Bias/Variance and Cross-Validation

J.F. Omhover

slides of Ryan Henning

- Explain the bias/variance tradeoff

- Define overfitting and underfitting

- Relate flexibility and complexity of a model to bias / variance, to overfitting / underfitting, and to training / testing error

- Define and implement cross-validation on a dataset

Quick Review: Regression vs. Classification
(in machine learning)

What is regression?

Use features to predict <u>real valued</u> targets.   E.g. predict future sales/revenue

What is classification?

Use features to predict <u>categorical</u> targets.   E.g. predict   yes/no,   male/female,   0-9

One goal is to make accurate ***predictions*** on future (unseen) data.

1. Define a business goal.

    e.g. make Tesla cars the most dependable vehicles on the market

2. Collect training data.

    e.g. Tesla cars' event logs + historical record of parts replaced

3. Train a model.

    e.g. **features:** event statistics, **target:** time until failure

4. Deploy the model.

    e.g. monitor cars' events in real time, send mechanics to replace parts that will soon fail

Questions!

Review: Linear Regression

We assume the world is built on linear relationships. Under that assumption, we can model the relationship between *features* and a *target* like this:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

target
(aka, outcome)

model parameters
(aka, coefficients)

features
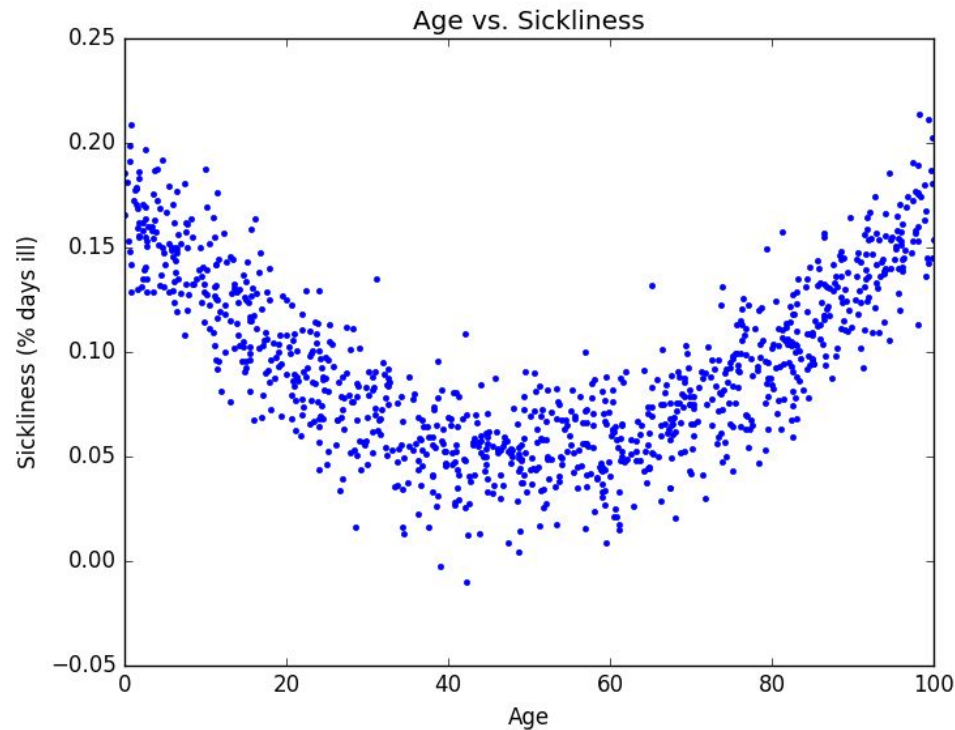(aka, predictors)

sampling error

We can make linear regression non-linear by inserting extra "interaction" features or higher-order features.

Age vs. Sickliness

Example:

$$Y = \beta_0 + \beta_1 * \text{age}$$

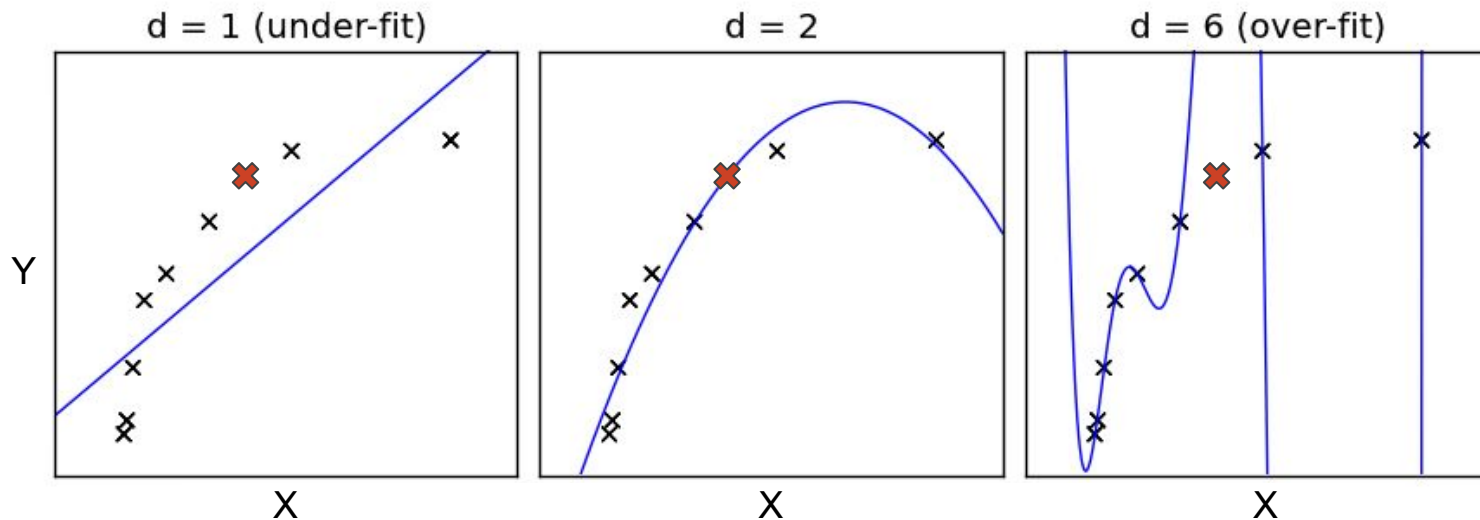$$Y = \beta_0 + \beta_1 * \text{age} + \beta_2 * \text{age}^2$$

We *could* just keep inserting interaction features until $R^2 = 1$.

Boom. I <u>solved</u> data science. Here's my idea:

```python
def train_super_awesome_perfect_model (X, y):
    while True:
        model = LinearRegression()
        model.fit(X, y)
        if calculate_r2(model, X, y)  >= 0.999:
            return model
        else:
            X = insert_random_interaction_feature(X)
```

Why is this a bad idea?

Oh the woes of overfitting…

What's bad about the <u>first</u> model?

What's bad about the <u>second</u> model?

What's bad about the <u>third</u> model?

Underfitting and Overfitting

Underfitting: The model doesn't fully capture the relationship between predictors and the target. The model has *not* learned the data's <u>signal</u>.

→ What should we do if our model underfits the data? (assume using lin. reg.)

Overfitting: The model has tried to capture the sampling error. The model has learned the data's signal *and* the <u>noise</u>.
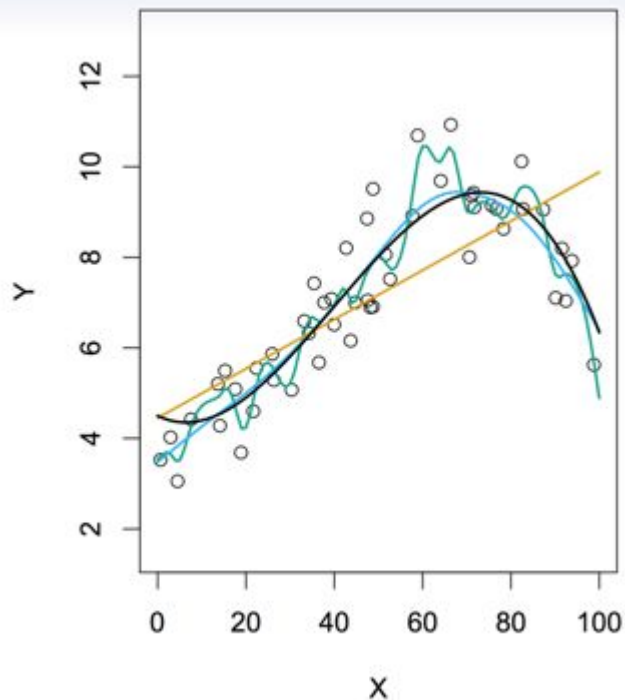
→ What should we do if our model overfits the data? (harder… any guesses?)

The Bias/Variance Tradeoff

Boardwork… build intuition...

Let's get an intuitive feel for the bias and the variance of a model… we'll see more math on the next slide.

Note: **Bias** and **Variance** are terms you will use A TON as a data scientist! Exciting times!

We assume the true predictor/target relationship is given by an unknown function plus some sampling error:

$$Y = f(X) + \epsilon$$

We estimate the true (unknown) function by fitting a model over the training set.

$$\hat{Y} = \hat{f}(X)$$

Let's evaluate this model using a test observation $(\mathbf{x_0}, \mathbf{y_0})$ drawn from the population. What is the model's expected squared prediction error on this test observation?

$$E[(y_o - \hat{f}(x_0))^2] = \ldots$$

Our model's expected squared prediction error will depend on (1) the variability of y_0 and (2) the variability of the training set used to train our model. We can break this into three pieces:

$$E[(y_o - \hat{f}(x_0))^2] = ... = \mathrm{Var}(\hat{f}(x_0)) + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\epsilon)$$

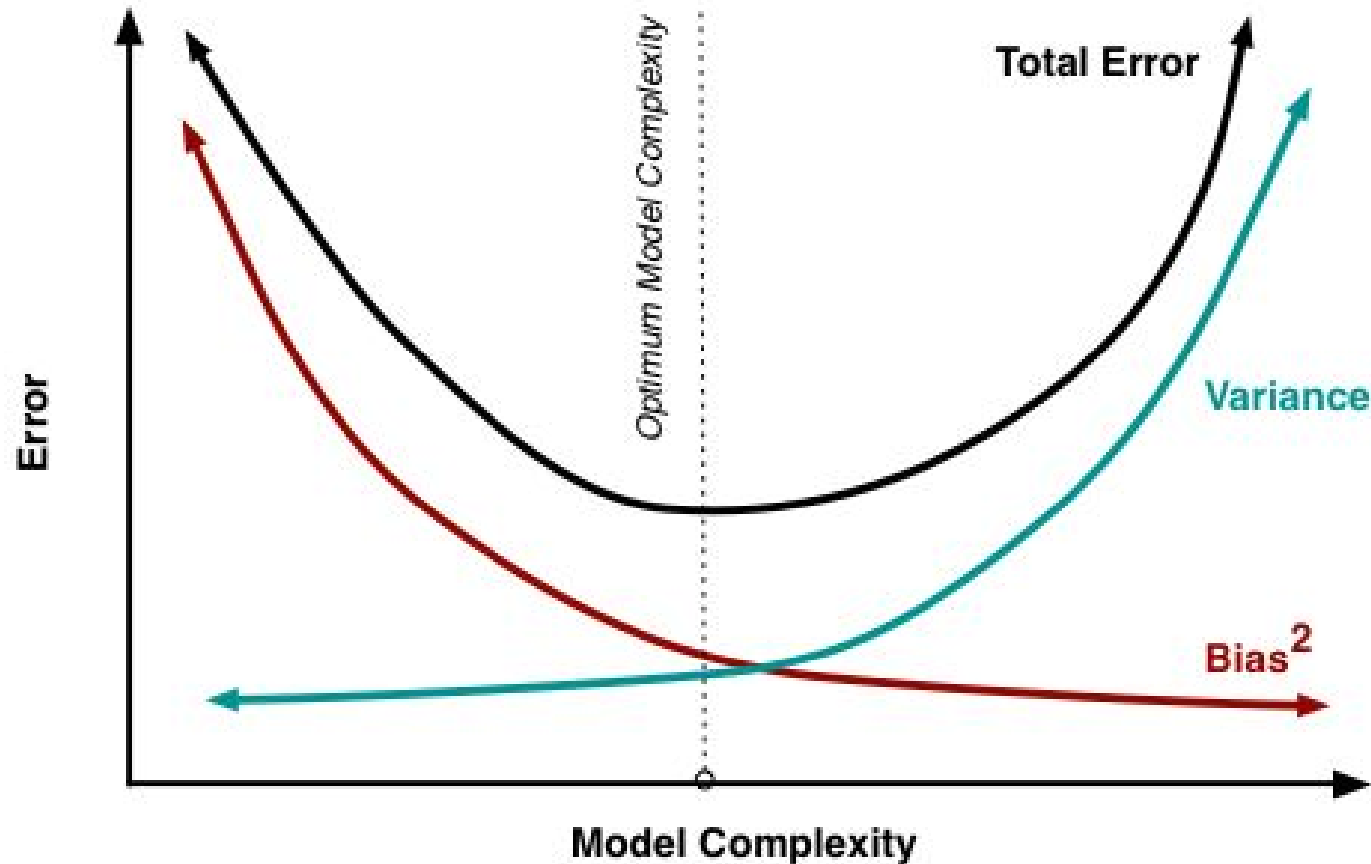The variance of our model's prediction of x_0 over all possible training sets

The difference between the true target and our model's average prediction over all possible training sets

The variance of the irreducible error.

$$\mathrm{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$$

How is the **bias/variance tradeoff** related to **underfitting** and **overfitting**?

How can we find the best tradeoff point? I.e. The optimum model complexity

QUESTION TIME !

1. What are the ways we can increase complexity of a model?

2. Why does variance increase and bias decrease
   as model complexity increases?

3. How do you determine the optimal complexity to use?

Cross-Validation

Main idea: **Don't use all your data for training.**

Instead: **Split your data into a "training set" and a "validation set".**

randomize!
(why?)

training set

where to split?
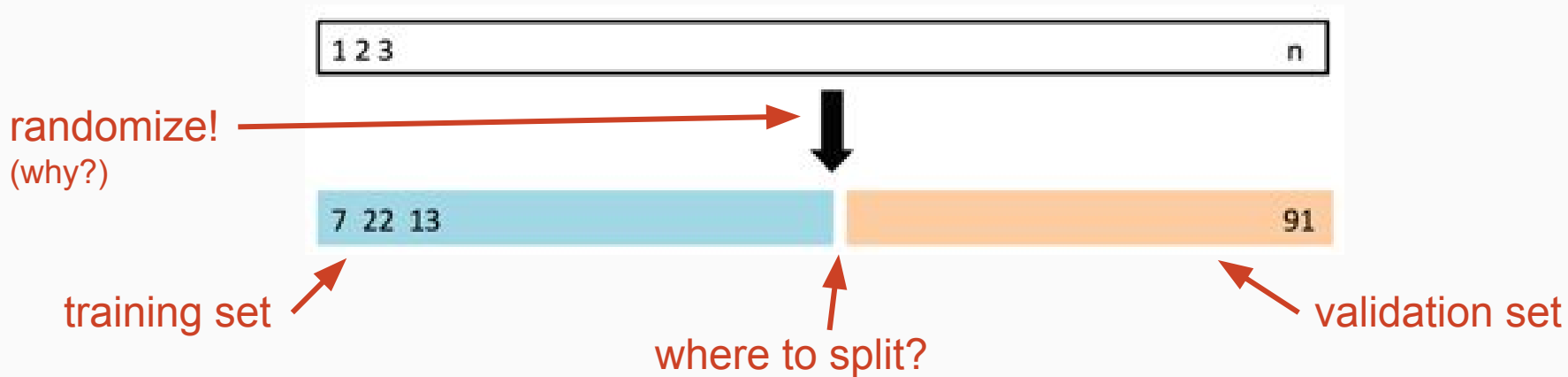
validation set

Cross-Validation

1. Split your data into training/validation sets.
   > 70/30 or 90/10 splits are commonly used

2. Use the training set to train several models of varying complexity.
   > e.g. linear regression (w/ and w/out interaction features), neural nets, decision trees, etc.
   >
   > (we'll talk about hyperparameter tuning, grid search, and feature engineering later)

3. Evaluate each model using the validation set.
   > calculate R^2, MSE, accuracy, or whatever you think is best

4. Keep the model that performs best over the **validation** set.

blue: training set

red: validation set

Cross-Validation Example
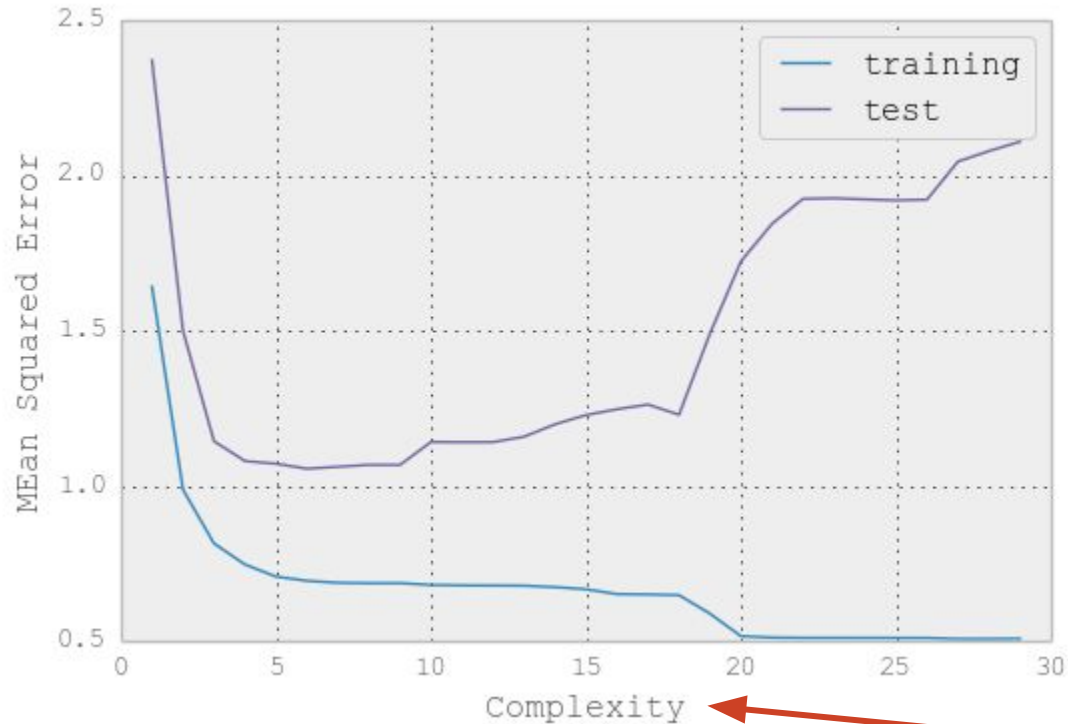
You will see this shape all the time!

You will wrestle with the bias/variance tradeoff constantly...

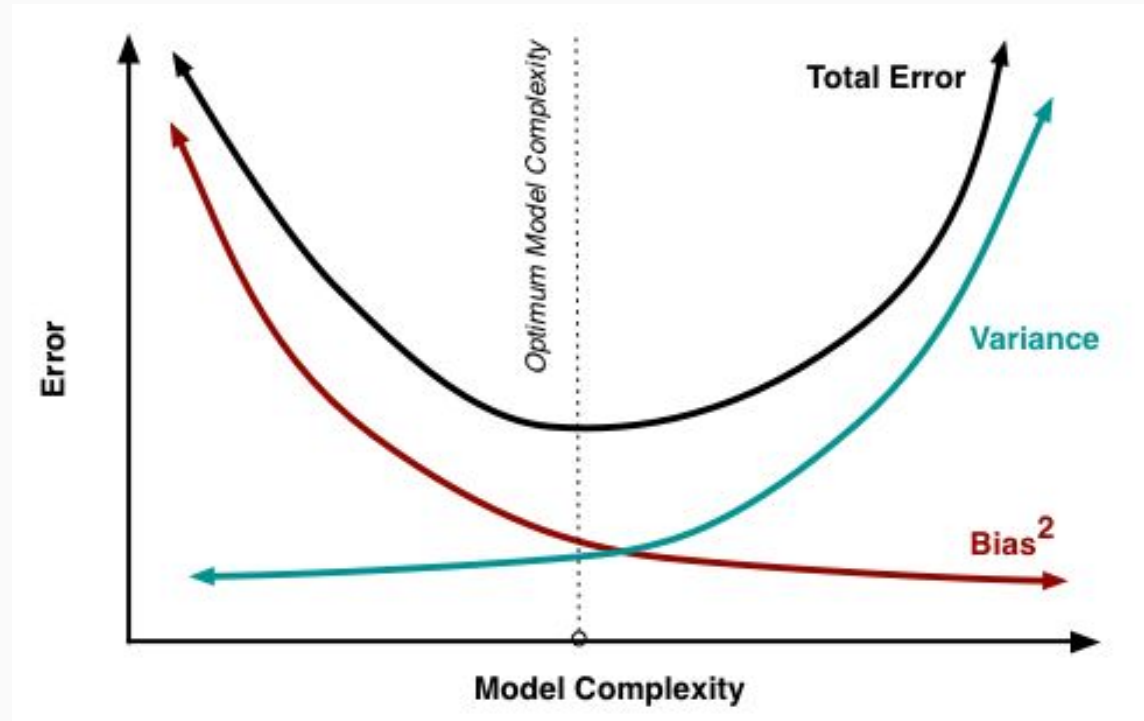E.g. linear regression w/ varying degree of polynomial

Fitting the training set perfectly is *easy*.
How?

Fitting future (unseen) data is *not easy*.

Cross validation helps us choose a model that performs well on unseen data.

Commonly, k=5 or k=10

randomize!

1. Split the dataset into k "folds".
2. Train using (k-1) folds. Validate using the one "leave out" fold. Record a validation metric such as RSS or accuracy.
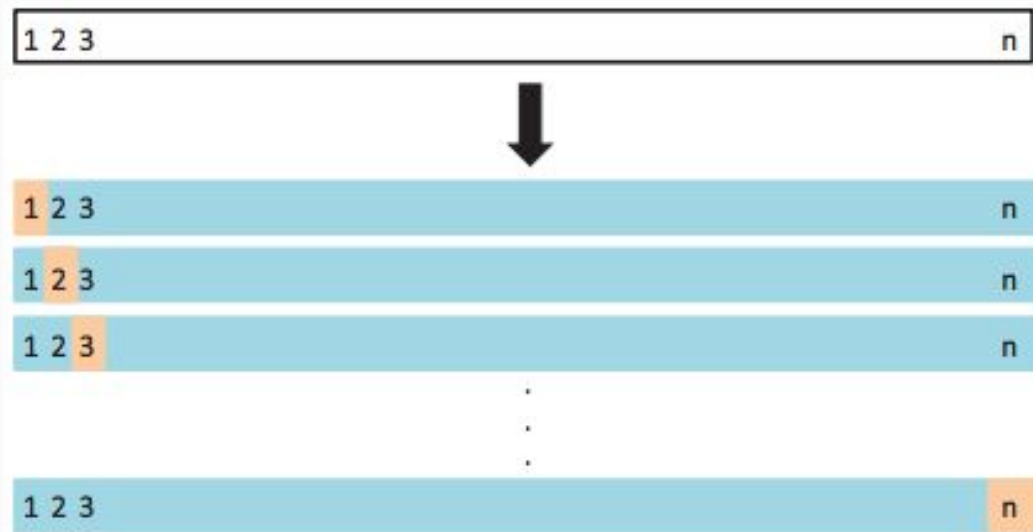3. Train *k* models, leaving out a different fold for each one.
4. Average the validation results.
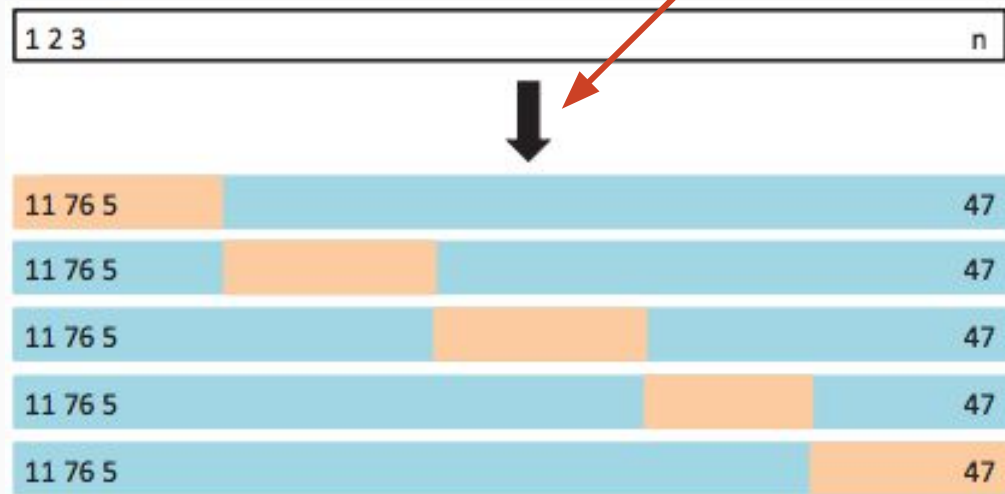
Assume we have *n* training examples.

A special case of k-fold CV is when k=n. This is called *leave-one-out cross-validation*.

Useful (only) if you have a tiny dataset where you can't afford a large validation set.

If our data has high dimensionality (many many predictors), then it becomes easy to overfit the data.

This is one result of the so-called **Curse of Dimensionality** (look it up).

Even linear regression might be too complex of a model for high dimensional data (and the smaller the dataset, the worse this problem is).
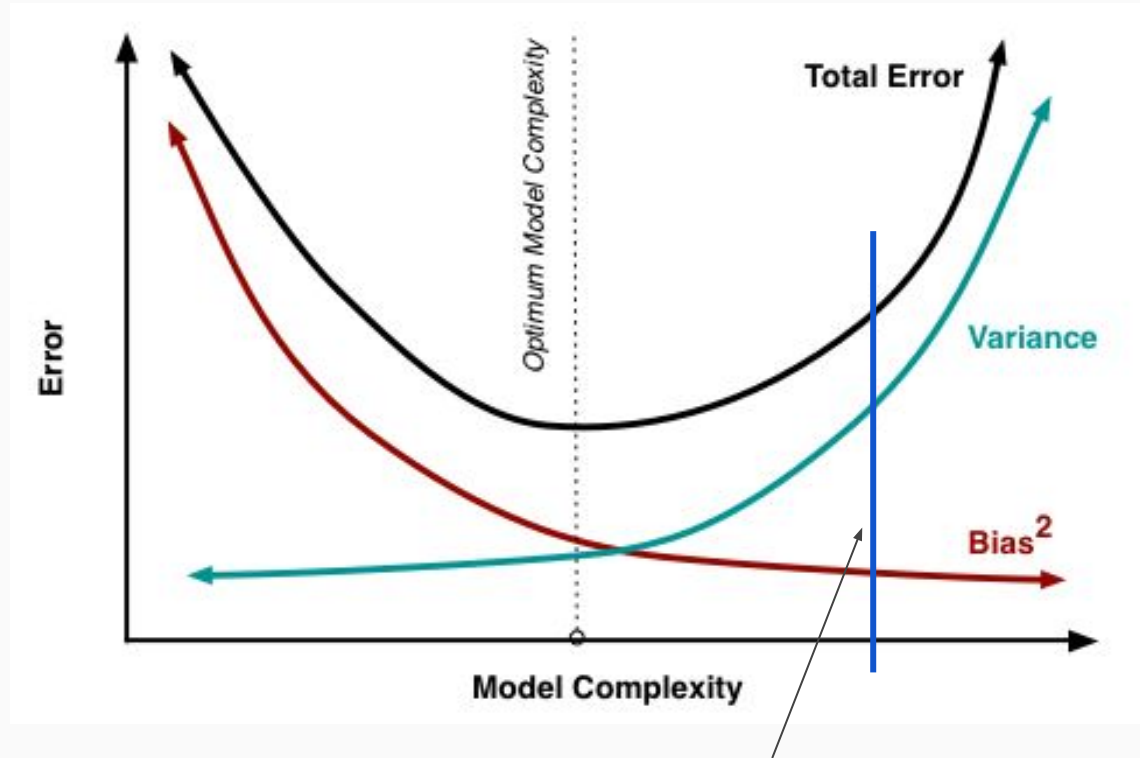
Linear regression in high dimensions

QUESTION TIME !

1. Why do we often prefer to use K-Fold Cross Validation instead of a simple train test split?

2. How many models are created in a 5-fold cross validation to compare two models (model A and model B)?

3. What do we do with all the models we created and how do we determine what single model to keep?

You have a few options.

1. Get more data… (not usually possible/practical)

2. **Subset Selection:** keep only a subset of your predictors (i.e, dimensions)

3. **Regularization:** restrict your model's parameter space

4. **Dimensionality Reduction:** project the data into a lower dimensional space

Subset Selection

Best subset: Try every model. Every possible combination of *p* predictors

- Computationally intensive. 2^p possible subsets of *p* predictors
- High chance of finding a "good" model by random chance.
  ... A sort-of monkeys-Shakespeare situation ...

Stepwise: Iteratively pick predictors to be in/out of the final model.

- Forward, backward, forward-backward strategies

$M_0 \rightarrow \quad M_1 \quad \rightarrow \quad M_2 \quad \rightarrow \quad .... \rightarrow M_p$

+1 predictor with smallest RSS ⇔ largest R^2

+1 predictor with smallest RSS ⇔ largest R^2

Now we have *p* candidate models

Are RSS and R^2 good ways to decide amongst the resulting *(p+1)* candidates?

Answer: Don't use RSS or R^2 for this part.
Use Mallow's C_p, or AIC, or BIC, or Adjusted R^2.

… or just use cross-validation with any error measurement.

$$C_p = \frac{1}{n}(RSS + 2p\hat{\sigma}^2)$$

Mallow's C_p
p is the total # of parameters
$\hat{\sigma}^2$ is an estimate of the variance of the error, ε

$$AIC = -2logL + 2 \cdot p$$

L is the maximized value of the likelihood function for the model estimated

$$BIC = \frac{1}{n}(RSS + log(n)p\hat{\sigma}^2)$$

This is Cp, except 2 is replaced by log(n). log(n) > 2 for n>7, so BIC generally exacts a heavier penalty for more variables

$$Adjusted\ R^2 = 1 - \frac{RSS/(n-p-1)}{TSS/(n-1)}$$

Similar to R^2, but pays price for more variables

Side Note: Can show AIC and Mallow's Cp are equivalent for linear case

You have a few options.

1. Get more data… (not usually possible/practical)

2. **Subset Selection:** keep only a subset of your predictors (i.e, dimensions)

3. **Regularization:** restrict your model's parameter space

4. **Dimensionality Reduction:** project the data into a lower dimensional space

TO BE CONTINUED...