Homework 02

STAT 430, Fall 2017

Due: Friday, September 22, 11:59 PM

Exercise 1

```r
library(tibble)
library(readr)

# define data generating process
make_hw02_data = function(n_obs = 250) {

  x = runif(n = n_obs, min = -2, max = 2)
  f = 5 * x ^ 3 -x ^ 5
  eps = rnorm(n = n_obs, mean = 0 , sd = 2)
  y = f + eps
  tibble(y, x)

}

# generate datasets
set.seed(42)
hw02_train_data = make_hw02_data(n_obs = 100)
hw02_test_data = make_hw02_data(n_obs = 900)

# write to files
write_csv(hw02_train_data, "hw02-train-data.csv")
write_csv(hw02_test_data, "hw02-test-data.csv")

# clean up
rm(hw02_train_data)
rm(hw02_test_data)
```

[**15 points**] This exercise will use data in `hw02-train-data.csv` and `hw02-test-data.csv` which are train and test datasets respectively. Both datasets contain a single predictor `x` and a numeric response `y`.

Fit a total of 20 linear models. Each will be a polynomial model. Use degrees from 1 to 20. So, the smallest model you fit will be:

- `y ~ poly(x, degree = 1)`

The largest model you fit will be:

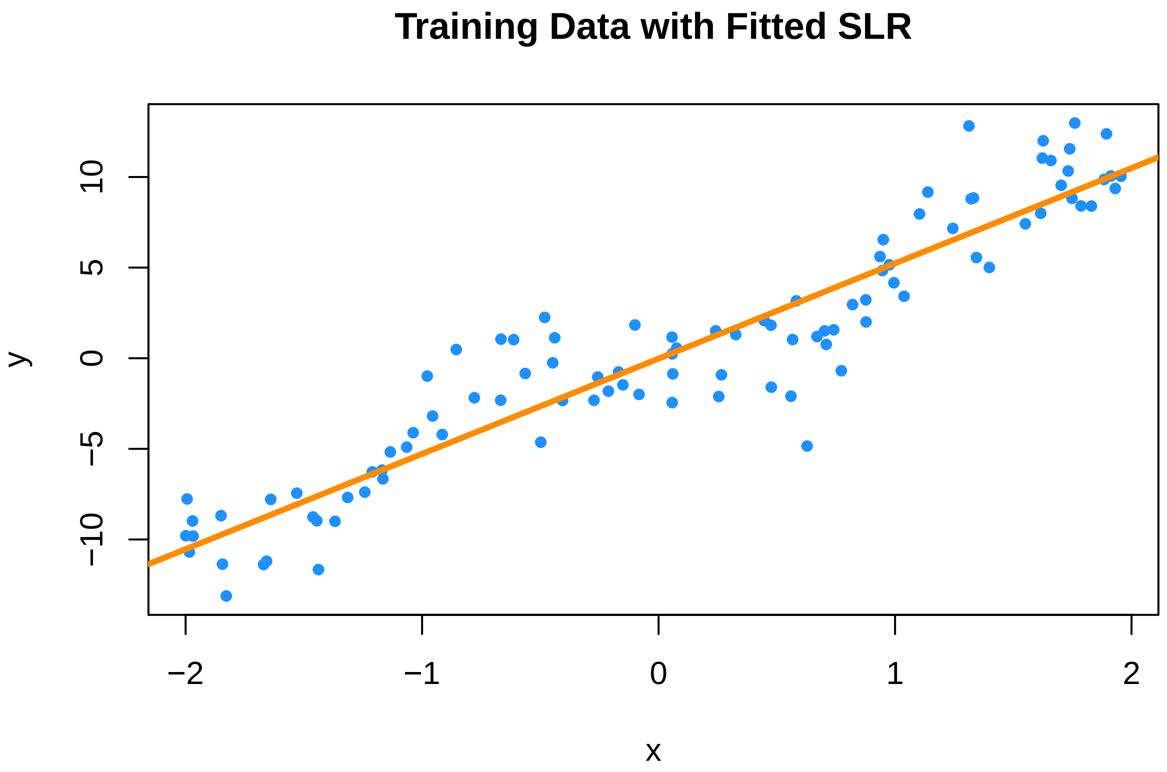
- `y ~ poly(x, degree = 20)`

For each model, calculate Train and Test RMSE. Summarize these results using a single plot which displays RMSE (both Train and Test) as a function of the degree of polynomial used. (Be sure to make the plot easy-to-read, and well labeled.) Note which polynomial degree appears to perform the "best," as well as which polynomial degrees appear to be underfitting and overfitting.

Solution:

```r
library(readr)
trn_data = read_csv("hw02-train-data.csv")
tst_data = read_csv("hw02-test-data.csv")
```

```r
plot(y ~ x, data = trn_data, pch = 20, col = "dodgerblue",
     main = "Training Data with Fitted SLR")
fit = lm(y ~ x, data = trn_data)
abline(fit, col = "darkorange", lwd = 3)
```

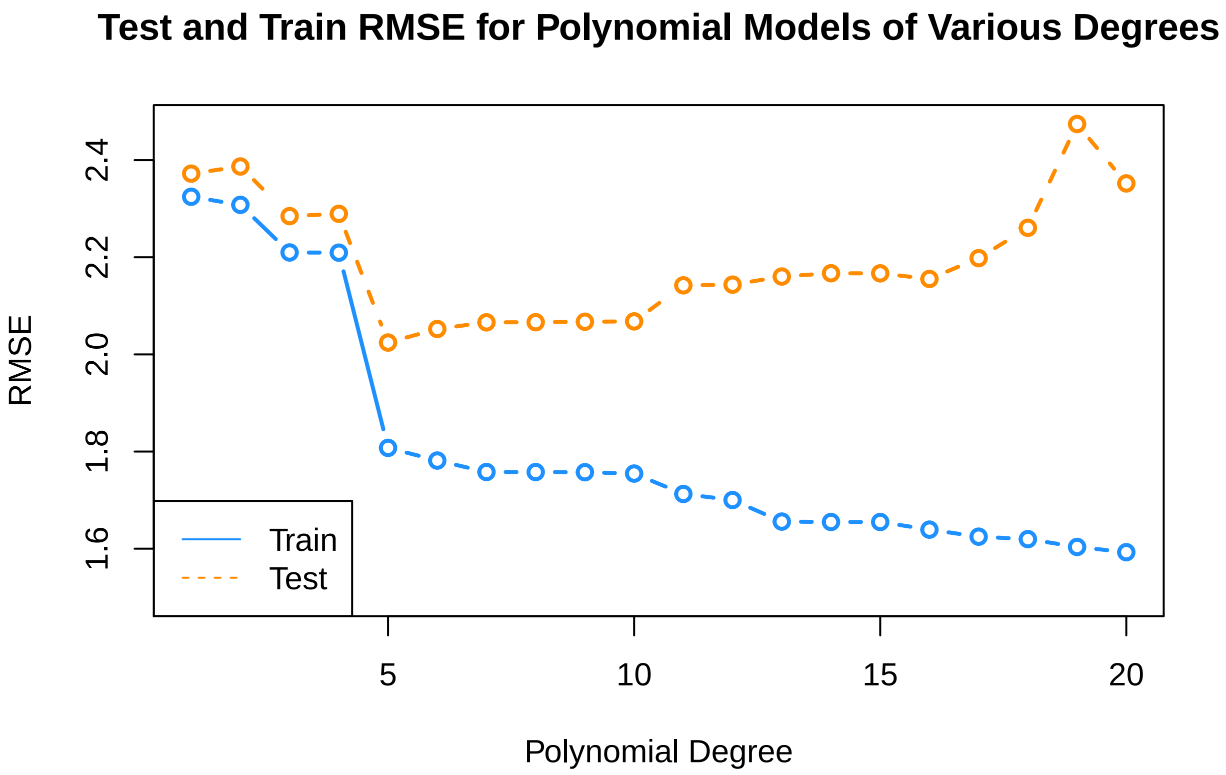


```r
# helper functions

# fit polynomial models of degree: degree
fit_poly = function(degree = 1, data) {
  lm(y ~ poly(x, degree = degree), data = data)
}

# calculate RMSE given actual and predicted values
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

# get RMSE given a model and data
get_rmse = function(model, data, response) {
  rmse(actual = data[, response],
       predicted = predict(model, data))
}
```

```r
# fit polynomial models of degree 1 through 20
degrees = 1:20
lm_model_list = lapply(degrees, fit_poly, data = trn_data)
```

```r
# calculate test and train RMSE for each model
lm_trn_rmse = sapply(lm_model_list, get_rmse, data = trn_data, response = "y")
lm_tst_rmse = sapply(lm_model_list, get_rmse, data = tst_data, response = "y")
```

```r
# plot results
plot(degrees, lm_trn_rmse, ylim = c(1.5, max(lm_tst_rmse)),
     type = "b", col = "dodgerblue",lwd = 2,
     ylab = "RMSE", xlab = "Polynomial Degree",
     main = "Test and Train RMSE for Polynomial Models of Various Degrees")
lines(degrees, lm_tst_rmse, type = "b", col = "darkorange", lwd = 2, lty = 2)
legend("bottomleft", c("Train", "Test"), lty = c(1, 2),
       col = c("dodgerblue", "darkorange"))
```



```r
# determine best, underfitting, and overfitting
which.min(lm_tst_rmse)
```

```
## [1] 5
```

```r
degrees[degrees < which.min(lm_tst_rmse)]
```

```
## [1] 1 2 3 4
```

```r
degrees[degrees > which.min(lm_tst_rmse)]
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Polynomial Degrees:

- Best: 5
- Underfitting: 1, 2, 3, 4
- Overfitting: 6, 7, 8, . . . , 20

Exercise 2

[**15 points**] This exercise will again use data in `hw02-train-data.csv` and `hw02-test-data.csv` which are train and test datasets respectively. Both datasets contain a single predictor `x` and a numeric response `y`.

Fit a total of 10 nearest neighbors models. Each will use a different value of `k`, the tuning parameter for the number of neighbors. Use the values of `k` defined in the following `R` chunk.

```r
k = seq(5, 50, by = 5)
```

For simplicity, do not worry about scaling the `x` variable.

For each value of the tuning parameter, calculate Train and Test RMSE. Summarize these results using a single well-formatted table which displays RMSE (both Train and Test), `k`, and whether or not that value of the tuning parameter appears to be overfitting, underfitting, or the "best" value of the tuning parameter. Consider rounding your results to show only enough precision to choose the "best" model.

Solution:

```r
# define helper function for getting knn.reg predictions
# note: this function is highly specific to this situation
make_knn_pred = function(k = 1, training, predicting) {
  pred = FNN::knn.reg(train = training["x"],
                      test = predicting["x"],
                      y = training$y, k = k)$pred
  act  = predicting$y
  rmse(predicted = pred, actual = act)
}
```

```r
# get requested train RMSEs
knn_trn_rmse = sapply(k, make_knn_pred,
                      training = trn_data,
                      predicting = trn_data)
# get requested test RMSEs
knn_tst_rmse = sapply(k, make_knn_pred,
                      training = trn_data,
                      predicting = tst_data)

# determine "best" k
best_k = k[which.min(knn_tst_rmse)]

# find overfitting, underfitting, and "best"" k
fit_status = ifelse(k < best_k, "Over", ifelse(k == best_k, "Best", "Under"))
```

```r
# summarize results
knn_results = data.frame(
  k,
  round(knn_trn_rmse, 2),
  round(knn_tst_rmse, 2),
  fit_status
)
colnames(knn_results) = c("k", "Train RMSE", "Test RMSE", "Fit?")

# display results
knitr::kable(knn_results)
```

k	Train RMSE	Test RMSE	Fit?
5	1.65	2.16	Over
10	1.70	2.08	Over
15	1.79	2.05	Best
20	1.93	2.06	Under
25	2.02	2.14	Under
30	2.28	2.36	Under
35	2.60	2.67	Under
40	2.96	2.99	Under
45	3.27	3.29	Under
50	3.58	3.57	Under