Homework 2

Tim Mastny

February 7, 2018

1

To make it easier to type, let $y_{new} = y$ and $\hat{f}(x_{new}) = \hat{f}$. Then

$$\begin{aligned} E(y - \hat{f})^2 &= E(y^2 + \hat{f}^2 - 2y\hat{f}) \\ &= E(y^2) + E(\hat{f}^2) - 2E(y\hat{f}) \\ &= E(y^2) - E^2(y) + E^2(y) + E(\hat{f}^2) - E^2(\hat{f}) + E^2(\hat{f}) - 2E(y\hat{f}) \\ &= Var(y) + E^2(y) + Var(\hat{f}) + E^2(\hat{f}) - 2E(y\hat{f}) \end{aligned}$$

Note that $E^2(y) = E^2[f + \epsilon]$. The expected error is zero, so $E^2[f + \epsilon] = E^2[f] = f^2$, where f is the "true" function for y, with error ϵ. And note that $E(y\hat{f}) = E(y)E(\hat{f}) = fE(\hat{f})$, since y and $\hat{f}$ are independent. Then

$$\begin{aligned} E(y - \hat{f})^2 &= Var(y) + Var(\hat{f}) + (f^2 + E^2(\hat{f}) - 2fE(\hat{f})) \\ &= Var(y) + Var(\hat{f}) + (f - E(\hat{f}))^2 \end{aligned}$$

And $Var(y) = E((y - E(y))^2) = E((y - f)^2) = E((f + \epsilon - f)^2) = E(\epsilon^2) = Var(\epsilon)$, since $E(\epsilon) = 0$. So

$$E(y - \hat{f})^2 = Var(\epsilon) + Var(\hat{f}) + Bias(\hat{f})^2$$

2.4 Question 1

(a)

Due to the large sample size and small number of predictors, we would expect the flexible model to perform no better, and maybe even worse than the inflexible model.

First, if the sample size is very large, we would expect that the sample well represents a known probability distribution that our classic, inflexible models are based on (such as normal, binomial, etc.). Second, since we have a very large n, the standard error of the mean should be very low. And since we are working in a low dimensional space, we don't need the complexity offered by more flexible models.

(b)

This scenario is the inverse of (a). One concern is dealing with the high-dimensional features. According to the chart on page 25, Lasso is considered an inflexible model, and it explicitly reduces the number of predictors.

Moreover, according to page 23, non-parametric methods, which are many of which are flexible models, need much more data than inflexible ones. And here we are very constrained on data.

Therefore, I would argue we need an inflexible model to deal with this situation.

(c)

Since we don't have constraints on data or predictors, a flexible model is much more likely to perform better. Page 23 asserts that non-parametric functions are much better at dealing with wiggly (non-linear) functions and non-parameter functions are the most flexible.

(d)

In this case the flexible model will perform worse. Page 22 and 26 make it clear: flexible models are prone to overfitting. Since the noise is so high, it is probable that a flexible model would fit the noise, while overlooking the main trend.

With inflexible models, we can explicitly state our assumptions (such as the relationship is linear), which may help to avoid overfitting and help us understand the data.

5.4 Question 2

(a)

Let X_1 be the random variable for the first bootstrap observation that equals the place (from 1 to n) in the original sample that the bootstrap observation was pulled from. We want to find $P(X_1 \neq j)$. Recall

$$\begin{aligned} P(X_1 \neq j) &= 1 - P(X_1 = j) \\ &= 1 - 1/n \end{aligned}$$

Where $P(X_1 = j) = 1/n$ for any observation in the original sample, since each are equally likely to be sampled.

(b)

Bootstraps are sampled indepedently and with replacement. Therefore the probability of the first bootstrap observation being the j^{th} observation from the original sample is equal to any bootstrap observation being the j^{th} observation.

(c)

Recall that if each event is indepedent, then $P(A \cap B) = P(A)P(B)$. We can apply this to find the probability that the j^{th} observation is not in the bootstrap:

$$\begin{aligned} P\left(\bigcap_{i=1}^{n} X_i \neq j\right) &= \prod_{i=1}^{n} P(X_i \neq j) \\ &= \prod_{i=1}^{n} 1 - P(X_i = j) \\ &= \prod_{i=1}^{n} 1 - 1/n \\ &= (1 - 1/n)^n \end{aligned}$$

(d)

Let's find the general formula. We want to the probability that the j^{th} observation is in the sample. That's just the opposite of the j^{th} observation not being in the sample:

$$\begin{aligned} P\left(\neg \bigcap_{i=1}^{n} X_i \neq j\right) &= 1 - P\left(\bigcap_{i=1}^{n} X_i \neq j\right) \\ &= 1 - (1 - 1/n)^n \end{aligned}$$

So when $n = 5$, the probability is 0.67232.

(e)

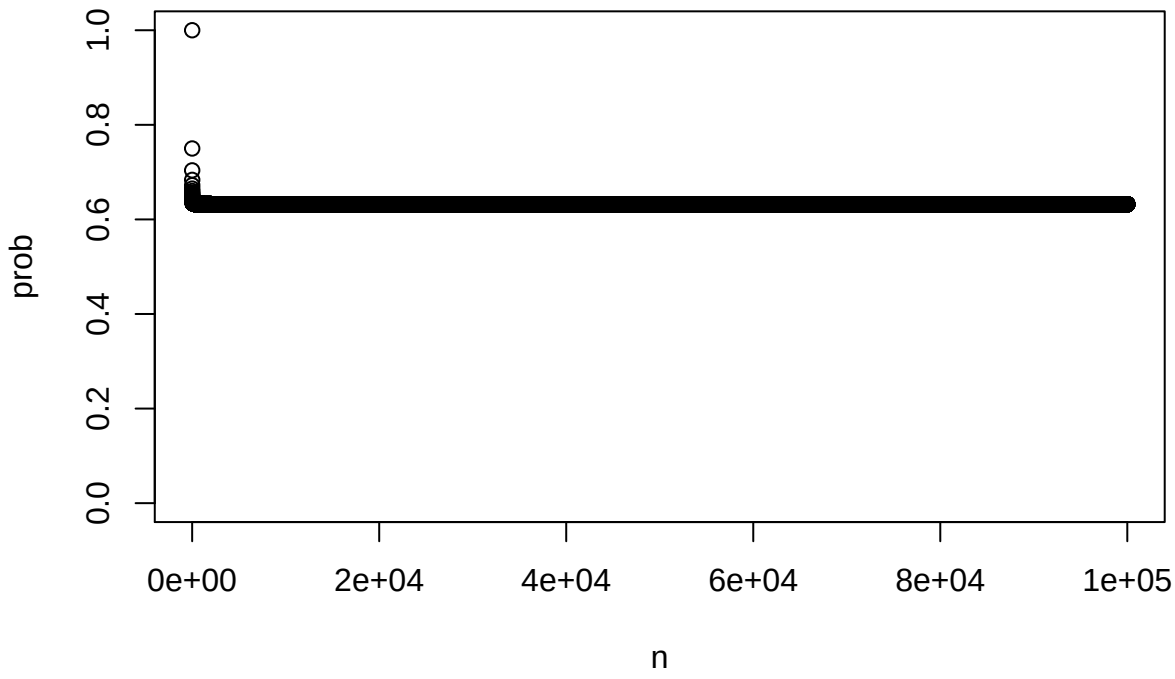
When $n = 100$, the probability is 0.6339677.

(f)

When $n = 10000$, the probability is 0.632139.

(g)

```
prob_in_sample <- function(n) {
  return(1 - (1 - 1/n)^n)
}
d <- data.frame(n = 1:1e5)
d$prob = prob_in_sample(d$n)
plot(prob ~ n, data = d, ylim = c(0, 1))
```



As n increases, the probability that any particular observation from the original sample is included appears to converge to ~0.63.

(h)

```
store=rep(NA, 1e4)
for(i in 1:1e4) {
  store[i]= sum(sample(1:100, rep=TRUE) == 4) > 0
}
mean(store)
```

```
## [1] 0.6396
```

Numerically, we recover the ~0.63 value we predicted based on the formula.

5.4 Question 8

(a)

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

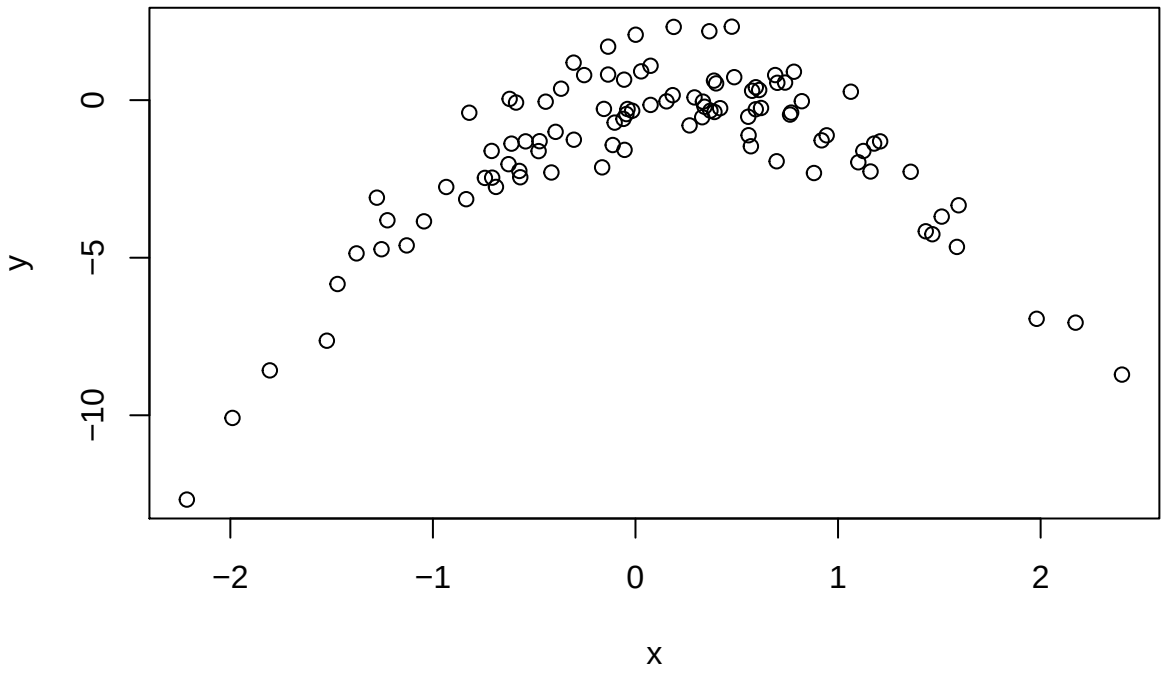
$n = 100$ and $p = 1$.

The data generating model:

$$
\begin{aligned}
x &\sim \text{normal}(0, 1^2) \\
y(x) &\sim \text{normal}(\mu, 1^2) \\
\mu &= x - 2x^2
\end{aligned}
$$

(b)

```
plot(y~x)
```



Unsurprisingly, we see a non-linear, quadratic relationship. This is what we expect based on the formula used to calculate y.

(c)

```
set.seed(2)
a <- data.frame(x = x, y = y)
```

```
models = list(
  function (d) { lm(y ~ x, data = d) },
  function (d) { lm(y ~ x + I(x^2), data = d) },
  function (d) { lm(y ~ x + I(x^2) + I(x^3), data = d) },
  function (d) { lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = d) }
)
```

```
loocv <- function(mod_fun, d) {
  mse <- 1:nrow(d)
  for (i in 1:nrow(d)) {
    m <- mod_fun(d[-i,])
    mse[i] <- (d$y[i] - predict(m, newdata = d[i,]))^2
  }
  return(mean(mse))
}
```

```
library(purrr)
map_dbl(models, ~loocv(., a))
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

(d)

```
set.seed(3)
map_dbl(models, ~loocv(., a))
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

The errors were the same. The data was generated with a separate seed, so those numbers didn't change. Moreover, the least squares algorithm is deterministic. While the coefficients have standard errors, the values obtained by least squares are fixed for fixed data (up to the expected inconsistencies of floating point arithmetic).

(e)

We would expect the model with the second degree polynomial to have the best fit, since the data was generated from x as a function with a second degree polynomial. And it turns out that one did have the lowest MSE.

(f)

```
library(broom)
map(models, ~tidy(.(a)))
```

```
## [[1]]
##          term  estimate std.error statistic      p.value
## 1 (Intercept) -1.625427 0.2619366 -6.205420 1.309300e-08
## 2           x  0.692497 0.2909418  2.380191 1.923846e-02
##
## [[2]]
##          term    estimate std.error  statistic      p.value
## 1 (Intercept)  0.05671501 0.1176555   0.482043 6.308613e-01
## 2           x  1.01716087 0.1079827   9.419666 2.403287e-15
## 3      I(x^2) -2.11892120 0.0847657 -24.997388 4.584330e-44
##
## [[3]]
##          term    estimate  std.error   statistic      p.value
## 1 (Intercept)  0.06150718 0.11950374   0.5146883 6.079538e-01
## 2           x  0.97528027 0.18728149   5.2075636 1.089350e-06
## 3      I(x^2) -2.12379099 0.08700251 -24.4106856 5.873444e-43
## 4      I(x^3)  0.01763858 0.06429037   0.2743580 7.843990e-01
##
## [[4]]
##          term     estimate  std.error   statistic      p.value
## 1 (Intercept)  0.156702953 0.13946192   1.1236253 2.640034e-01
## 2           x  1.030825643 0.19133655   5.3874999 5.174326e-07
## 3      I(x^2) -2.409898183 0.23485506 -10.2612148 4.575229e-17
## 4      I(x^3) -0.009132904 0.06722881  -0.1358481 8.922288e-01
## 5      I(x^4)  0.069785421 0.05324006   1.3107691 1.930956e-01
```

In this case, the statistical significance on each model compares well to model selection via LOOCV.

The x parameter of the linear model is statistically significant at $p = 0.019$. If we stopped there and declared victory, that would be a problem. However, in some ways it is true that x is significant, as we see in our model generating formula. For the quadratic model, both terms are very statistically significant.

For all the other polynomial models, x^3 and x^4 are not actually statistically significant. Meaning if we did our model selection by some forward/backward selection, we would have recovered the original model. This is not possible in general, and forward/backward selection is a bad idea.

3 Graduate Student Problem

```
library(tidyverse)
library(rsample)
library(magrittr)

set.seed(8456)

d <- tibble(
  x = runif(200, -pi, pi),
  y = rnorm(200, sin(x), 0.5 * abs(x) + 0.5)
)
d
```

```
## # A tibble: 200 x 2
##         x      y
##     <dbl>  <dbl>
##  1 -3.12  -1.61
##  2 -1.66  -1.99
##  3  1.23   0.393
##  4  0.213  0.474
##  5  1.99   1.74
##  6  1.26   1.51
##  7 -0.641  0.397
##  8  1.99   1.65
##  9  2.49   3.71
## 10 -2.60  -0.626
## # ... with 190 more rows
```

```
models <- tibble(
  model = list(
    function (d) { lm(y ~ x, data = d) },
    function (d) { lm(y ~ x + I(x^2), data = d) },
    function (d) { lm(y ~ x + I(x^2) + I(x^3), data = d) },
    function (d) { lm(y ~ I(sin(x)) + I(cos(x)), data = d) },
    function (d) {
      lm(y ~ 1 + I(sin(x)) + I(cos(x)) + I(sin(2 * x)) + I(cos(2 * x)), data = d)
    },
    function (d) {
      lm(
        y ~ x + I(x^2) + I(x^3) + I(sin(x)) +
          I(cos(x))  + I(sin(2 * x)) + I(cos(2 * x)),
        data = d)
    }
  )
)
models %<>% rowid_to_column()
```

```
fold_mse <- function (mod_fun, splits) {
  map_dbl(
```

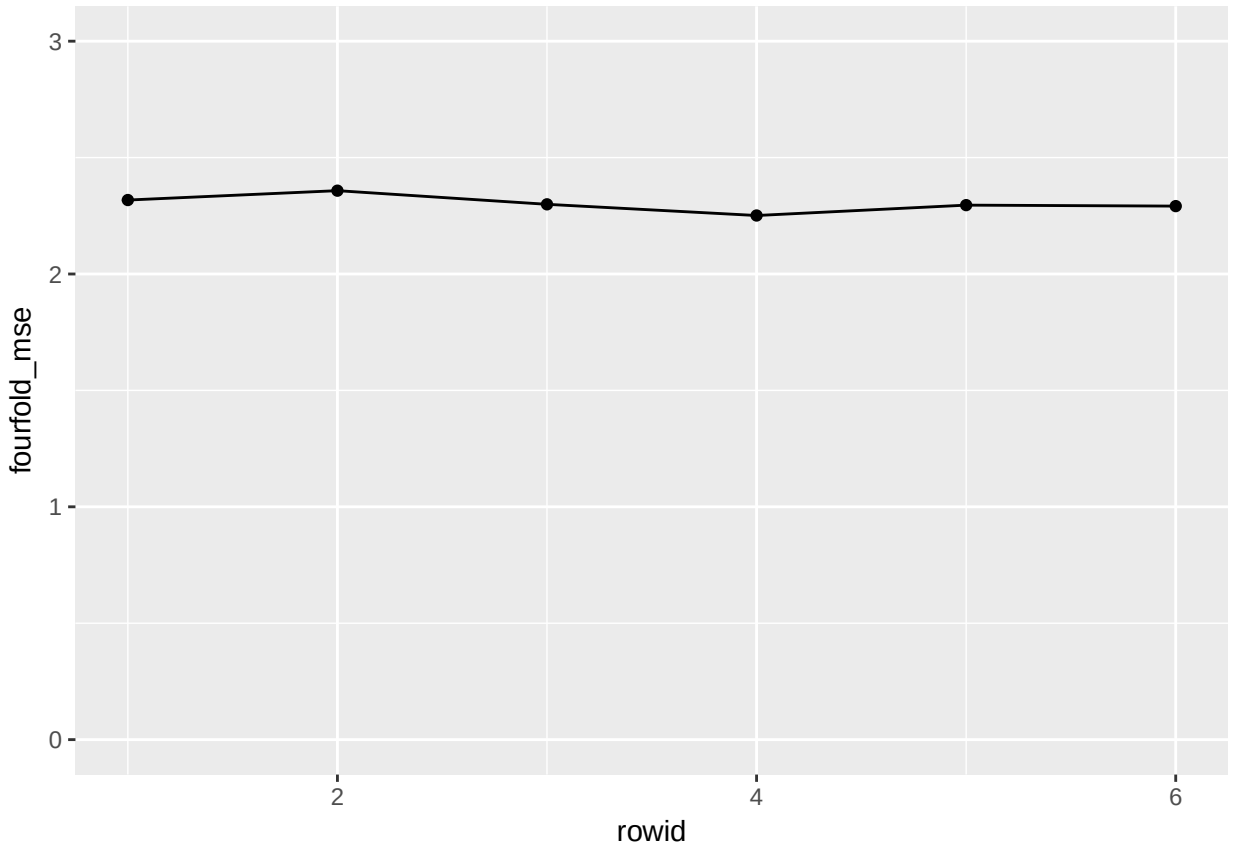
```
    splits,
    function (split) {
      trained_mod <- mod_fun(analysis(split))
      broom::augment(trained_mod, newdata = assessment(split)) %>%
        summarise(mse = mean((y - .fitted)^2)) %>%
        .$mse
    }
  )
}

cv_mse <- function (mods, splits) {
  if(!is_list(mods)) { # so cv_mse works with one function or a list of funs
    mods <- list(mods)
  }
  map_dbl(mods, ~mean(fold_mse(., splits)))
}
```

```
cv_folds <- vfold_cv(d, v = 4)
models %<>%
  mutate(fourfold_mse = cv_mse(model, cv_folds$splits))
models
```

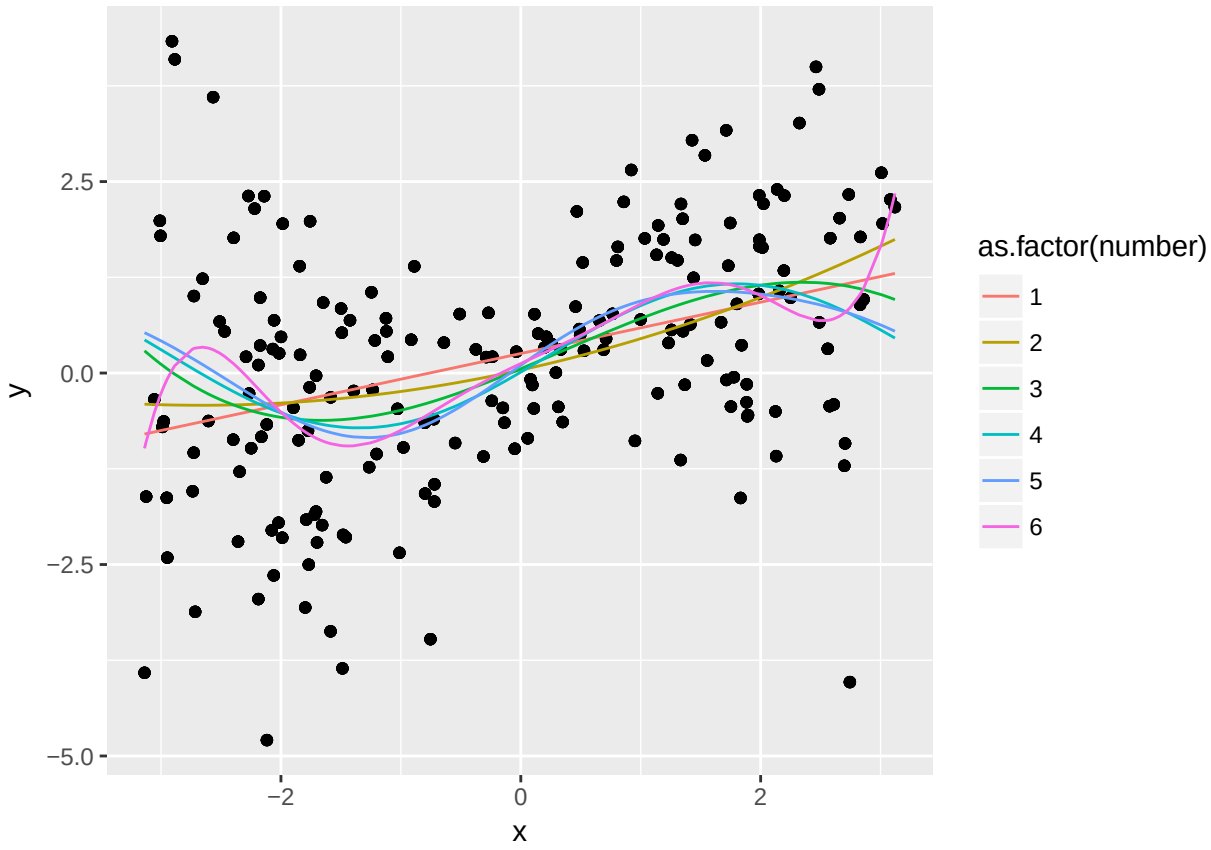
```
## # A tibble: 6 x 3
##   rowid model  fourfold_mse
##   <int> <list>        <dbl>
## 1     1 <fn>           2.32
## 2     2 <fn>           2.36
## 3     3 <fn>           2.30
## 4     4 <fn>           2.25
## 5     5 <fn>           2.30
## 6     6 <fn>           2.29
```

```
models %>%
  ggplot(aes(x = rowid, y = fourfold_mse)) +
  geom_point() +
  geom_line() +
  ylim(0, 3)
```



There doesn't seem to be much difference between any of the models, when evaluated on MSE. However, the model with the lowest MSE appears to be model 4.

```
pmap_dfr(
  list(models$model, models$rowid),
  function (fun, num) {
    d %>%
      mutate(pred = predict(fun(d), newdata = d)) %>%
      mutate(number = num)
  }
) %>%
  ggplot(aes(x = x)) +
  geom_point(aes(y = y)) +
  geom_line(aes(y = pred, color = as.factor(number)))
```


2.5
0.0
−2.5
−5.0
−2
0
2
x
y
as.factor(number)
1
2
3
4
5
6