Homework #1: Supervised Learning

Hyunsuk Ko

Due: Wed Sept 07 | 11:45am

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
```

Problem 1: Evaluating a Regression Model

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

$$
\begin{aligned}
X &\sim \mathcal{N}(0, 1) \\
Y &= -1 + .5X + .2X^2 + \epsilon \\
\epsilon &\sim \mathcal{N}(0, \sigma)
\end{aligned}
$$

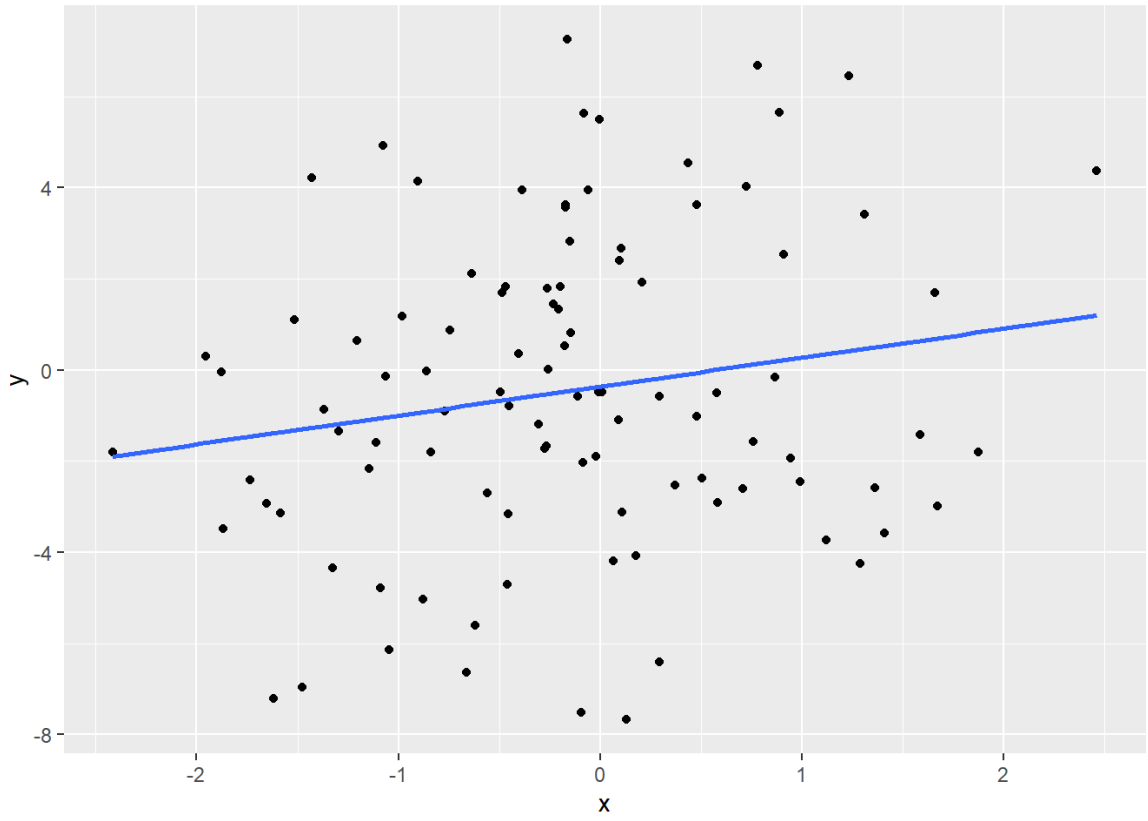
```
sim_x <- function(n) rnorm(n, mean = 0, sd = 1)
sim_y <- function(x, sd){              # generate Y|X from N{f(x),sd}
  n = length(x)
  f <- function(x) -1 + 0.5 * x + 0.2 * (x^2)
  f(x) + rnorm(n, mean = 0 ,sd=sd)  # error
}
```

b. Simulate $n = 100$ realizations from these distributions using $\sigma = 3$. Produce a scatterplot and draw the true regression line $f(x) = E[Y \mid X = x]$.

- Use `set.seed(611)` prior to generating the data.

```
set.seed(611)

n <- 100
sd <- 3

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

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
```

```
## `geom_smooth()` using formula 'y ~ x'
```

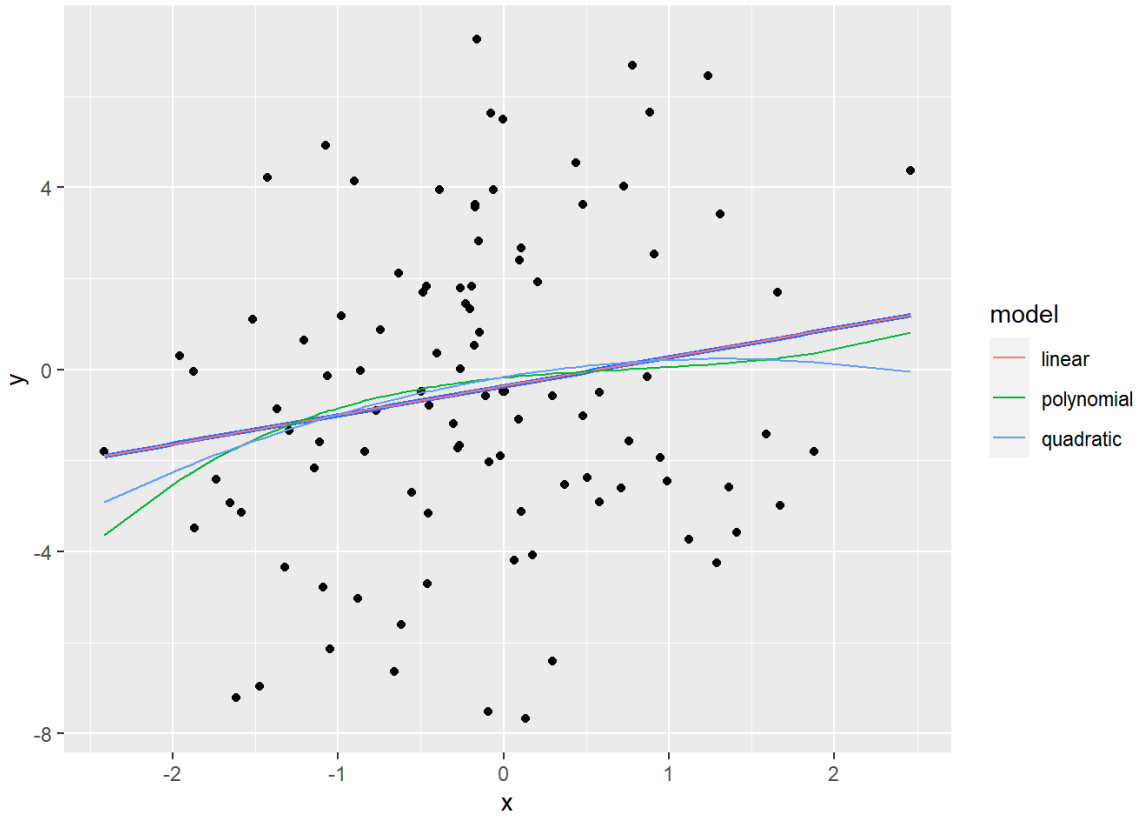


c. Fit three polynomial regression models using least squares: linear, quadratic, and cubic. Produce another scatterplot, add the fitted lines and true population line $f(x)$ using different colors, and add a legend that maps the line color to a model.

- Note: The true model is quadratic, but we are also fitting linear (less complex) and cubic (more complex) models.

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

m1 <- lm(y~x, data = data_train)
yhat1 <- predict(m1, newdata = xeval)

m2 <- lm(y~poly(x, degree=2), data=data_train)
yhat2 <- predict(m2, newdata=xeval)

m3 <- lm(y~poly(x, degree=3), data = data_train)
yhat3 <- predict(m3, newdata =xeval)

quad.data <- tibble(x = xseq, linear=yhat1, quadratic=yhat2, polynomial = yhat3)
%>%  # long data
  pivot_longer(-x, names_to="model", values_to="y")

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

```
## `geom_smooth()` using formula 'y ~ x'
```



d. Simulate a *test data* set of 10,000 observations from the same distributions. Use `set.seed(612)` prior to generating the test data.

- **Calculate the estimated mean squared error (MSE) for each model.**
  MSE: linear = 9.29, quadratic = 9.58, polynomial = 9.65

- **Are the results as expected?**
  Unlike expectation, test MSE keeps increasing as the degree increases.

```
#-- Generate Test Data
ntest <- 10000                          # Number of test samples
set.seed(612)                           # set *different* seed
xtest = sim_x(ntest)                    # generate test X's
ytest = sim_y(xtest, sd=sd)             # generate test Y's
data_test = tibble(x=xtest, y=ytest)    # test data

#-- Function to evaluate polynomials
poly_eval <- function(deg, data_train, data_test){
  if(deg==1) m <- lm(y~x, data = data_train)  # linear model

  else if (deg==2) m <- lm(y~poly(x, degree=2), data=data_train)  # quadratic mo
del

  else m <- lm(y~poly(x, degree=3), data = data_train) # polynomial model

  p = length(coef(m))                       # number of parameters
  mse.train = mean(m$residuals^2)           # training MSE
  yhat = predict(m, data_test)              # predictions at test X's
  mse.test = mean( (data_test$y - yhat)^2 ) # test MSE
  tibble(degree=deg, edf=p, mse.train, mse.test)
}

Deg = seq(1, 3, by=1)

# Using purrr:map_df()
data_poly = map_df(Deg, poly_eval, data_train=data_train, data_test=data_test)
data_poly
```

degree <dbl>	edf <int>	mse.train <dbl>	mse.test <dbl>
1	2	11.29355	9.293776
2	3	11.22076	9.583155
3	4	11.19797	9.648192

3 rows

e. What is the best achievable MSE? That is, what is the MSE if the true $f(x)$ was used to evaluate the test set? How close does the best method come to achieving the optimum?

: The true $f(x)$ gives the best achievable MSE, which is 9.09.

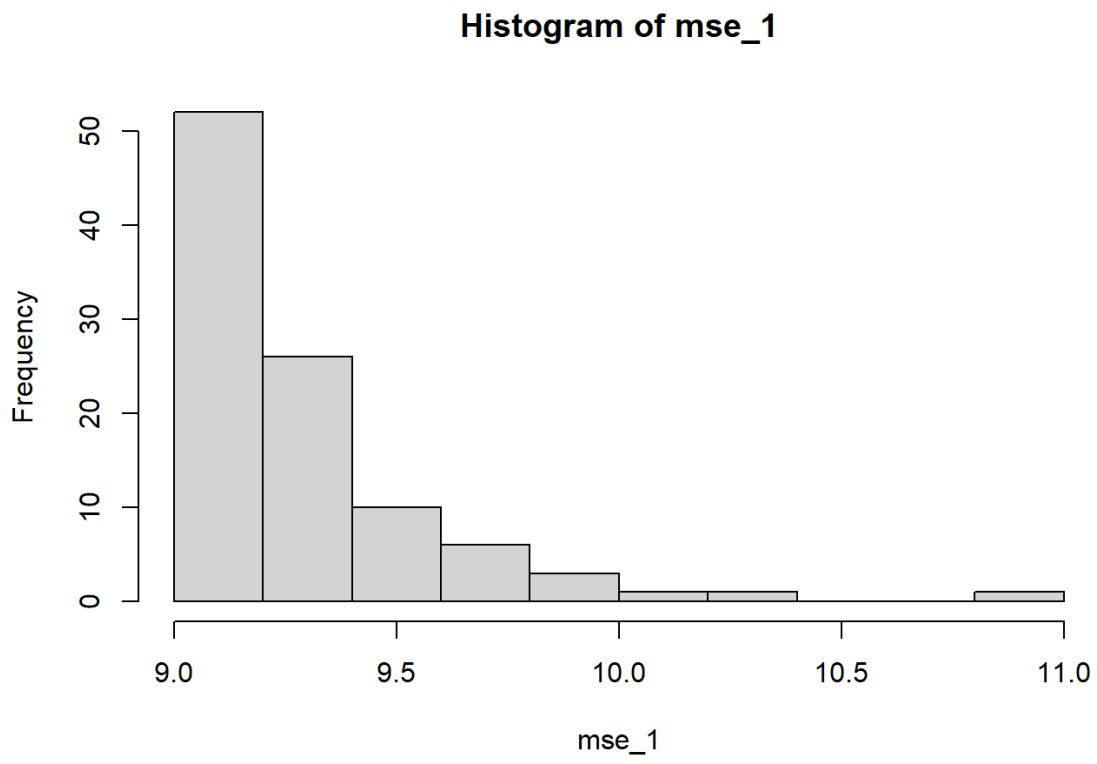
```
m4 <- lm(-1 + 0.5 * x + 0.2 * (x^2)  ~ x, data = data_train)
mse.train = mean(m4$residuals^2)          # training MSE
yhat = predict(m4, data_test)             # predictions at test X's
mse.test = mean( (data_test$y - yhat)^2 ) # test MSE
mse.test
```

```
## [1] 9.090774
```

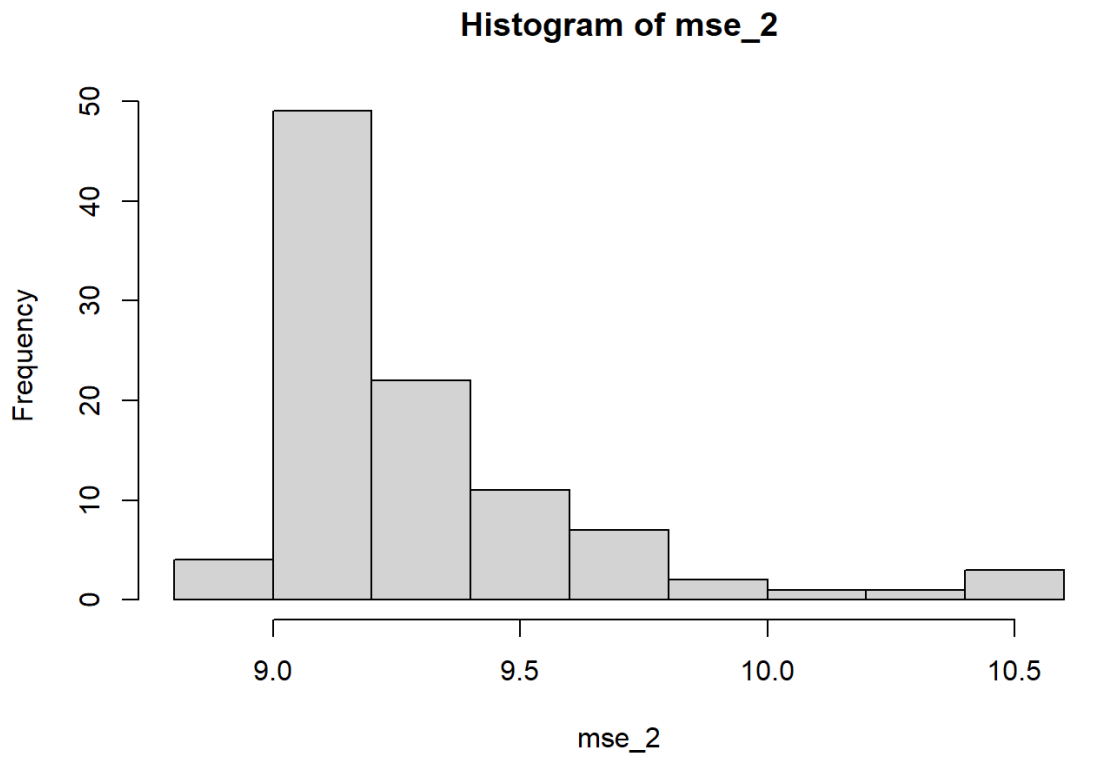
f. The MSE scores obtained in part *d* came from one realization of training data. Here will we explore how much variation there is in the MSE scores by replicating the simulation many times.

- Re-run parts b. and c. (i.e., generate training data and fit models) 100 times.
- Calculate the MSE for all simulations.
- Create kernel density or histogram plots of the resulting MSE values for each model.
- Use `set.seed(613)` prior to running the simulation and do not set the seed in any other places.
- Use the same test data from part d. (This question is only about the variability that comes from the training data).

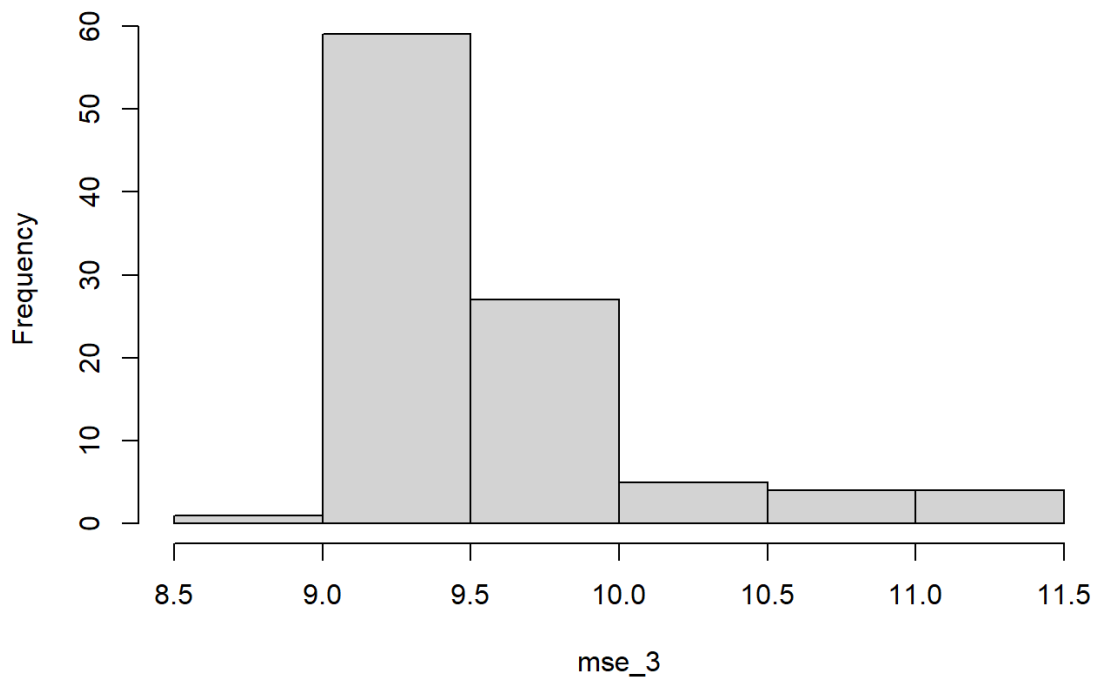
```
set.seed(613)

mse_1 <- c()
mse_2 <- c()
mse_3 <- c()
n <- 100
sd <- 3

for (i in 1:100){
  x <- sim_x(n)    # get x values
  y <- sim_y(x, sd=sd)
  data_train <- tibble(x, y)               # training data tibble
  xseq <- x
  xeval <- tibble(x = xseq)

  m1 <- lm(y~x, data = data_train)
  yhat1 <- predict(m1, newdata = xeval)

  m2 <- lm(y~poly(x, degree=2), data=data_train)
  yhat2 <- predict(m2, newdata=xeval)

  m3 <- lm(y~poly(x, degree=3), data = data_train)
  yhat3 <- predict(m3, newdata =xeval)

  quad.data <- tibble(x = xseq, linear=yhat1, quadratic=yhat2, polynomial = yhat
3) %>%  # long data
    pivot_longer(-x, names_to="model", values_to="y")

  data_poly = map_df(Deg, poly_eval, data_train=data_train, data_test=data_test)

  mse_1 <- append(mse_1, data_poly$mse.test[1])
  mse_2 <-append(mse_2, data_poly$mse.test[2])
  mse_3 <-append(mse_3, data_poly$mse.test[3])
}

df <- data.frame(mse_1 = mse_1, mse_2 = mse_2, mse_3 = mse_3)

hist(mse_1)
```




```
hist(mse_2)
```




```
hist(mse_3)
```



g. Show a count of how many times each model was the best. That is, out of the 100 simulations, count how many times each model had the lowest MSE.

: Out of 100 simulations, for linear, quadratic, and polynomial model, each of them had the lowest MSE 28, 65, and 7 times.

```
results <- c()

for (i in 1:100){
  if (min(df[i,]) == df[i,1]){
    results <- append(results, 1)
  }
  else if (min(df[i,]) == df[i,2]){
    results <- append(results, 2)
  }

  else {
    results <- append(results, 3)
  }
}

table(results)
```

```
## results
##  1  2  3
## 28 65  7
```

h. Write a function that implements the simulation in part *f*. The function should have arguments for i) the size of the training data n, and ii) the standard deviation of the random error σ. Use the same `set.seed(613)`.

```
sim_mse <- function(n, sigma) {
  set.seed(613)
  mse_1 <- c()
  mse_2 <- c()
  mse_3 <- c()

  for (i in 1:100){
    x <- sim_x(n)
    y <- sim_y(x, sigma)
    data_train <- tibble(x, y)
    xseq <- x
    xeval <- tibble(x = xseq)

    m1 <- lm(y ~ x, data = data_train)
    yhat1 <- predict(m1, newdata = xeval)

    m2 <- lm(y ~ poly(x, degree=2), data=data_train)
    yhat2 <- predict(m2, newdata=xeval)

    m3 <- lm(y ~ poly(x, degree=3), data = data_train)
    yhat3 <- predict(m3, newdata =xeval)

    data_poly = map_df(seq(1, 3, by=1), poly_eval, data_train=data_train, data_t
est=data_test)

    mse_1 <- append(mse_1, data_poly$mse.test[1])
    mse_2 <- append(mse_2, data_poly$mse.test[2])
    mse_3 <- append(mse_3, data_poly$mse.test[3])
  }

  df <- data.frame(mse_1 = mse_1, mse_2 = mse_2, mse_3 = mse_3)
  results <- c()

  for (i in 1:100){
  if (min(df[i,]) == df[i,1]){
    results <- append(results, 1)
  }
  else if (min(df[i,]) == df[i,2]){
    results <- append(results, 2)
  }

  else {
    results <- append(results, 3)
  }
}

  return (table(results))
}
```

i. Use your function to repeat the simulation in part *f*, but use $\sigma = 2$. Report the number of times each model was best (you do not need to produce any plots).

: Out of 100 simulations, for linear, quadratic, and polynomial models, each of them had the lowest MSE 14, 68, and 18 times.

```
sim_mse(100, 2)
```

```
## results
##  1  2  3
## 14 68 18
```

j. Repeat *i*, but now use $\sigma = 4$ and $n = 300$.

: Out of 100 simulations, for linear, quadratic, and polynomial models, each of them had the lowest MSE 11, 78, and 11 times.

```
sim_mse(300, 4)
```

```
## results
##  1  2  3
## 11 78 11
```

k. Describe the effects σ and n has on selection of the best model? Why is the *true* model form (i.e., quadratic) not always the *best* model to use when prediction is the goal?

If we have larger n, then we have a robust model as it has more training data. Also, with higher sigma, the varaince in both training data and test data increases, thus harder to predict, having lower MSE. If the test data has a different distribution than training data, then my model may not be the best model to predict. That is why we do not always have quadratic model the lowest MSE.