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 model the data came 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 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)*

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

Small if predictions are close to response.

y_i: response for i^{th} observation

$\hat{f}(x_i)$: prediction for i^{th} observation

Sometimes also talk about "root MSE"

RMSE = $\sqrt{MSE}$

(same scale as the response)

based on the training data (used to fit the 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 training data.

Suppose we fit our model to training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and obtain $\hat{f}$.

We can compute $\hat{f}(x_1), \ldots, \hat{f}(x_n)$ if these are close to $y_1, \ldots, y_n \Rightarrow$ 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

$$\text{Average}\left[\left(y_0 - \hat{f}(x_0)\right)^2\right]$$

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 obs. not used to fit the model.

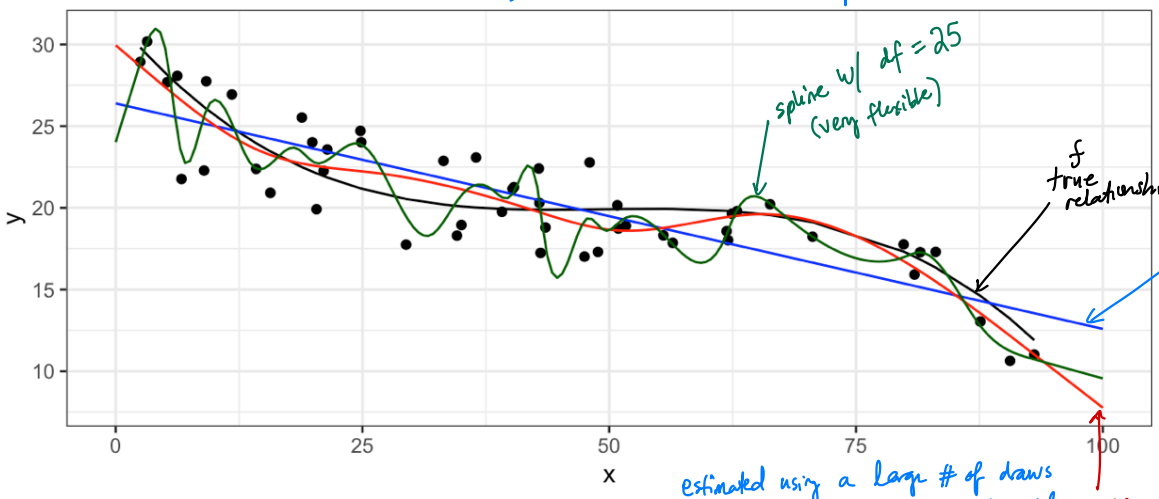
But what if we don't have a test set available?

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 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.

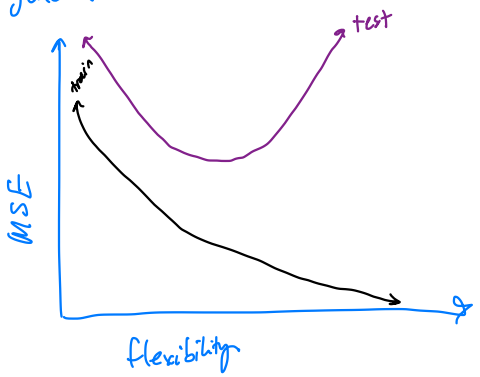
model	df	Test MSE	Train MSE
Linear Regression	2	34.4168	4.9654
Smoothing Spline	6	38.9525	3.5248
Smoothing Spline	25	39.9288	2.3107

least flex ↓ most flex

best training MSE ⇒ fits training data the best.

not the best test MSE.

In general



⇒ Need to estimate test MSE!

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.

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.

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

y_i: label for i^{th} observation; $\hat{y}_i$: predicted label for i^{th} observation

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)

The test error rate is

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

$\hat{y}_0$: 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

average test MSE we would obtain if we repeatedly est. f at many training data sets and predict x_0.

irreducible error

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

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/ different training data.

In general, more flexible methods have higher variance because they fit the training data so closely $\Rightarrow$ new data mean big changes 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 data are actually linear $\Rightarrow$ there will be some bias.

In general:

$\uparrow$ flexibility $\Rightarrow$ $\downarrow$ bias + $\uparrow$ variance.

how much these change determine test MSE.

Similar ideas hold for 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 rate.

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

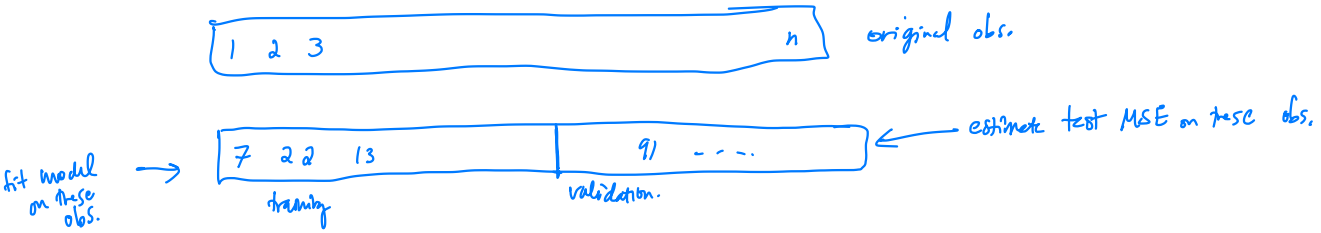
Split up data you already have (training data).

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

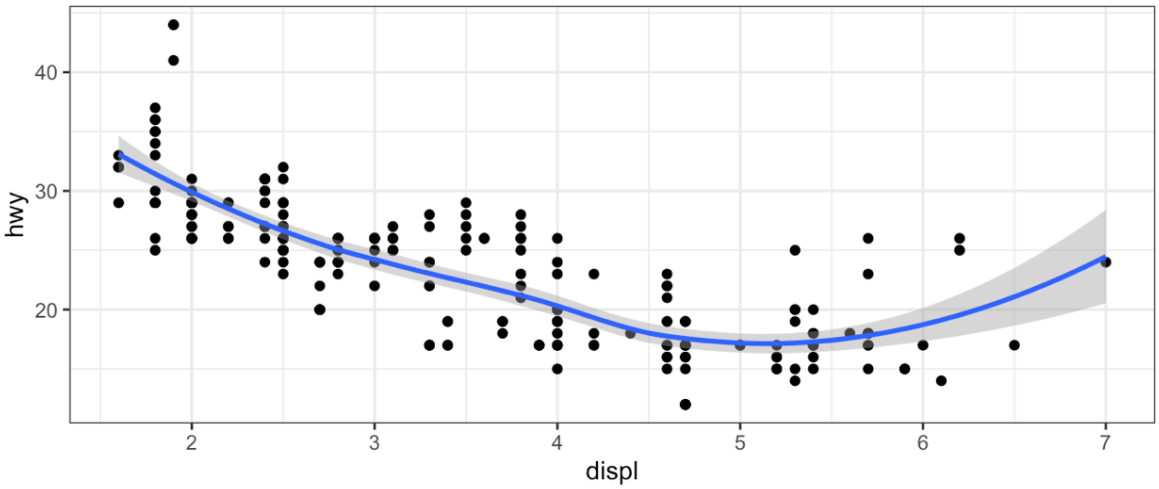
2.1 Validation Set

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.



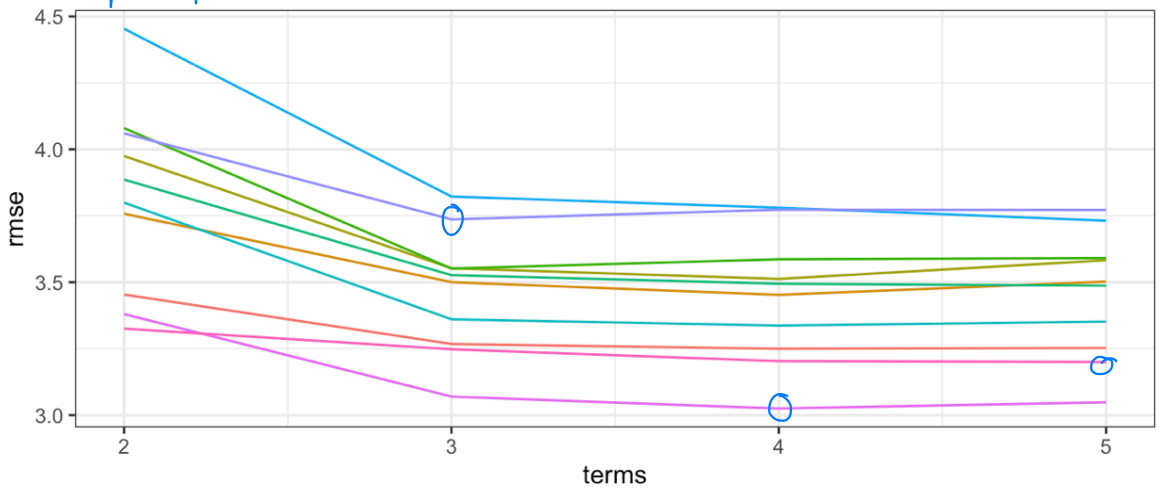
We fit the model with a squared term $\texttt{displ}^2$, but we might be wondering if we can get better predictive performance by including higher power terms!

$displ^3$, $displ^4$

```
## 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

Repeated process 10 times.



A lot of variability here!

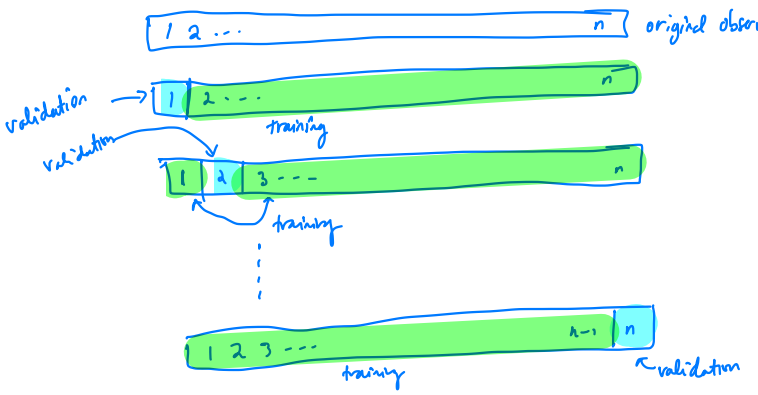
- The validation estimate of the test error is highly variable! Depends on which obs. we hold out.
- only a subset used to fit the model. Artificially reducing sample size.

⇒ 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 only use a single observation for validation



(1) fit model on $n-1$ observations

(2) $\hat{y}_i$ prediction for held out observation

$MSE_i = (y_i - \hat{y}_i)^2$

↖ unbiased for test error but highly variable!

The LOOCV estimate for the test MSE is

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

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

Advantages

- since we fit using $n-1$ observations (instead of $\approx \frac{n}{2}$ for the validation approach),
  ⇒ LOOCV does not overestimate the true test error as much as validation approach.
- No randomness in the approach. ⇒ will get the same answer every time.

Disadvantage

- sometimes stat learning models can be expensive to fit (i.e. on order of days)
  LOOCV requires us to fit the model n times.
  ⇒ could be slow.

we want n "folds"
↳ we want to fit this model n times.

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

new splitting of data

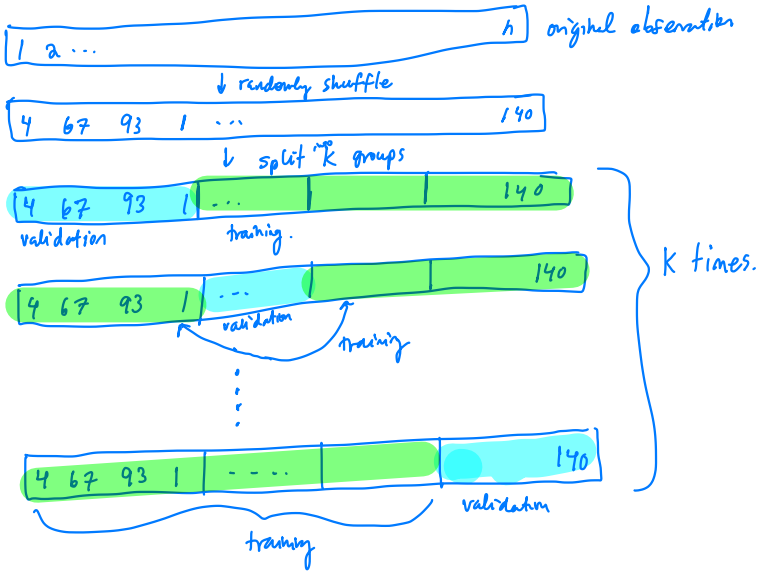
```
## models
m0 <- workflow() |> add_model(lm_spec) |> add_recipe(linear_recipe) |>
        fit_resamples(resamples = mpg_loocv)
m1 <- workflow() |> add_model(lm_spec) |> add_recipe(quad_recipe) |>
        fit_resamples(resamples = mpg_loocv)
m2 <- workflow() |> add_model(lm_spec) |> add_recipe(cubic_recipe)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

model	rmse
linear	2.808356
quadratic	2.675896
cubic	2.615363
quartic	2.643536

we choose the level of flexibility w/ lowest CV(n) estimate of test error.

2.3 k-Fold Cross Validation

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



The k-fold CV estimate is computed by averaging

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

estimate test MSE based on i^{th} fold.

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

Why k-fold over LOOCV?

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

Computational advantage! Now have to fit model k times (not n). → can be huge.

Another less obvious advantage due to bias-variance trade-off (come back to later).

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

## 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()
```

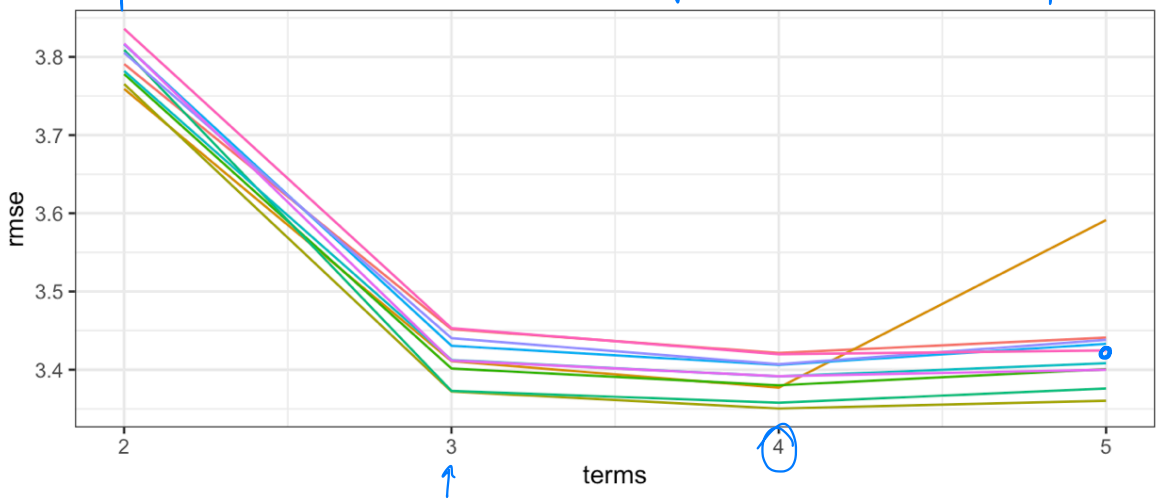
change number of folds! 10-fold CV.

model	rmse
linear	3.805566
quadratic	3.432052
cubic	3.409391
quartic	3.408420

close.

Repeat 10 times.

Now again there is some randomness in procedure.



some would choose quadratic model.

Similar outcomes.

When we perform CV we can be interested in estimating test error.

Most often we use it to find minimum estimated test error to help us choose a model (or model parameter)

↳ 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 advantage (but potentially more important).

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

We know the validation approach can overestimate the test error because we use only half (approx.) 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 level 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.

r.v.s X, Y
$\text{Var}\left(\frac{X+Y}{2}\right) = \frac{1}{4}\left[\text{Var}X + \text{Var}Y + 2\,\text{Cov}(X,Y)\right]$

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

Why?
LOOCV fit 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).

mean of highly correlated quantities has higher variance than mean of less correlated quantities!
$\Rightarrow$ LOOCV 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.

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.

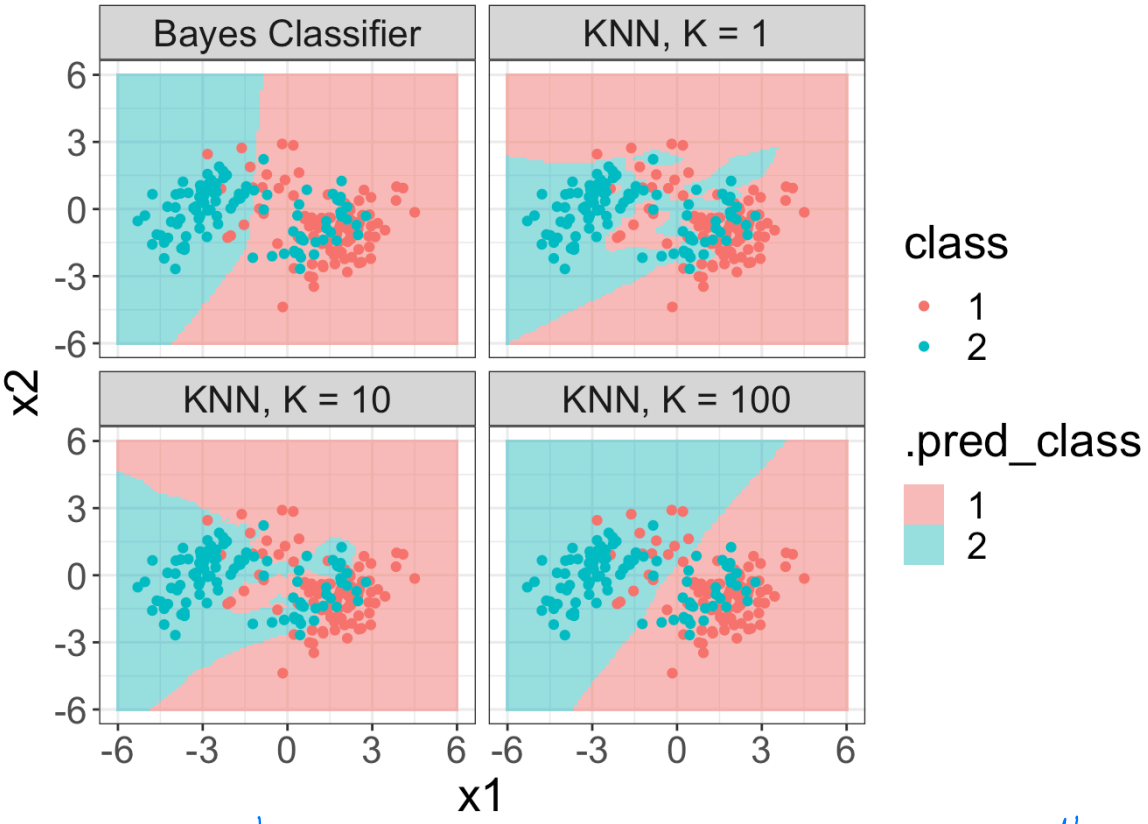
→ categorical response

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

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

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

K-fold and validation errors estimated accordingly.



recall this simulated example.

Can choose K using a CV approach!

↑

tune KNN!

split data for 10-fold CV

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

tuning data frame.

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

model specification for KNN

```
knn_spec <- nearest_neighbor(mode = "classification", neighbors =
        tune("neighbors"))
```

tune the parameter called neighbors

column name to check for values of k

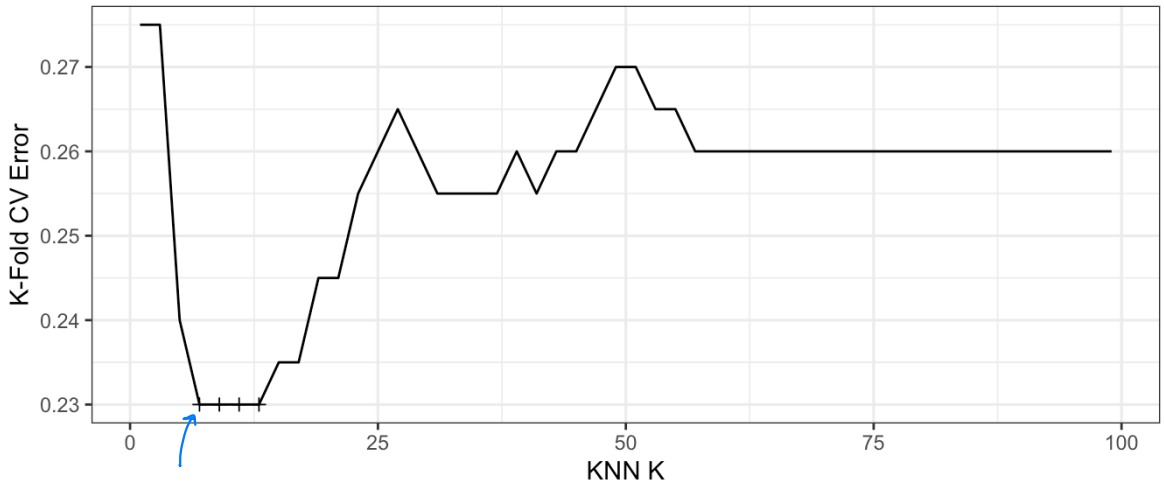
```
knn_spec |>
  tune_grid(class ~ x1 + x2, resamples = train_cv, grid = grid_large)
```

formula to pass to model.

10-fold CV

tune model on a grid of K's

```
  |>
  collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(error = 1 - mean) -> knn_err
```



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

So we might choose K=7. And fit KNN w/ K=7 on the entire training data set.