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 model for every situation!

↳ unless you know the true model the data comes 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?

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

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 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)*

Sometimes talk about root MSE (RMSE)

$\text{RMSE} = \sqrt{\text{MSE}}$

(same scale as response $\Rightarrow$ more interpretable)

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

response for i^{th} training observation

prediction for i^{th} training observation

small if predictions are close to responses

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

We don't really care how well our methods work on the training 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 response values for the training data!

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

Can compute $\hat{f}(x_1),\dots,\hat{f}(x_n)$ if these are close to $y_1,\dots,y_n \Rightarrow$ small training MSE.

But we care more about:

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

Compute $\underbrace{\text{Ave}\left(y_0 - \hat{f}(x_0)\right)^2}_{\text{test MSE}}$ for large # of test observations (x_0,y_0).

Want to choose model with lowest test MSE.

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

Sometimes we have a test data set available to us based on scientific problem.

↳ access to a set of observations that were not used to fit the model.

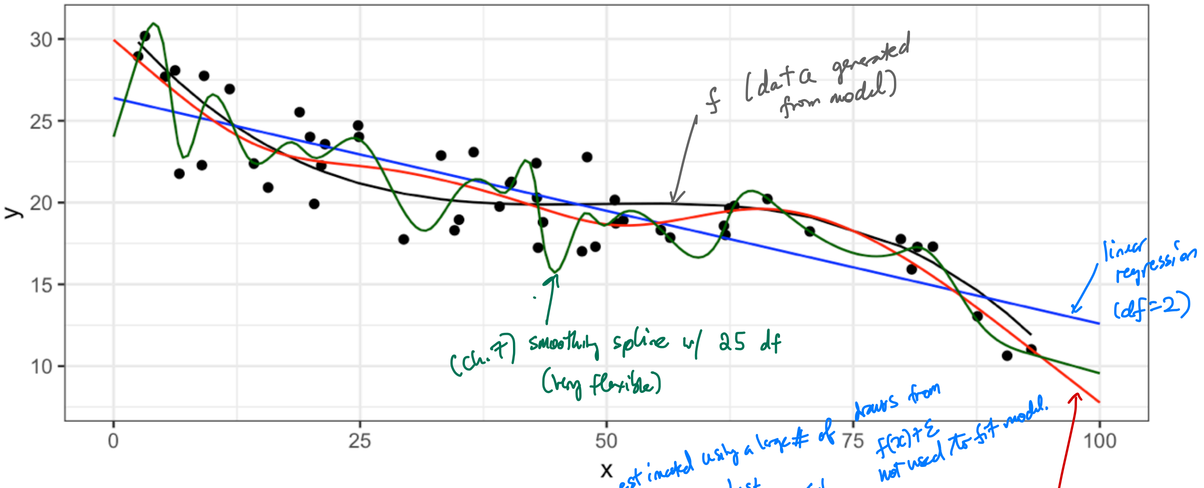
But what if we don't have a test set available?

Maybe we just minimize train MSE?

Problem: there is no guarantee lowering training MSE lowers test MSE!

Because many stat learning methods estimate coef's to lower train MSE

⟹ train MSE can be small but test MSE large!



estimated using a large # of draws from f(x)+ε not used to fit model.

best test MSE!

smoothing spline w/ df=6

model	df	Test MSE	Train MSE
Linear Regression	2	34.4168	4.9654
Smoothing Spline	6	38.9525	3.5248
Smoothing Spline	25	39.9288	2.3107

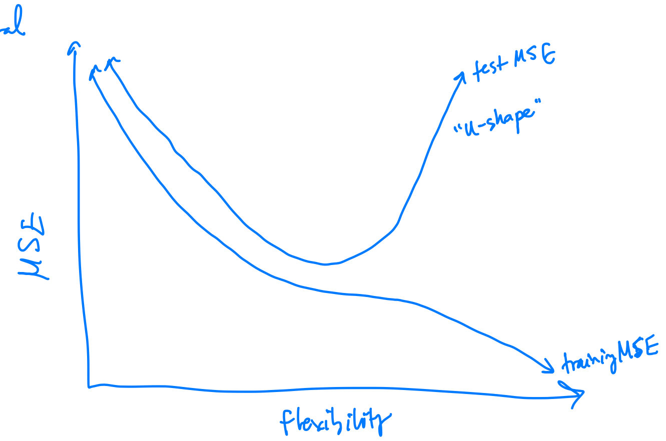
least flexible ↓ most flexible.

worst training MSE

best training MSE! fits training data the best.

not the best test MSE

In general



How to choose the proper 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 seek 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.

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

true label for ith training obs.

predicted label for ith training obs.

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

Could also talk about "training accuracy" = 1 − training error rate.

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

The test error rate is

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

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

average test MSE we would obtain if we repeatedly obtained many training data sets, estimated f, and predicted @ x_0.

irreducible error.

$$\rightarrow \underbrace{E\left[\left(y_0 - \hat{f}(x_0)\right)^2\right]} = \underbrace{Var\left(\hat{f}(x_0)\right)}_{\geq 0} + \underbrace{\left[Bias\left(\hat{f}(x_0)\right)\right]^2}_{\geq 0} + Var(\varepsilon)$$

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 w/ differet training data.

In general, more flexible methods have higher variance because they fit the data so closely $\Rightarrow$ new data means big changes in $\hat{f}$.

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

ex. linear regression assumes a linear form. It is unlikely any real-world problems are actually linear

In general:

$\uparrow$ flexibility $\Rightarrow$ $\downarrow$ bias + $\uparrow$ 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 always the case.

In contrast, the training error can be easily calculated.

But training error can wildly underestimate test error.

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

Split our data.

- ↳ randomly
- ↳ systematic split?

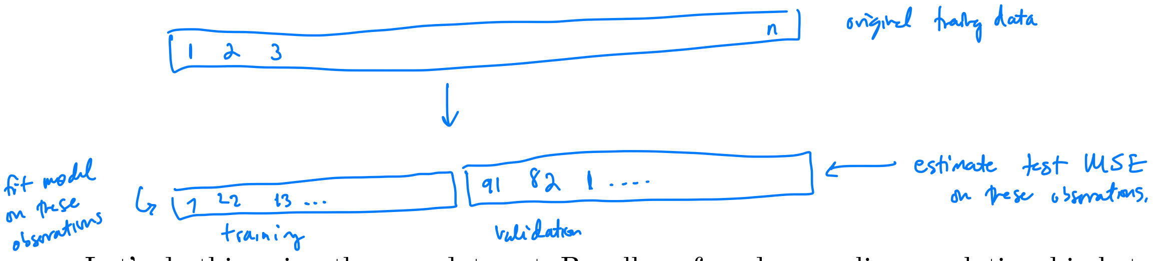
For now we will assume we are in the regression setting (quantitative response), but concepts are the same for classification.

Categorical response (replace MSE w/ error rate).

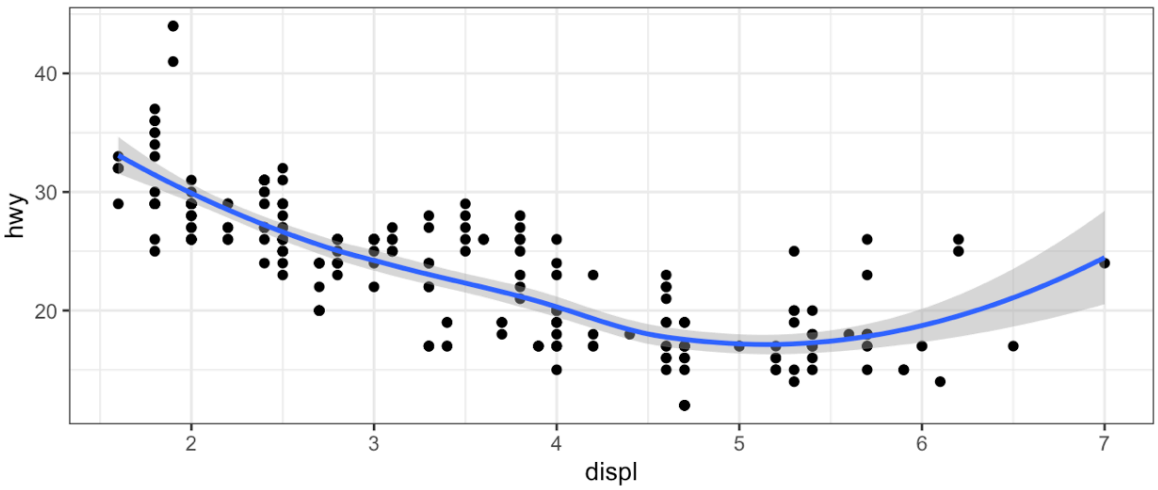
2.1 Validation Set

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 set into two parts: training and validation.


1 2 3
n
original training data
fit model on these observations
1 22 13 ...
training
91 82 1 ....
validation
estimate test MSE on these observations.

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 **`displ`**^{2}, but we might be wondering if we can get better predictive performance by including higher power terms!

$displ^3$

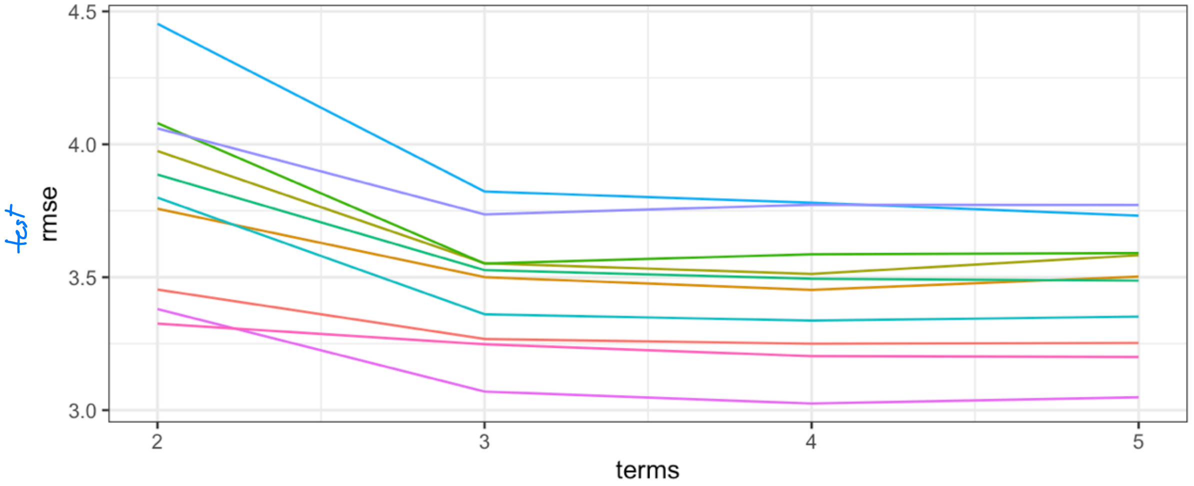
$displ^4$

```
## get index of training observations
# take 60% of observations as training and 40% for validation
mpg_val <- validation_split(mpg, prop = 0.6)

## models
lm_spec <- linear_reg()

linear_recipe <- recipe(hwy ~ displ, data = mpg)
quad_recipe <- linear_recipe |> step_mutate(displ2 = displ^2)
cubic_recipe <- quad_recipe |> step_mutate(displ3 = displ^3)
quart_recipe <- cubic_recipe |> step_mutate(displ4 = displ^4)

m0 <- workflow() |> add_model(lm_spec) |> add_recipe(linear_recipe) |>
         fit_resamples(resamples = mpg_val)
m1 <- workflow() |> add_model(lm_spec) |> add_recipe(quad_recipe) |>
         fit_resamples(resamples = mpg_val)
m2 <- workflow() |> add_model(lm_spec) |> add_recipe(cubic_recipe) |>
         fit_resamples(resamples = mpg_val)
m3 <- workflow() |> add_model(lm_spec) |> add_recipe(quart_recipe) |>
         fit_resamples(resamples = mpg_val)

## estimate test MSE
collect_metrics(m0) |> mutate(model = "linear") |>
  bind_rows(collect_metrics(m1) |> mutate(model = "quadratic")) |>
  bind_rows(collect_metrics(m2) |> mutate(model = "cubic")) |>
  bind_rows(collect_metrics(m3) |> mutate(model = "quartic")) |>
  select(model, .metric, mean) |>
  pivot_wider(names_from = .metric, values_from = mean) |>
  select(-rsq) |>
  kable()
```

model	rmse
linear	4.318968
quadratic	3.882112
cubic	3.866194
quartic	3.860612

← lowest value => best model?

Repeated process 10 times



a lot of variability here

- The validation estimate of the test MSE is highly variable. Depends on which observations are held out!

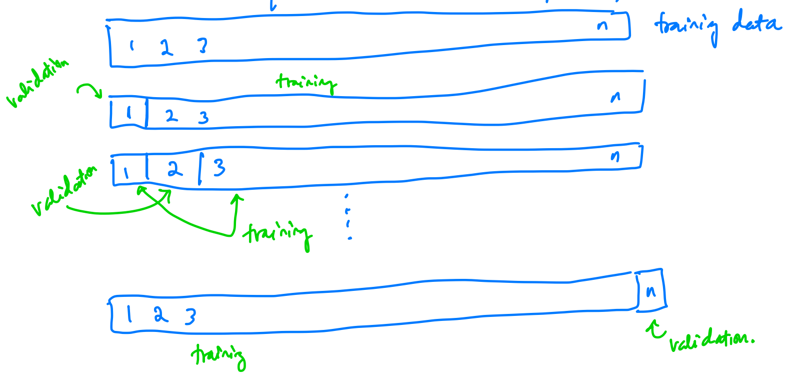
- only a subset of training data used to fit model. Since statistical models tend to do better w/ more data, the validation set error can overestimate test error.

$\Longrightarrow$ 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.

LOOCV still splits data into 2 parts, but now a single observation is used for validation.



① fit model on n-1 observations

② prediction $\hat{y}$ for heldout observation.

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

The LOOCV estimate for the test MSE is

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

LOOCV has a couple major advantages and a few disadvantages. (compared to validation approach).

Advantages

- less bias
- since we fit using n-1 observations (instead of ≈ $\frac{n}{2}$ for validation)

  ⇒ LOOCV does not overestimate true test error as much as validation approach.
- no randomness in this approach. ⇒ will get same result every time.

Disadvantages

- sometimes stat learning can be expensive to fit (i.e. on order of days)

  LOOCV requires us to fit model n times

  ↳ could be very very slow.

```
## perform LOOCV on the mpg dataset
mpg_loocv <- vfold_cv(mpg, v = nrow(mpg))

## models
m0 <- workflow() |> add_model(lm_spec) |> add_recipe(linear_recipe) |>
        fit_resamples(resamples = mpg_loocv)
m1 <- workflow() |> add_model(lm_spec) |> add_recipe(quad_recipe) |>
        fit_resamples(resamples = mpg_loocv)
m2 <- workflow() |> add_model(lm_spec) |> add_recipe(cubic_recipe) |>
        fit_resamples(resamples = mpg_loocv)
m3 <- workflow() |> add_model(lm_spec) |> add_recipe(quart_recipe) |>
        fit_resamples(resamples = mpg_loocv)

## estimate test MSE
collect_metrics(m0) |> mutate(model = "linear") |>
  bind_rows(collect_metrics(m1) |> mutate(model = "quadratic")) |>
  bind_rows(collect_metrics(m2) |> mutate(model = "cubic")) |>
  bind_rows(collect_metrics(m3) |> mutate(model = "quartic")) |>
  select(model, .metric, mean) |>
  pivot_wider(names_from = .metric, values_from = mean) |>
  select(-rsq) |>
  kable()
```

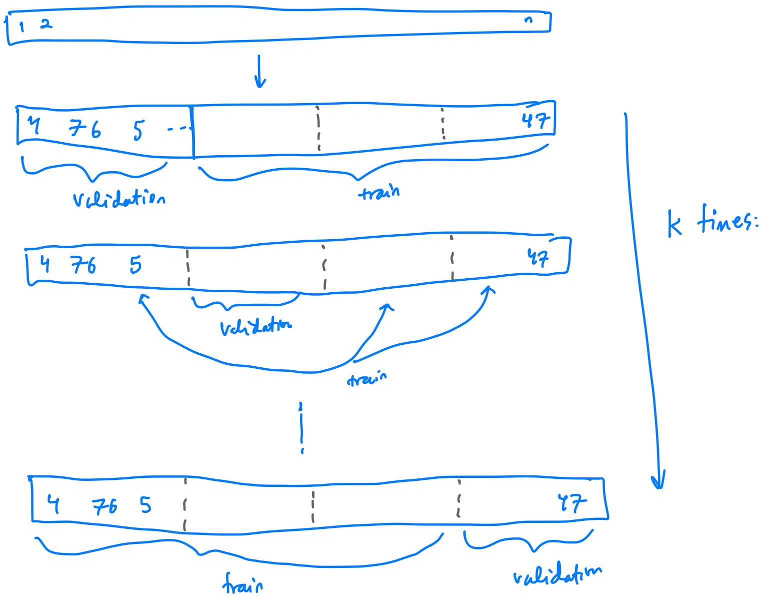
$*$ n

model	rmse
linear	2.808356
quadratic	2.675896
cubic	2.615363
quartic	2.643536

choose level of flexibility w/ lowest CV(n) estimate of test RMSE.

2.3 k-Fold Cross Validation

An alternative to LOOCV is *k*-fold CV. → randomly divide the training data into k groups or "folds".



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

② predict the held out fold

get MSE; for left out fold.

The k-fold CV estimate is computed by averaging

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{|F_k|}\sum_{j\in \underbrace{F_k}_{\text{fold k.}}}(y_j-\hat{y}_j)^2$$

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

Why k-fold over LOOCV?

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

Computational advantage! Now have to fit model k times (not n).

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

```r
## perform k-fold on the mpg dataset
mpg_10foldcv <- vfold_cv(mpg, v = 10)
```

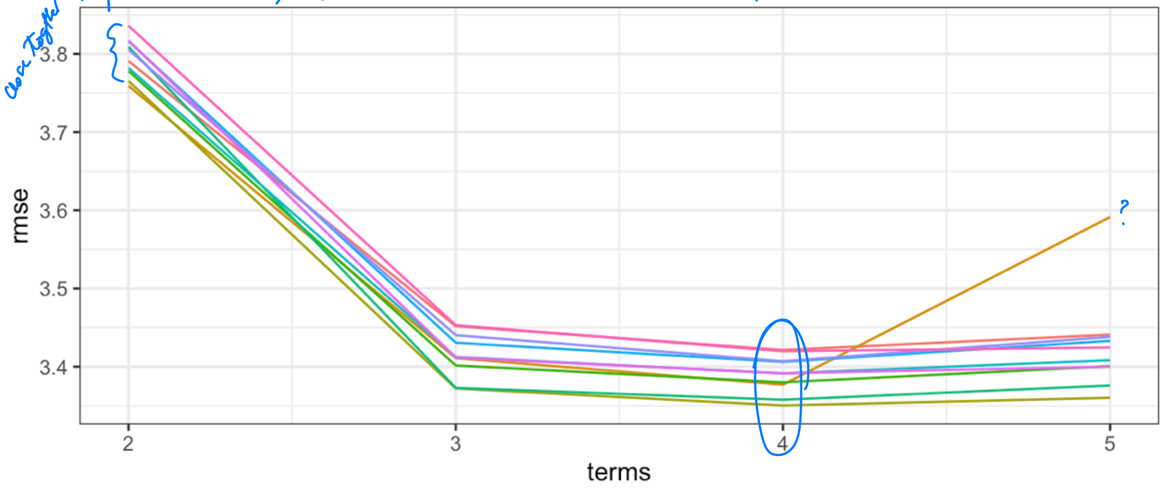
k-fold, k=10.

```r
## models
m0 <- workflow() |> add_model(lm_spec) |> add_recipe(linear_recipe) |>
         fit_resamples(resamples = mpg_10foldcv)
m1 <- workflow() |> add_model(lm_spec) |> add_recipe(quad_recipe) |>
         fit_resamples(resamples = mpg_10foldcv)
m2 <- workflow() |> add_model(lm_spec) |> add_recipe(cubic_recipe) |>
         fit_resamples(resamples = mpg_10foldcv)
m3 <- workflow() |> add_model(lm_spec) |> add_recipe(quart_recipe) |>
         fit_resamples(resamples = mpg_10foldcv)

## estimate test MSE
collect_metrics(m0) |> mutate(model = "linear") |>
  bind_rows(collect_metrics(m1) |> mutate(model = "quadratic")) |>
  bind_rows(collect_metrics(m2) |> mutate(model = "cubic")) |>
  bind_rows(collect_metrics(m3) |> mutate(model = "quartic")) |>
  select(model, .metric, mean) |>
  pivot_wider(names_from = .metric, values_from = mean) |>
  select(-rsq) |>
  kable()
```

model	rmse
linear	3.805566
quadratic	3.432052
cubic	3.409391
quartic	3.408420

close

Repeated 10 times, again there is some randomness in the assignment.



When we perform CV, we are interested in estimating test error

Most often we use it to find the minimum CV error to help us choose a model (or a set of model parameters).

↑ "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 also a less obvious advantage (but potentially more important).

↳ often k-fold CV gives more accurate estimates of test error than LOOCV!

bias

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

By this logic, LOOCV gives approximately unbiased estimates of the test error (uses $n-1 \approx n$ points to fit).

K-fold gives intermediate levels of bias. (uses $\frac{(k-1)}{k}n$ obs to fit).

$\Rightarrow$ LOOCV gives lowest bias.

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

LOOCV has higher variance than k-fold CV when $k<n$.

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

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

mean of highly correlated quantities has higher variance than mean of less correlated values!

$\mathrm{Var}(X_1+X_2) = \mathrm{Var}\,X_1 + \mathrm{Var}\,X_2 + 2\mathrm{Cov}(X_1,X_2)$.

$\Rightarrow$ LOOCV estimate has higher variance than k-fold.

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.

2.5 Cross-Validation for Classification Problems

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

numeric response

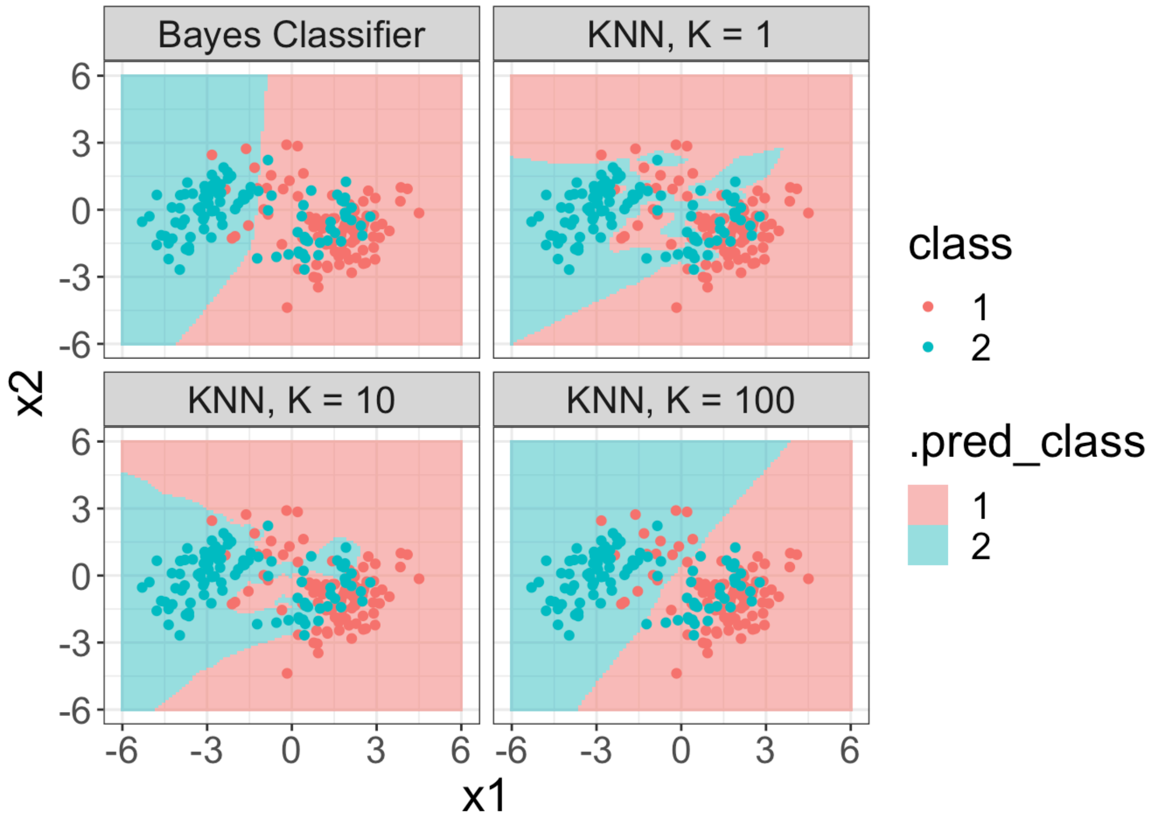
use MSE to quantify test error.

categorical response!

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

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

k-fold and validation errors estimated accordingly.



simulated example (recall).

Can we choose k using CV approach?

"tune" k.

```
k_fold <- 10
train_cv <- vfold_cv(train, v = k_fold)

grid_large <- tibble(neighbors = seq(1, 100, by = 2))

knn_spec <- nearest_neighbor(mode = "classification", neighbors =
          tune("neighbors"))
knn_spec |>
  tune_grid(class ~ x1 + x2, resamples = train_cv, grid = grid_large)
          |>
  collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(error = 1 - mean) -> knn_err
```

values of k I want to check 1,3,5,7,...., 99

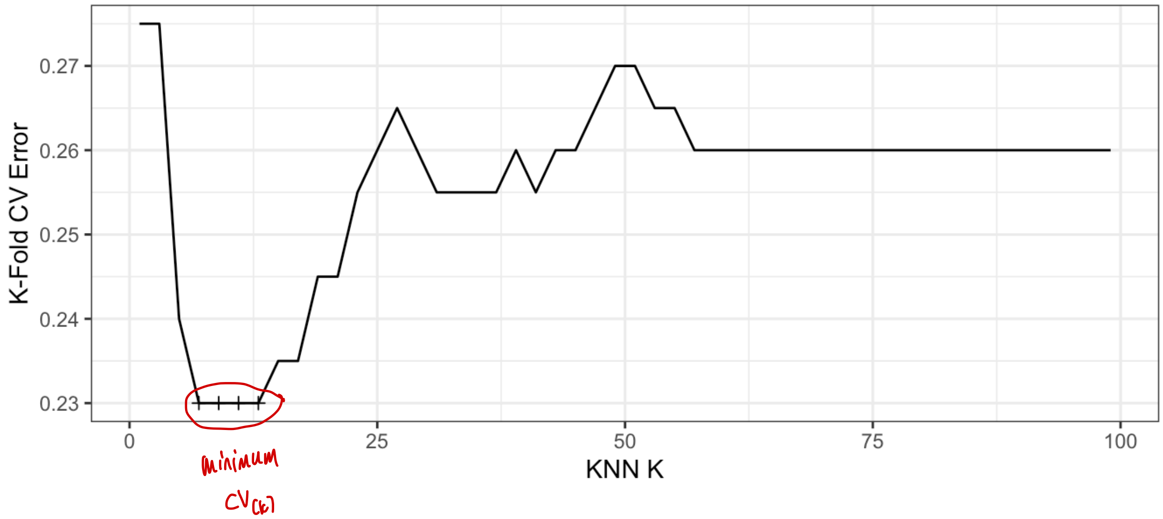
needs up! We're tuning on neighbors (k).

fit based on tuning grid.

formula for model.

how we split data

what values to tune on.



Minimum CV error of 0.23 found at $K = 7$.

use 7 nearest neighbers in KNN.

So we might choose k=7 and fit on entire training data set...