

Lecture 2: Prediction

Part I: Splines, Additive Models

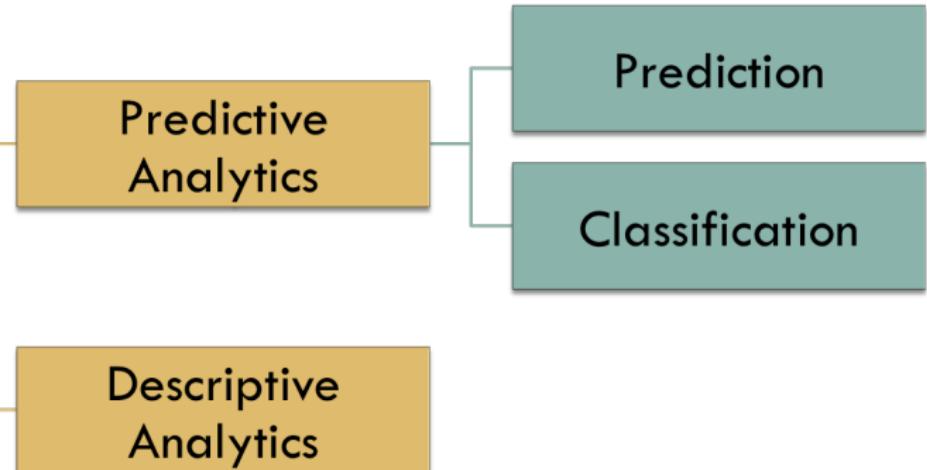
Part II: Model Selection and Validation

Prof. Alexandra Chouldechova

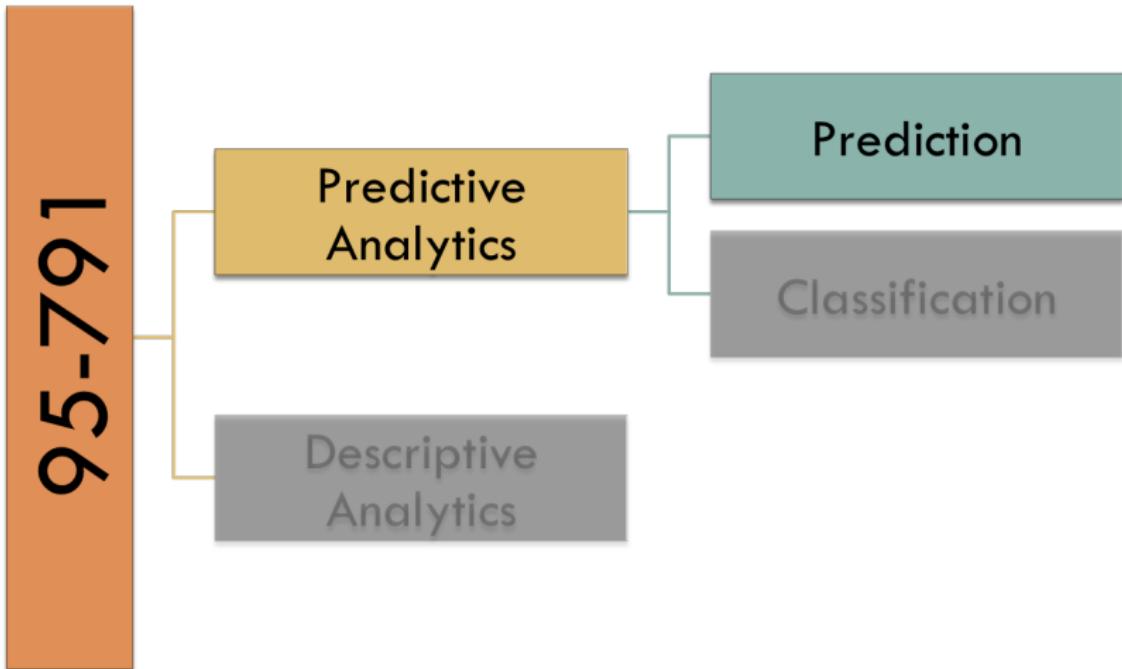
95-791: Data Mining

Course Roadmap

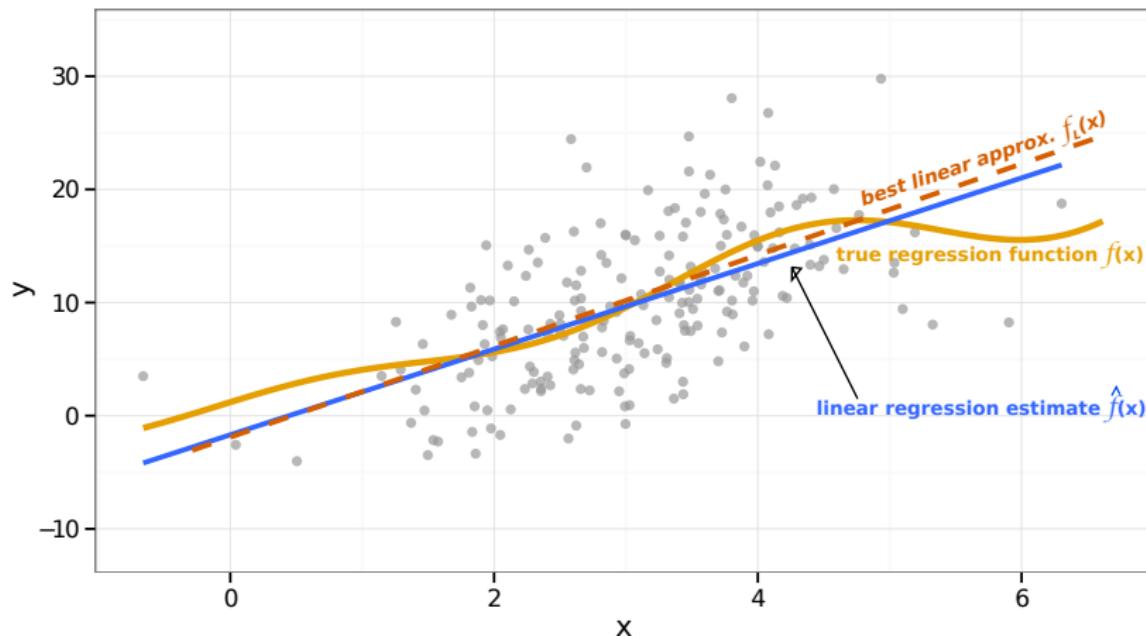
95-791



Course Roadmap



Recap of last class



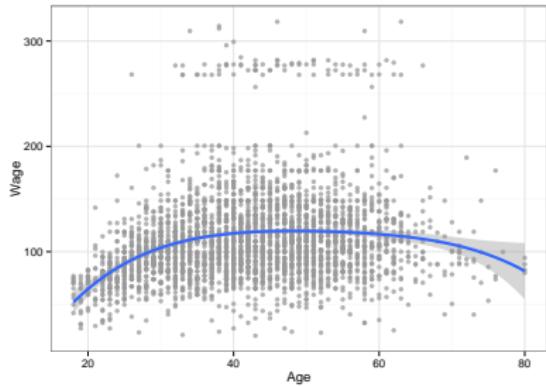
- The goal of prediction is to estimate the **true, unknown regression function**, f

Recap of last class

- Linear regression imposes **two key restrictions** on the model: We assume the relationship between the response Y and the predictors X_1, \dots, X_p is:
 - ➊ Linear
 - ➋ Additive
- The truth is almost never linear; but often the linearity and additivity assumptions are *good enough*
- When we think **linearity** might not hold, we can try...
 - Polynomials
 - Step functions
 - Splines
 - Local regression
 - Generalized additive models
- When we think the **additivity** assumption doesn't hold, we can incorporate **interaction terms**
- These variants offer increased **flexibility**, while retaining much of the ease and **interpretability** of ordinary linear regression

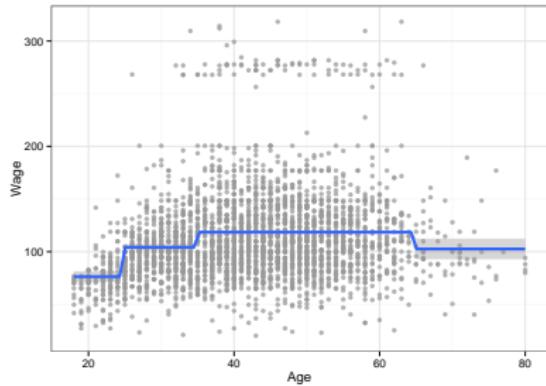
Recap of last class

Polynomials



```
lm(wage ~ poly(age, 4), data = Wage)
```

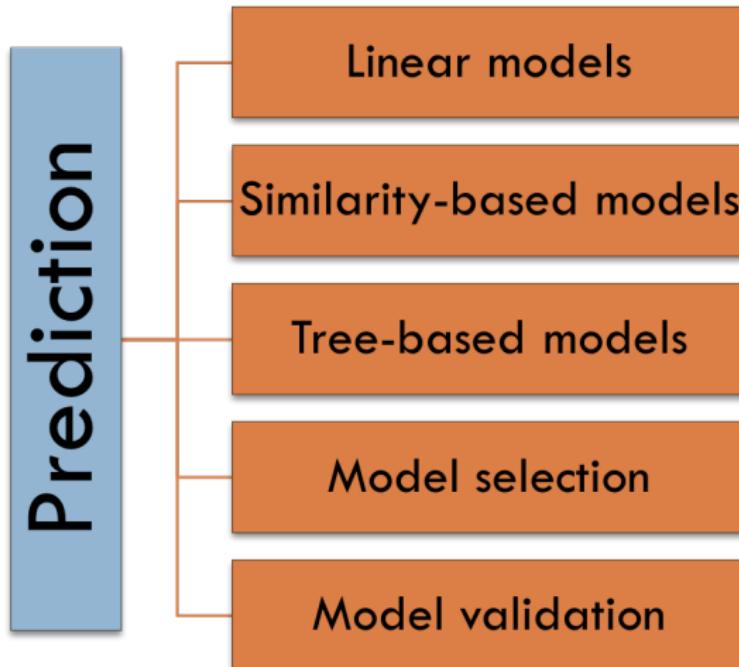
Step Functions



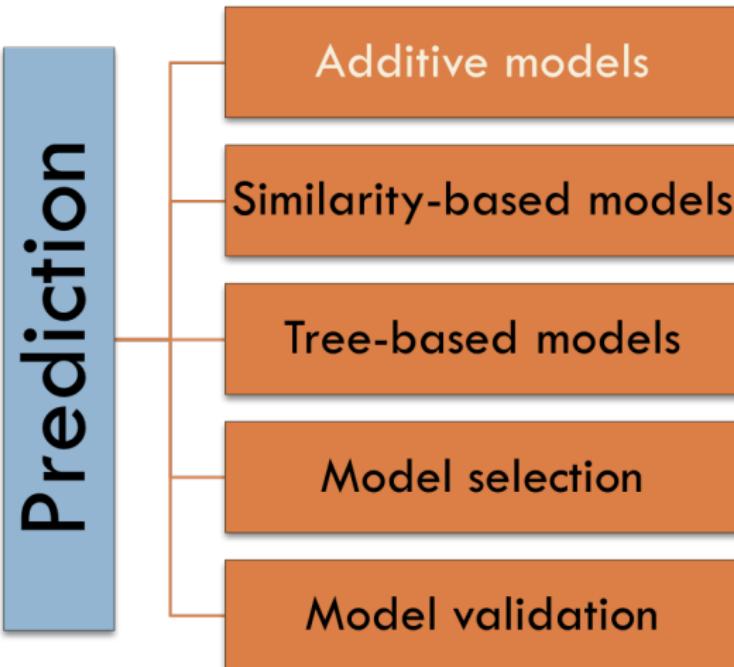
```
lm(wage ~ cut(age,  
breaks = c(-Inf, 25, 35, 65, Inf)),  
data = Wage)
```

- We can think of **polynomials** and **step functions** as simple forms of **feature engineering**
- These extensions of linear regression enable us to fit much more **flexible** models

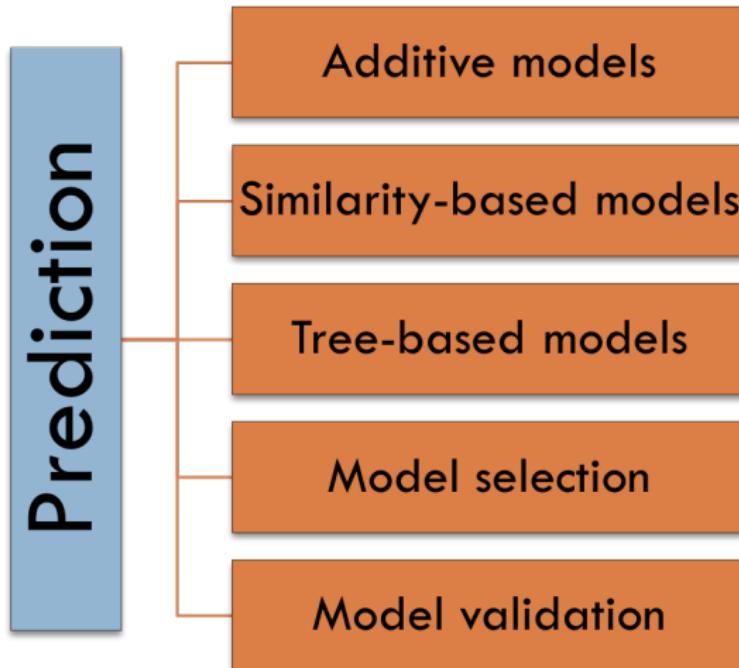
Prediction topics: Part I



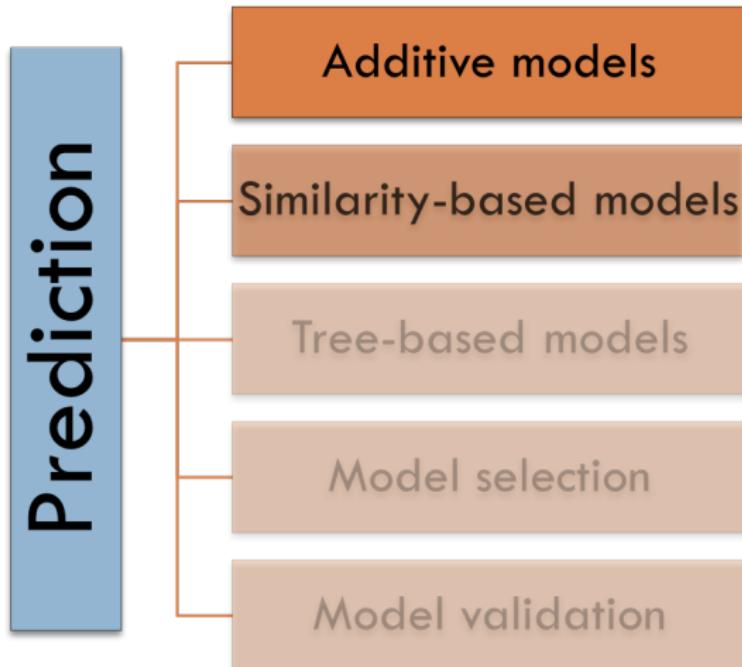
Prediction topics: Part I



Prediction topics: Part I



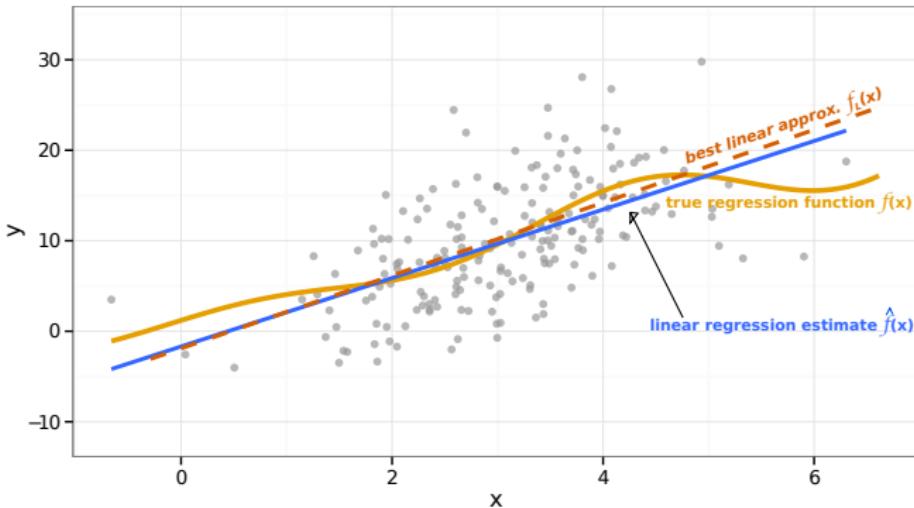
Prediction topics: Part I



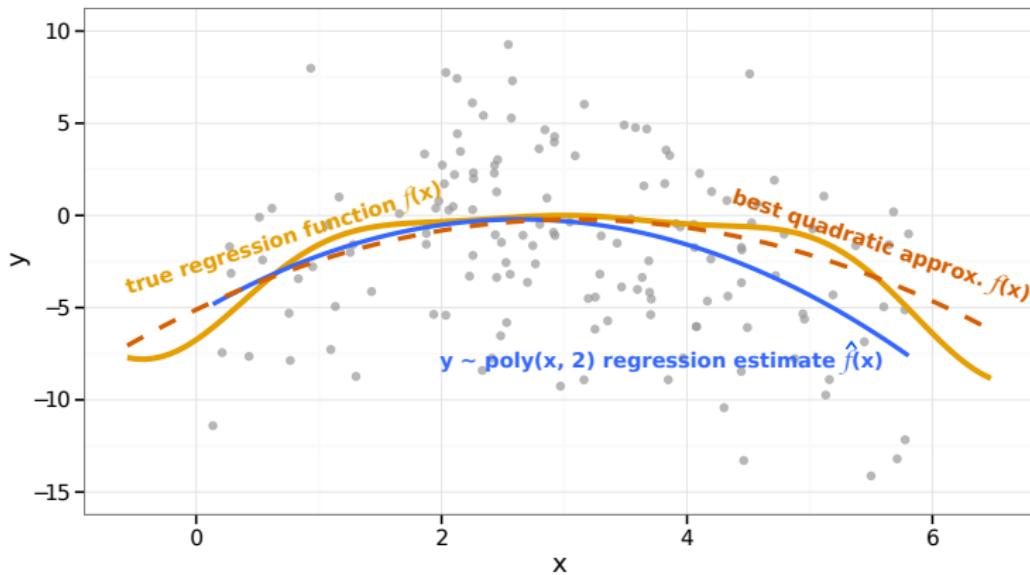
Agenda for Part I

- Piecewise polynomial fits
- Splines
- Additive models

Some motivation



- If the true regression function f is non-linear, ordinary linear regression cannot estimate it consistently
- No matter how much data you get, the best ordinary linear regression can do is converge to the best linear approximation to f



- The same problem persists with polynomial regression
- No matter how much data you get, the best degree- k polynomial regression can do is converge to the best degree- k approximation to f

Consistency

Consistent estimator

An estimator $\hat{f}(x)$ is **consistent** if, as our sample size grows, $\hat{f}(x)$ converges to the true regression function $f(x) = \mathbb{E}(Y | X = x)$

- Unless $f(x)$ is a polynomial of degree $\leq k$, **degree- k polynomial regression** is **not consistent**

Q: Can we get a **consistent** estimator of a generic regression function f ?

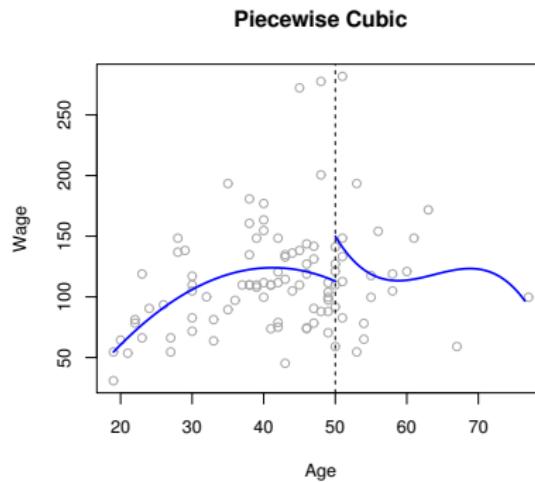
A: If we're willing to assume f is *smooth*, then **YES**

Use **splines!**

Piecewise polynomials

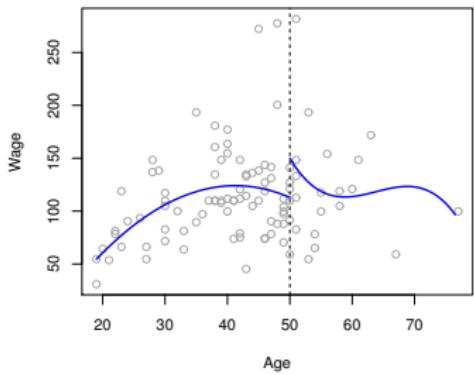
To understand splines, we first need to understand piecewise polynomials

- We can think of step functions as piecewise constant models
- It's easy to generalize this idea to:
 - Piecewise linear
 - Piecewise quadratic ...
 - Piecewise polynomial

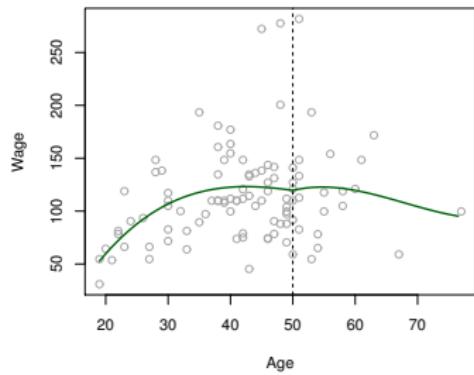


Piecewise Polynomial vs. Regression Splines

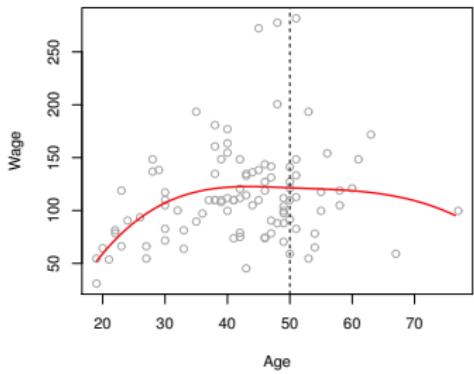
Piecewise Cubic



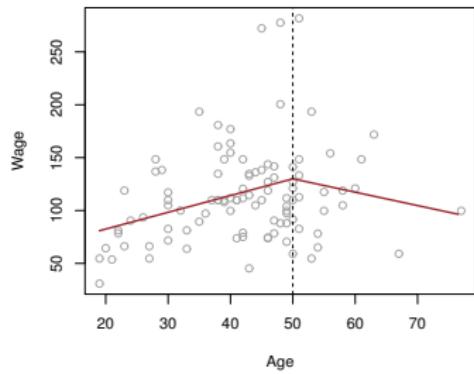
Continuous Piecewise Cubic



Cubic Spline

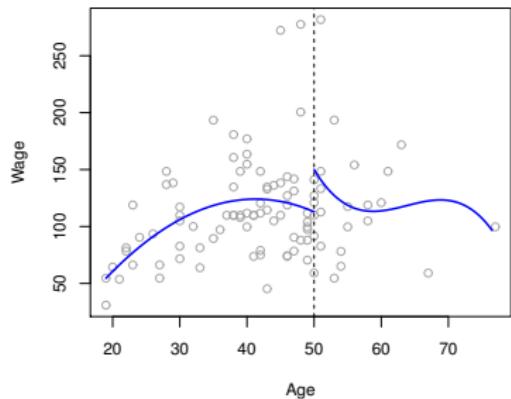


Linear Spline

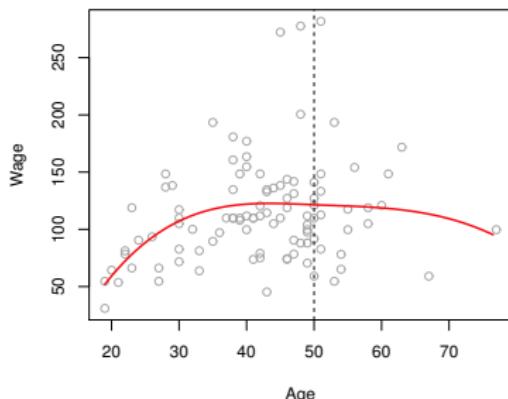


Piecewise polynomial vs. Regression splines

Piecewise Cubic



Cubic Spline



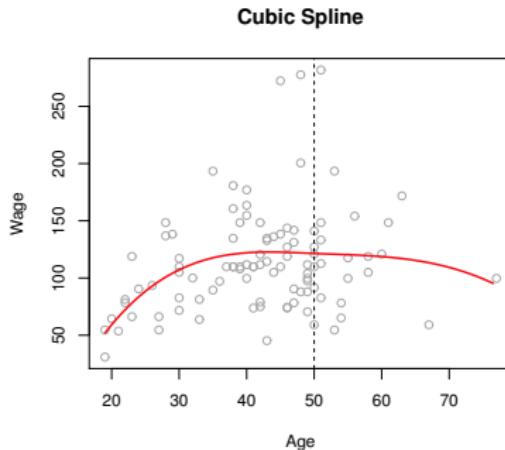
1 break at Age = 50

1 knot at Age = 50

Definition: Cubic spline

A **cubic spline** with **knots** at x -values ξ_1, \dots, ξ_K is a **continuous piecewise cubic polynomial** with *continuous derivates and continuous second derivatives* at each knot.

Cubic Splines



- Turns out, **cubic splines** are sufficiently **flexible** to *consistently estimate* smooth regression functions f
- You can use higher-degree splines, but *there's no need to*
- To fit a cubic spline, we just need to pick the **knots**

Polynomial regression vs. Cubic splines

- In polynomial regression, we had to choose the degree
- For cubic splines, we need to choose the knots
- Q: How complex is a cubic spline with K knots?
- Paraphrasing... A cubic spline with K knots is as complex as a polynomial of degree ____?
 - Turns out, there exist functions $b_k(x)$ ¹ such that a cubic spline with K knots can be modeled as

$$y = \beta_0 + \beta_1 b_1(x) + \beta_2 b_2(x) + \cdots \beta_{K+3} b_{K+3}(x) + \epsilon$$

- So... A cubic spline with K knots is as complex as a polynomial of degree $K + 3$.

¹See ISLR pg. 273 for details

Degrees of freedom

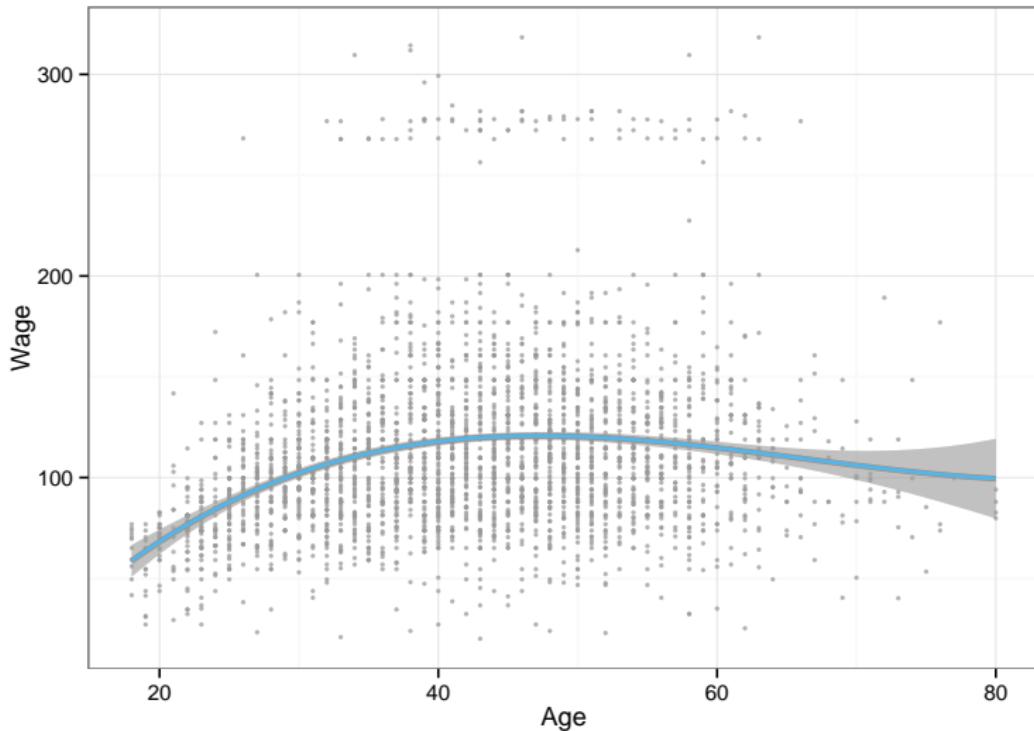
- Degrees of freedom capture the complexity of a regression model
- A linear regression model with p independent predictors is said to have p degrees of freedom²
- Take-away from the previous slide:

Model	# knots	Degrees of freedom
Regression, p predictors		p
Degree- k polynomial regression		k
Cubic spline	k	$k + 3$
Degree- d Spline	k	$k + d$

- In the following slides, we compare cubic splines to polynomial regression, allowing each method the same degrees of freedom

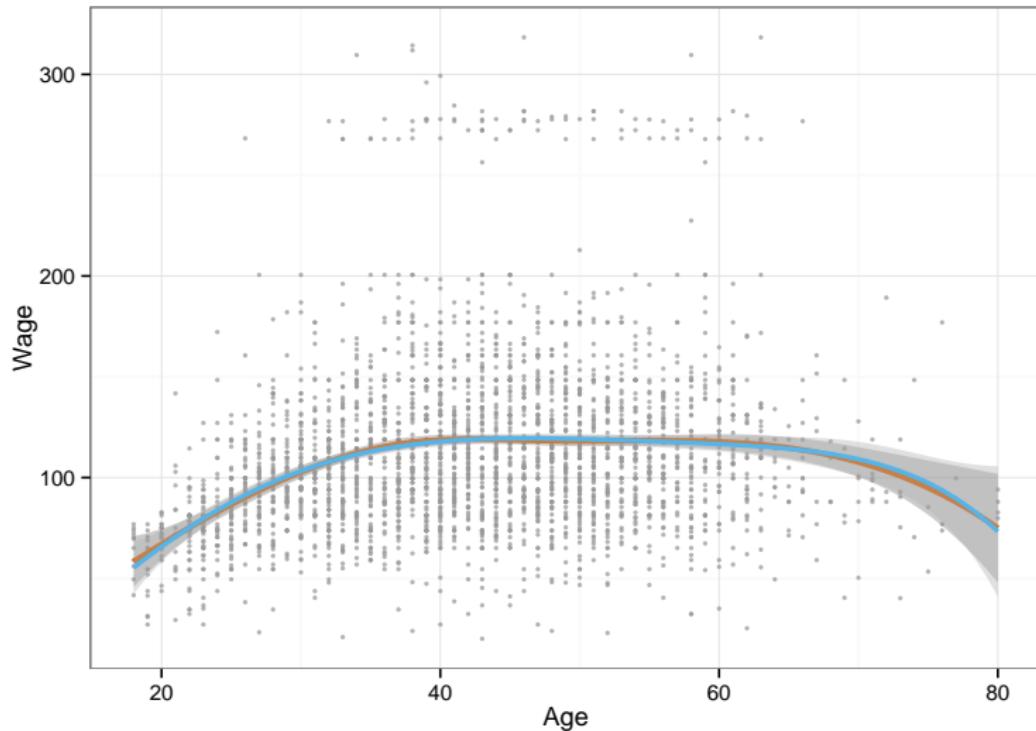
²Technically, $p + 1$ if you count the intercept. To be consistent with R's `df` arguments, here we do not count the intercept.

Polynomial regression vs. Cubic splines



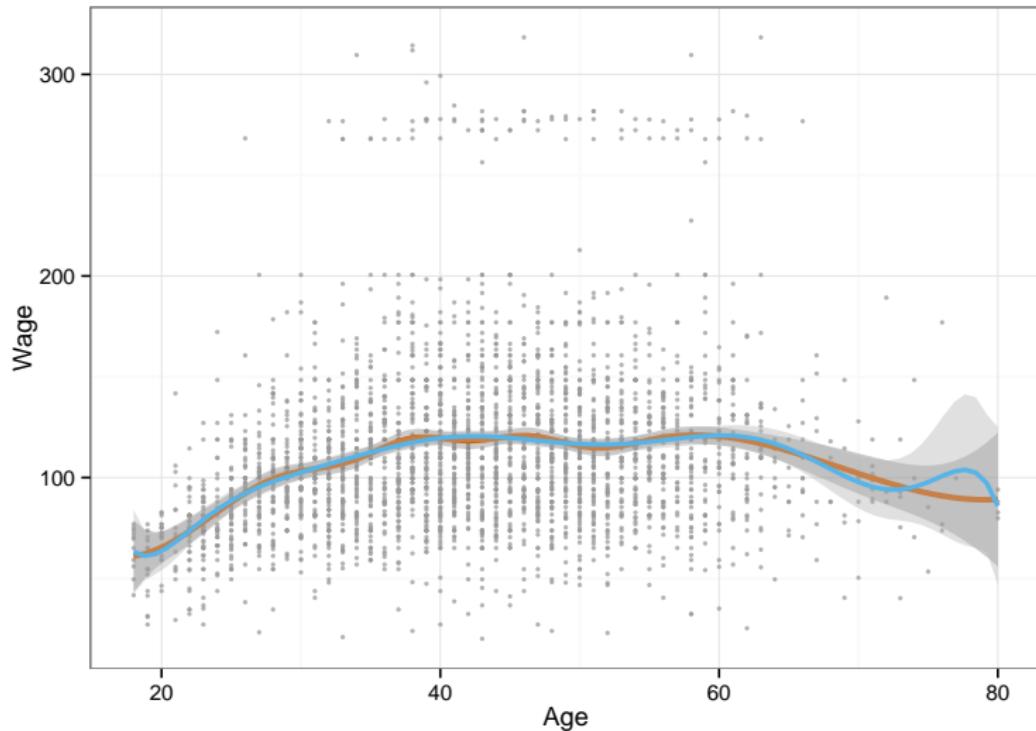
```
lm(wage ~ bs(age, df = 3))  
lm(wage ~ poly(age, degree = 3))
```

Polynomial regression vs. Cubic splines



```
lm(wage ~ bs(age, df = 5))  
lm(wage ~ poly(age, degree = 5))
```

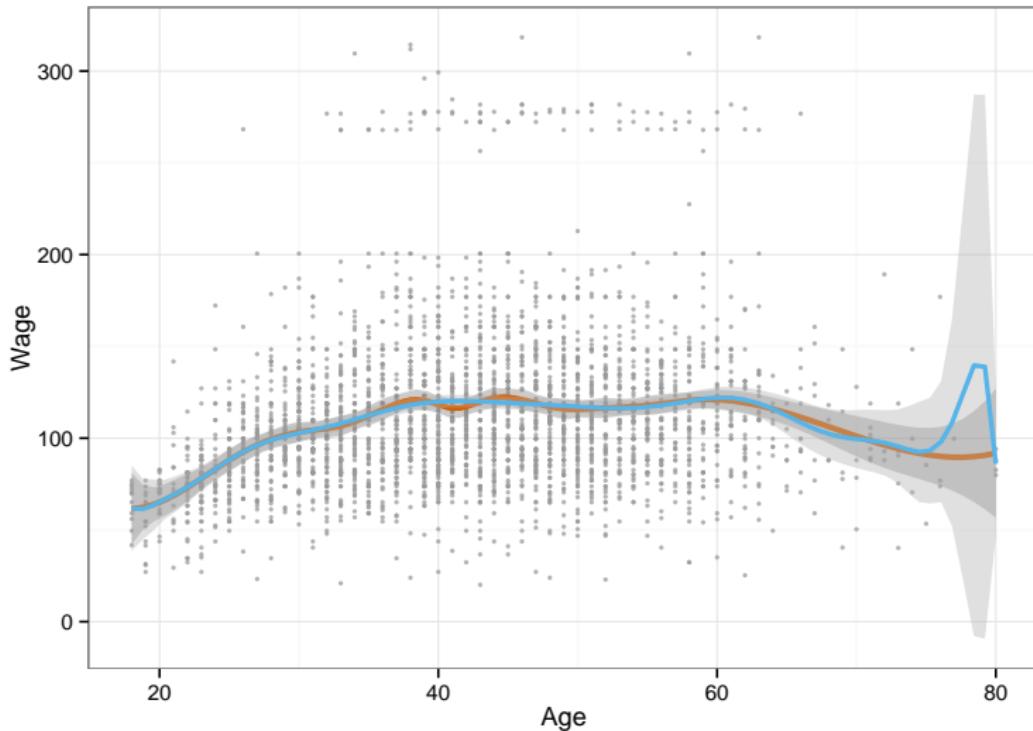
Polynomial regression vs. Cubic splines



`lm(wage ~ bs(age, df = 10))`

`lm(wage ~ poly(age, degree = 10))`

Polynomial regression vs. Cubic splines



`lm(wage ~ bs(age, df = 15))`

`lm(wage ~ poly(age, degree = 15))`

Natural Cubic Splines

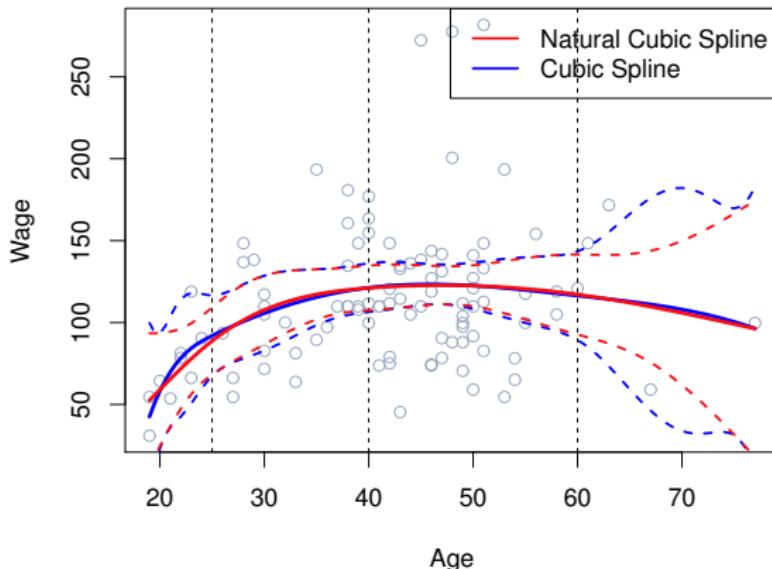


Figure: 7.4 from ISLR. Natural cubic splines are cubic splines that extrapolate linearly beyond the boundary knots. A NCS with K knots uses just $K + 1$ degrees of freedom — the same as a cubic spline with 2 fewer knots

Natural Cubic Splines vs Polynomial regression

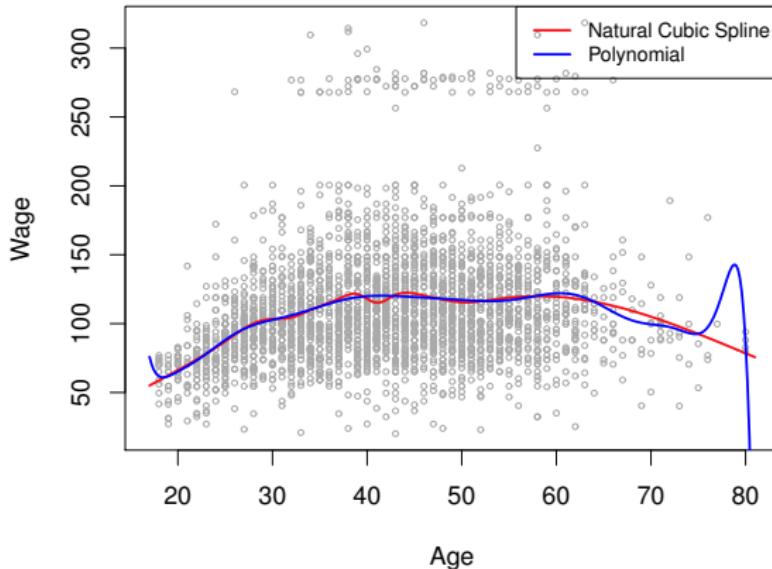
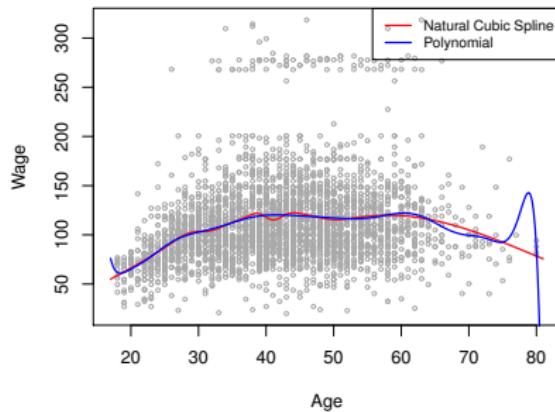


Figure: 7.7 from ISLR. **Natural cubic splines** are very nicely behaved at the tails of the data. **Polynomial regression** shows erratic behaviour. (14 degrees of freedom used for both)

Summary: Cubic Splines vs Polynomial regression



- **Polynomial regression** must use a **high degree** in order to produce **flexible fits**
- With **splines**, we can keep the degree fixed, and increase flexibility by **adding knots**
- Splines generally tend to be **better behaved** at the same level of *model complexity*

Knot placement: Rules of thumb

- Place more knots where f appears to be changing rapidly
- Place fewer knots where f appears to be slowly varying

R's defaults:

The most common way of specifying splines in R is in terms of the spline's degrees of freedom (df). R then places knots at suitably chosen quantiles of the x variable.

Model	R command	# internal knots
Cubic Spline	$\sim \text{bs}(x, \text{df})$	$\text{df} - 3$
Natural Cubic Spline	$\sim \text{ns}(x, \text{df})$	$\text{df} - 1$
Degree- d Spline	$\sim \text{bs}(x, \text{df}, \text{degree} = d)$	$\text{df} - d$

- You may also specify the internal knots *manually* for `ns` and `bs` by specifying the `knots =` argument directly. E.g.,
`lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)`

Smoothing splines

- **Q:** What is the best way to *automatically* place knots?
- *Paraphrasing...* If I know how many **degrees of freedom** I want my spline to have, is there a method for automatically choosing the **best locations** for the knots?
- **A:** Smoothing splines

Smoothing splines

- The smoothing spline estimator is the solution \hat{g} to the problem

$$\text{minimize } \underbrace{\sum_{i=1}^n (y_i - g(x_i))^2}_{\text{RSS}} + \lambda \underbrace{\int g''(t)^2 dt}_{\text{Roughness penalty}}$$

- This is a penalized regression problem³
- We're saying we want a function that:
 - ① Fits the data well; and
 - ② isn't too wiggly
- Large $\lambda \implies \hat{g}$ will have low variability (& higher bias)
- Small $\lambda \implies \hat{g}$ will have high variability (& lower bias)

How is this *at all* related to splines?

³We'll see more examples of penalized regression next class.

Smoothing splines

$$\text{minimize } \underbrace{\sum_{i=1}^n (y_i - g(x_i))^2}_{\text{RSS}} + \lambda \underbrace{\int g''(t)^2 dt}_{\text{Roughness penalty}} \quad (*)$$

It turns out...

- The solution to $(*)$ is a **natural cubic spline**
- The solution has knots at every unique value of x
- The **effective degrees of freedom** of the solution is calculable
- $\lambda \longleftrightarrow df$

Coding tip: In **R** with the `gam` library you can use the syntax `s(x, df)` in your regression formula to fit a smoothing spline with `df` effective degrees of freedom.

Smoothing Spline

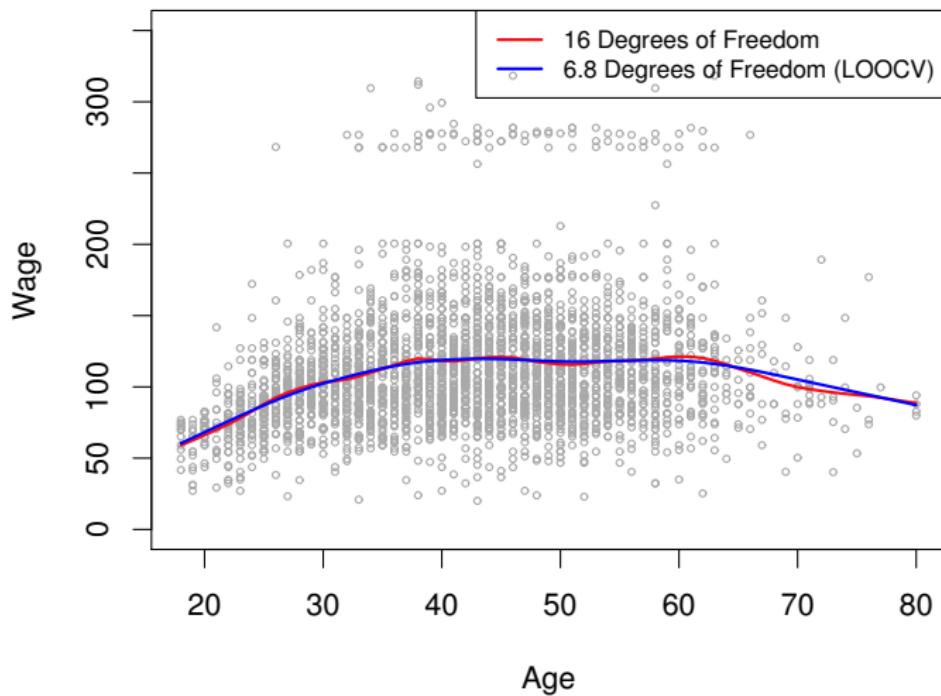


Figure: 7.8 from ISLR. We'll introduce LOOCV (leave-one-out cross-validation) in Part II of today's class

Another method people like: Local regression

Local Regression

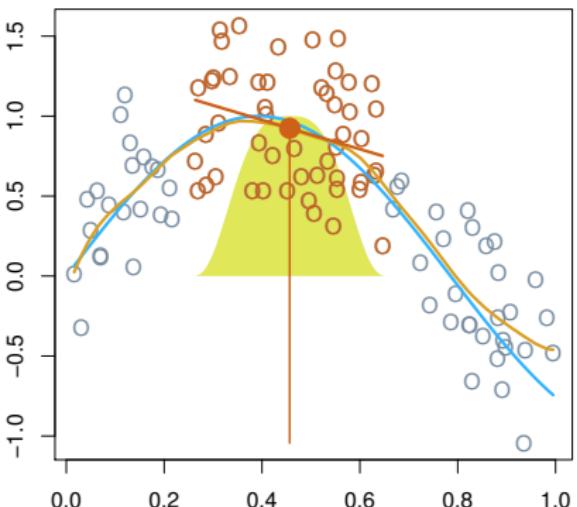
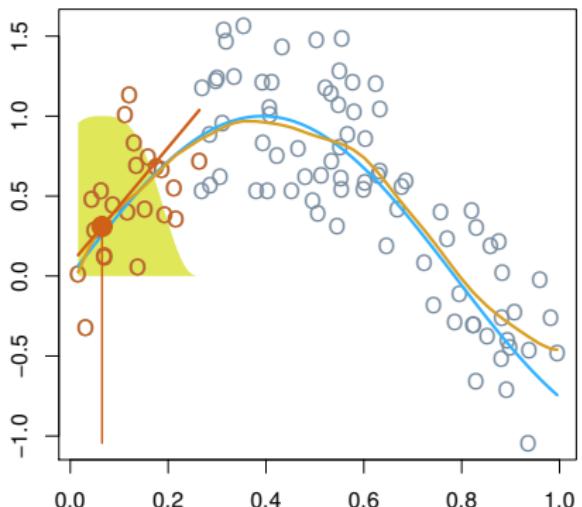
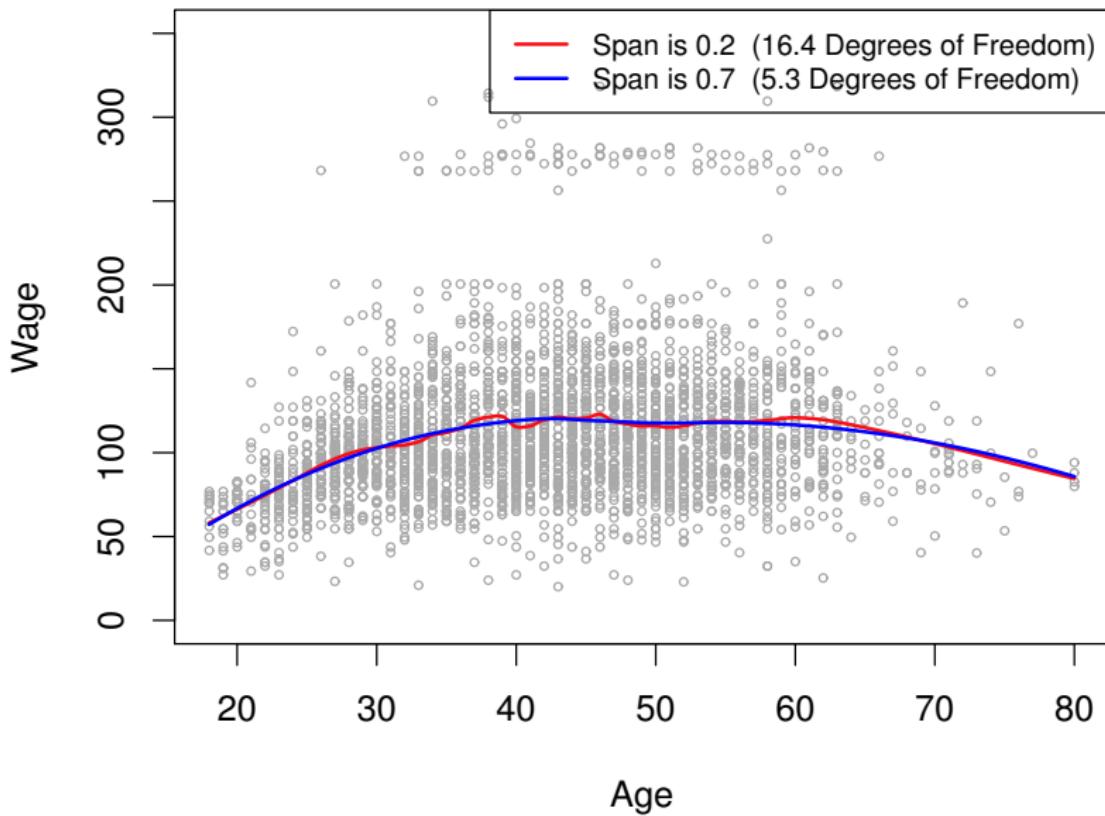


Figure: 7.9 from ISLR. Local regression (enabled as `lo(x)` and `loess(x)`)

Local Linear Regression



Putting everything together: Additive Models

- Recall the Linear Regression Model

$$Y = \beta_0 + \sum_{j=1}^p \beta_j X_j + \epsilon$$

- We can now extend this to the far more flexible Additive Model

$$Y = \beta_0 + \sum_{j=1}^p f_j(X_j) + \epsilon$$

- Each f_j can be any of the different methods we just talked about:
Linear term ($\beta_j X_j$), Polynomial, Step Function, Piecewise
Polynomial, Degree- k spline, Natural cubic spline, Smoothing spline,
Local linear regression fit, ...
- You can mix-and-match different kinds of terms
- The `gam` and `mgcv` packages enable Additive Models in **R**

Additive Models: Boston housing data

Using the `gam`⁴ library, we fit the model

```
gam(medv ~ s(lstat, 5) + lo(ptratio) +  
      cut(crim, breaks = c(-Inf, 1, 10, 25, Inf)),  
      data = Boston)
```

- This amounts to an additive model

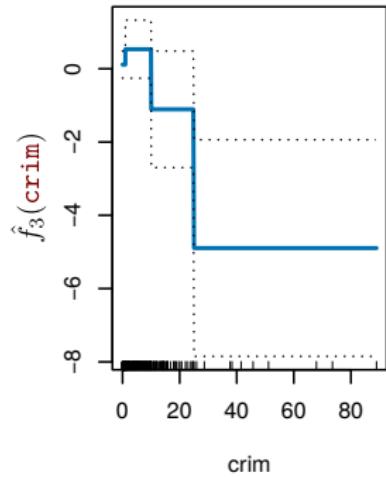
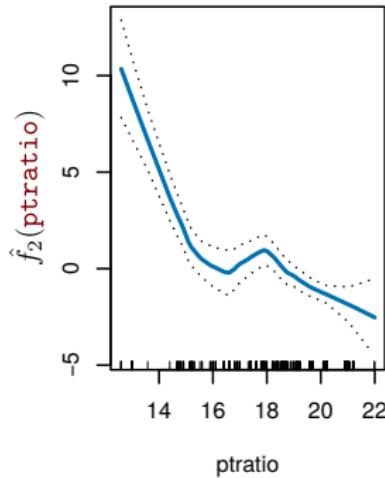
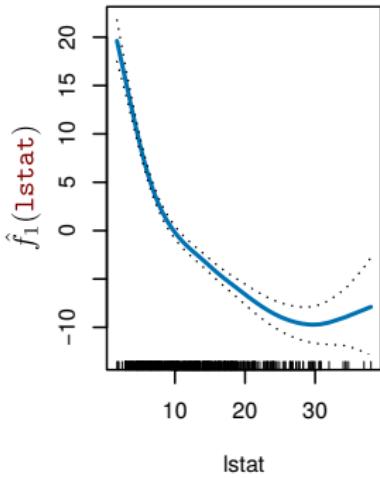
$$\text{medv} = f_1(\text{lstat}) + f_2(\text{ptratio}) + f_3(\text{crim}) + \epsilon$$

with terms:

- $f_1(\text{lstat})$ smoothing spline with 5 df
- $f_2(\text{ptratio})$ local linear regression
- $f_3(\text{crim})$ step function with breaks at $\text{crim} = 1, 10, 25$

⁴You can use `lm` here instead, but `gam` has built-in plotting routines to help better visualize the model fits.

Additive Models: Boston housing data



```
gam(medv ~ s(lstat, 5) + lo(ptratio) +
    cut(crim, breaks = c(-Inf, 1, 10, 25, Inf)),
    data = Boston)
```

Summary

- **Splines** are a nice way of modeling *smooth* regression functions
- To increase the *flexibility* of a **spline**, we increase the number of **knots**
- **Natural cubic splines** allow us *retain the model complexity of a cubic spline while adding two extra interior knots at the cost of restricting our model to be linear outside the range of the observed data*
- **Smoothing splines** enable us to avoid the problem of **knot selection** altogether, and instead specify a single parameter: the desired effective **degrees of freedom** for the fit
- We can put everything together into an **Additive Model**

$$Y = \beta_0 + \sum_{j=1}^p f_j(X_j) + \epsilon$$

where each f_j can be any of the fits we talked about.

Introduction to Model Selection

At this stage, we have a lot of questions.

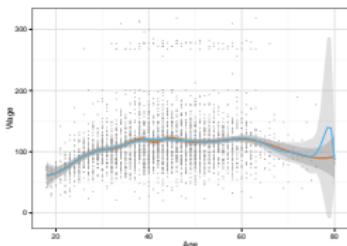
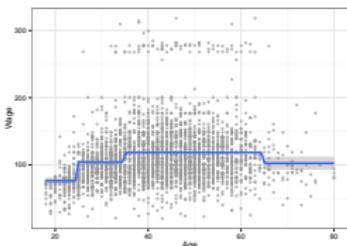
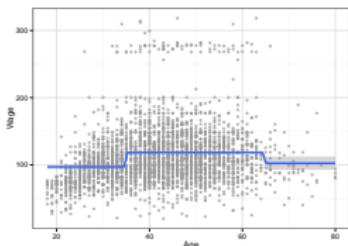
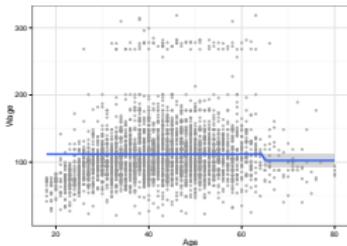
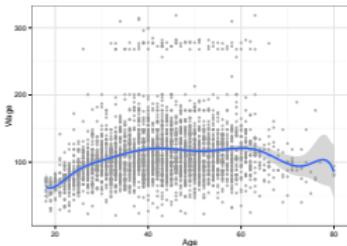
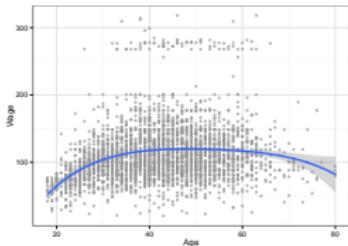
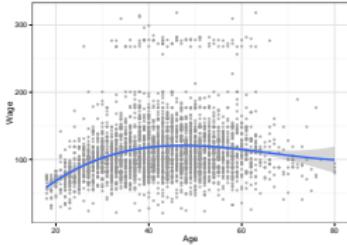
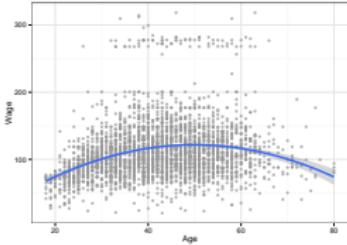
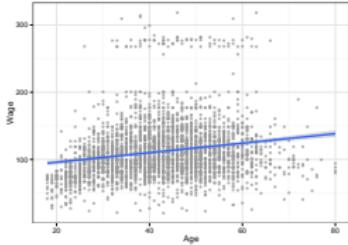
- How do we choose which variables to include in a linear regression?
- How do we choose the degree k in a polynomial regression?
- How do we choose the cuts in a step function regression?
- How do we decide on how many knots to place and where to place them when fitting regression splines?
- How should we choose λ or the effective degrees of freedom for a smoothing spline?
- Which variables should we include in an additive model, and what form should we pick for each f_j term?

All of these questions are essentially asking the same thing...

How do we pick the best model?



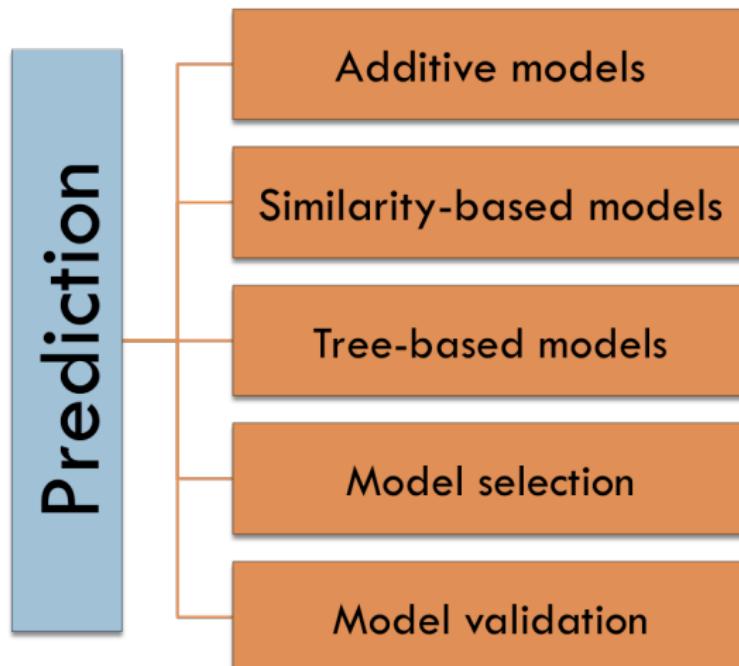
How do we pick the best **statistical** model?



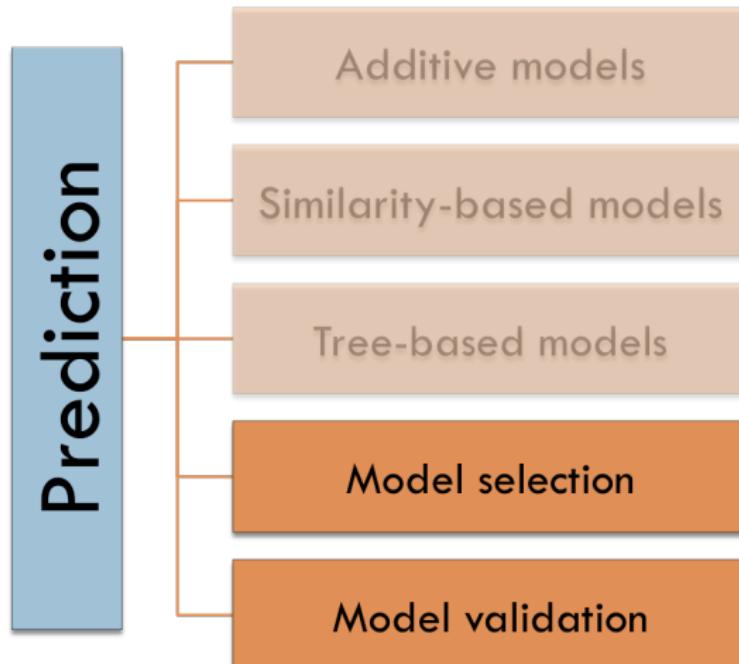
End of Part I

10 minute break

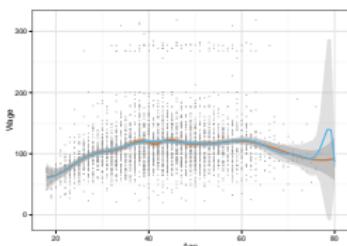
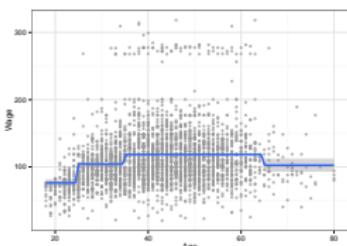
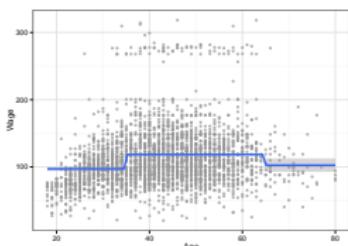
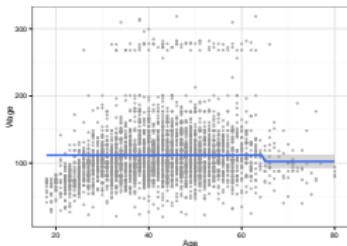
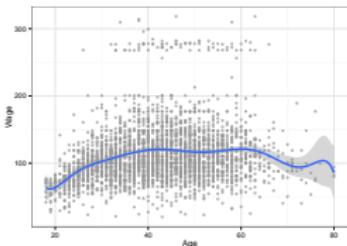
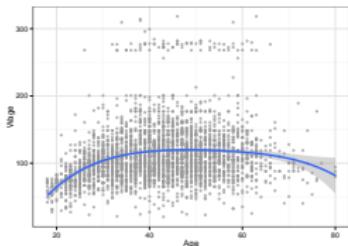
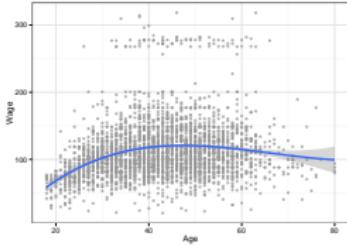
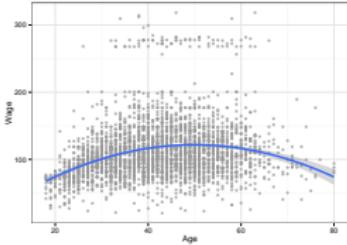
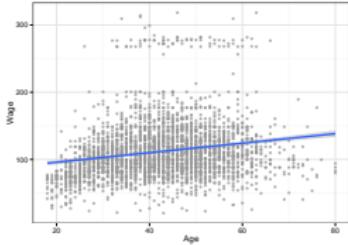
Prediction topics: Part II



Prediction topics: Part II



How do we pick the best **statistical** model?



Resampling methods

- Part II of today's lecture covers several topics that fall into the category of Resampling methods
- These are approaches for using a single observed data set to answer questions such as:
 - Which model generalizes best to *unseen data*?
 - What is the prediction error of my model?
 - What is the uncertainty of my estimate?
 - How can I form a confidence interval for a complex parameter?

Agenda for Part II

- **Review: Central Themes**
- **Test error vs. Training error**
- **Resampling methods**
 - Validation set approach (Train/Test split)
 - Cross-validation
- **Stepwise/Criteria-based methods (Next class)**
 - Best subset selection
 - Stepwise model selection
 - AIC, BIC

How should we think about model selection?

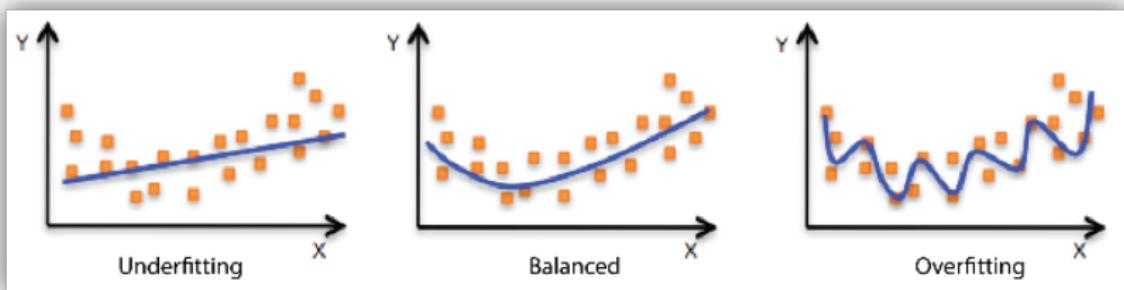
Let's remind ourselves of the first Central Theme of this class.

1. **Generalizability:** We want to construct models that **generalize well to unseen data**
 - i.e., We want to:
 - ➊ Add variables/flexibility as long as doing so helps capture **meaningful trends in the data** (avoid *underfitting*)
 - ➋ Ignore meaningless random fluctuations in the day (avoid *overfitting*)

How should we think about model selection?

Let's remind ourselves of the first Central Theme of this class.

1. **Generalizability:** We want to construct models that **generalize well to unseen data**
 - i.e., We want to:
 - ① Add variables/flexibility as long as doing so helps capture **meaningful trends in the data** (avoid *underfitting*)
 - ② Ignore meaningless random fluctuations in the data (avoid *overfitting*)



Assessing Model Accuracy

- Suppose we fit a model $\hat{f}(x)$ to some **training data**:
 $\text{Train} = \{x_i, y_i\}_{i=1}^n$.
- We want to assess how well \hat{f} performs
- Can compute: Average squared prediction error over Train

$$\text{MSE}_{\text{Train}} = \text{Ave}_{i \in \text{Train}} \left[(y_i - \hat{f}(x_i))^2 \right]$$

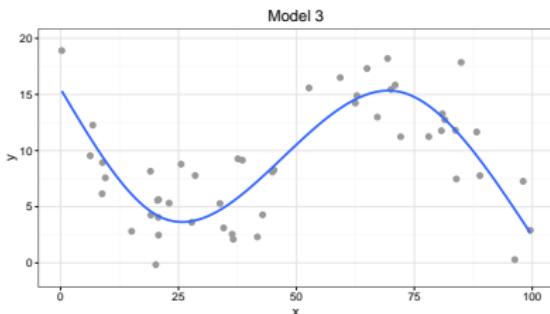
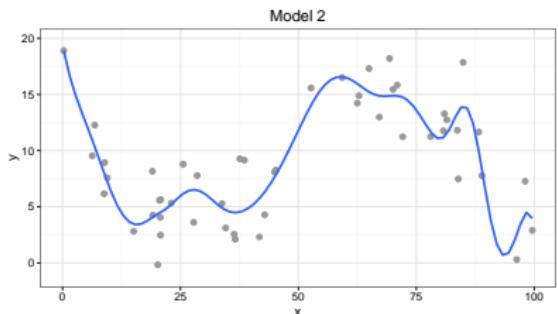
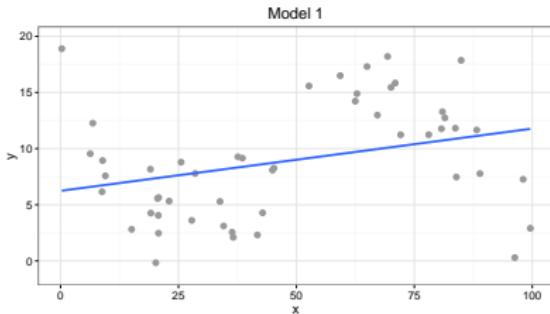
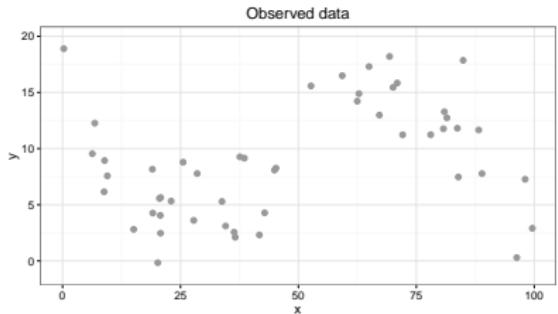
- But this may push us towards more *overfit* models.
- Instead, we should compute it using fresh **test data**:
 $\text{Test} = \{x_i, y_i\}_{i=1}^m$:

$$\text{MSE}_{\text{Test}} = \text{Ave}_{i \in \text{Test}} \left[(y_i - \hat{f}(x_i))^2 \right]$$

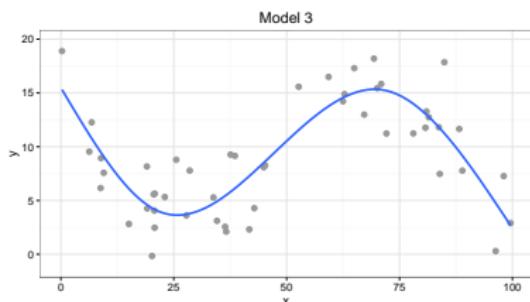
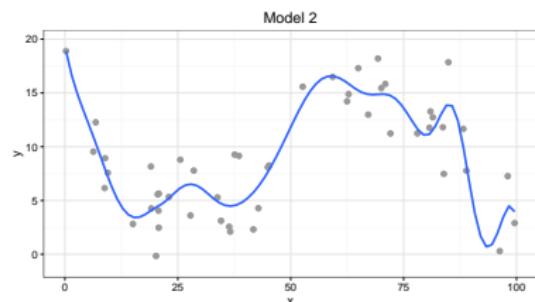
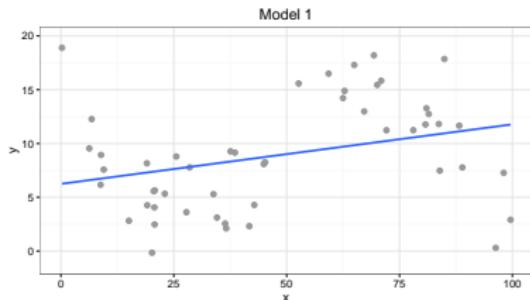
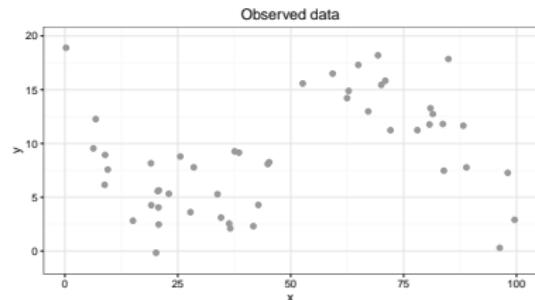
- This would tell us if \hat{f} generalizes well to new data

Assessing Model Accuracy: Training Error vs. Testing Error

Here are three different models fit to the same small Train data set.
Which of these three is the best model?



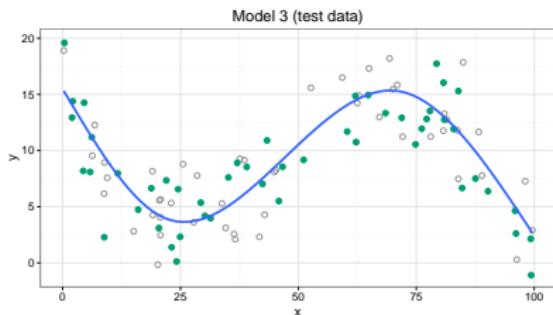
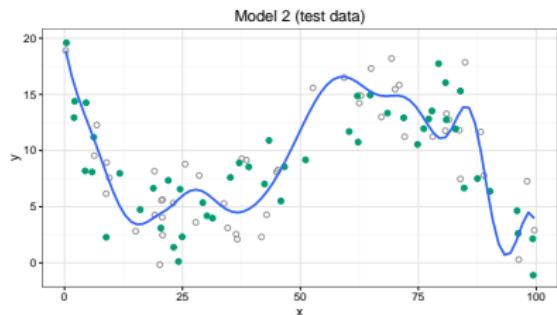
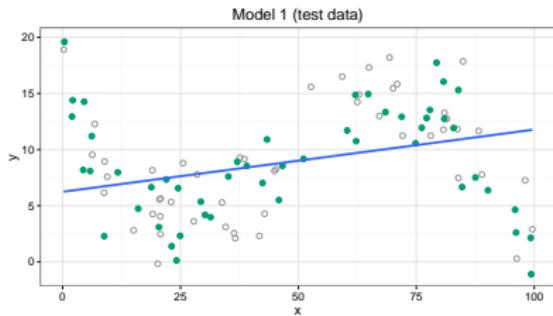
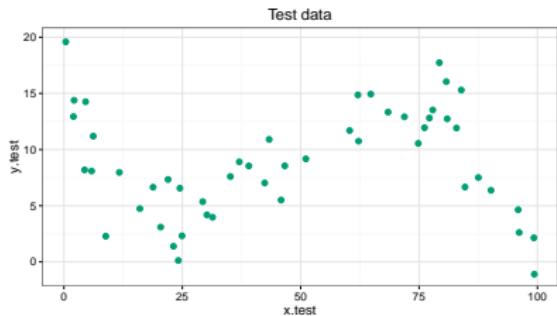
Assessing Model Accuracy: Training Error vs. Testing Error



Model	1	2	3
$MSE_{Train} = RSS/n$	23.2	5.2	7.5

Assessing Model Accuracy: Training Error vs. Testing Error

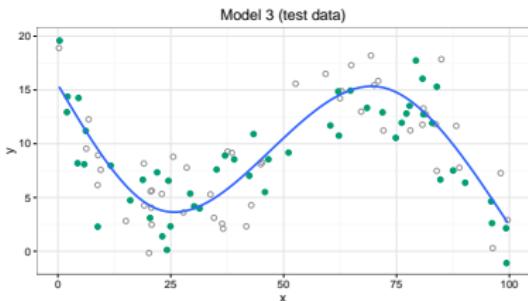
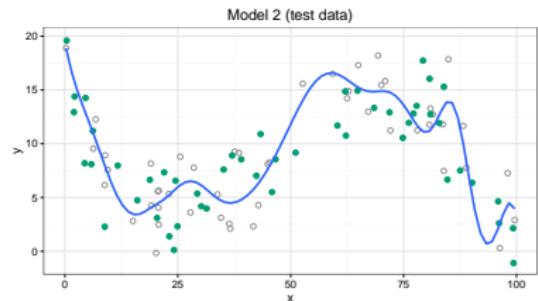
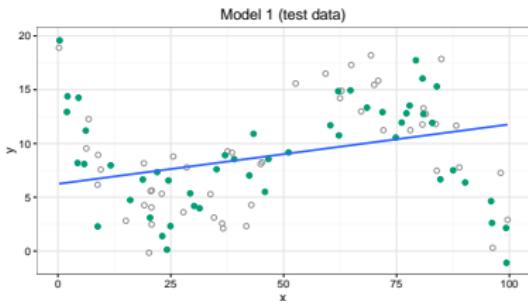
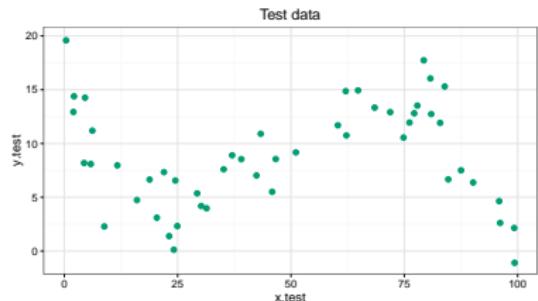
Here are some new observations, which form our Test data. How well do our models fit the test data?



Solid green points: Test data

Open grey circles: Train data

Assessing Model Accuracy: Training Error vs. Testing Error



Model	1	2	3
MSE _{Train}	23.2	5.2	7.5
MSE _{Test}	24.6	10.3	7.0

Assessing Model Accuracy

- As we increase the **flexibility** of our model, our **training set error** always decreases
- The same is **not** true for **test set error**
- The **test set error** will decrease as we add flexibility that helps to capture useful trends
- As we add *too much* flexibility, the **test set error** will begin to *increase* due to model overfitting

How well could we possibly do?

- There is a limit on how well we can do on any given prediction task
- Even if we knew the true regression function $f(x) = \mathbb{E}(Y | X = x)$, we would still have error:

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

- $\epsilon = Y - f(x)$ is the **irreducible error**.
- Even if we knew $f(x)$, we would still make errors in prediction: Because at each value of x , there's typically a distribution of possible Y values
- Our average prediction error will always be at least $\text{Var}(\epsilon)$

Bias-Variance trade-off in prediction

Let's remind ourselves of the second Central Theme of the class:

2. **Bias-Variance trade-off:** To minimize prediction error, we need to find the right balance between the *bias* and *variance* of our predictor \hat{f}
- Suppose we some Train data, which we use to build a predictor \hat{f}
- For any predictor \hat{f} , one can decompose the the expected test MSE^5 at a new data point x_0 as:

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

⁵For more details, read §2.2.2 of ISL

Bias-Variance trade-off in prediction

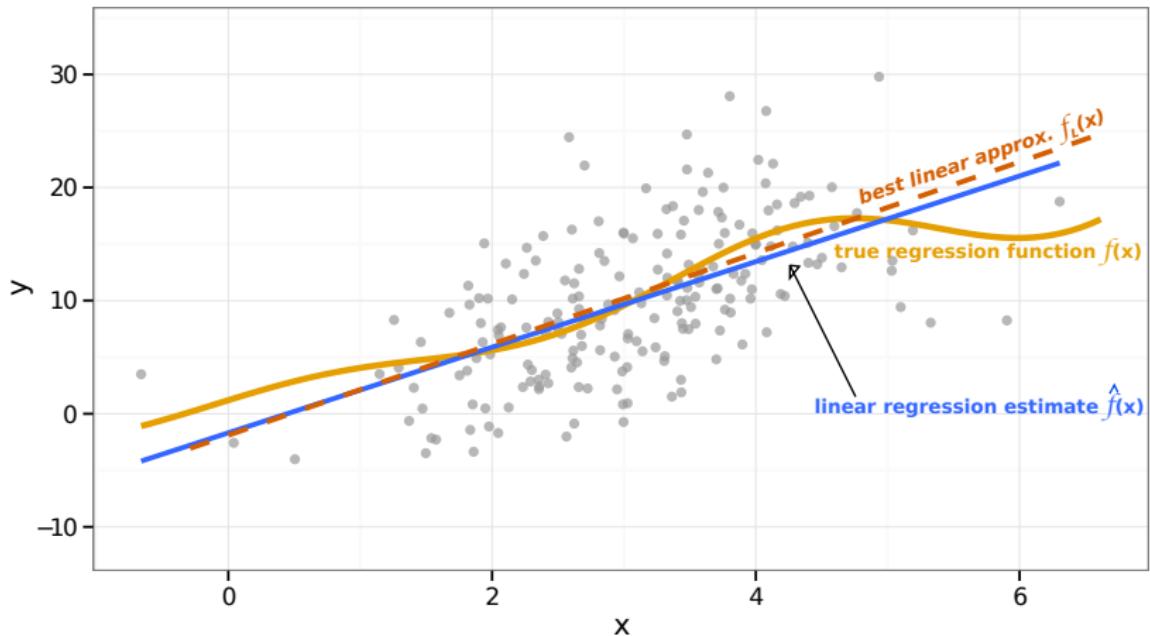
Let's understand what this equation is saying

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

- $\mathbb{E}[(y_0 - \hat{f}(x_0))^2]$ is the **expected test MSE**:
 - It's the average test MSE we would get if we repeatedly constructed \hat{f} using a large number of random training sets, and tested each at x_0 , with random realizations of y_0 .
- $\text{Var}(\hat{f}(x_0))$ is the **variance** of \hat{f} at x_0 .
 - It's the variability of $\hat{f}(x_0)$ around $\mathbb{E} \hat{f}(x_0)$
- $\text{Bias}(\hat{f}(x_0))$ is the **bias** of \hat{f} at x_0

$$\begin{aligned}\text{Bias}(\hat{f}(x_0)) &= \mathbb{E} \hat{f}(x_0) - \mathbb{E}(Y \mid X = x_0) \\ &= \mathbb{E} \hat{f}(x_0) - f(x_0)\end{aligned}$$

- $\text{Var}(\epsilon)$ is the **irreducible error**



Variance measures how much $\hat{f}(x_0)$ varies around $f_L(x_0)$

Bias measures how far $f_L(x_0)$ is from the true regression function $f(x_0)$

Bias-Variance trade-off in prediction

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

- This quantity is the *expected test MSE* at a particular value x_0
- The **overall test MSE** can be calculated by further averaging this quantity over all possible values of x_0 in the test set

Bias-Variance trade-off in prediction

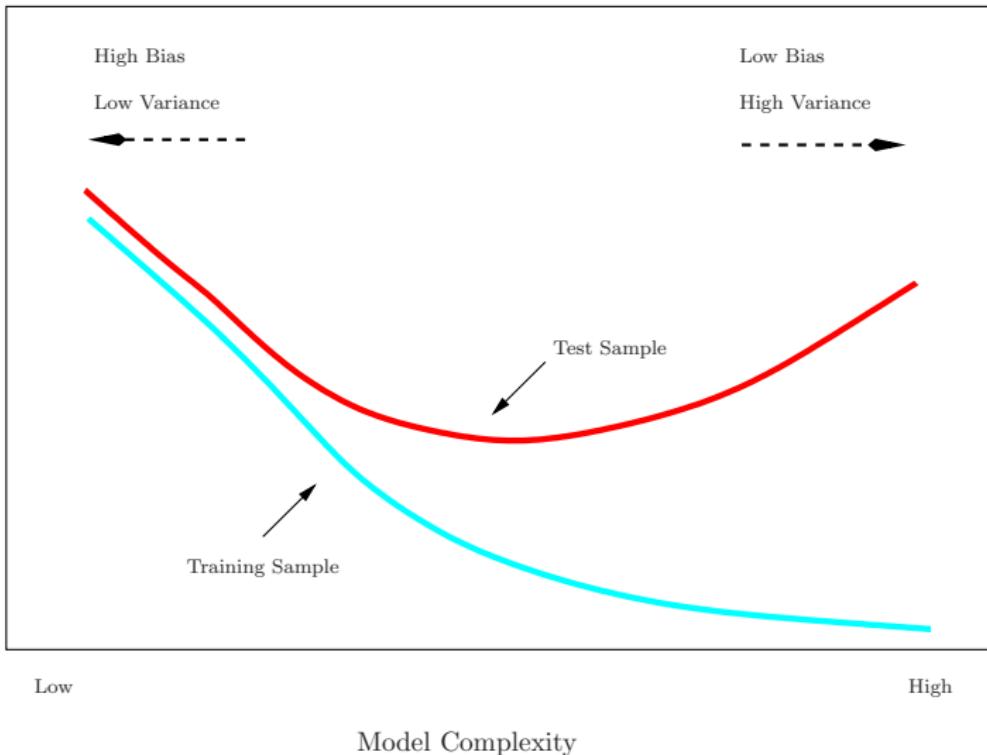
Let's think about this equation in practical terms

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

- **Variance** refers to how much \hat{f} would change if we estimated it using a different random set of training data
 - Training data is random, different Training data will result in different \hat{f}
 - Ideally, \hat{f} should not change tremendously between different Training sets
 - If small changes in Training data change \hat{f} by a lot, then \hat{f} would have **High Variance**
- **Bias** refers to error introduced by modeling a complex real-world problem by a simpler statistical model
 - E.g., Linear regression assumes a linear relationship between Y and the inputs X_1, \dots, X_p . This is almost always an over-simplification
 - If the true f is very complex and \hat{f} is too inflexible, \hat{f} will have **High Bias**

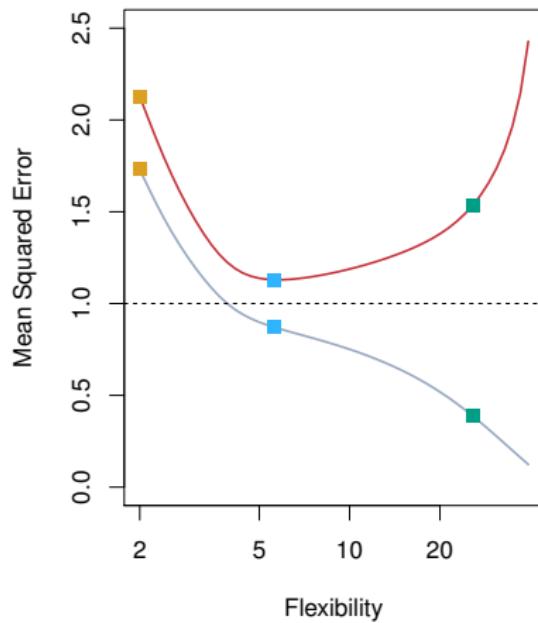
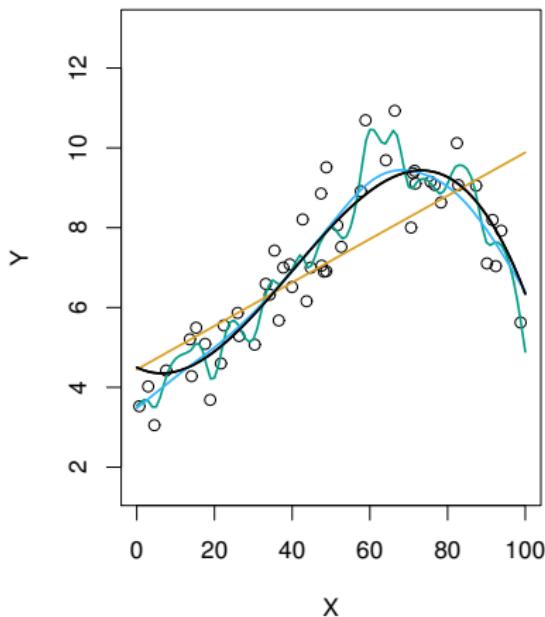
Key Picture

Prediction Error



Training Error vs Test Error: Example 1

Dashed line on MSE plot shows the irreducible error $\text{Var}(\epsilon)$

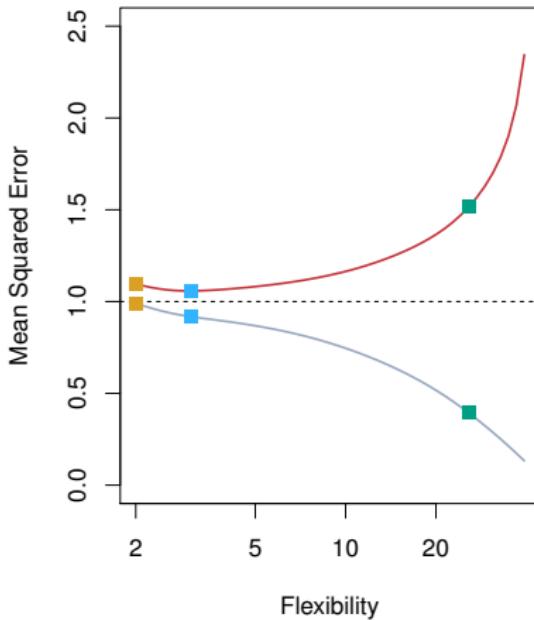
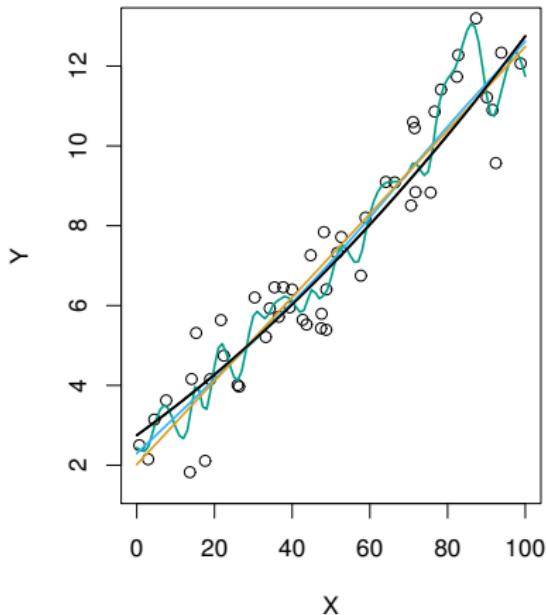


Left: Models fit to training data. Black curve is the truth.

Right: $\text{MSE}_{\text{Train}}$ in grey, MSE_{Test} in red

Training Error vs Test Error: Example 2

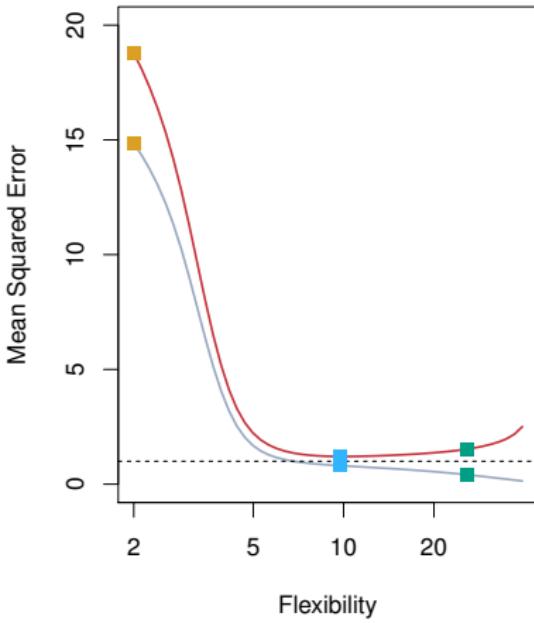
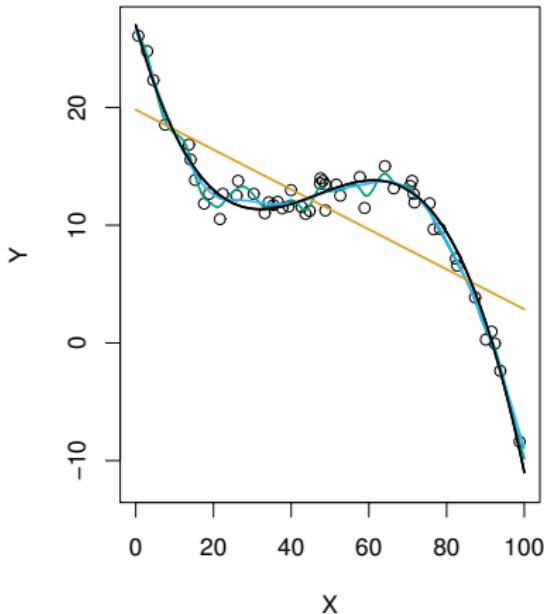
Dashed line on MSE plot shows the irreducible error $\text{Var}(\epsilon)$



Truth (black curve) is simpler/smooth (nearly linear), so the smoother fit and linear model both perform really well.

Training Error vs Test Error: Example 3

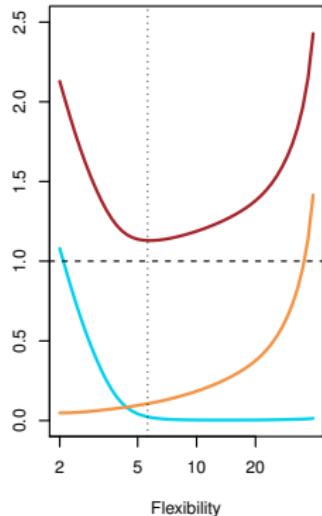
Dashed line on MSE plot shows the irreducible error $\text{Var}(\epsilon)$



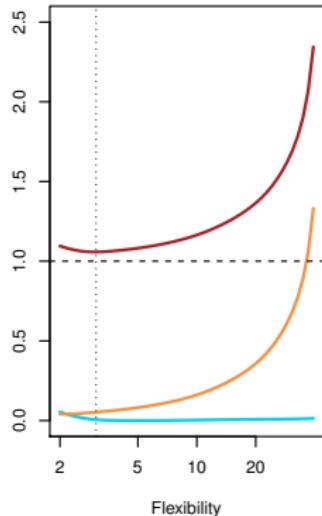
Truth is wiggly, but noise level is very low. The more flexible fits perform the best.

Bias-Variance Trade-off for the three examples

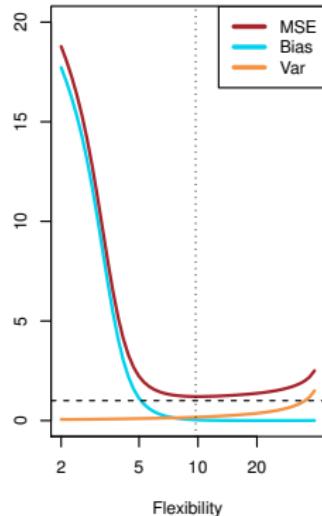
Dashed lines show the irreducible error $\text{Var}(\epsilon)$



Example 1



Example 2



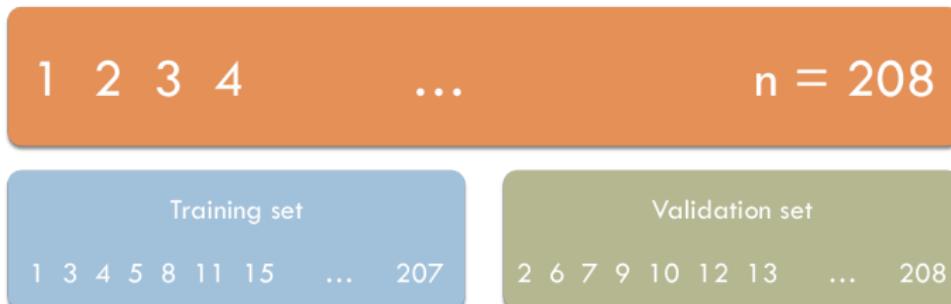
Example 3

How do we estimate the Test Error?

- In reality, we only get one set of data to use for model fitting, model selection, and model assessment
- We will now discuss two methods for estimating Test Error:
 - Validation set approach
 - Cross-validation
- We will explain why Cross-validation is generally the best approach

Validation set approach

- I. Randomly divide the available **data** into two parts: a **Training set** and a **Validation** or **Hold-out** set



2. Construct \hat{f} by fitting your model on the **Training set**
3. Use \hat{f} to predict responses for all the points in the **Validation set**, and calculate the resulting MSE
4. Pick the simplest model that has among the *lowest* MSE on the **Validation set**

Example: Automobile data

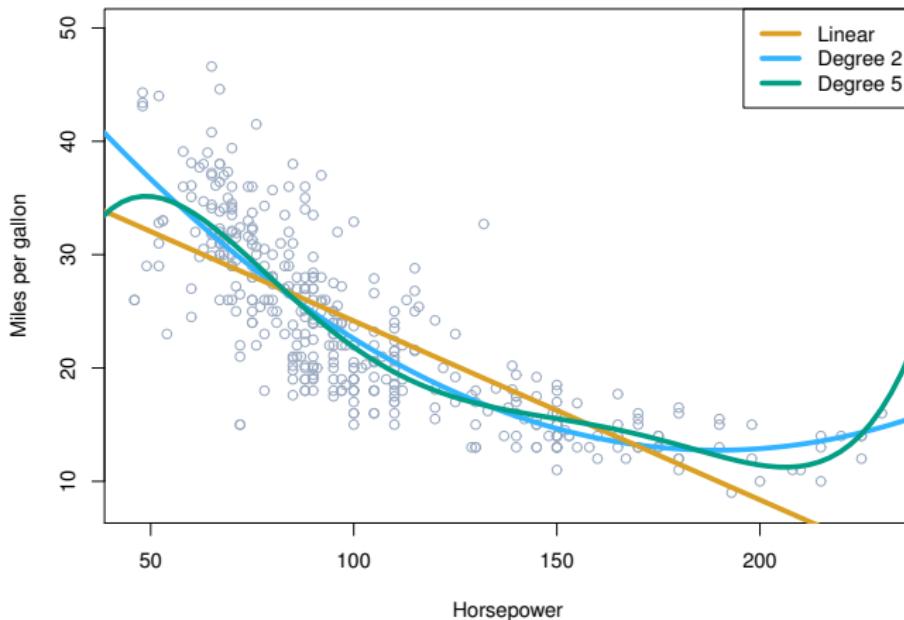


Figure: 3.8 from ISL. We want to figure out what degree polynomial we want to use to model the relationship between $Y = \text{Miles per gallon}$ and $X_1 = \text{Horsepower}$

Example: Automobile data

- Want to figure out what degree polynomial to use
- Randomly split the 392 observations into two sets of 196 data points.
Train on the first, calculate errors on the second.

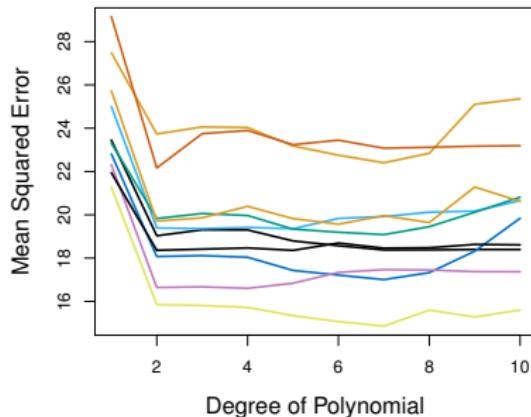
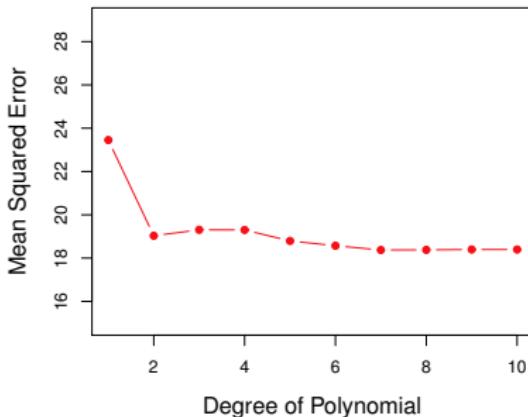


Figure 5.2 from ISL. Left panel shows MSE for a single split. Right panel shows MSE curves for 10 randomly chosen splits.

Example: Automobile data

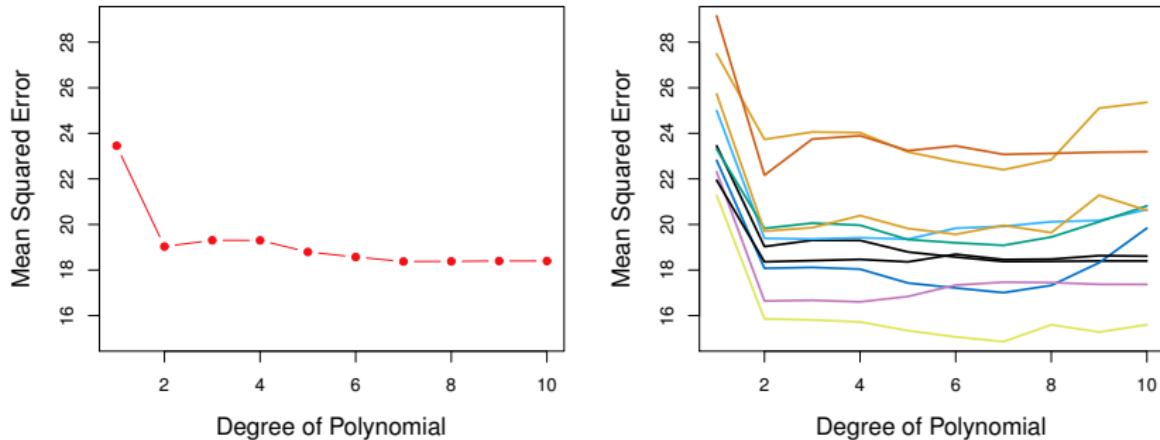
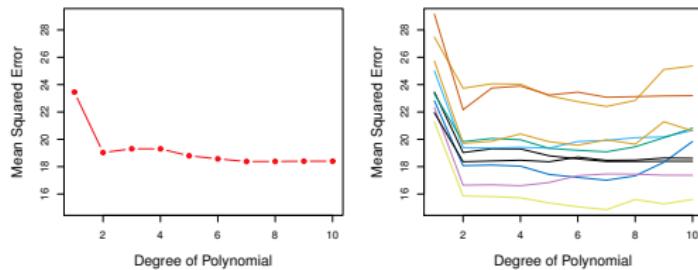


Figure 5.2 from ISL. Left panel shows MSE for a single split. Right panel shows MSE curves for 10 randomly chosen splits.

- All splits agree that *quadratic* is much better than *linear*
- Degree > 2 fits don't seem to perform considerably better than *quadratic*

Problems with the Validation set approach



1. As observed in the right-hand panel, the **estimates of Test Error** are **highly variable**. Estimates depend a lot on the randomly chosen split.
2. Only a subset of the data (the ones randomized to the Training set) are used to fit the model.
 - Models tend to perform worse when trained on fewer observations
 - The *Validation set error* tends to **overestimate** the *Test Error* for the model trained on the entire data set

Cross-validation is a refinement of the Validation set approach that addresses these two issues.

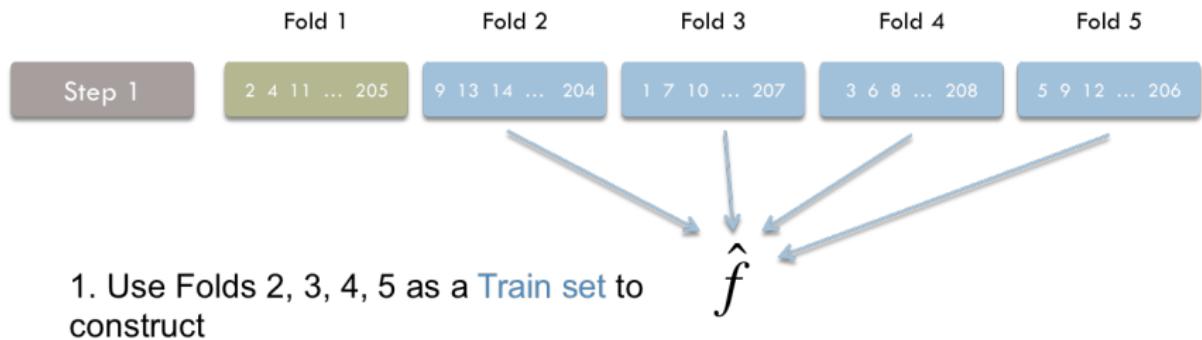
K-fold Cross-validation (CV)

- The most **widely used approach** for estimating Test Error
- Error estimates can be used to **select** the best model, and also reflect the **test error of the final chosen model**
- Main idea: ***K*-fold Cross-validation**
 1. Randomly split the data into K equal-sized parts (“folds”)
 2. Give each part the chance to be the **Validation set**, treating the other $K - 1$ parts (combined) as the **Training set**
 3. Average the test error over all of the folds
 4. Pick the simplest model among those with the lowest CV-error
 5. **Final model:** Refit model on the entire data set.
- Most common choices of K : 5, 10, n
 - The case $K = n$ is also known as **Leave-one-out Cross-validation (LOOCV)**

The Picture for 5-fold Cross-validation

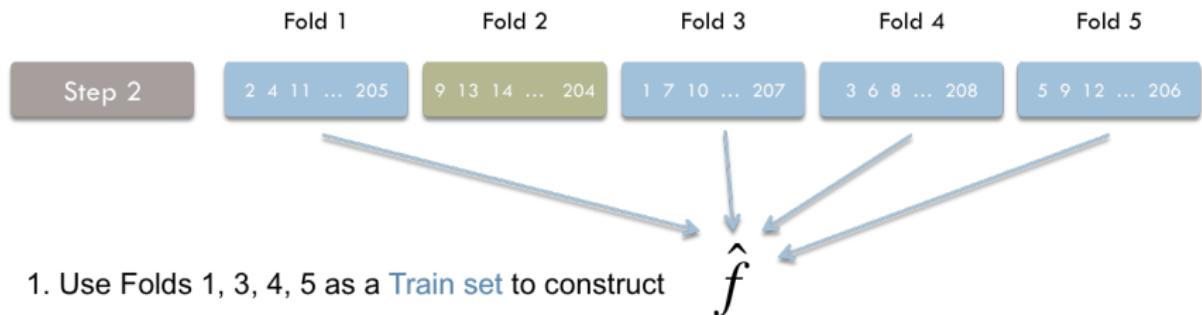
Full data	1	2	3	4	...	n = 208																			
Step 1	2	4	11	...	205	9	13	14	...	204	1	7	10	...	207	3	6	8	...	208	5	9	12	...	206
Step 2	2	4	11	...	205	9	13	14	...	204	1	7	10	...	207	3	6	8	...	208	5	9	12	...	206
Step 3	2	4	11	...	205	9	13	14	...	204	1	7	10	...	207	3	6	8	...	208	5	9	12	...	206
Step 4	2	4	11	...	205	9	13	14	...	204	1	7	10	...	207	3	6	8	...	208	5	9	12	...	206
Step 5	2	4	11	...	205	9	13	14	...	204	1	7	10	...	207	3	6	8	...	208	5	9	12	...	206

The Picture for 5-fold Cross-validation



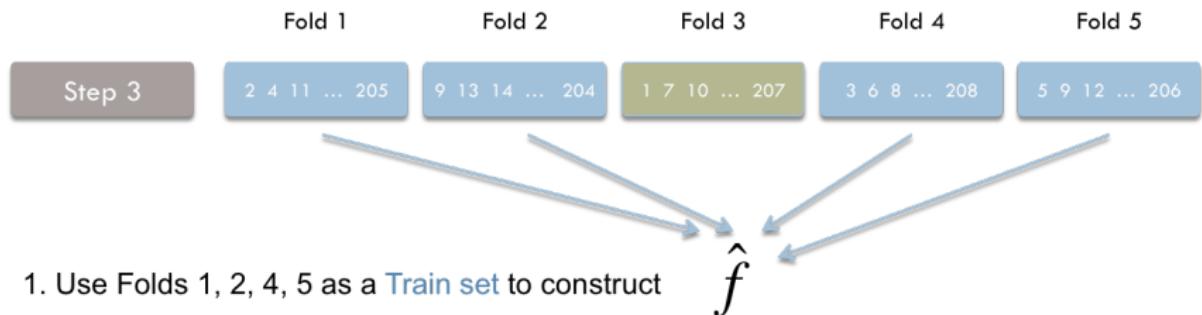
$$MSE_1 = \sum_{i \in \text{Fold 1}} (y_i - \hat{f}(x_i))^2$$

The Picture for 5-fold Cross-validation



$$\text{MSE}_2 = \sum_{i \in \text{Fold 2}} (y_i - \hat{f}(x_i))^2$$

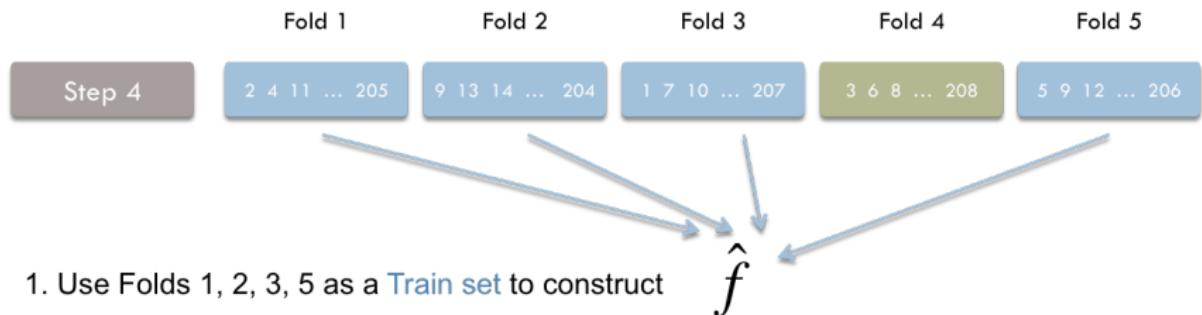
The Picture for 5-fold Cross-validation



2. Use Fold 3 as Test set to calculate error:

$$\text{MSE}_3 = \sum_{i \in \text{Fold 3}} (y_i - \hat{f}(x_i))^2$$

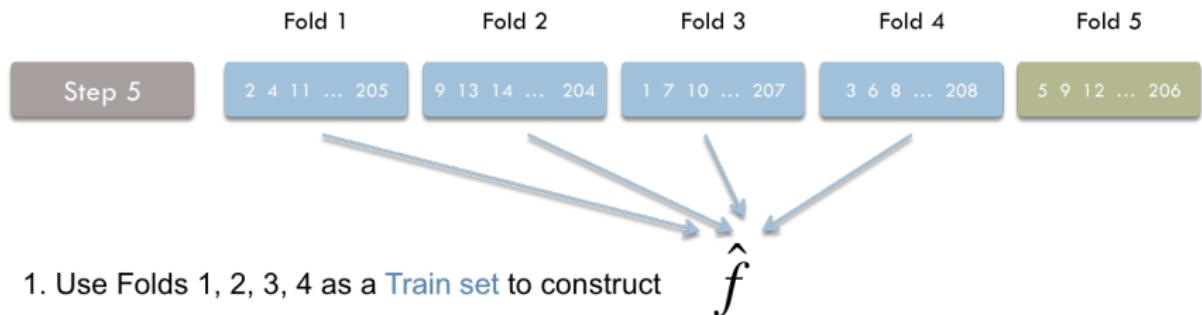
The Picture for 5-fold Cross-validation



2. Use Fold 4 as Test set to calculate error:

$$\text{MSE}_4 = \sum_{i \in \text{Fold 4}} (y_i - \hat{f}(x_i))^2$$

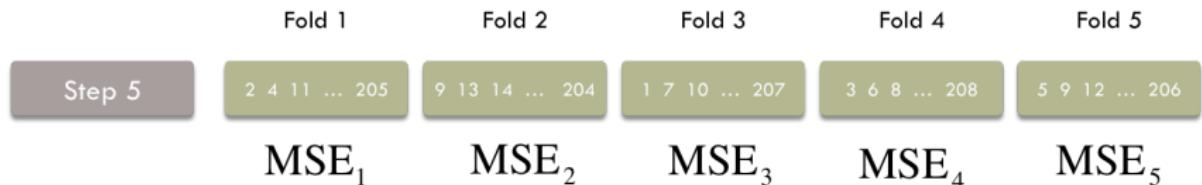
The Picture for 5-fold Cross-validation



2. Use Fold 5 as Test set to calculate error:

$$\text{MSE}_5 = \sum_{i \in \text{Fold 5}} (y_i - \hat{f}(x_i))^2$$

The Picture for 5-fold Cross-validation



Form 5-fold CV estimate of prediction error: $CV_{(5)} = \frac{1}{5} \sum_{k=1}^5 MSE_k$

Cross-validation standard error

- The K -fold CV estimate of prediction error is

$$\text{CV}_{(K)} = \frac{1}{K} \sum_{k=1}^K \text{MSE}_k$$

where MSE_k is the error calculated on Fold k .

- It is also useful to calculate the standard error of the CV estimate
- The typical way of doing this: Calculate the sample standard deviation of $\{\text{MSE}_1, \dots, \text{MSE}_K\}$, then divide by \sqrt{K} :⁶

$$\text{SE}(\text{CV}_{(K)}) = \frac{1}{\sqrt{K}} \text{SD}(\text{MSE}_1, \dots, \text{MSE}_K)$$

⁶This calculation isn't quite right, but it's a widely accepted approach for calculating standard errors for CV error estimates.

Back to the Automobile data

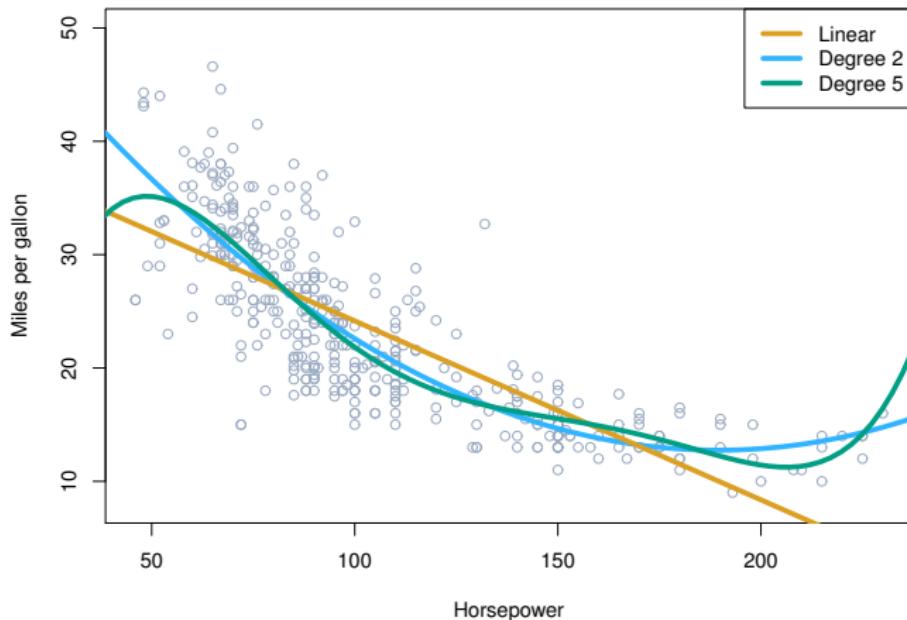


Figure: 3.8 from ISL. We want to figure out what degree polynomial we want to use to model the relationship between $Y = \text{Miles per gallon}$ and $X_1 = \text{Horsepower}$

Example: Automobile data

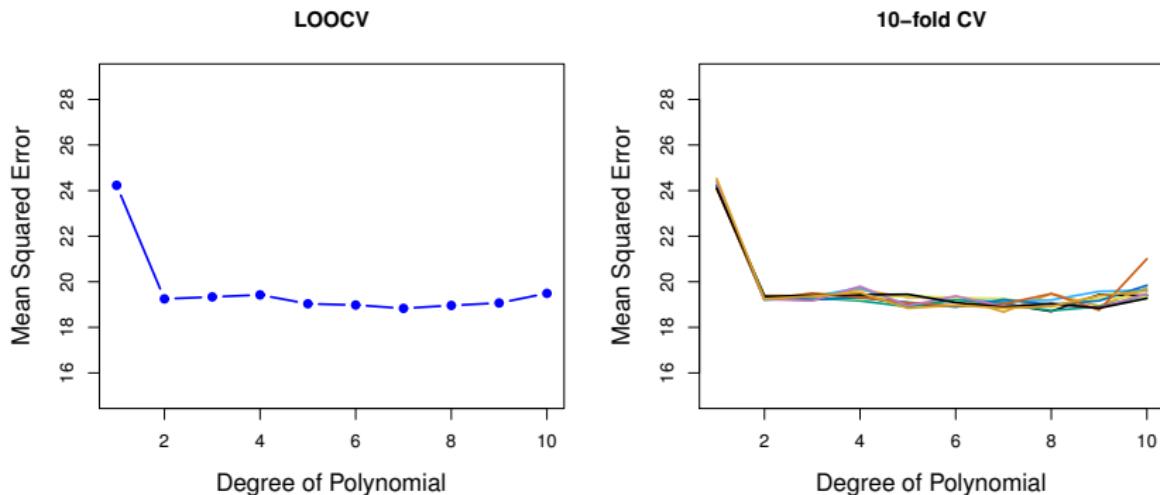
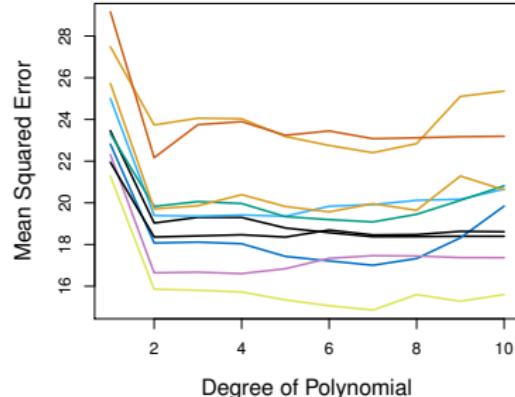


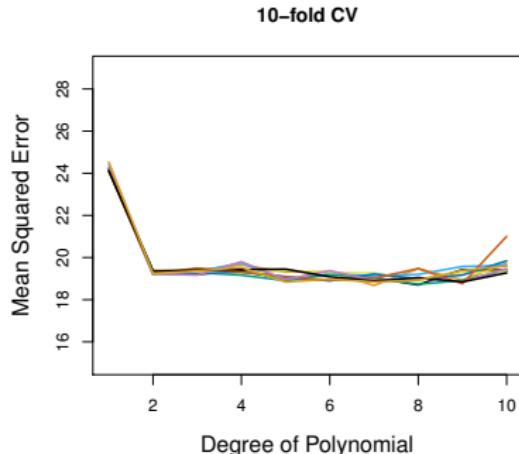
Figure 5.4 from ISL. Left panel shows estimated Test Error for LOOCV. Right panel shows estimated Test Error for 10-fold CV run nine separate times.

Same conclusion as Validation set approach: We should use a [quadratic model](#)

Validation set approach vs. 10-fold CV



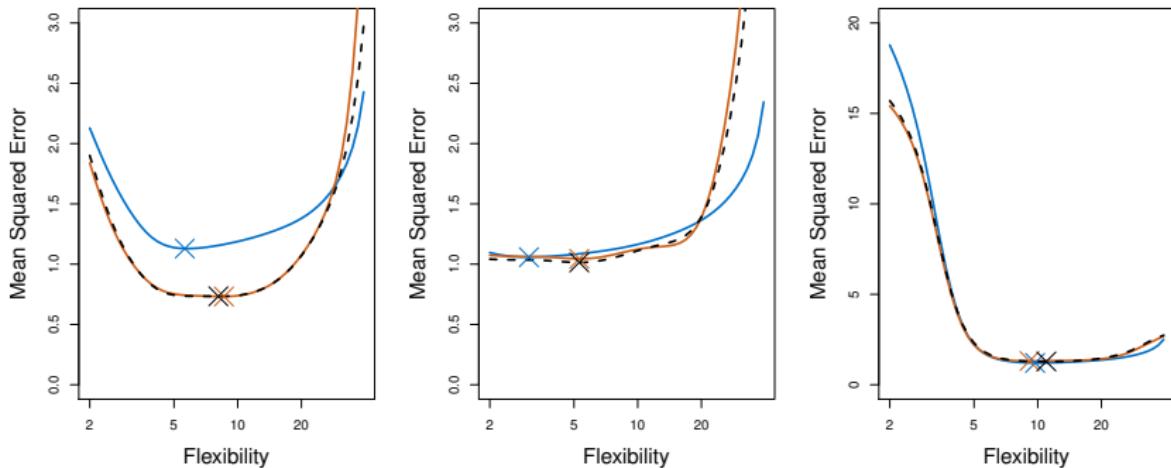
Validation set approach



10-fold CV

- Each plot shows estimated Test Error curves for multiple random splits of the data.
- The 10-fold CV error estimates are **much more stable**
- The Validation set error estimates are **highly variable**

CV error estimate vs. Actual test error



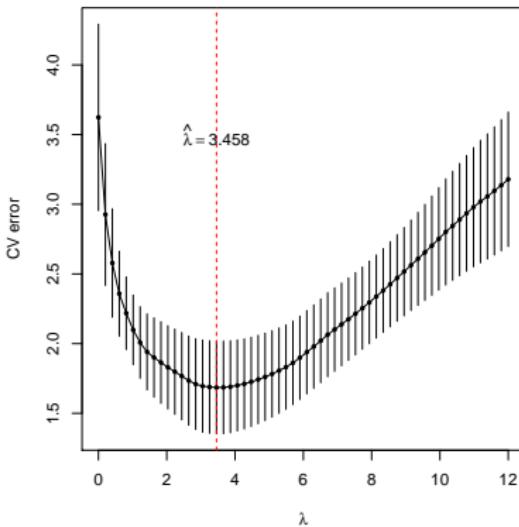
Blue curve: True Test Error

Dashed black curve: LOOCV estimate

Orange curve: 10-fold CV estimate

×'s: indicate *minimum* of each of the curves

The 1-standard error rule

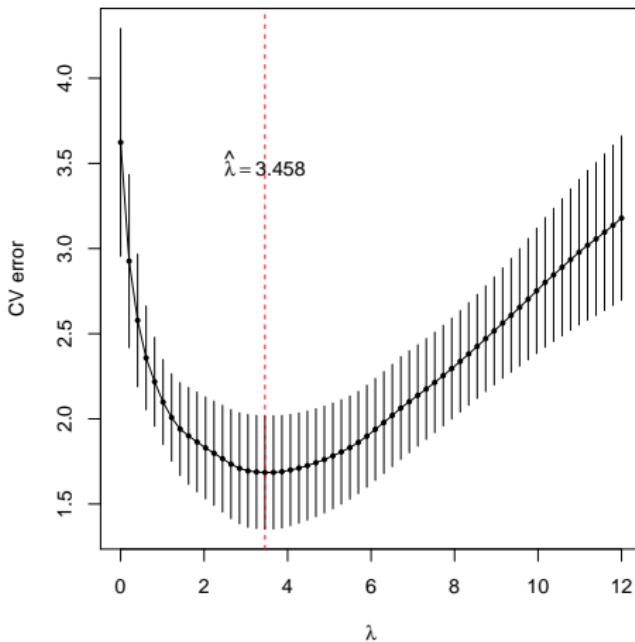


10-fold CV error curve as the tuning parameter λ varies

This plot shows CV error estimates and 1-standard-error bars for a bunch of different choices of a tuning parameter λ .⁷

⁷You can think of λ as the smoothing spline penalty. Large $\lambda \Rightarrow$ simpler model.

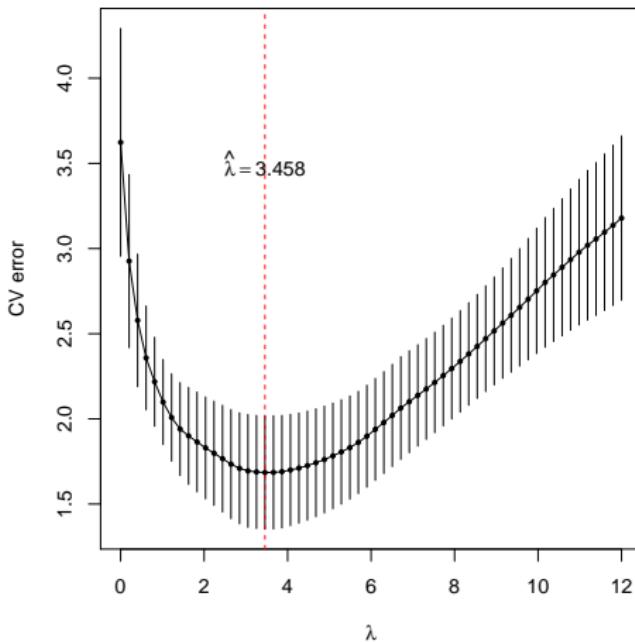
The 1-standard error rule



10-fold CV error curve as the tuning parameter λ varies

- $\lambda = 3.458$ gives us the model with the smallest estimated CV error.
- But we can see from the wide error bars that our prediction error estimates have high uncertainty.

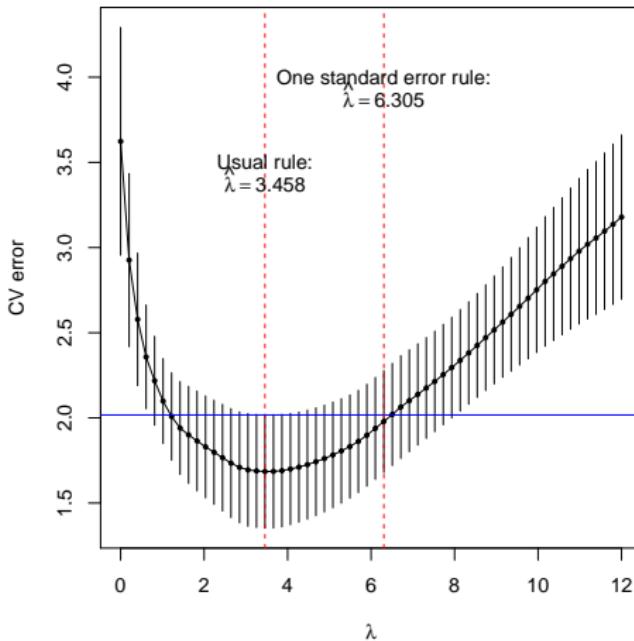
The 1-standard error rule



10-fold CV error curve as the tuning parameter λ varies

- The 1-standard error rule tells us to pick the *simplest model* whose CV error falls inside the 1-SE error bars of the lowest CV error model.

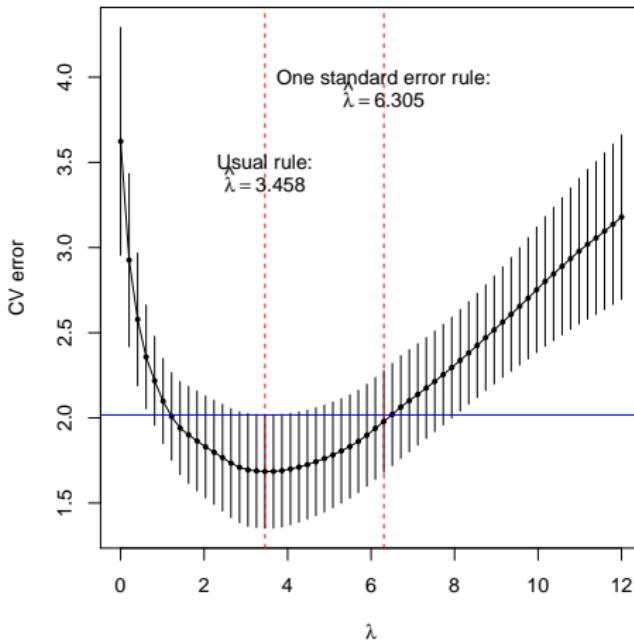
The 1-standard error rule



10-fold CV error curve as the tuning parameter λ varies

- The **1-standard error rule** tells us to pick the *simplest model* whose CV error falls inside the 1-SE error bars of the lowest CV error model.

The 1-standard error rule

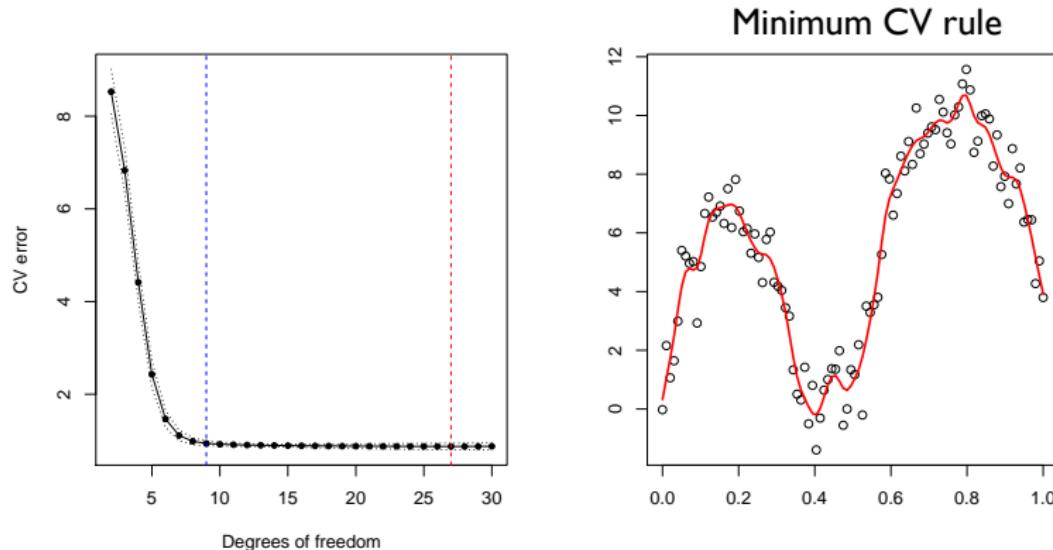


10-fold CV error curve as the tuning parameter λ varies

- **Basic idea:** We can't be certain that the $\lambda = 6.305$ model actually has higher prediction error than the $\lambda = 3.458$ model, so let's err on the side of caution and go with the *simpler* $\lambda = 6.305$ model.

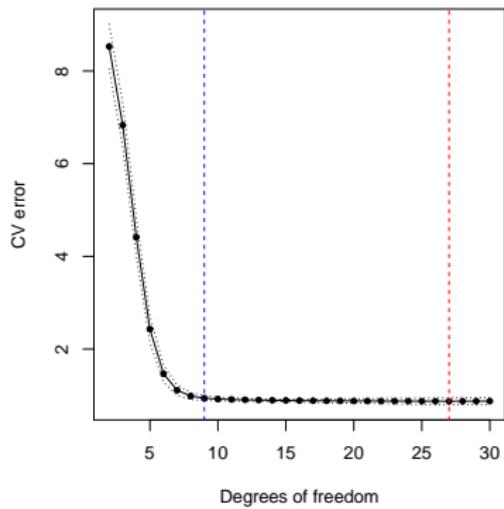
Smoothing spline example

These plots show the results of applying 5-fold cross-validation to select the effective degrees of freedom for a smoothing spline fit to the points in the right panel.

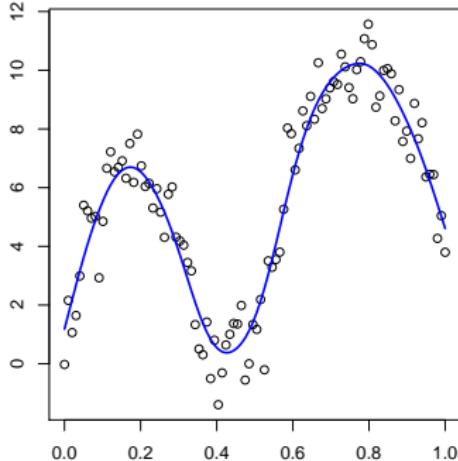


- Even at very large degrees of freedom, the smoothing spline is nicely behaved and has low CV error
- The minimum CV error rule selects a model with 27 degrees of freedom

Smoothing spline example



One standard error rule



- The one standard error rule selects a model with 9 degrees of freedom

Summary: Cross-validation

- We started with the question: *How do we pick the best model?*
- One answer: *Pick the model with the lowest prediction error.*
- The **Validation set approach** and **K -fold Cross-validation** are two **resampling-based** methods for estimating the *prediction error* of a model
- **K -fold Cross-validation** gives much more *stable* and *accurate* estimates of prediction error
- Once we get CV error estimates for the models we're considering, we can either:
 - Pick the model that has the **minimum CV error**; or
 - Use **1-SE rule** and pick the *simplest* model whose error is within 1 standard error of the minimum CV error.
- From this we get: Our chosen model \hat{f} , and an estimate of its *prediction error*

Acknowledgements

All of the lectures notes for this class feature content borrowed with or without modification from the following sources:

- 36-462/36-662 Lecture notes (Prof. Tibshirani, Prof. G'Sell, Prof. Shalizi)
- 95-791 Lecture notes (Prof. Dubrawski)
- *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani
- *Applied Predictive Modeling*, (Springer, 2013), Max Kuhn and Kjell Johnson