HW4

Nathan Krieger

2026-02-17

Problem 1

(a)

Hypothesis testing is valuable here because while the summary shows us that the estimate is non-zero, it doesn't mean that the predictor X_3 actually influences Y. It would be rare to see an estimate that is truly zero due to noise.

We can use hypothesis testing to determine if the estimate is a real signal or just random luck.

In the case of X_3, the estimate is non-zero, but the P-value is 0.334 which is much higher than the $\alpha = 0.05$ threshold, therefore, X_3 is not a good predictor of Y

(b)

I disagree with this claim. Even if we knew the true $f(X)$, there is still error (ϵ) that we cannot predict.

$Y = f(X) + \epsilon$

(c)

False. The expected test MSE is evaluated based on the data (x_0, y_0) that's not part of the training set.

(d)

True. Sometimes a model that is too complex can overfit the noise. If we remove a predictor we may increase bias but also decrease the variance. This leads to a better test MSE.

(e)

False. The expected test MSE includes $Var(\epsilon)$ so there's no way it can be smaller than it.

From the slides: "The expected test MSE is never smaller than the irreducible error."

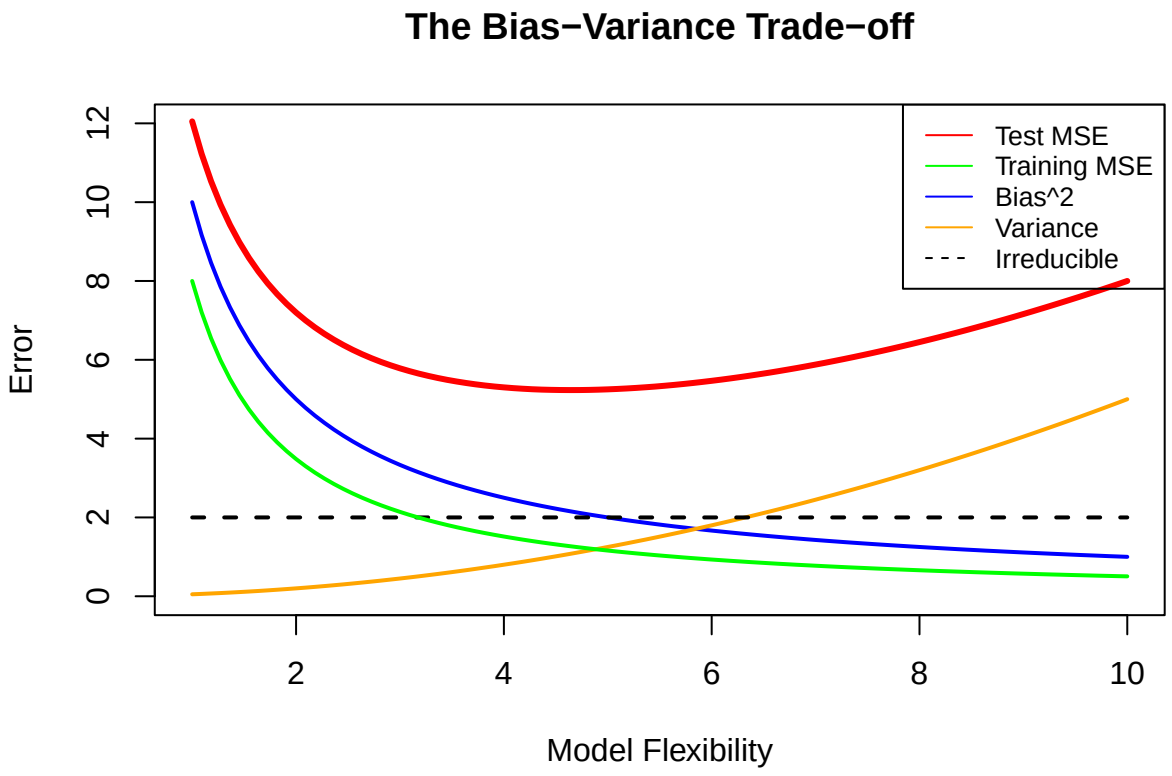
(f)

True. We could fit a model to be exactly the same (overfitting) which can effectively cover all points exactly, making the training MSE = 0 which is less than the irreducible error.

Problem 2

(a)

```r
x <- seq(1, 10, length.out = 100)

bias_sq   <- 10 / x
variance  <- 0.05 * x^2
irred     <- rep(2, 100)
test_mse  <- bias_sq + variance + irred
train_mse <- 8 / x^1.2

plot(x, test_mse, type = "n", ylim = c(0, 12),
     xlab = "Model Flexibility", ylab = "Error",
     main = "The Bias-Variance Trade-off")

lines(x, bias_sq,   col = "blue",   lwd = 2) # Squared Bias
lines(x, variance,  col = "orange", lwd = 2) # Variance
lines(x, irred,     col = "black",  lwd = 2, lty = 2) # Irreducible Error
lines(x, test_mse,  col = "red",    lwd = 3) # Test MSE
lines(x, train_mse, col = "green",  lwd = 2) # Training MSE

legend("topright", cex = 0.8,
       legend = c("Test MSE", "Training MSE", "Bias^2", "Variance", "Irreducible"),
       col = c("red", "green", "blue", "orange", "black"),
       lty = c(1, 1, 1, 1, 2))
```



(b)

Expected test MSE: A measurement of how well the model predicts new unseen data.

Training MSE: How well the model performs on the exact data that it used to learn.

Bias: The error that comes from using a model that is too simple to capture the true pattern. It consistently

misses the mark in a predictable way.

Variance: How much the model's predictions change if you trained it on a different set of data.

Irreducible Error: The error that stays even if you have a perfect model. It is inherent to the problem itself.

(c)

Squared bias: The error that comes with trying to estimate the function.

Variance: Increasing because a model that is very flexible follows the data very closely. This means that if you add new data the line would have to change its entire shape to accommodate them.

Expected test MSE: This is how well the model performs on the test data. This MSE is much more important than the training MSE.

Training MSE: This is decreasing because as the flexibility increases it is able to get closer to more of the datapoints and accommodate for the exact shape of the data.

Irreducible error: This is a flat line. Since it's the variance of the random noise (ϵ), and ϵ is inherent to the data and doesn't care about the model, it's a horizontal line.

(d)

I would expect the cubic model to have a lower training MSE because it can adapt to the specific training data better since it is more flexible.

(e)

I would expect the linear model to have a lower test MSE because the true relationship is linear. The cubic model will overfit the noise in the training data leading to a higher test MSE.

Problem 3

```
set.seed(1)


library(ISLR2)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(ggplot2)
#install.packages('caret')

k = 5 # Number of folds

max_degree <- 9
errors <- numeric(max_degree)

# Create folds
flds <- createFolds(Boston$medv, k, list = TRUE)
flds[[1]]
```

```
##   [1]   1   5   7   8   9  16  17  20  22  24  31  32  55  67  69  72  82  85
##  [19]  88  98  99 122 137 138 148 150 153 157 161 167 172 175 192 199 202 211
```

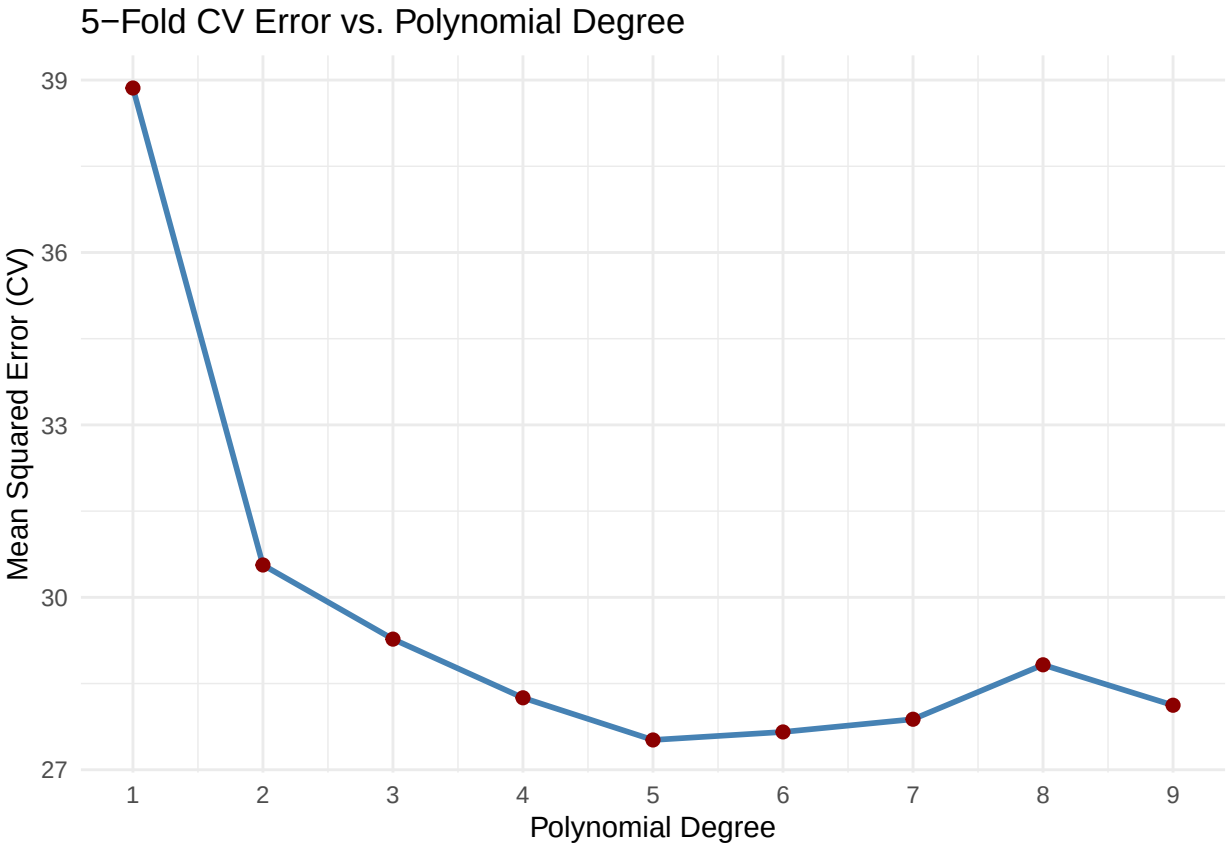
```
##  [37] 217 218 222 224 237 241 243 249 253 254 258 261 262 263 271 275 279 287
##  [55] 290 299 300 303 304 314 323 324 327 339 342 343 346 347 351 354 359 364
##  [73] 379 381 382 389 399 401 405 406 408 413 432 434 437 438 442 457 459 466
##  [91] 468 472 478 480 483 484 486 497 500 503 504
```

```
for (d in 1:max_degree) {

  folds <- numeric(k)

  for(i in 1:k){
    # Get test indices for this fold
    test_index = flds[[i]]
    test = Boston[test_index, ]
    train = Boston[-test_index, ]

    m <- lm(medv ~ poly(lstat, d), data = train)

    predict <- predict(m, newdata = test)

    folds[i] = mean((test$medv - predict)^2)

  }

  errors[d] = mean(folds)
}

cv_results <- data.frame(Degree = 1:max_degree, CV_Error = errors)
print(cv_results)
```

```
##   Degree CV_Error
## 1      1 38.86182
## 2      2 30.56280
## 3      3 29.27295
## 4      4 28.25229
## 5      5 27.51844
## 6      6 27.65806
## 7      7 27.87931
## 8      8 28.82625
## 9      9 28.12261
```

```
ggplot(cv_results, aes(x = Degree, y = CV_Error)) +
  geom_line(color = "steelblue", size = 1) +
  geom_point(color = "darkred", size = 2) +
  scale_x_continuous(breaks = 1:9) +
  labs(title = "5-Fold CV Error vs. Polynomial Degree",
       x = "Polynomial Degree",
       y = "Mean Squared Error (CV)") +
  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 per session.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Based on the data, the model with degree of 5 has the lowest MSE. For that reason, when building the model I would select degree 5.

Problem 4

(a)

The createFolds method shuffles the data and splits into k equal sized groups (or folds).

Then, we iterate through a loop k times doing the following:

- Select one fold to be the test/validation set.
- Use the remaining k-1 folds combined as the training set.
- Fit the model on the training set and evaluate its performance on the test fold.

Once every fold has been the test set exactly once, calculate the average of the k resulting test errors. This average is your final CV estimate of the model's performance.

(b)

(i)

k-fold validation is much more consistent compared to a basic validation set approach. instead of only training on ~50% of the data, the k fold approach trains on $(k-1)/k\%$

(ii)

(c)

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

(d)

```
set.seed(1)

Data = data.frame(x, y)

n <- nrow(Data)

MSE_M1 = rep(0,n) #vector
MSE_M2 = rep(0,n) #vector
MSE_M3 = rep(0,n) #vector
MSE_M4 = rep(0,n) #vector

for(i in 1:n){
  test = Data[i, ]
  train = Data[-i, ]

  M1 <- lm(y ~ x, data = train)

  M2 <- lm(y ~ poly(x, degree = 2, raw = TRUE), data = train)

  M3 <- lm(y ~ poly(x, degree = 3, raw = TRUE), data = train)

  M4 <- lm(y ~ poly(x, degree = 4, raw = TRUE), data = train)

  M1_test = predict(M1, newdata = test)
  M2_test = predict(M2, newdata = test)
  M3_test = predict(M3, newdata = test)
  M4_test = predict(M4, newdata = test)


  MSE_M1[i] = (test$y - M1_test)^2
  MSE_M2[i] = (test$y - M2_test)^2
  MSE_M3[i] = (test$y - M3_test)^2
  MSE_M4[i] = (test$y - M4_test)^2

}

LOOCV_MSE_M1 = mean(MSE_M1)
LOOCV_MSE_M2 = mean(MSE_M2)
LOOCV_MSE_M3 = mean(MSE_M3)
LOOCV_MSE_M4 = mean(MSE_M4)

LOOCV_MSE_M1
```

```
## [1] 7.288162
```

```
LOOCV_MSE_M2
```

```
## [1] 0.9374236
```

```
LOOCV_MSE_M3
```

```
## [1] 0.9566218
```

```
LOOCV_MSE_M4
```

```
## [1] 0.9539049
```

(e)

```
set.seed(10101)

Data = data.frame(x, y)

n <- nrow(Data)

MSE_M1 = rep(0,n) #vector
MSE_M2 = rep(0,n) #vector
MSE_M3 = rep(0,n) #vector
MSE_M4 = rep(0,n) #vector

for(i in 1:n){
  test = Data[i, ]
  train = Data[-i, ]

  M1 <- lm(y ~ x, data = train)

  M2 <- lm(y ~ poly(x, degree = 2, raw = TRUE), data = train)

  M3 <- lm(y ~ poly(x, degree = 3, raw = TRUE), data = train)

  M4 <- lm(y ~ poly(x, degree = 4, raw = TRUE), data = train)

  M1_test = predict(M1, newdata = test)
  M2_test = predict(M2, newdata = test)
  M3_test = predict(M3, newdata = test)
  M4_test = predict(M4, newdata = test)


  MSE_M1[i] = (test$y - M1_test)^2
  MSE_M2[i] = (test$y - M2_test)^2
  MSE_M3[i] = (test$y - M3_test)^2
  MSE_M4[i] = (test$y - M4_test)^2

}

LOOCV_MSE_M1 = mean(MSE_M1)
LOOCV_MSE_M2 = mean(MSE_M2)
LOOCV_MSE_M3 = mean(MSE_M3)
LOOCV_MSE_M4 = mean(MSE_M4)
```

```
LOOCV_MSE_M1
```

```
## [1] 7.288162
```

```
LOOCV_MSE_M2
```

```
## [1] 0.9374236
```

```
LOOCV_MSE_M3
```

```
## [1] 0.9566218
```

```
LOOCV_MSE_M4
```

```
## [1] 0.9539049
```

The results for part e remained the exact same as part d even with a different seed value. This is because LOOCV is deterministic and works by leaving just one data point out at a time and doing it n times. This means that each data point is left out exactly once regardless of the seed number.

(f)

M2 (degree 2) had the smallest LOOCV error. This is what I expected because $f(X)$ is an equation with a degree of 2.

(g)

LOOCV trains the model on n - 1 observations so it gives a better, more accurate estimate of the test error when comparing it to a standard test, training set approach.

(h)

As k increases, the bias decreases. This is because the training data gets larger (closer to the size of the full data set) as k gets larger.

(i)

TODO

As k increases, the variance increases. This is because as k increases, the size of the testing data decreases which increases variance.