MATH560 HW 2

Jared Andreatta

2025-03-19

```r
library(ggplot2)
library(tidyr)
```

Section 2.4

Problem 3

a.

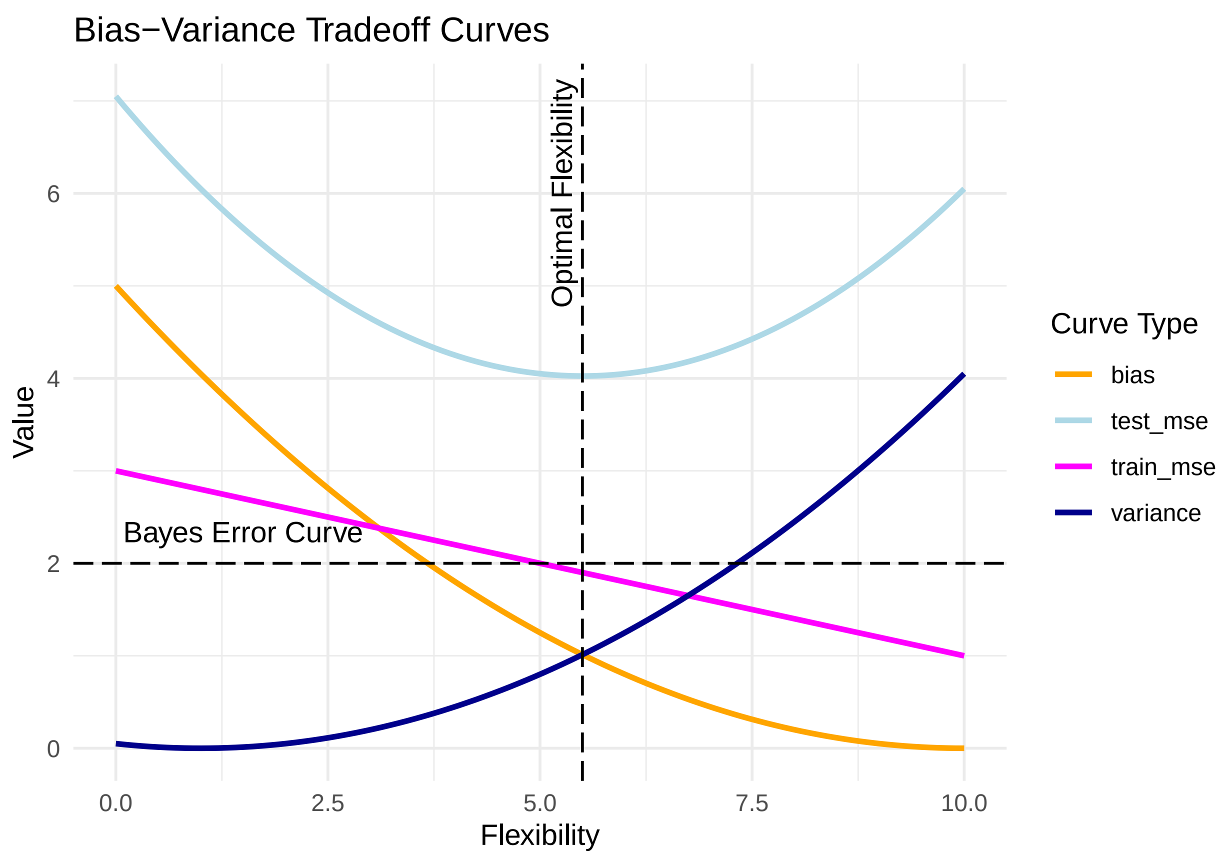
```r
x <- seq(0,10, length.out=100) # x-axis
variance <- (x-1)^2 / 20 # variance
bias <- (10-x)^2 / 20 # Bias
test_mse <- variance + bias + 2 # test MSE
train_mse <- 3 - 0.2 * x

df <- data.frame(x, test_mse, train_mse, variance, bias)

df_long <- pivot_longer(df, cols = c(test_mse, train_mse, variance, bias),
                        names_to = "curve", values_to = "value")

ggplot(df_long, aes(x = x, y = value, color = curve)) +
  geom_line(size = 1) +
  geom_vline(xintercept = 5.5, linetype = "longdash") +
  annotate("text", x = 5.5, y = 6,
           label = "Optimal Flexibility",
           angle = 90, vjust = -0.5) +
  geom_hline(yintercept = 2, linetype = "longdash") +
   annotate("text", x = 1.5, y = 2,
           label = "Bayes Error Curve",
           vjust = -1) +
  labs(title = "Bias-Variance Tradeoff Curves",
       x = "Flexibility",
       y = "Value",
       color = "Curve Type") +
  scale_color_manual(values = c("orange", "lightblue", "magenta", "darkblue")) +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



b.

- Bias is downward sloping because rigid models tend to provide biased estimates that consistently under/overestimate the true relationship of the data.
- Variance is upward sloping because more flexible models tend to overfit on noise and cannot generalize as well to unseen data.
- The test MSE follows a quadratic shape because it is minimized at the optimal level of bias and variance in a model. Too much bias or too much variance can cause the test MSE to increase.
- The train MSE curve is monotonically decreasing; as model complexity increases, it will fit the to the training set better as the model complexity increases.
- The Bayes error curve is a constant line that increases the test MSE because it is an intrinsic error that cannot be reduced, no matter the complexity of the model.
- The Optimal Flexibility line is the desired model flexibility that minimizes the test MSE, which is a single point on the test MSE curve.

Section 5.4

Problem 3

a.

Here is a short rundown of how k-Fold CV works:

1. Initialize first fold as a validation set.
2. Train on remaining $k - 1$ folds and validate on the held-out fold.
3. Compute MSE_i where i denotes the fold being held-out as a validation set.
4. Repeat this k times with each iteration using a different group of observations as a validation set.

Essentially, it looks like this mathematically

$$CV_k = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

b.

k-Fold vs Validation Set The advantages of k-Fold CV is that the estimates are more accurate, as the estimates are averaged out over k folds instead of 1 random sample. Also, k-Fold doesn't contain all of the data to one hold-out set.

The disadvantage of k-Fold over Validation Set CV is the computational cost, as it requires more iterations than the simple validation set approach.

k-Fold vs LOOCV One big advantage over LOOCV is the computational cost; LOOCV is a special case where $k = n$, but when $k < n$, it is easier to compute, and often it is unnecessary to estimate the MSE n times. Also, the MSE estimate from k-Fold has less variance since it leaves out more than one observation, which gives a more stable error estimate.

The disadvantage is that LOOCV is more accurate (in theory) in estimating MSE. k-Fold also gives a more biased estimate of MSE since k-Fold uses smaller training sets.

Problem 8

a.

```r
set.seed(1)
x <-rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
```

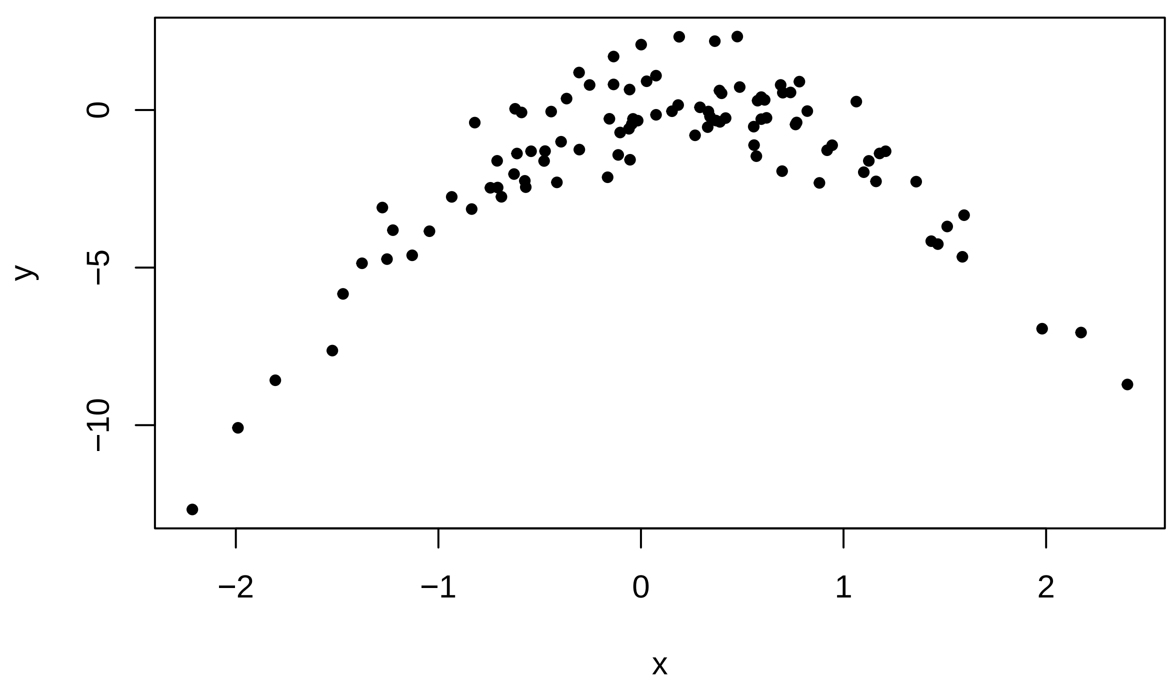
In this data, $n = 100$ and $p = 1$. The equation is

$$y = x - 2x^2 + \epsilon \quad \text{where } \epsilon \sim N(0, 1)$$

b.

I notice a concave function that is not perfectly smooth, since we added a random perturbance term ϵ, which introduces some noise in the curve.

```r
plot(x,y, pch=20)
```



c.

```r
library(boot)
```

```
## Warning: package 'boot' was built under R version 4.4.3
```

```r
set.seed(42)

data <- data.frame(x,y)

lm_1 <- glm(y ~ x, data=data)
lm_2 <- glm(y ~ poly(x, 2), data=data)
lm_3 <- glm(y ~ poly(x, 3), data=data)
lm_4 <- glm(y ~ poly(x, 4), data=data)

cv_1 <- cv.glm(data, lm_1)
```

```r
cv_2 <- cv.glm(data, lm_2)
cv_3 <- cv.glm(data, lm_3)
cv_4 <- cv.glm(data, lm_4)

cv_list <- list(cv_1$delta, cv_2$delta, cv_3$delta, cv_4$delta)

print(cv_list)
```

```
## [[1]]
## [1] 7.288162 7.284744
##
## [[2]]
## [1] 0.9374236 0.9371789
##
## [[3]]
## [1] 0.9566218 0.9562538
##
## [[4]]
## [1] 0.9539049 0.9534453
```

d.

The results are similar, but they are not the same. This is because the random error term that's produced is different since the seed is different.

```r
set.seed(1000)

data <- data.frame(x,y)

lm_1 <- glm(y ~ x, data=data)
lm_2 <- glm(y ~ poly(x, 2), data=data)
lm_3 <- glm(y ~ poly(x, 3), data=data)
lm_4 <- glm(y ~ poly(x, 4), data=data)

cv_1 <- cv.glm(data, lm_1)
cv_2 <- cv.glm(data, lm_2)
cv_3 <- cv.glm(data, lm_3)
cv_4 <- cv.glm(data, lm_4)

cv_list <- list(cv_1$delta, cv_2$delta, cv_3$delta, cv_4$delta)

print(cv_list)
```

```
## [[1]]
## [1] 7.288162 7.284744
##
## [[2]]
## [1] 0.9374236 0.9371789
##
## [[3]]
## [1] 0.9566218 0.9562538
##
```

```
## [[4]]
## [1] 0.9539049 0.9534453
```

e.

The 2nd degree polynomial model produces the lowest LOOCV error. This is to be expected, as the simulated data follows a quadratic form.

f.

By seeing the summary of each model, we can simply expect the significance codes, which are denoted by asterisks. In the first linear model, we see that the coefficient β_1 is significant on the 95% level. On the 2nd model, we see that all terms are significant on the >99.9% levels. For the third and fourth models, we note that β_0, β_1, and β_2 are all significant on the >99.9% levels, but the higher order terms are not significant. This entails that higher order models are unnecessary and the patterns of the data are best captured by a second order polynomial fit.

```r
summary(lm_1)
```

```
##
## Call:
## glm(formula = y ~ x, data = data)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.6254     0.2619  -6.205 1.31e-08 ***
## x             0.6925     0.2909   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.760719)
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
```

```r
summary(lm_2)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 2), data = data)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
## poly(x, 2)1   6.1888     0.9580    6.46 4.18e-09 ***
## poly(x, 2)2 -23.9483     0.9580  -25.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
```

```
summary(lm_3)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 3), data = data)
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
## poly(x, 3)1   6.18883    0.96263   6.429 4.97e-09 ***
## poly(x, 3)2 -23.94830    0.96263 -24.878  < 2e-16 ***
## poly(x, 3)3   0.26411    0.96263   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
##
## Number of Fisher Scoring iterations: 2
```

```
summary(lm_4)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 4), data = data)
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, 4)3   0.26411    0.95905   0.275    0.784
## poly(x, 4)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
```

```
##
## Number of Fisher Scoring iterations: 2
```