# Chapter 5: Assessing Model Accuracy

One of the key aims of this course is to introduce you to a wide range of statistical learning techniques. Why so many? Why not just the "best one"?

There is no Best one for every situation!

↳ unless you know the true model the data come from (which you won't)

Hence, it's important to decide for any given set of data which method produces the best results.

How to decide?


not like this!

https://xkcd.com/1838/

# 1 Measuring Quality of Fit

With linear regression we talked about some ways to measure fit of the model

$R^2$, Residual standard error.

In general, we need a way to measure fit and compare *across models*.

not just for linear regression

One way could be to measure how well its predictions match the observed data. In a regression session, the most commonly used measure is the *mean-squared error (MSE)*

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

$y_i$: response for the $i^{th}$ obs; $\hat{f}(x_i)$: prediction for the $i^{th}$ obs.

Small if predictions are close to true responses.

based on training data (used to fit the model) ⇒ "training MSE"

We don't really care how well our methods work on the training data.

want our model to make good predictions on new data!

Instead, we are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen data. Why?

test data

We already know the responses for the training data!

Suppose we fit our learning method on our training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and obtain an estimate $\hat{f}$.

We can compute $\hat{f}(x_1), \ldots, \hat{f}(x_n)$, if those are close to $y_1, \ldots, y_n$ ⇒ small training MSE

But we care about:

$\hat{f}(x_0) \approx y_0$ for $(x_0, y_0)$ unseen data NOT used to fit the model.

Want to choose the model that gives lowest test MSE

Ave $(y_0 - \hat{f}(x_0))^2$ for a large # of test observations $(x_0, y_0)$.

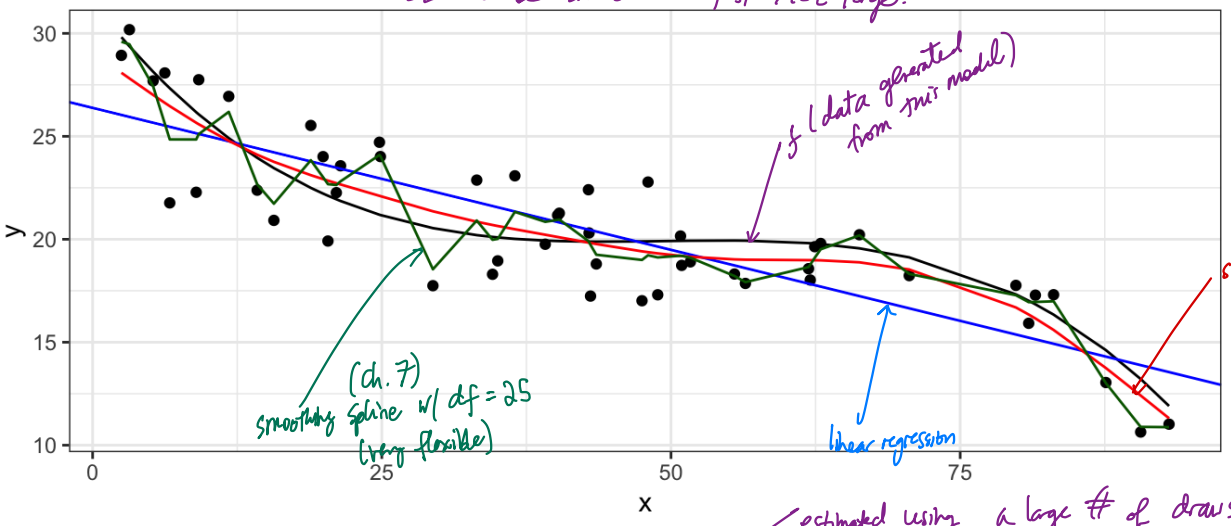
So how do we select a method that minimizes the test MSE?

Sometimes we have a test data set available to us based on the scientific problem.
↳ access to set of observations that were not used to fit the model.

But what if we don't have a test set available?

Maybe just minimize training MSE?

Problem: there is no guarantee that lowering training MSE will lower test MSE!
because many stat learning methods estimate coefs to lower training MSE
⇒ train MSE can be small but test MSE large!




estimated using a large # of draws from $f(x)+\varepsilon$ not used to fit model

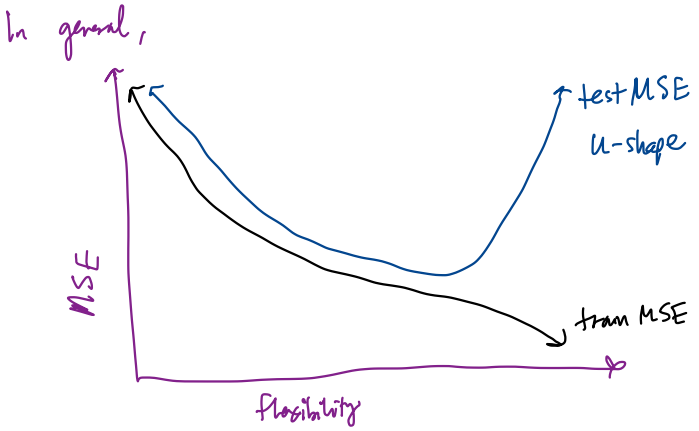
| model | df | Test MSE | Train MSE |
|---|---|---|---|
| Linear Regression | 2 | 36.0399 | 4.9654 |
| Smoothing Spline | 6 | 40.2160 | 3.5441 |
| Smoothing Spline | 25 | 38.8952 | 1.8645 |

least flexible ↓ most flexible

← linear regression has best test MSE!

not the best test MSE

best training MSE
⇒ fits training data the best!




How to choose model?
Need to estimate test MSE!
(next).

## 1.1 Classification Setting

So far, we have talked about assessing model accuracy in the regression setting, but we also need a way to assess the accuracy of classification models.

categorical response

Suppose we see to estimate $f$ on the basis of training observations where now the response is categorical. The most common approach for quantifying the accuracy is the training error rate.

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i \neq \hat{y}_i) \quad \text{where} \quad \mathbb{I}(y_i \neq \hat{y}_i) = \begin{cases} 1 & \text{if } y_i \neq \hat{y}_i \\ 0 & \text{if } y_i = \hat{y}_i \end{cases}$$

($\hat{y}_i$: predicted label for $i^{th}$ obs; $y_i$: label for $i^{th}$ obs; (correctly classified).)

This is called the *training error rate* because it is based on the data that was used to train the classifier.

As with the regression setting, we are more interested in error rates for data *not* in our training data, i.e. test data $(x_0, y_0)$.

$$\text{Test error rate} = \text{Ave}\left(\mathbb{I}(y_0 \neq \hat{y}_0)\right)$$

($\hat{y}_0$: predicted class for test obs w/ predictor $x_0$)

A good classifier is one for which the test error rate is small.

## 1.2 Bias-Variance Trade-off

The U-shape in the test MSE curve compared with flexibility is the result of two competing properties of statistical learning methods. It is possible to show that the expected test MSE, for a given test value $x_0$, can be decomposed

"irreducible error"

$$E\left[(y_0 - \hat{f}(x_0))^2\right] = \mathrm{Var}\left(\hat{f}(x_0)\right) + \left[\mathrm{Bias}\left(\hat{f}(x_0)\right)\right]^2 + \mathrm{Var}(\varepsilon)$$

$\geq 0$ $\geq 0$

average test MSE we would obtain if we repeatedly estimate $f$ at many training data sets and predict $y_0$

overall expected test MSE obtained by averaging $E\left[(y_0 - \hat{f}(x_0))^2\right]$ over many test points $(x_0, y_0)$

This tells us in order to minimize the expected test error, we need to select a statistical learning method that siulatenously achieves *low variance* and *low bias*.

Variance – the amount by which $\hat{f}$ would change if we estimated it using different training data. In general, more flexible methods have higher variance because they fit the data so closely $\Rightarrow$ new data means in big change in $\hat{f}$.

Bias – the error that is introduced by approximating a real life problem by a much simpler model.

ex. linear regression assumes a linear form. It is unlikely that any real-world problem is actually linear $\Rightarrow$ there will be some bias.

In general,

↑ flexibility $\Rightarrow$ ↓ bias + ↑ variance

how much these change determines test MSE

Similar ideas hold for the classification setting and test error rate.

# 2 Cross-Validation

As we have seen, the test error can be easily calculated when there is a test data set available.

Unfortunately this is not usually the case.

In contrast, the training error can be easily calculated.

But training can wildly underestimate test error rate.

In the absense of a very large designated test set that can be used to estimate the test error rate, what to do?

hold out some training data as test data (smartly).
↳ hopefully we have a lot of data

maybe make more data that is similar to training data (but is different)
↳ could be expensive.

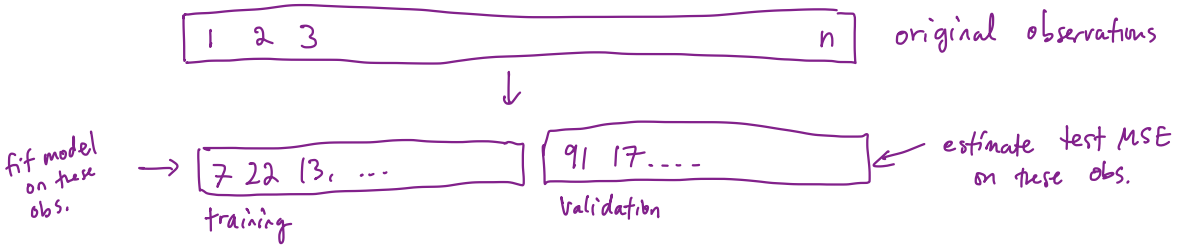
For now we will assume we are in the regression setting (quantitative response), but concepts are the same for classification.

↳ qualitative response
categorical.

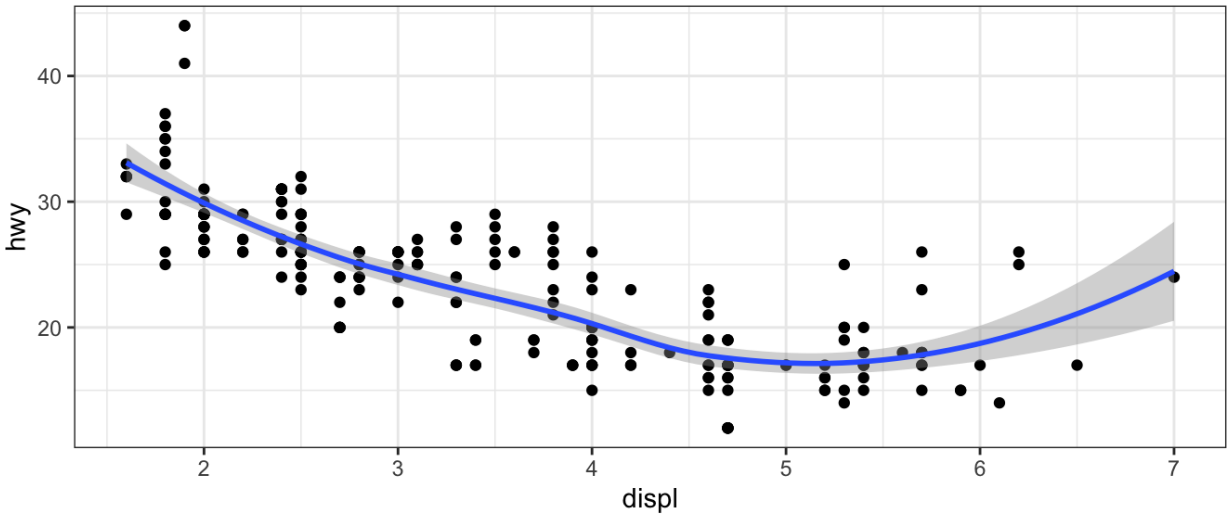
# 2.1 Validation Set

Suppose we would like to estimate the test error rate for a particular statistical learning method on a set of observations. What is the easiest thing we can think to do?

We could randomly divide the available data into two parts: training & validation.




Let's do this using the `mpg` data set. Recall we found a non-linear relationship between `displ` and `hwy` mpg.

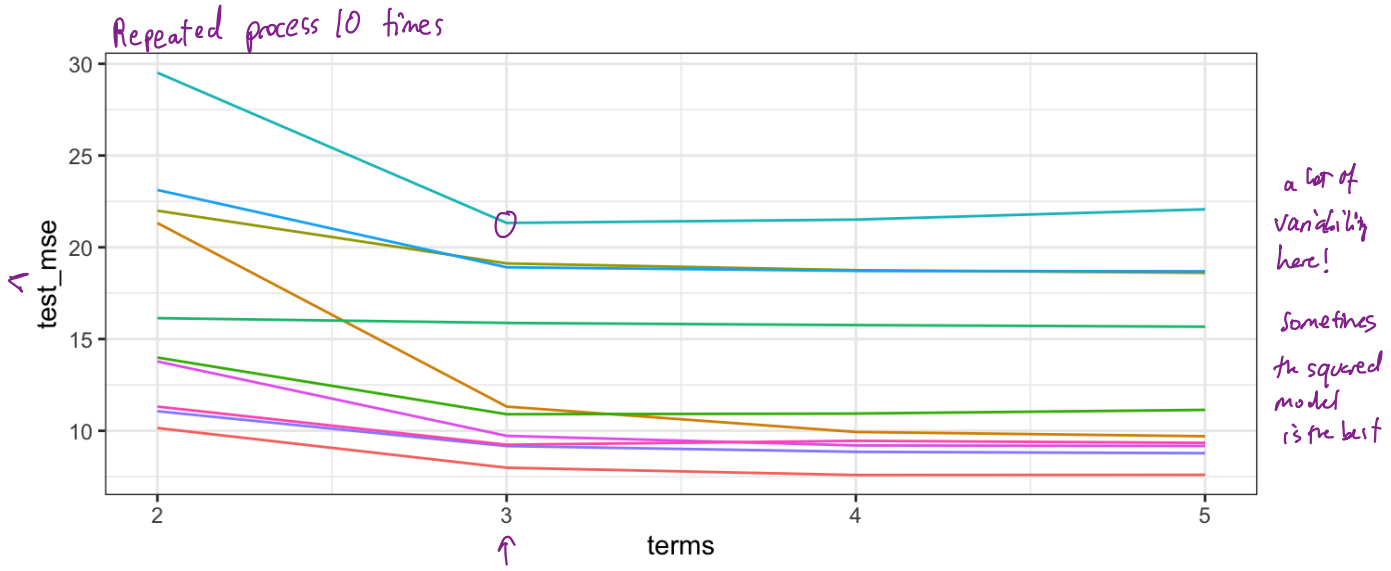



We fit the model with a squared term $\texttt{displ}^2$, but we might be wondering if we can get better predictive performance by including higher power terms!

$displ^3$, $displ^4$ ?

```r
## get index of training observations
# take 60% of observations as training and 40% for validation
n <- nrow(mpg)
trn <- seq_len(n) %in% sample(seq_len(n), round(0.6*n))

## fit models
m0 <- lm(hwy ~ displ, data = mpg[trn, ])
m1 <- lm(hwy ~ displ + I(displ^2), data = mpg[trn, ])
m2 <- lm(hwy ~ displ + I(displ^2) + I(displ^3), data = mpg[trn, ])
m3 <- lm(hwy ~ displ + I(displ^2) + I(displ^3) + I(displ^4), data =
  mpg[trn, ])

## predict on validation set
pred0 <- predict(m0, mpg[!trn, ])
pred1 <- predict(m1, mpg[!trn, ])
pred2 <- predict(m2, mpg[!trn, ])
pred3 <- predict(m3, mpg[!trn, ])

## estimate test MSE
true_hwy <-  mpg[!trn, ]$hwy # truth vector

data.frame(terms = 2, model = "linear", true = true_hwy, pred =
  pred0) %>%
  bind_rows(data.frame(terms = 3, model = "quadratic", true =
  true_hwy, pred = pred1)) %>%
  bind_rows(data.frame(terms = 4, model = "cubic", true = true_hwy,
  pred = pred2)) %>%
  bind_rows(data.frame(terms = 5, model = "quartic", true = true_hwy,
  pred = pred3)) %>% ## bind predictions together
  mutate(se = (true - pred)^2) %>% # squared errors
  group_by(terms, model) %>% # group by model
  summarise(test_mse = mean(se)) %>% ## get test mse
  kable() ## pretty table
```

| terms | model | test_mse |
|---|---|---|
| 2 | linear | 14.17119 |
| 3 | quadratic | 11.26710 |
| 4 | cubic | 11.08535 |
| 5 | quartic | 11.04907 |



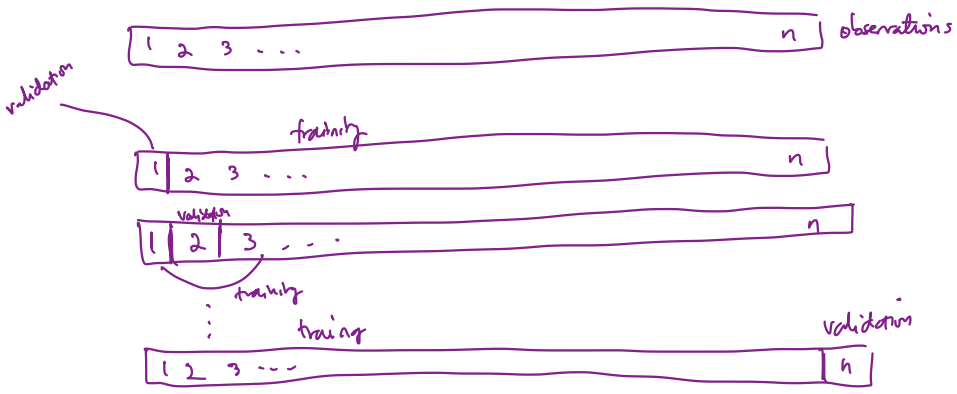

- The validation estimate of test MSE is highly variable! Depends on which observations were held out!
- Only a subset used to fit model, Since statistical models tend to do better with more data, the validation set error can overestimate the test error.

⇒ Cross-validation is a method to address these weaknesses ...

## 2.2 Leave-One-Out Cross Validation

*Leave-one-out cross-validation* (LOOCV) is closely related to the validation set approach, but it attempts to address the method's drawbacks.

LOO CV still splits data into 2 parts, but now a single obs. is used for validation,




(1) fit model on n-1 observations

(2) $\hat{y}_i$ prediction for heldout obs.

$MSE_i = (y_i - \hat{y}_i)^2$ unbiased for test error but highly variable!

The LOOCV estimate for the test MSE is

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

LOOCV has a couple major advantages and a few disadvantages. (over the validation method).

Advantages

- less bias
  since we fit using n-1 obs (instead of $\approx \frac{n}{2}$ for validation approach)
  $\Rightarrow$ LOO CV doesn't overestimate test error as much as validation approach.
- No randomness in the approach $\Rightarrow$ will get the same result every time.

Disadvantages

- sometimes stat learning can be expensive to fit (i.e. on the order of days)
  LOO CV requires us to fit the model n times
  $\Rightarrow$ could be slow!

Let's fit models of increasing flexibility to hwy ~ displ on mpg.

```
## perform LOOCV on the mpg dataset
res <- data.frame() ## store results
for(i in seq_len(n)) { # repeat for each observation
  trn <- seq_len(n) != i # leave one out
              vector of T/F indicating which obs to leave out  i=1,...,n
  ## fit models
  m0 <- lm(hwy ~ displ, data = mpg[trn, ])
  m1 <- lm(hwy ~ displ + I(displ^2), data = mpg[trn, ])
  m2 <- lm(hwy ~ displ + I(displ^2) + I(displ^3), data = mpg[trn, ])
  m3 <- lm(hwy ~ displ + I(displ^2) + I(displ^3) + I(displ^4), data =
  mpg[trn, ])

  ## predict on validation set
  pred0 <- predict(m0, mpg[!trn, ])
  pred1 <- predict(m1, mpg[!trn, ])
  pred2 <- predict(m2, mpg[!trn, ])
  pred3 <- predict(m3, mpg[!trn, ])

  ## estimate test MSE
  true_hwy <- mpg[!trn, ]$hwy # get truth vector

  res %>% ## store results for use outside the loop
    bind_rows(data.frame(terms = 2, model = "linear", true =
  true_hwy, pred = pred0)) %>%
    bind_rows(data.frame(terms = 3, model = "quadratic", true =
  true_hwy, pred = pred1)) %>%
    bind_rows(data.frame(terms = 4, model = "cubic", true = true_hwy,
  pred = pred2)) %>%
    bind_rows(data.frame(terms = 5, model = "quartic", true =
  true_hwy, pred = pred3)) %>% ## bind predictions together
    mutate(mse = (true - pred)^2) -> res
}
            ↑ MSE_i

res %>%
  group_by(terms, model) %>%
  summarise(LOOCV_test_MSE = mean(mse)) %>%
  kable()
```

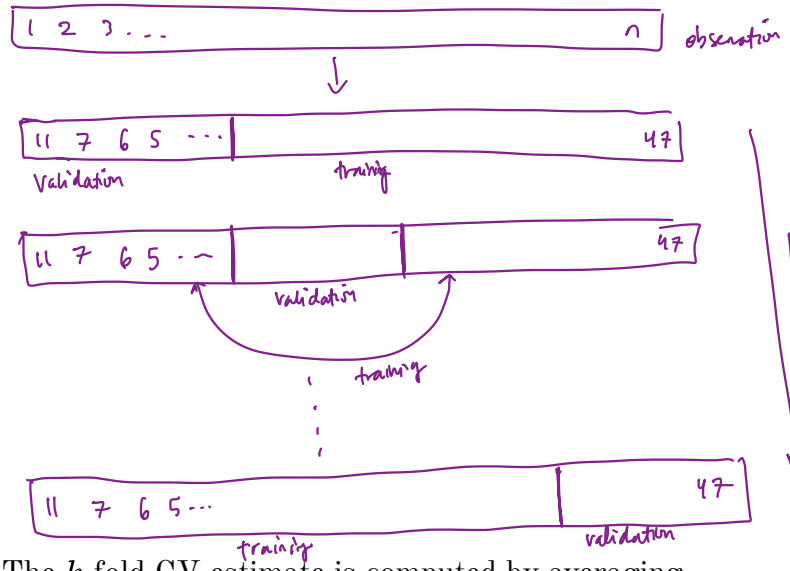
$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$

| terms | model | LOOCV_test_MSE |
|---|---|---|
| 2 | linear | 14.92437 |
| 3 | quadratic | 11.91775 |
| 4 | cubic | 11.78047 |
| 5 | quartic | 11.93978 |

We would choose cubic model level of flexibility w/ lowest $CV_{(n)}$ estimate of test MSE.

## 2.3 k-Fold Cross Validation

An alternative to LOOCV is $k$-fold CV. → randomly divide the set of observations into k groups or folds




(1) hold out 1 fold
fit model on remaining k-1 folds.

(2) predict held out fold
compute $MSE_i$ for left out fold.

The $k$-fold CV estimate is computed by averaging

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{|F_i|}\sum_{j\in F_i}(y_j - \hat{y}_j)^2$$

$F_i$: fold $i$

usually use $k=5$ or $k=10$.

Why $k$-fold over LOOCV?

LOO CV is a special case of k-fold CV in which $k=n$.

Computational advantage! Now we only have to fit the model k times (not n)

Also other advantages due to bias-variance trade-off (more later).

```r
## perform k-fold on the mpg dataset
res <- data.frame() ## store results

## get the folds
k <- 10
folds <- sample(seq_len(10), n, replace = TRUE) ## approximately
  equal sized

for(i in seq_len(k)) { # repeat for each observation
  trn <- folds != i # leave ith fold out

  ## fit models
  m0 <- lm(hwy ~ displ, data = mpg[trn, ])
  m1 <- lm(hwy ~ displ + I(displ^2), data = mpg[trn, ])
  m2 <- lm(hwy ~ displ + I(displ^2) + I(displ^3), data = mpg[trn, ])
  m3 <- lm(hwy ~ displ + I(displ^2) + I(displ^3) + I(displ^4), data =
  mpg[trn, ])

  ## predict on validation set
  pred0 <- predict(m0, mpg[!trn, ])
  pred1 <- predict(m1, mpg[!trn, ])
  pred2 <- predict(m2, mpg[!trn, ])
  pred3 <- predict(m3, mpg[!trn, ])

  ## estimate test MSE
  true_hwy <- mpg[!trn, ]$hwy # get truth vector

  data.frame(terms = 2, model = "linear", true = true_hwy, pred =
  pred0) %>%
    bind_rows(data.frame(terms = 3, model = "quadratic", true =
  true_hwy, pred = pred1)) %>%
    bind_rows(data.frame(terms = 4, model = "cubic", true = true_hwy,
  pred = pred2)) %>%
    bind_rows(data.frame(terms = 5, model = "quartic", true =
  true_hwy, pred = pred3)) %>% ## bind predictions together
    mutate(mse = (true - pred)^2) %>%
    group_by(terms, model) %>%
    summarise(mse = mean(mse)) -> test_mse_k

  res %>% bind_rows(test_mse_k) -> res
}
```

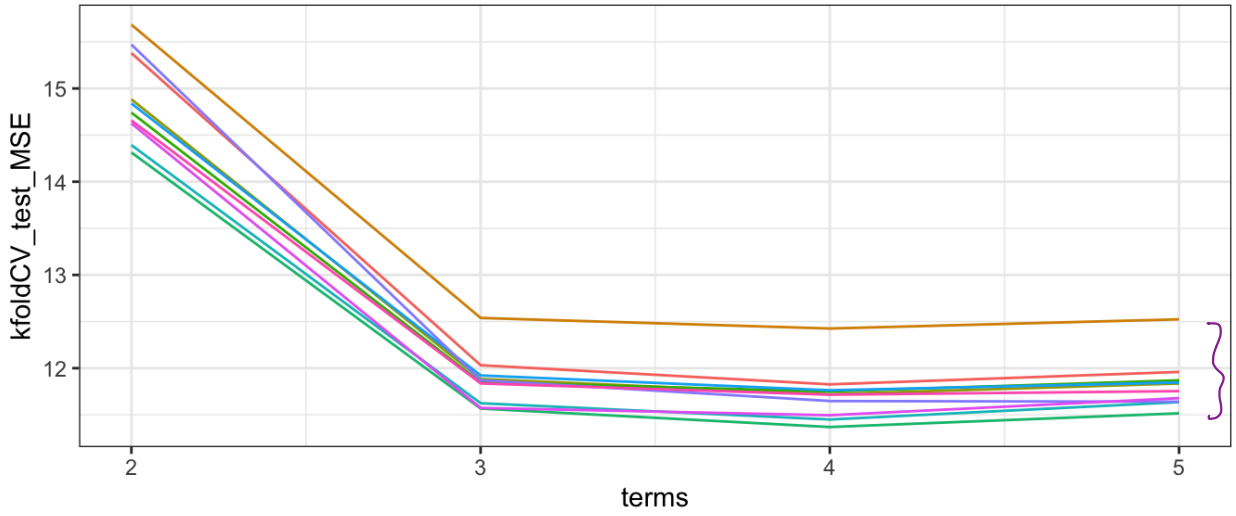
$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$

```
res %>%
  group_by(terms, model) %>%
  summarise(kfoldCV_test_MSE = mean(mse)) %>%
  kable()
```

| terms | model | kfoldCV_test_MSE |
|---|---|---|
| 2 | linear | 14.77098 |
| 3 | quadratic | 12.14423 |
| 4 | cubic | 11.94037 |
| 5 | quartic | 11.78830 |

close

Now again there is randomness in the method.




very similar results across different selections of folds.

When we perform CV we are often interested in estimating test error.
We then use test error to help pick model by selecting lowest CV error.

or pick some parameters called "tuning" the model.

## 2.4 Bias-Variance Trade-off for $k$-Fold Cross Validation

$k$-Fold CV with $k < n$ has a computational advantage to LOOCV.

There is a less obvious (but maybe more important) advantage

Sometimes we will get better accuracy in estimating test error w/ k-fold than LOO.

We know the validation approach can overestimate the test error because we use only half of the data to fit the statistical learning method.

By this logic LOO CV gives approximately unbiased estimates of the test error rate (use $n-1 \approx n$ obs. to fit model).

K-fold gives an intermediate amount of bias (use $\frac{(K-1)n}{K}$ obs to fit model)

$\Rightarrow$ LOO CV gives lowest bias.

But we know that bias is only half the story! We also need to consider the procedure's variance.

LOO CV has higher variance than k-fold w/ $k<n$

Why?

LOO CV fits n models on almost identical data points $\Rightarrow$ averages outputs highly correlated w/ each other

K-fold averages k outputs w/ more different observations (overlap is smaller).

$\Rightarrow$ mean of highly (positively) correlated quantities has higher variance than mean of less correlated values.

$\Rightarrow$ LOO CV has higher variance than k-fold CV.

To summarise, there is a bias-variance trade-off associated with the choice of $k$ in $k$-fold CV. Typically we use $k = 5$ or $k = 10$ because these have been shown empirically to yield test error rates closest to the truth.

↓

in numerical experiments (simulation)

## 2.5 Cross-Validation for Classification Problems

So far we have talked only about CV for regression problems.

numeric response

use MSE to quantify test error

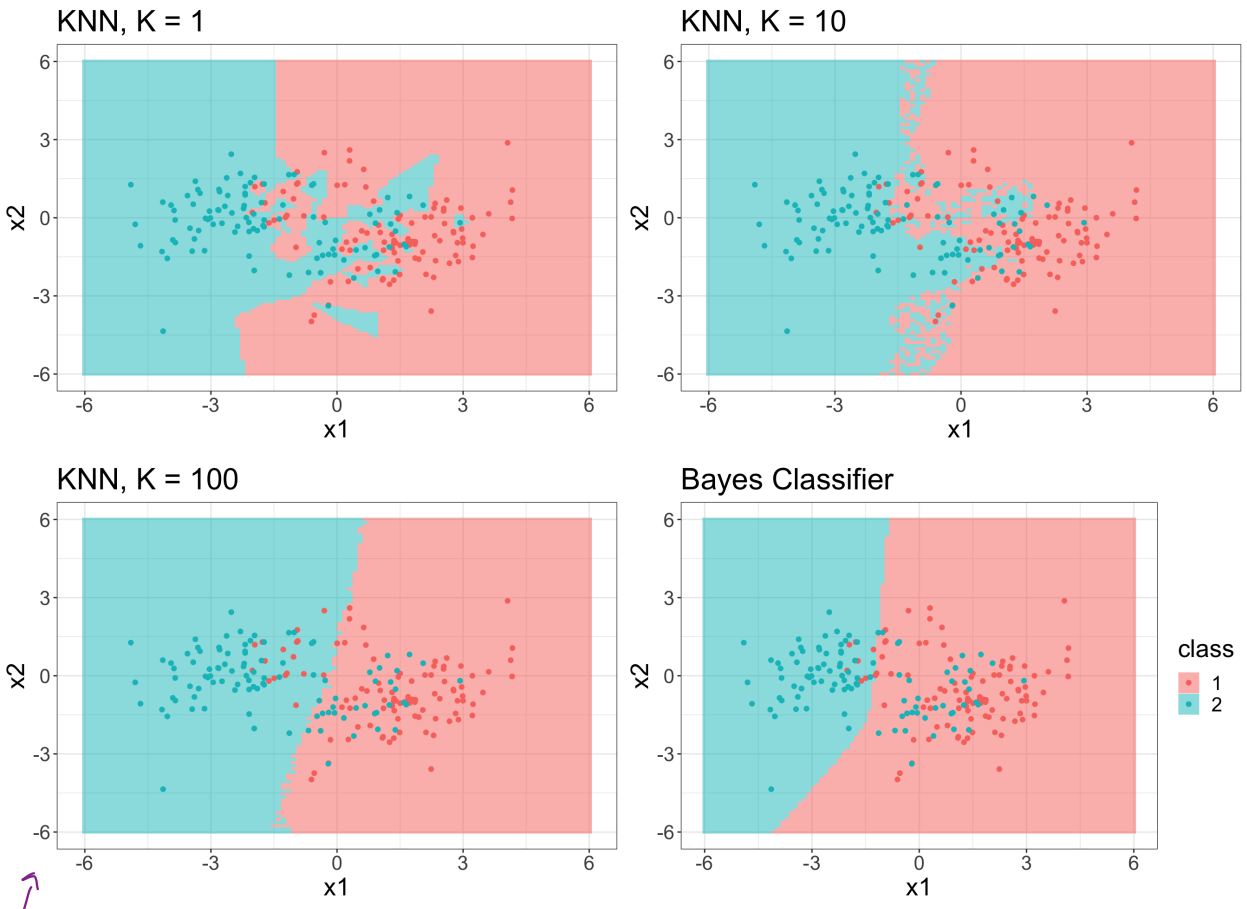
categorical response

But CV can also be very useful for classification problems! For example, the LOOCV error rate for classification problems takes the form

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i$$

$$\text{where } Err_i = \mathbb{I}(y_i \neq \hat{y}_i) = \begin{cases} 1 & y_i \neq \hat{y}_i \\ 0 & \text{o.w.} \end{cases}$$

k-fold and validation errors estimated accordingly.




recall simulated example.

Can choose k using CV approach!

"tune" KNN!

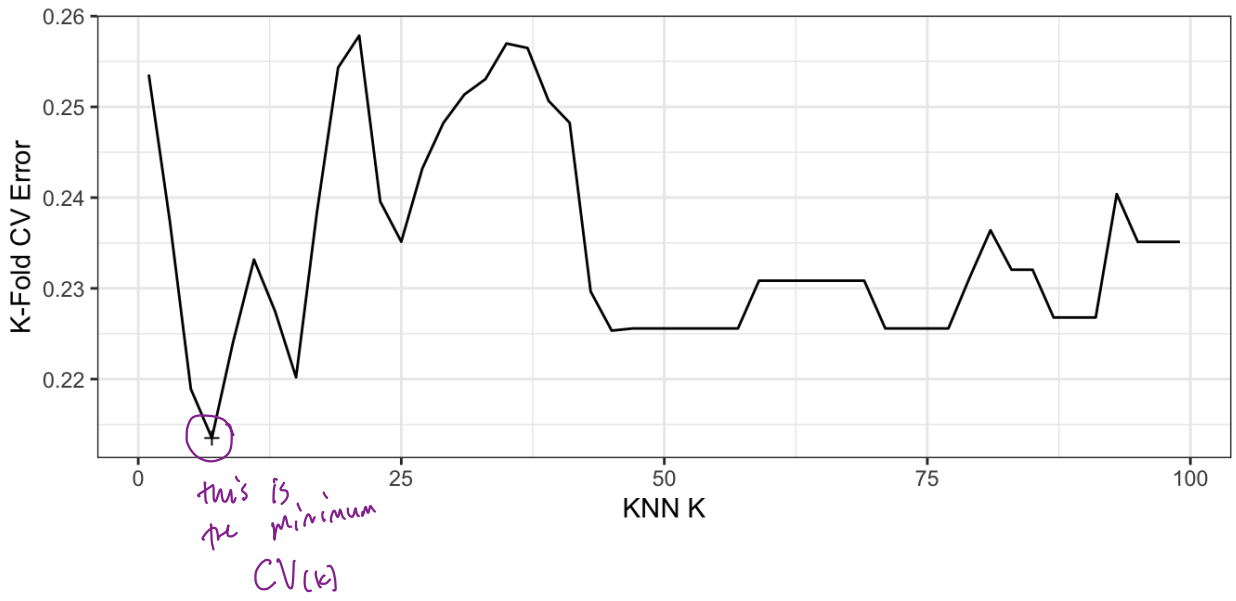
10-fold CV

split into folds

```
k_fold <- 10
cv_label <- sample(seq_len(k_fold), nrow(train), replace = TRUE)
err <- rep(NA, k) # store errors for each flexibility level

for(k in seq(1, 100, by = 2)) {
  err_cv <- rep(NA, k_fold) # store error rates for each fold
  for(ell in seq_len(k_fold)) {
    trn_vec <- cv_label != ell # fit model on these
    tst_vec <- cv_label == ell # estimate error on these

    ## fit knn
    knn_fit <- knn(train[trn_vec, -1], train[tst_vec, -1],
  train[trn_vec, ]$class, k = k)
    ## error rate
    err_cv[ell] <- mean(knn_fit != train[tst_vec, ]$class)
  }
  err[k] <- mean(err_cv)
}
err <- na.omit(err)
```

bunch of KNN k-values

KNN k-value.




this is the minimum CV(k)

Minimum CV error of 0.2135 found at $K = 7$.

So we might choose k=7 and fit KNN on whole training data set w/ k=7.