Homework 2

STAT 430, Spring 2017

Due: Friday, February 10 by 11:59 PM

Exercise 1

[**14 points**] In this exercise, you will investigate the bias-variance tradeoff when estimating the function f defined below.

```
f = function(x1, x2) {
  x1 ^ 3 + x2 ^ 3
}
```

The following code defines the data generating process and should we used to simulate data.

```
get_sim_data = function(f, sample_size = 100) {
  x1 = runif(n = sample_size, min = -1, max = 1)
  x2 = runif(n = sample_size, min = -1, max = 1)
  y = f(x1, x2) + rnorm(n = sample_size, mean = 0, sd = 0.5)
  data.frame(x1, x2, y)
}
```

Use simulation to investigate the bias and variance of *five* models at the point $\mathbf{x} = (x_1, x_2) = (0.75, 0.95)$. The five models are of the form

- `y ~ poly(x1, degree = k) + poly(x2, degree = k)`

for $k = 1, 2, 3, 4, 5$. Use 500 simulated samples each of size 200. Before performing the simulations, you should set a seed equal to your UIN. For example,

```
uin = 123456789
set.seed(uin)
```

Summarize your results as a *single* plot which compares both **squared bias** and **variance** of the estimates to the **degree** of the polynomials used. That is, the x-axis should be **degree** and you should have a line for both **squared bias** and **variance**. Comment on the plot. Are the results what you expected? Explain. (A few points may not strictly follow the general pattern as a result of the randomness of the simulation.)

Solution:

```
n_sims = 500
n_models = 5
x0 = data.frame(x1 = 0.75, x2 = 0.95)
predictions = matrix(0, nrow = n_sims, ncol = n_models)

for (i in 1:n_sims) {

  sim_data = get_sim_data(f, sample_size = 200)

  fit_1 = lm(y ~ poly(x1, 1) + poly(x2, 1), data = sim_data)
  fit_2 = lm(y ~ poly(x1, 2) + poly(x2, 2), data = sim_data)
  fit_3 = lm(y ~ poly(x1, 3) + poly(x2, 3), data = sim_data)
  fit_4 = lm(y ~ poly(x1, 4) + poly(x2, 4), data = sim_data)
  fit_5 = lm(y ~ poly(x1, 5) + poly(x2, 5), data = sim_data)
```

```r
  predictions[i, ] <- c(
    predict(fit_1, newdata = x0),
    predict(fit_2, newdata = x0),
    predict(fit_3, newdata = x0),
    predict(fit_4, newdata = x0),
    predict(fit_5, newdata = x0)
  )
}
```

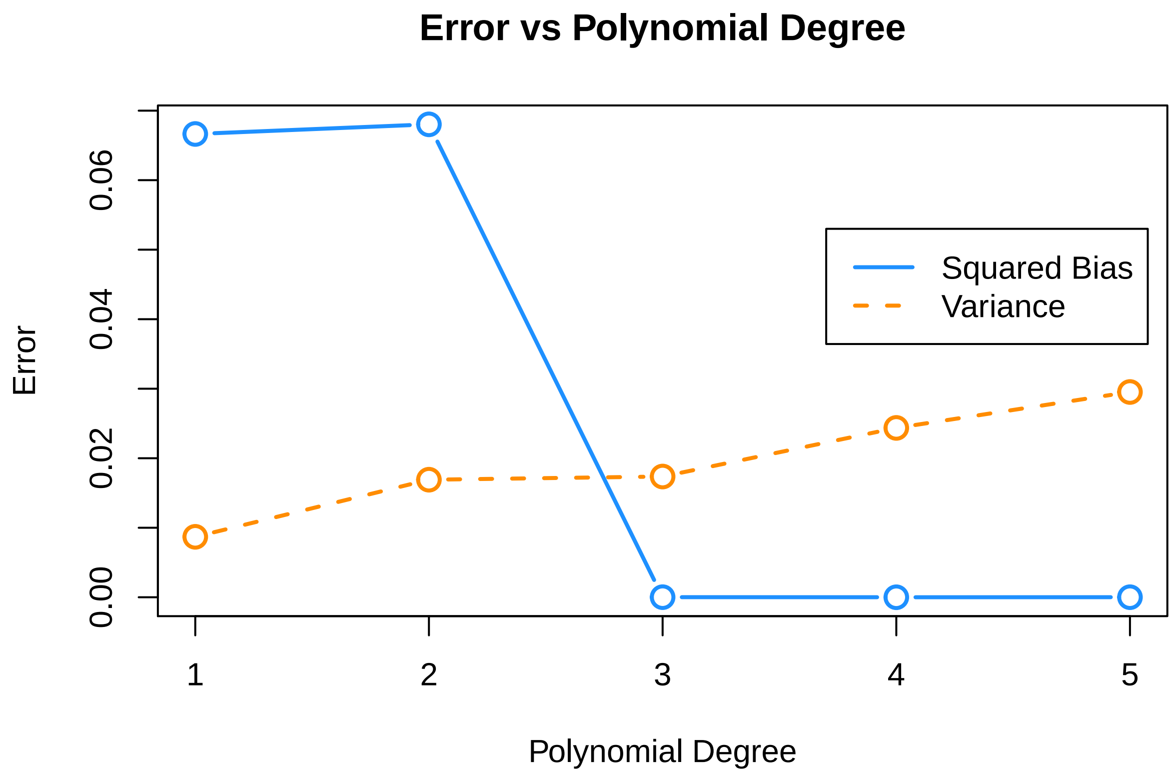
```r
eps = rnorm(n = n_sims, mean = 0, sd = 0.5)
y0 = f(x1 = 0.75, x2 = 0.95) + eps
```

```r
get_bias = function(estimate, truth) {
  mean(estimate) - truth
}

get_mse = function(estimate, truth) {
  mean((estimate - truth) ^ 2)
}
```

```r
bias = apply(predictions, 2, get_bias, f(x1 = 0.75, x2 = 0.95))
variance = apply(predictions, 2, var)
mse = apply(predictions, 2, get_mse, y0)
```

```r
plot(bias ^ 2, type = "b", col = "dodgerblue", cex = 1.5, lwd = 2,
     xlab = "Polynomial Degree", ylab = "Error",
     main = "Error vs Polynomial Degree")
lines(variance, type = "b", col = "darkorange", cex = 1.5, lwd = 2, lty = 2)
legend(x = 3.7, y = 0.053,
       c("Squared Bias", "Variance"),
       col = c("dodgerblue", "darkorange", "green", "orange", "black"),
       lty = c(1, 2), lwd = 2)
```



Here we mostly see the expected pattern. Squared bias (blue) generally decreases as the model complexity (polynomial degree) increases. On the other hand, variance (orange) increases as model complexity increases.

Exercise 2

[**8 points**] For this exercise use the data found in `hw02-train.csv` and `hw02-test.csv` which contain train and test data respectively.

```
library(tibble)
library(readr)

g = function(x1, x2, x3, x4) {
  3 + 0.5 * x1 * x2 + x3 + 0.2 * x4 ^ 3
}

make_hw02_data = function(n_obs = 1000) {

  x1 = runif(n = n_obs, min = 0, max = 3)
  x2 = rbinom(n = n_obs, size = 1, p = 0.5)
  x3 = runif(n = n_obs, min = 0, max = 1)
  x4 = rnorm(n = n_obs, mean = 0, sd = 2)

  eps = rnorm(n = n_obs, mean = 0 , sd = 1)

  y = g(x1, x2, x3, x4) + eps
```

```
  tibble(y, x1, x2, x3, x4)

}

set.seed(42)
hw02_train = make_hw02_data()
hw02_test  = make_hw02_data()
write_csv(hw02_train, "hw02-train.csv")
write_csv(hw02_test, "hw02-test.csv")
```

Find a model by fitting to the training data which achieves:

- Train RMSE less than 1.08
- Test RMSE less than 1.01

Report the model you found (you may use `R` formula notation), as well as the two metrics.

Solution:

We first read in the data, and write some helper function.

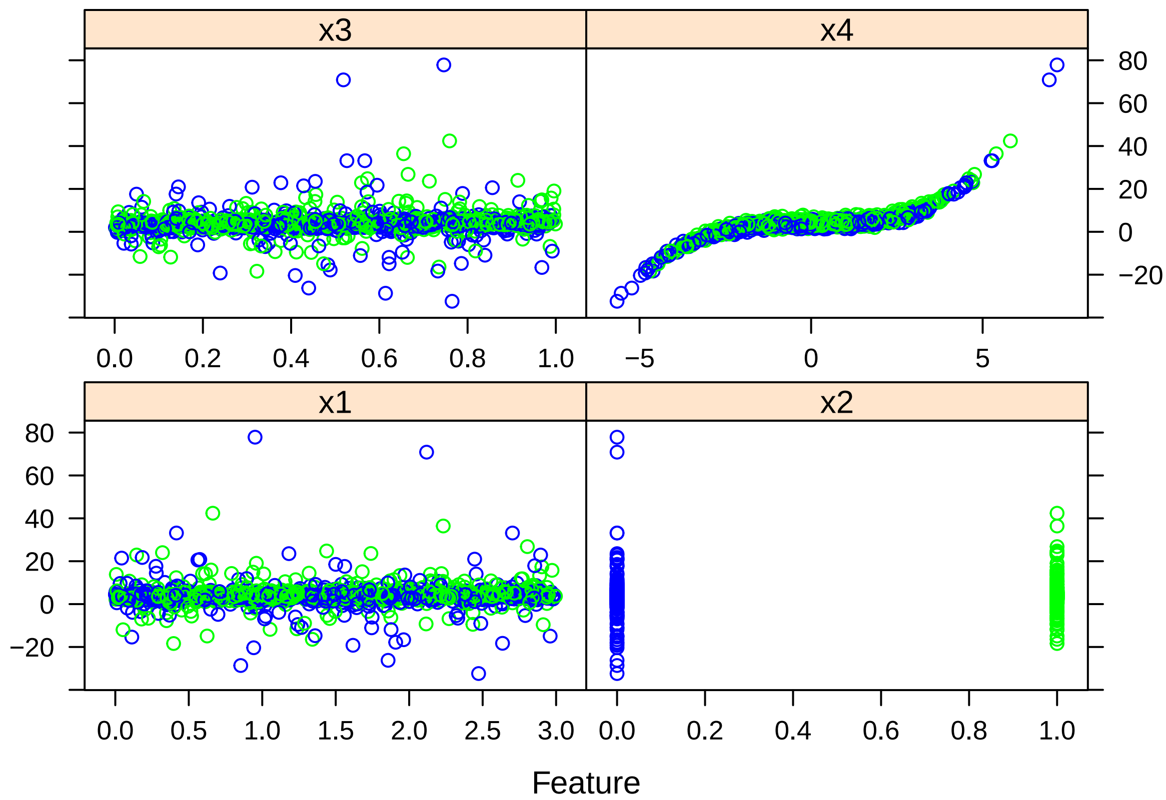
```
hw02_train = read_csv("hw02-train.csv")
hw02_test = read_csv("hw02-test.csv")
```

```
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

get_rmse = function(model, data, response) {
  rmse(actual = data[, response],
       predicted = predict(model, data))
}
```

By plotting the data, we immediately get a sense of a good model, in particular, one which contains a polynomial term of degree 3 for `x4`.

```
library(caret)
featurePlot(x = hw02_train[, c("x1", "x2", "x3", "x4")],
            y = hw02_train$y,
            col = ifelse(hw02_train$x2, "Green", "Blue"))
```



Here we report the best model. (See the data generation above.) Mostly models using all predictors and a polynomial term of degree 3 for `x4` should work well.

```
fit = lm(y ~ x1 * x2 + x3 + poly(x4, degree = 3), data = hw02_train)
c(get_rmse(model = fit, data = hw02_train, response = "y"),
  get_rmse(model = fit, data = hw02_test, response = "y"))
```

```
## [1] 1.029970 0.979488
```

Model	Train RMSE	Test RMSE
`y ~ x1 * x2 + x3 + poly(x4, degree = 3)`	1.0299703	0.979488

Exercise 3

[**8 points**] For this exercise use the data found in `auto-train.csv` and `auto-test.csv` which contain train and test data respectively. `auto.csv` is provided but not used. It is a modification of the `Auto` data from the `ISLR` package. For information on the original data:

```
library(ISLR)
#?Auto
```

Use the training data to train a classifier for `mpg` which achieves:

- Train Accuracy greater than 0.89
- Test Accuracy greater than 0.89

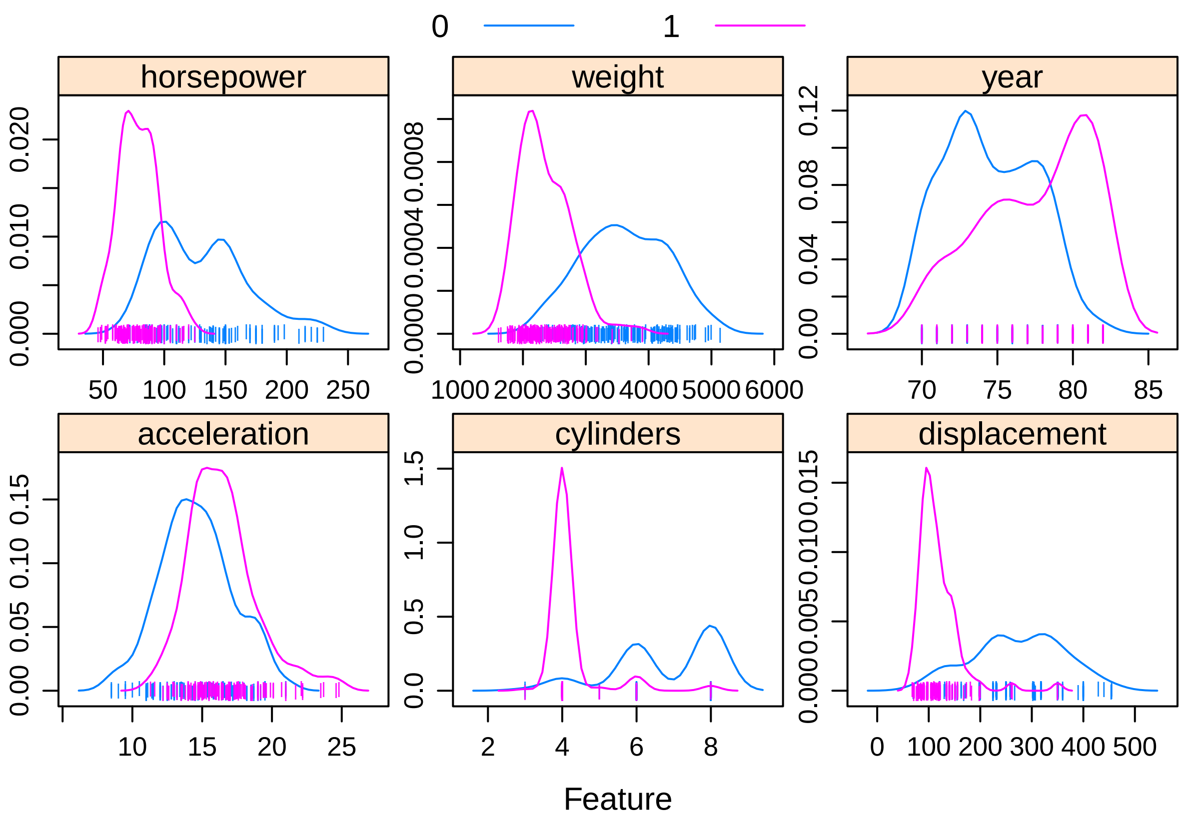
Report these metrics, as well as the confusion matrix, sensitivity, and specificity for the test data.

Solution:

```
auto_train = read_csv("auto-train.csv")
auto_test = read_csv("auto-test.csv")
```

By plotting the data, we "train" a classifier by noticing a reasonable split around a displacement of 185.

```
featurePlot(x = auto_train[, 2:7],
            y = as.factor(auto_train$mpg),
            plot = "density",
            scales = list(x = list(relation="free"),
                          y = list(relation="free")),
            adjust = 1,
            pch = "|",
            layout = c(3, 2),
            auto.key = list(columns = 2))
```



We write a simple R function to perform this classification.

```
simple_class = function(x, cutoff, above = 1, below = 0) {
  ifelse(x > cutoff, above, below)
}
```

We then obtain predictions for both the train and test data, which we then place into crosstables with their true values.

```
train_pred = simple_class(auto_train$displacement,
```

```r
                           cutoff = 185, above = 0, below = 1)
test_pred  = simple_class(auto_test$displacement,
                           cutoff = 185, above = 0, below = 1)
```

```r
train_tab = table(predicted = train_pred, actual = auto_train$mpg)
test_tab = table(predicted = test_pred, actual = auto_test$mpg)
```

Train Accuracy:

```r
confusionMatrix(train_tab)$overall["Accuracy"]
```

```
## Accuracy
##      0.9
```

Test Confusion:

```r
confusionMatrix(test_tab)$table
```

```
##          actual
## predicted  0  1
##         0 42  0
##         1  8 42
```

Test Metrics:

```r
c(confusionMatrix(test_tab)$overall["Accuracy"],
  confusionMatrix(test_tab)$byClass["Sensitivity"],
  confusionMatrix(test_tab)$byClass["Specificity"])
```

```
##    Accuracy Sensitivity Specificity
##   0.9130435   0.8400000   1.0000000
```