

# (Some of) Machine Learning

EC 421, Set 12

Edward Rubin

# Prologue

# Schedule

## Today

A one-lecture introduction to machine-learning methods

## Upcoming

The end is near.

# Prediction: What's the goal?

# Prediction: What's the goal?

## What's different?

Machine-learning methods *typically* focus on **prediction**. What's different?

# Prediction: What's the goal?

## What's different?

Machine-learning methods *typically* focus on **prediction**. What's different?

Up to this point, we've focused on causal **identification/inference** of  $\beta$ , *i.e.*,

$$\mathbf{Y}_i = \mathbf{X}_i\beta + u_i$$

meaning we want an unbiased (consistent) and precise estimate  $\hat{\beta}$ .

# Prediction: What's the goal?

## What's different?

Machine-learning methods *typically* focus on **prediction**. What's different?

Up to this point, we've focused on causal **identification/inference** of  $\beta$ , i.e.,

$$\mathbf{Y}_i = \mathbf{X}_i\beta + u_i$$

meaning we want an unbiased (consistent) and precise estimate  $\hat{\beta}$ .

With **prediction**, we shift our focus to accurately estimating outcomes.

In other words, how can we best construct  $\hat{\mathbf{Y}}_i$ ?

# Prediction: What's the goal?

... so?

So we want "nice"-performing estimates  $\hat{y}$  instead of  $\hat{\beta}$ .

Q Can't we just use the same methods (*i.e.*, OLS)?

# Prediction: What's the goal?

... so?

So we want "nice"-performing estimates  $\hat{y}$  instead of  $\hat{\beta}$ .

**Q** Can't we just use the same methods (*i.e.*, OLS)?

**A** It depends.

# Prediction: What's the goal?

... so?

So we want "nice"-performing estimates  $\hat{y}$  instead of  $\hat{\beta}$ .

**Q** Can't we just use the same methods (*i.e.*, OLS)?

**A** It depends. How well does your **linear**-regression model approximate the underlying data? (And how do you plan to select your model?)

# Prediction: What's the goal?

... so?

So we want "nice"-performing estimates  $\hat{y}$  instead of  $\hat{\beta}$ .

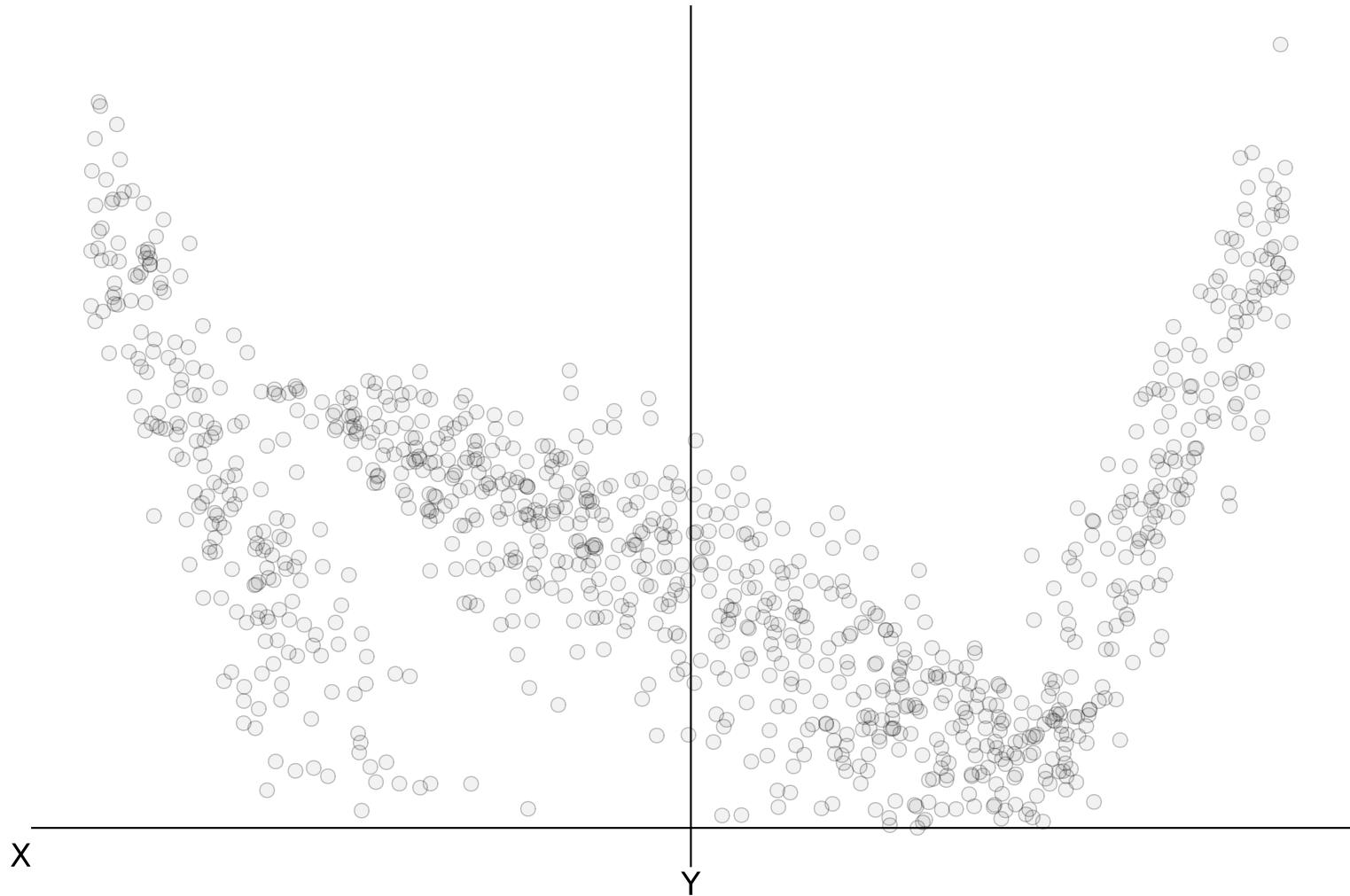
**Q** Can't we just use the same methods (*i.e.*, OLS)?

**A** It depends. How well does your **linear**-regression model approximate the underlying data? (And how do you plan to select your model?)

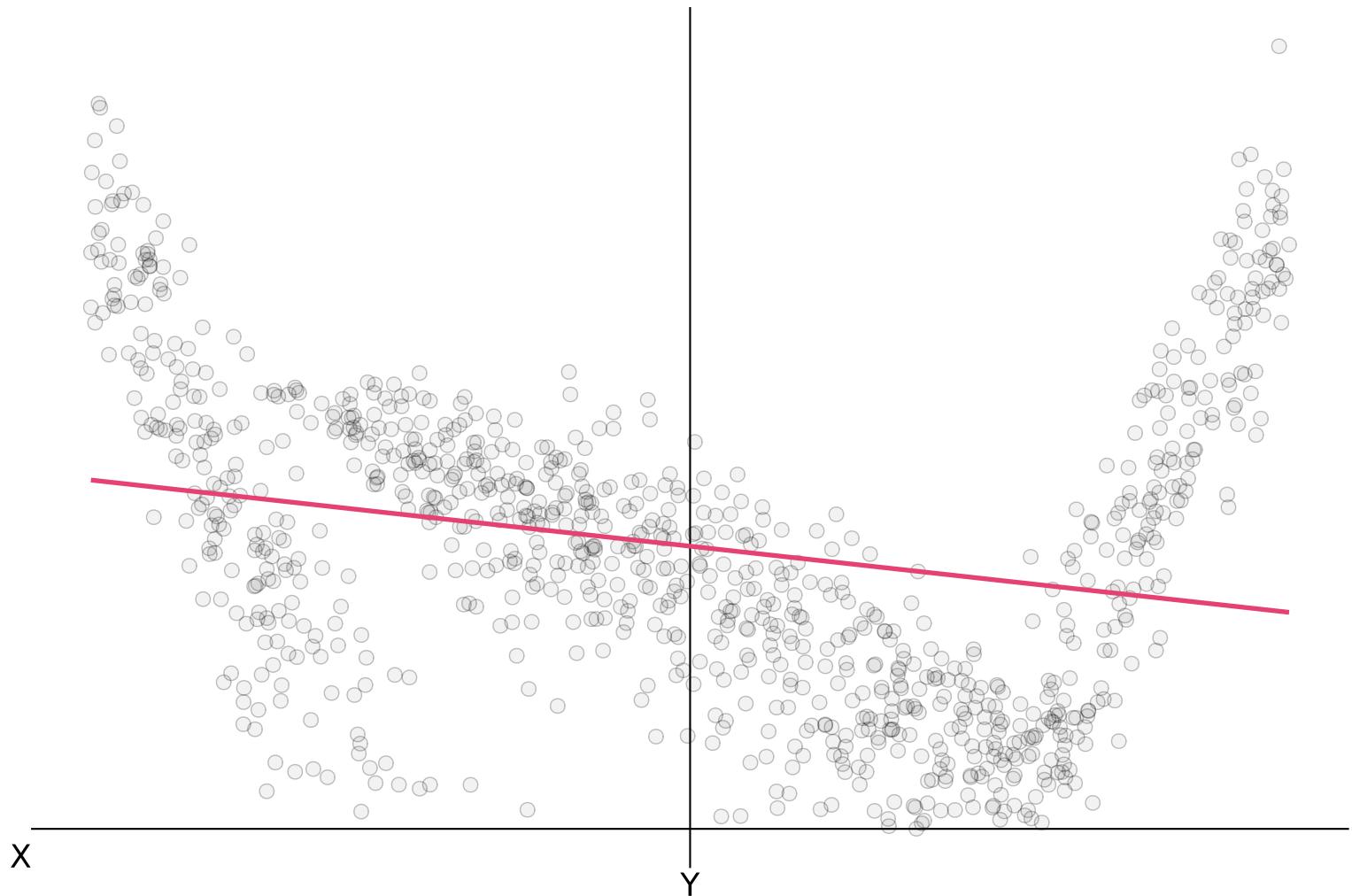
*Recall* Least-squares regression is a great **linear** estimator and predictor.

Data data be tricky<sup>†</sup>—as can understanding many relationships.

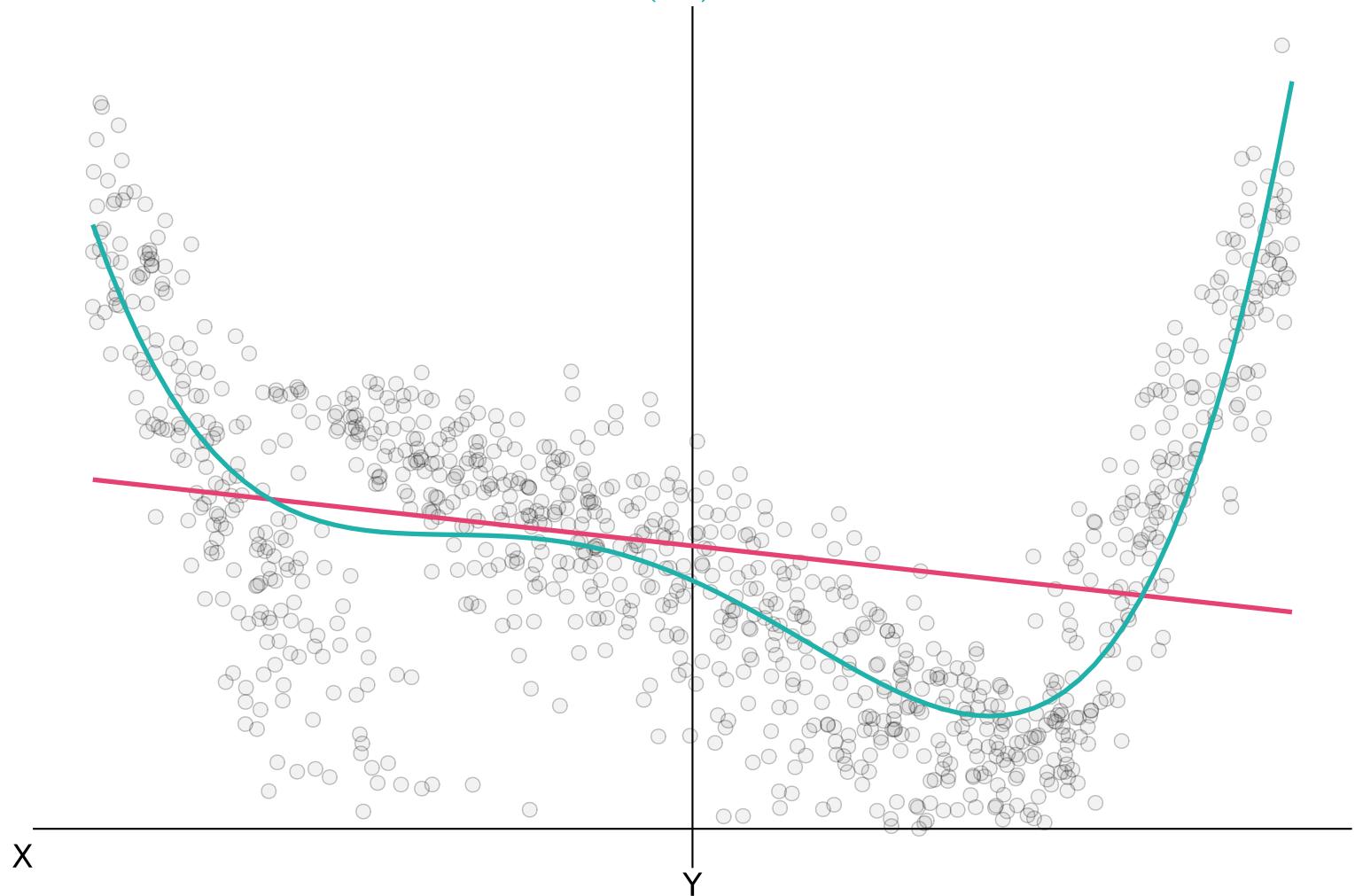
<sup>†</sup> "Tricky" might mean nonlinear... or many other things...



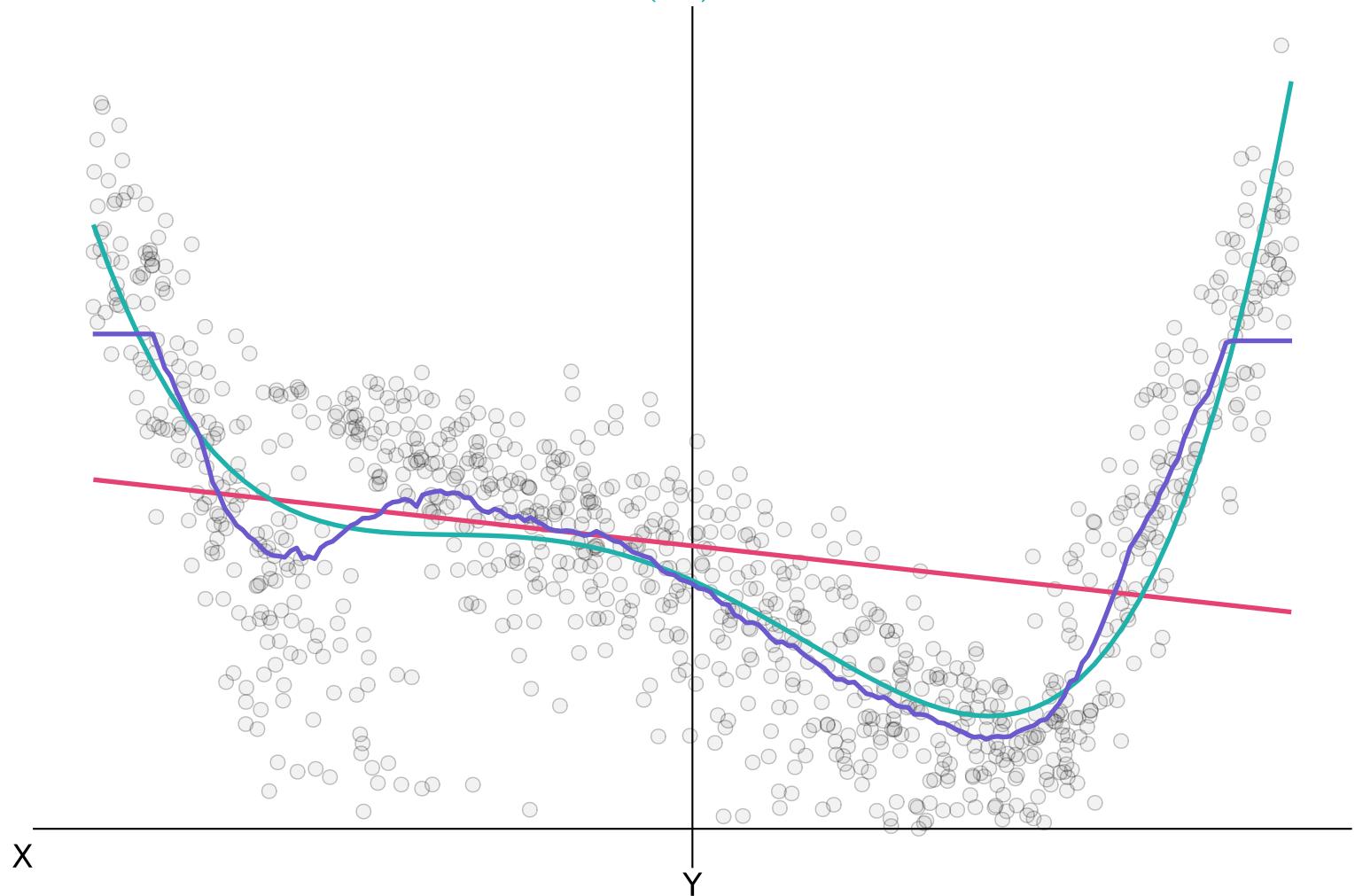
## Linear regression



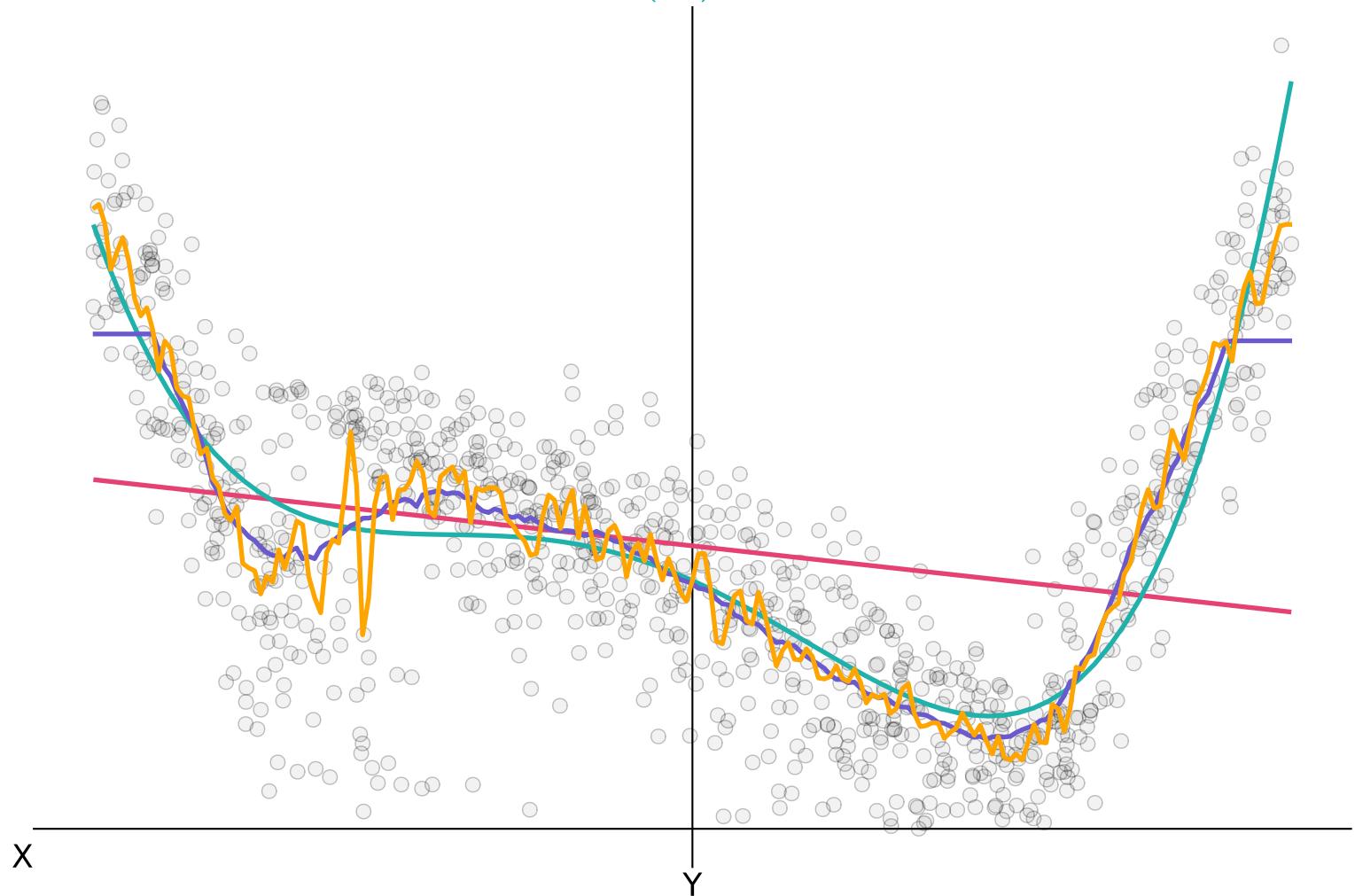
## Linear regression, linear regression ( $x^4$ )



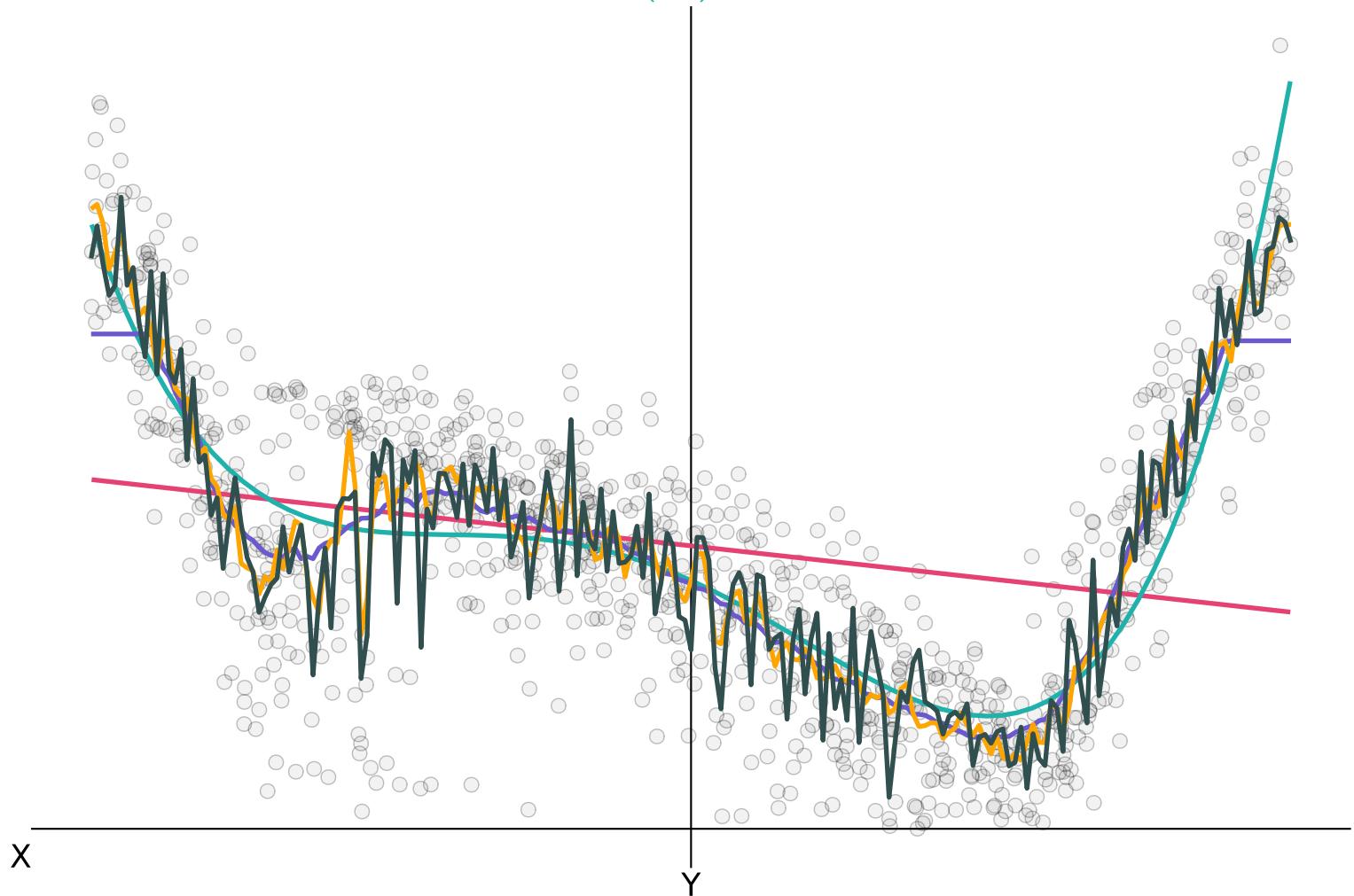
Linear regression, linear regression ( $x^4$ ), KNN (100)



Linear regression, linear regression ( $x^4$ ), KNN (100), KNN (10)



Linear regression, linear regression ( $x^4$ ), KNN (100), KNN (10), random forest



*Note* That example only had one predictor...

# What's the goal?

## Tradeoffs

In prediction, we constantly face many tradeoffs, *e.g.*,

- **flexibility** and **parametric structure** (and interpretability)
- performance in **training** and **test** samples
- **variance** and **bias**

# What's the goal?

## Tradeoffs

In prediction, we constantly face many tradeoffs, e.g.,

- **flexibility** and **parametric structure** (and interpretability)
- performance in **training** and **test** samples
- **variance** and **bias**

As your economic training should have predicted, in each setting, we need to **balance the additional benefits and costs** of adjusting these tradeoffs.

# What's the goal?

## Tradeoffs

In prediction, we constantly face many tradeoffs, e.g.,

- **flexibility** and **parametric structure** (and interpretability)
- performance in **training** and **test** samples
- **variance** and **bias**

As your economic training should have predicted, in each setting, we need to **balance the additional benefits and costs** of adjusting these tradeoffs.

Many machine-learning (ML) techniques/algorithms are crafted to optimize with these tradeoffs, but the practitioner (you) still needs to be careful.

# What's the goal?

There are many reasons to step outside the world of linear regression...

# What's the goal?

There are many reasons to step outside the world of linear regression...

## **Multi-class** classification problems

- Rather than {0,1}, we need to classify  $y_i$  into 1 of K classes
- *E.g.*, ER patients: {heart attack, drug overdose, stroke, nothing}

# What's the goal?

There are many reasons to step outside the world of linear regression...

## **Multi-class** classification problems

- Rather than {0,1}, we need to classify  $y_i$  into 1 of K classes
- *E.g.*, ER patients: {heart attack, drug overdose, stroke, nothing}

## **Text analysis** and **image recognition**

- Comb through sentences (pixels) to glean insights from relationships
- *E.g.*, detect sentiments in tweets or roof-top solar in satellite imagery

# What's the goal?

There are many reasons to step outside the world of linear regression...

## **Multi-class** classification problems

- Rather than {0,1}, we need to classify  $y_i$  into 1 of K classes
- *E.g.*, ER patients: {heart attack, drug overdose, stroke, nothing}

## **Text analysis** and **image recognition**

- Comb through sentences (pixels) to glean insights from relationships
- *E.g.*, detect sentiments in tweets or roof-top solar in satellite imagery

## **Unsupervised learning**

- You don't know groupings, but you think there are relevant groups
- *E.g.*, classify spatial data into groups



**Stanford University (Stanford, CA ) researchers have developed a deep-learning algorithm that can evaluate chest X-ray images for signs of disease at a level exceeding practicing radiologists.**



# Parking Lot Vehicle Detection Using Deep Learning

THE  
NEW YORKER

A REPORTER AT LARGE OCTOBER 14, 2019 ISSUE

# The Next Word |

*Where will predictive text take us?*

Text by John Seabrook



Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
Microsoft	94.0%	79.2%	100%	98.3%	20.8%
FACE++	99.3%	65.5%	99.2%	94.0%	33.8%
IBM	88.0%	65.3%	99.7%	92.9%	34.4%



Flexibility is huge, but we still want to avoid overfitting.

# Statistical learning

What is it good for?

# Statistical learning

## What is it good for?

A lot of things.

# Statistical learning

## What is it good for?

A lot of things. We tend to break statistical-learning into two(-ish) classes:

1. **Supervised learning** builds ("learns") a statistical model for predicting an **output ( $y$ )** given a set of **inputs ( $x_1, \dots, x_p$ )**,

# Statistical learning

## What is it good for?

A lot of things. We tend to break statistical-learning into two(-ish) classes:

1. **Supervised learning** builds ("learns") a statistical model for predicting an **output** ( $\mathbf{y}$ ) given a set of **inputs** ( $\mathbf{x}_1, \dots, \mathbf{x}_p$ ), i.e., we want to build a model/function  $f$

$$\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_p)$$

that accurately describes  $\mathbf{y}$  given some values of  $\mathbf{x}_1, \dots, \mathbf{x}_p$ .

# Statistical learning

## What is it good for?

A lot of things. We tend to break statistical-learning into two(-ish) classes:

1. **Supervised learning** builds ("learns") a statistical model for predicting an **output** ( $\mathbf{y}$ ) given a set of **inputs** ( $\mathbf{x}_1, \dots, \mathbf{x}_p$ ), i.e., we want to build a model/function  $f$

$$\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_p)$$

that accurately describes  $\mathbf{y}$  given some values of  $\mathbf{x}_1, \dots, \mathbf{x}_p$ .

2. **Unsupervised learning** learns relationships and structure using only **inputs** ( $x_1, \dots, x_p$ ) without any *supervising* output

# Statistical learning

## What is it good for?

A lot of things. We tend to break statistical-learning into two(-ish) classes:

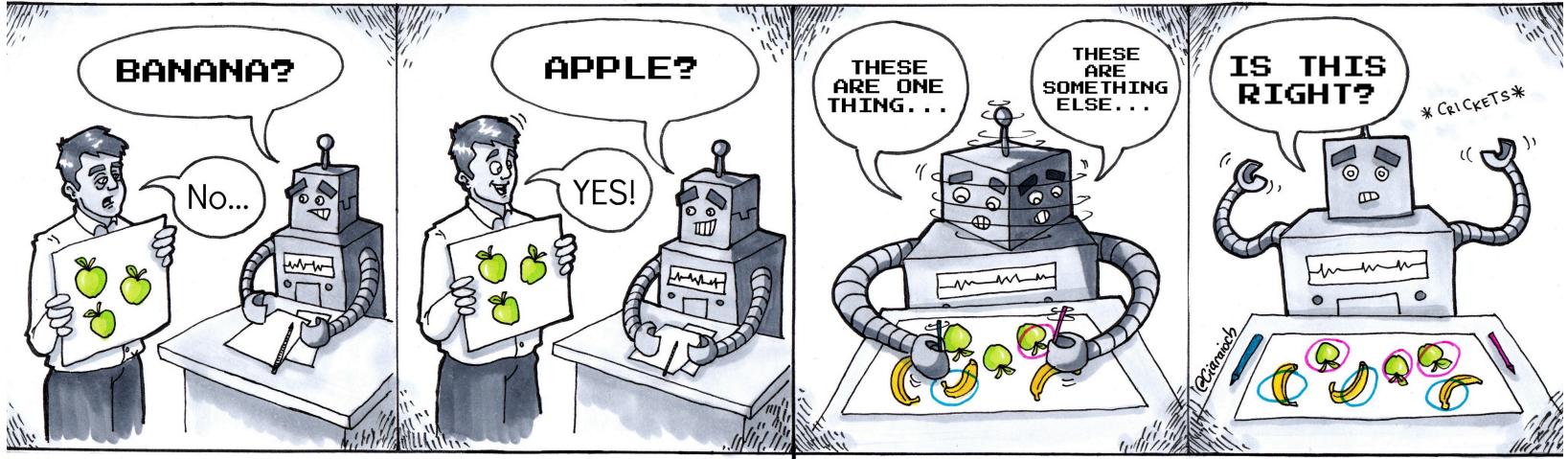
1. **Supervised learning** builds ("learns") a statistical model for predicting an **output** ( $\mathbf{y}$ ) given a set of **inputs** ( $\mathbf{x}_1, \dots, \mathbf{x}_p$ ), i.e., we want to build a model/function  $f$

$$\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_p)$$

that accurately describes  $\mathbf{y}$  given some values of  $\mathbf{x}_1, \dots, \mathbf{x}_p$ .

2. **Unsupervised learning** learns relationships and structure using only **inputs** ( $x_1, \dots, x_p$ ) without any *supervising output*—letting the data "speak for itself."

**Semi-supervised learning** falls somewhere between these supervised and unsupervised learning—generally applied to supervised tasks when labeled **outputs** are incomplete.



## Supervised Learning

## Unsupervised Learning

Source

# Statistical learning

## Output

We tend to further break **supervised learning** into two groups, based upon the **output** (the **outcome** we want to predict):

# Statistical learning

## Output

We tend to further break **supervised learning** into two groups, based upon the **output** (the **outcome** we want to predict):

1. **Classification tasks** for which the values of **y** are discrete categories  
*E.g., race, sex, loan default, hazard, disease, flight status*
2. **Regression tasks** in which **y** takes on continuous, numeric values.  
*E.g., price, arrival time, number of emails, temperature*

*Note<sub>1</sub>* The use of *regression* differs from our use of *linear regression*.

# Statistical learning

## Output

We tend to further break **supervised learning** into two groups, based upon the **output** (the **outcome** we want to predict):

1. **Classification tasks** for which the values of **y** are discrete categories  
*E.g., race, sex, loan default, hazard, disease, flight status*
2. **Regression tasks** in which **y** takes on continuous, numeric values.  
*E.g., price, arrival time, number of emails, temperature*

*Note<sub>1</sub>* The use of *regression* differs from our use of *linear regression*.

*Note<sub>2</sub>* Don't get tricked: Not all numbers represent continuous, numerical values—e.g., zip codes, industry codes, social security numbers.<sup>†</sup>

<sup>†</sup> **Q** Where would you put responses to 5-item Likert scales?

# Statistical learning

## The goal

As defined before, we want to *learn* a model to understand our data.

# Statistical learning

## The goal

As defined before, we want to *learn* a model to understand our data.

1. Take our (numeric) output  $\mathbf{y}$ .
2. Imagine there is a function  $f$  that takes inputs  $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_p$  and maps them, plus a random, mean-zero error term  $\varepsilon$ , to the output.

$$\mathbf{y} = f(\mathbf{X}) + \varepsilon$$

# Statistical learning

## Learning from $\hat{f}$

There are two main reasons we want to learn about  $\hat{f}$

1. **Causal inference settings** How do changes in  $\mathbf{X}$  affect  $\mathbf{y}$ ?

What we've done all quarter.

# Statistical learning

## Learning from $\hat{f}$

There are two main reasons we want to learn about  $f$

1. **Causal inference settings** How do changes in  $X$  affect  $y$ ?

What we've done all quarter.

2. **Prediction problems** Predict  $y$  using our estimated  $f$ , i.e.,

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

our *black-box setting* where we care less about  $f$  than  $\hat{y}$ .<sup>†</sup>

<sup>†</sup> You shouldn't actually treat your prediction methods as total black boxes.

# Statistical learning

## Learning from $\hat{f}$

There are two main reasons we want to learn about  $f$

1. **Causal inference settings** How do changes in  $X$  affect  $y$ ?

What we've done all quarter.

2. **Prediction problems** Predict  $y$  using our estimated  $f$ , i.e.,

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

our *black-box setting* where we care less about  $f$  than  $\hat{y}$ .<sup>†</sup>

Similarly, in causal-inference settings, we don't particularly care about  $\hat{y}$ .

<sup>†</sup> You shouldn't actually treat your prediction methods as total black boxes.

# Statistical learning

## Prediction errors

As tends to be the case in life, you will make errors in predicting  $\hat{\mathbf{y}}$ .

The accuracy of  $\hat{\mathbf{y}}$  depends upon **two errors**:

# Statistical learning

## Prediction errors

As tends to be the case in life, you will make errors in predicting  $\hat{\mathbf{y}}$ .

The accuracy of  $\hat{\mathbf{y}}$  depends upon **two errors**:

1. **Reducible error** The error due to  $\hat{f}$  imperfectly estimating  $f$ .  
*Reducible* in the sense that we could improve  $\hat{f}$ .

# Statistical learning

## Prediction errors

As tends to be the case in life, you will make errors in predicting  $\mathbf{y}$ .

The accuracy of  $\hat{\mathbf{y}}$  depends upon **two errors**:

1. **Reducible error** The error due to  $\hat{f}$  imperfectly estimating  $f$ .  
*Reducible* in the sense that we could improve  $\hat{f}$ .
2. **Irreducible error** The error component that is outside of the model  $f$ .  
*Irreducible* because we defined an error term  $\varepsilon$  unexplained by  $f$ .

# Statistical learning

## Prediction errors

As tends to be the case in life, you will make errors in predicting  $\hat{\mathbf{y}}$ .

The accuracy of  $\hat{\mathbf{y}}$  depends upon **two errors**:

1. **Reducible error** The error due to  $\hat{f}$  imperfectly estimating  $f$ .  
*Reducible* in the sense that we could improve  $\hat{f}$ .
2. **Irreducible error** The error component that is outside of the model  $f$ .  
*Irreducible* because we defined an error term  $\varepsilon$  unexplained by  $f$ .

**Note** As its name implies, you can't get rid of *irreducible* error—but we can try to get rid of *reducible* errors.

# Statistical learning

## Prediction errors

Why we're stuck with *irreducible* error

$$\begin{aligned} E[\{\mathbf{y} - \hat{\mathbf{y}}\}^2] &= E\left[\left\{\mathbf{f}(\mathbf{X}) + \varepsilon + \hat{\mathbf{f}}(\mathbf{X})\right\}^2\right] \\ &= \underbrace{\left[\mathbf{f}(\mathbf{X}) - \hat{\mathbf{f}}(\mathbf{X})\right]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\varepsilon)}_{\text{Irreducible}} \end{aligned}$$

In less math:

- If  $\varepsilon$  exists, then  $\mathbf{X}$  cannot perfectly explain  $\mathbf{y}$ .
- So even if  $\hat{\mathbf{f}} = \mathbf{f}$ , we still have irreducible error.

# Statistical learning

## Prediction errors

Why we're stuck with *irreducible* error

$$\begin{aligned} E[\{\mathbf{y} - \hat{\mathbf{y}}\}^2] &= E\left[\left\{\mathbf{f}(\mathbf{X}) + \varepsilon + \hat{\mathbf{f}}(\mathbf{X})\right\}^2\right] \\ &= \underbrace{\left[\mathbf{f}(\mathbf{X}) - \hat{\mathbf{f}}(\mathbf{X})\right]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\varepsilon)}_{\text{Irreducible}} \end{aligned}$$

In less math:

- If  $\varepsilon$  exists, then  $\mathbf{X}$  cannot perfectly explain  $\mathbf{y}$ .
- So even if  $\hat{\mathbf{f}} = \mathbf{f}$ , we still have irreducible error.

Thus, to form our **best predictors**, we will **minimize reducible error**.

# Model accuracy

## MSE

Mean squared error (MSE) is the most common<sup>†</sup> way to measure model performance in a regression setting.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \left[ \mathbf{y}_i - \hat{\mathbf{f}}(\mathbf{x}_i) \right]^2$$

Recall:  $\mathbf{y}_i - \hat{\mathbf{f}}(\mathbf{x}_i) = \mathbf{y}_i - \hat{\mathbf{y}}_i$  is our prediction error.

<sup>†</sup> Most common does not mean best—it just means lots of people use it.

# Model accuracy

## MSE

Mean squared error (MSE) is the most common<sup>†</sup> way to measure model performance in a regression setting.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \left[ \mathbf{y}_i - \hat{\mathbf{f}}(\mathbf{x}_i) \right]^2$$

Recall:  $\mathbf{y}_i - \hat{\mathbf{f}}(\mathbf{x}_i) = \mathbf{y}_i - \hat{\mathbf{y}}_i$  is our prediction error.

Two notes about MSE

1. MSE will be (relatively) very small when **prediction error** is nearly zero.
2. MSE **penalizes** big errors more than little errors (the squared part).

<sup>†</sup> Most common does not mean best—it just means lots of people use it.

# Model accuracy

## Training or testing?

Low MSE (accurate performance) on the data that trained the model isn't actually impressive—maybe the model is just overfitting our data.<sup>†</sup>

*What we want:* How well does the model perform **on data it has never seen?**

<sup>†</sup> Recall the kNN performance for k=1.

# Model accuracy

## Training or testing?

Low MSE (accurate performance) on the data that trained the model isn't actually impressive—maybe the model is just overfitting our data.<sup>†</sup>

*What we want:* How well does the model perform **on data it has never seen**?

This introduces an important distinction:

1. **Training data:** The observations  $(y_i, x_i)$  used to **train** our model  $\hat{f}$ .
2. **Testing data:** The observations  $(y_0, x_0)$  that our model has yet to see—and which we can use to evaluate the performance of  $\hat{f}$ .

<sup>†</sup> Recall the kNN performance for k=1.

# Model accuracy

## Training or testing?

Low MSE (accurate performance) on the data that trained the model isn't actually impressive—maybe the model is just overfitting our data.<sup>†</sup>

*What we want:* How well does the model perform **on data it has never seen**?

This introduces an important distinction:

1. **Training data:** The observations  $(y_i, x_i)$  used to **train** our model  $\hat{f}$ .
2. **Testing data:** The observations  $(y_0, x_0)$  that our model has yet to see—and which we can use to evaluate the performance of  $\hat{f}$ .

**Real goal: Low test-sample MSE** (not the training MSE from before).

<sup>†</sup> Recall the kNN performance for k=1.

# Model accuracy

## Regression and loss

For **regression settings**, the loss is our **prediction**'s distance from **truth**, i.e.,

$$\text{error}_i = \mathbf{y}_i - \hat{\mathbf{y}}_i \quad \text{loss}_i = |\mathbf{y}_i - \hat{\mathbf{y}}_i| = |\text{error}_i|$$

Depending upon our ultimate goal, we choose **loss/objective functions**.

$$\text{L1 loss} = \sum_i |\mathbf{y}_i - \hat{\mathbf{y}}_i|$$

$$\text{MAE} = \frac{1}{n} \sum_i |\mathbf{y}_i - \hat{\mathbf{y}}_i|$$

$$\text{L2 loss} = \sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$$

$$\text{MSE} = \frac{1}{n} \sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$$

Whatever we're using, we care about **test performance** (e.g., test MSE), rather than training performance.

# Model accuracy

## Classification

For **classification problems**, we often use the **test error rate**.

$$\frac{1}{n} \sum_{i=1}^n \mathbb{I}(y_i \neq \hat{y}_i)$$

The **Bayes classifier**

1. predicts class  $j$  when  $\Pr(y_0 = j | \mathbf{X} = \mathbf{x}_0)$  exceeds all other classes.
2. produces the **Bayes decision boundary**—the decision boundary with the lowest test error rate.
3. is unknown: we must predict  $\Pr(y_0 = j | \mathbf{X} = \mathbf{x}_0)$ .

# Flexibility

## The bias-variance tradeoff

Finding the optimal level of flexibility highlights the **bias-variance tradeoff**.

**Bias** The error that comes from inaccurately estimating  $\hat{f}$ .

- More flexible models are better equipped to recover complex relationships ( $f$ ), reducing bias. (Real life is seldom linear.)
- Simpler (less flexible) models typically increase bias.

**Variance** The amount  $\hat{f}$  would change with a different **training sample**

- If new **training sets** drastically change  $\hat{f}$ , then we have a lot of uncertainty about  $f$  (and, in general,  $\hat{f} \not\approx f$ ).
- More flexible models generally add variance to  $f$ .

# Flexibility

## The bias-variance tradeoff

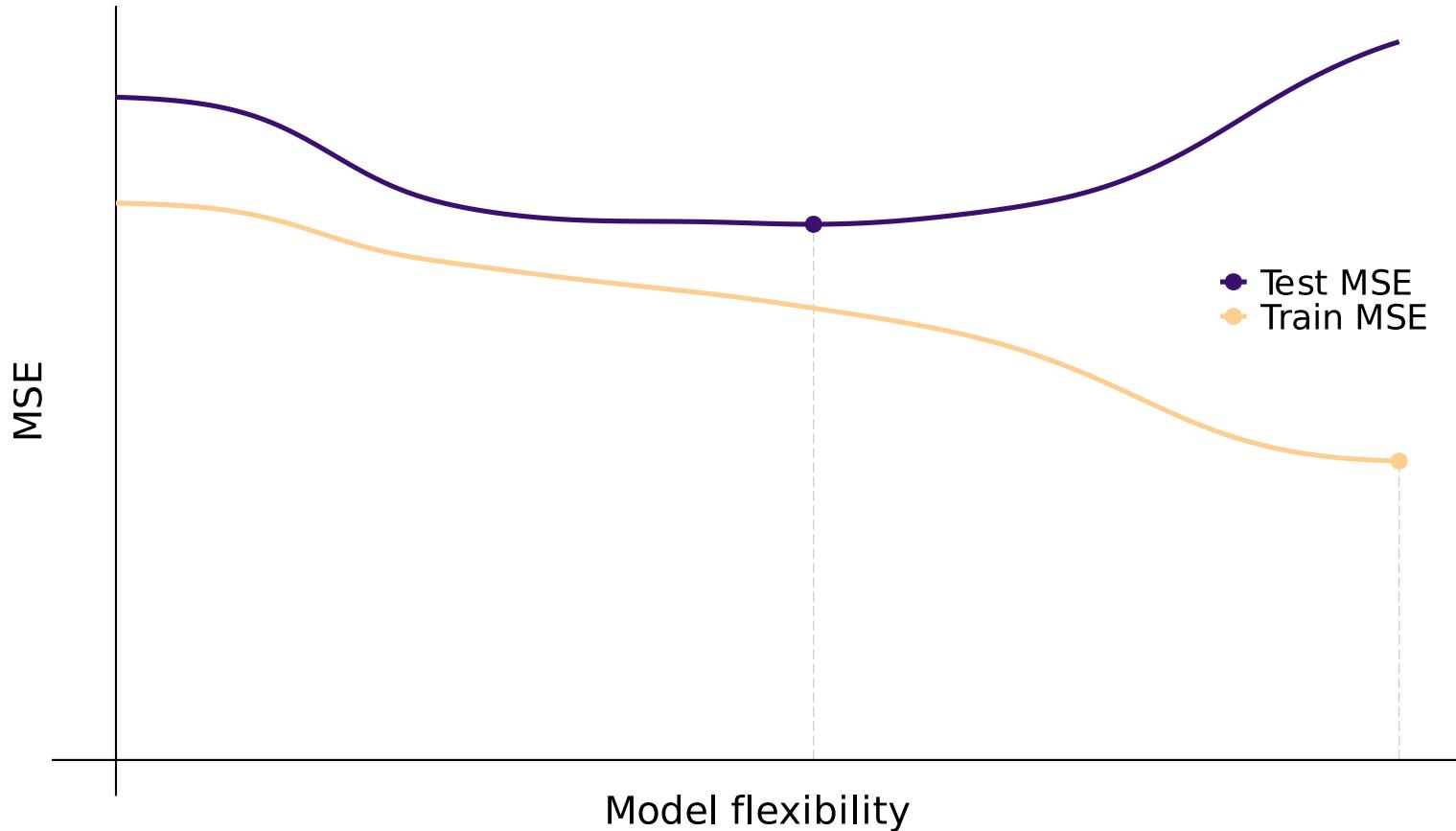
The expected value<sup>†</sup> of the **test MSE** can be written

$$E\left[\left(\mathbf{y}_0 - \hat{\mathbf{f}}(\mathbf{X}_0)\right)^2\right] = \underbrace{\text{Var}\left(\hat{\mathbf{f}}(\mathbf{X}_0)\right)}_{\text{Variance}} + \underbrace{\left[\text{Bias}\left(\hat{\mathbf{f}}(\mathbf{X}_0)\right)\right]^2}_{\text{Bias}} + \underbrace{\text{Var}(\varepsilon)}_{\text{Irr. error}}$$

### The tradeoff in terms of model flexibility

- Increasing flexibility *from total inflexibility* generally **reduces bias more** than it increases variance (reducing test MSE).
- At some point, the marginal benefits of flexibility **equal** marginal costs.
- Past this point (optimal flexibility), we **increase variance more** than we reduce bias (increasing test MSE).

**U-shaped test MSE** with respect to model flexibility (KNN here).  
Increases in variance eventually overcome reductions in (squared) bias.



# Resampling refresher

**Resampling methods** help understand uncertainty in statistical modeling.

The process behind the magic of resampling methods:

1. **Repeatedly draw samples** from the **training data**.
2. **Fit your model**(s) on each random sample.
3. **Compare** model performance (or estimates) **across samples**.
4. Infer the **variability/uncertainty in your model** from (3).

Sounds familiar, right?

# Resampling

## Hold out

*Recall:* We want to find the model that **minimizes out-of-sample test error.**

If we have a large test dataset, we can use it (once).

**Q<sub>1</sub>** What if we don't have a test set?

**Q<sub>2</sub>** What if we need to select and train a model?

**Q<sub>3</sub>** How can we avoid overfitting our training<sup>†</sup> data during model selection?

<sup>†</sup> Also relevant for *testing* data.

# Resampling

## Hold out

*Recall:* We want to find the model that **minimizes out-of-sample test error.**

If we have a large test dataset, we can use it (once).

**Q<sub>1</sub>** What if we don't have a test set?

**Q<sub>2</sub>** What if we need to select and train a model?

**Q<sub>3</sub>** How can we avoid overfitting our training<sup>†</sup> data during model selection?

**A<sub>1,2,3</sub>** **Hold-out methods** (e.g., cross validation) use training data to estimate test performance—**holding out** a mini "test" sample of the training data that we use to estimate the test error.

<sup>†</sup> Also relevant for *testing* data.

# Hold-out methods

## Option 1: The *validation set* approach

To estimate the **test error**, we can *hold out* a subset of our **training data** and then **validate** (evaluate) our model on this held out **validation set**.

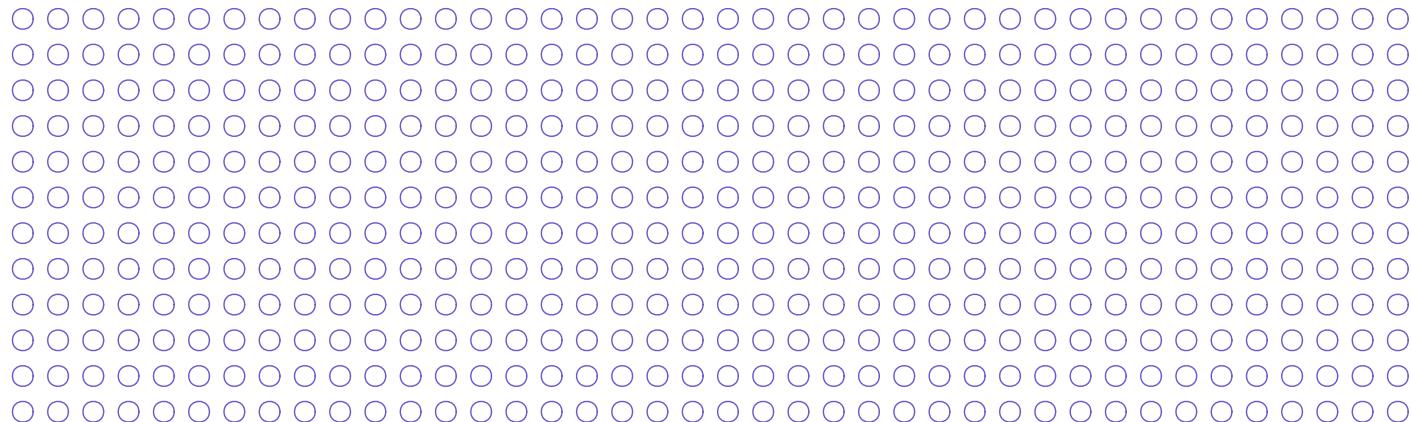
- The **validation error rate** estimates the **test error rate**
- The model only "sees" the non-validation subset of the **training data**.

# Hold-out methods

## Option 1: The *validation set* approach

To estimate the **test error**, we can *hold out* a subset of our **training data** and then **validate** (evaluate) our model on this held out **validation set**.

- The **validation error rate** estimates the **test error rate**
- The model only "sees" the non-validation subset of the **training data**.



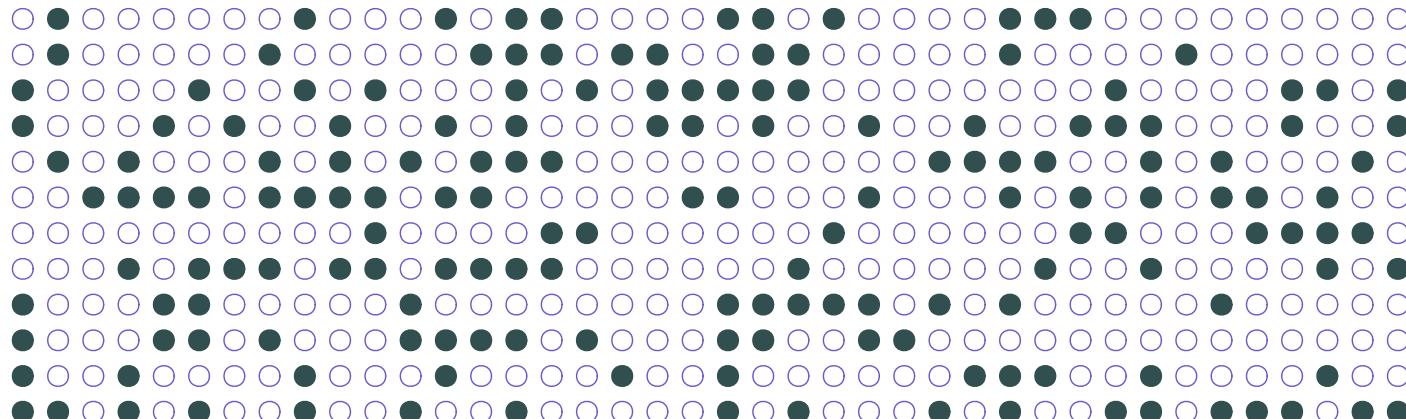
**Initial training set**

# Hold-out methods

## Option 1: The *validation set* approach

To estimate the **test error**, we can *hold out* a subset of our **training data** and then **validate** (evaluate) our model on this held out **validation set**.

- The **validation error rate** estimates the **test error rate**
- The model only "sees" the non-validation subset of the **training data**.



Validation (sub)set

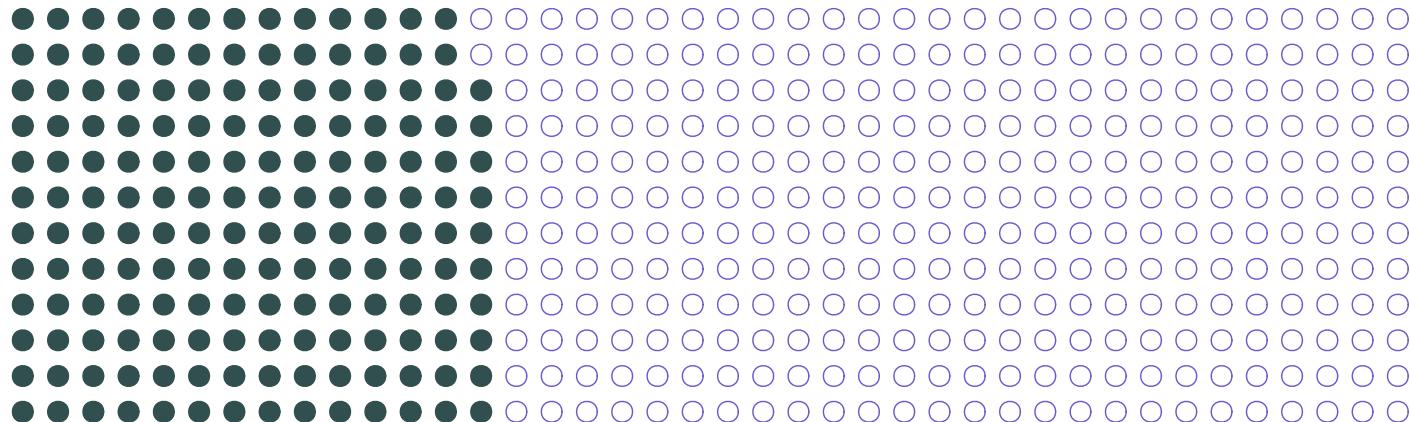
Training set: Model training

# Hold-out methods

## Option 1: The *validation set* approach

To estimate the **test error**, we can *hold out* a subset of our **training data** and then **validate** (evaluate) our model on this held out **validation set**.

- The **validation error rate** estimates the **test error rate**
- The model only "sees" the non-validation subset of the **training data**.



Validation (sub)set

Training set: Model training

# Hold-out methods

## Option 1: The *validation set* approach

*Example* We could use the validation-set approach to help select the degree of a polynomial for a linear-regression model.

# Hold-out methods

## Option 1: The *validation set* approach

*Example* We could use the validation-set approach to help select the degree of a polynomial for a linear-regression model.

The goal of the validation set is to **estimate out-of-sample (test) error.**

Q So what?

# Hold-out methods

## Option 1: The *validation set approach*

*Example* We could use the validation-set approach to help select the degree of a polynomial for a linear-regression model.

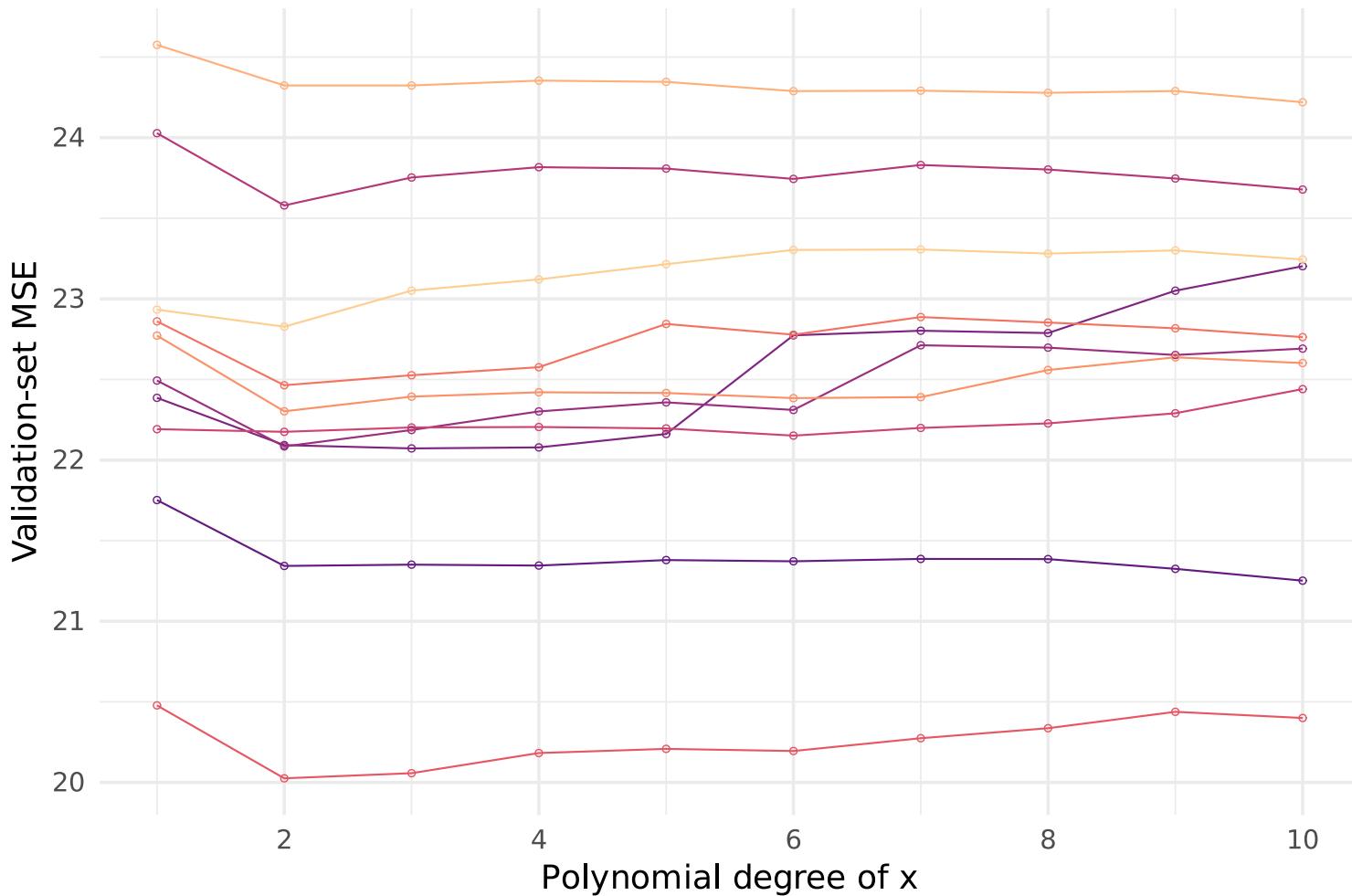
The goal of the validation set is to **estimate out-of-sample (test) error.**

**Q** So what?

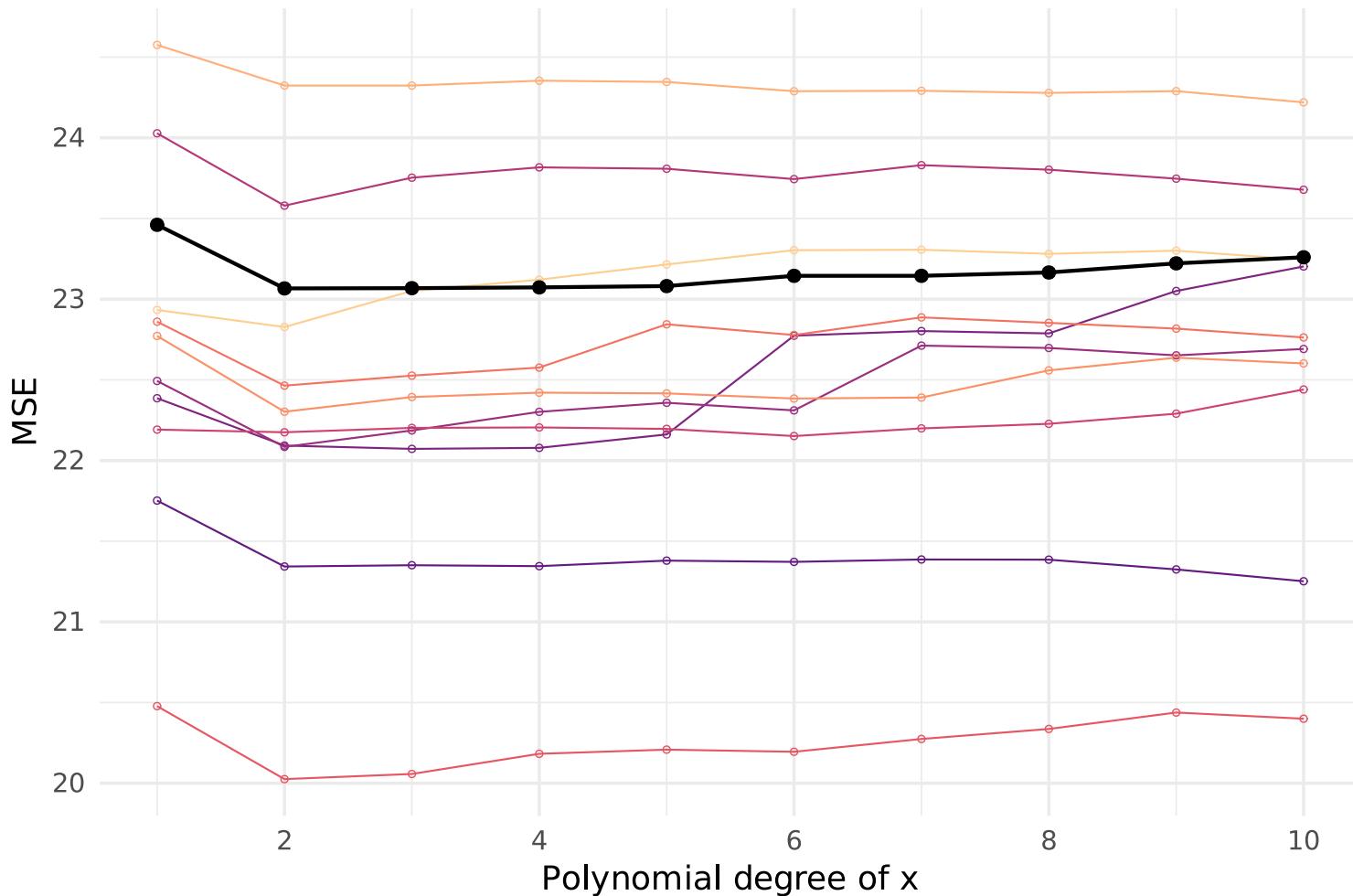
- Estimates come with **uncertainty**—varying from sample to sample.
- Variability (standard errors) is larger with **smaller samples**.

**Problem** This estimated error is often based upon a fairly small sample (<30% of our training data). So its variance can be large.

## Validation MSE for 10 different validation samples



## True test MSE compared to validation-set estimates



# Hold-out methods

## Option 1: The *validation set* approach

Put differently: The validation-set approach has ( $\geq$ ) two major drawbacks:

1. **High variability** Which observations are included in the validation set can greatly affect the validation MSE.
2. **Inefficiency in training our model** We're essentially throwing away the validation data when training the model—"wasting" observations.

# Hold-out methods

## Option 1: The *validation set* approach

Put differently: The validation-set approach has ( $\geq$ ) two major drawbacks:

1. **High variability** Which observations are included in the validation set can greatly affect the validation MSE.
2. **Inefficiency in training our model** We're essentially throwing away the validation data when training the model—"wasting" observations.  
(2)  $\Rightarrow$  validation MSE may overestimate test MSE.

Even if the validation-set approach provides an unbiased estimator for test error, it is likely a pretty noisy estimator.

# Hold-out methods

## Option 2: Leave-one-out cross validation

**Cross validation** solves the validation-set method's main problems.

- Use more (= all) of the data for training (lower variability; less bias).
- Still maintains separation between training and validation subsets.

# Hold-out methods

## Option 2: Leave-one-out cross validation

**Cross validation** solves the validation-set method's main problems.

- Use more (= all) of the data for training (lower variability; less bias).
- Still maintains separation between training and validation subsets.

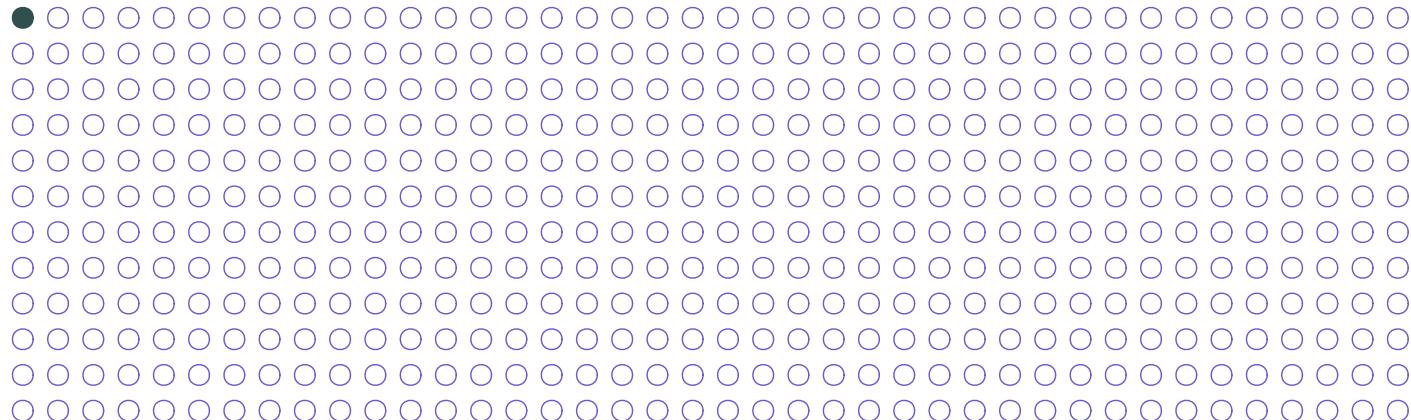
**Leave-one-out cross validation** (LOOCV) is perhaps the cross-validation method most similar to the validation-set approach.

- Your validation set is exactly one observation.
- *New* You repeat the validation exercise for every observation.
- *New* Estimate MSE as the mean across all observations.

# Hold-out methods

## Option 2: Leave-one-out cross validation

Each observation takes a turn as the **validation set**,  
while the other  $n-1$  observations get to **train the model**.

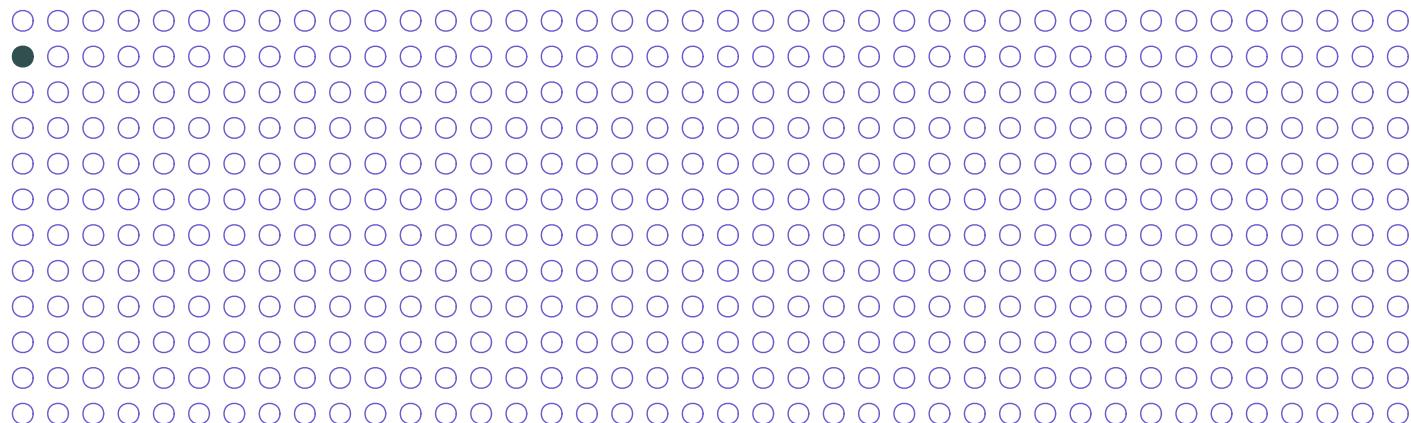


Observation 1's turn for validation produces  $MSE_1$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Each observation takes a turn as the **validation set**,  
while the other n-1 observations get to **train the model**.

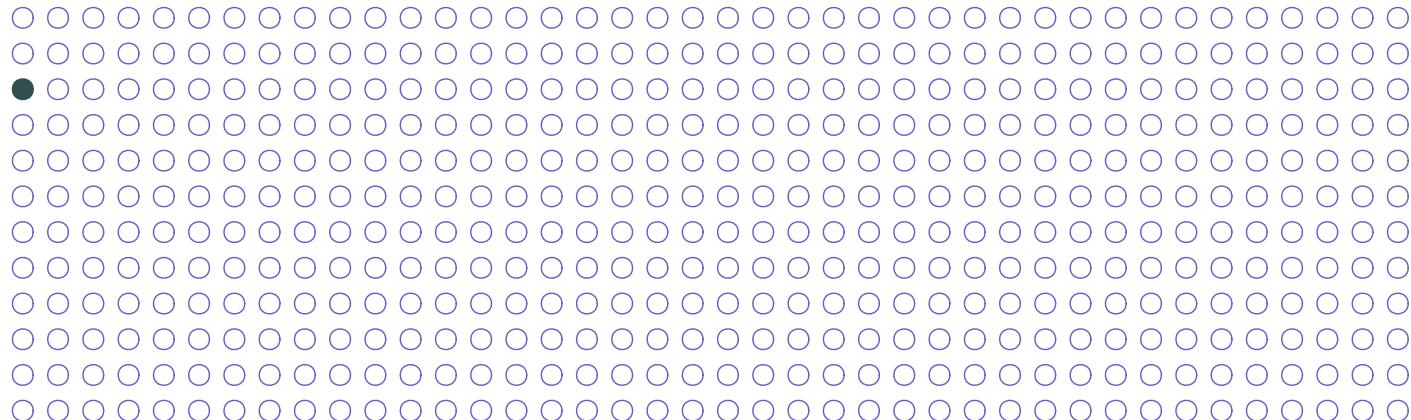


Observation 2's turn for validation produces  $MSE_2$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Each observation takes a turn as the **validation set**,  
while the other n-1 observations get to **train the model**.

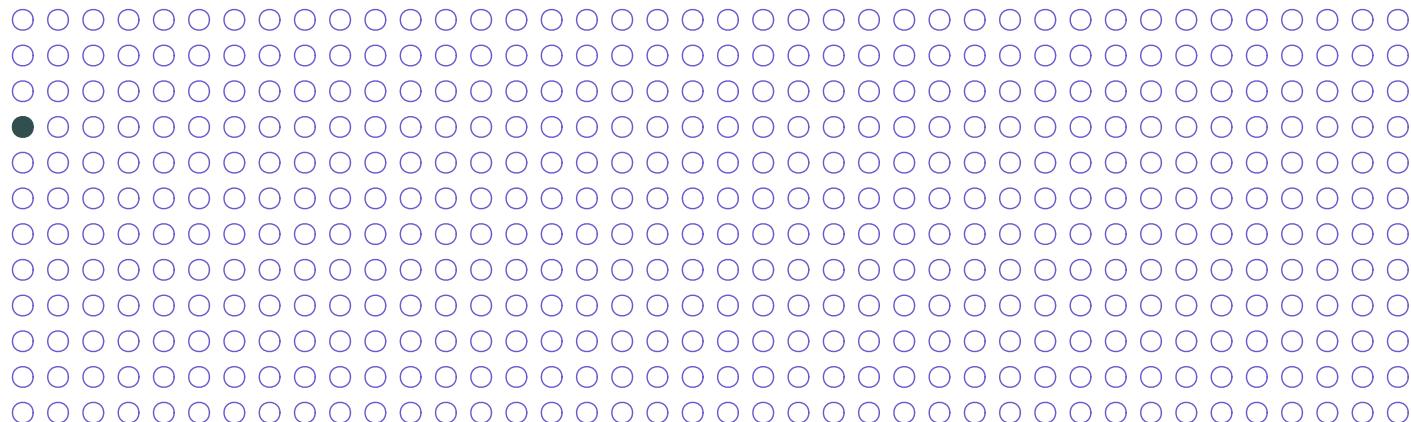


Observation 3's turn for validation produces  $MSE_3$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Each observation takes a turn as the **validation set**,  
while the other n-1 observations get to **train the model**.

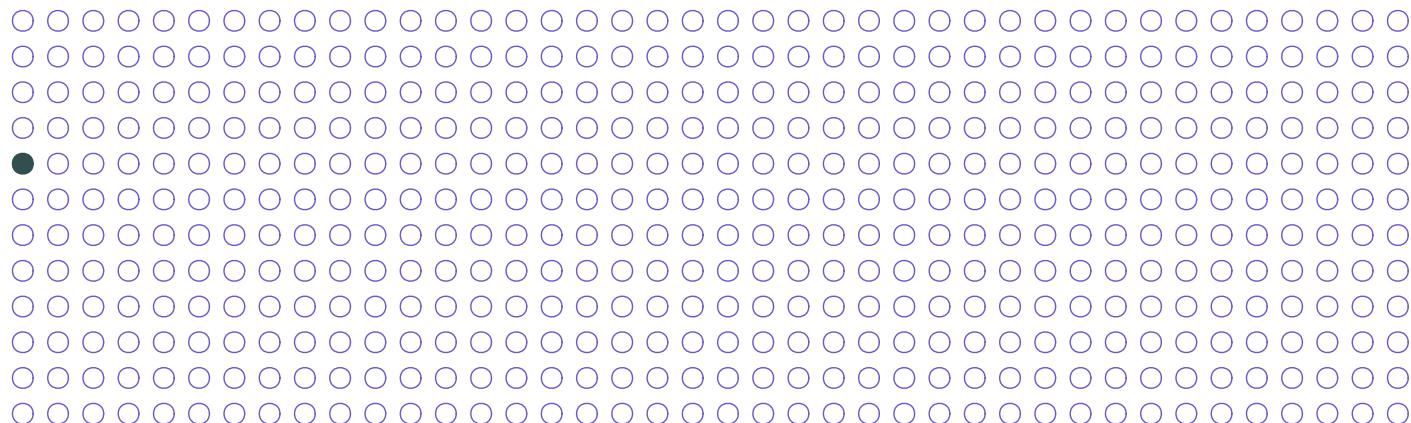


Observation 4's turn for validation produces  $MSE_4$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Each observation takes a turn as the **validation set**,  
while the other  $n-1$  observations get to **train the model**.

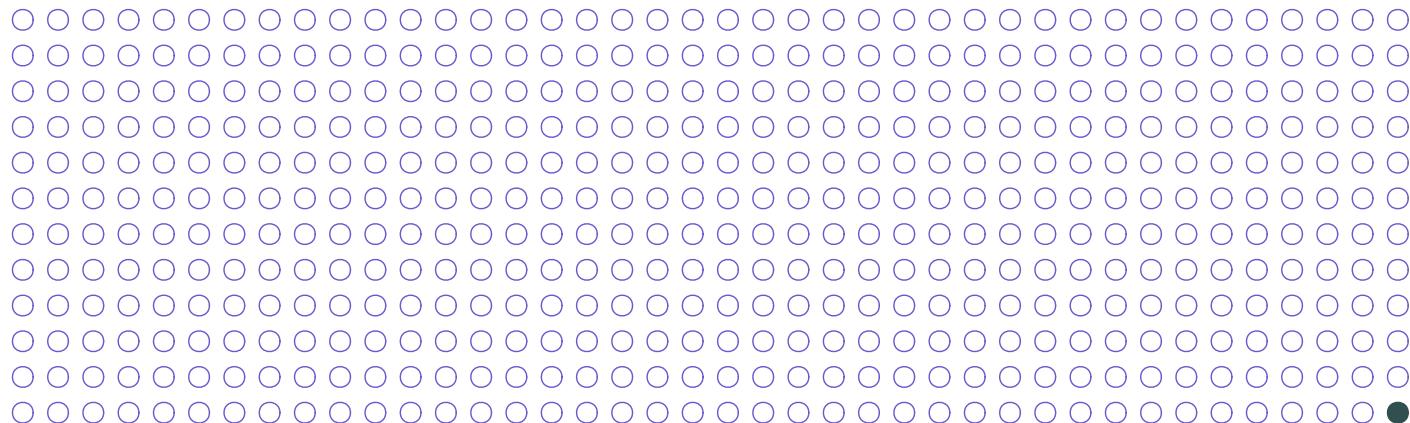


Observation 5's turn for validation produces  $\text{MSE}_5$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Each observation takes a turn as the **validation set**,  
while the other  $n-1$  observations get to **train the model**.



Observation n's turn for validation produces  $MSE_n$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Because **LOOCV uses  $n-1$  observations** to train the model,<sup>†</sup>  $MSE_i$  (validation MSE from observation  $i$ ) is approximately unbiased for test MSE.

**Problem**  $MSE_i$  is a terribly noisy estimator for test MSE (albeit  $\approx$ unbiased).

<sup>†</sup> And because often  $n-1 \approx n$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Because **LOOCV uses n-1 observations** to train the model,<sup>†</sup>  $\text{MSE}_i$  (validation MSE from observation i) is approximately unbiased for test MSE.

**Problem**  $\text{MSE}_i$  is a terribly noisy estimator for test MSE (albeit  $\approx$ unbiased).

**Solution** Take the mean!

$$\text{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^n \text{MSE}_i$$

<sup>†</sup> And because often  $n-1 \approx n$ .

# Hold-out methods

## Option 2: Leave-one-out cross validation

Because **LOOCV uses n-1 observations** to train the model,<sup>†</sup>  $\text{MSE}_i$  (validation MSE from observation i) is approximately unbiased for test MSE.

**Problem**  $\text{MSE}_i$  is a terribly noisy estimator for test MSE (albeit  $\approx$ unbiased).

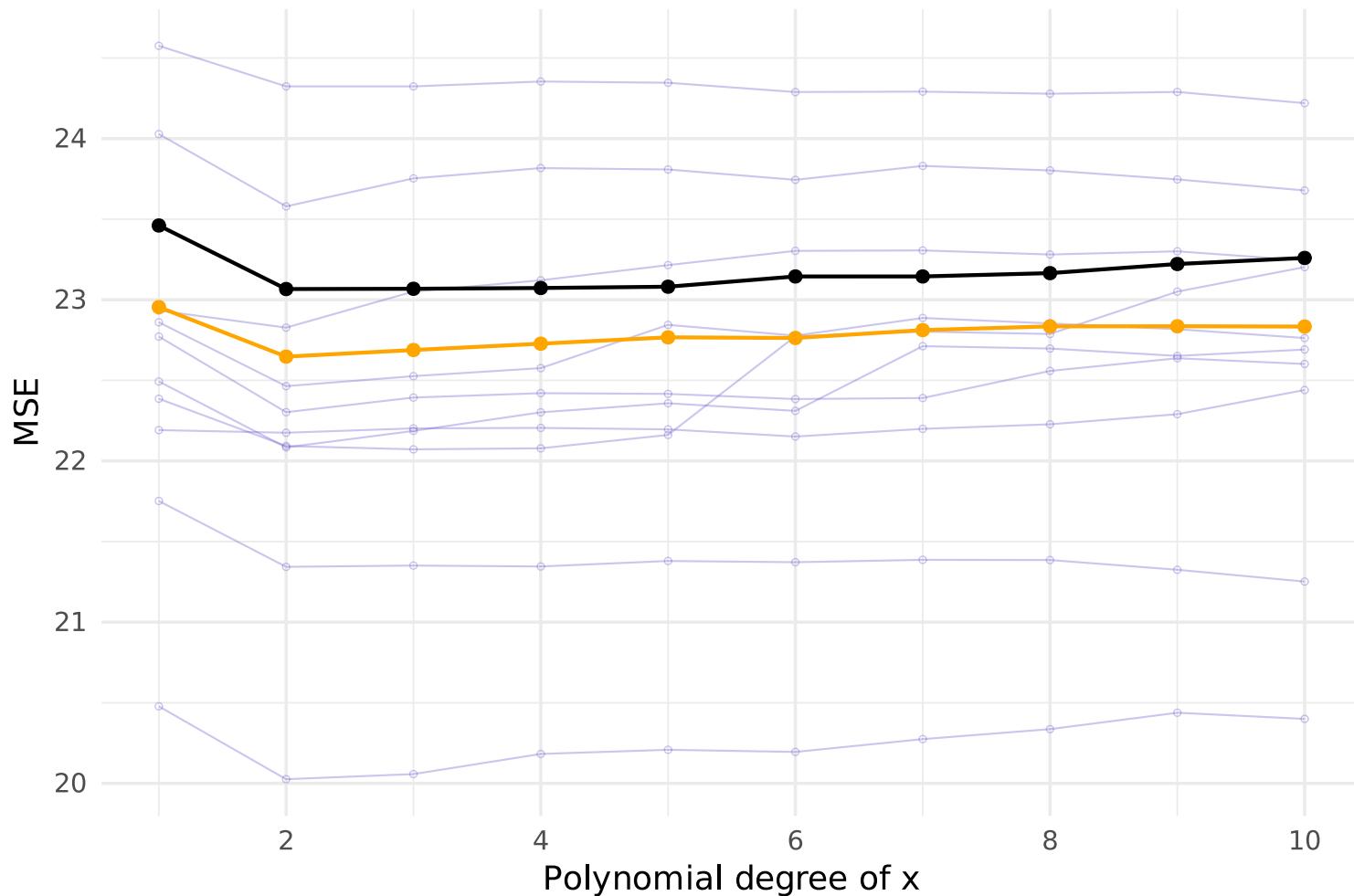
**Solution** Take the mean!

$$\text{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^n \text{MSE}_i$$

1. LOOCV **reduces bias** by using n-1 (almost all) observations for training.
2. LOOCV **resolves variance**: it makes all possible comparison (no dependence upon which validation-test split you make).

<sup>†</sup> And because often  $n-1 \approx n$ .

## True test MSE and LOOCV MSE compared to validation-set estimates



# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:  
**k-fold cross validation.**

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once  
(training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:  
**k-fold cross validation.**

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once (training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

Benefits?

# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:  
**k-fold cross validation.**

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once (training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

Benefits?

1. **Less computationally demanding** (fit model  $k = 5$  or 10 times; not  $n$ ).

# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:  
**k-fold cross validation.**

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once (training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

Benefits?

1. **Less computationally demanding** (fit model  $k = 5$  or 10 times; not  $n$ ).
2. **Greater accuracy** (in general) due to bias-variance tradeoff!

# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:

### k-fold cross validation.

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once (training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

Benefits?

1. **Less computationally demanding** (fit model  $k = 5$  or 10 times; not  $n$ ).
2. **Greater accuracy** (in general) due to bias-variance tradeoff!
  - Somewhat higher bias, relative to LOOCV:  $n - 1$  vs.  $(k - 1)/k$ .

# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:  
**k-fold cross validation.**

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once (training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

Benefits?

1. **Less computationally demanding** (fit model  $k = 5$  or 10 times; not  $n$ ).
2. **Greater accuracy** (in general) due to bias-variance tradeoff!
  - Somewhat higher bias, relative to LOOCV:  $n - 1$  vs.  $(k - 1)/k$ .
  - Lower variance due to high-degree of correlation in LOOCV  $MSE_i$ .

# Hold-out methods

## Option 3: k-fold cross validation

Leave-one-out cross validation is a special case of a broader strategy:  
**k-fold cross validation.**

1. **Divide** the training data into  $k$  equally sized groups (folds).
2. **Iterate** over the  $k$  folds, treating each as a validation set once (training the model on the other  $k - 1$  folds).
3. **Average** the folds' MSEs to estimate test MSE.

Benefits?

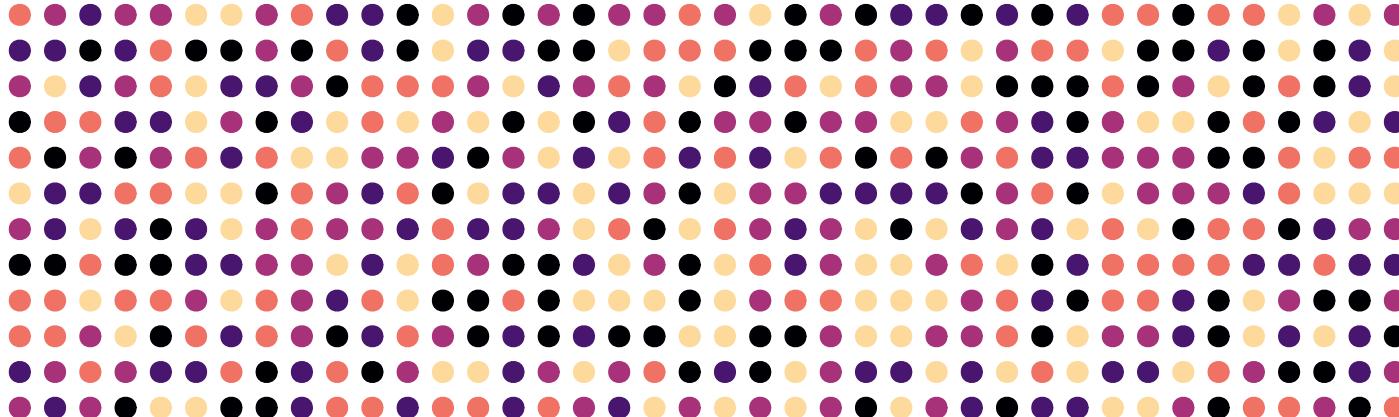
1. **Less computationally demanding** (fit model  $k = 5$  or 10 times; not  $n$ ).
2. **Greater accuracy** (in general) due to bias-variance tradeoff!
  - Somewhat higher bias, relative to LOOCV:  $n - 1$  vs.  $(k - 1)/k$ .
  - Lower variance due to high-degree of correlation in LOOCV  $\text{MSE}_i$ . 

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



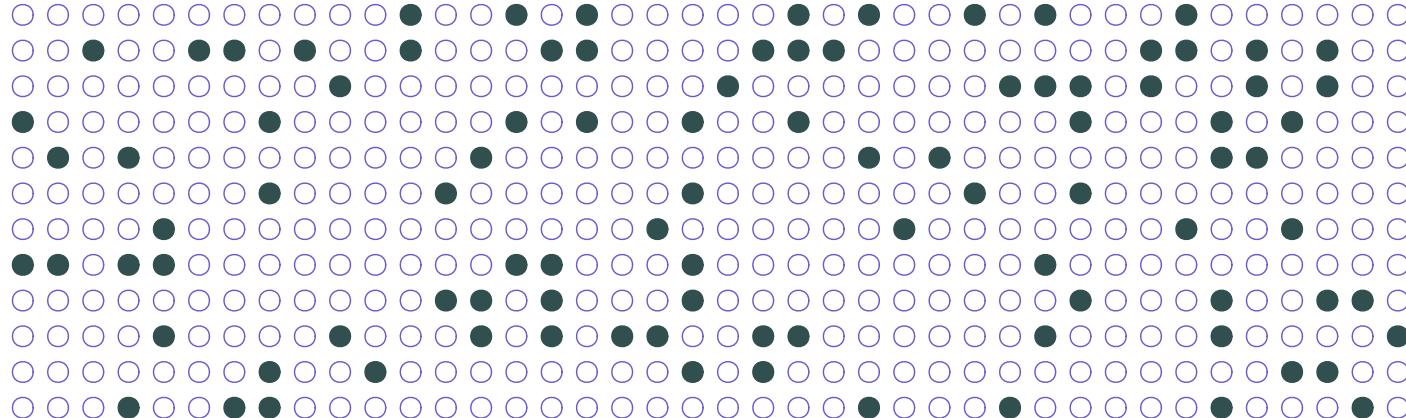
Our  $k = 5$  folds.

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



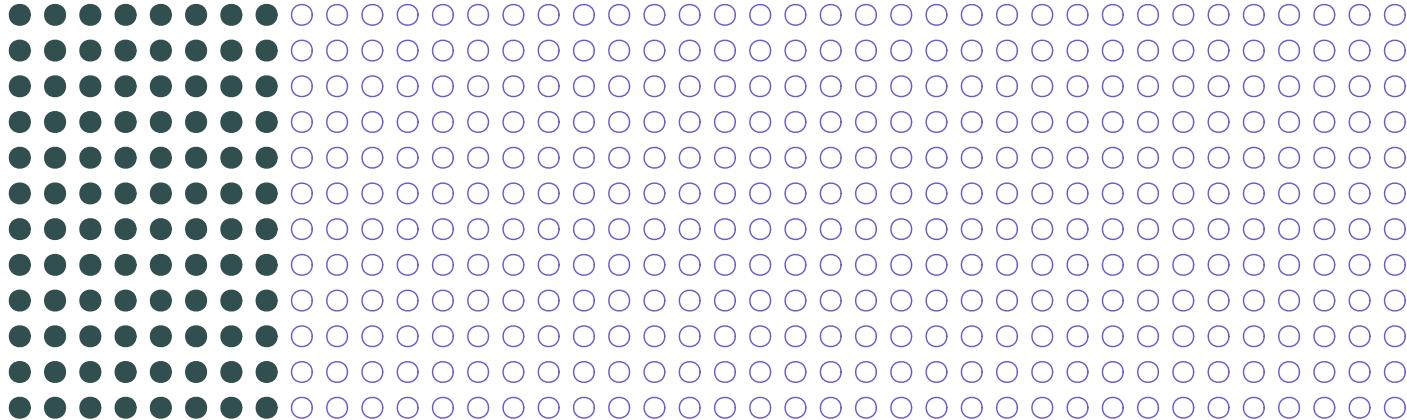
Each fold takes a turn at **validation**. The other  $k - 1$  folds **train**.

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



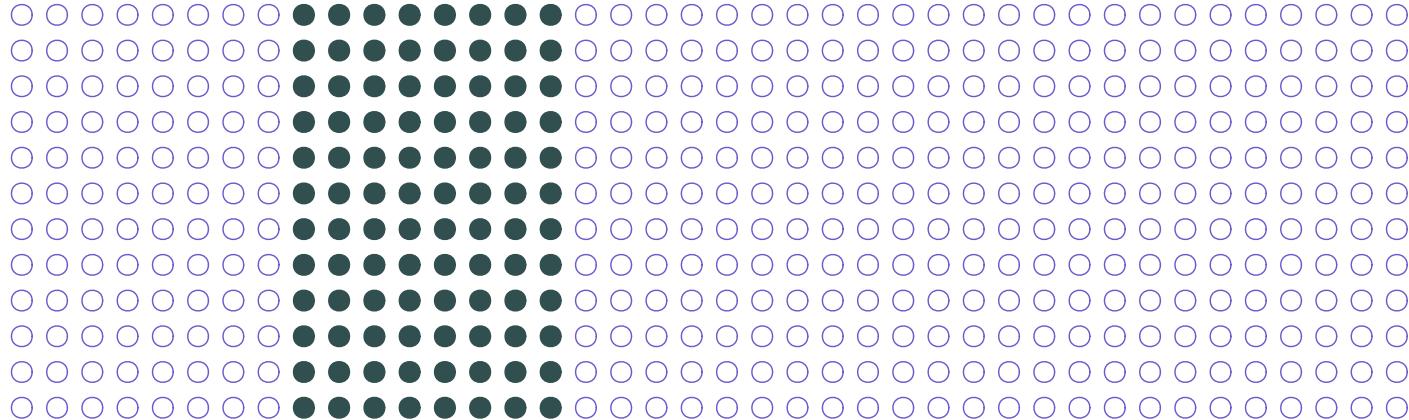
For  $k = 5$ , fold number 1 as the **validation set** produces  $\text{MSE}_{k=1}$ .

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



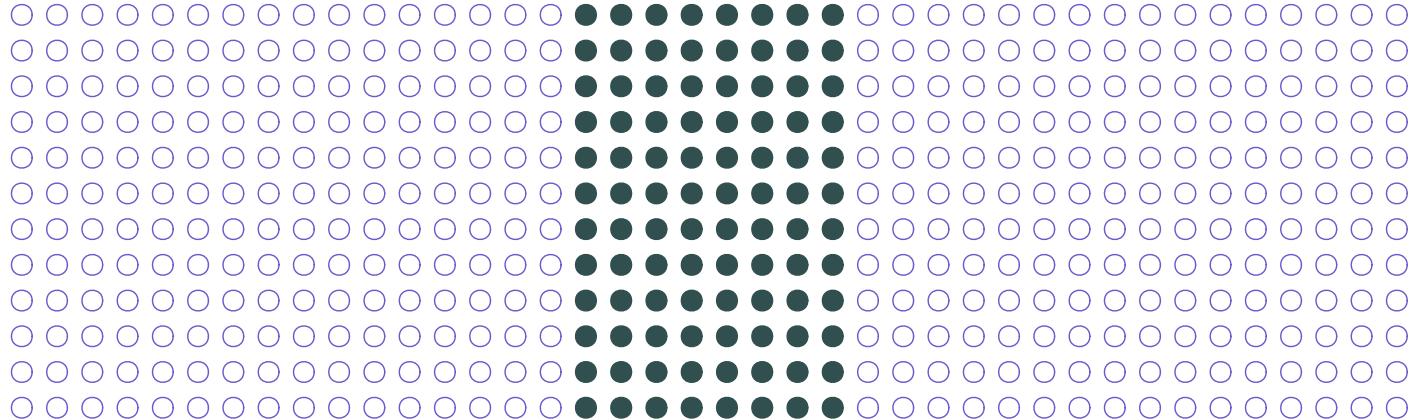
For  $k = 5$ , fold number 2 as the **validation set** produces  $\text{MSE}_{k=2}$ .

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



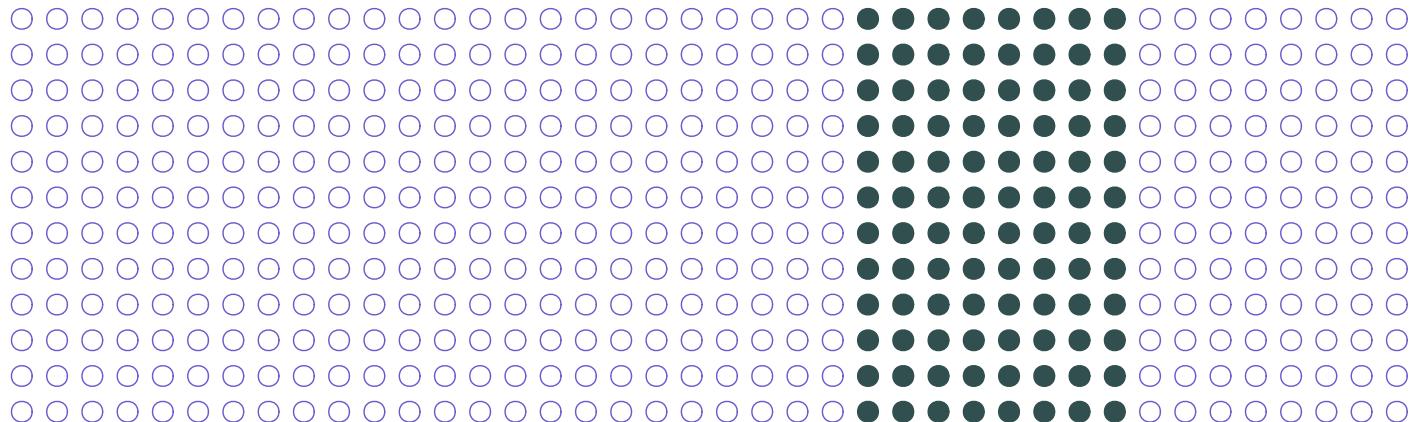
For  $k = 5$ , fold number 3 as the **validation set** produces  $\text{MSE}_{k=3}$ .

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



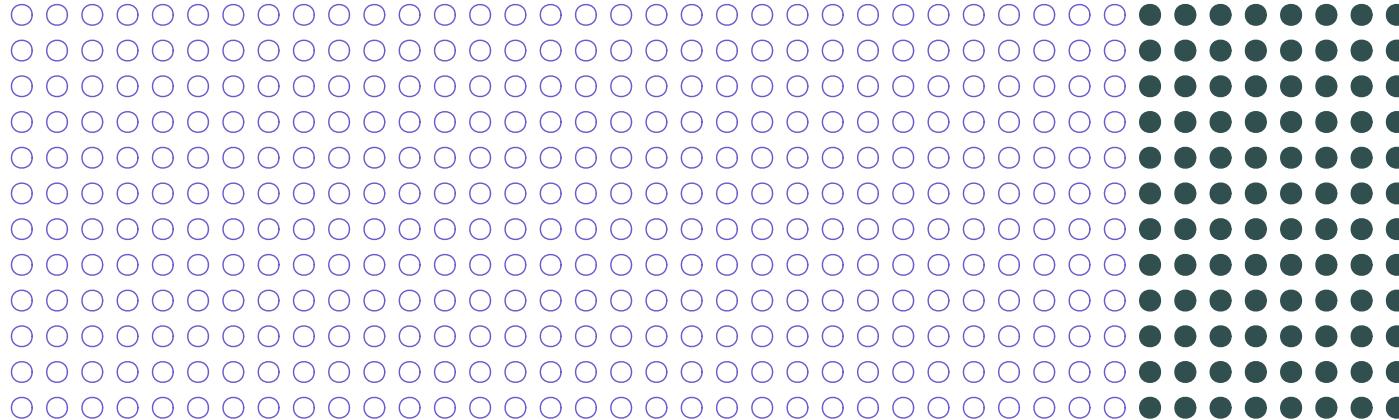
For  $k = 5$ , fold number 4 as the **validation set** produces  $\text{MSE}_{k=4}$ .

# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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



For  $k = 5$ , fold number 5 as the **validation set** produces  $\text{MSE}_{k=5}$ .

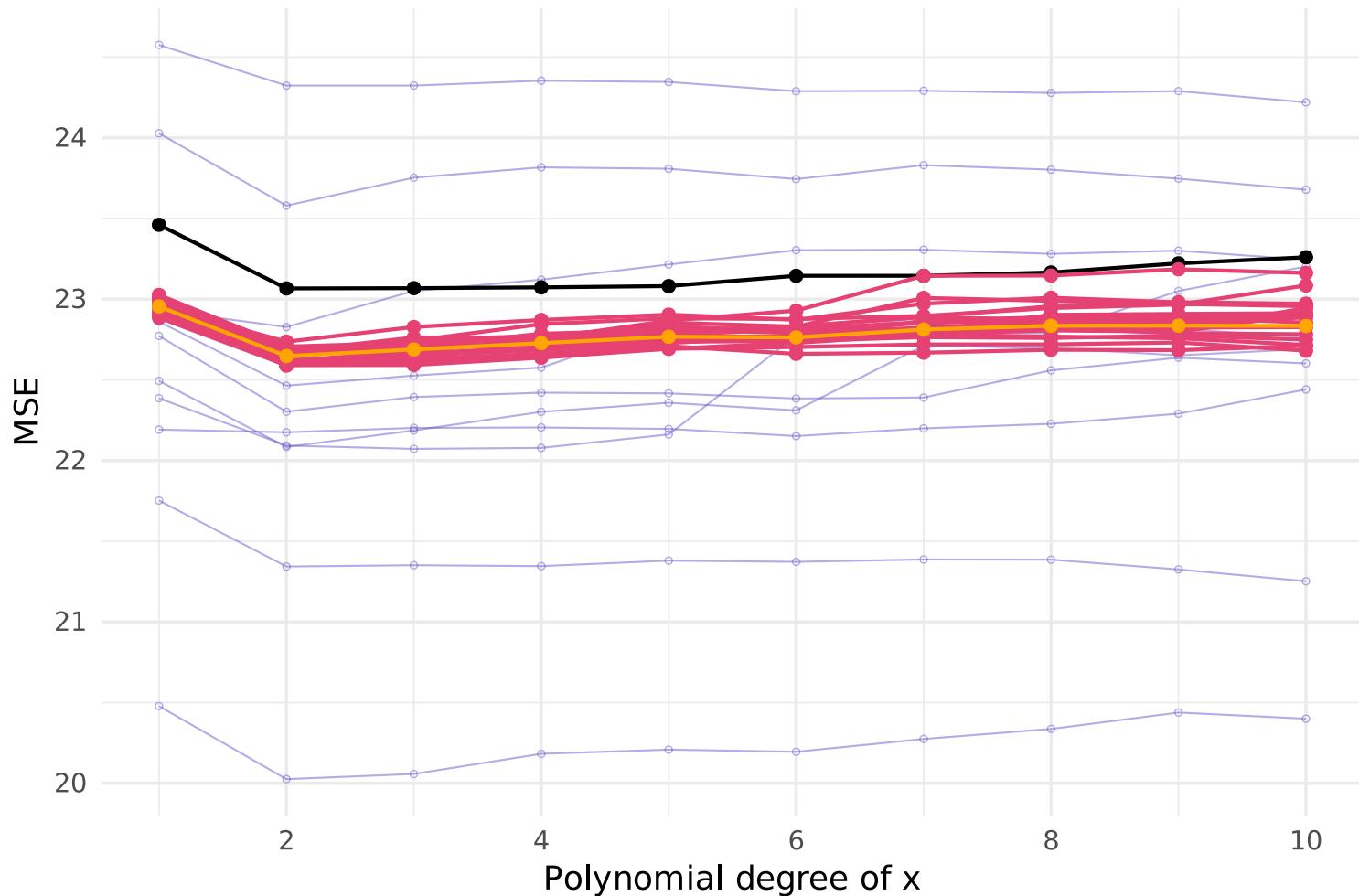
# Hold-out methods

## Option 3: k-fold cross validation

With  $k$ -fold cross validation, we estimate test MSE as

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

## Test MSE vs. estimates: LOOCV, 5-fold CV (20x), and validation set (10x)



Note: Each of these methods extends to classification settings, e.g., LOOCV

$$\text{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(\textcolor{orange}{y_i} \neq \hat{y}_i)$$

# Hold-out methods

## Caveat

So far, we've treated each observation as separate/independent from each other observation.

The methods that we've defined so far actually need this independence.

# Hold-out methods

## Goals and alternatives

You can use CV for either of two important **modeling tasks**:

- **Model selection** Choosing and tuning a model
- **Model assessment** Evaluating a model's accuracy

# Hold-out methods

## Goals and alternatives

You can use CV for either of two important **modeling tasks**:

- **Model selection** Choosing and tuning a model
- **Model assessment** Evaluating a model's accuracy

Alternative approach: **Shrinkage methods**

- fit a model that contains *all p* predictors
- simultaneously: shrink<sup>†</sup> coefficients toward zero

<sup>†</sup> Synonyms for shrink: constrain or regularize

# Hold-out methods

## Goals and alternatives

You can use CV for either of two important **modeling tasks**:

- **Model selection** Choosing and tuning a model
- **Model assessment** Evaluating a model's accuracy

Alternative approach: **Shrinkage methods**

- fit a model that contains *all p* predictors
- simultaneously: shrink<sup>†</sup> coefficients toward zero

*Idea:* Penalize the model for coefficients as they move away from zero.

<sup>†</sup> Synonyms for shrink: constrain or regularize

# Shrinkage

## Why?

Q How could shrinking coefficients toward zero help or predictions?

# Shrinkage

## Why?

- Q** How could shrinking coefficients toward zero help or predictions?
- A** Remember we're generally facing a tradeoff between bias and variance.

# Shrinkage

## Why?

**Q** How could shrinking coefficients toward zero help or predictions?

**A** Remember we're generally facing a tradeoff between bias and variance.

- Shrinking our coefficients toward zero **reduces the model's variance**.<sup>†</sup>
- **Penalizing** our model for **larger coefficients** shrinks them toward zero.
- The **optimal penalty** will balance reduced variance with increased bias.

<sup>†</sup> Imagine the extreme case: a model whose coefficients are all zeros has no variance.

# Shrinkage

## Why?

**Q** How could shrinking coefficients toward zero help or predictions?

**A** Remember we're generally facing a tradeoff between bias and variance.

- Shrinking our coefficients toward zero **reduces the model's variance**.<sup>†</sup>
- **Penalizing** our model for **larger coefficients** shrinks them toward zero.
- The **optimal penalty** will balance reduced variance with increased bias.

Now you understand shrinkage methods.

- **Ridge regression**
- **Lasso**
- **Elasticnet**

<sup>†</sup> Imagine the extreme case: a model whose coefficients are all zeros has no variance.

# Ridge regression

# Ridge regression

## Back to least squares (again)

Remember OLS? Least-squares regression finds  $\hat{\beta}_j$ 's by minimizing RSS

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^n e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^n \left( \textcolor{orange}{y_i} - \underbrace{\left[ \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

# Ridge regression

## Back to least squares (again)

Remember OLS? Least-squares regression finds  $\hat{\beta}_j$ 's by minimizing RSS

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^n e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^n \left( \textcolor{orange}{y_i} - \underbrace{\left[ \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

Ridge regression makes a small change

- adds a **shrinkage penalty** = the sum of squared coefficients  $(\lambda \sum_j \beta_j^2)$
- minimizes the (weighted) sum of RSS and the shrinkage penalty

# Ridge regression

## Back to least squares (again)

Remember OLS? Least-squares regression finds  $\hat{\beta}_j$ 's by minimizing RSS

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^n e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^n \left( \mathbf{y}_i - \underbrace{\left[ \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

Ridge regression makes a small change

- adds a **shrinkage penalty** = the sum of squared coefficients  $(\lambda \sum_j \beta_j^2)$
- minimizes the (weighted) sum of RSS and the shrinkage penalty

$$\min_{\hat{\beta}^R} \sum_{i=1}^n \left( \mathbf{y}_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

# Ridge regression

## Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^n \left( \textcolor{orange}{y}_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

## Least squares

$$\min_{\hat{\beta}} \sum_{i=1}^n \left( \textcolor{orange}{y}_i - \hat{y}_i \right)^2$$

$\lambda$  ( $\geq 0$ ) is a tuning parameter for the harshness of the penalty.

$\lambda = 0$  implies no penalty: we are back to least squares.

# Ridge regression

## Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^n \left( \textcolor{orange}{y}_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

## Least squares

$$\min_{\hat{\beta}} \sum_{i=1}^n \left( \textcolor{orange}{y}_i - \hat{y}_i \right)^2$$

$\lambda$  ( $\geq 0$ ) is a tuning parameter for the harshness of the penalty.

$\lambda = 0$  implies no penalty: we are back to least squares.

Each value of  $\lambda$  produces a new set of coefficients.

# Ridge regression

## Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^n \left( \mathbf{y}_i - \hat{\mathbf{y}}_i \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

## Least squares

$$\min_{\hat{\beta}} \sum_{i=1}^n \left( \mathbf{y}_i - \hat{\mathbf{y}}_i \right)^2$$

$\lambda$  ( $\geq 0$ ) is a tuning parameter for the harshness of the penalty.

$\lambda = 0$  implies no penalty: we are back to least squares.

Each value of  $\lambda$  produces a new set of coefficients.

Ridge's approach to the bias-variance tradeoff: **Balance**

- reducing **RSS**, i.e.,  $\sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
- reducing **coefficients' magnitudes** (ignoring the intercept)

$\lambda$  determines how much ridge "cares about" these two quantities.<sup>†</sup>

<sup>†</sup> With  $\lambda = 0$ , least-squares regression only "cares about" RSS.

# Ridge regression

## $\lambda$ and penalization

Choosing a *good* value for  $\lambda$  is key.

- If  $\lambda$  is too small, then our model is essentially back to OLS.
- If  $\lambda$  is too large, then we shrink all of our coefficients too close to zero.

# Ridge regression

## $\lambda$ and penalization

Choosing a *good* value for  $\lambda$  is key.

- If  $\lambda$  is too small, then our model is essentially back to OLS.
- If  $\lambda$  is too large, then we shrink all of our coefficients too close to zero.

Q So what do we do?

# Ridge regression

## $\lambda$ and penalization

Choosing a *good* value for  $\lambda$  is key.

- If  $\lambda$  is too small, then our model is essentially back to OLS.
- If  $\lambda$  is too large, then we shrink all of our coefficients too close to zero.

**Q** So what do we do?

**A** Cross validate!

(You saw that coming, right?)

# Ridge regression

## Penalization and standardization

**Important** Predictors' **units** can drastically **affect ridge regression results**.

**Why?**

# Ridge regression

## Penalization and standardization

**Important** Predictors' **units** can drastically **affect ridge regression results**.

**Why?** Because  $\mathbf{x}_j$ 's units affect  $\beta_j$ , and ridge is very sensitive to  $\beta_j$ .

# Ridge regression

## Penalization and standardization

**Important** Predictors' **units** can drastically **affect ridge regression results**.

**Why?** Because  $\mathbf{x}_j$ 's units affect  $\beta_j$ , and ridge is very sensitive to  $\beta_j$ .

*Example* Let  $x_1$  denote distance.

### Least-squares regression

If  $x_1$  is *meters* and  $\beta_1 = 3$ , then when  $x_1$  is *km*,  $\beta_1 = 3,000$ .

The scale/units of predictors do not affect least squares' estimates.

# Ridge regression

## Penalization and standardization

**Important** Predictors' **units** can drastically **affect ridge regression results**.

**Why?** Because  $\mathbf{x}_j$ 's units affect  $\beta_j$ , and ridge is very sensitive to  $\beta_j$ .

*Example* Let  $x_1$  denote distance.

### Least-squares regression

If  $x_1$  is *meters* and  $\beta_1 = 3$ , then when  $x_1$  is *km*,  $\beta_1 = 3,000$ .

The scale/units of predictors do not affect least squares' estimates.

**Ridge regression** pays a much larger penalty for  $\beta_1 = 3,000$  than  $\beta_1 = 3$ .

You will not get the same (scaled) estimates when you change units.

# Ridge regression

## Penalization and standardization

**Important** Predictors' **units** can drastically **affect ridge regression results**.

**Why?** Because  $\mathbf{x}_j$ 's units affect  $\beta_j$ , and ridge is very sensitive to  $\beta_j$ .

*Example* Let  $x_1$  denote distance.

### Least-squares regression

If  $x_1$  is *meters* and  $\beta_1 = 3$ , then when  $x_1$  is *km*,  $\beta_1 = 3,000$ .

The scale/units of predictors do not affect least squares' estimates.

**Ridge regression** pays a much larger penalty for  $\beta_1 = 3,000$  than  $\beta_1 = 3$ .

You will not get the same (scaled) estimates when you change units.

*Solution* Standardize your variables, *i.e.*,  $x_{\text{stnd}} = (x - \text{mean}(x))/\text{sd}(x)$ .

# Lasso

# Lasso

## Intro

**Lasso** simply replaces ridge's *squared* coefficients with absolute values.

# Lasso

## Intro

Lasso simply replaces ridge's *squared* coefficients with absolute values.

## Ridge regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^n (\textcolor{orange}{y_i} - \hat{y}_i)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

## Lasso

$$\min_{\hat{\beta}^L} \sum_{i=1}^n (\textcolor{orange}{y_i} - \hat{y}_i)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

Everything else will be the same—except one aspect...

# Lasso

## Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of  $\beta_j$ .

You always pay  $\lambda$  to increase  $|\beta_j|$  by one unit.

# Lasso

## Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of  $\beta_j$ .

You always pay  $\lambda$  to increase  $|\beta_j|$  by one unit.

The only way to avoid lasso's penalty is to **set coefficients to zero**.

# Lasso

## Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of  $\beta_j$ .

You always pay  $\lambda$  to increase  $|\beta_j|$  by one unit.

The only way to avoid lasso's penalty is to **set coefficients to zero**.

This feature has two **benefits**

1. Some coefficients will be **set to zero**—we get "sparse" models.
2. Lasso can be used for subset/feature **selection**.

# Lasso

## Shrinkage

Unlike ridge, lasso's penalty does not increase with the size of  $\beta_j$ .

You always pay  $\lambda$  to increase  $|\beta_j|$  by one unit.

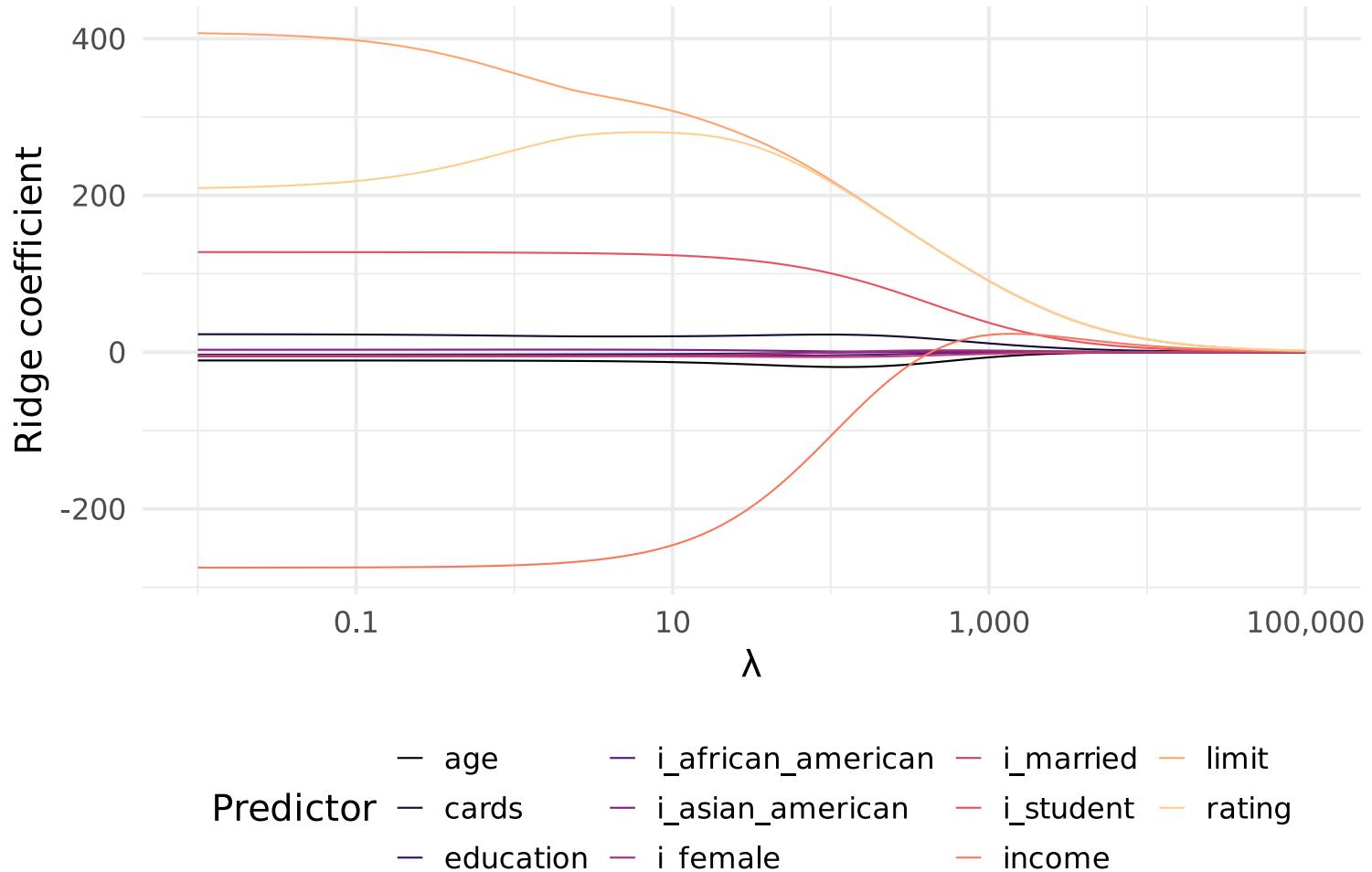
The only way to avoid lasso's penalty is to **set coefficients to zero**.

This feature has two **benefits**

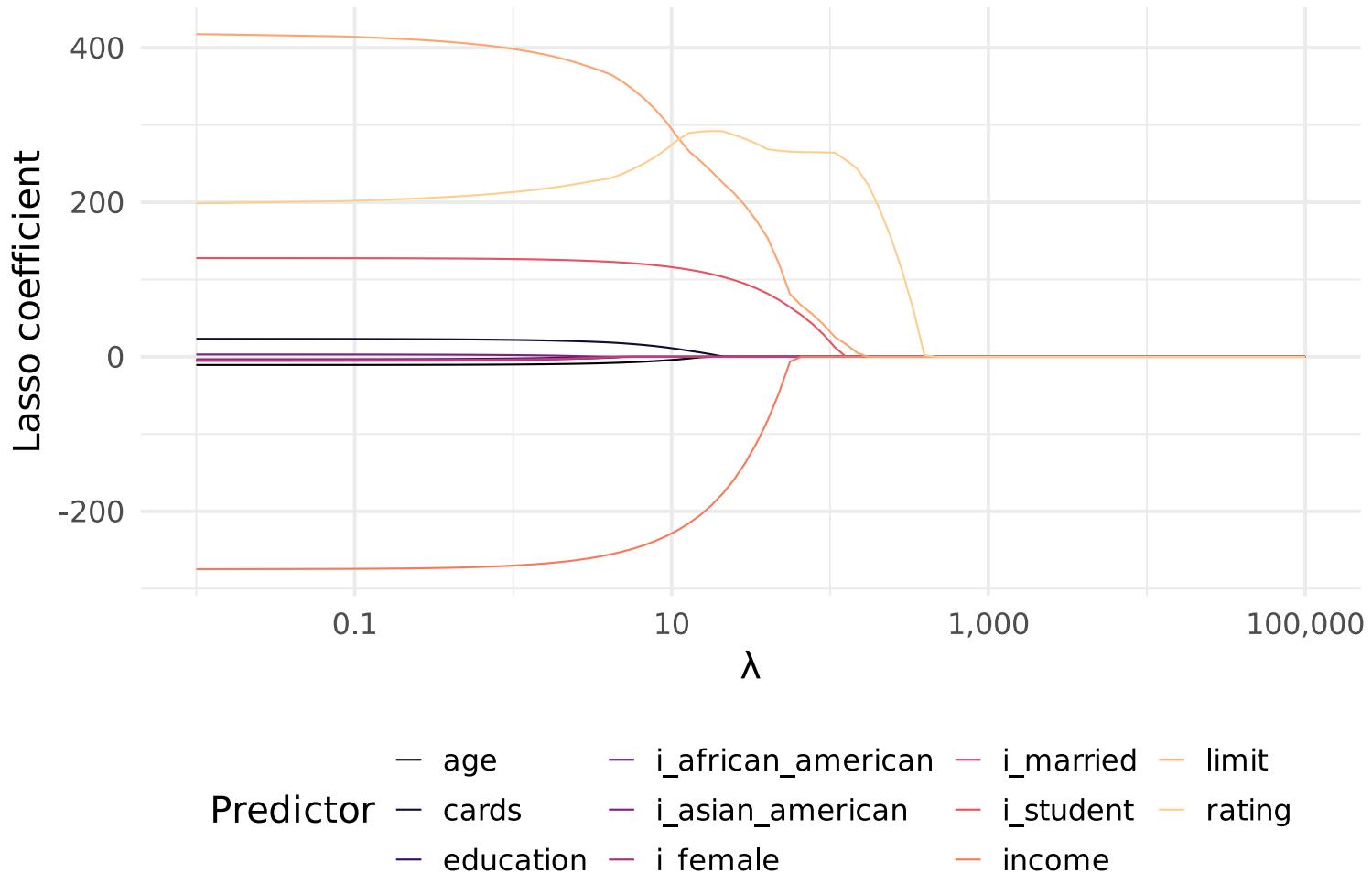
1. Some coefficients will be **set to zero**—we get "sparse" models.
2. Lasso can be used for subset/feature **selection**.

We will still need to carefully select  $\lambda$ .

## Ridge regression coefficients for $\lambda$ between 0.01 and 100,000



## Lasso coefficients for $\lambda$ between 0.01 and 100,000



# Machine learning

## Summary

Now you understand the basic tenants of machine learning:

- How **prediction** differs from causal inference
- **Bias-variance tradeoff** (the benefits and costs of flexibility)
- **Cross validation:** Performance and tuning
- In- vs. out-of-sample **performance**

But there's a lot more...

Trees (    )

# Trees ( )

## Fundamentals

### Decision trees

- split the *predictor space* (our **X**) into regions
- then predict the most-common value within a region

# Trees (🌲🌴🌳)

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**

# Trees (🌲🌴🌳)

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**

# Trees (🌲🌴🌳)

## Fundamentals

### Decision trees

- split the *predictor space* (our **X**) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**
3. are relatively **simple** and **interpretable**

# Trees (🌲🌴🌳)

## Fundamentals

### Decision trees

- split the *predictor space* (our **X**) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**
3. are relatively **simple** and **interpretable**
4. often **underperform** relative to competing methods

# Trees (🌲🌴🌳)

## Fundamentals

### Decision trees

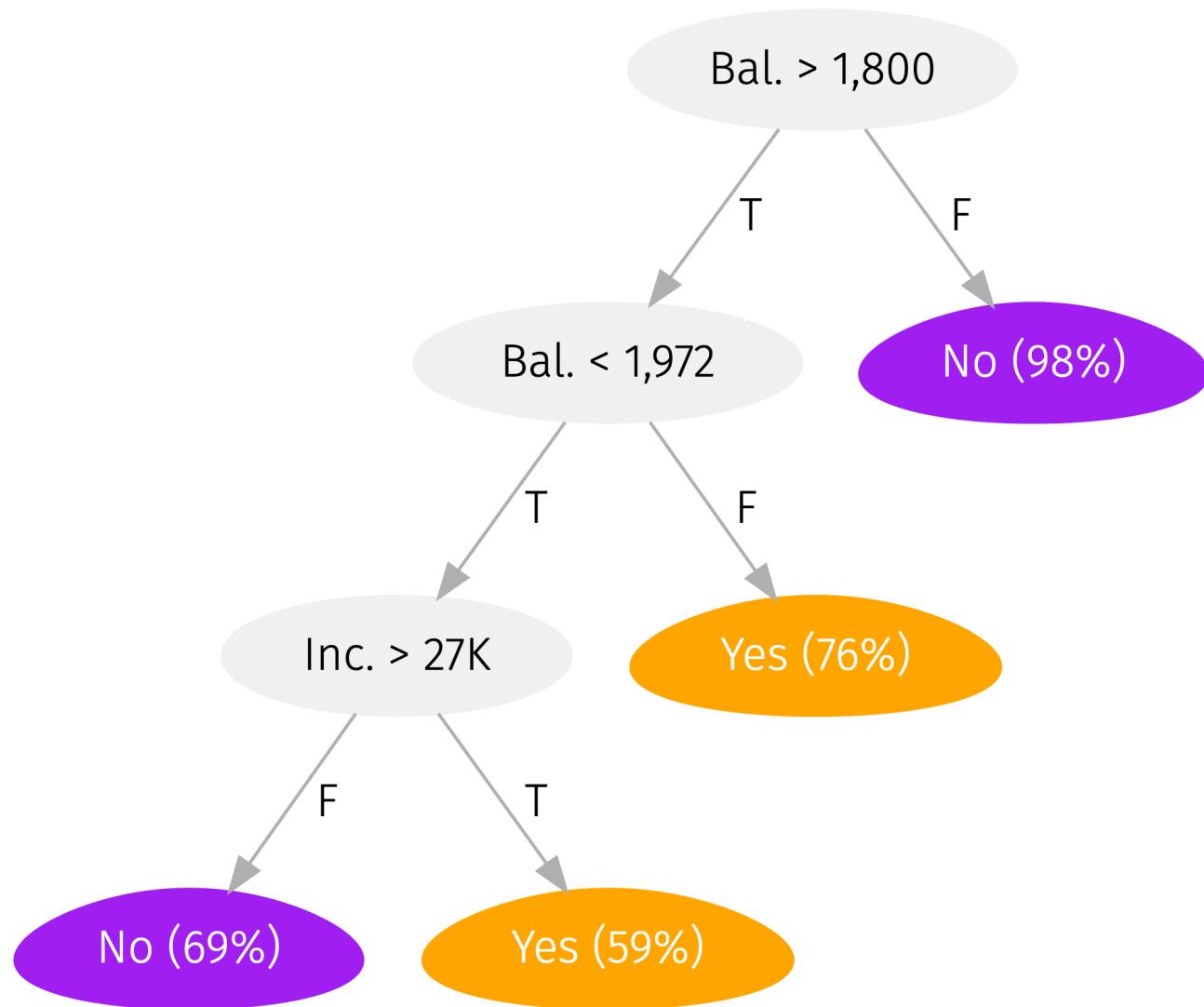
- split the *predictor space* (our **X**) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**
3. are relatively **simple** and **interpretable**
4. often **underperform** relative to competing methods
5. easily extend to **very competitive ensemble methods** (many trees) 🌲

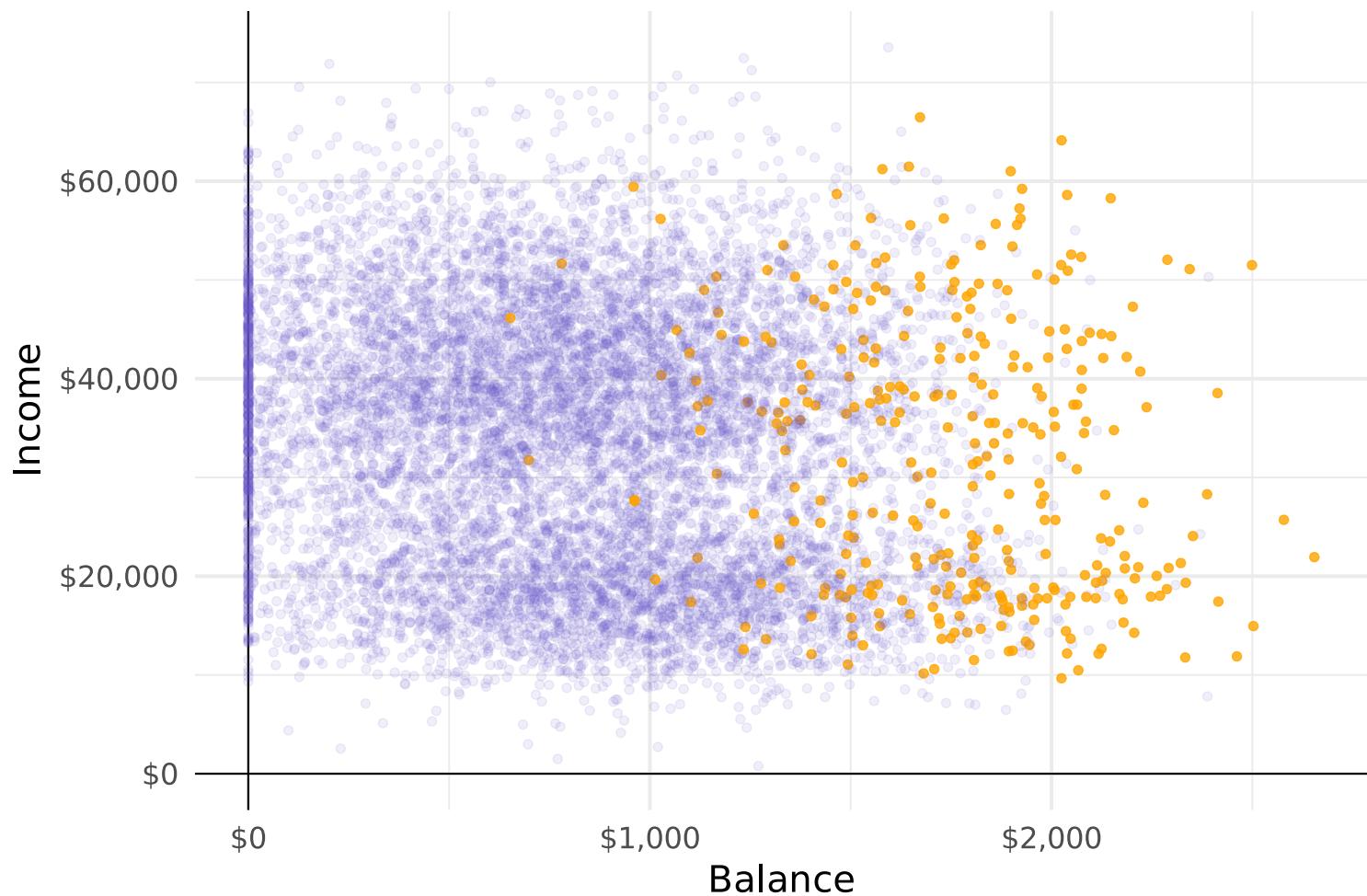
🌲 Though the ensembles will be much less interpretable.

*Example:* **A simple decision tree** classifying credit-card default



Let's see how the tree works

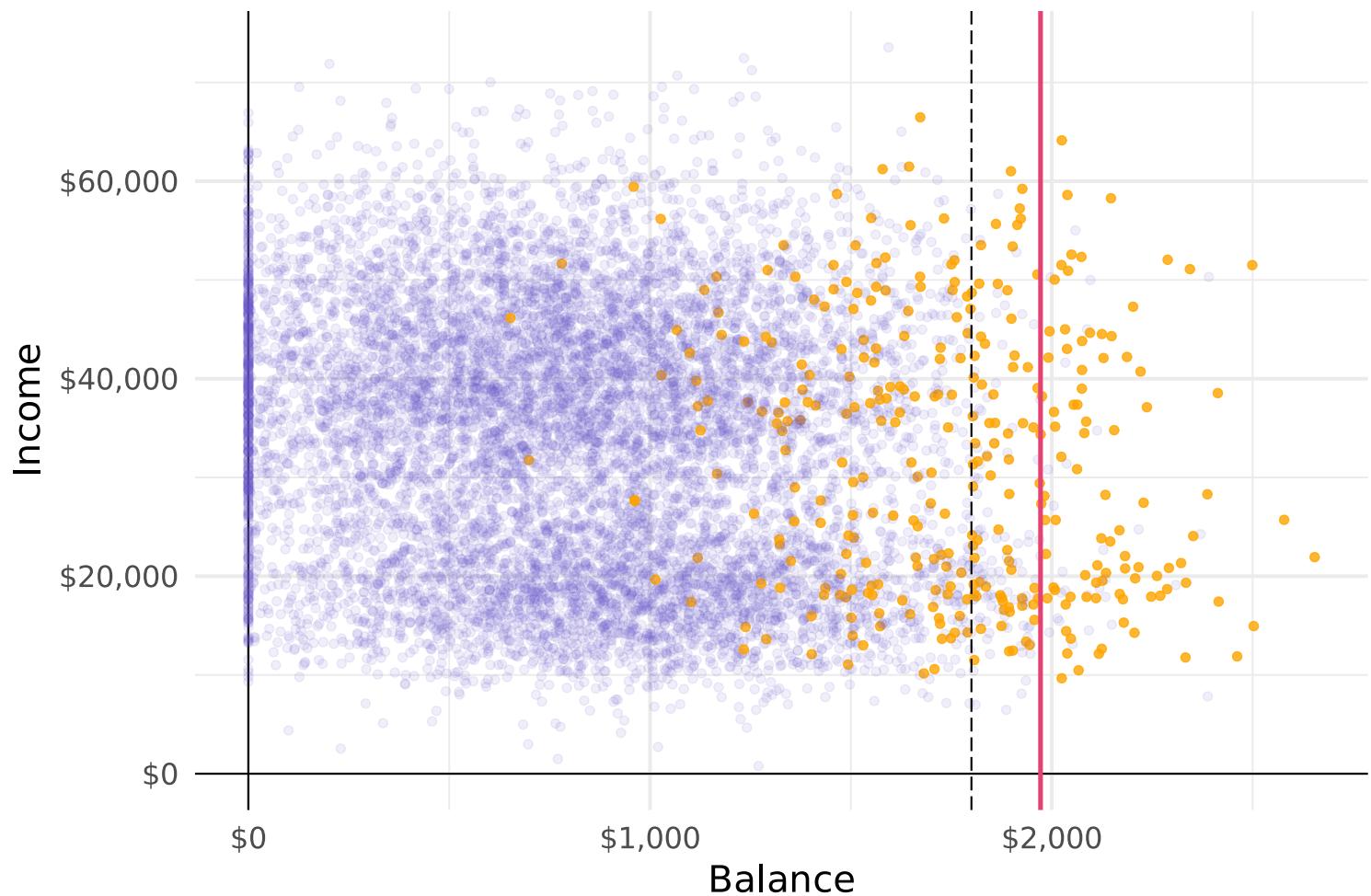
Let's see how the tree works—starting with credit data (default: Yes vs. No).



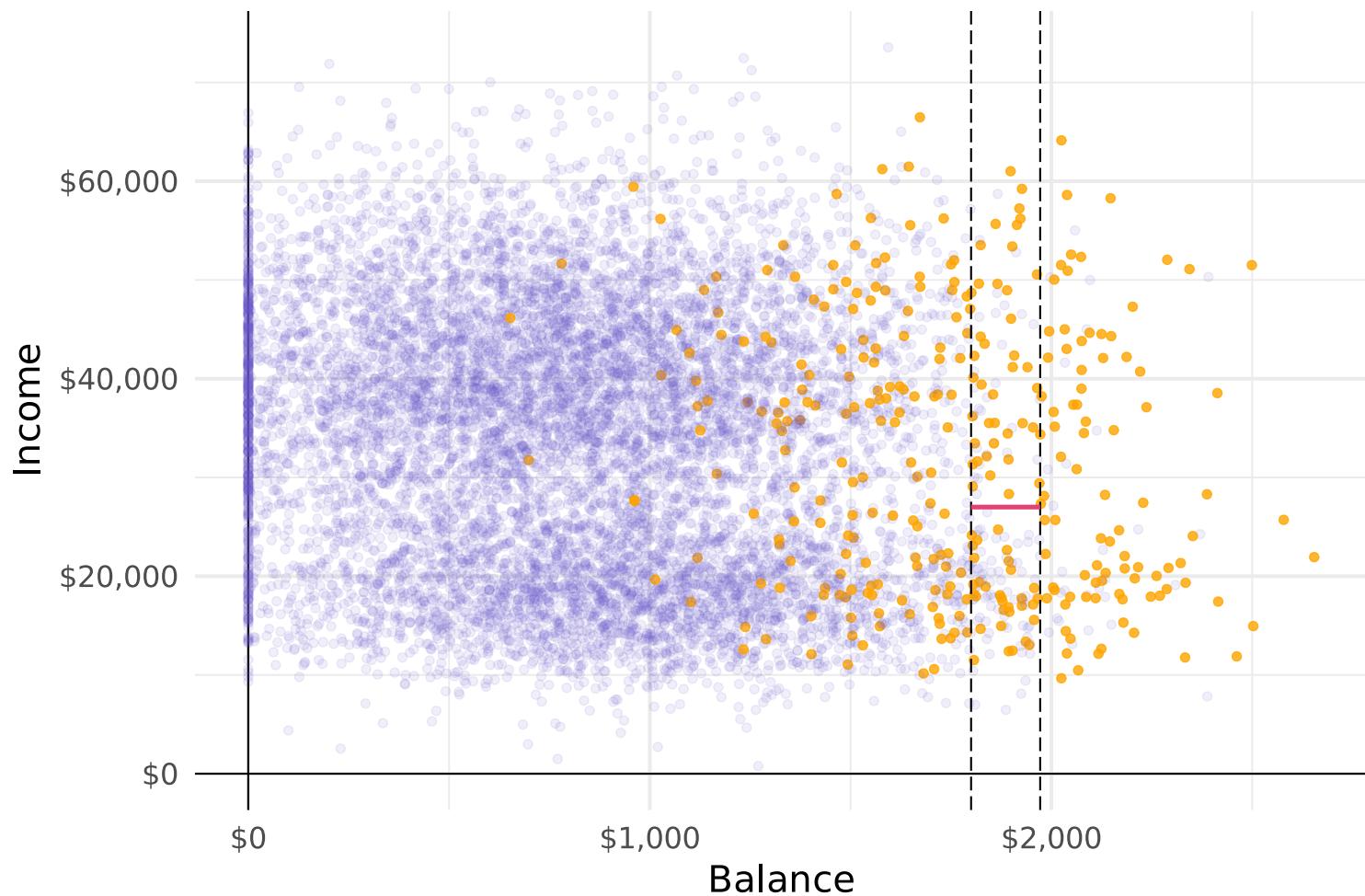
The **first partition** splits balance at \$1,800.



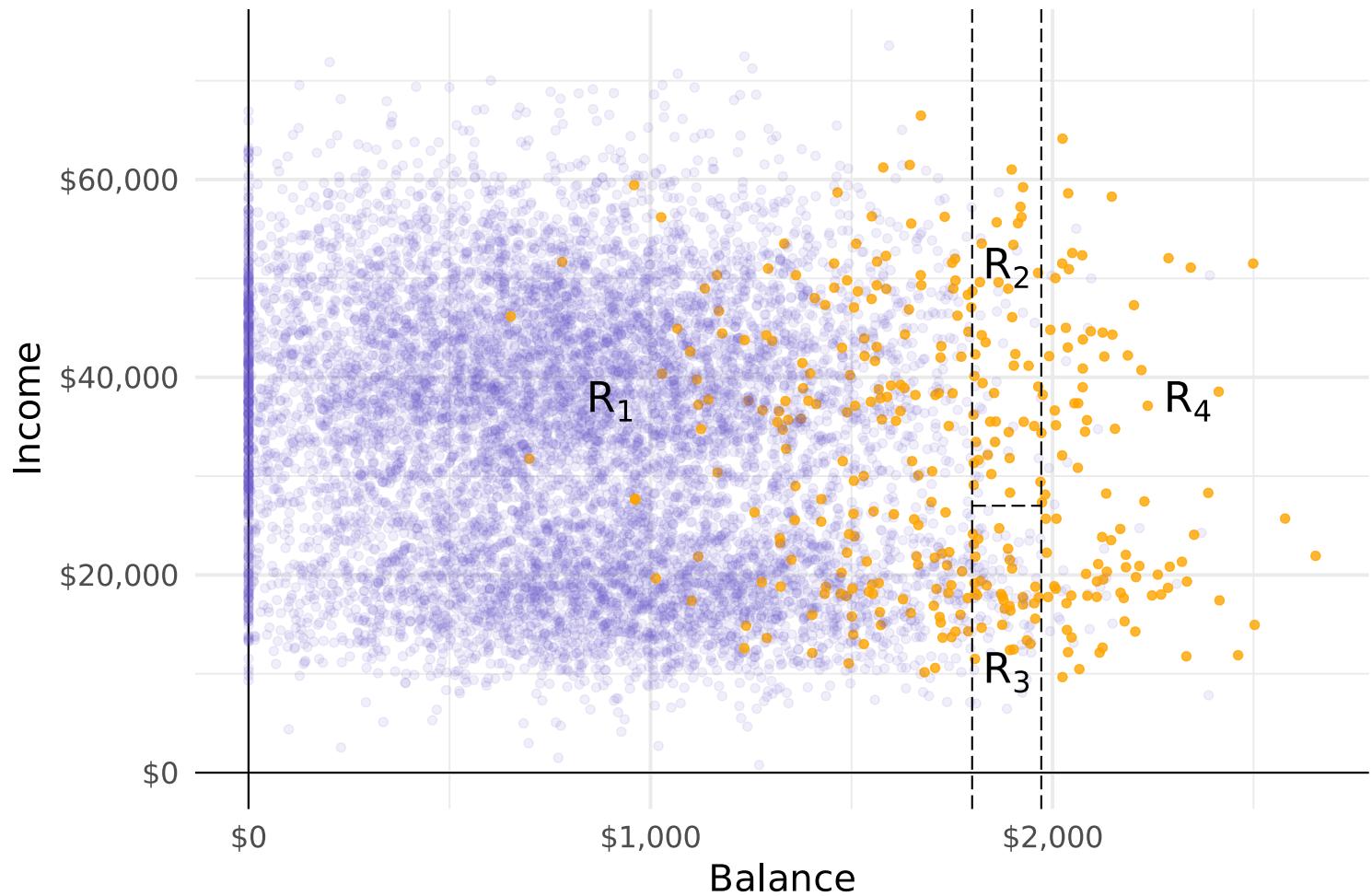
The **second partition** splits balance at \$1,972, (conditional on bal. > \$1,800).



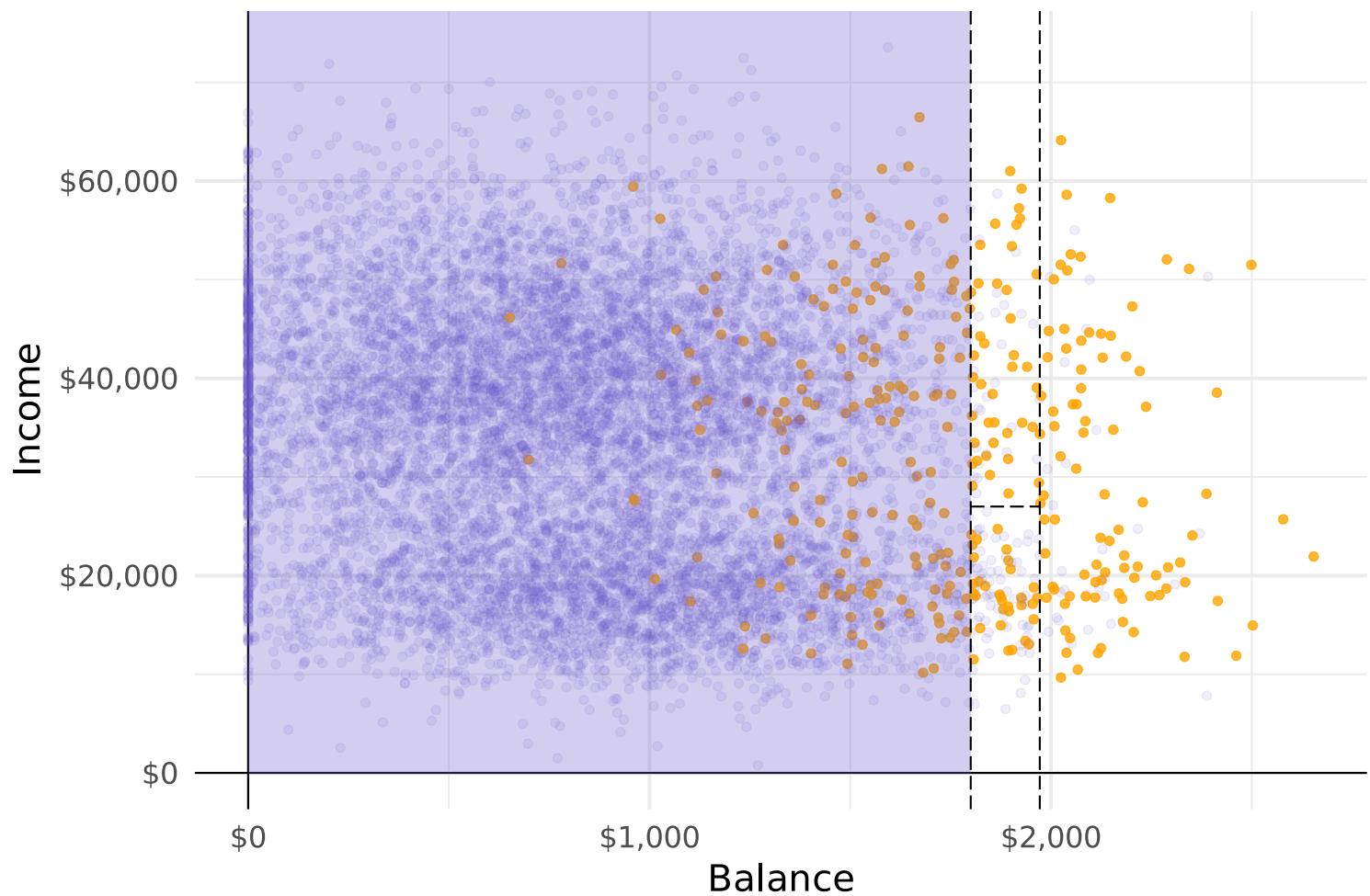
The **third partition** splits income at \$27K **for** bal. between \$1,800 and \$1,972.



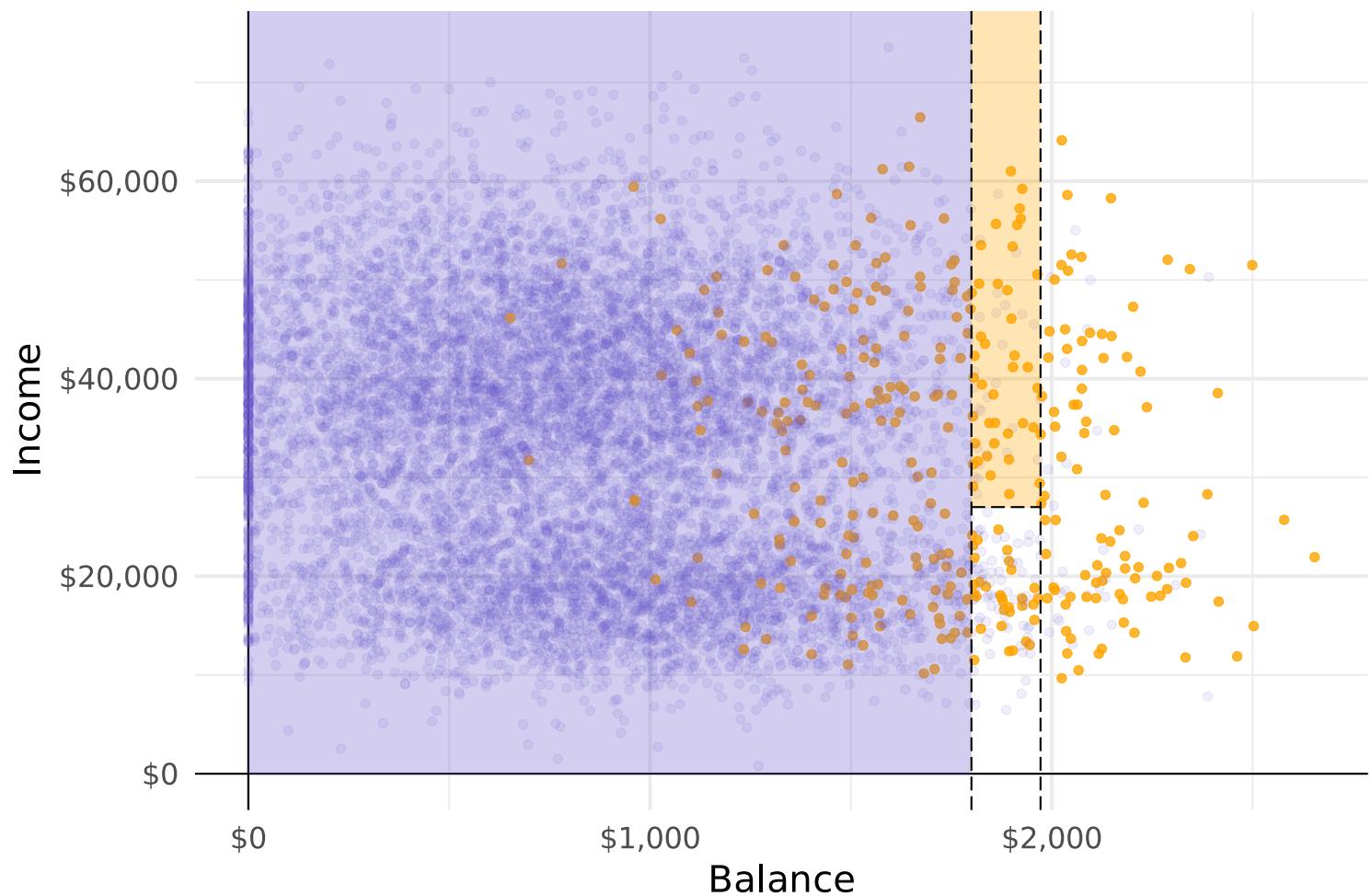
These three partitions give us four **regions**...



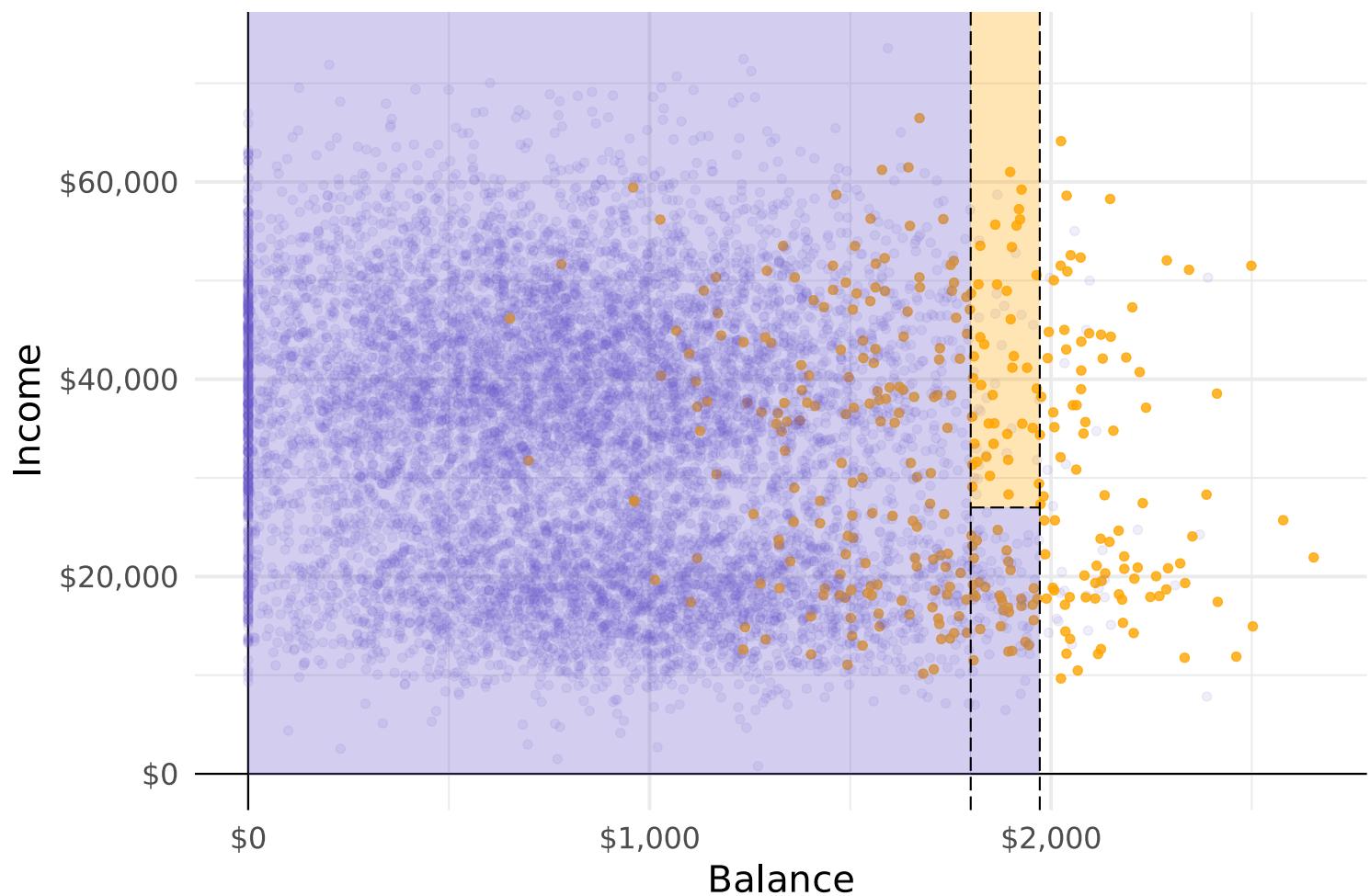
**Predictions** cover each region (e.g., using the region's most common class).



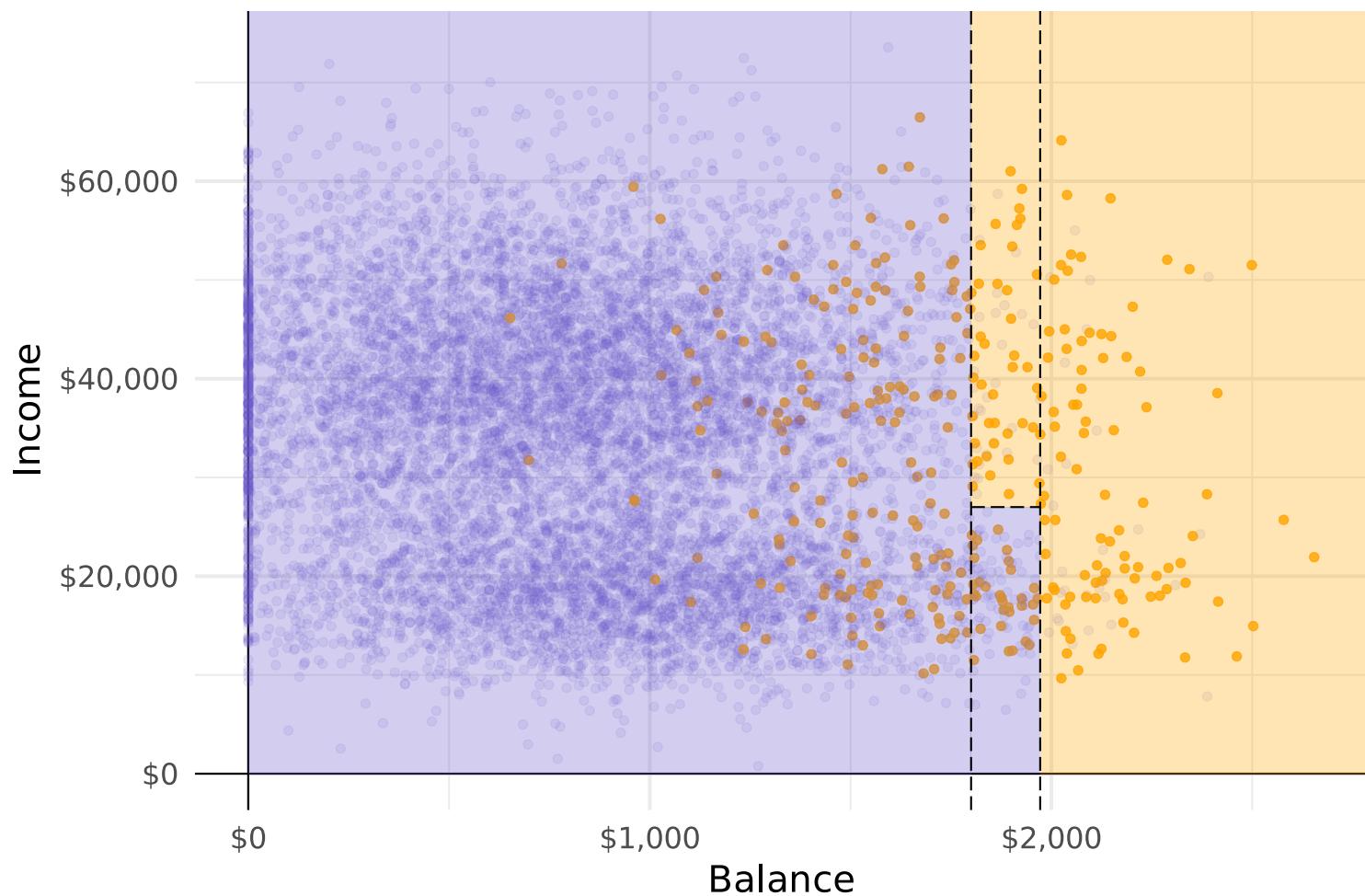
**Predictions** cover each region (e.g., using the region's most common class).



**Predictions** cover each region (e.g., using the region's most common class).



**Predictions** cover each region (e.g., using the region's most common class).



**Q** Where do trees come from?

**A** Seeds! 

**Q** Where do trees come from?

**A** Seeds! 

**Q** How do we train (grow) trees?

# Decision trees

## Growing trees

We will start with **regression trees**, i.e., trees used in regression settings.

# Decision trees

## Growing trees

We will start with **regression trees**, i.e., trees used in regression settings.

As we saw, the task of **growing a tree** involves two main steps:

1. **Divide the predictor space** into  $J$  regions (using predictors  $\mathbf{x}_1, \dots, \mathbf{x}_p$ )

# Decision trees

## Growing trees

We will start with **regression trees**, i.e., trees used in regression settings.

As we saw, the task of **growing a tree** involves two main steps:

1. **Divide the predictor space** into  $J$  regions (using predictors  $\mathbf{x}_1, \dots, \mathbf{x}_p$ )
2. **Make predictions** using the regions' mean outcome.

For region  $R_j$  predict  $\hat{y}_{R_j}$  where

$$\hat{y}_{R_j} = \frac{1}{n_j} \sum_{i \in R_j} y$$

# Decision trees

## Growing trees

We **choose the regions to minimize RSS** across all  $J$  regions, i.e.,

$$\sum_{j=1}^J \left( y_i - \hat{y}_{R_j} \right)^2$$

# Decision trees

## Growing trees

We **choose the regions to minimize RSS** across all  $J$  regions, i.e.,

$$\sum_{j=1}^J \left( y_i - \hat{y}_{R_j} \right)^2$$

**Problem:** Examining every possible partition is computationally infeasible.

# Decision trees

## Growing trees

We **choose the regions to minimize RSS** across all  $J$  regions, i.e.,

$$\sum_{j=1}^J \left( y_i - \hat{y}_{R_j} \right)^2$$

**Problem:** Examining every possible partition is computationally infeasible.

**Solution:** a *top-down, greedy* algorithm named **recursive binary splitting**

- **recursive** start with the "best" split, then find the next "best" split, ...
- **binary** each split creates two branches—"yes" and "no"
- **greedy** each step makes *best* split—no consideration of overall process

# Decision trees

## Growing trees: Choosing a split

*Recall* Regression trees choose the split that minimizes RSS.

To find this split, we need

1. a predictor,  $\mathbf{x}_j$
2. a cutoff  $s$  that splits  $\mathbf{x}_j$  into two parts: (1)  $\mathbf{x}_j < s$  and (2)  $\mathbf{x}_j \geq s$

# Decision trees

## Growing trees: Choosing a split

*Recall* Regression trees choose the split that minimizes RSS.

To find this split, we need

1. a predictor,  $\mathbf{x}_j$
2. a cutoff  $s$  that splits  $\mathbf{x}_j$  into two parts: (1)  $\mathbf{x}_j < s$  and (2)  $\mathbf{x}_j \geq s$

Searching across each of our predictors  $j$  and all of their cutoffs  $s$ , we choose the combination that **minimizes RSS**.

# Decision trees

## Example: Splitting

*Example* Consider the dataset

i	y	x <sub>1</sub>	x <sub>2</sub>
1	0	1	4
2	8	3	2
3	6	5	6

# Decision trees

## Example: Splitting

*Example* Consider the dataset

i	y	x <sub>1</sub>	x <sub>2</sub>
1	0	1	4
2	8	3	2
3	6	5	6

With just three observations, each variable only has two actual splits. 

 You can think about cutoffs as the ways we divide observations into two groups.

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	y	$x_1$	$x_2$
1	0	1	4
2	8	3	2
3	6	5	6

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	pred.	y	$x_1$	$x_2$
1	0	0	1	4
2	7	8	3	2
3	7	6	5	6

This split yields an RSS of  $0^2 + 1^2 + (-1)^2 = 2$ .

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	pred.	y	$x_1$	$x_2$
1	0	0	1	4
2	7	8	3	2
3	7	6	5	6

This split yields an RSS of  $0^2 + 1^2 + (-1)^2 = 2$ .

Note<sub>1</sub> Splitting  $x_1$  at 2 yields the same results as 1.5, 2.5—anything in (1, 3).

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	pred.	y	$x_1$	$x_2$
1	0	0	1	4
2	7	8	3	2
3	7	6	5	6

This split yields an RSS of  $0^2 + 1^2 + (-1)^2 = 2$ .

Note<sub>1</sub> Splitting  $x_1$  at 2 yields the same results as 1.5, 2.5—anything in (1, 3).

Note<sub>2</sub> Trees often grow until they hit some number of observations in a leaf.

# Decision trees

## Example: Splitting

An alternative split:  $x_1$  at 4, which yields (1)  $x_1 < 4$  vs. (2)  $x_1 \geq 4$

i	pred.	y	$x_1$	$x_2$
1	4	0	1	4
2	4	8	3	2
3	6	6	5	6

This split yields an RSS of  $(-4)^2 + 4^2 + 0^2 = 32$ .

# Decision trees

## Example: Splitting

An alternative split:  $x_1$  at 4, which yields (1)  $x_1 < 4$  vs. (2)  $x_1 \geq 4$

i	pred.	y	$x_1$	$x_2$
1	4	0	1	4
2	4	8	3	2
3	6	6	5	6

This split yields an RSS of  $(-4)^2 + 4^2 + 0^2 = 32$ .

Previous: Splitting  $x_1$  at 4 yielded RSS = 2. (*Much better*)

# Decision trees

## Example: Splitting

Another split:  $x_2$  at 3, which yields (1)  $x_1 < 3$  vs. (2)  $x_1 \geq 3$

i	pred.	y	$x_1$	$x_2$
1	3	0	1	4
2	8	8	3	2
3	3	6	5	6

This split yields an RSS of  $(-3)^2 + 0^2 + 3^2 = 18$ .

# Decision trees

## Example: Splitting

Final split:  $x_2$  at 5, which yields (1)  $x_1 < 5$  vs. (2)  $x_1 \geq 5$

i	pred.	y	$x_1$	$x_2$
1	4	0	1	4
2	4	8	3	2
3	6	6	5	6

This split yields an RSS of  $(-4)^2 + 4^2 + 0^2 = 32$ .

# Decision trees

## Example: Splitting

Across our four possible splits (two variables each with two splits)

- $x_1$  with a cutoff of 2: **RSS** = 2
- $x_1$  with a cutoff of 4: **RSS** = 32
- $x_2$  with a cutoff of 3: **RSS** = 18
- $x_2$  with a cutoff of 5: **RSS** = 32

our split of  $x_1$  at 2 generates the lowest RSS.

Note: Categorical predictors work in exactly the same way.

We want to try **all possible combinations** of the categories.

*Ex:* For a four-level categorical predictor (levels: A, B, C, D)

- Split 1: A|B|C vs. D
- Split 2: A|B|D vs. C
- Split 3: A|C|D vs. B
- Split 4: B|C|D vs. A
- Split 5: A|B vs. C|D
- Split 6: A|C vs. B|D
- Split 7: A|D vs. B|C

we would need to try 7 possible splits.

# Decision trees

## More splits

Once we make our a split, we then continue splitting, **conditional** on the regions from our previous splits.

So if our first split creates  $R_1$  and  $R_2$ , then our next split searches the predictor space only in  $R_1$  or  $R_2$ . 

 We are no longer searching the full space—it is conditional on the previous splits.

# Decision trees

## More splits

Once we make our a split, we then continue splitting, **conditional** on the regions from our previous splits.

So if our first split creates  $R_1$  and  $R_2$ , then our next split searches the predictor space only in  $R_1$  or  $R_2$ . 

The tree continue to **grow until** it hits some specified threshold, e.g., at most 5 observations in each leaf.

 We are no longer searching the full space—it is conditional on the previous splits.

# Decision trees

## Too many splits?

One can have too many splits.

Q Why?

# Decision trees

## Too many splits?

One can have too many splits.

**Q** Why?

**A** "More splits" means

1. more flexibility (think about the bias-variance tradeoff/overfitting)
2. less interpretability (one of the selling points for trees)

# Decision trees

## Too many splits?

One can have too many splits.

**Q** Why?

**A** "More splits" means

1. more flexibility (think about the bias-variance tradeoff/overfitting)
2. less interpretability (one of the selling points for trees)

**Q** So what can we do?

# Decision trees

## Too many splits?

One can have too many splits.

**Q** Why?

**A** "More splits" means

1. more flexibility (think about the bias-variance tradeoff/overfitting)
2. less interpretability (one of the selling points for trees)

**Q** So what can we do?

**A** Prune your trees!

# Decision trees

## Pruning

**Pruning** allows us to trim our trees back to their "best selves."

*The idea:* Some regions may increase **variance** more than they reduce **bias**. By removing these regions, we gain in test MSE.

*Candidates for trimming:* Regions that do not **reduce RSS** very much.

# Decision trees

## Pruning

**Pruning** allows us to trim our trees back to their "best selves."

*The idea:* Some regions may increase **variance** more than they reduce **bias**. By removing these regions, we gain in test MSE.

*Candidates for trimming:* Regions that do not **reduce RSS** very much.

*Updated strategy:* Grow big trees  $T_0$  and then trim  $T_0$  to an optimal **subtree**.

# Decision trees

## Pruning

**Pruning** allows us to trim our trees back to their "best selves."

*The idea:* Some regions may increase **variance** more than they reduce **bias**. By removing these regions, we gain in test MSE.

*Candidates for trimming:* Regions that do not **reduce RSS** very much.

*Updated strategy:* Grow big trees  $T_0$  and then trim  $T_0$  to an optimal **subtree**.

*Updated problem:* Considering all possible subtrees can get expensive.

# Decision trees

## Pruning

Cost-complexity pruning<sup>▲</sup> offers a solution.

Just as we did with lasso, **cost-complexity pruning** forces the tree to pay a price (penalty) to become more complex.

Complexity here is defined as the number of regions  $|T|$ .

▲ Also called: *weakest-link pruning*.

# Decision trees

## Pruning

Specifically, **cost-complexity pruning** adds a penalty of  $\alpha|T|$  to the RSS, *i.e.*,

$$\sum_{m=1}^{|T|} \sum_{i:x \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

For any value of  $\alpha(\geq 0)$ , we get a subtree  $T \subset T_0$ .

# Decision trees

## Pruning

Specifically, **cost-complexity pruning** adds a penalty of  $\alpha|T|$  to the RSS, i.e.,

$$\sum_{m=1}^{|T|} \sum_{i:x \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

For any value of  $\alpha (\geq 0)$ , we get a subtree  $T \subset T_0$ .

$\alpha = 0$  generates  $T_0$ , but as  $\alpha$  increases, we begin to cut back the tree.

# Decision trees

## Pruning

Specifically, **cost-complexity pruning** adds a penalty of  $\alpha|T|$  to the RSS, i.e.,

$$\sum_{m=1}^{|T|} \sum_{i:x \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

For any value of  $\alpha (\geq 0)$ , we get a subtree  $T \subset T_0$ .

$\alpha = 0$  generates  $T_0$ , but as  $\alpha$  increases, we begin to cut back the tree.

We choose  $\alpha$  via cross validation.

# Decision trees

## Classification trees

Classification with trees is very similar to regression.

# Decision trees

## Classification trees

Classification with trees is very similar to regression.

### Regression trees

- **Predict:** Region's mean
- **Split:** Minimize RSS
- **Prune:** Penalized RSS

# Decision trees

## Classification trees

Classification with trees is very similar to regression.

### Regression trees

- **Predict:** Region's mean
- **Split:** Minimize RSS
- **Prune:** Penalized RSS

### Classification trees

- **Predict:** Region's mode
- **Split:** Min. Gini or entropy 
- **Prune:** Penalized error rate 

An additional nuance for **classification trees**: We typically care about the **proportions of classes in the leaves**—not just the final prediction.

 Defined on the next slide.  ... or Gini index or entropy

# Decision trees

## The Gini index

Let  $\hat{p}_{mk}$  denote the proportion of observations in class  $k$  and region  $m$ .

# Decision trees

## The Gini index

Let  $\hat{p}_{mk}$  denote the proportion of observations in class  $k$  and region  $m$ .

The **Gini index** tells us about a region's "purity"

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

if a region is very homogeneous, then the Gini index will be small.

Homogenous regions are easier to predict.

Reducing the Gini index yields to more homogeneous regions

∴ We want to minimize the Gini index.

 This vocabulary is Voldemort's contribution to the machine-learning literature.

# Decision trees

## Entropy

Let  $\hat{p}_{mk}$  denote the proportion of observations in class  $k$  and region  $m$ .

**Entropy** also measures the "purity" of a node/leaf

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$$

**Entropy** is also minimized when  $\hat{p}_{mk}$  values are close to 0 and 1.

# Decision trees

## Rational

Q Why are we using the Gini index or entropy (vs. error rate)?

# Decision trees

## Rational

**Q** Why are we using the Gini index or entropy (vs. error rate)?

**A** The error rate isn't sufficiently sensitive to grow good trees.

The Gini index and entropy tell us about the **composition** of the leaf.

# Decision trees

## Rational

**Q** Why are we using the Gini index or entropy (vs. error rate)?

**A** The error rate isn't sufficiently sensitive to grow good trees.

The Gini index and entropy tell us about the **composition** of the leaf.

*Ex.* Consider two different leaves in a three-level classification.

### Leaf 1

- **A:** 51, **B:** 49, **C:** 00
- **Error rate:** 49%
- **Gini index:** 0.4998
- **Entropy:** 0.6929

### Leaf 2

- **A:** 51, **B:** 25, **C:** 24
- **Error rate:** 49%
- **Gini index:** 0.6198
- **Entropy:** 1.0325

The **Gini index** and **entropy** tell us about the distribution.

# Decision trees

## Classification trees

When **growing** classification trees, we want to use the Gini index or entropy.

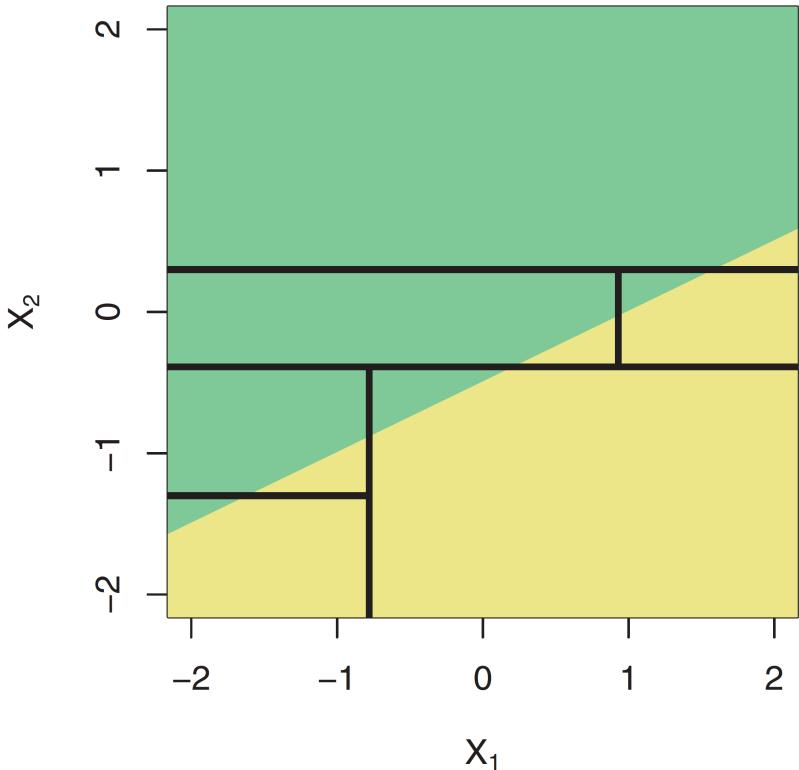
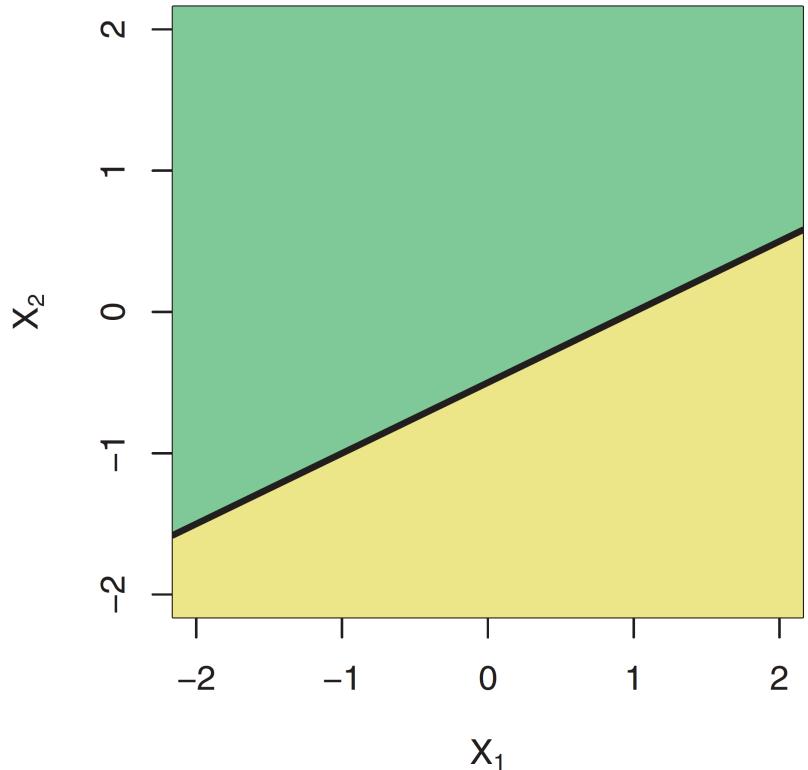
However, when **pruning**, the error rate is typically fine—especially if accuracy will be the final criterion.

**Q** How do trees compare to linear models?

**Q** How do trees compare to linear models?

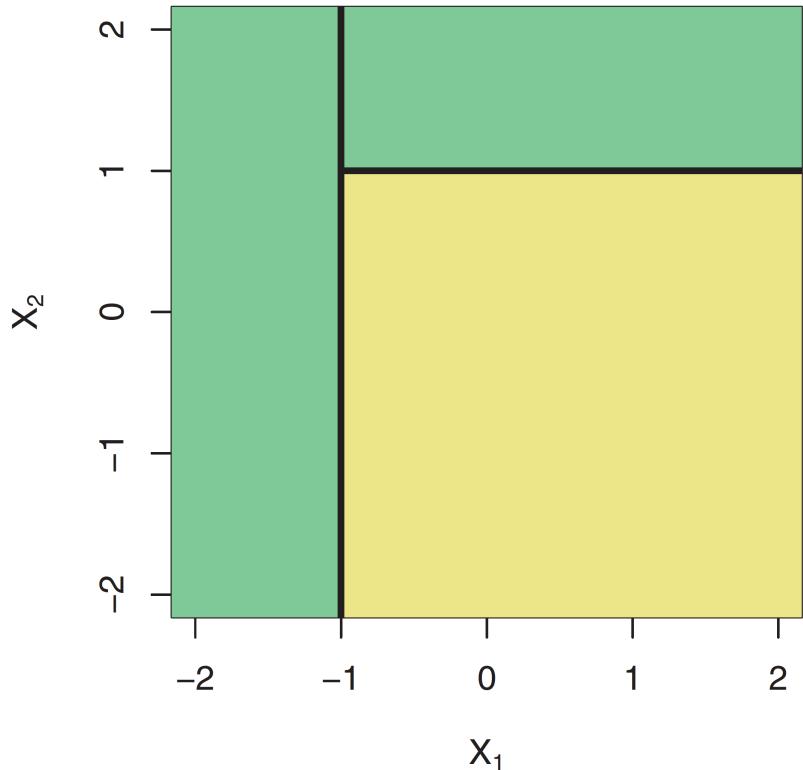
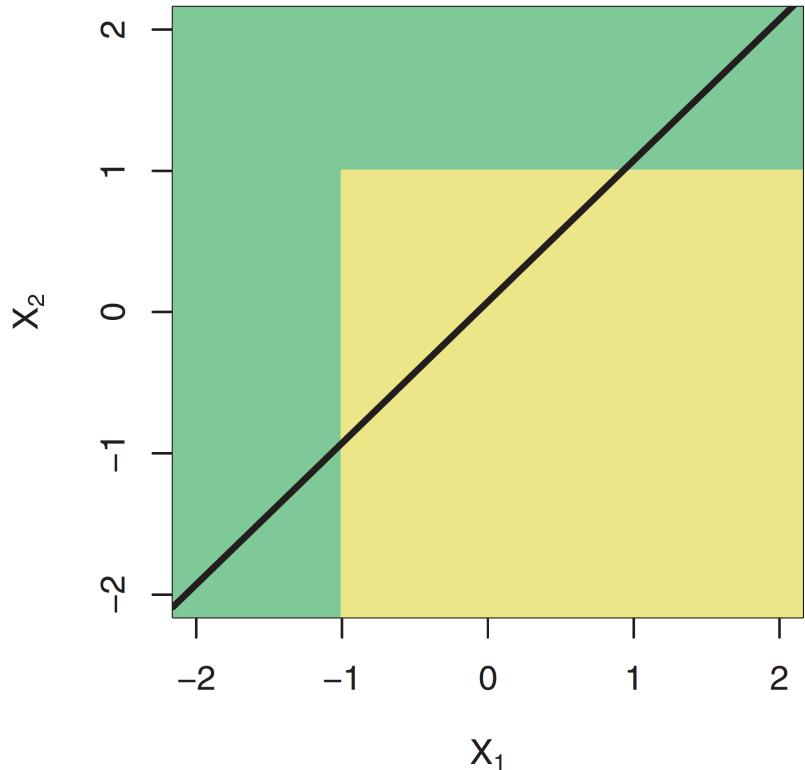
**A** It depends how linear the true boundary is.

**Linear boundary:** trees struggle to recreate a line.



Source: ISL, p. 315

**Nonlinear boundary:** trees easily replicate the nonlinear boundary.



Source: ISL, p. 315

# Decision trees

## Strengths and weaknesses

As with any method, decision trees have tradeoffs.

### Strengths

- + Easily explained/interpreted
- + Include several graphical options
- + Mirror human decision making?
- + Handle num. or cat. on LHS/RHS 

-  Without needing to create lots of dummy variables!

# Decision trees

## Strengths and weaknesses

As with any method, decision trees have tradeoffs.

### Strengths

- + Easily explained/interpreted
- + Include several graphical options
- + Mirror human decision making?
- + Handle num. or cat. on LHS/RHS

### Weaknesses

- Outperformed by other methods
- Struggle with linearity
- Can be very "non-robust"

**Non-robust:** Small data changes can cause huge changes in our tree.

 Without needing to create lots of dummy variables!

# Decision trees

## Strengths and weaknesses

As with any method, decision trees have tradeoffs.

### Strengths

- + Easily explained/interpreted
- + Include several graphical options
- + Mirror human decision making?
- + Handle num. or cat. on LHS/RHS 

### Weaknesses

- Outperformed by other methods
- Struggle with linearity
- Can be very "non-robust"

**Non-robust:** Small data changes can cause huge changes in our tree.

Next: Create ensembles of trees  to strengthen these weaknesses. 

-  Without needing to create lots of dummy variables!
-  Forests!  Which will also weaken some of the strengths.

# Ensemble methods

# Ensemble methods

## Intro

Rather than focusing on training a **single**, highly accurate model, **ensemble methods** combine **many** low-accuracy models into a *meta-model*.

# Ensemble methods

## Intro

Rather than focusing on training a **single**, highly accurate model, **ensemble methods** combine **many** low-accuracy models into a *meta-model*.

Today: Three common methods for **combining individual trees**

1. **Bagging**
2. **Random forests**
3. **Boosting**

# Ensemble methods

## Intro

Rather than focusing on training a **single**, highly accurate model, **ensemble methods** combine **many** low-accuracy models into a *meta-model*.

Today: Three common methods for **combining individual trees**

1. **Bagging**
2. **Random forests**
3. **Boosting**

**Why?** While individual trees may be highly variable and inaccurate, a combination of trees is often quite stable and accurate.

# Ensemble methods

## Intro

Rather than focusing on training a **single**, highly accurate model, **ensemble methods** combine **many** low-accuracy models into a *meta-model*.

Today: Three common methods for **combining individual trees**

1. **Bagging**
2. **Random forests**
3. **Boosting**

**Why?** While individual trees may be highly variable and inaccurate, a combination of trees is often quite stable and accurate. 

 We will lose interpretability.

# Ensemble methods

## Bagging

Bagging creates additional samples via bootstrapping.

# Ensemble methods

## Bagging

Bagging creates additional samples via bootstrapping.

Q How does bootstrapping help?

# Ensemble methods

## Bagging

Bagging creates additional samples via bootstrapping.

Q How does bootstrapping help?

A Recall: Individual decision trees suffer from variability (*non-robust*).

# Ensemble methods

## Bagging

Bagging creates additional samples via bootstrapping.

Q How does bootstrapping help?

A Recall: Individual decision trees suffer from variability (*non-robust*).

This *non-robustness* means trees can change *a lot* based upon which observations are included/excluded.

# Ensemble methods

## Bagging

Bagging creates additional samples via bootstrapping.

Q How does bootstrapping help?

A Recall: Individual decision trees suffer from variability (*non-robust*).

This *non-robustness* means trees can change *a lot* based upon which observations are included/excluded.

We're essentially using many "draws" instead of a single one. 

 Recall that an estimator's variance typically decreases as the sample size increases.

# Ensemble methods

## Bagging

Bootstrap aggregation (bagging) reduces this type of variability.

1. Create  $B$  bootstrapped samples
2. Train an estimator (tree)  $\hat{f}^b(x)$  on each of the  $B$  samples
3. Aggregate across your  $B$  bootstrapped models:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

This aggregated model  $\hat{f}_{\text{bag}}(x)$  is your final model.

# Ensemble methods

## Bagging trees

When we apply bagging to decision trees,

- we typically **grow the trees deep and do not prune**
- for **regression**, we **average** across the  $B$  trees' regions
- for **classification**, we have more options—but often take **plurality**

# Ensemble methods

## Bagging trees

When we apply bagging to decision trees,

- we typically **grow the trees deep and do not prune**
- for **regression**, we **average** across the  $B$  trees' regions
- for **classification**, we have more options—but often take **plurality**

**Individual** (unpruned) trees will be very **flexible** and **noisy**,  
but their **aggregate** will be quite **stable**.

# Ensemble methods

## Bagging trees

When we apply bagging to decision trees,

- we typically **grow the trees deep and do not prune**
- for **regression**, we **average** across the  $B$  trees' regions
- for **classification**, we have more options—but often take **plurality**

**Individual** (unpruned) trees will be very **flexible** and **noisy**,  
but their **aggregate** will be quite **stable**.

The number of trees  $B$  is generally not critical with bagging.  
 $B = 100$  often works fine.

# Ensemble methods

## Out-of-bag error estimation

Bagging also offers a convenient method for evaluating performance.

# Ensemble methods

## Out-of-bag error estimation

Bagging also offers a convenient method for evaluating performance.

For any bootstrapped sample, we omit  $\sim n/3$  observations.

**Out-of-bag (OOB) error estimation** estimates the test error rate using observations **randomly omitted** from each bootstrapped sample.

# Ensemble methods

## Out-of-bag error estimation

Bagging also offers a convenient method for evaluating performance.

For any bootstrapped sample, we omit  $\sim n/3$  observations.

**Out-of-bag (OOB) error estimation** estimates the test error rate using observations **randomly omitted** from each bootstrapped sample.

For each observation  $i$ :

1. Find all samples  $S_i$  in which  $i$  was omitted from training.
2. Aggregate the  $|S_i|$  predictions  $\hat{f}^b(\mathbf{x}_i)$ , e.g., using their mean or mode
3. Calculate the error, e.g.,  $y_i - \hat{f}_{i,\text{OOB},i}(x_i)$

# Ensemble methods

## Out-of-bag error estimation

When  $B$  is big enough, the OOB error rate will be very close to LOOCV.

# Ensemble methods

## Out-of-bag error estimation

When  $B$  is big enough, the OOB error rate will be very close to LOOCV.

Q Why use OOB error rate?

# Ensemble methods

## Out-of-bag error estimation

When  $B$  is big enough, the OOB error rate will be very close to LOOCV.

**Q** Why use OOB error rate?

**A** When  $B$  and  $n$  are large, cross validation—with any number of folds—can become pretty computationally intensive.

# Ensemble methods

## Bagging

Bagging has one additional shortcoming...

If one variable dominates other variables, the **trees will be very correlated.**

# Ensemble methods

## Bagging

Bagging has one additional shortcoming...

If one variable dominates other variables, the **trees will be very correlated.**

If the trees are very correlated, then bagging loses its advantage.

# Ensemble methods

## Bagging

Bagging has one additional shortcoming...

If one variable dominates other variables, the **trees will be very correlated.**

If the trees are very correlated, then bagging loses its advantage.

*Solution* We should make the trees less correlated.

# Ensemble methods

## Random forests

**Random forests** improve upon bagged trees by *decorrelating* the trees.

# Ensemble methods

## Random forests

**Random forests** improve upon bagged trees by *decorrelating* the trees.

In order to decorrelate its trees, a **random forest** only **considers** a random subset of  $m$  ( $\approx \sqrt{p}$ ) **predictors** when making each split (for each tree).

# Ensemble methods

## Random forests

Random forests improve upon bagged trees by *decorrelating* the trees.

In order to decorrelate its trees, a random forest only considers a random subset of  $m$  ( $\approx \sqrt{p}$ ) predictors when making each split (for each tree).

Restricting the variables our tree sees at a given split

# Ensemble methods

## Random forests

**Random forests** improve upon bagged trees by *decorrelating* the trees.

In order to decorrelate its trees, a **random forest** only **considers a random subset of  $m$  ( $\approx \sqrt{p}$ ) predictors** when making each split (for each tree).

Restricting the variables our tree sees at a given split

- nudges trees away from always using the same variables,

# Ensemble methods

## Random forests

Random forests improve upon bagged trees by *decorrelating* the trees.

In order to decorrelate its trees, a random forest only considers a random subset of  $m$  ( $\approx \sqrt{p}$ ) predictors when making each split (for each tree).

Restricting the variables our tree sees at a given split

- nudges trees away from always using the same variables,
- increasing the variation across trees in our forest,

# Ensemble methods

## Random forests

Random forests improve upon bagged trees by *decorrelating* the trees.

In order to decorrelate its trees, a random forest only considers a random subset of  $m$  ( $\approx \sqrt{p}$ ) predictors when making each split (for each tree).

Restricting the variables our tree sees at a given split

- nudges trees away from always using the same variables,
- increasing the variation across trees in our forest,
- which potentially reduces the variance of our estimates.

# Ensemble methods

## Random forests

Random forests improve upon bagged trees by *decorrelating* the trees.

In order to decorrelate its trees, a random forest only considers a random subset of  $m$  ( $\approx \sqrt{p}$ ) predictors when making each split (for each tree).

Restricting the variables our tree sees at a given split

- nudges trees away from always using the same variables,
- increasing the variation across trees in our forest,
- which potentially reduces the variance of our estimates.

If our predictors are very correlated, we may want to shrink  $m$ .

# Ensemble methods

## Random forests

Random forests thus introduce **two dimensions of random variation**

1. the **bootstrapped sample**
2. the  $m$  **randomly selected predictors** (for the split)

Everything else about random forests works just as it did with bagging. 

 And just as it did with plain, old decision trees.

# Ensemble methods

## Boosting

So far, the elements of our ensembles have been acting independently: any single tree knows nothing about the rest of the forest.

# Ensemble methods

## Boosting

So far, the elements of our ensembles have been acting independently: any single tree knows nothing about the rest of the forest.

**Boosting** allows trees to pass on information to each other.

# Ensemble methods

## Boosting

So far, the elements of our ensembles have been acting independently: any single tree knows nothing about the rest of the forest.

**Boosting** allows trees to pass on information to each other.

Specifically, **boosting** trains its trees  *sequentially*—each new tree trains on the residuals (mistakes) from its predecessors.

 As with bagging, boosting can be applied to many methods (in addition to trees).

# Ensemble methods

## Boosting

So far, the elements of our ensembles have been acting independently: any single tree knows nothing about the rest of the forest.

**Boosting** allows trees to pass on information to each other.

Specifically, **boosting** trains its trees  sequentially—each new tree trains on the residuals (mistakes) from its predecessors.

- We add each new tree to our model  $\hat{f}$  (and update our residuals).
- Trees are typically small—slowly improving  $\hat{f}$  where it struggles.

 As with bagging, boosting can be applied to many methods (in addition to trees).

# Ensemble methods

## Boosting

Boosting has three **tuning parameters**.

1. The **number of trees**  $B$  can be important to prevent overfitting.

# Ensemble methods

## Boosting

Boosting has three **tuning parameters**.

1. The **number of trees**  $B$  can be important to prevent overfitting.
2. The **shrinkage parameter**  $\lambda$ , which controls boosting's *learning rate* (often 0.01 or 0.001).

# Ensemble methods

## Boosting

Boosting has three **tuning parameters**.

1. The **number of trees**  $B$  can be important to prevent overfitting.
2. The **shrinkage parameter**  $\lambda$ , which controls boosting's *learning rate* (often 0.01 or 0.001).
3. The **number of splits**  $d$  in each tree (trees' complexity).

# Ensemble methods

## Boosting

Boosting has three **tuning parameters**.

1. The **number of trees**  $B$  can be important to prevent overfitting.
2. The **shrinkage parameter**  $\lambda$ , which controls boosting's *learning rate* (often 0.01 or 0.001).
3. The **number of splits**  $d$  in each tree (trees' complexity).
  - Individual trees are typically short—often  $d = 1$  ("stumps").
  - *Remember* Trees learn from predecessors' mistakes, so no single tree needs to offer a perfect model.

# Ensemble methods

## How to boost

**Step 1:** Set  $\hat{f}(x) = 0$ , which yields residuals  $r_i = y_i$  for all  $i$ .

# Ensemble methods

## How to boost

**Step 1:** Set  $\hat{f}(x) = 0$ , which yields residuals  $r_i = y_i$  for all  $i$ .

**Step 2:** For  $b = 1, 2 \dots, B$  do:

A. Fit a tree  $\hat{f}^b$  with  $d$  splits.

# Ensemble methods

## How to boost

**Step 1:** Set  $\hat{f}(x) = 0$ , which yields residuals  $r_i = y_i$  for all  $i$ .

**Step 2:** For  $b = 1, 2 \dots, B$  do:

**A.** Fit a tree  $\hat{f}^b$  with  $d$  splits.

**B.** Update the model  $\hat{f}$  with "shrunken version" of new treee  $\hat{f}^b$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

# Ensemble methods

## How to boost

**Step 1:** Set  $\hat{f}(x) = 0$ , which yields residuals  $r_i = y_i$  for all  $i$ .

**Step 2:** For  $b = 1, 2 \dots, B$  do:

**A.** Fit a tree  $\hat{f}^b$  with  $d$  splits.

**B.** Update the model  $\hat{f}$  with "shrunken version" of new treee  $\hat{f}^b$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

**C.** Update the residuals:  $r_i \leftarrow r_i - \lambda \hat{f}^b(x)$ .

# Ensemble methods

## How to boost

**Step 1:** Set  $\hat{f}(x) = 0$ , which yields residuals  $r_i = y_i$  for all  $i$ .

**Step 2:** For  $b = 1, 2 \dots, B$  do:

**A.** Fit a tree  $\hat{f}^b$  with  $d$  splits.

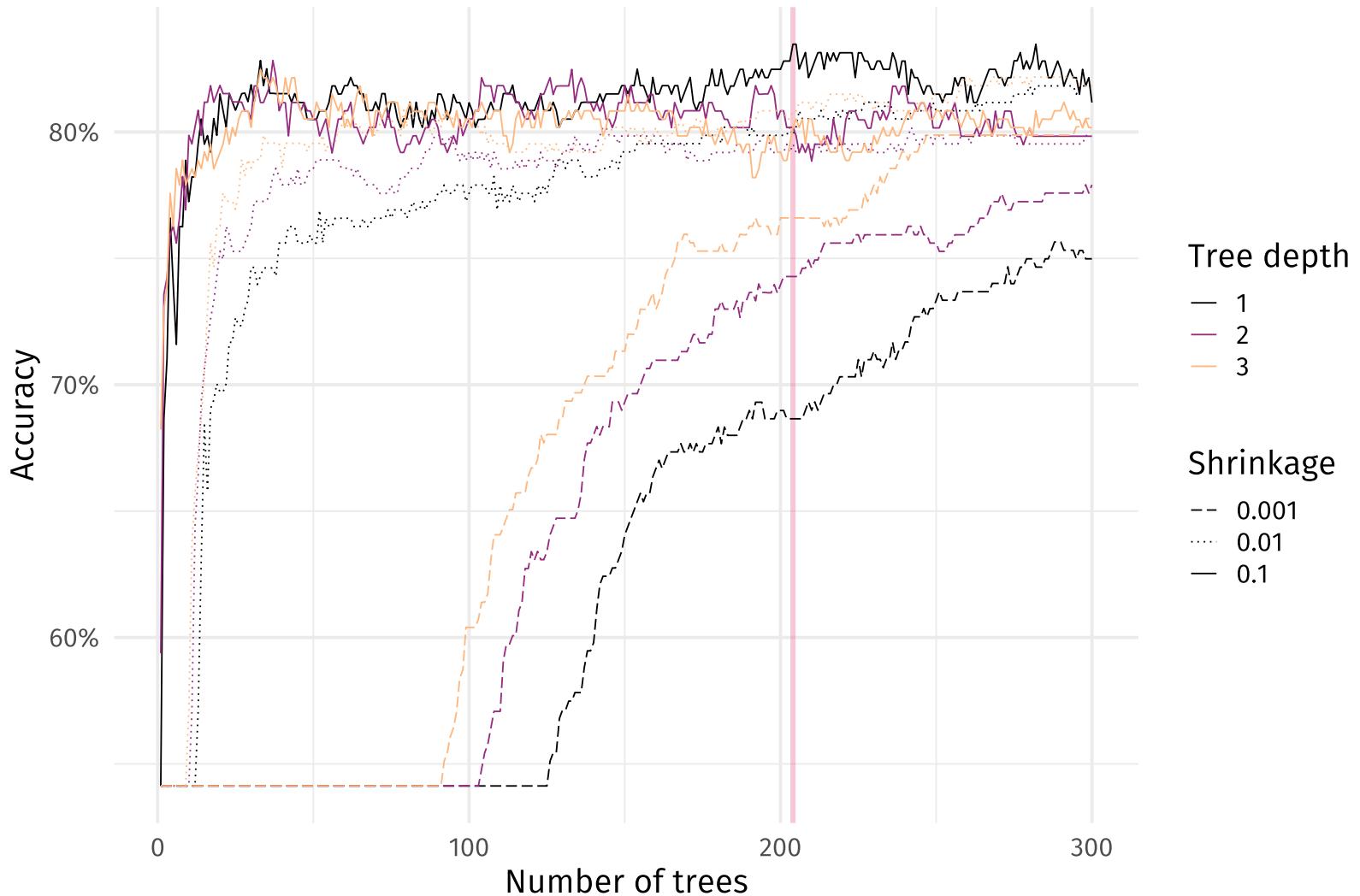
**B.** Update the model  $\hat{f}$  with "shrunken version" of new treee  $\hat{f}^b$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

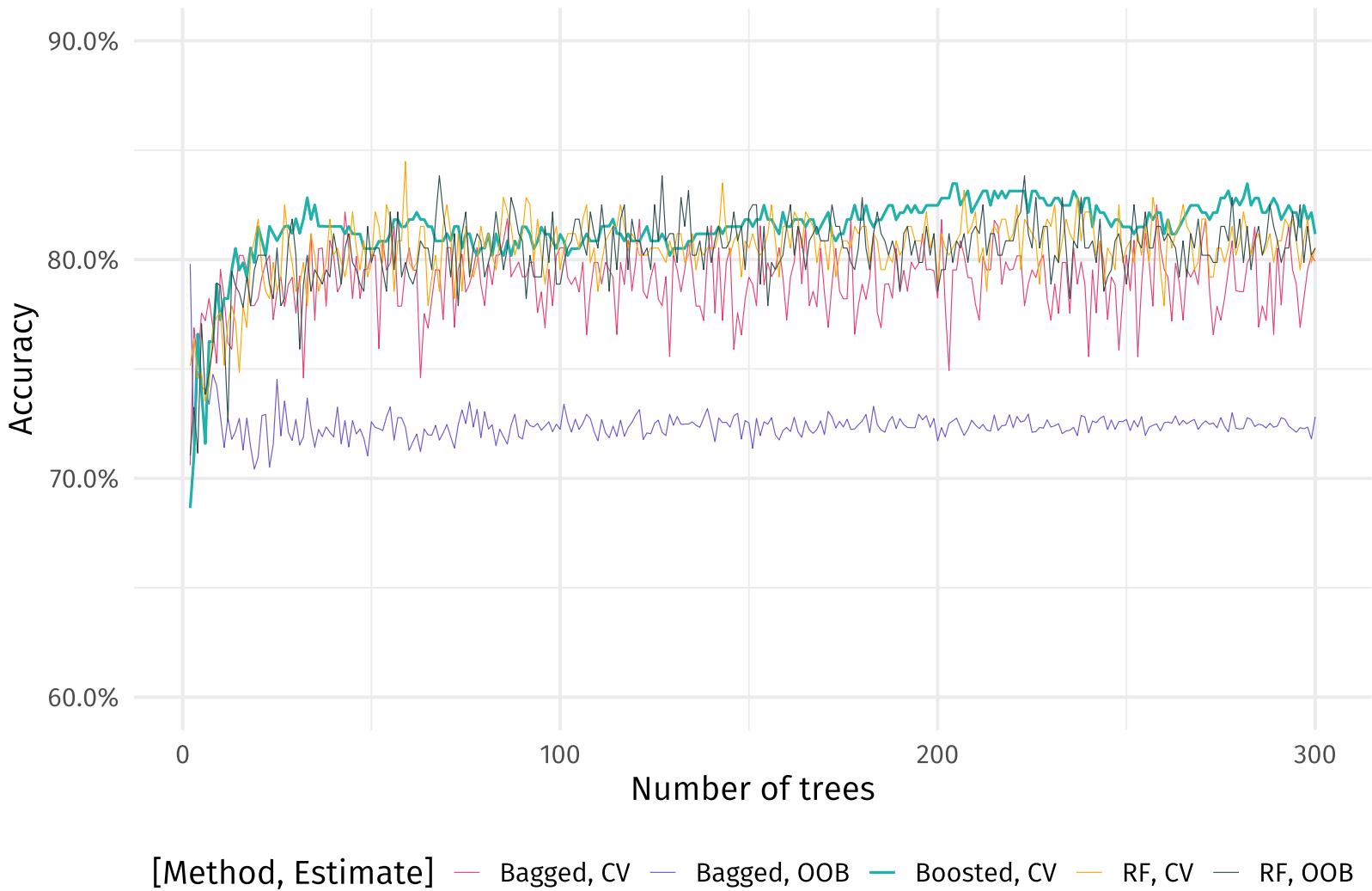
**C.** Update the residuals:  $r_i \leftarrow r_i - \lambda \hat{f}^b(x)$ .

**Step 3:** Output the boosted model:  $\hat{f}(x) = \sum_b \lambda \hat{f}^b(x)$ .

## Comparing boosting parameters—notice the rates of learning



## Tree ensembles and the number of trees



# Sources

Sources (articles) of images

- Deep learning and radiology
- Parking lot detection
- *New Yorker* writing
- Gender Shades

Tree-classification boundary examples come from [ISL](#).

I pulled the comic from [Twitter](#).