SIRCSS – Machine Learning for Social Science

Lecture 1

Martin Arvidsson
Institute of Analytical Sociology, Linköping University

2025-12-01

Some background

Who am I?

- Post doc at IAS, LIU (PhD in *Analytical Sociology*, 2022)
- Before my PhD: MSc in *Statistics & Machine Learning*.
- My research: social networks & collective behavior; CSS data & tools.

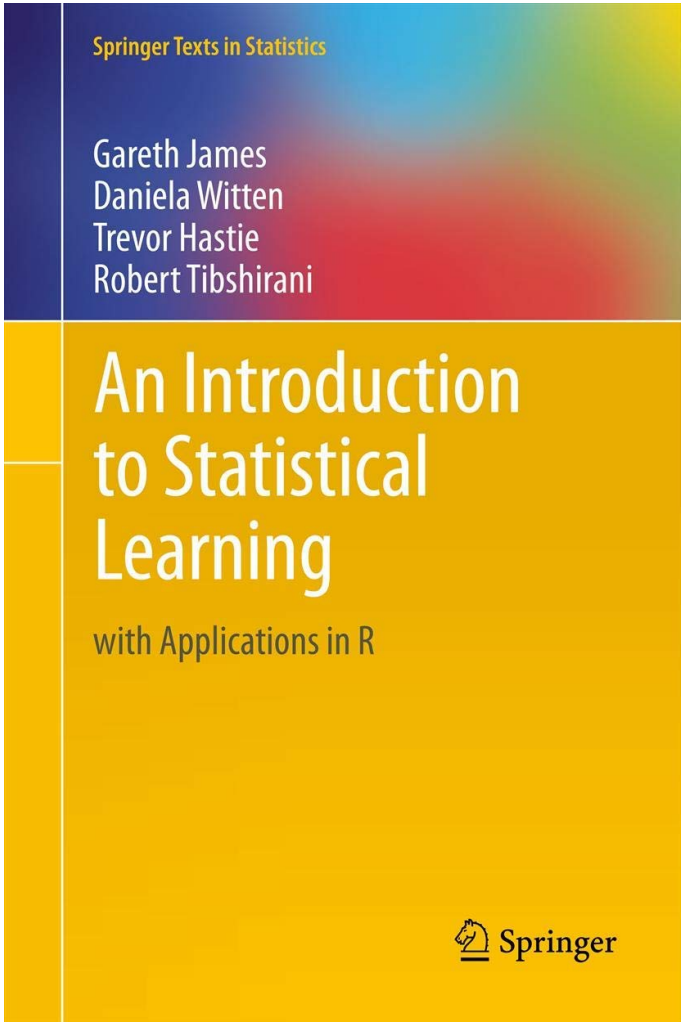
Helping me out: Maël Lecoursonnais & Kazuki Sakamoto


- Maël recent postdoc, Kaz PhD student — both here at IAS.
- They both use ML to study spatial dynamics.
  - Maël: register data
  - Kaz: satellite data

Today

- Introduction to the course
- Lecture (10–12, 13–14) and lab (14–17) on *supervised learning*

Course is based primarily on:



Available for free: https://www.statlearning.com/

Course outline

- Day 1: ML fundamentals & introduction to supervised learning
- Day 2: Deep learning and image classification
- Day 3: Machine learning to improve causal inference

Course outline

- Day 1: ML fundamentals & introduction to supervised learning
- Day 2: Deep learning and image classification
- Day 3: Machine learning to improve causal inference

Each day

- *Lecture* before lunch: 10:00–12:00
- *Computer lab* (`R`) after lunch: 13:00–16:00

What is Machine Learning (ML)?

What is ML?

*"Machine Learning is the field of study that gives the computer the ability to **learn** without being explicitly programmed." — Arthur Samuel (1959)*

Figure 1: Arthur Samuel teaching computer to learn game of checkers

What is ML?

Traditional computer science vs. ML:

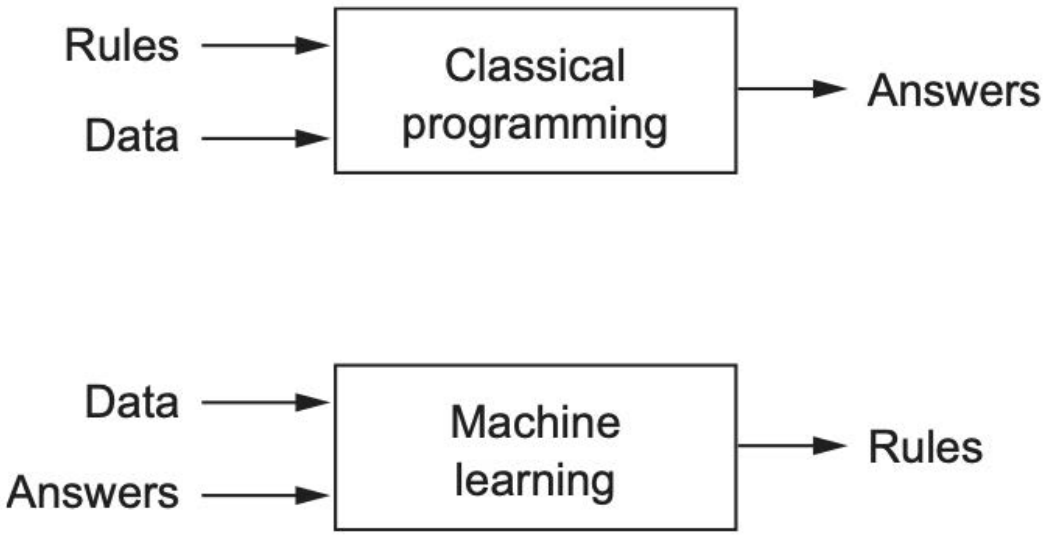


Figure 2: A new paradigm? (Chollet, 2018, Figure 1.2)

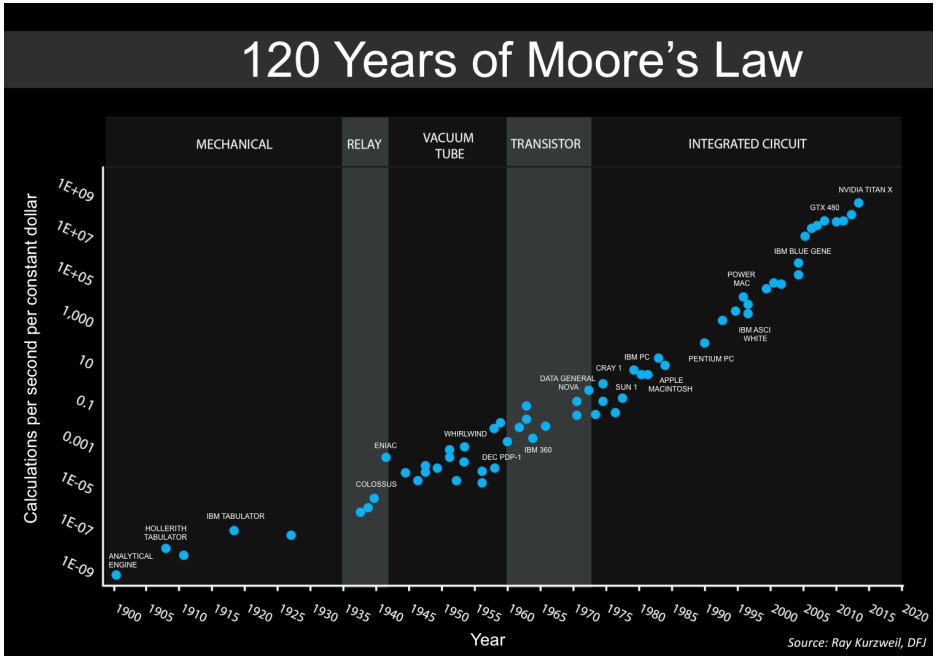
Why all the fuss **now**? Digital and computational revolution

— Big data —

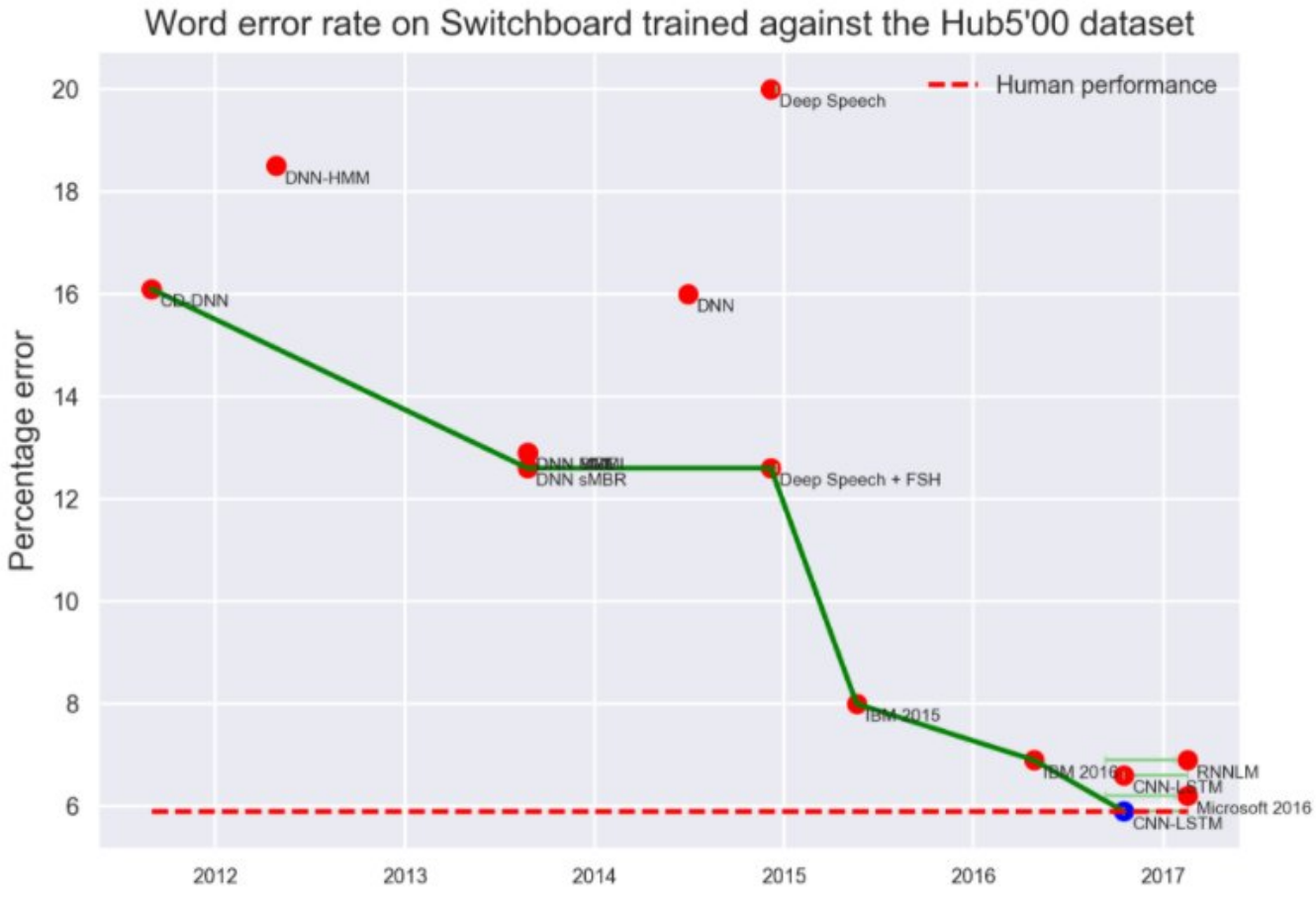
"A billion hours ago modern homo sapiens emerged. A billion minutes ago, Christianity began. A billion seconds ago, the IBM PC was released. ***A billion Google searches ago . . . was this morning****."*

— Hal Varian (2013)

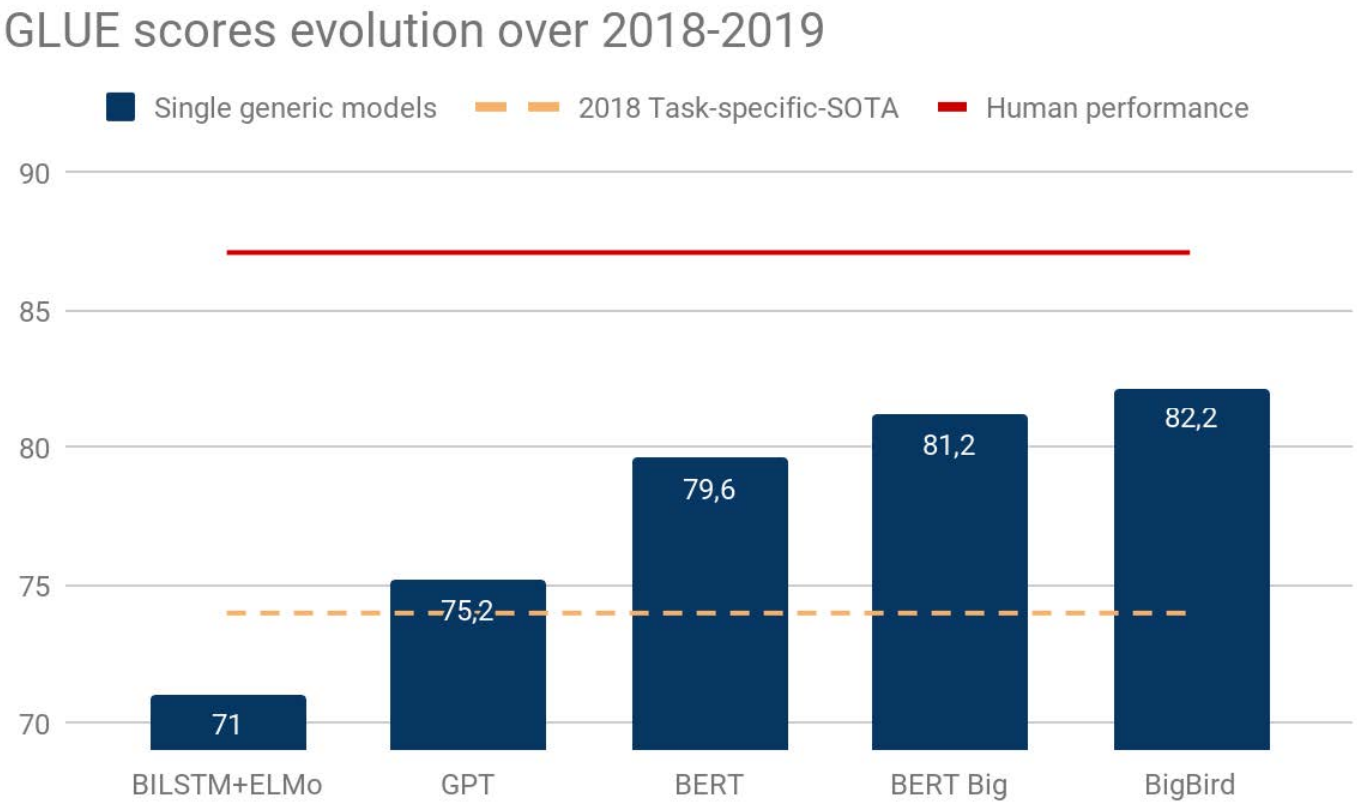
— Computational power —



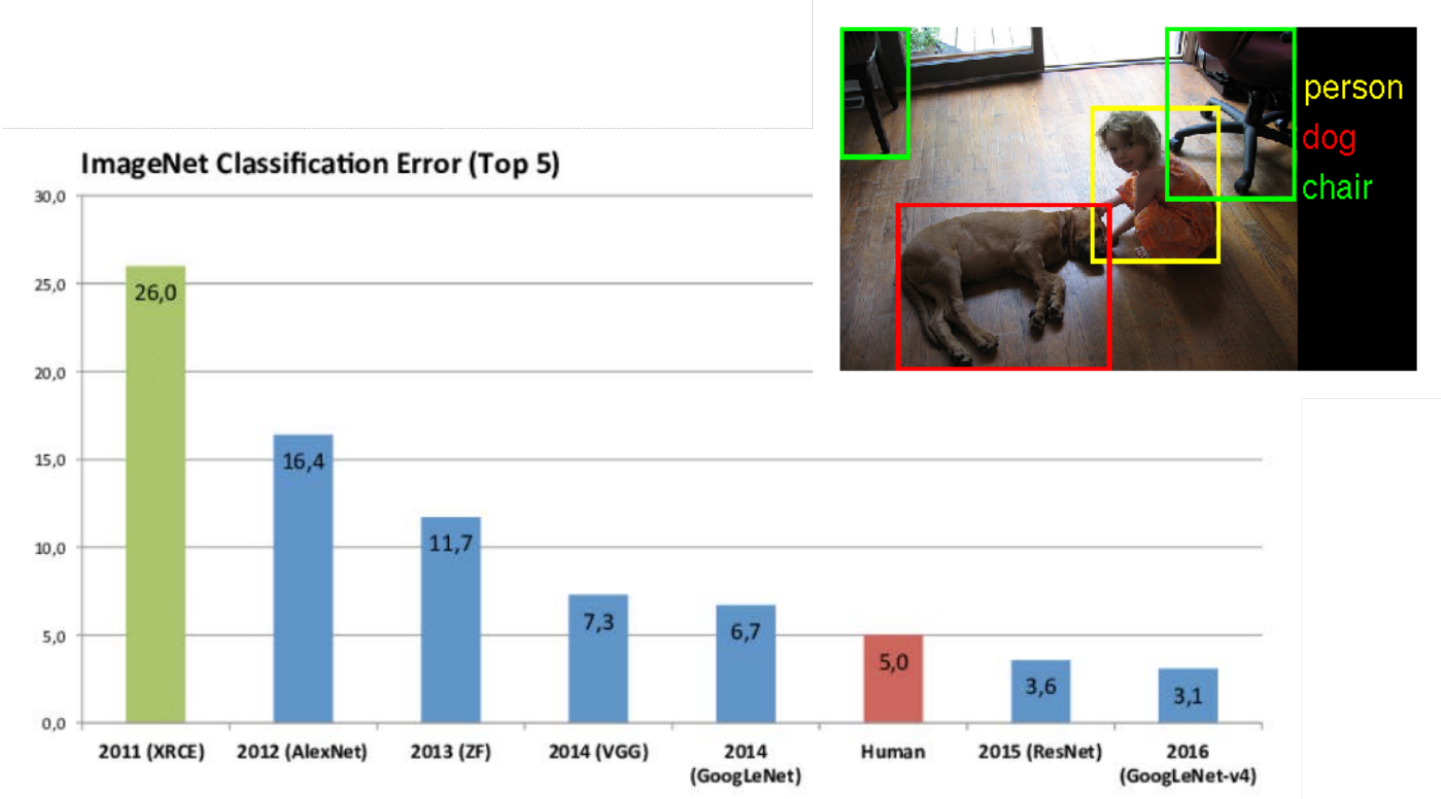
What this has enabled... **Speech recognition**



What this has enabled... **Language understanding**



What this has enabled... **Image recognition**



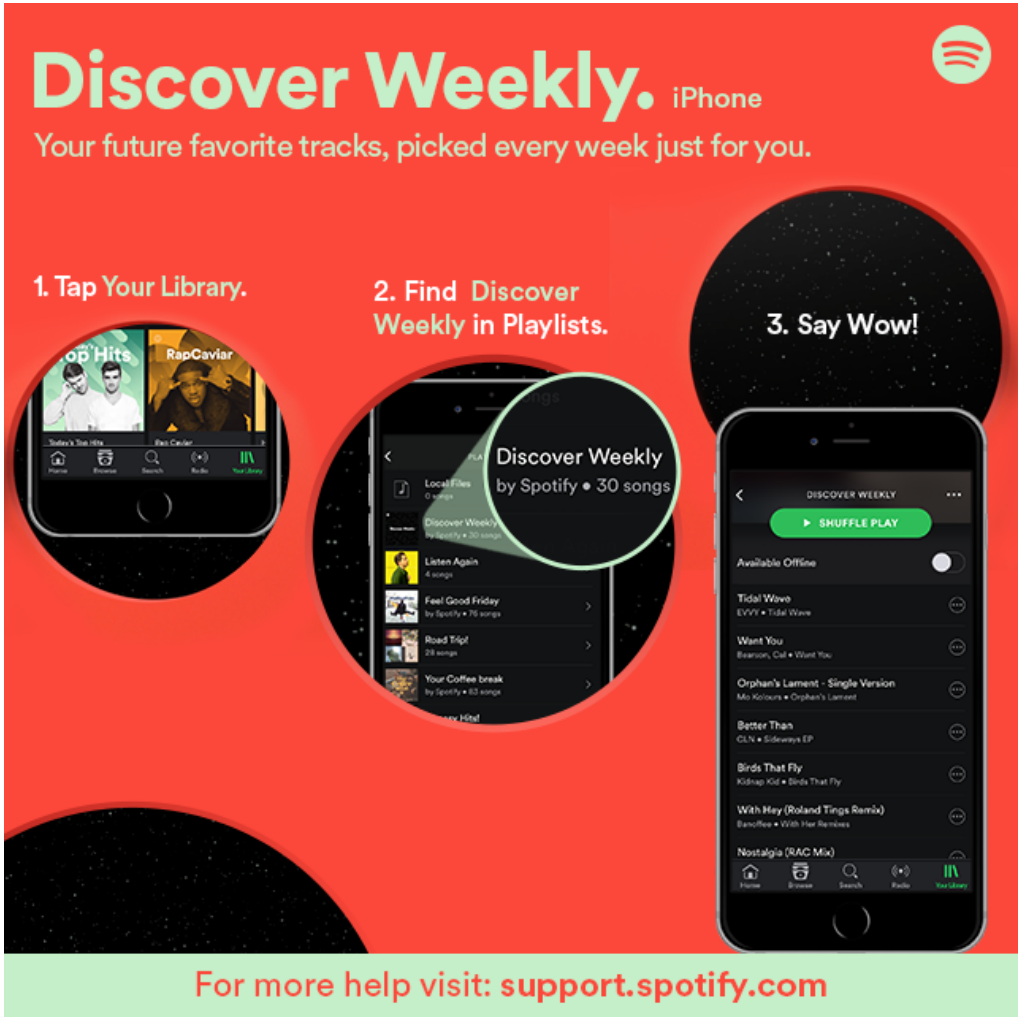
Applications. . . **Search engine**

Applications. . . **Language translation**

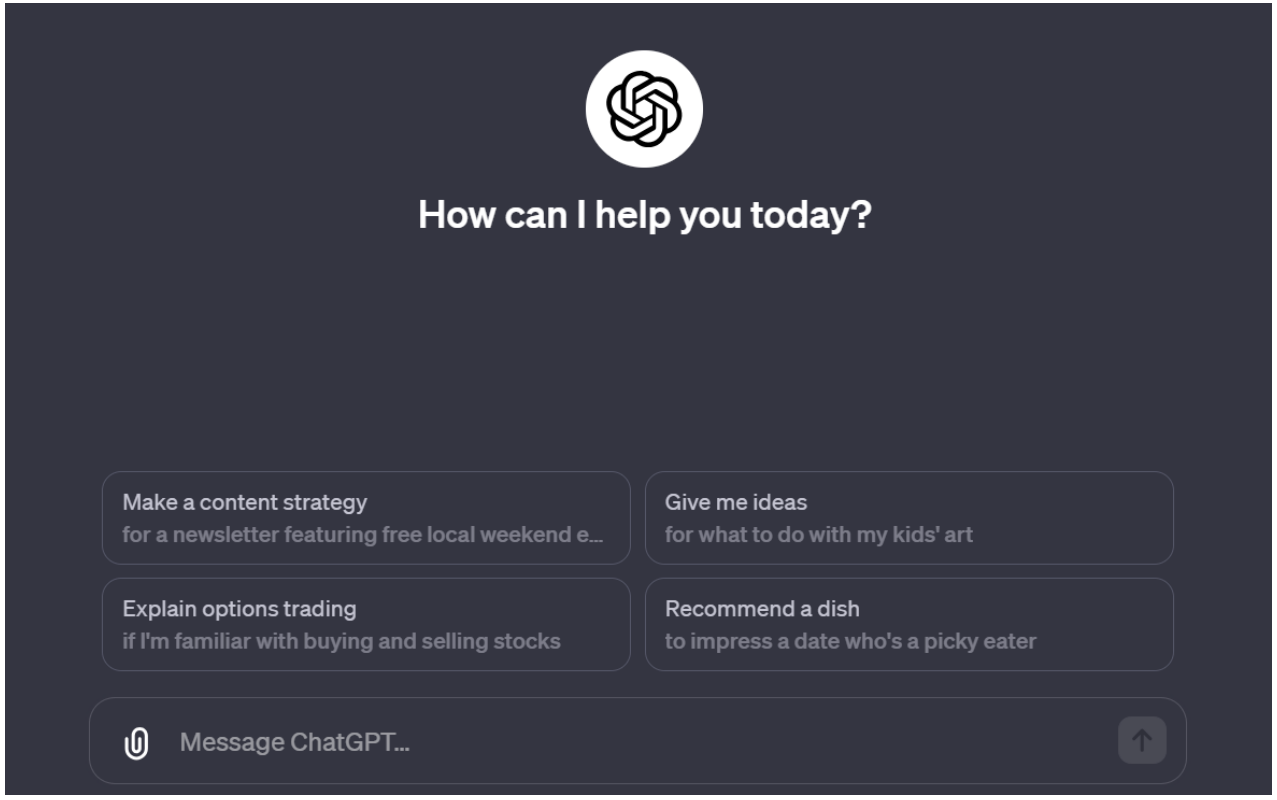
Applications. . . **Self-driving cars**

Applications. . . **''Intelligent'' virtual assistants**

Applications. . . **Recommender systems**



Applications. . . **Chat bots**



How is this relevant for the social sciences?

Some of the ''big'' data that is emerging reflects human social behaviors. ML is useful to fully exploit such data.

- **Digital traces**, e.g.,
  - Posts on Reddit and Twitter.
  - Listens on Spotify.
  - Google searches
  - ...

How is this relevant for the social sciences?

Some of the ''big'' data that is emerging reflects human social behaviors. ML is useful to fully exploit such data.

- **Digital traces**, e.g.,
  - Posts on Reddit and Twitter.
  - Listens on Spotify.
  - Google searches
  - ...
- **Digitized content**
  - Political debates
  - Newspapers (contemporary, historical archives)
  - Large-scale adminstrative registers
  - ...

How is this relevant for the social sciences?

Some of the ''big'' data that is emerging reflects human social behaviors. ML is useful to fully exploit such data.

- **Digital traces**, e.g.,
  - Posts on Reddit and Twitter.
  - Listens on Spotify.
  - Google searches
  - ...
- **Digitized content**
  - Political debates
  - Newspapers (contemporary, historical archives)
  - Large-scale adminstrative registers
  - ...
- **Image data**
  - Satellite data
  - Google street view
  - ...

Types of Machine Learning

Three major categories of ML:

- Supervised learning — prediction
- Unsupervised learning — pattern discovery
- Reinforcement learning — learn actions from rewards

Our focus today: supervised learning

Supervised learning

What is supervised learning?

- **Basic idea**: *learn* (estimate) a function $\hat{f}$ that makes predictions of some outcome Y based on data X: $\hat{f}(X) \approx Y$

What is supervised learning?

- **Basic idea**: *learn* (estimate) a function $\hat{f}$ that makes predictions of some outcome Y based on data X: $\hat{f}(X) \approx Y$
- A first supervised learning task[1]: Recognize Martin Luther King Jr.
  - X: Image
  - Y: yes/no

What is supervised learning?

- **Basic idea**: *learn* (estimate) a function $\hat{f}$ that makes predictions of some outcome Y based on data X: $\hat{f}(X) \approx Y$
- A first supervised learning task[1]: Recognize Martin Luther King Jr.
  - X: Image
  - Y: yes/no

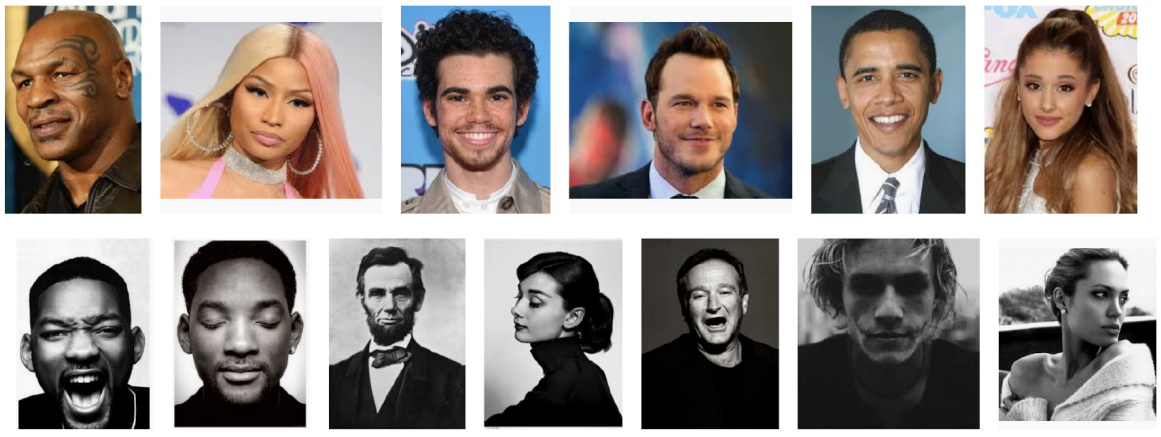
$\Rightarrow$ Learn a function $\hat{f}$ that predicts whether it is MLK/not in an image.

How do we learn such a function?

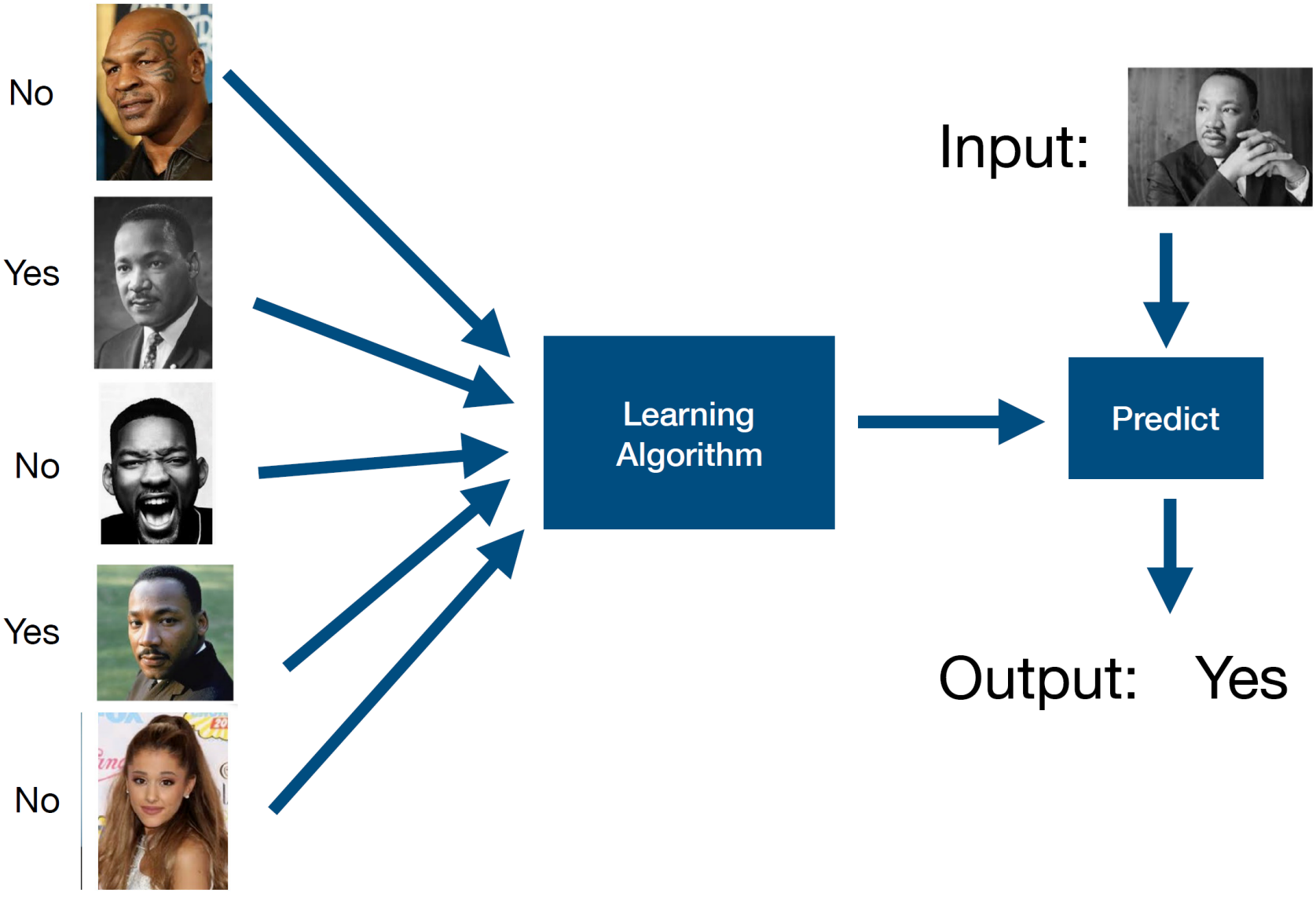
- **Basic idea**: "supervise" the computer by providing examples $\{x_i, y_i\}$

Yes

No

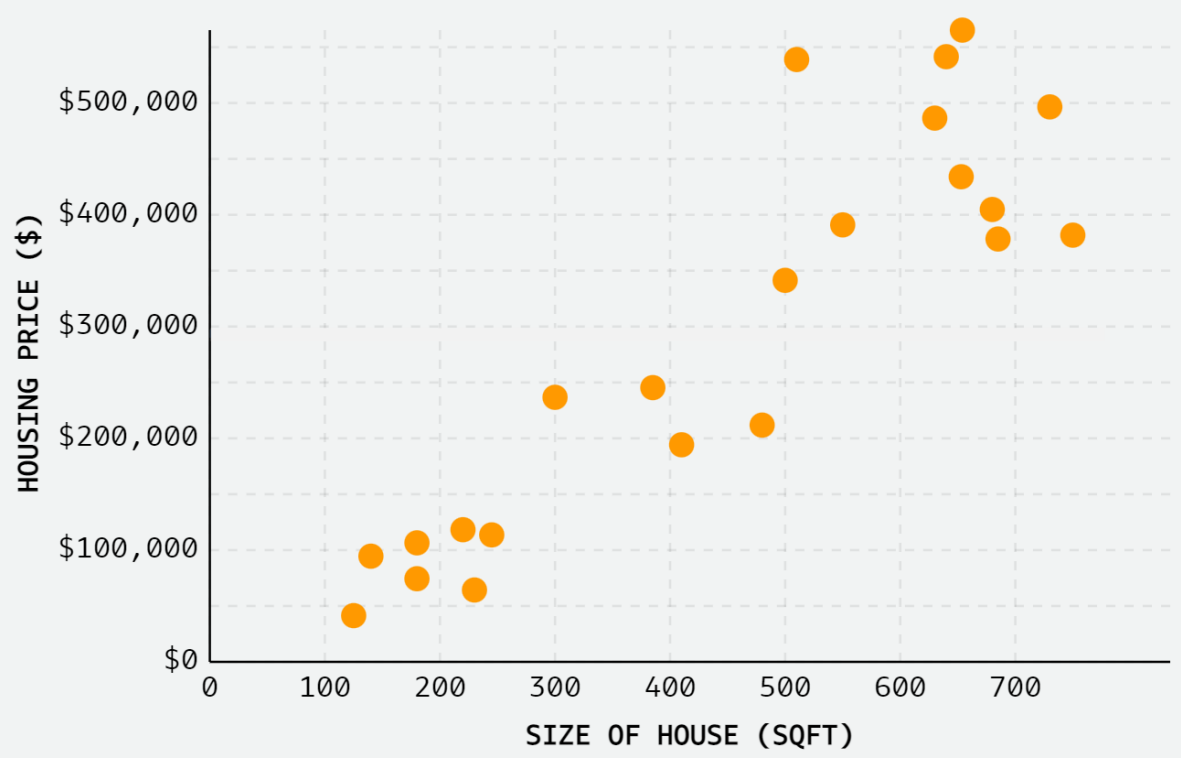


Learning from examples



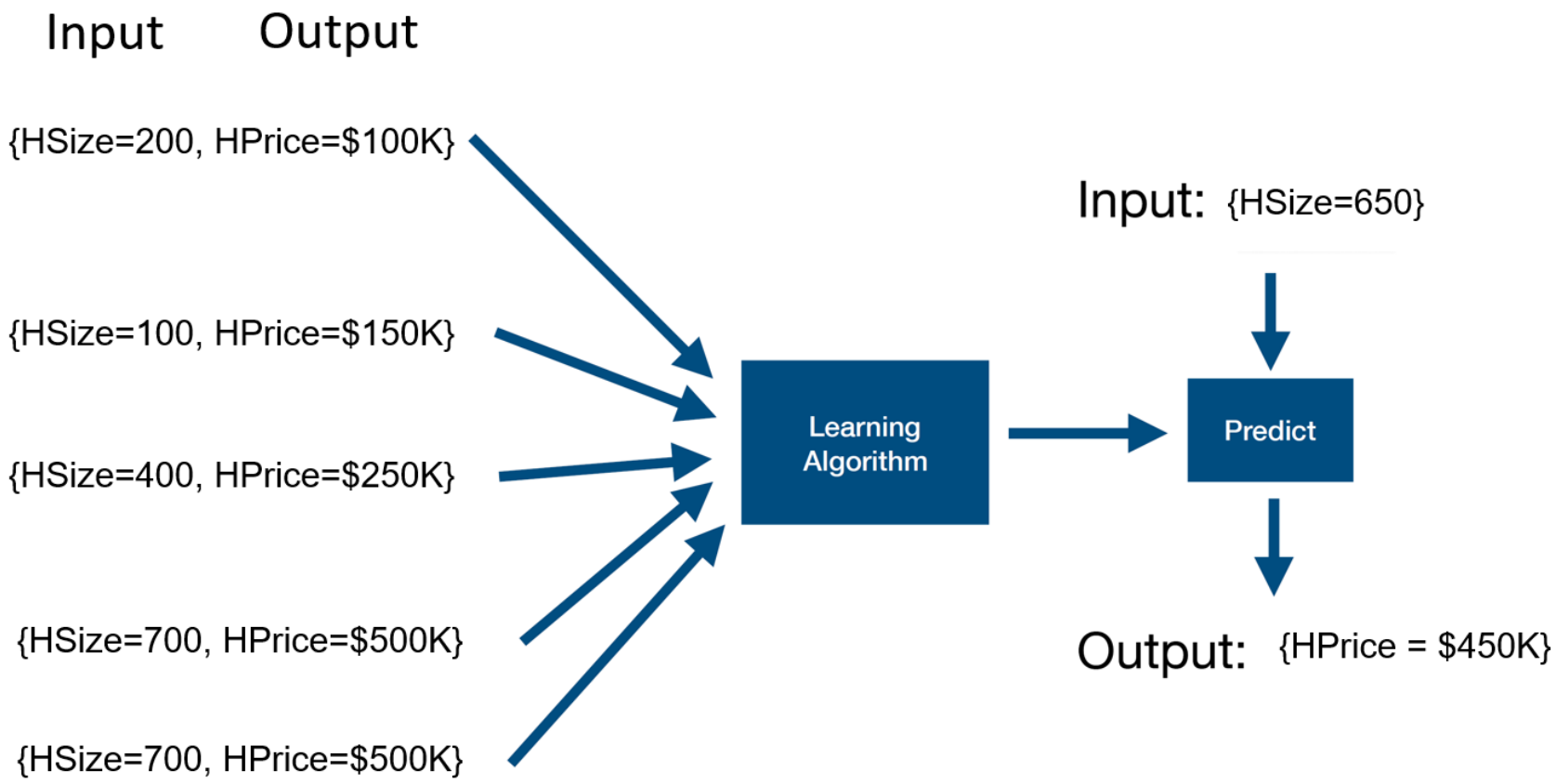
Another—more familiar—type of supervised learning task

- Predict the *price of a house* based on its *size*
  - X: Size (sqft)
  - Y: Price ($)



$\Rightarrow$ Learn a function $\hat{f}$ that predicts *house price* based on *house size*.

How? Again, we learn from examples



In more detail: How do we learn from examples?

1. Specify a "*loss function*"—quantifying how well our $\hat{f}$ fits data.
   - Popular loss function (continuous Y): *squared loss*

$$L(\hat{f}) = \sum_{i}^{N} (\underbrace{y_i}_{observed} - \underbrace{\hat{f}(x_i)}_{predicted})^2$$

$$= \sum_{i}^{N} (\underbrace{y_i}_{observed} - \underbrace{\hat{y}_i}_{predicted})^2$$

In more detail: How do we learn from examples?

1. Specify a "*loss function*"—quantifying how well our $\hat{f}$ fits data.
   - Popular loss function (continuous Y): *squared loss*

$$L(\hat{f}) = \sum_{i}^{N} (\underbrace{y_i}_{observed} - \underbrace{\hat{f}(x_i)}_{predicted})^2$$

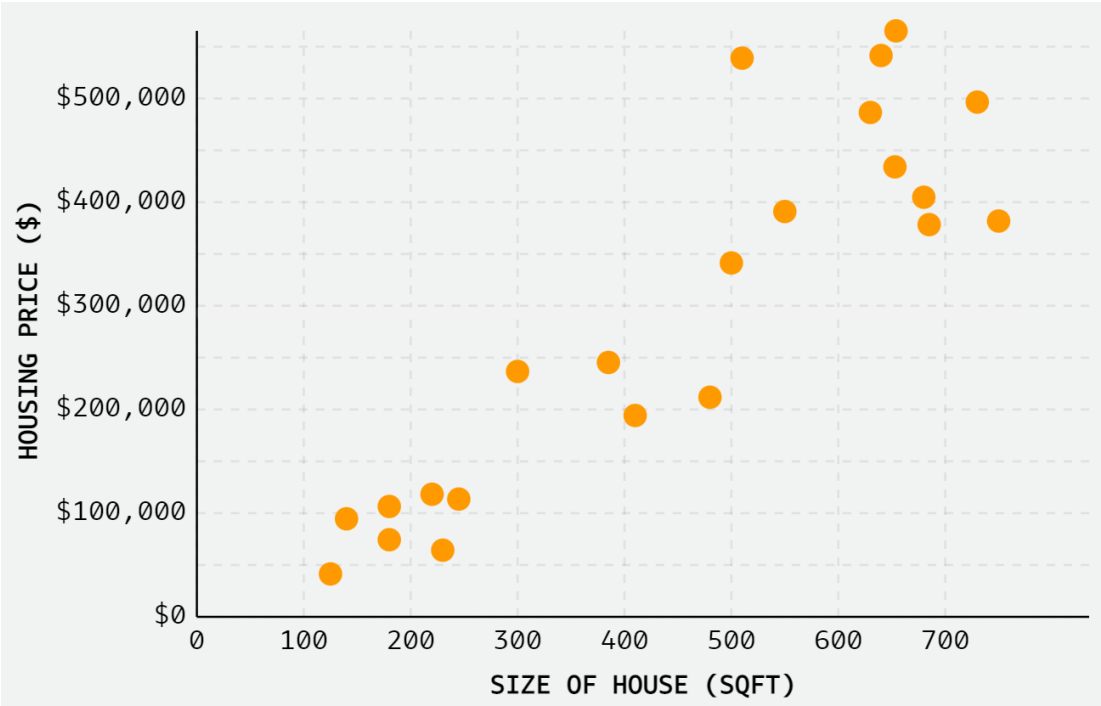
$$= \sum_{i}^{N} (\underbrace{y_i}_{observed} - \underbrace{\hat{y}_i}_{predicted})^2$$

2. Optimize $\hat{f}$ so that *loss function is minimized*.
   - I.e., minimize the distance between *prediction line* and *observations*.
   - Workhorse optimizer in ML: *gradient descent* (details out of scope).

Illustration

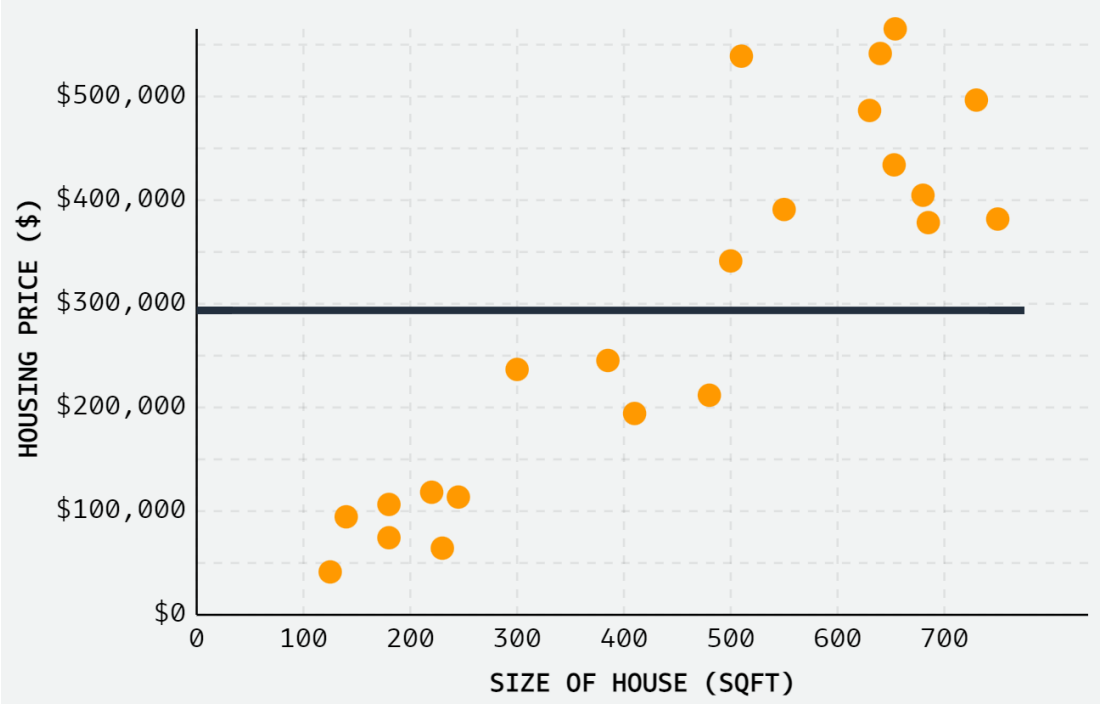
Let us return to the "house price prediction task".

- What is a good $\hat{f}$ here?
- If we were to draw a "prediction line", how would we draw it?



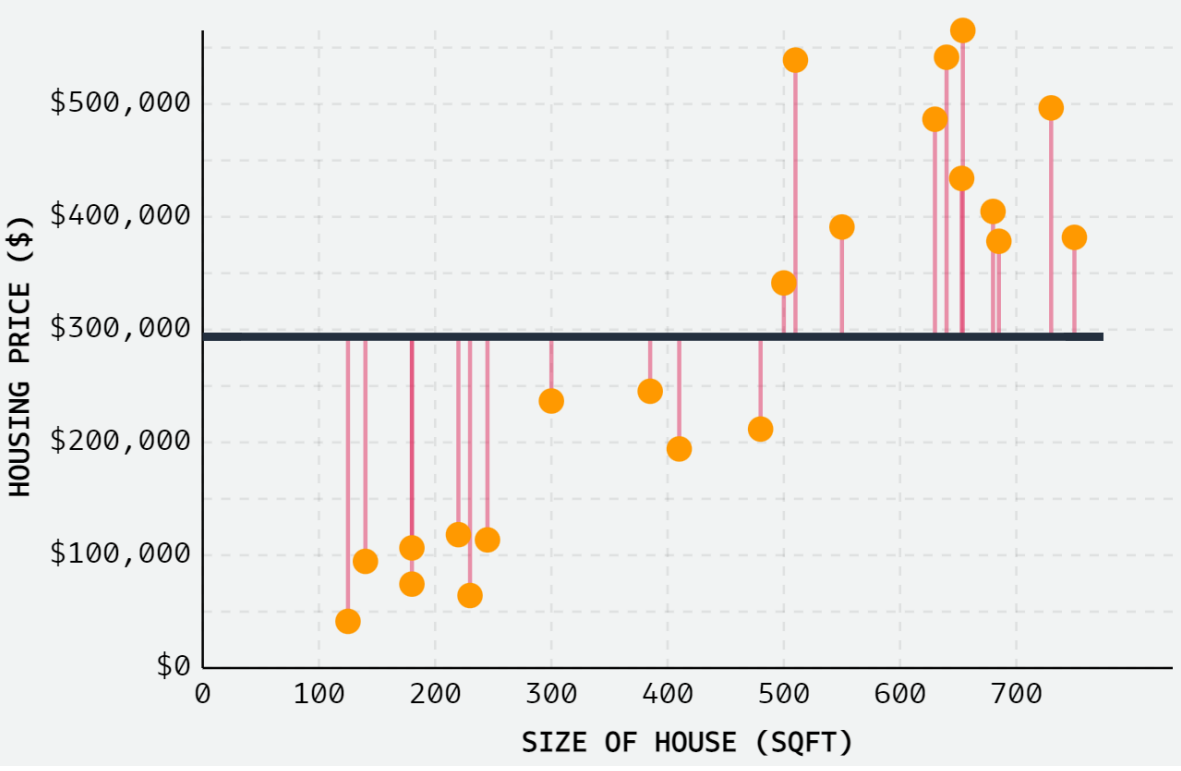
Here is one possible $\hat{f}$

- This $\hat{f}$ predicts $300,000$ for every observation.



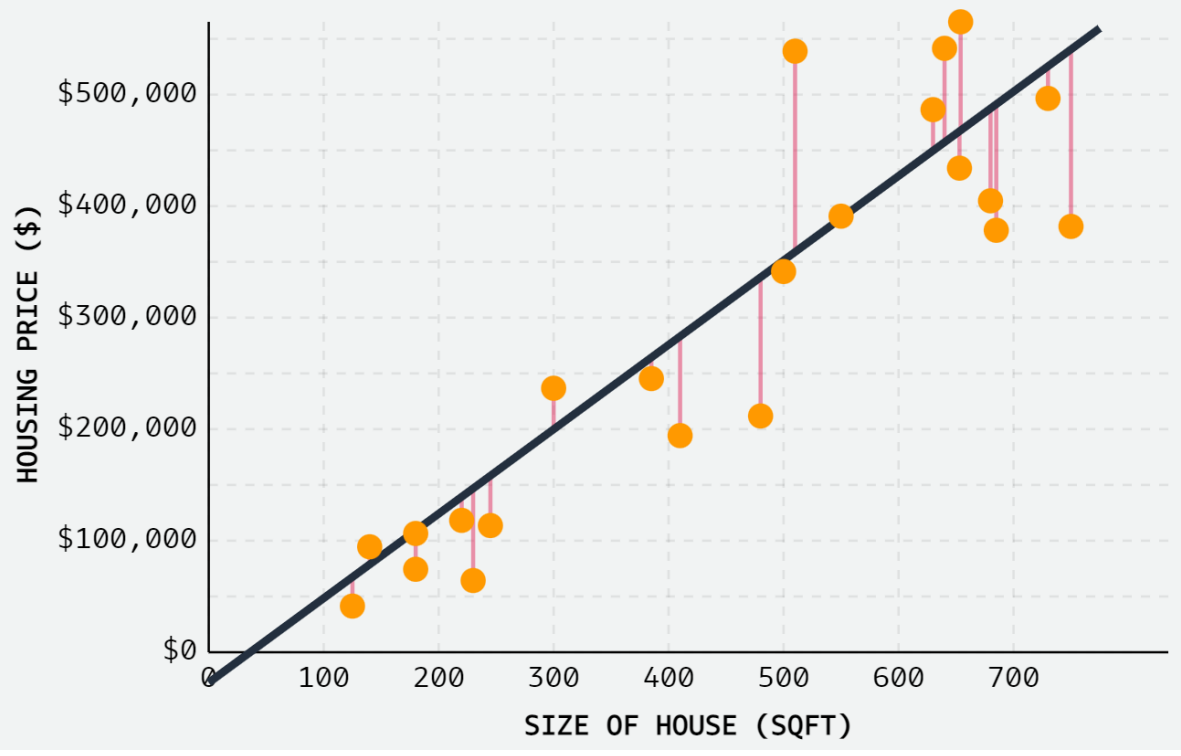
Doesn't look very good, right? How to *quantify* this intuition?

$\rightarrow$ Measure its errors: $y_i - \hat{f}(x_i)$



- Squared loss: sum of the (squared) lengths of the pink lines.
- Eye-balling: feels like we can change $\hat{f}$ (black line) to reduce the squared distances (or: pink lines).

Here is a better one:



Why is this better?

- Because the (sqaured) sum of the pink lines is much smaller.

Does this appear familiar? It should.

- Minimizing the distance between the (regression) line and the data is also what we do in...

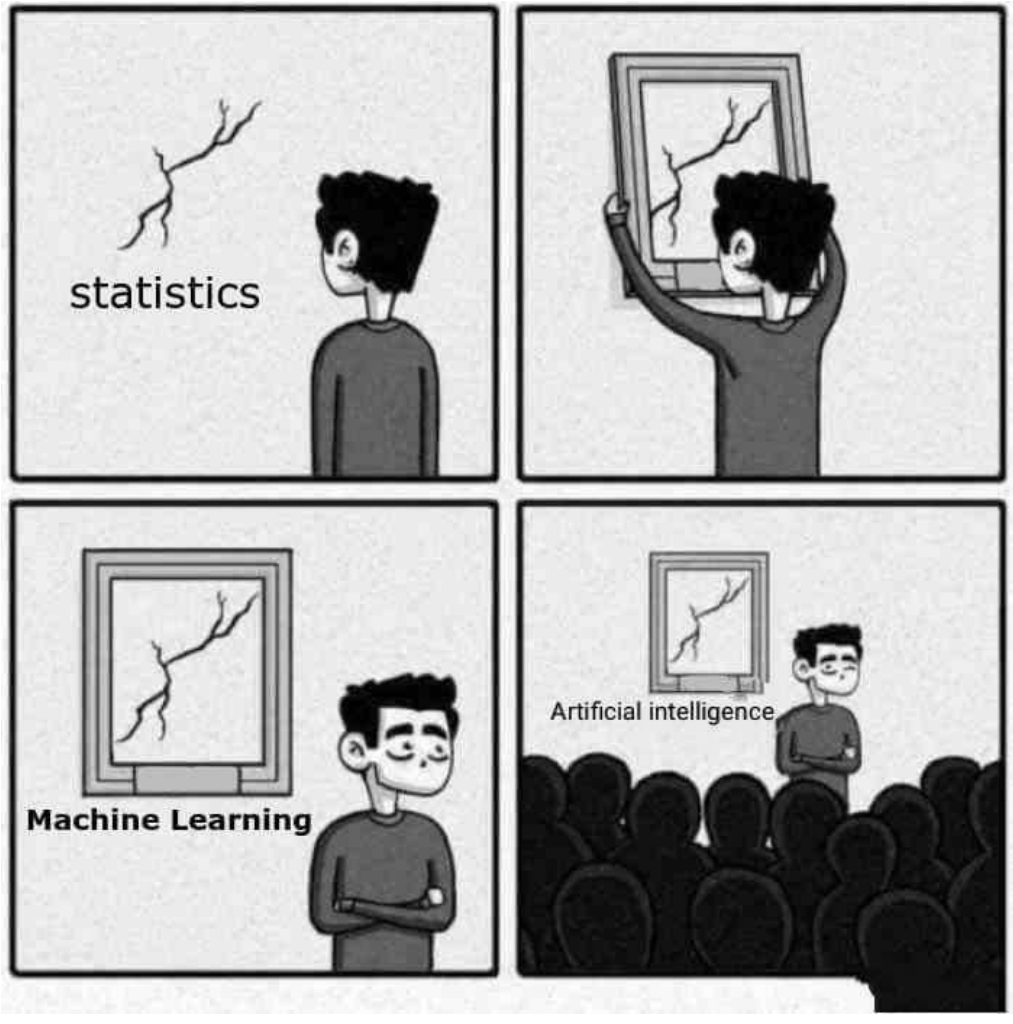
Does this appear familiar? It should.

- Minimizing the distance between the (regression) line and the data is also what we do in...

- **Ordinary least squares regression**
  - Y: continuous variable.
  - f: linear combination of X: $y = \beta_0 + \beta_1 X_1 + \ldots$
  - $L(\hat{f})$: squared loss.

So what is all the fuss about AI/ML?

. . . do we know it already?



So what is all the fuzz about AI/ML?

- While there is some truth to this meme—it is **not all just hype**.
- Linear regression can be seen an ML technique
  - Learn function $\hat{f}$ that map X to Y by minimizing a loss function
  - Indeed, it is typically found in the *beginning of ML textbooks*.

So what is all the fuzz about AI/ML?

- While there is some truth to this meme—it is **not all just hype**.
- Linear regression can be seen an ML technique
  - Learn function $\hat{f}$ that map X to Y by minimizing a loss function
  - Indeed, it is typically found in the *beginning of ML textbooks*.

$\Rightarrow$ Why should we want to *move beyond* the standard OLS?

It depends on (a) our *objective*, and (b) our *data*.

For many purposes and data sets—traditional OLS fully sufficient.

In ISL, James et al. (2013) make a useful distinction between two types of tasks in supervised learning: prediction and inference.

It depends on (a) our *objective*, and (b) our *data*.

For many purposes and data sets—traditional OLS fully sufficient.

In ISL, James et al. (2013) make a useful distinction between two types of tasks in supervised learning: prediction and inference.

If we assume the following *true* model:

$$f: \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

It depends on (a) our *objective*, and (b) our *data*.

For many purposes and data sets—traditional OLS fully sufficient.

In ISL, James et al. (2013) make a useful distinction between two types of tasks in supervised learning: prediction and inference.

If we assume the following *true* model:

$$f: \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

- **Prediction**
  - Objective: predict Y as good as possible.
  - $\hat{f}$ often treated as a black box.
  - That is, an emphasis on $\hat{Y}$.

It depends on (a) our *objective*, and (b) our *data*.

For many purposes and data sets—traditional OLS fully sufficient.

In ISL, James et al. (2013) make a useful distinction between two types of tasks in supervised learning: prediction and inference.

If we assume the following *true* model:

$$f: \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

- **Prediction**
  - Objective: predict Y as good as possible.
  - $\hat{f}$ often treated as a black box.
  - That is, an emphasis on $\hat{Y}$.
- **Inference**
  - Objective: learn about how X is associated with Y.
  - That is, an emphasis on $\hat{\beta}$

For the purpose of **inference**...

Traditional statistical toolkit (e.g., OLS) has a lot of attractive features—when their assumptions about f holds:

- Interpretable: 1-unit change in X associated with β change in Y

For the purpose of **inference**...

Traditional statistical toolkit (e.g., OLS) has a lot of attractive features—when their assumptions about f holds:

- Interpretable: 1-unit change in X associated with β change in Y
- Theoretical standard errors and p-values: significance testing.

For the purpose of **inference**...

Traditional statistical toolkit (e.g., OLS) has a lot of attractive features—when their assumptions about f holds:

- Interpretable: 1-unit change in X associated with β change in Y
- Theoretical standard errors and p-values: significance testing.
- Unbiased: Upon repeated samples, average of $\hat{\beta}$ will converge to true population β.

But for the purpose of **prediction**...

Standard linear paradigm (OLS) has certain limitations:

But for the purpose of **prediction**...

Standard linear paradigm (OLS) has certain limitations:

- Assumes **linearity**
  - $Y = \beta_1 X_1 \ldots$: One unit change in $X_1 \rightarrow \beta_1$ change in Y.

But for the purpose of **prediction**...

Standard linear paradigm (OLS) has certain limitations:

- Assumes **linearity**
  - $Y = \beta_1 X_1 \ldots$: One unit change in $X_1 \rightarrow \beta_1$ change in Y.
- Assumes **additivity**
  - $Y = \beta_1 X_1 + \beta_2 X_2$: Contribution of X_1 and X_2 *added* to predict Y.

But for the purpose of **prediction**...

Standard linear paradigm (OLS) has certain limitations:

- Assumes **linearity**
  - $Y = \beta_1 X_1 \ldots$: One unit change in $X_1 \rightarrow \beta_1$ change in Y.
- Assumes **additivity**
  - $Y = \beta_1 X_1 + \beta_2 X_2$: Contribution of X_1 and X_2 *added* to predict Y.
- Only really works when no. obs $>>$ no. columns.

But for the purpose of **prediction**...

Standard linear paradigm (OLS) has certain limitations:

- Assumes **linearity**
  - $Y = \beta_1 X_1 \ldots$: One unit change in $X_1 \rightarrow \beta_1$ change in Y.
- Assumes **additivity**
  - $Y = \beta_1 X_1 + \beta_2 X_2$: Contribution of X_1 and X_2 *added* to predict Y.
- Only really works when no. obs $>>$ no. columns.
- Assumes we know *a priori* **which variables $X_1, X_2 \ldots$ to include**.

But for the purpose of **prediction**...

Standard linear paradigm (OLS) has certain limitations:

- Assumes **linearity**
  - $Y = \beta_1 X_1 \ldots$: One unit change in $X_1 \rightarrow \beta_1$ change in Y.
- Assumes **additivity**
  - $Y = \beta_1 X_1 + \beta_2 X_2$: Contribution of X_1 and X_2 *added* to predict Y.
- Only really works when no. obs $>>$ no. columns.
- Assumes we know *a priori* **which variables $X_1, X_2 \ldots$ to include**.

Thus, if:

a. Mapping between X–Y is complex,
b. X has many columns, and
c. We have little knowledge about which variables are relevant or how they are related to Y...

$\Rightarrow$ standard linear paradigm (OLS) struggles to make good predictions.

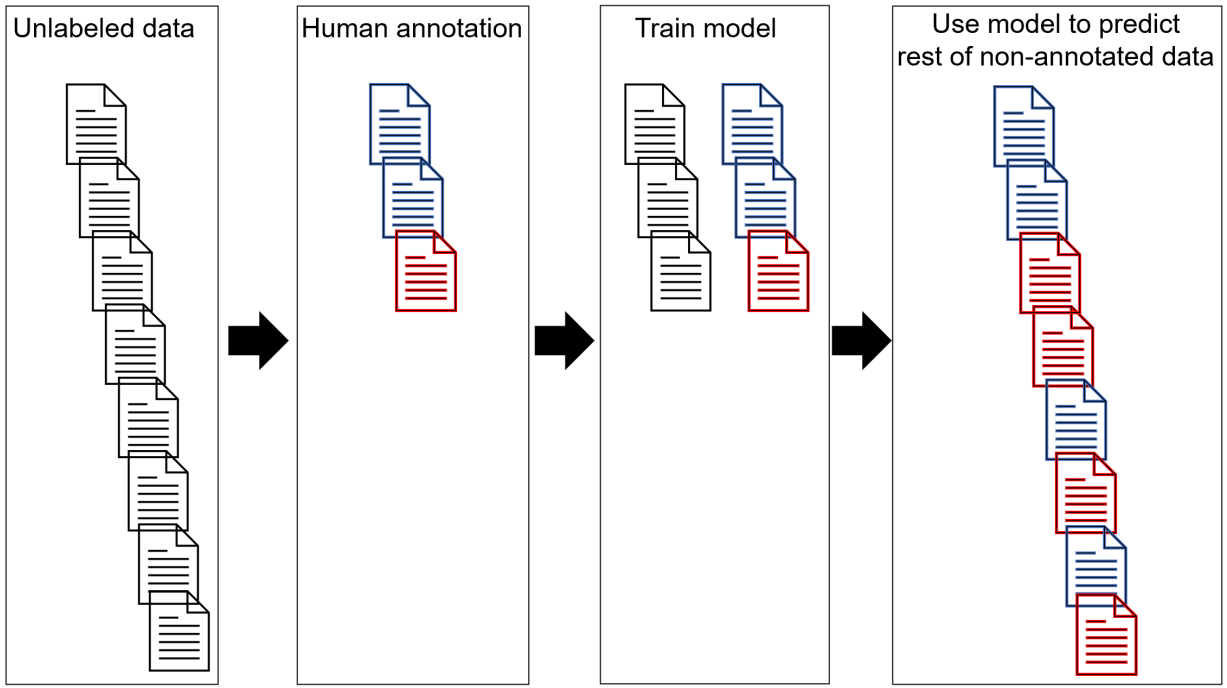
Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

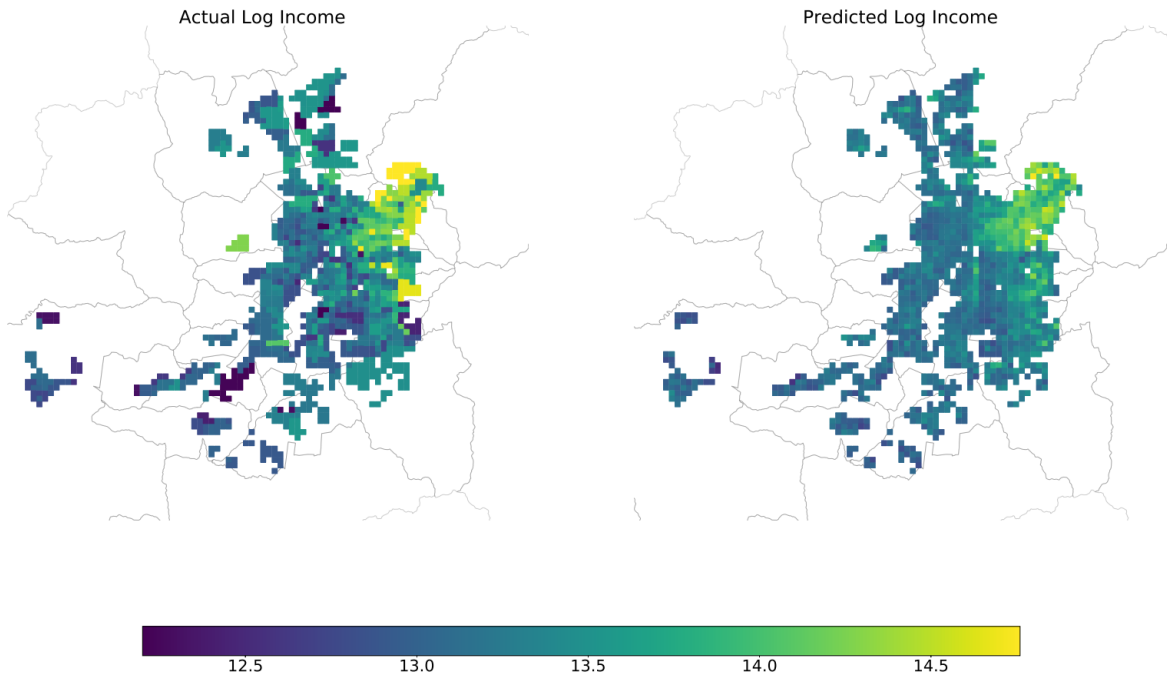
- **Measurement**
  - Scale up human labeling. Human codes some docs, then "machine" learns to code like a human, and repeats for millions of docs.



Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

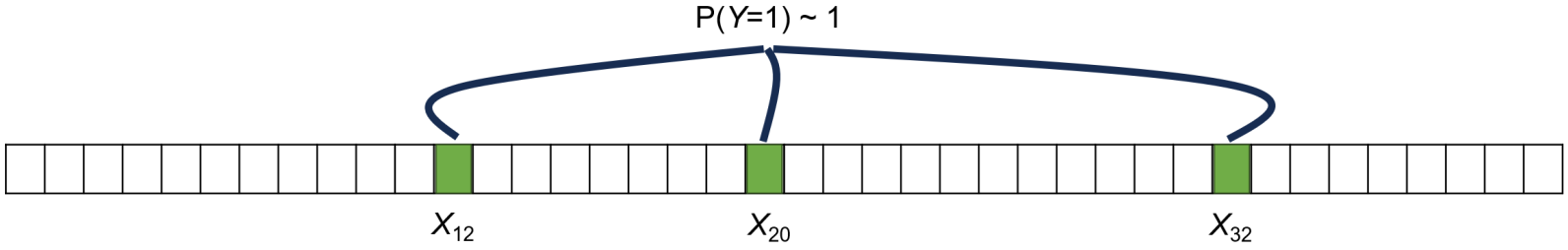
- **Measurement**
  - Use *satelite data* to predict *economic variables* in geographic regions where such data is lacking (e.g., Piaggesi et al. 2019).

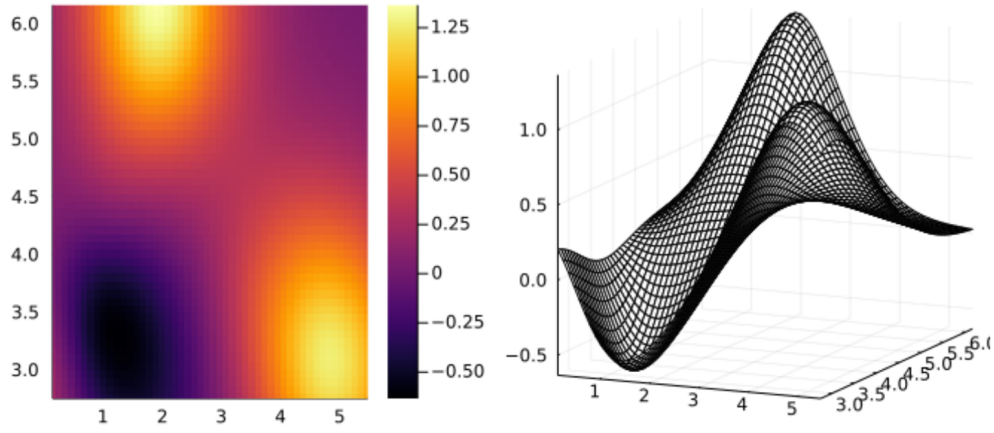


Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

- **Discovery**
  - Identify new predictive relationships/interactions.

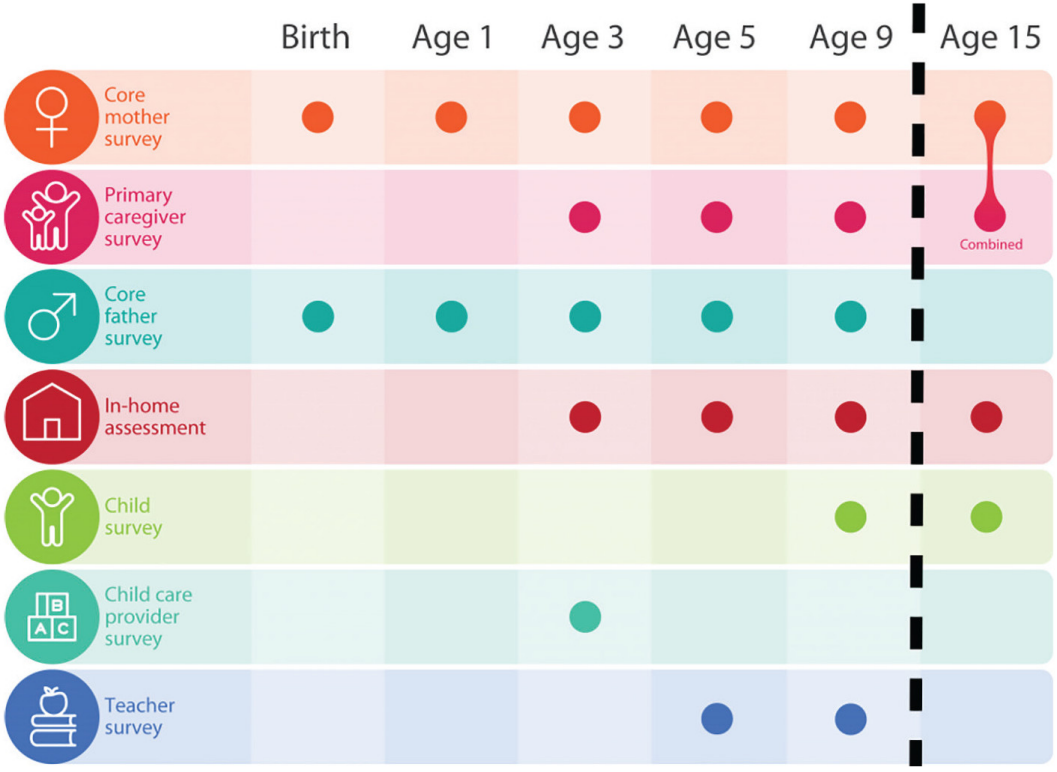




Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

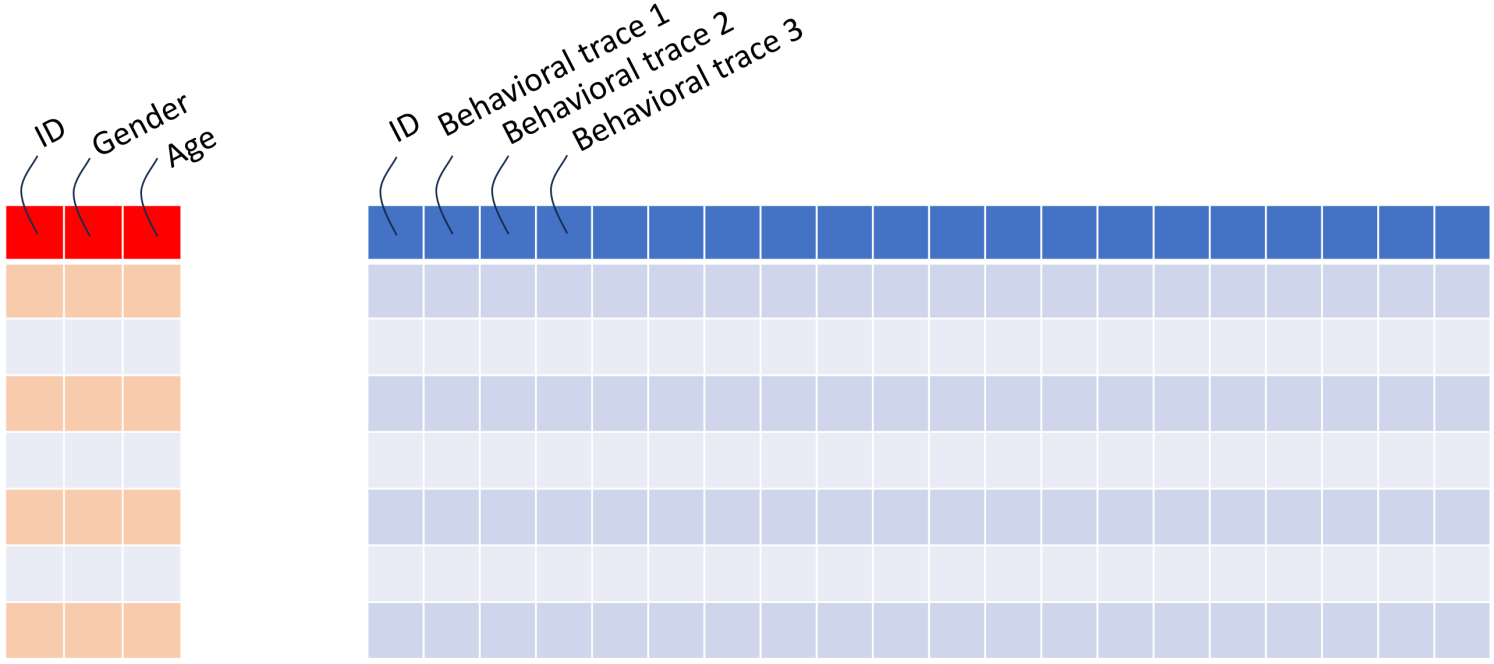
- **Prediction as an end in itself**
  - E.g., early warning prediction of fragile families (Salganik et al. 2020)



Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

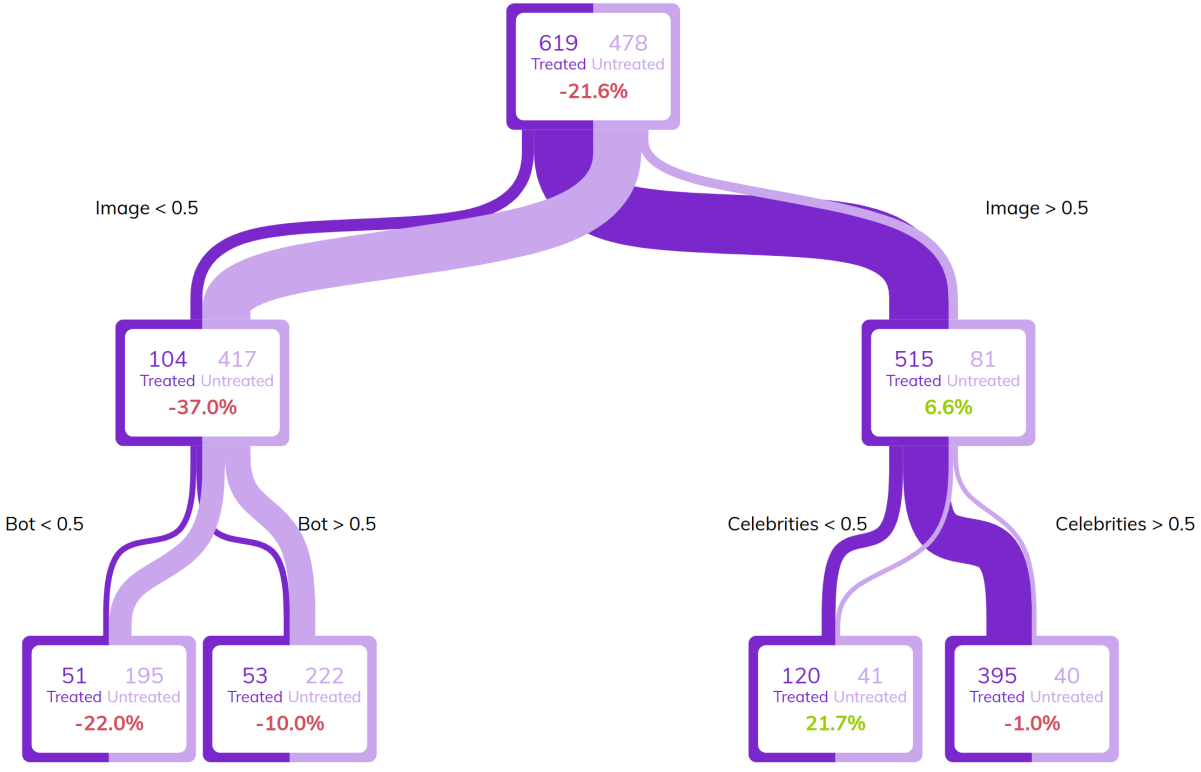
- **Facilitating causal inference**
  - When data is *non-linear* or contain *many variables* (e.g., Eckles and Bakshy 2021)



Ok, but why should we care about prediction?

ML still recent — not yet fully answered. Examples of good use cases:

- **Facilitating causal inference**
  - Discovering *heterogeneous treatment effects* (e.g., Wager and Athey 2018)



Model evaluation

Model evaluation

Before elaborating on why complex mappings between X–Y and high-dimensional X are difficult for the traditional toolkit we must first be a bit more precise about:

- what we mean when we say $\hat{f}$ is a **good** at predicting Y, and
- how we quantify "prediction performance".

So, what characterizes an $\hat{f}$ that is good a *prediction*?

In ML, a model is considered to be *good at prediction* if it can predict *accurately* on **data which it has not seen before**.

So, what characterizes an $\hat{f}$ that is good a *prediction*?

In ML, a model is considered to be *good at prediction* if it can predict *accurately* on **data which it has not seen before**.

Reason for the emphasis on *new/unseen data*:

So, what characterizes an $\hat{f}$ that is good a *prediction*?

In ML, a model is considered to be *good at prediction* if it can predict *accurately* on **data which it has not seen before**.

Reason for the emphasis on *new/unseen data*:

- **Usefulness**: if prediction is the goal, we want to be able to use our $\hat{f}$ to predict things we don't yet have the answers to.

So, what characterizes an $\hat{f}$ that is good a *prediction*?

In ML, a model is considered to be *good at prediction* if it can predict *accurately* on **data which it has not seen before**.

Reason for the emphasis on *new/unseen data*:

- **Usefulness**: if prediction is the goal, we want to be able to use our $\hat{f}$ to predict things we don't yet have the answers to.
- The **prediction error** obtained on the data it was trained (X_{train}) is often *not a good proxy* for how well it predicts in general (X_{test}).
  - That is, often: $error(X_{train}) \neq error(X_{test})$.

How do we quantify prediction performance?

- Before elaborating on why $error(X_{train}) \neq error(X_{test})\ldots$ how can we *quantify* these errors?

How do we quantify prediction performance?

- Before elaborating on why $error(X_{train}) \neq error(X_{test})\ldots$ how can we *quantify* these errors?
- For a *continuous* Y, the most common way to measure how well a model fits one's data is the **mean squared error** (MSE):

How do we quantify prediction performance?

- Before elaborating on why $error(X_{train}) \neq error(X_{test})\ldots$ how can we *quantify* these errors?
- For a *continuous* Y, the most common way to measure how well a model fits one's data is the **mean squared error** (MSE):
- Basically the **squared loss** — just that we *take the average*.

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2$$

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

How do we quantify prediction performance?

- Before elaborating on why $error(X_{train}) \neq error(X_{test})\ldots$ how can we *quantify* these errors?
- For a *continuous* Y, the most common way to measure how well a model fits one's data is the **mean squared error** (MSE):
- Basically the **squared loss** — just that we *take the average*.

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2$$

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

- Facilitates comparison btw samples of different sizes (eg X_{train},X_{test})

Why often: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Now that we know h.t. *quantify errors*. Why: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Why often: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Now that we know h.t. *quantify errors*. Why: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Because of something called **overfitting**

Why often: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Now that we know h.t. *quantify errors*. Why: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Because of something called **overfitting**

- This happens when $\hat{f}$ becomes so flexible that it models "noise" in the data instead of approximating the true f.

Why often: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Now that we know h.t. *quantify errors*. Why: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Because of something called **overfitting**

- This happens when $\hat{f}$ becomes so flexible that it models "noise" in the data instead of approximating the true f.
  - Flexibility $\approx$ number of parameters of $\hat{f}$.

Why often: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Now that we know h.t. *quantify errors*. Why: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Because of something called **overfitting**

- This happens when $\hat{f}$ becomes so flexible that it models "noise" in the data instead of approximating the true f.
  - Flexibility $\approx$ number of parameters of $\hat{f}$.
- *Consequently*:
  - On the data it was trained, it fits data very closely.
  - But on new data—because it captured random noise specific to the training data—it will not predict well.

Why often: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

Now that we know h.t. *quantify errors*. Why: $\text{MSE}(X_{train}) \neq \text{MSE}(X_{test})$?

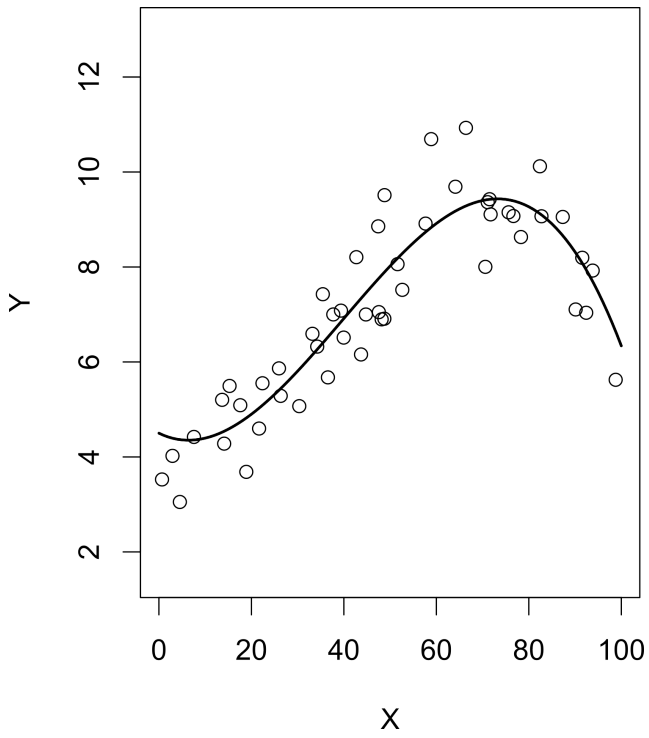
Because of something called **overfitting**

- This happens when $\hat{f}$ becomes so flexible that it models "noise" in the data instead of approximating the true f.
  - Flexibility $\approx$ number of parameters of $\hat{f}$.
- *Consequently*:
  - On the data it was trained, it fits data very closely.
  - But on new data—because it captured random noise specific to the training data—it will not predict well.
- To get some intuition for this, let's consider *some examples*.

Illustration

Consider example from ISL (Figure 2.9, page 31)

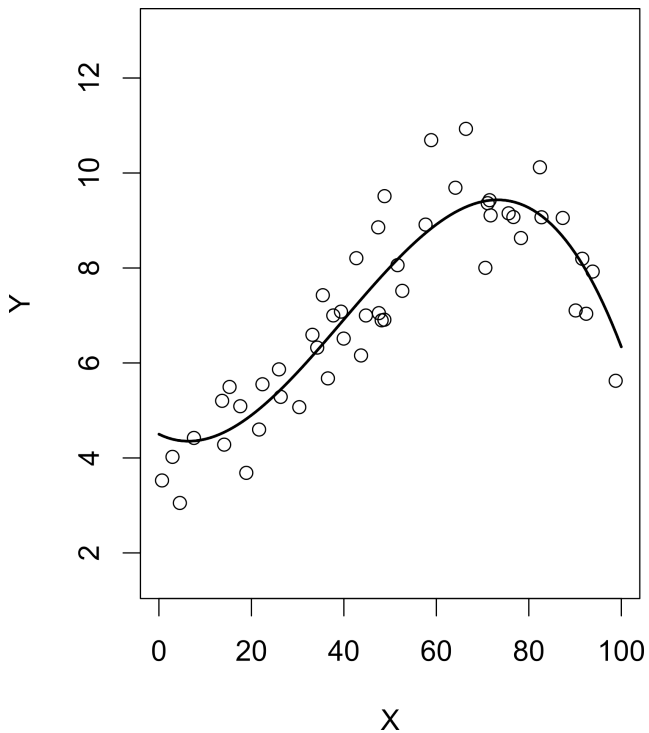
- *True f* represented by **black line**.
- A sample of 50 data points generated from *f* (empty circles).
- The sum of the distance between the two = *the irreducible error*.



Illustration

Consider example from ISL (Figure 2.9, page 31)

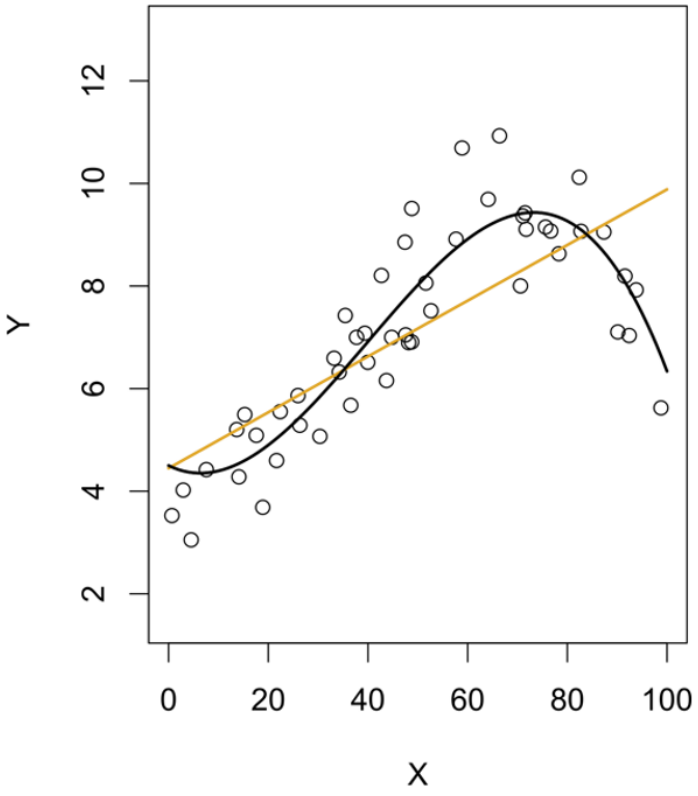
- *True f* represented by **black line**.
- A sample of 50 data points generated from f (empty circles).
- The sum of the distance between the two = *the irreducible error*.

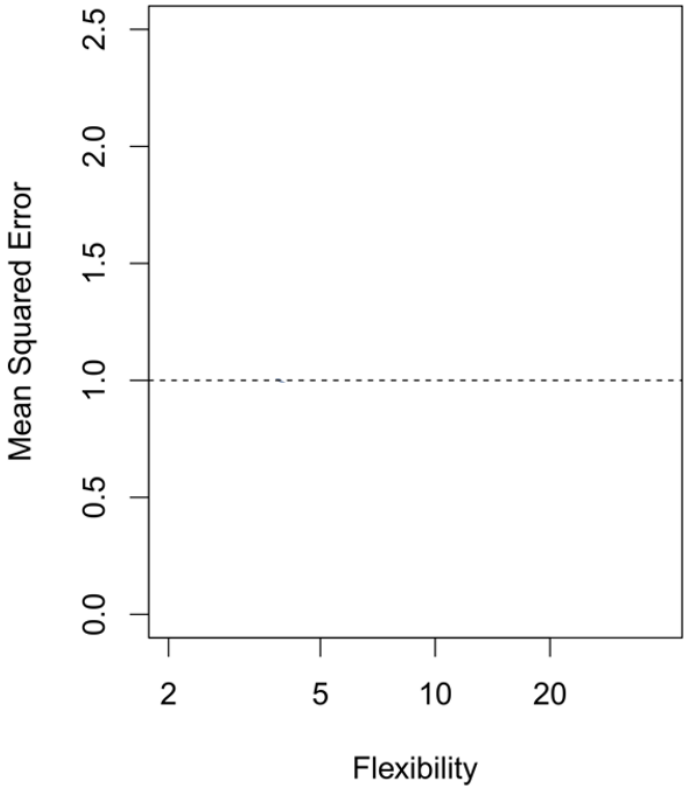


Now, let's estimate three different $\hat{f}$ (of differing flexibility) based on this sample—and measure both MSE(X_{train}) and MSE(X_{test}).

A simple $\hat{f}$ (linear)

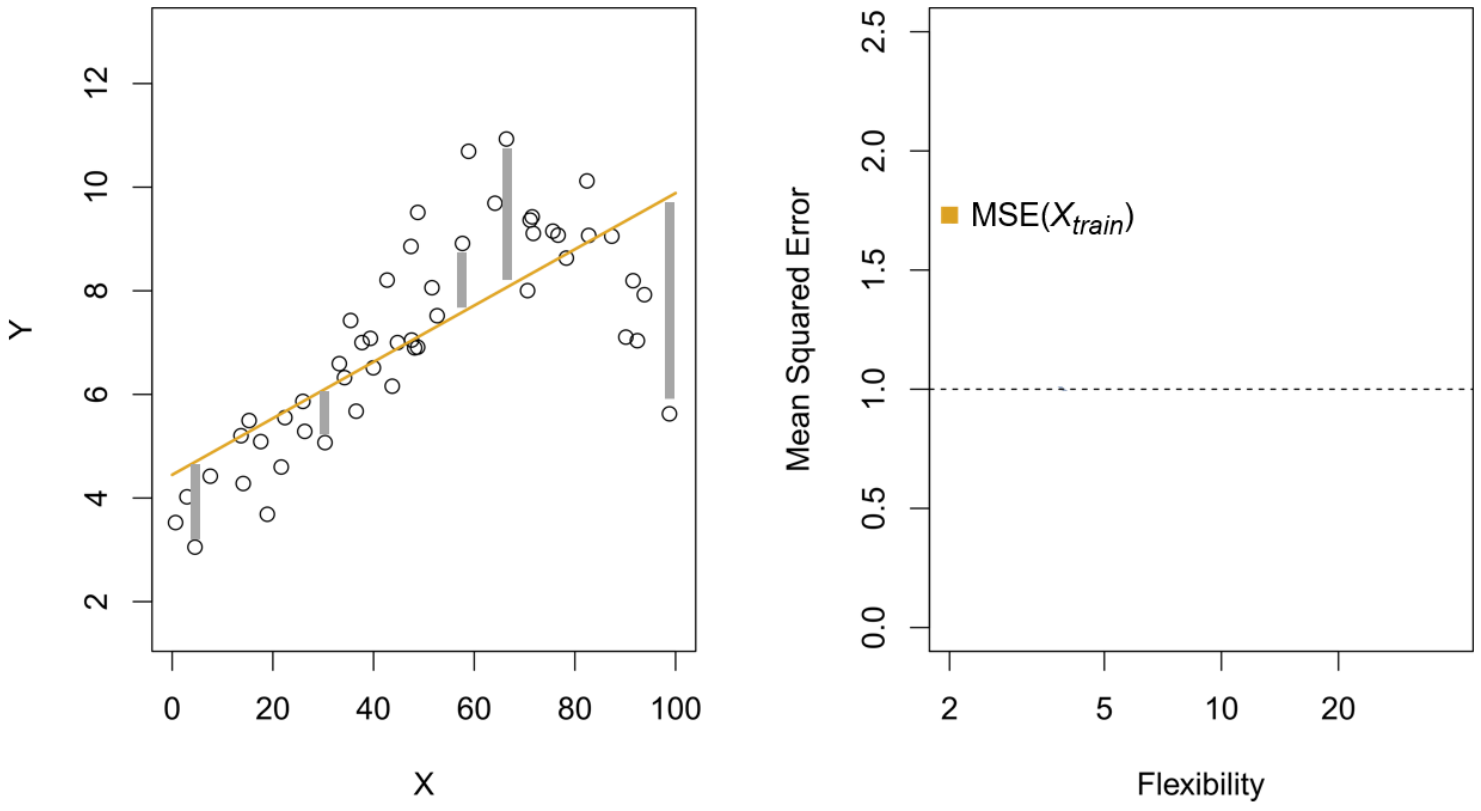
- Prediction line from $\hat{f}_{simple}$ represented by orange line.





A simple $\hat{f}$ (linear)

- Prediction line from $\hat{f}_{simple}$ represented by orange line.

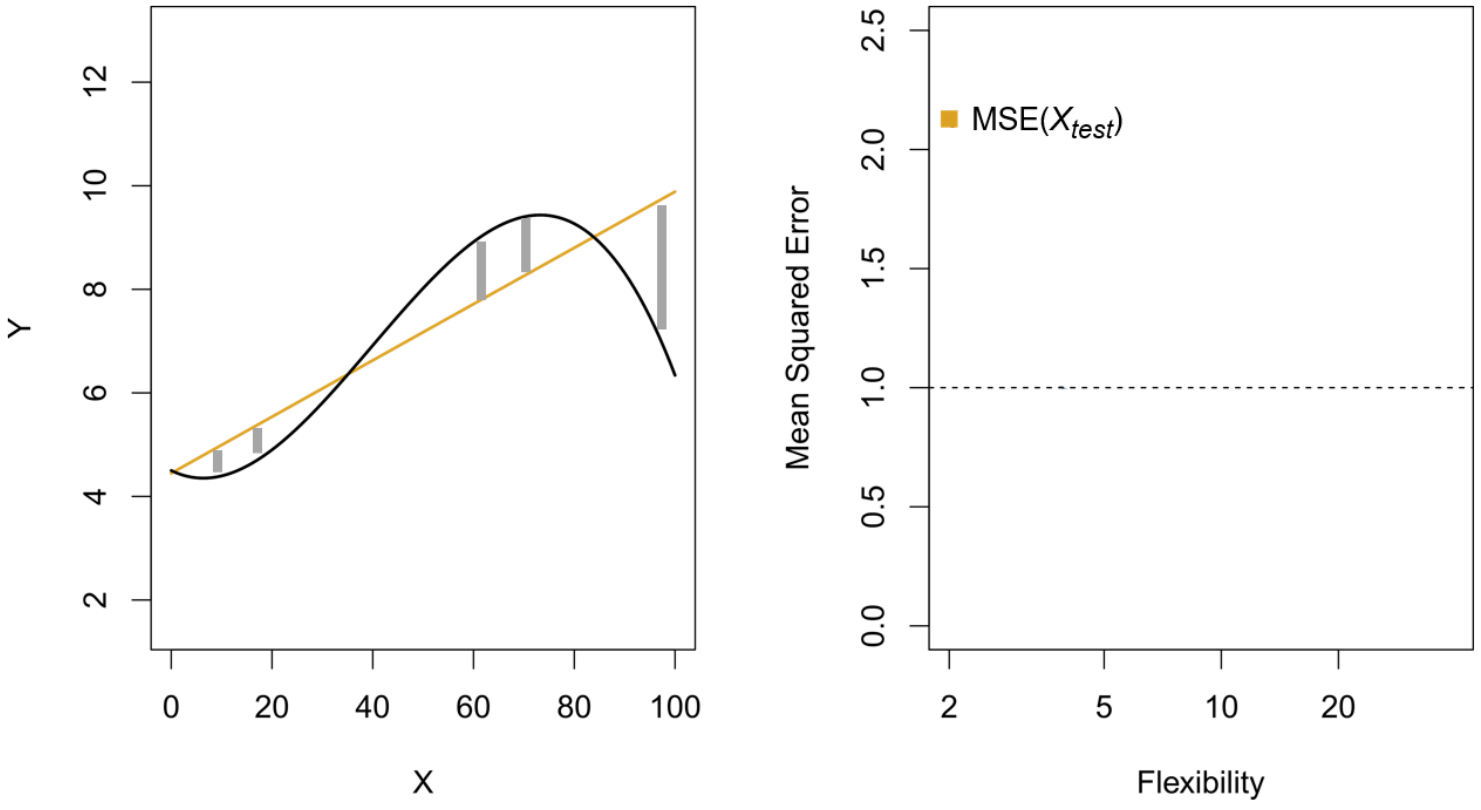


Measuring errors:

- Distance between orange line and *empty circles* = MSE(X_{train}).

A simple $\hat{f}$ (linear)

- Prediction line from $\hat{f}_{simple}$ represented by orange line.

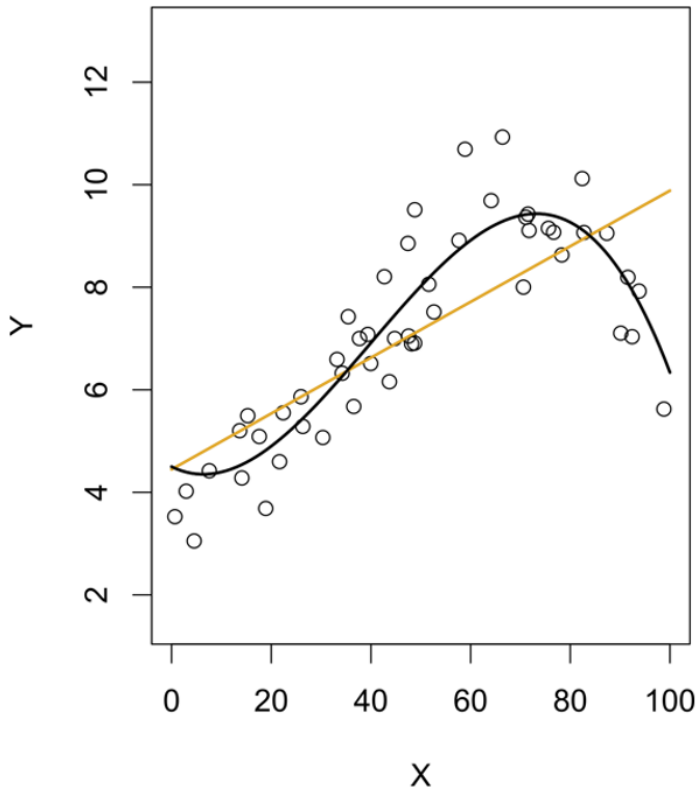


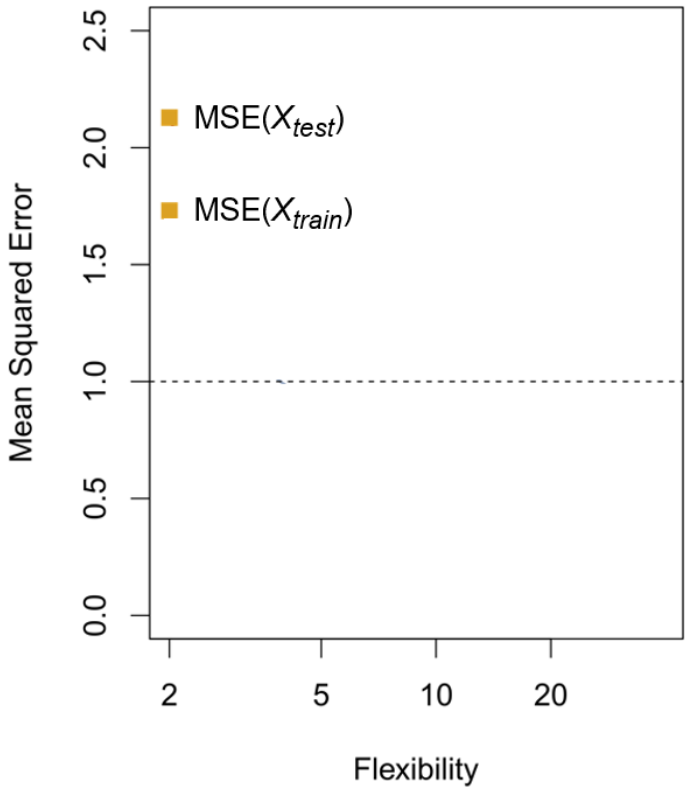
Measuring errors:

- Distance between orange line and *empty circles* = MSE(X_{train}).
- Distance between orange line and *black line* = E[MSE(X_{test})]

A simple $\hat{f}$ (linear)

- Prediction line from $\hat{f}_{simple}$ represented by orange line.



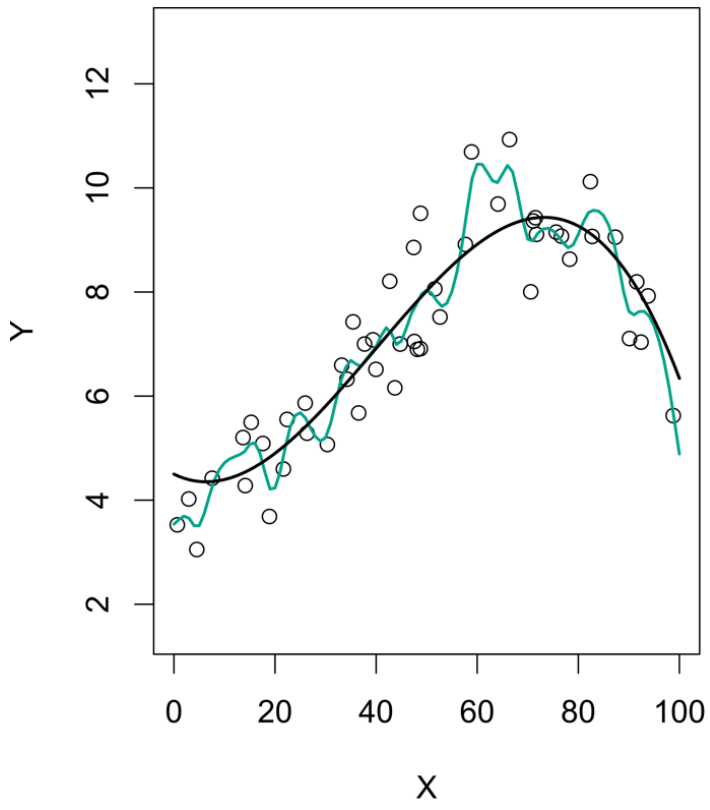


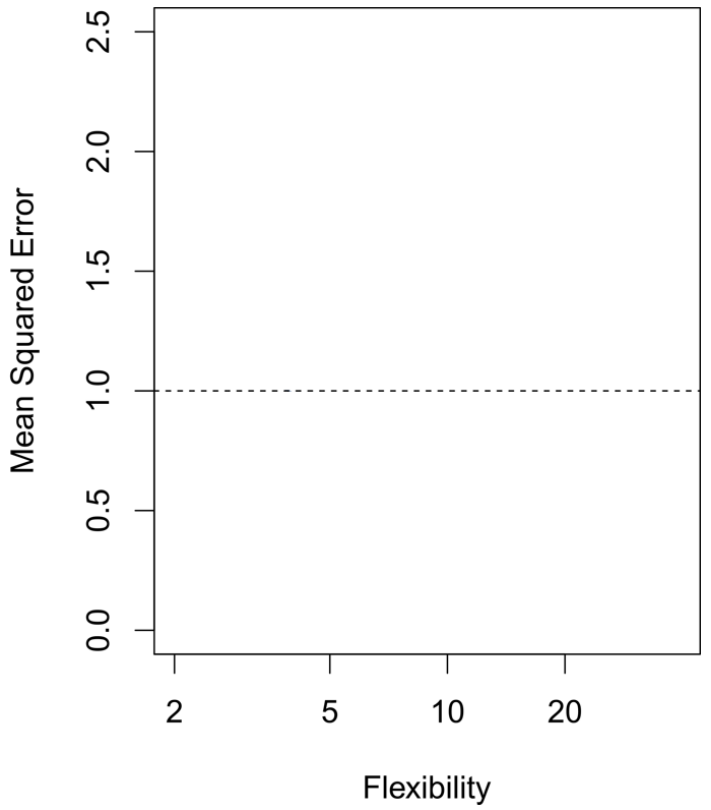
Measuring errors:

- Distance between orange line and *empty circles* = MSE(X_{train}).
- Distance between orange line and *black line* = E[MSE(X_{test})]
- Thus: MSE(X_{test}) is quite well-approximated by MSE(X_{train}).

A very flexible $\hat{f}$

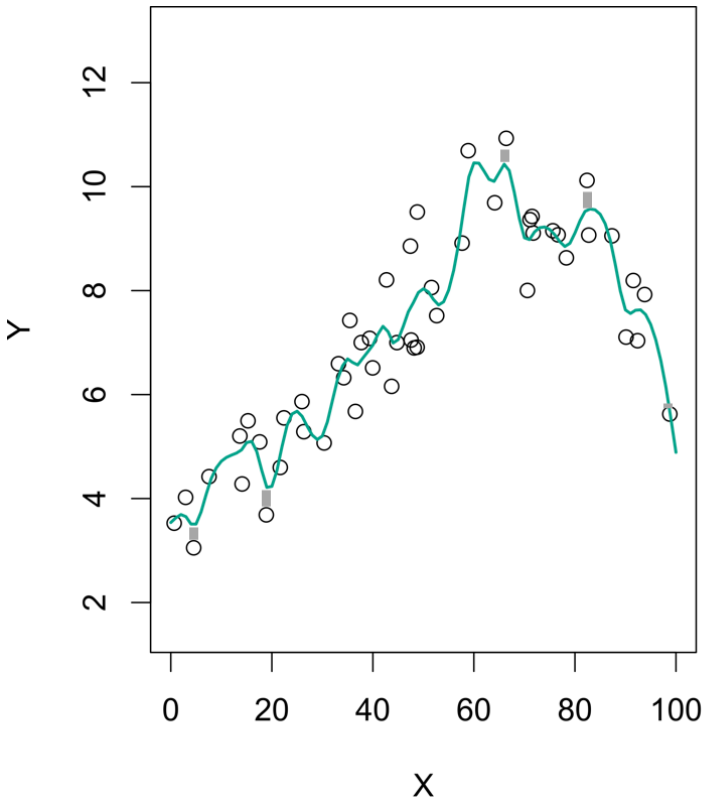
- Prediction line from $\hat{f}_{flexible}$ represented by green line.

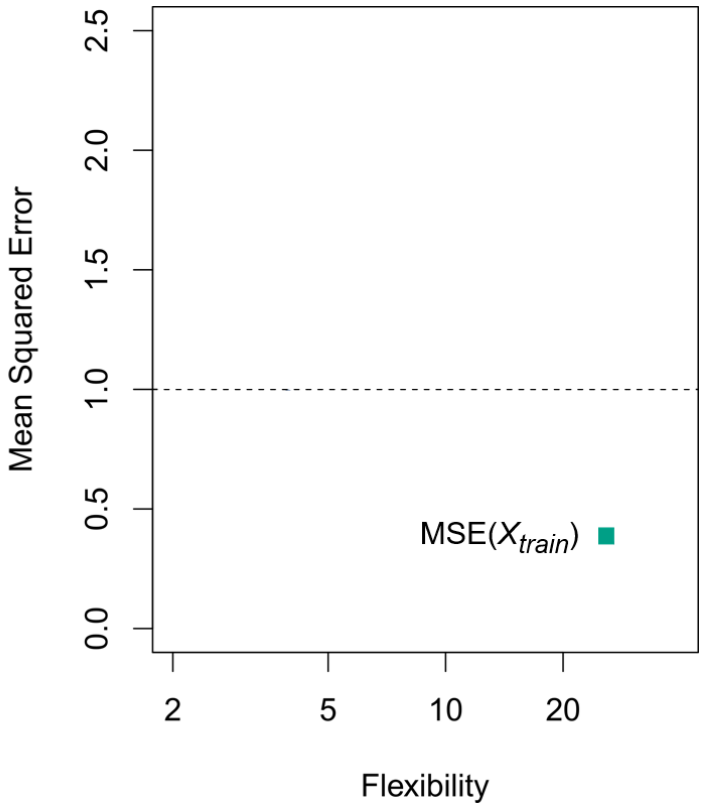




A very flexible $\hat{f}$

- Prediction line from $\hat{f}_{flexible}$ represented by green line.



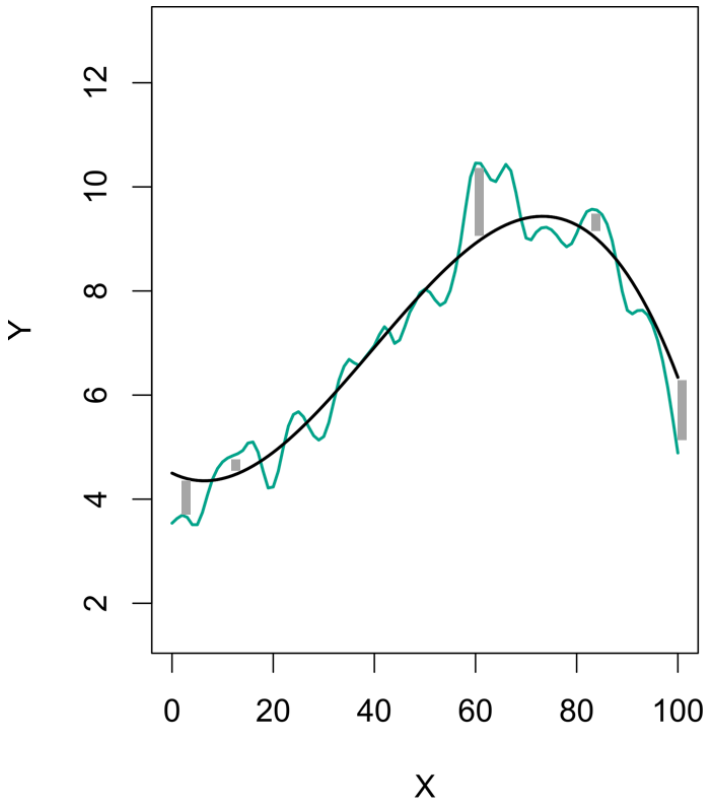


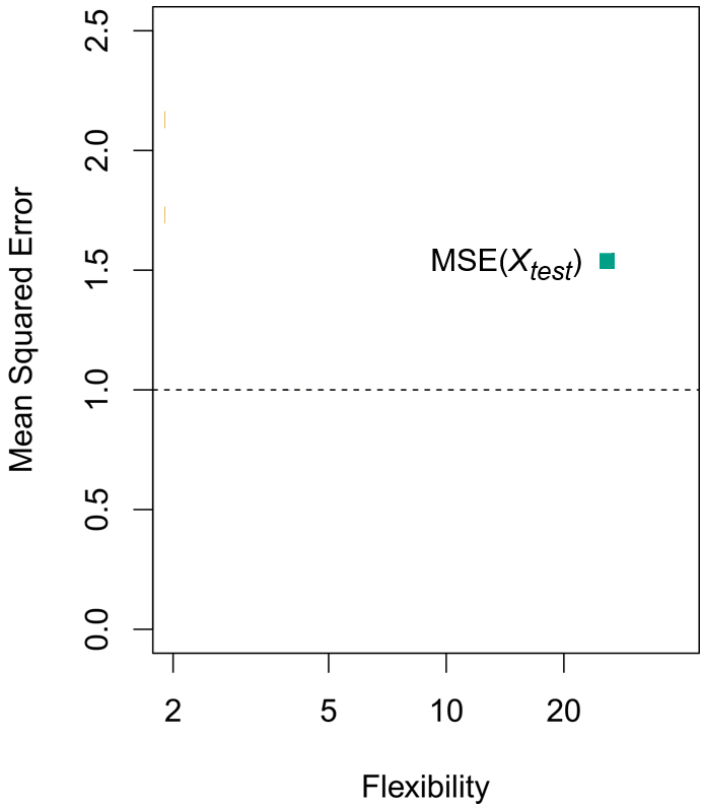
Measuring errors:

- Distance between green line and *empty circles* = MSE(X_{train}).

A very flexible $\hat{f}$

- Prediction line from $\hat{f}_{flexible}$ represented by green line.



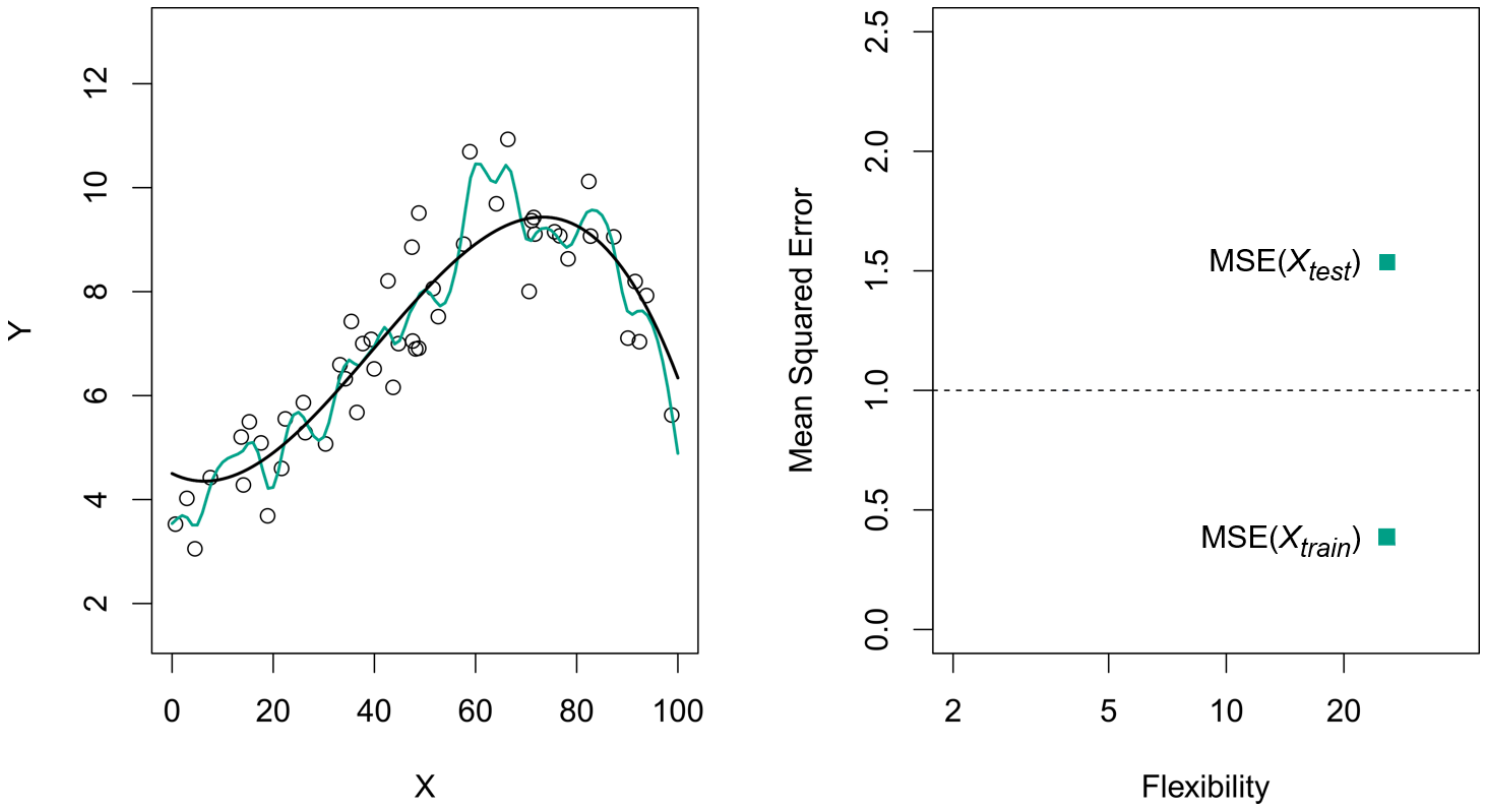


Measuring errors:

- Distance between green line and *empty circles* = MSE(X_{train}).
- Distance between green line and *black line* = E[MSE(X_{test})]

A very flexible $\hat{f}$

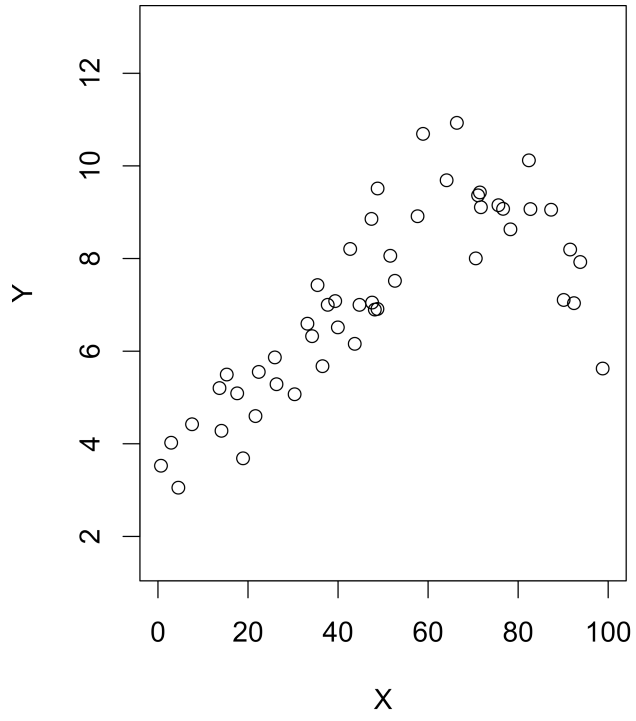
- Prediction line from $\hat{f}_{flexible}$ represented by green line.

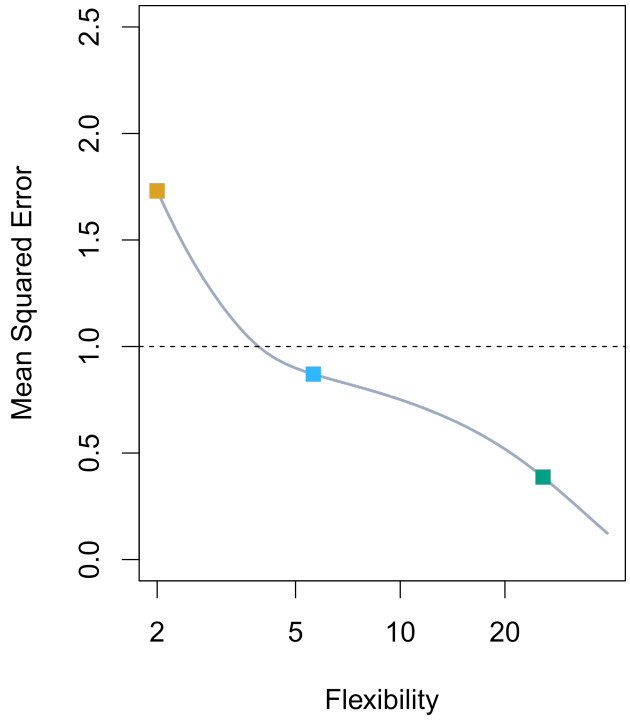


Measuring errors:

- Distance between green line and *empty circles* = MSE(X_{train}).
- Distance between green line and *black line* = E[MSE(X_{test})]
- Thus: MSE(X_{test}) is **not** well-approximated by MSE(X_{train}).

More generally: flexibility vs. error

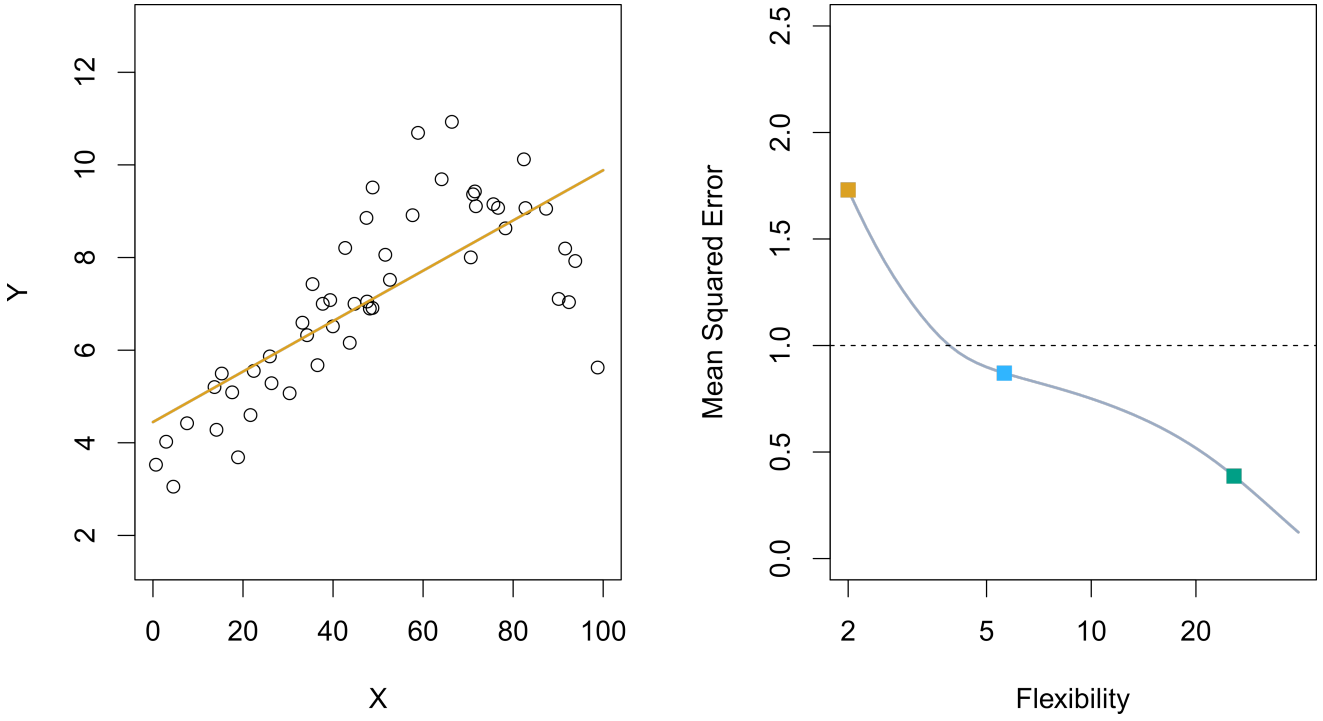




As we increase model flexibility...

- *Training* error improves *monotonically*.

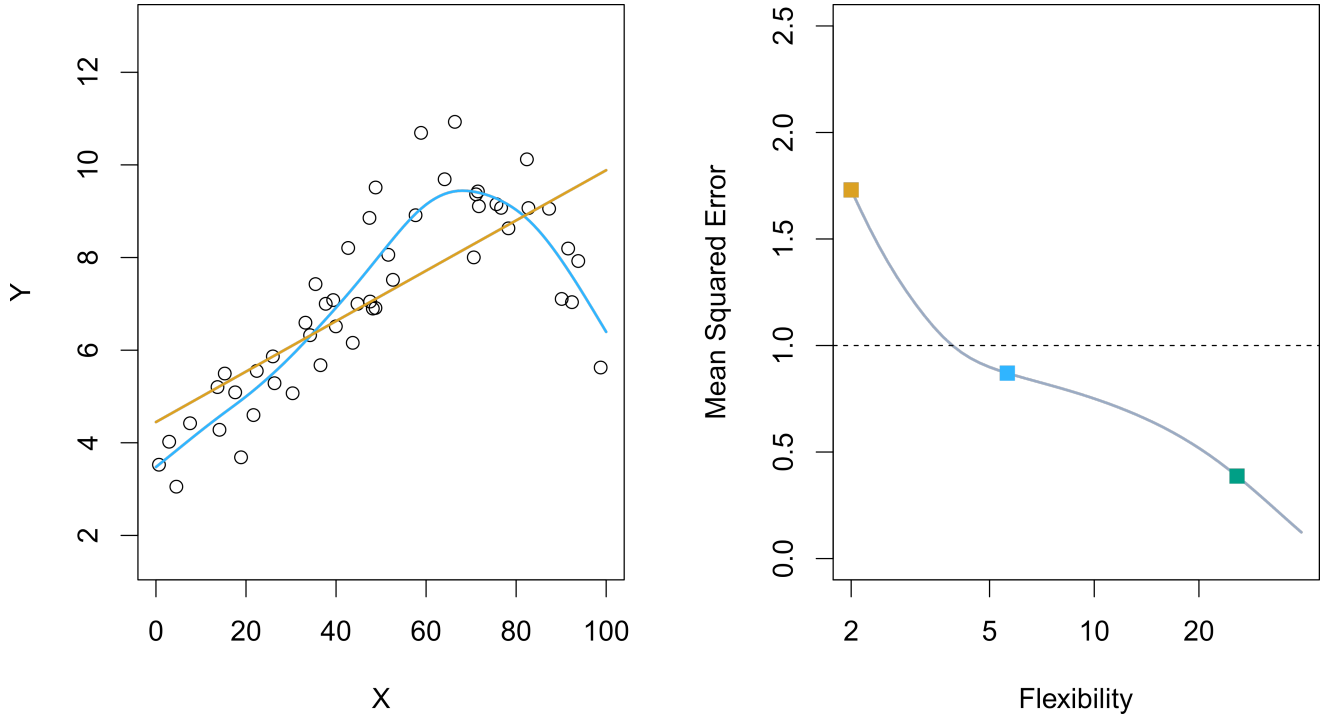
More generally: flexibility vs. error



As we increase model flexibility...

- *Training* error improves *monotonically*.

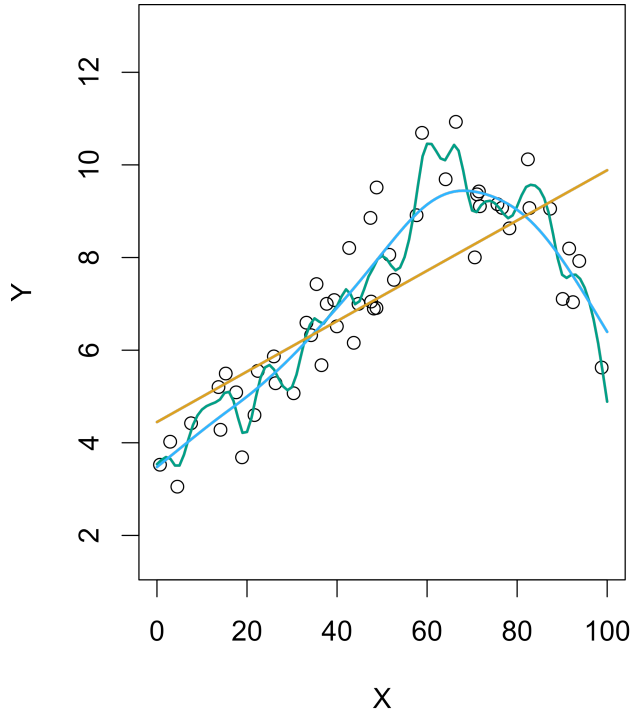
More generally: flexibility vs. error

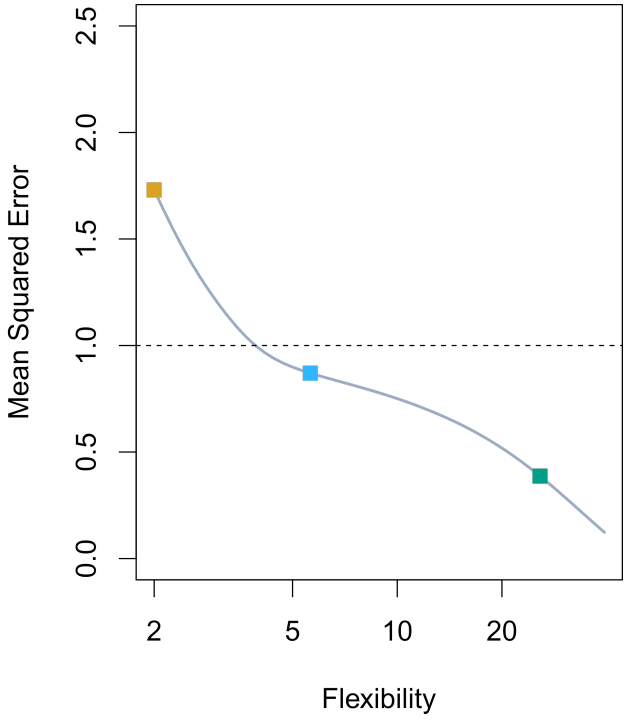


As we increase model flexibility...

- *Training* error improves *monotonically*.

More generally: flexibility vs. error

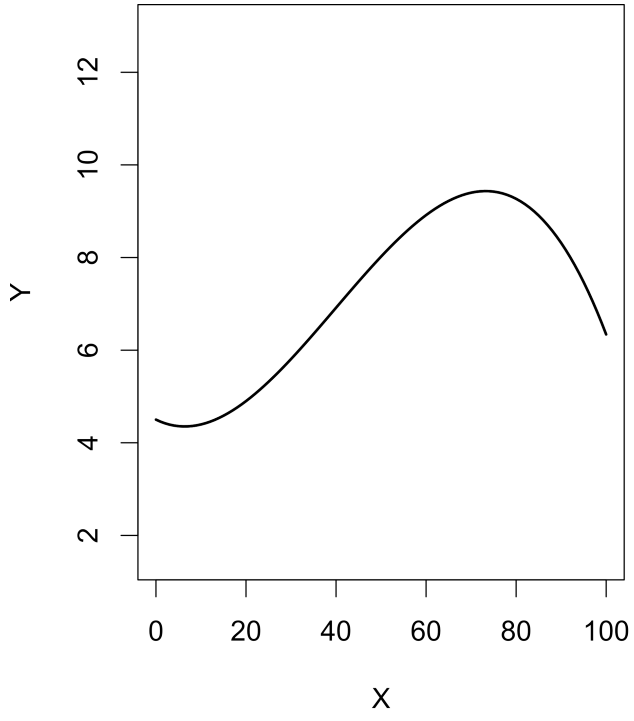


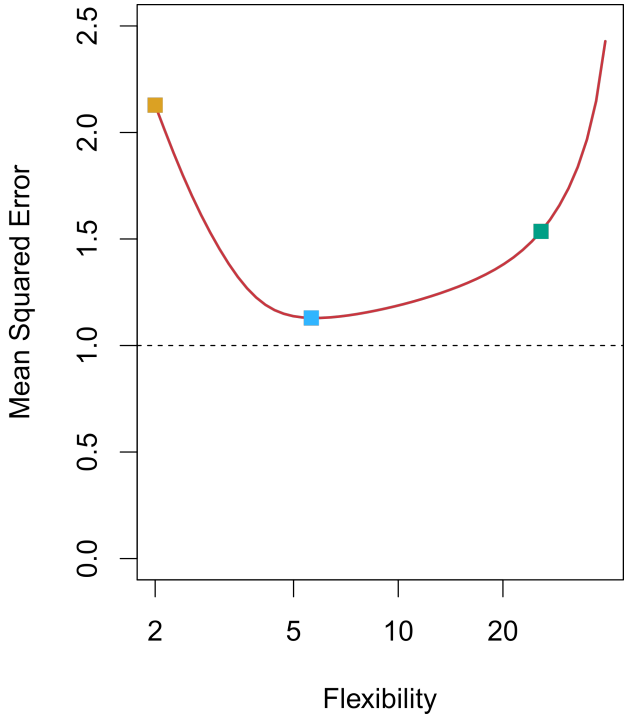


As we increase model flexibility...

- *Training* error improves *monotonically*.

More generally: flexibility vs. error

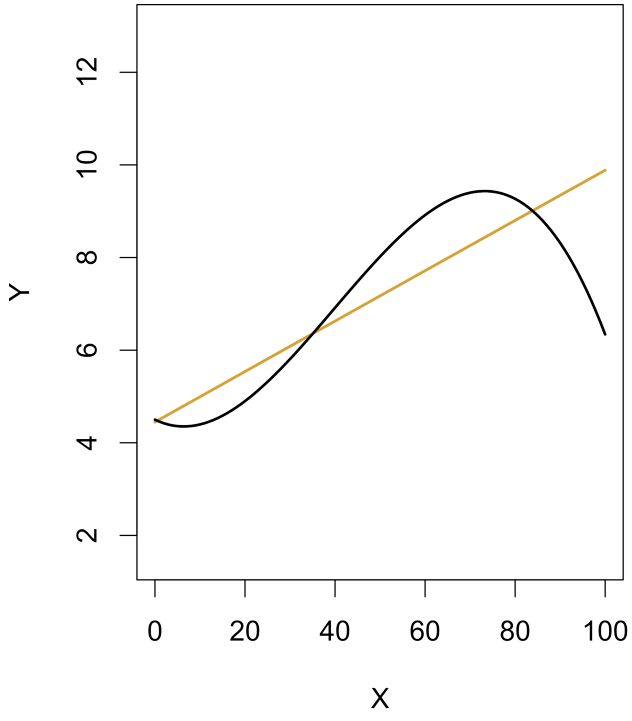


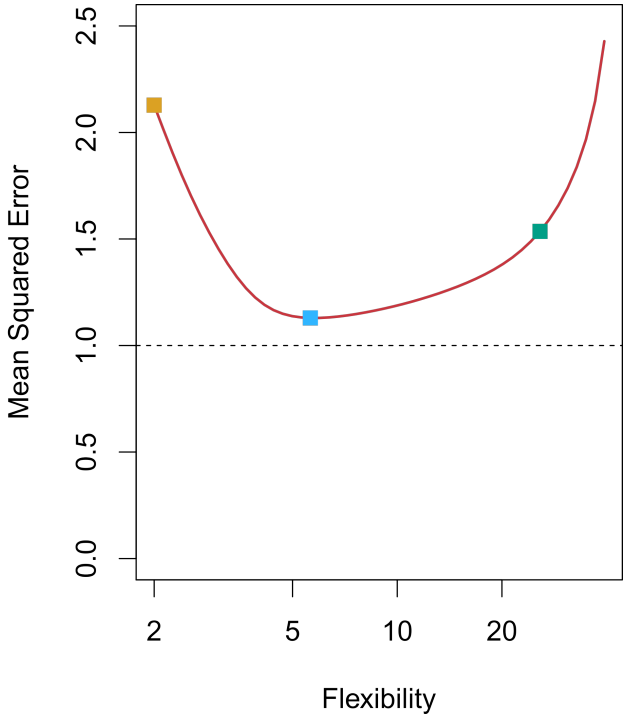


As we increase model flexibility...

- *Training* error improves *monotonically*.
- *Test* error usually improves *initially*, but then increases.

More generally: flexibility vs. error

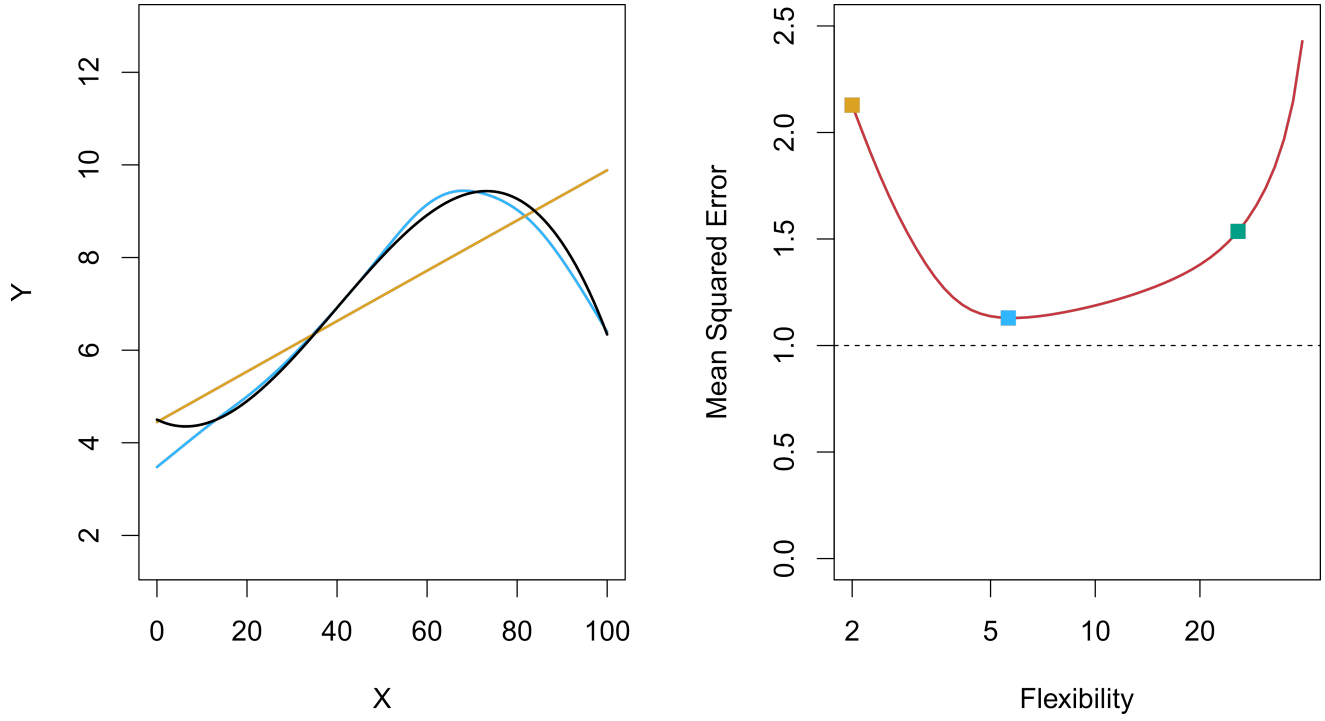




As we increase model flexibility...

- *Training* error improves *monotonically*.
- *Test* error usually improves *initially*, but then increases.
  - With a linear slope, we cannot capture pattern of true f.

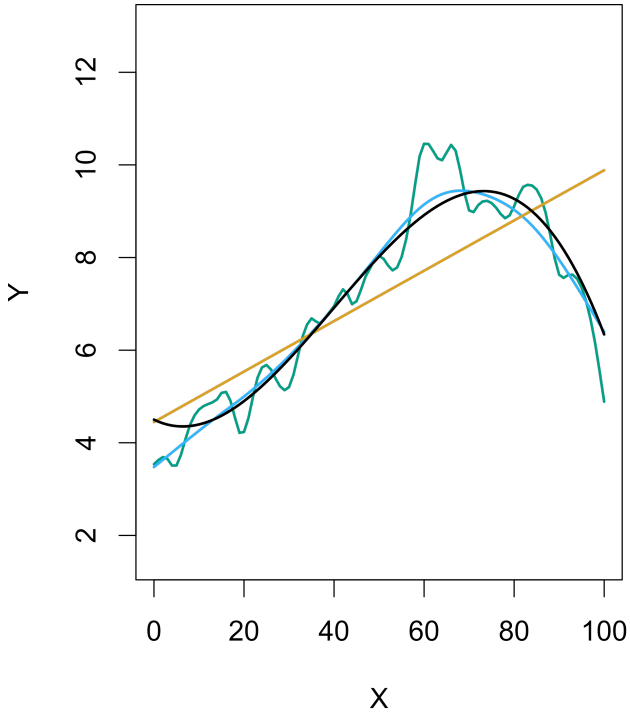
More generally: flexibility vs. error

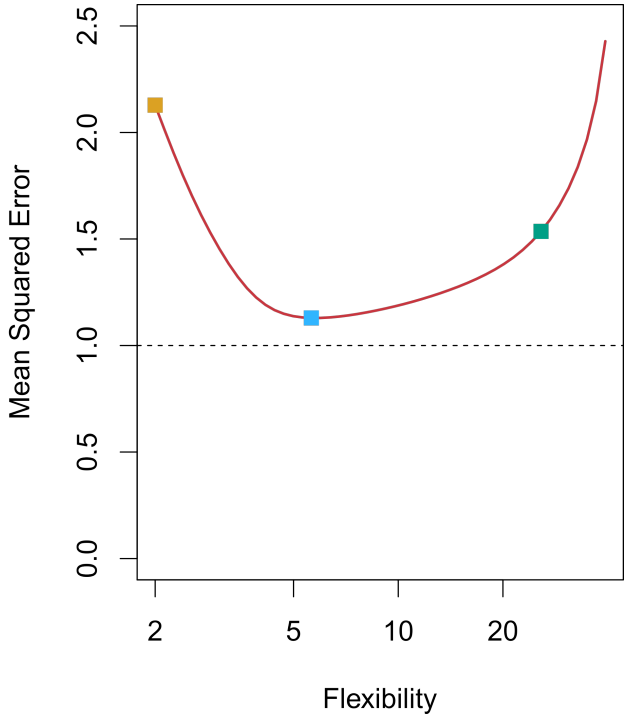


As we increase model flexibility...

- *Training* error improves *monotonically*.
- *Test* error usually improves *initially*, but then increases.
  - With a linear slope, we cannot capture pattern of true f.
  - Adding some flexibility, we get close to pattern of true f.

More generally: flexibility vs. error





As we increase model flexibility...

- *Training* error improves *monotonically*.
- *Test* error usually improves *initially*, but then increases.
  - With a linear slope, we cannot capture pattern of true f.
  - Adding some flexibility, we get close to pattern of true f.
  - But then, test error increases because of capturing noise in X_{train}.

Bias-Variance Trade-off

- The U-shape of the *test error* (red line) is a result of *two competing—fundamental—properties* of learning methods.
  - Bias
  - Variance

Bias-Variance Trade-off

- The U-shape of the *test error* (red line) is a result of *two competing—fundamental—properties* of learning methods.
  - Bias
  - Variance

$$\text{Test error} = \text{Bias} + \text{Variance} + \text{Irreducible error}$$

Bias-Variance Trade-off

- The U-shape of the *test error* (red line) is a result of *two competing—fundamental—properties* of learning methods.
  - Bias
  - Variance

$$\text{Test error} = \text{Bias} + \text{Variance} + \text{Irreducible error}$$

- Thus, to minimize test error $\rightarrow$ select an $\hat{f}$ that *simultaneously* achieves *low variance* and *low bias*.

Bias-Variance Trade-off

- The U-shape of the *test error* (red line) is a result of *two competing—fundamental—properties* of learning methods.
  - Bias
  - Variance

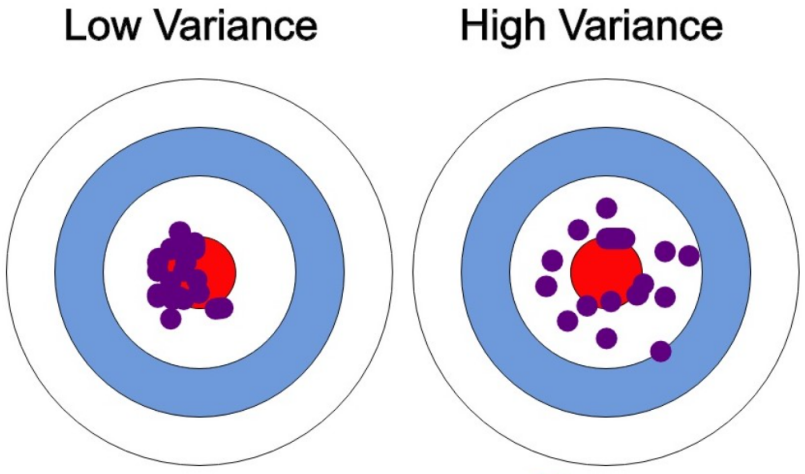
$$\text{Test error} = \text{Bias} + \text{Variance} + \text{Irreducible error}$$

- Thus, to minimize test error $\rightarrow$ select an $\hat{f}$ that *simultaneously* achieves *low variance* and *low bias*.

- Begs the question—what is bias and what is variance?
  - Let's consider them one by one.

Variance

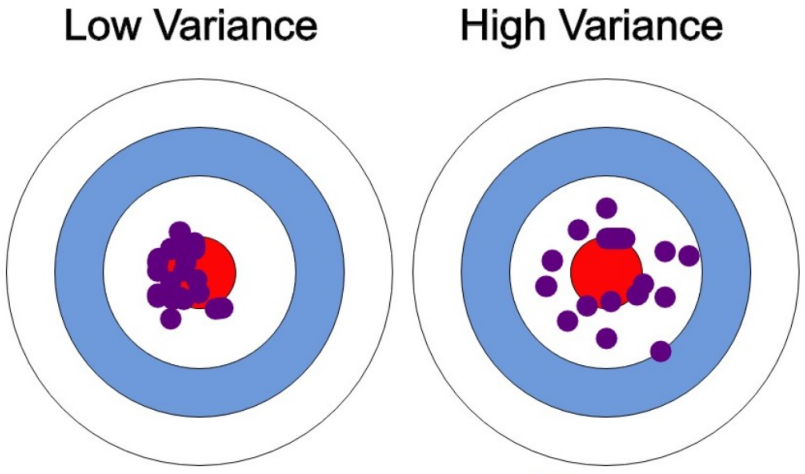
- Variance = amount by which $\hat{f}$ would change if we estimated using a *different training data set*.



- High variance: small changes in training data $\rightarrow$ large changes in $\hat{f}$.

Variance

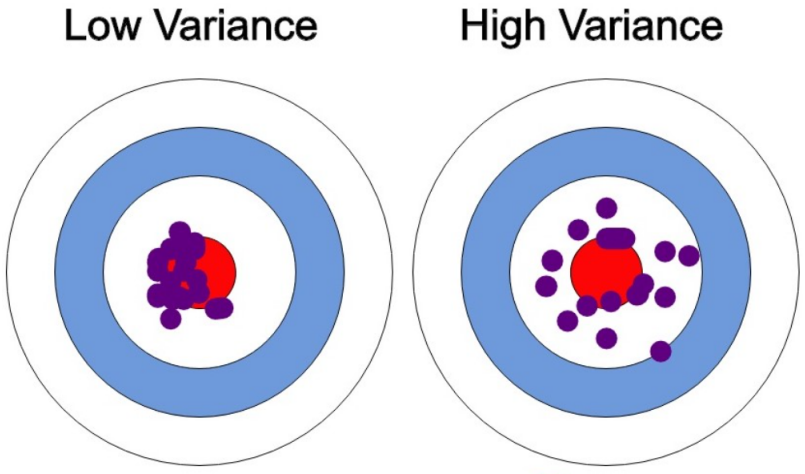
- Variance = amount by which $\hat{f}$ would change if we estimated using a *different training data set*.



- High variance: small changes in training data $\rightarrow$ large changes in $\hat{f}$.
- In general, more flexible models $\rightarrow$ higher variance.
  - Since they are more likely to pick up particularities in training data.

Variance

- Variance = amount by which $\hat{f}$ would change if we estimated using a *different training data set*.

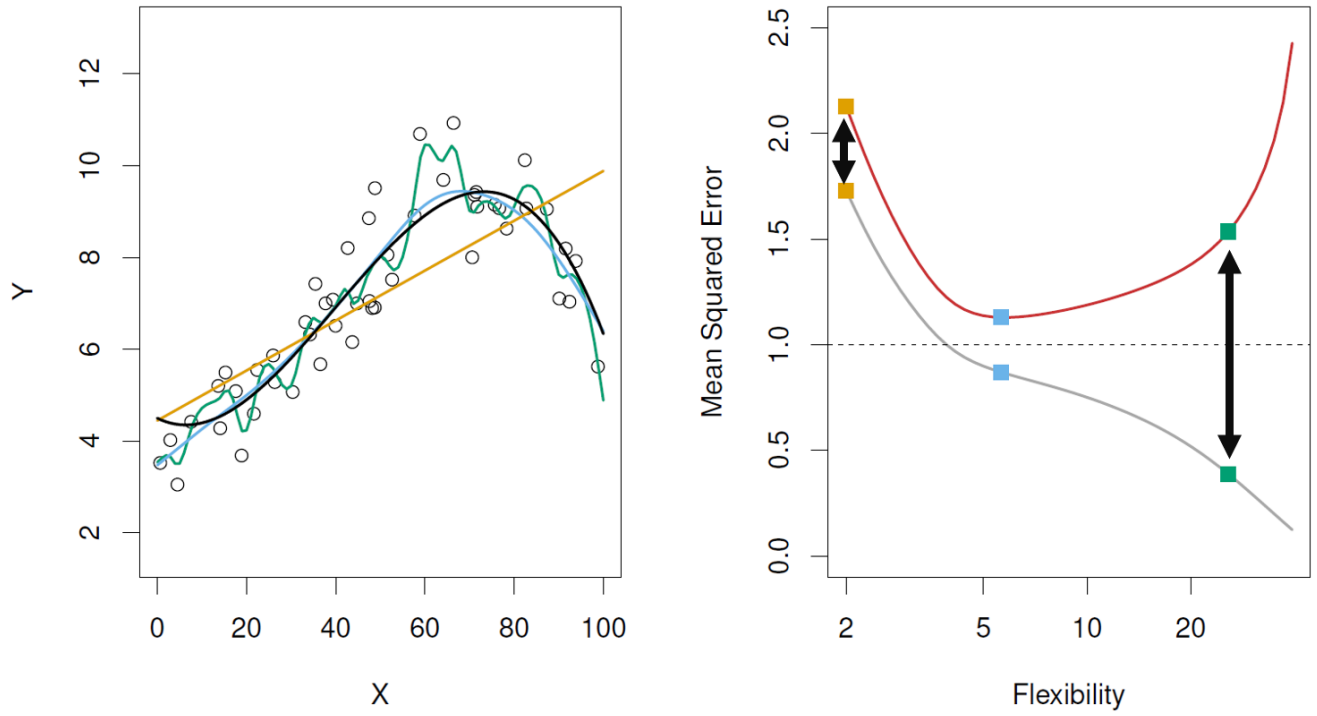


- High variance: small changes in training data $\rightarrow$ large changes in $\hat{f}$.
- In general, more flexible models $\rightarrow$ higher variance.
  - Since they are more likely to pick up particularities in training data.
- And thus: high variance $\approx$ overfitting.

Variance

Reconsidering Fig. 2.9

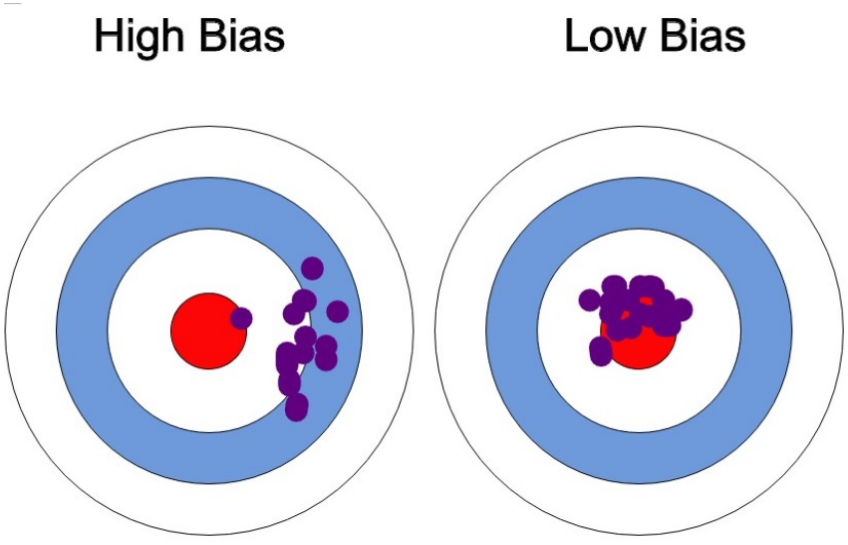
- Green curve (flexible fit): **high** variance
- Orange curve (linear fit): **low** variance



(As indicated by distance between red & gray line.)

Bias

- Bias = amount by which $\hat{f}$ fails to capture the real-life problem; the underlying relations—e.g., excluded variables and non-linearity.

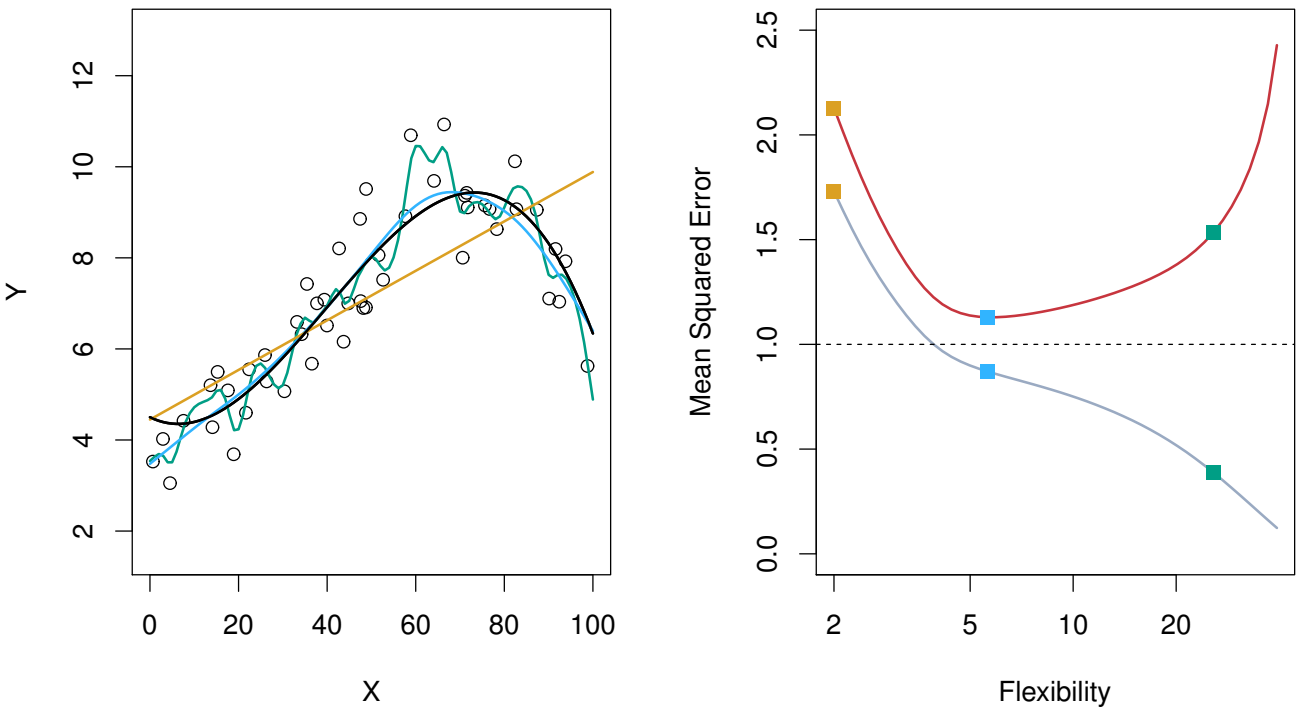


- In general, less flexible models $\rightarrow$ higher bias.
- And: high bias $\approx$ underfitting.

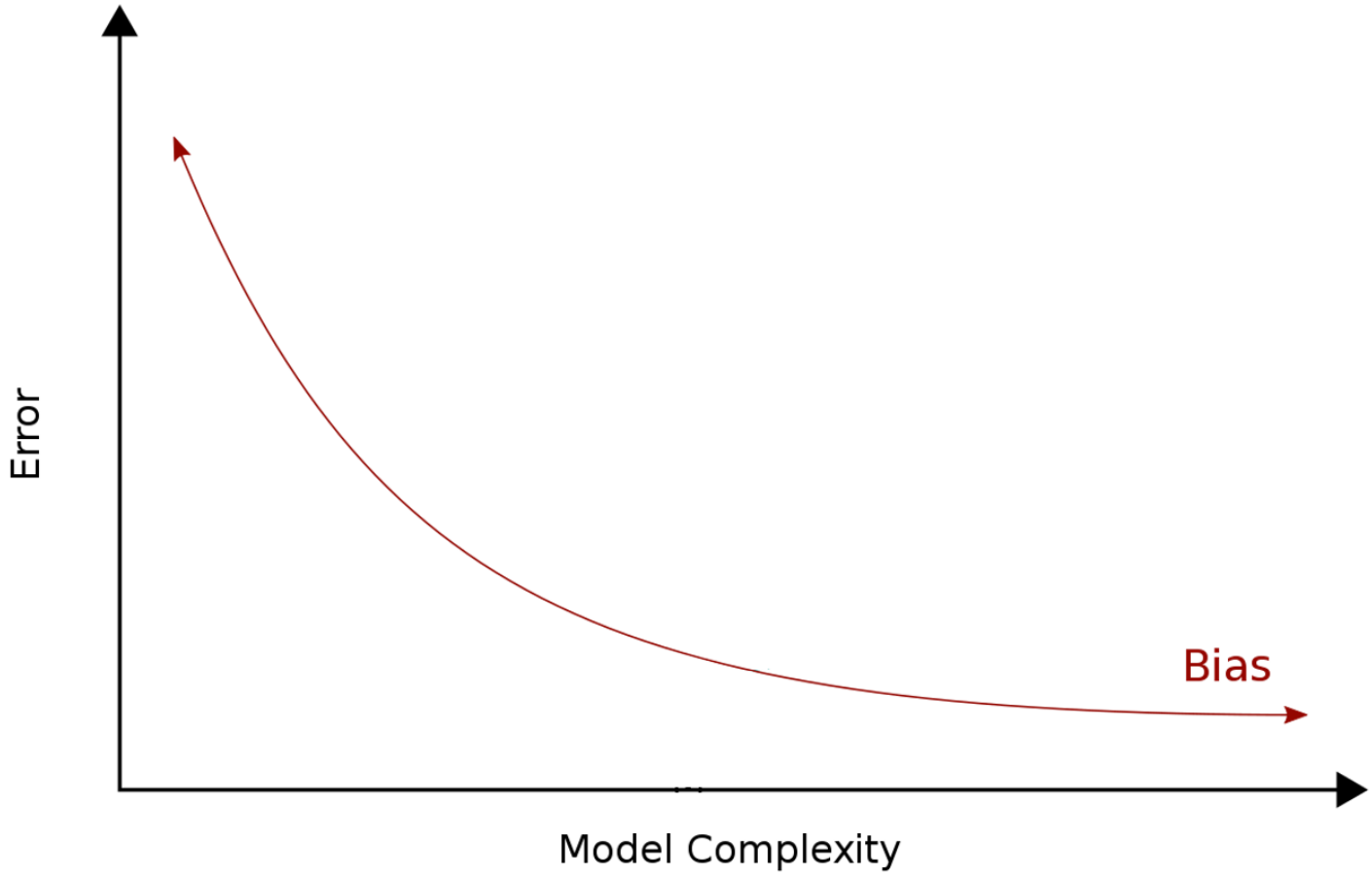
Bias

Reconsidering Fig. 2.9

- Orange curve (linear fit): **high** bias—no matter how many obs, linear fit will not capture underlying f.



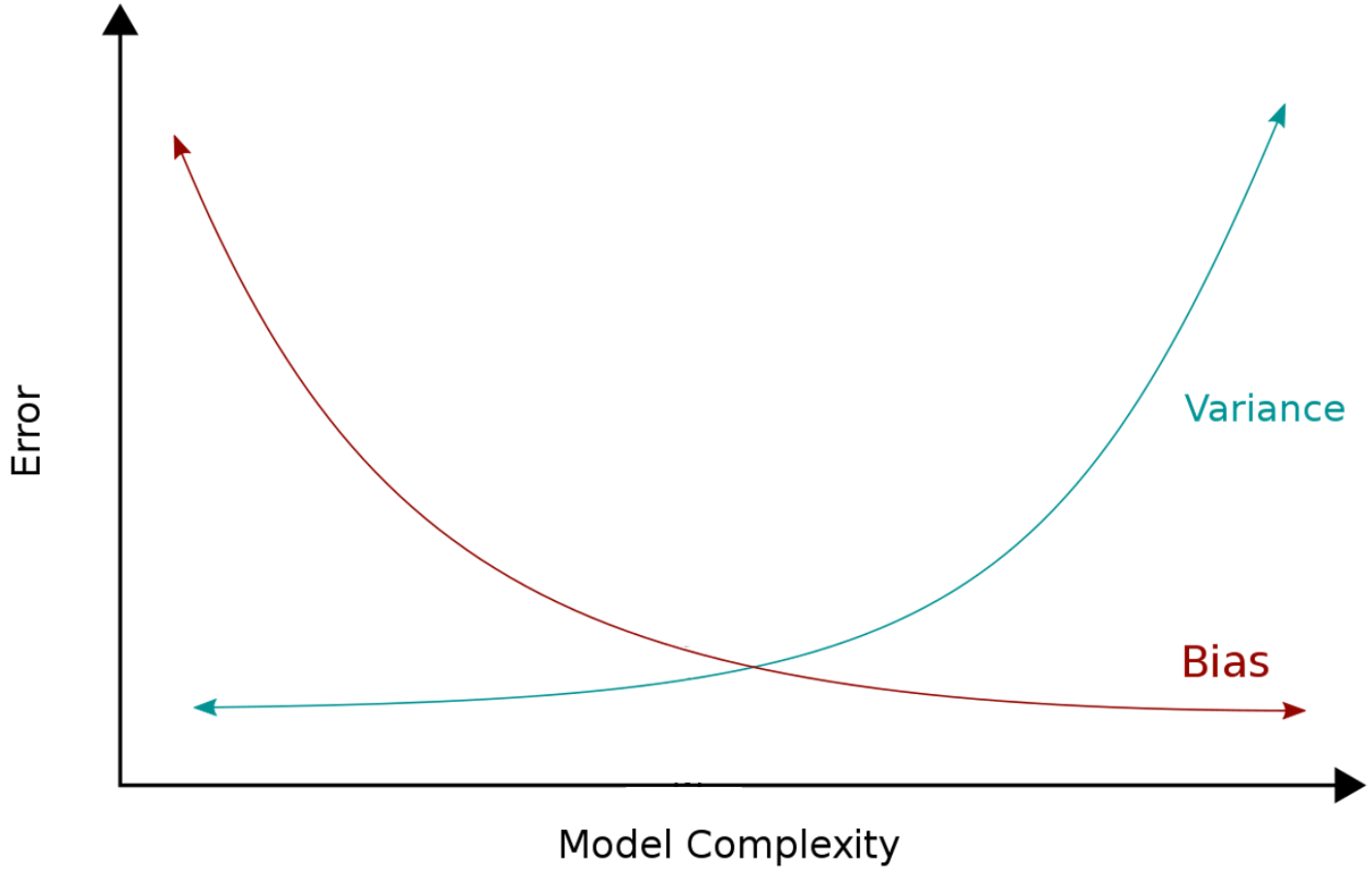
Thus: the ''Bias-Variance Trade-off''



Increasing model flexibility/complexity...

- **Reduces** bias

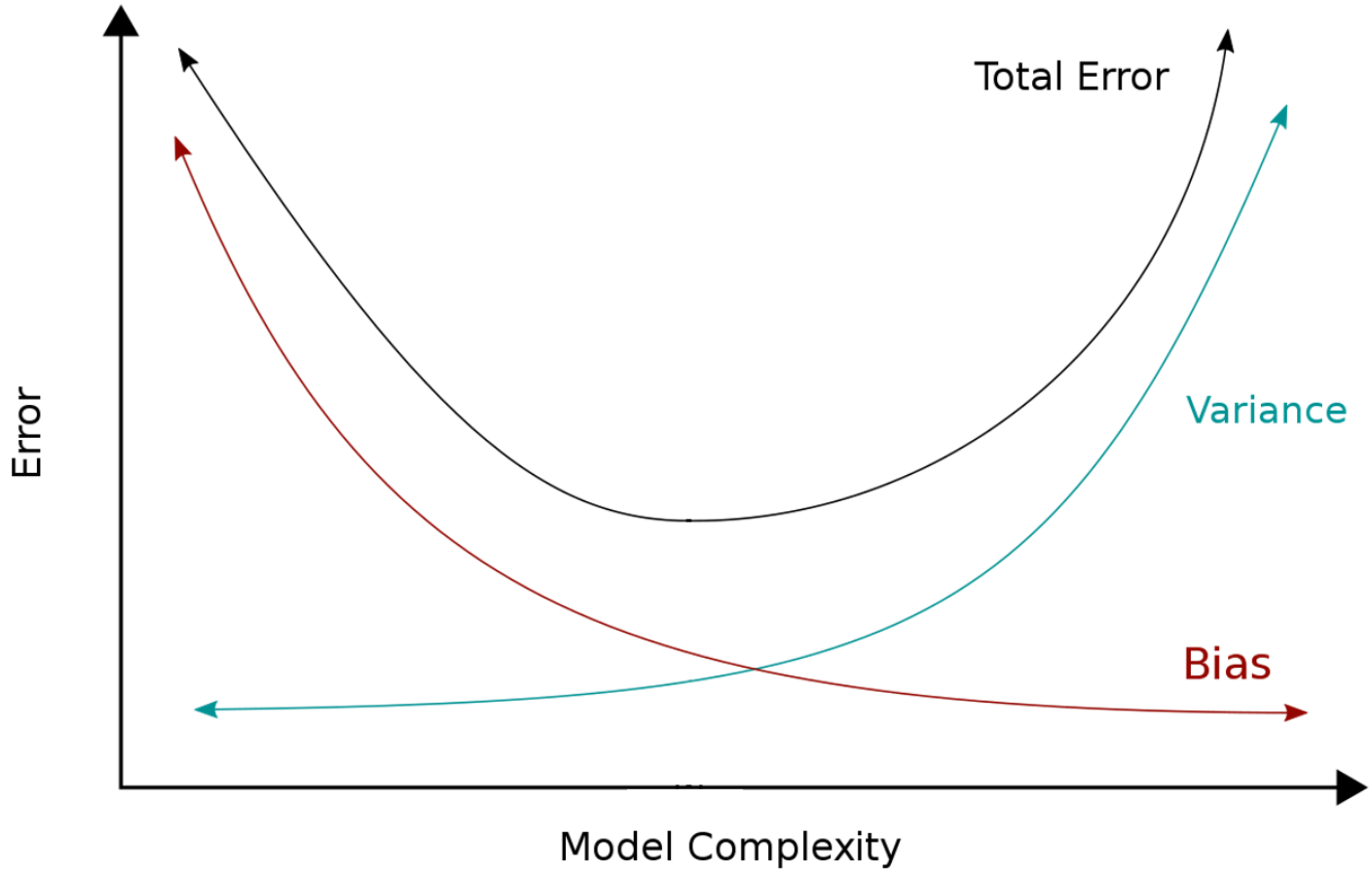
Thus: the ''Bias-Variance Trade-off''



Increasing model flexibility/complexity...

- **Reduces** bias
- But: **increases** variance

Thus: the ''Bias-Variance Trade-off''

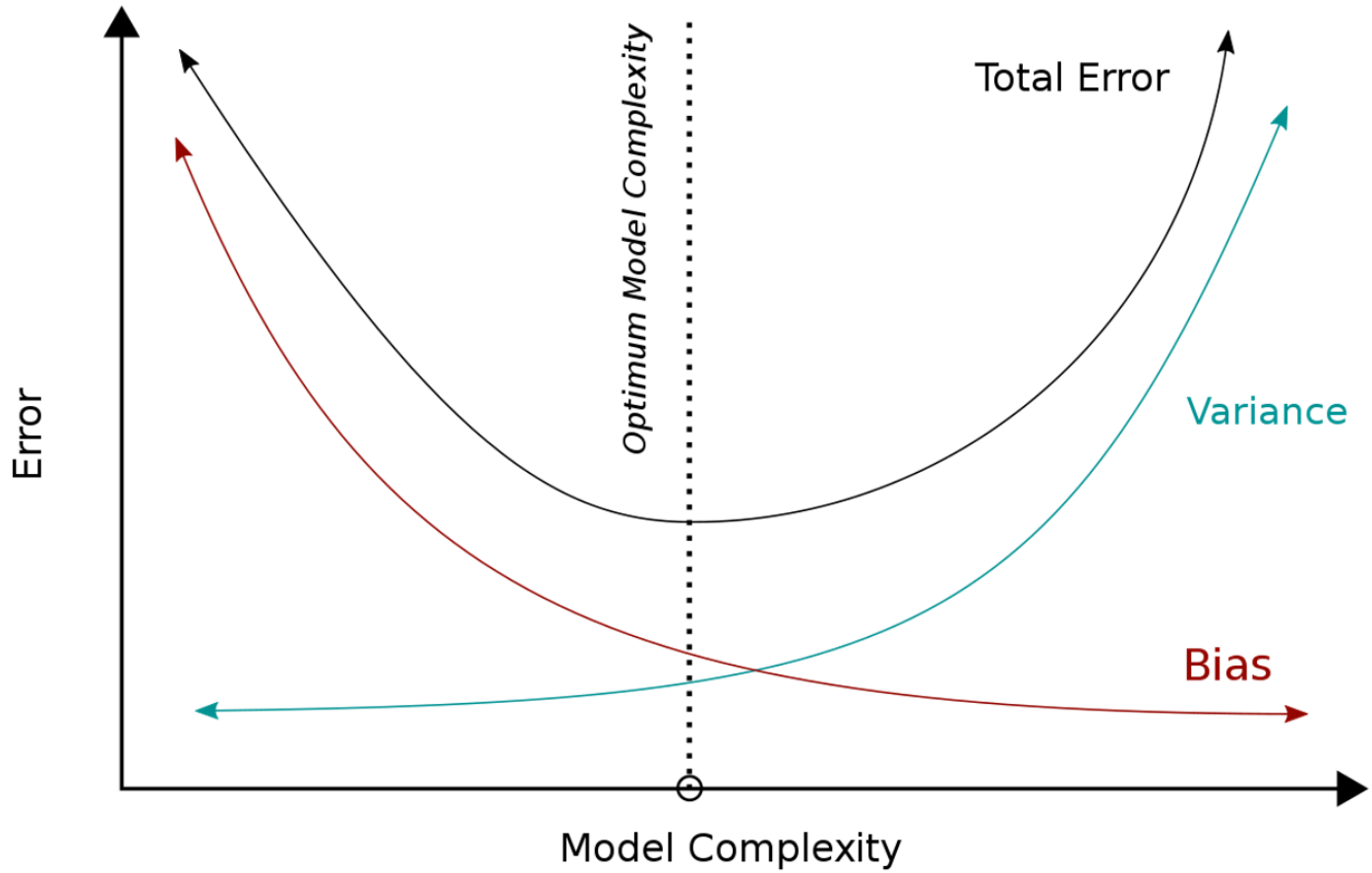


Increasing model flexibility/complexity...

- **Reduces** bias
- But: **increases** variance

$\Rightarrow$ A good $\hat{f}$ finds "the right balance" between the two.

Thus: the ''Bias-Variance Trade-off''



Increasing model flexibility/complexity...

- **Reduces** bias
- But: **increases** variance

$\Rightarrow$ A good $\hat{f}$ finds "the right balance" between the two.

Recap & what's next

Ok, so now we have a good understanding of:

- The *bias-variance trade-off* and the *tension* of model *complexity*:
  - Too much flexibility $\rightarrow$ overfitting
  - Too little flexibility $\rightarrow$ underfitting

Recap & what's next

Ok, so now we have a good understanding of:

- The *bias-variance trade-off* and the *tension* of model *complexity*:
  - Too much flexibility $\rightarrow$ overfitting
  - Too little flexibility $\rightarrow$ underfitting

Next: turn to the problem of **measuring the test error in practice**.

- Because we do not in general know the true f, we cannot—as we did before—use it to compute $\mathrm{E}[\mathrm{MSE}(X_{test})]$.

Recap & what's next

Ok, so now we have a good understanding of:

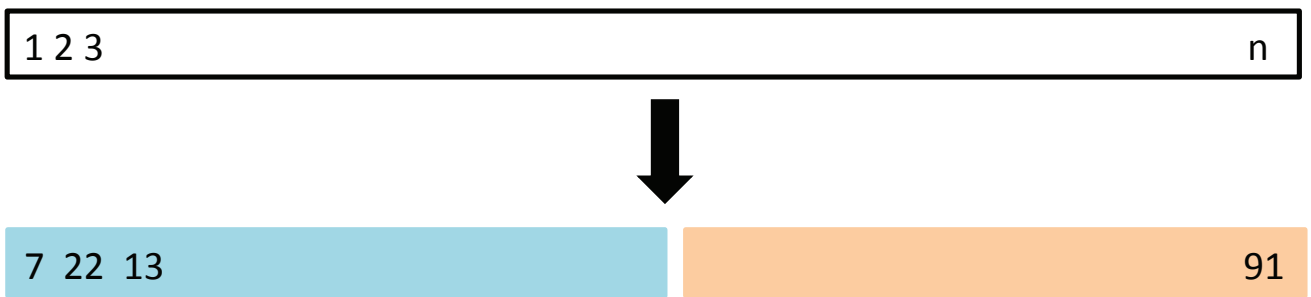
- The *bias-variance trade-off* and the *tension* of model *complexity*:
  - Too much flexibility $\rightarrow$ overfitting
  - Too little flexibility $\rightarrow$ underfitting

Next: turn to the problem of **measuring the test error in practice**.

- Because we do not in general know the true f, we cannot—as we did before—use it to compute $\mathrm{E}[\mathrm{MSE}(X_{test})]$.
- Instead, we have to **estimate** it based on our data — how??

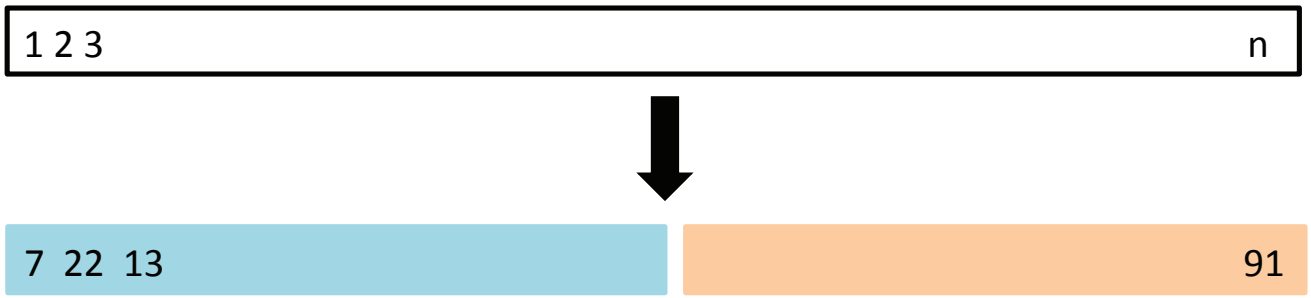
A first approach: the "validation set" approach

- Very simple—randomly divide our data into two parts:
  - A "training" set
  - A "validation" set



A first approach: the "validation set" approach

- Very simple—randomly divide our data into two parts:
  - A "training" set
  - A "validation" set

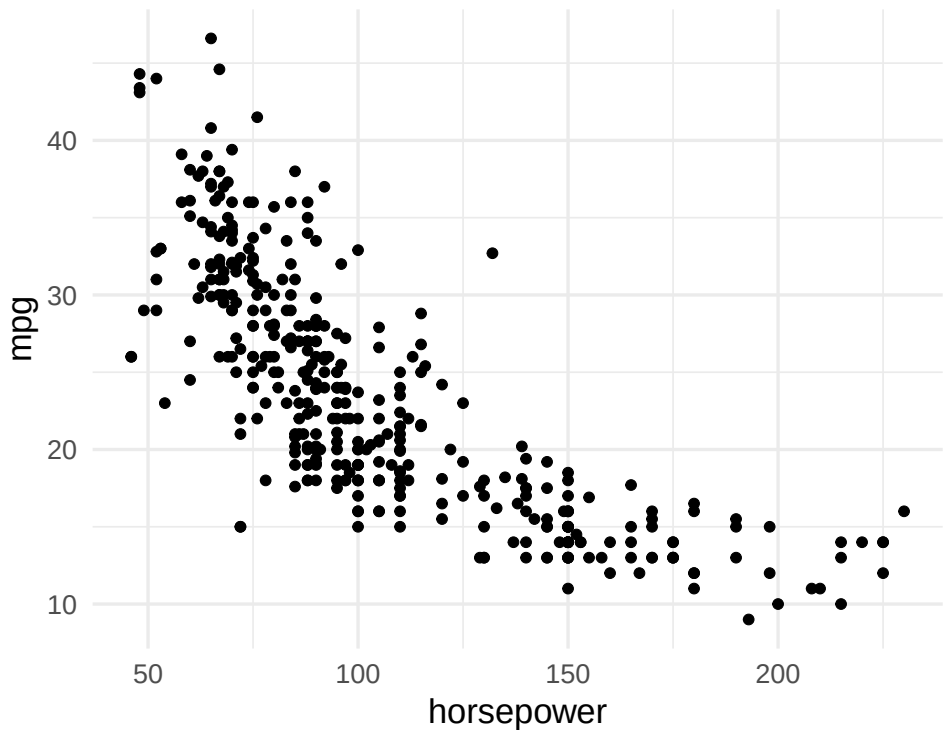


- Then:
  1. Estimate $\hat{f}$ on training set
  2. Make predictions using $\hat{f}$ on the validation set.
  3. Calculate MSE based on #2 $\rightarrow$ estimate of test error.

Example

Let's consider an example from ISL (p.198):

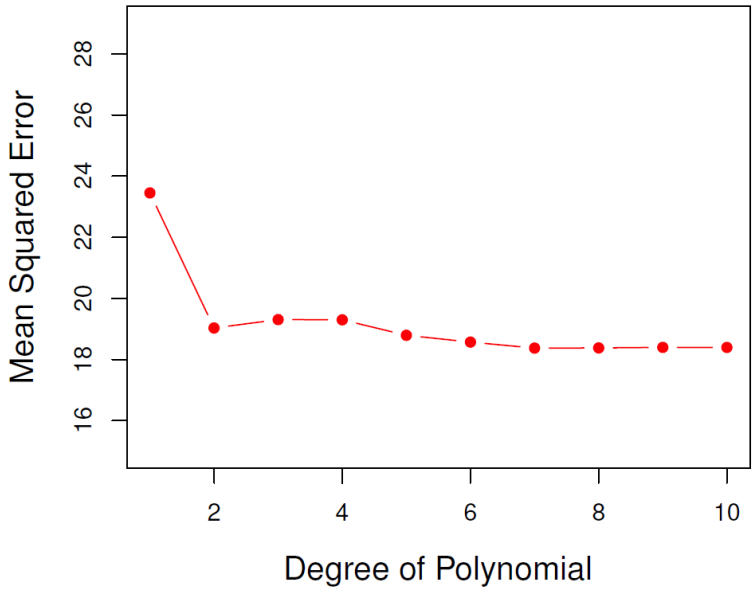
- *Objective*: Learn $\hat{f}$ that predicts `mpg`[1] based on `horsepower`.
- *Question*: Using linear regression,
  which polynomial degree provides the best predictions?



[1]=Milage per gallon

Using validation-set approach...

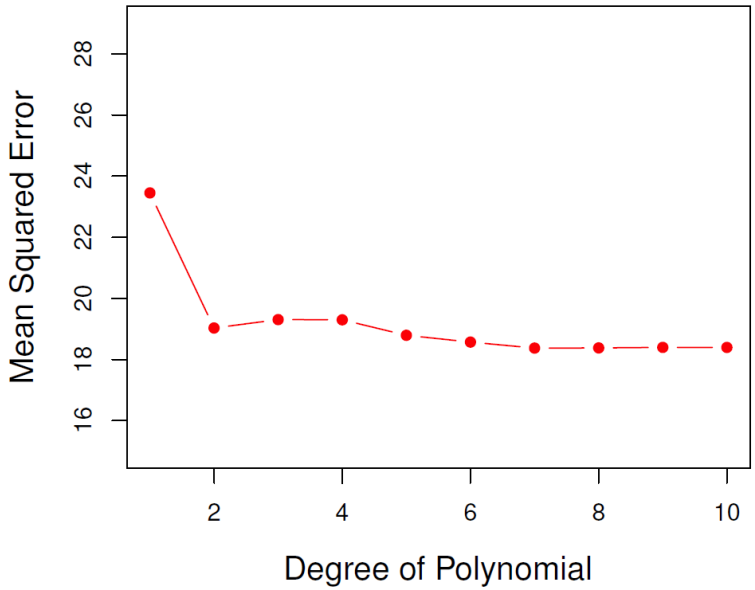
1. Split data into two sets (train, validation)
2. Estimate 10 models (one for each polynomial)
3. Predict on validation set and calculate errors.



$\rightarrow$ 2!

Using validation-set approach...

1. Split data into two sets (train, validation)
2. Estimate 10 models (one for each polynomial)
3. Predict on validation set and calculate errors.

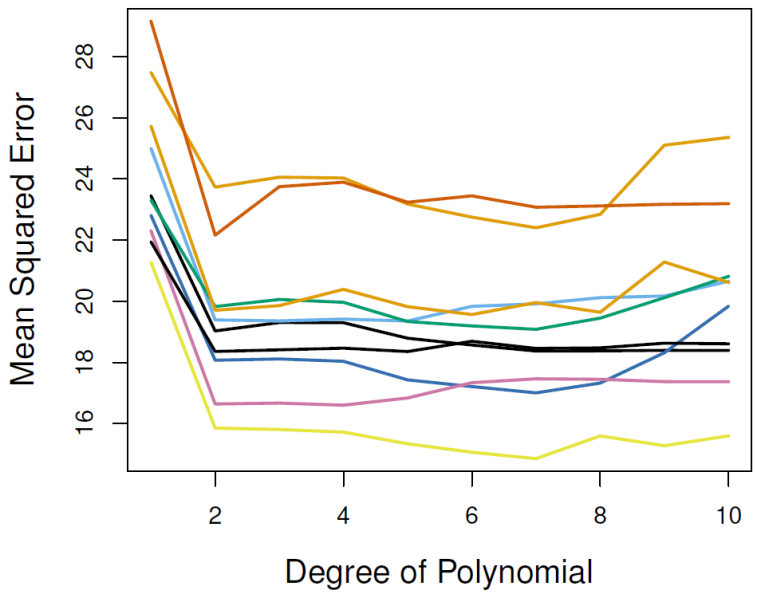


$\rightarrow$ 2!

(After 2: very little gain for more complexity)

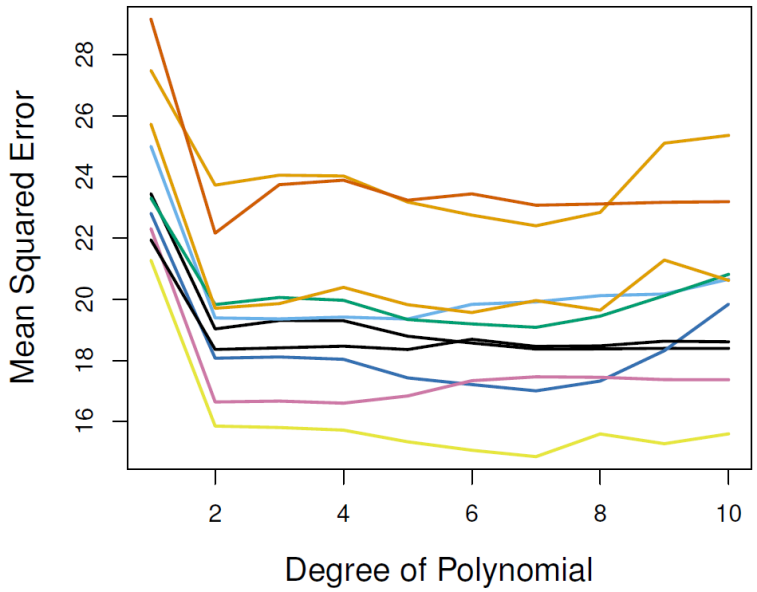
Drawbacks

1. The estimate of the test error can be **highly variable** — depending on exactly which observations are included in training/validation set.



Drawbacks

1. The estimate of the test error can be **highly variable** — depending on exactly which observations are included in training/validation set.



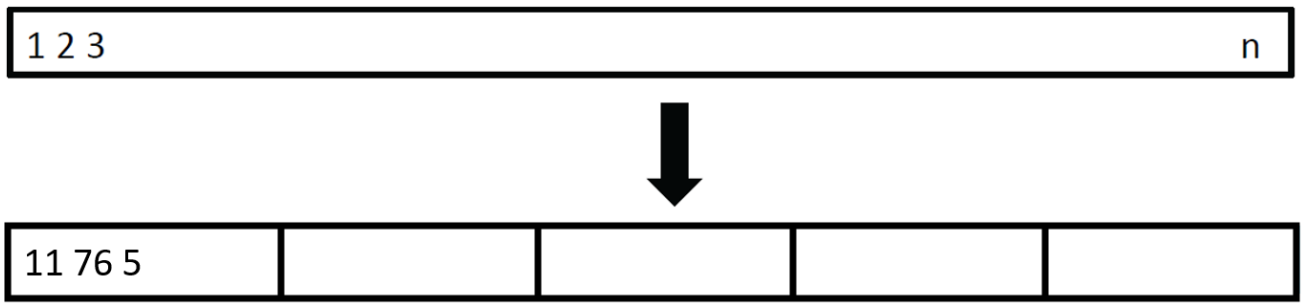
2. Because **only a subset** of observation are used to fit the model—and because statistical methods tend to perform worse when trained on fewer obs $\rightarrow$ validation set error may **overestimate test error**.

K-fold Cross-validation

- Slightly more sophisticated approach that address these issues.

K-fold Cross-validation

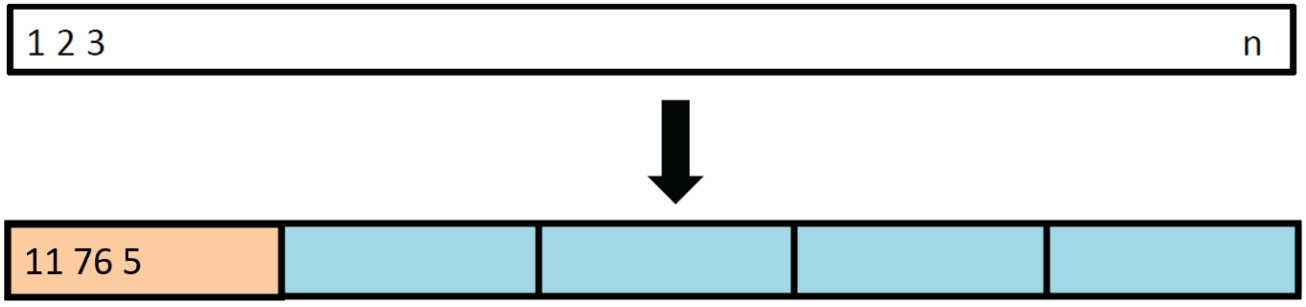
- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.

K-fold Cross-validation

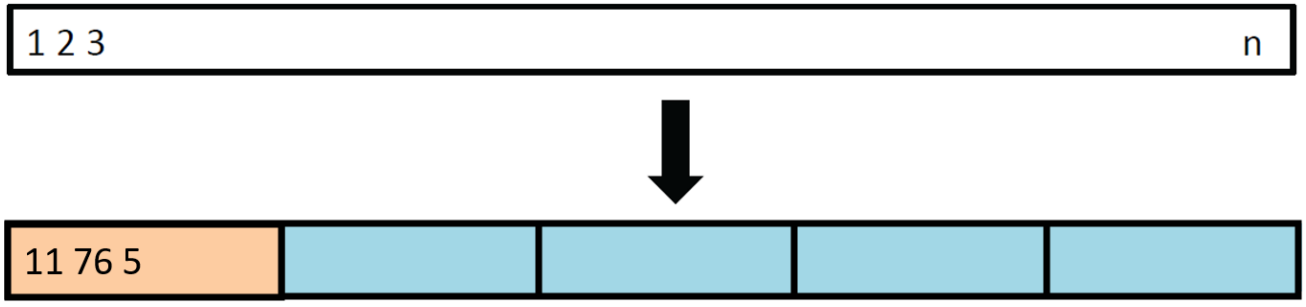
- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.
2. Leave out one part, k, and fit model $\hat{f}_k$ to the other $K-1$ parts.

K-fold Cross-validation

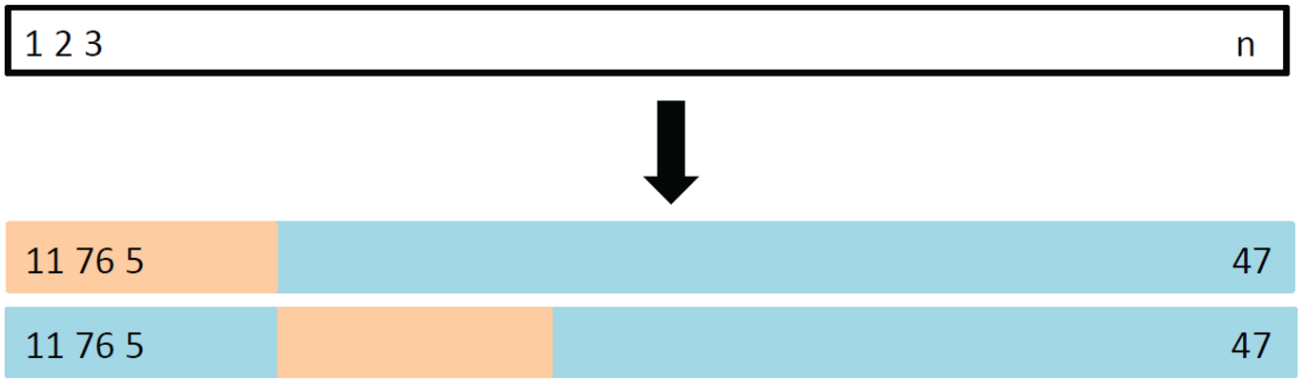
- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.
2. Leave out one part, k, and fit model $\hat{f}_k$ to the other $K-1$ parts.
3. Make predictions using $\hat{f}_k$ on left-out part k, and calculate MSE_k.

K-fold Cross-validation

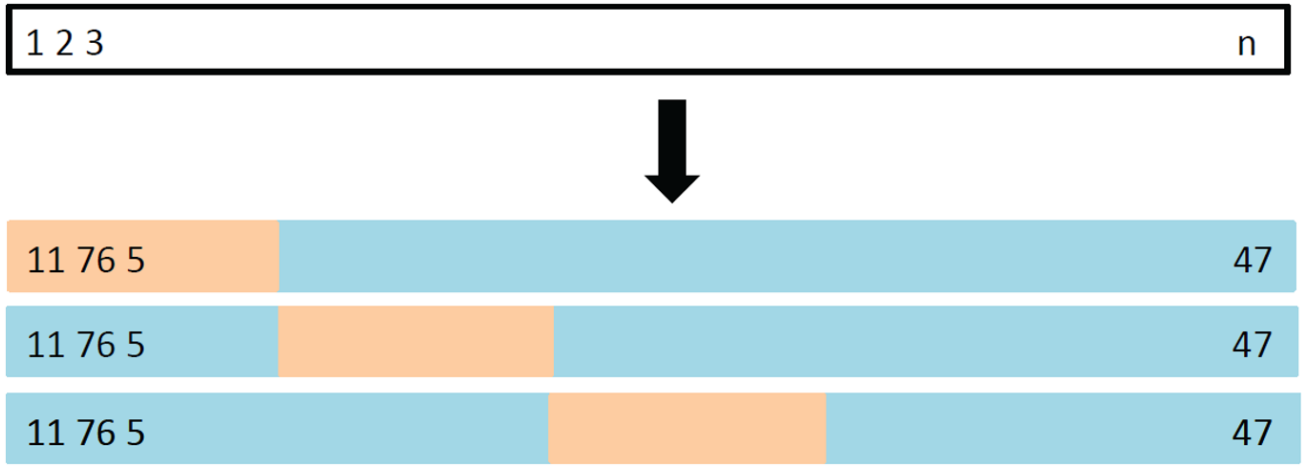
- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.
2. Leave out one part, k, and fit model $\hat{f}_k$ to the other $K-1$ parts.
3. Make predictions using $\hat{f}_k$ on left-out part k, and calculate MSE_k.
4. Repeat steps 1–3 so that each of the $k = 1...K$ parts get to be left-out once. Then *average* results: $MSE_{CV} = \frac{1}{K}\sum_k^K MSE_k$

K-fold Cross-validation

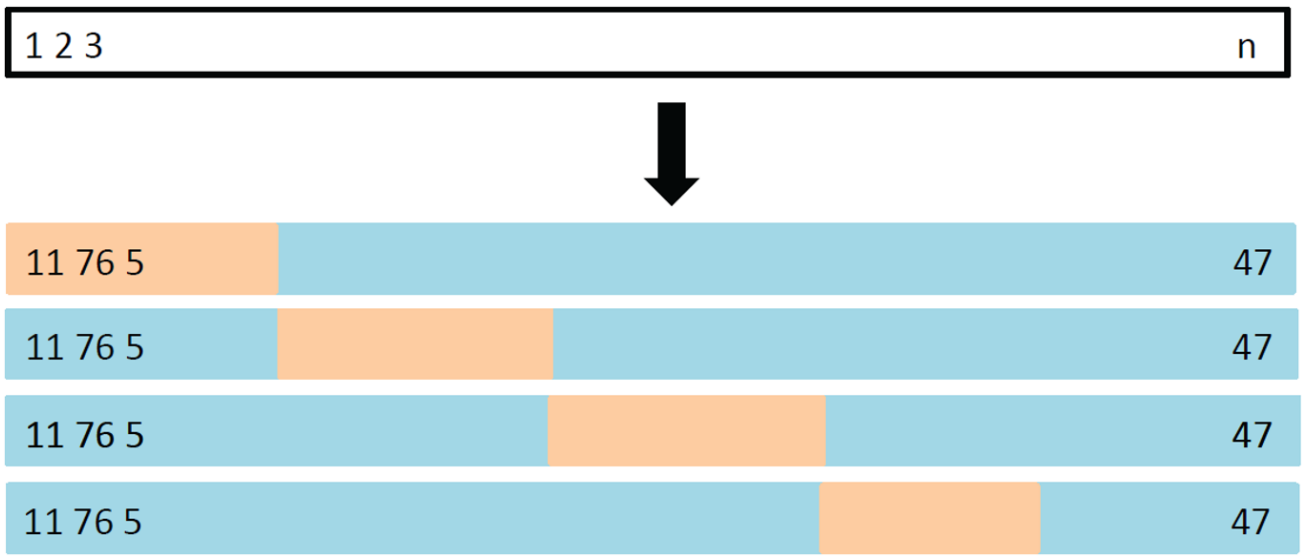
- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.
2. Leave out one part, k, and fit model $\hat{f}_k$ to the other $K-1$ parts.
3. Make predictions using $\hat{f}_k$ on left-out part k, and calculate MSE_k.
4. Repeat steps 1–3 so that each of the $k = 1...K$ parts get to be left-out once. Then *average* results: $MSE_{CV} = \frac{1}{K}\sum_k^K MSE_k$

K-fold Cross-validation

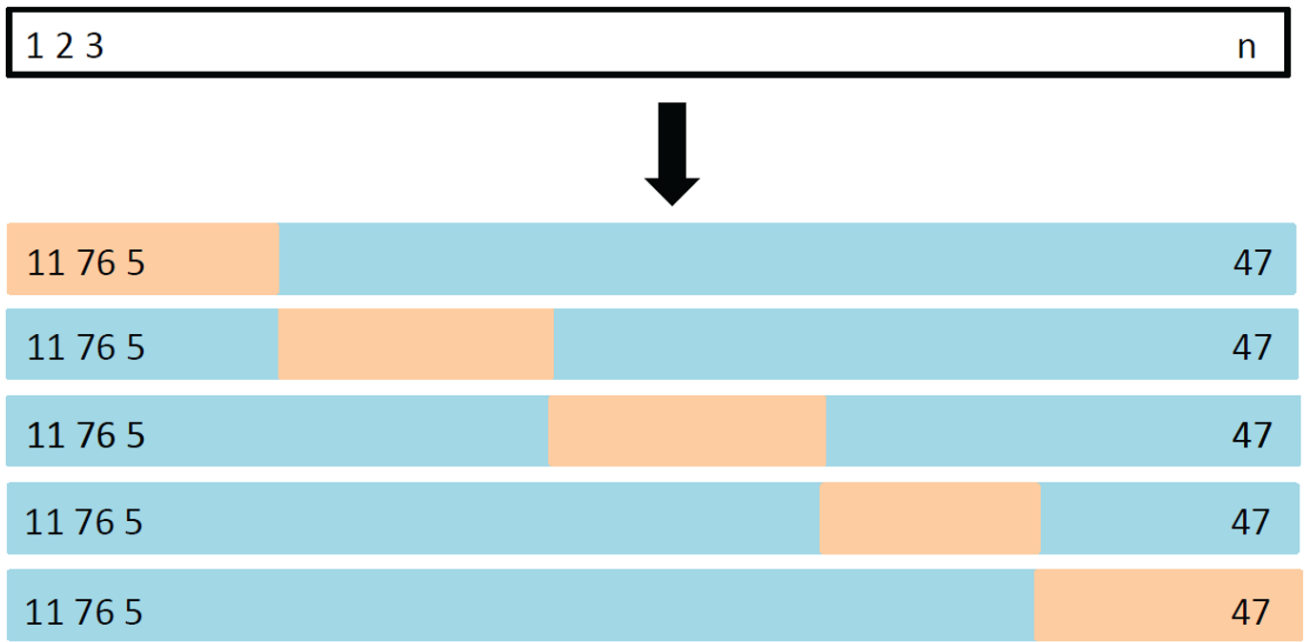
- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.
2. Leave out one part, k, and fit model $\hat{f}_k$ to the other $K-1$ parts.
3. Make predictions using $\hat{f}_k$ on left-out part k, and calculate MSE_k.
4. Repeat steps 1–3 so that each of the $k = 1...K$ parts get to be left-out once. Then *average* results: $MSE_{CV} = \frac{1}{K}\sum_k^K MSE_k$

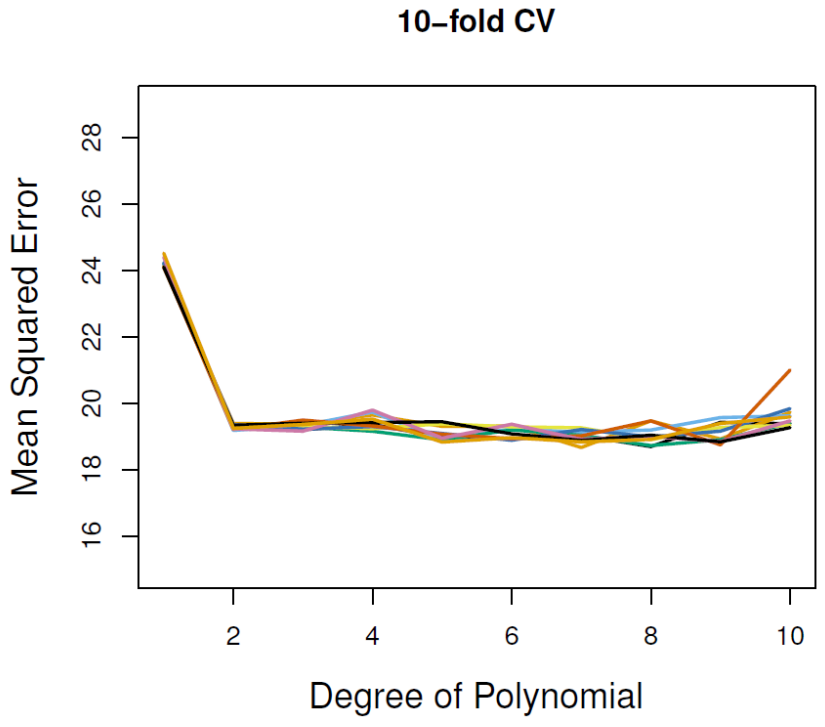
K-fold Cross-validation

- Slightly more sophisticated approach that address these issues.
- **Procedure**:



1. Randomly divide the data into K equal-sized parts.
2. Leave out one part, k, and fit model $\hat{f}_k$ to the other $K-1$ parts.
3. Make predictions using $\hat{f}_k$ on left-out part k, and calculate MSE_k.
4. Repeat steps 1–3 so that each of the $k = 1...K$ parts get to be left-out once. Then *average* results: $MSE_{CV} = \frac{1}{K}\sum_k^K MSE_k$

HP/MPG example revisited



$\Rightarrow$ Much *less variability*!

Considerations

- K-fold cross-validation provides more stable and less biased estimates of test error compared to validation-set approach.
- This improvement comes at the cost of computational time, which, depending on the size of one's data, could be a drawback.
- In practice, K usually set to 5 or 10 to balance these factors.

Recap & what's next

So, now we have a good grip of...

a. *What we mean* by good predictive performance
b. The concepts of *bias* and *variance*
c. *How we measure* prediction performance in practice.

Recap & what's next

So, now we have a good grip of...

a. *What we mean* by good predictive performance
b. The concepts of *bias* and *variance*
c. *How we measure* prediction performance in practice.

Let us now—finally—return to the challenge of:

- High-dimensionality
- Non-linearity

Recap & what's next

So, now we have a good grip of...

a. *What we mean* by good predictive performance
b. The concepts of *bias* and *variance*
c. *How we measure* prediction performance in practice.

Let us now—finally—return to the challenge of:

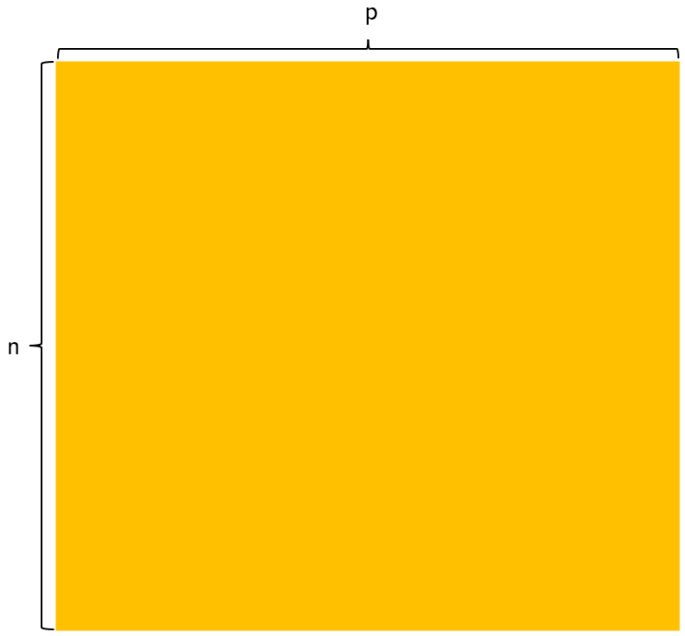
- High-dimensionality
- Non-linearity

While it is fairly clear why *non-linearity* is a problem, the problem of *high-dimensionality* is not as clear — let's begin there!

High-dimensionality

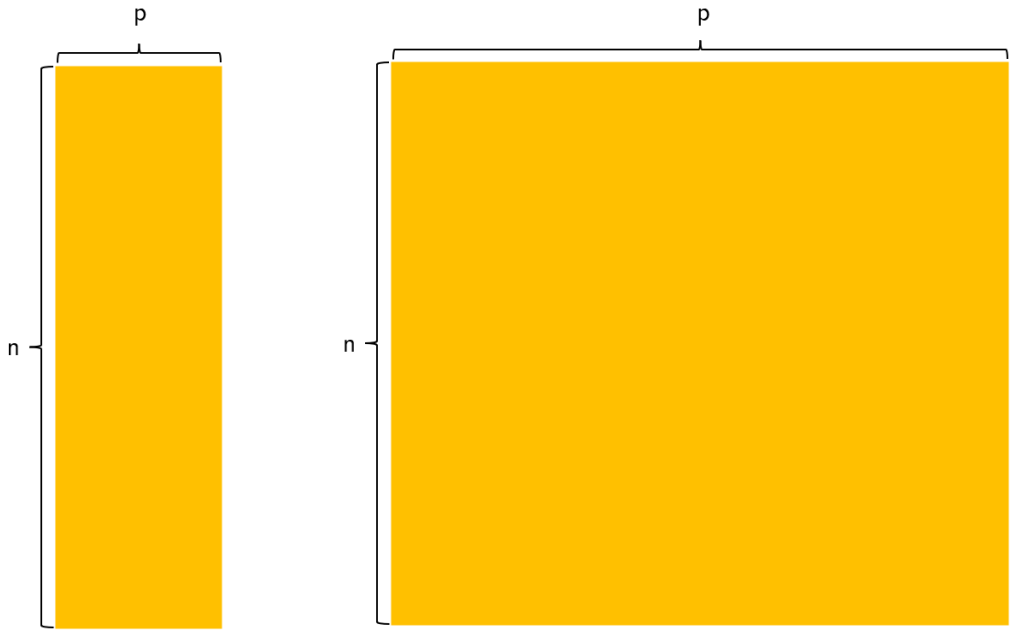
What is "high-dimensionality" ?

- High-dimensionality describes problems where the nr. of variables (p) in X is large relative to the nr. of observations (n).
  - $p \approx n$ or even $p \geq n$



What is "high-dimensionality" ?

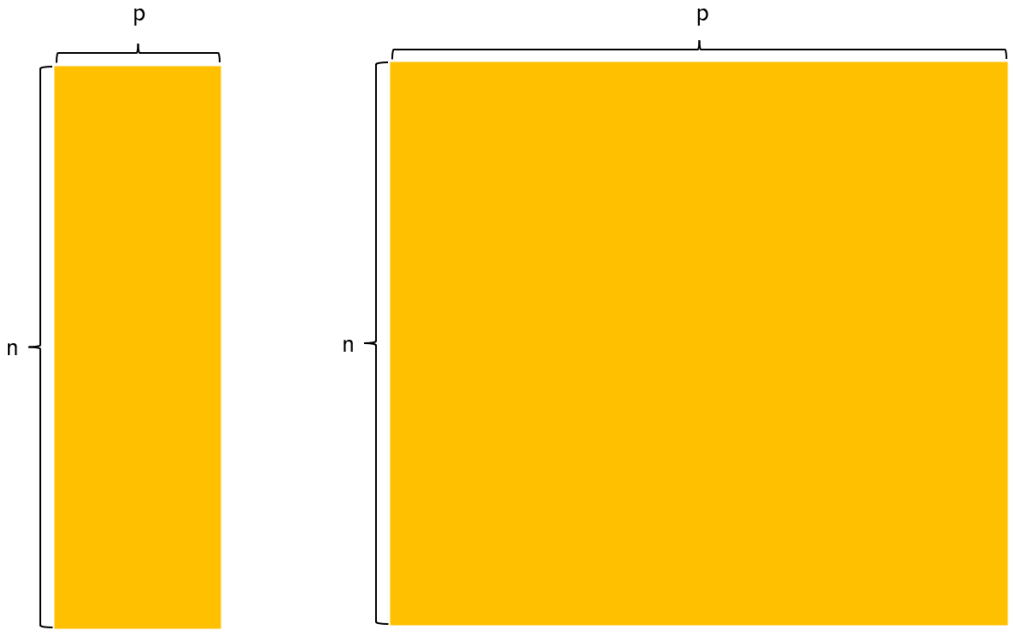
- High-dimensionality describes problems where the nr. of variables (p) in X is large relative to the nr. of observations (n).
  - $p \approx n$ or even $p \geq n$



- Traditional data sets: typically many more rows than columns
  - $n >> p$

What is "high-dimensionality" ?

- High-dimensionality describes problems where the nr. of variables (p) in X is large relative to the nr. of observations (n).
  - $p \approx n$ or even $p \geq n$



- Traditional data sets: typically many more rows than columns
  - $n >> p$
- In contrast — modern (e.g., *digital trace*) data sets often have a high-dimensional character.

So, what goes wrong in high dimensions?

- As the #*variables* (*parameters*) increase in relation to the #*observations* —OLS becomes increasingly likely to **overfit**

So, what goes wrong in high dimensions?

- As the #*variables* (*parameters*) increase in relation to the #*observations* —OLS becomes increasingly likely to **overfit**
- Logic — *more freedom* to *fine-tune parameters* to each data-point.

So, what goes wrong in high dimensions?

- As the #*variables* (*parameters*) increase in relation to the #*observations* —OLS becomes increasingly likely to **overfit**
- Logic — *more freedom* to *fine-tune parameters* to each data-point.
- In the extreme case of $p \geq n$, the standard OLS:
  - produces a **perfect fit** of the training data.
  - only fit n parameters. Rest are set to `NA`.
  - basically — one parameter tuned to each data point

So, what goes wrong in high dimensions?

- As the #*variables* (*parameters*) increase in relation to the #*observations* —OLS becomes increasingly likely to **overfit**
- Logic — *more freedom* to *fine-tune parameters* to each data-point.
- In the extreme case of $p \geq n$, the standard OLS:
  - produces a **perfect fit** of the training data.
  - only fit n parameters. Rest are set to `NA`.
  - basically — one parameter tuned to each data point
- This can easily be shown using *simulated data*—let's do so!

OLS when $n = p$:

Here, I simulate the following—independent—variables 9 times ($n = 2$):

$$X = N(\mu = 0, \sigma = 1)$$

$$Y = N(\mu = 0, \sigma = 1)$$

OLS when $n = p$:

Here, I simulate the following—independent—variables 9 times ($n = 2$):

$$X = N(\mu = 0, \sigma = 1)$$

$$Y = N(\mu = 0, \sigma = 1)$$

And in each, fit an OLS specified: $Y_i = \beta_0 + \beta_1 X_i$

OLS when $n = p$:

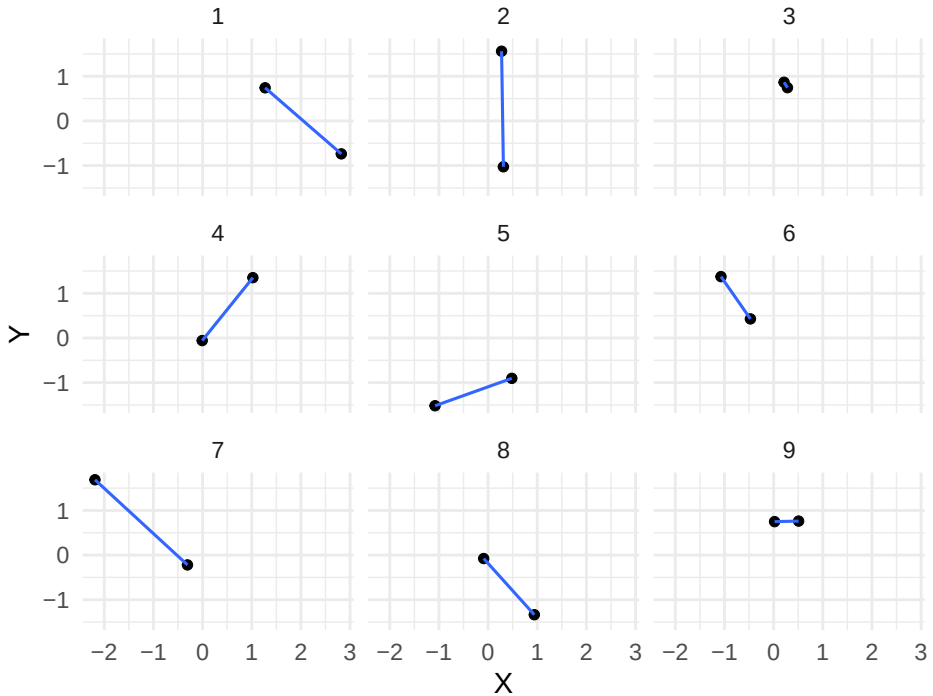
Here, I simulate the following—independent—variables 9 times ($n = 2$):

$$X = N(\mu = 0, \sigma = 1)$$

$$Y = N(\mu = 0, \sigma = 1)$$

And in each, fit an OLS specified: $Y_i = \beta_0 + \beta_1 X_i$ **—Results:**

- β_0 = first data point.
- β_1 perfectly intersects second point.



OLS when $n = p$:

Thus:

- In each of the 9 cases, OLS produces a **perfect fit** on the training data ($R^2 = 100\%$)

OLS when $n = p$:

Thus:

- In each of the 9 cases, OLS produces a **perfect fit** on the training data ($R^2 = 100\%$)
- On the other hand, the estimated **slope varies tremendously** between each sample
  - Very high variance!
  - Nonsensical predictions.

OLS when $n = p$:

Thus:

- In each of the 9 cases, OLS produces a **perfect fit** on the training data ($R^2 = 100\%$)
- On the other hand, the estimated **slope varies tremendously** between each sample
  - Very high variance!
  - Nonsensical predictions.

$\Rightarrow$ When we have as many parameters and as we have observations ($p = n$), OLS becomes a *too-flexible-model*, and **overfits** our data.

In conclusion: high-dimensionality

- Just as we can overfit when we have many parameters for one variable (polynomials) and few observations.
  - E.g., a 10-degree polynomial and 10 observations.
- We can also overfit our data when we include many parameters because of many variables and only have few observations.
  - E.g., 10 variables and 10 observations.

In conclusion: high-dimensionality

- Just as we can overfit when we have many parameters for one variable (polynomials) and few observations.
  - E.g., a 10-degree polynomial and 10 observations.
- We can also overfit our data when we include many parameters because of many variables and only have few observations.
  - E.g., 10 variables and 10 observations.
- In general, unless $n >> p$: a lot of variability in OLS $\rightarrow$ overfitting $\rightarrow$ poor predictions on new data.

How to "solve" the high-dimensionality problem?

A general & important idea in ML: **regularization**

How to "solve" the high-dimensionality problem?

A general & important idea in ML: **regularization**

- Basic idea — sacrifice some increase in bias for a (much greater) reduction in variance.

How to "solve" the high-dimensionality problem?

A general & important idea in ML: **regularization**

- Basic idea — sacrifice some increase in bias for a (much greater) reduction in variance.
- That is — make performance slightly worse on training set in order to get better *generalized performance*.

How to "solve" the high-dimensionality problem?

A general & important idea in ML: **regularization**

- Basic idea — sacrifice some increase in bias for a (much greater) reduction in variance.
- That is — make performance slightly worse on training set in order to get better *generalized performance*.
- How do we regularize $\hat{f}$?
  - Varies by method.
  - Linear models: Shrink parameters towards 0.

How to "solve" the high-dimensionality problem?

A general & important idea in ML: **regularization**

- Basic idea — sacrifice some increase in bias for a (much greater) reduction in variance.
- That is — make performance slightly worse on training set in order to get better *generalized performance*.
- How do we regularize $\hat{f}$?
  - Varies by method.
  - Linear models: Shrink parameters towards 0.
- We will consider the most popular linear regularizer:
  - Ridge regression
  - Lasso regression (appendix)

Ridge regression

Recall, **OLS** finds parameters $\hat{\beta}$ by *minimizing* the *residual sum of squares* (RSS)[2]:

$$RSS = \sum_{i=1}^{n} \Big(\underbrace{y_i}_{Observed} \underbrace{- \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}}_{Predicted} \Big)^2$$

[2] Equivalent to *squared loss*

Ridge regression

Recall, **OLS** finds parameters $\hat{\beta}$ by *minimizing* the *residual sum of squares* (RSS)[2]:

$$RSS = \sum_{i=1}^{n}\Big(\underbrace{y_i}_{Observed} \underbrace{- \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}}_{Predicted} \Big)^2$$

Ridge extends upon OLS by also including a penalty term in its loss function:

$$\underbrace{\sum_{i=1}^{n}\Big(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\Big)^2}_{RSS} + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{Penalty}$$

[2]Equivalent to *squared loss*

Ridge regression

Recall, **OLS** finds parameters $\hat{\beta}$ by *minimizing* the *residual sum of squares* (RSS)[2]:

$$RSS = \sum_{i=1}^{n}\Big(\underbrace{y_i}_{Observed} \underbrace{- \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}}_{Predicted} \Big)^2$$

Ridge extends upon OLS by also including a penalty term in its loss function:

$$\underbrace{\sum_{i=1}^{n}\Big(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\Big)^2}_{RSS} + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{Penalty}$$

Because we want to *minimize* this expression $\rightarrow$ penalty term has the function of **punishing solutions** of $\hat{\beta}$ where the *squared sum* of $\hat{\beta}_j$ is **big**.

[2] Equivalent to *squared loss*

Ridge regression

Recall, **OLS** finds parameters $\hat{\beta}$ by *minimizing* the *residual sum of squares* (RSS)[2]:

$$RSS = \sum_{i=1}^{n}\Big(\underbrace{y_i}_{Observed} \underbrace{- \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}}_{Predicted}\Big)^2$$

Ridge extends upon OLS by also including a penalty term in its loss function:

$$\underbrace{\sum_{i=1}^{n}\Big(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\Big)^2}_{RSS} + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{Penalty}$$

Because we want to *minimize* this expression $\rightarrow$ penalty term has the function of **punishing solutions** of $\hat{\beta}$ where the *squared sum* of $\hat{\beta}_j$ is **big**.

- λ (set by researcher) controls the severity of this penalty. Extremes:
  - $\lambda = 0 \rightarrow \hat{\beta}_{OLS}$
  - $\lambda = \infty \rightarrow \hat{\beta} = 0$

[2] Equivalent to *squared loss*

Why should this help? Intuitively:

How standard OLS works:

- When optimizing its parameters, the coefficient for a variable X_1 ($\hat{\beta}_{X_1}$) gets a **larger value** if X_1 is predictive of Y.
- That is: $\hat{\beta}_{X_1}$ *increased* if it helps reduce the RSS (on training set).

Why should this help? Intuitively:

How standard OLS works:

- When optimizing its parameters, the coefficient for a variable X_1 ($\hat{\beta}_{X_1}$) gets a **larger value** if X_1 is predictive of Y.
- That is: $\hat{\beta}_{X_1}$ *increased* if it helps reduce the RSS (on training set).

The penalty term, instead:

- **Decreases the value** for $\hat{\beta}$'s, and thus reduces the fit on the training data (increases bias), but hopefully improves the fit on the test data (reduces variance).

Why should this help? Intuitively:

How standard OLS works:

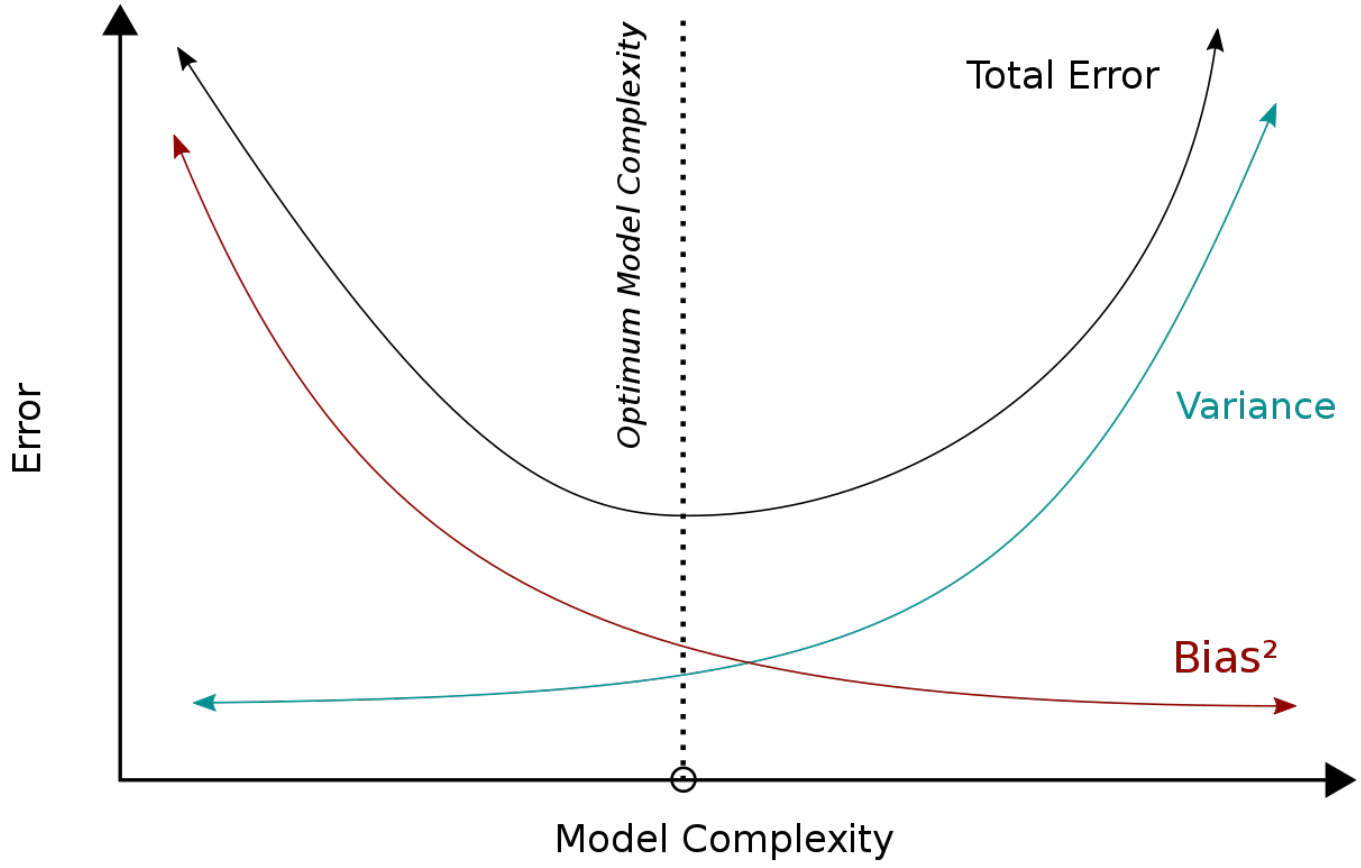
- When optimizing its parameters, the coefficient for a variable X_1 ($\hat{\beta}_{X_1}$) gets a **larger value** if X_1 is predictive of Y.
- That is: $\hat{\beta}_{X_1}$ *increased* if it helps reduce the RSS (on training set).

The penalty term, instead:

- **Decreases the value** for $\hat{\beta}$'s, and thus reduces the fit on the training data (increases bias), but hopefully improves the fit on the test data (reduces variance).

$\Rightarrow$ In other words — the penalty reduces overfitting.

Recall: Bias-Variance Trade-off



Regularization (ridge) moves us from the right of this plot to the left.

- *Big reduction* in variance
- *Small increase* in bias

How to think about the λ parameter?

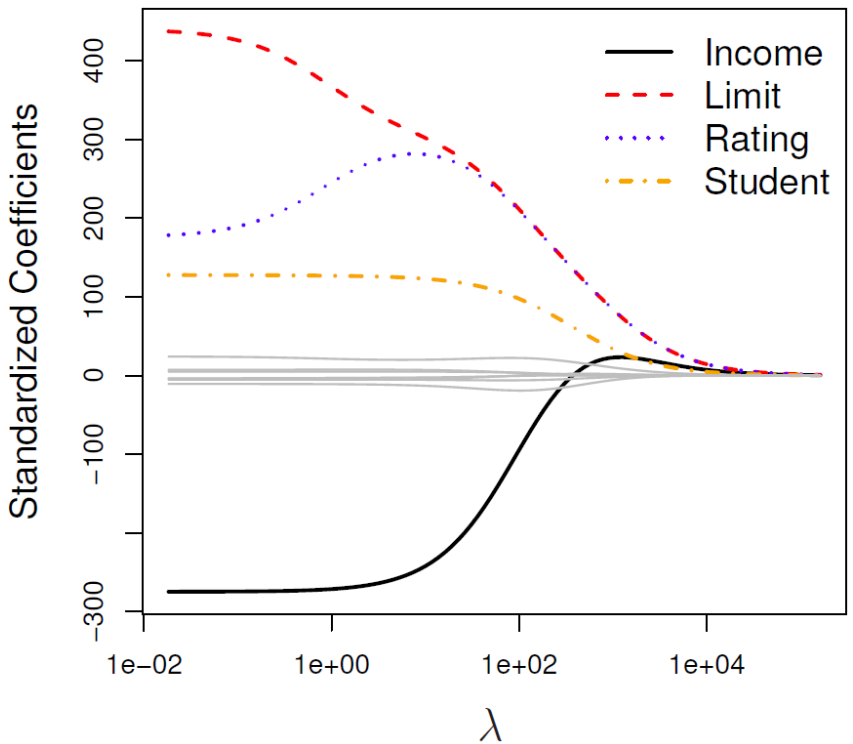
- As mentioned before, λ is a constant set by the researcher and it controls the **degree of regularization**.
- Thus—also controls how much *variance* is reduced (*bias* increased).
- How do we decide on which λ value to use?
  - You might have guessed: **cross-validation**!

For some intuition about how $\hat{\beta}_{ridge}$ changes with λ...

... let's consider another example in ISL (Fig. 6.4)

- Details of the example does not matter. This is just for intuition.

Fig. 6.4 shows how β changes—for different variables—as we vary λ.



As we increase λ—the value of individual coefficients decreases.[3]

[3] But note exceptions.

OBS: Standardize X!

Something to note—we must standardize our data prior to regularization.

OBS: Standardize X!

Something to note—we must standardize our data prior to regularization.

Why?

If variables are on different scales—e.g., *income* on 10^5 scale and *height* 10^2 scale—then regularization will punish the variables in *larger scales less*.

OBS: Standardize X!

Something to note—we must standardize our data prior to regularization.

Why?

If variables are on different scales—e.g., *income* on 10^5 scale and *height* 10^2 scale—then regularization will punish the variables in *larger scales less*.

- Such variables naturally obtain smaller coefficients because of scale (easy to demonstrate: see appendix).

Pros and cons (ridge)

Pros

- Enables the "linear paradigm" (OLS) to work when $p \geq n$.
- Often improves predictability in high-dimensional contexts.
- Improves interpretability—in the sense that fewer covariates have $>> 0$ weights.

Pros and cons (ridge)

Pros

- Enables the "linear paradigm" (OLS) to work when $p \geq n$.
- Often improves predictability in high-dimensional contexts.
- Improves interpretability—in the sense that fewer covariates have $>> 0$ weights.

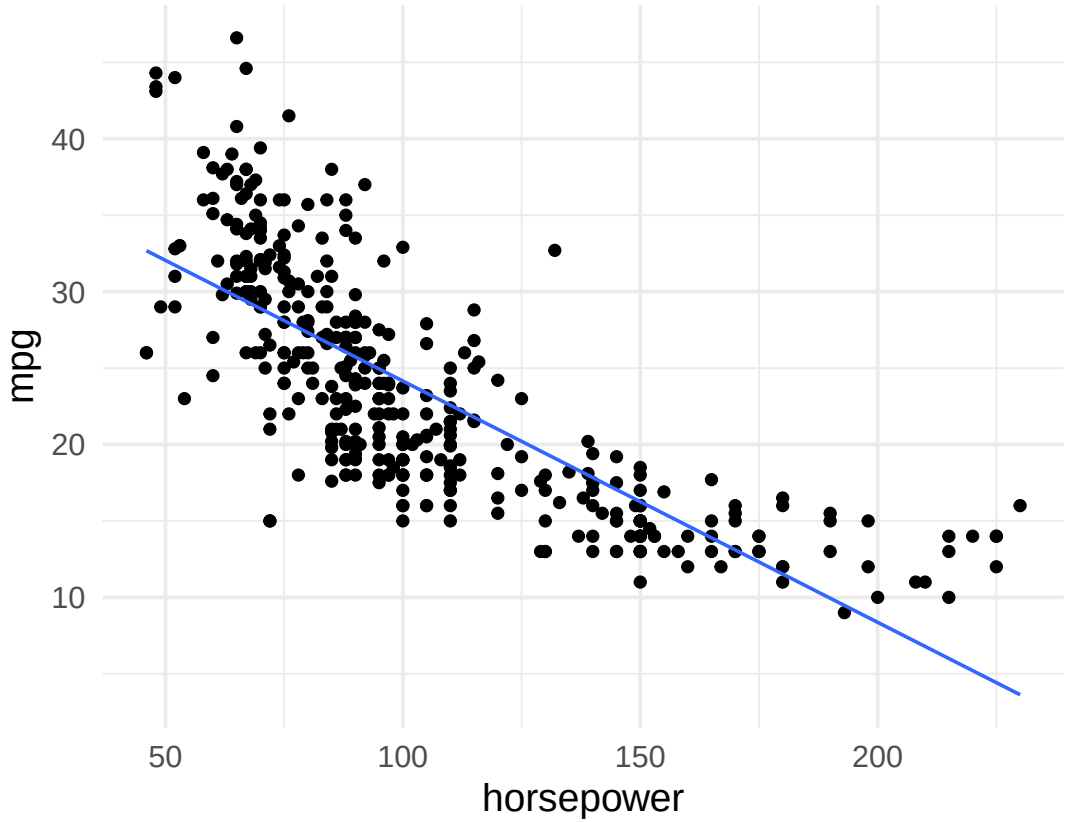
Cons

- Reduced "inferential strength"
  - No standard error.
  - Biased estimates of $\hat{\beta}$.
- Does not, by itself, address *non-linearity*.

Non-linearity

The issue of non-linearity

When the *true f* is *non-linear*, the standard OLS—which only relies on the *original variables*—will "underfit" the data.



No matter the sample size, $\hat{f}_{linear}$ will not approximate the true f well.

Polynomial regression

The *traditional* way of extending OLS to capture *non-linearity*.

Polynomial regression

The *traditional* way of extending OLS to capture *non-linearity*.

Basic idea: Assuming a single input variable x_1, we simply replace the standard linear regression model:

$$y_i = \beta_0 + \beta_1 x_{1i} + \epsilon_i$$

with a polynomial function (of degree d):

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{1i}^2 + \cdots + \beta_d x_{1i}^d + \epsilon_i$$

Polynomial regression

The *traditional* way of extending OLS to capture *non-linearity*.

Basic idea: Assuming a single input variable x_1, we simply replace the standard linear regression model:

$$y_i = \beta_0 + \beta_1 x_{1i} + \epsilon_i$$

with a polynomial function (of degree d):

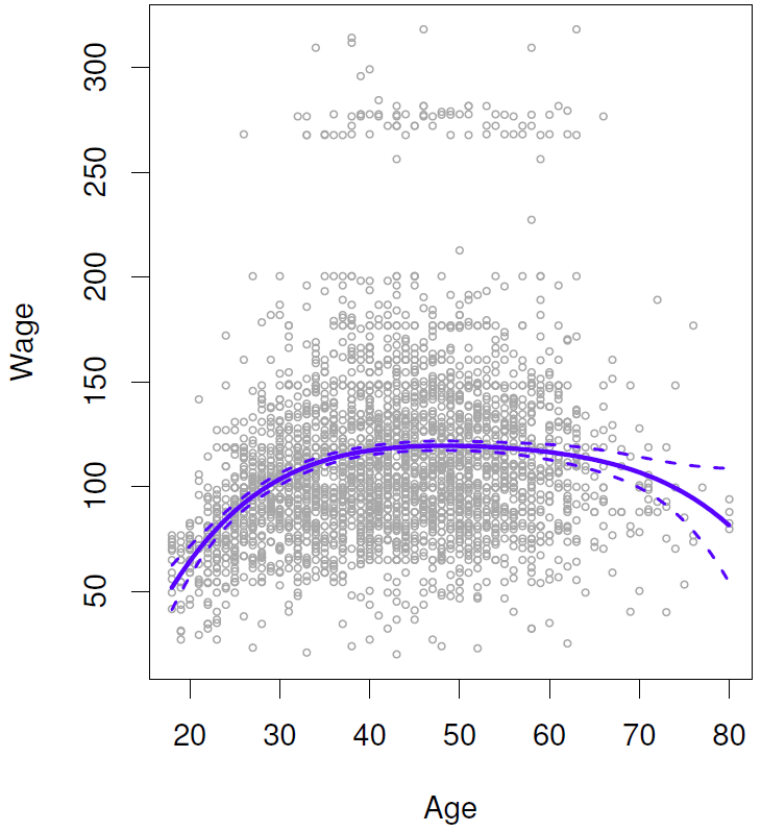
$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{1i}^2 + \cdots + \beta_d x_{1i}^d + \epsilon_i$$

That is: simply create $(d-1)$ extra variables which are *transformations* of the original variables, and include them in our model.

Illustration (ISL, Fig. 7.1)

4-degree polynomial regression fit to `Wage` data.

- Y: `Wage`
- X: `Age`



Pros and cons

Pros

- Can produce very non-linear curves (as d is increased).
- Maintains *interpretability* and *inferential strengths* of standard OLS.

Pros and cons

Pros

- Can produce very non-linear curves (as d is increased).
- Maintains *interpretability* and *inferential strengths* of standard OLS.

Cons

- Easily gets overly flexible with higher d's (usually max: 3-4).
- We must manually specify d for each variable — gets tricky and burdensome for higher dimensions.
- Does not address interactions between variables.

Pros and cons

Pros

- Can produce very non-linear curves (as d is increased).
- Maintains *interpretability* and *inferential strengths* of standard OLS.

Cons

- Easily gets overly flexible with higher d's (usually max: 3-4).
- We must manually specify d for each variable — gets tricky and burdensome for higher dimensions.
- Does not address interactions between variables.

Next—consider class of methods that address these limitations.