, y)               # training data tibble

gg_myplot <- ggplot(data_train, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)

gg_myplot
```



c. Fit a 5th degree polynomial. Produce a scatterplot and draw the *estimated* regression curve.

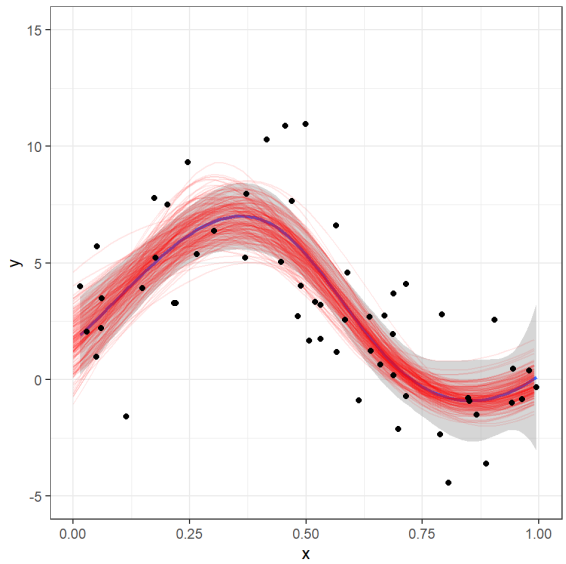
```
xseq <- x
xeval <- tibble(x = xseq)

m5 <- lm(y~poly(x, degree=5), data = data_train)
yhat5 <- predict(m5, newdata =xeval)

fifth.data <- tibble(x = xseq, fifth = yhat5) %>%  # long data
  pivot_longer(-x, names_to="model", values_to="y")

gg_myplot +
  geom_line(data=fifth.data, aes(color=model))
```



d. Make 200 bootstrap samples. For each bootstrap sample, fit a 5th degree polynomial and make predictions at `eval_pts = seq(0, 2, length=100)`

- Set the seed (use `set.seed(212)`) so your results are reproducible.
- Produce a scatterplot with the original data and add the 200 bootstrap curves

```
set.seed(212)

n <- length(x) # length of observed data
M <- 200 # number of bootstrap samples
p <- numeric(M) # initialize vector for test statistic
data_eval<- tibble(x = seq(0,2,length=100)) # evaluation points
YHAT <- matrix(NA, nrow(data_eval), M)
kts_bdry <- c(-.2,1.2)

for (m in 1:M) {
  ind = sample(n, replace = TRUE)  # sample indices with replacement
  xboot = x[ind]                   # bootstrap sample
  p[m] = mean(xboot)
  m_boot = lm(y ~ bs(x, df = 5, Boundary.knots = kts_bdry)-1, data = data_train
[ind,]) # fit bootstrap data
  # predict from bootstrap model
  YHAT[,m] = predict(m_boot, data_eval)
}
#print(YHAT)

data_fit <- as_tibble(YHAT) %>% # convert matrix into tibble
  bind_cols(data_eval) %>%      # add the eval points
  pivot_longer(-x, names_to = "simulation", values_to = "y") # convert to long f
ormat

ggplot(data_train, aes(x,y)) +
  geom_smooth(method = "lm", formula = as.formula('y~bs(x, df=5, deg=3, Boundar
y.knots = kts_bdry)-1')) +
  geom_line(data=data_fit, color="red", alpha=.1, aes(group=simulation)) +
  geom_point() +
  scale_x_continuous(limits = c(0, 1)) +
  scale_y_continuous(limits = c(-5,15))
```



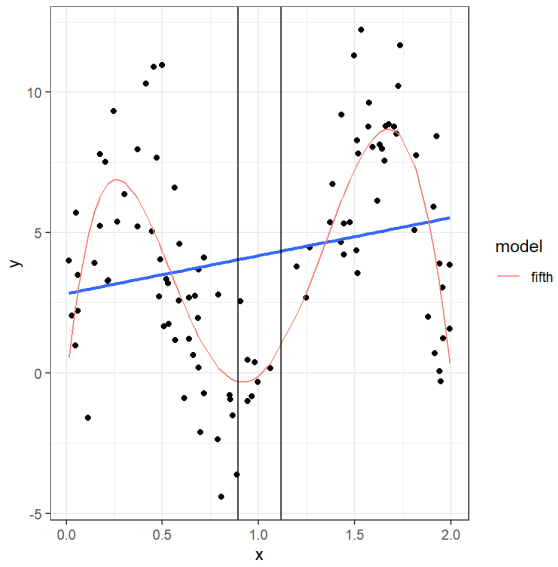
e. Calculate the pointwise 95% confidence intervals from the bootstrap samples. That is, for each $x \in \text{eval\_pts}$, calculate the upper and lower limits such that only 5% of the curves fall outside the interval at x.

- Remake the plot from part *c*, but add the upper and lower boundaries from the 95% confidence intervals.

```
quantile(p, probs = c(.025, .975)) # 95% bootstrap interval
```

```
#>   2.5%  97.5%
#> 0.8938 1.1181
```

```
gg_myplot +
  geom_line(data=fifth.data, aes(color=model)) +
  geom_vline(xintercept = quantile(p, probs = c(.025, .975))[1]) +
  geom_vline(xintercept = quantile(p, probs = c(.025, .975))[2])
```



Problem 2: V-Fold cross-validation with k nearest neighbors

Run 10-fold cross-validation on the data generated in part 1b to select the optimal k in a k-nearest neighbor (kNN) model. Then evaluate how well cross-validation performed by evaluating the performance on a large test set. The steps below will guide you.

a. Use 10-fold cross-validation to find the value of k (i.e., neighborhood size) that provides the smallest cross-validated MSE using a kNN model.

- Search over $k = 3, 4, \ldots, 40$.
- Use `set.seed(221)` prior to generating the folds to ensure the results are replicable.
- Show the following:
  - the optimal k (as determined by cross-validation)
  - the corresponding estimated MSE
  - produce a plot with k on the x-axis and the estimated MSE on the y-axis (optional: add 1-standard error bars).
- Notation: The k is the tuning paramter for the kNN model. The $v = 10$ is the number of folds in V-fold cross-validation. Don't get yourself confused.

```
knn_eval <- function(k, data_train, data_test){
  # fit model and eval on training data
  knn = knn.reg(data_train[,'x', drop=FALSE],
                y = data_train$y,
                test = data_train[,'x', drop=FALSE],
                k = k)
  r = data_train$y- knn$pred      # residuals on training data
  mse.train = mean(r^2)            # training MSE

  # fit model and eval on test data
  knn.test = knn.reg(data_train[,'x', drop=FALSE],
                     y = data_train$y,
                     test=data_test[,'x', drop=FALSE],
                     k=k)
  r.test = data_test$y - knn.test$pred # residuals on test data
  mse.test = mean(r.test^2)          # test MSE
    # results
  edf = nrow(data_train)/k         # effective dof (edof)
  tibble(k=k, edf=edf, mse.train, mse.test)
}
```

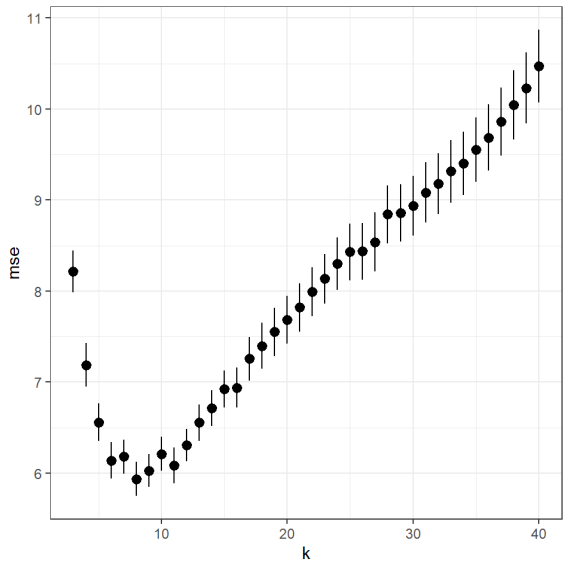
```
n <- nrow(data_train) # number of training observations (= 100)
n.folds <- 10 # number of folds for cross-validation
set.seed(221)
fold <- sample(rep(1:n.folds, length = n)) # vector of fold labels
data_knn = tibble()

# Iterate over folds
for (j in 1:n.folds) {
  # Set training / val data
  val = which(fold == j) # indices of holdout / validation data
  train = which(fold != j) # indices of fitting / training data
  n.val = length(val) # number of observations in validation

  # Fit and evaluate models
  K = seq(3,40,1)
  for(k in K){
    tmp = knn_eval(k, data_train= slice(data_train,train), data_test= slice(data
_train, val))
    data_knn = bind_rows(data_knn, tmp)
  }
}

df_result <- data_knn %>% group_by(k) %>%
  summarize(mse = mean(mse.test), sd = sd(mse.test), se = sd / sqrt(n)) %>%
  arrange(mse)
df_result
```

k <dbl>	mse <dbl>	sd <dbl>	se <dbl>
8	5.937	1.885	0.1885
9	6.028	1.788	0.1788

k <dbl>	mse <dbl>	sd <dbl>	se <dbl>
11	6.085	1.961	0.1961
6	6.139	2.000	0.2000
7	6.181	1.866	0.1866
10	6.211	1.879	0.1879
12	6.308	1.781	0.1781
13	6.554	1.981	0.1981
5	6.559	2.078	0.2078
14	6.714	1.963	0.1963

1-10 of 38 rows Previous **1** 2 3 4 Next

```
ggplot(df_result, aes(x=k, y = mse)) +
  geom_pointrange(aes(ymin = mse-se, ymax = mse+se))
```



b. The k (number of neighbors) in a kNN model determines the effective degrees of freedom *edf*. What is the optimal *edf*? Be sure to use the correct sample size when making this calculation. Produce a plot similar to that from part *a*, but use *edf* (effective degrees of freedom) on the x-axis.

: Our optimal *edf* = 11.250

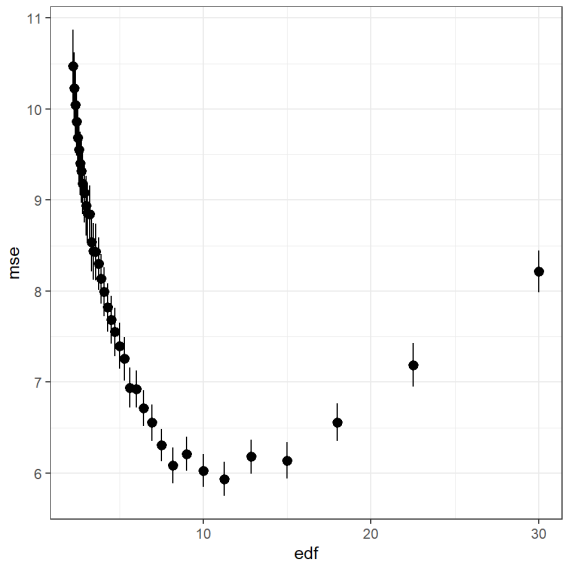
```
df_edf <- df_result %>%
  mutate(edf = 90 / k)
df_edf
```

k <dbl>	mse <dbl>	sd <dbl>	se <dbl>	edf <dbl>
8	5.937	1.885	0.1885	11.250
9	6.028	1.788	0.1788	10.000

k <dbl>	mse <dbl>	sd <dbl>	se <dbl>	edf <dbl>
11	6.085	1.961	0.1961	8.182
6	6.139	2.000	0.2000	15.000
7	6.181	1.866	0.1866	12.857
10	6.211	1.879	0.1879	9.000
12	6.308	1.781	0.1781	7.500
13	6.554	1.981	0.1981	6.923
5	6.559	2.078	0.2078	18.000
14	6.714	1.963	0.1963	6.429

1-10 of 38 rows Previous **1** 2 3 4 Next

```
ggplot(df_edf, aes(x = edf, y = mse)) +
  geom_pointrange(aes(ymin = mse-se, ymax = mse+se))
```



c. After running cross-validation, a final model fit from *all* of the training data needs to be produced to make predictions. What value of k would you choose? Why?

As k = 8 gives the lowest mean test MSE, this will be our optimal k.

d. Now we will see how well cross-validation performed. Simulate a test data set of 50000 observations from the same distributions. Use `set.seed(223)` prior to generating the test data.

- Fit a set of kNN models, using the full training data, and calculate the mean squared error (MSE) on the test data for each model. Use the same k values in *a*.
- Report the optimal k, the corresponding *edf*, and MSE based on the test set.

: Our optimal $k = 13$, *edf* = 7.692, and test MSE = 7.109

```
ntest <- 50000                          # Number of test samples
set.seed(223)                           # set *different* seed
xtest = sim_x(ntest)                    # generate test X's
ytest = sim_y(xtest)             # generate test Y's
data_test = tibble(x=xtest, y=ytest)    # test data
```

```
data_knn2 = tibble()
K = seq(3,40,1)
for(k in K){
  tmp = knn_eval(k, data_train= data_train, data_test= data_test)
  data_knn2 = bind_rows(data_knn2, tmp)
}

data_knn2 %>% arrange(by = mse.test)
```

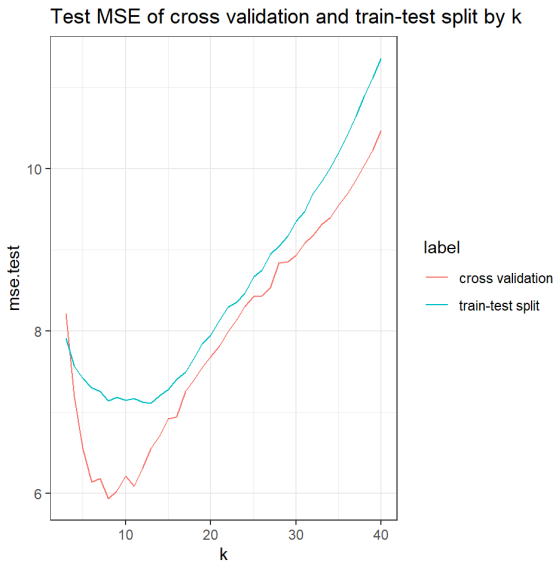
k <dbl>	edf <dbl>	mse.train <dbl>	mse.test <dbl>
13	7.692	5.170	7.109
12	8.333	5.117	7.126
8	12.500	4.799	7.144
10	10.000	4.835	7.147
11	9.091	5.023	7.169
9	11.111	4.846	7.188
14	7.143	5.396	7.205
7	14.286	4.710	7.258
15	6.667	5.566	7.284
6	16.667	4.697	7.303

1-10 of 38 rows Previous **1** 2 3 4 Next

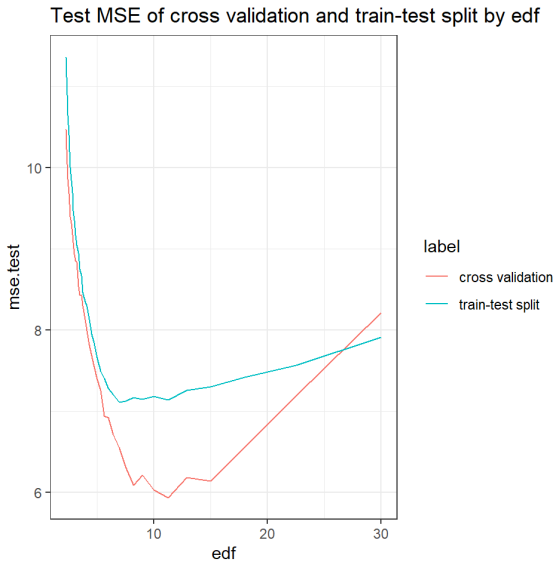
e. Plot both the cross-validation estimated and (true) error calculated from the test data on the same plot. See Figure 5.6 in ISL (pg 182) as a guide.

- Produce two plots: one with k on the x-axis and one with *edf* on the x-axis.
- Each plot should have two lines: one from part *a* and one from part *d*

```
split_knn <- data_knn2 %>% mutate(label = 'train-test split', edf = 90 / k)
cv_knn <- data_knn %>% group_by(k) %>%
  summarize(mse.test = mean(mse.test), label = 'cross validation') %>%
  mutate(edf = 90 / k)

ggplot() +
  geom_line(data = cv_knn, aes(x = k, y = mse.test, color = label)) +
  geom_line(data = split_knn, aes(x = k, y = mse.test, color = label)) +
  labs(title = "Test MSE of cross validation and train-test split by k")
```




```
ggplot() +
  geom_line(data = cv_knn, aes(x = edf, y = mse.test, color = label)) +
  geom_line(data = split_knn, aes(x = edf, y = mse.test, color = label)) +
  labs(title = "Test MSE of cross validation and train-test split by edf")
```



f. Based on the plots from *e*, does it appear that cross-validation worked as intended? How sensitive is the choice of k on the resulting test MSE?

As expected, cross-validation (red line) gives lower MSE than simple train-test split (blue line). At first, as k increases, test mse starts to decrease, but when k reaches a certain point, test MSE starts to increase again.