Lecture 11: Machine Learning

A Primer on Machine Learning

James Sears*
AFRE 891 SS24
Michigan State University

*Parts of these slides are adapted from **"Prediction and Machine-Learning in Econometrics"** by Ed Rubin, used under **CC BY-NC-SA 4.0**.

Table of Contents

Intro to Machine Learning

Prologue

Packages we'll use today:

```
#if (!require("DT")) remotes::install_github("rstudio/DT")

pacman::p_load(broom, data.table, furrr, future, ISLR, parallel, tidyverse,
       tibble)
```

What is Machine Learning?

Machine Learning uses algorithms that **learn** based on the data they're given

So far in your econometric training you've focused largely on well-behaved estimators with desirable properties and causal identification

We've expanded this some in our class, but machine learning can help us tackle an additional set of new tasks

Applications of Machine Learning

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 though sentences (pixels) to glean insights from relationships
- *E.g.* detect sentiments in tweets or roof-top solar and crop type in satellite imagery

Applications of Machine Learning

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

Unsupervised learning

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

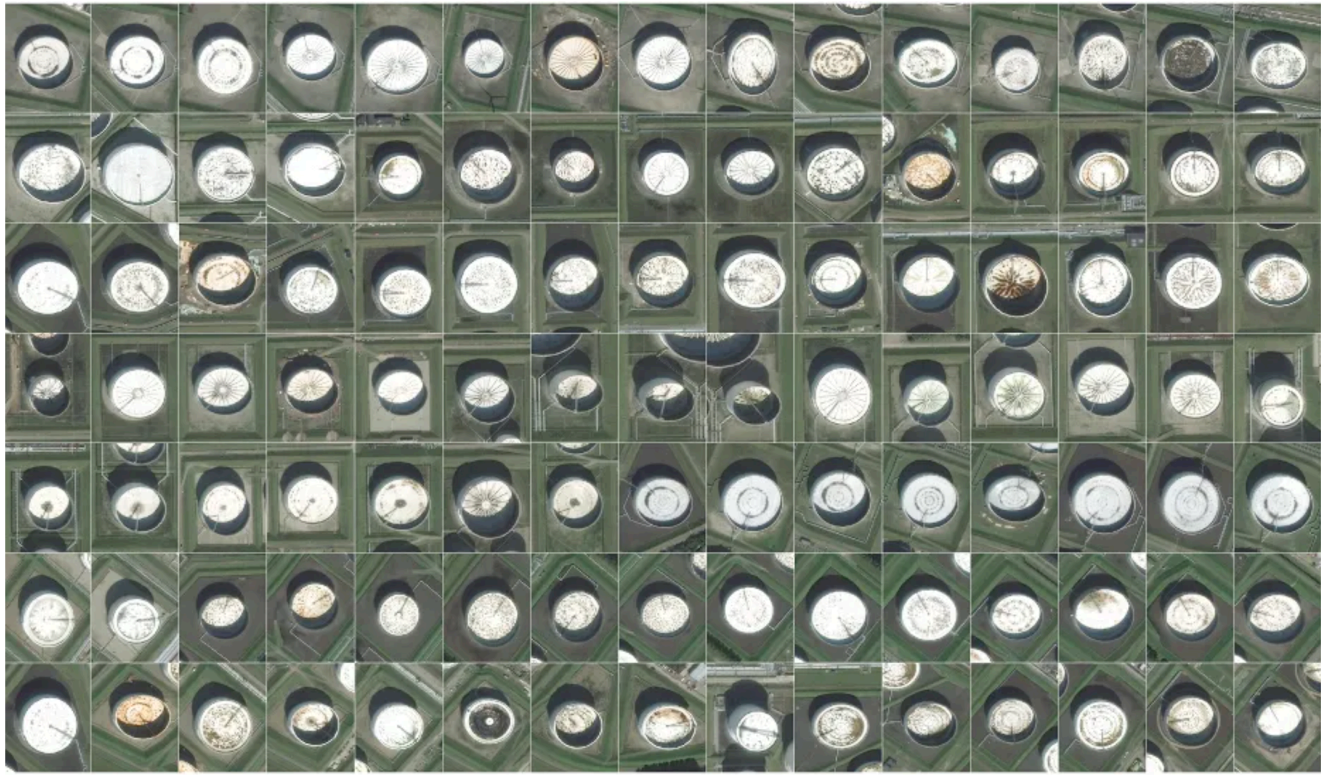
Treatment Effect Heterogeneity

- You want to go beyond an average causal effect
- *E.g.* estimate conditional average treatment effects, perform simulations

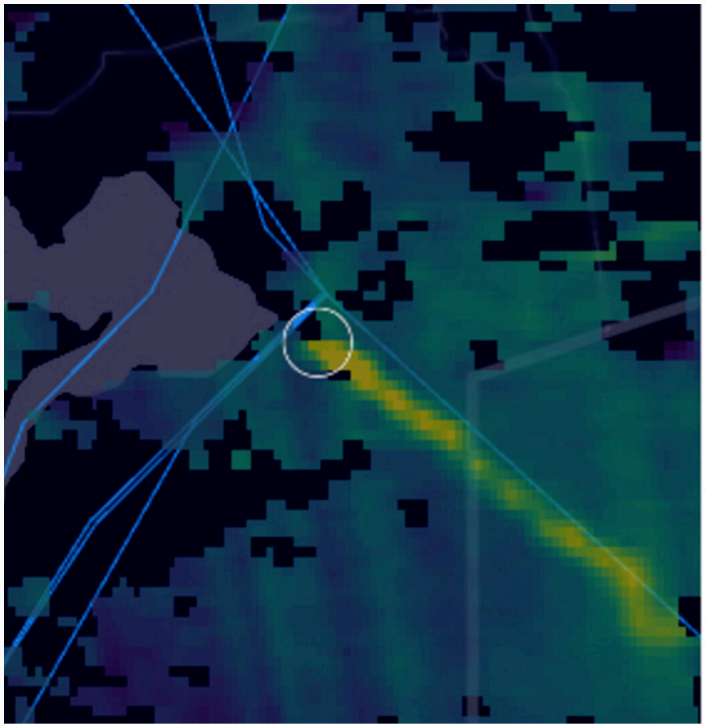
Stanford University (Stanford, CA) researchers have developed a deep-learning algorithm that can evaluate chest X-ray images for signs of disease at a level exceeding practicing radiologists.

Parking Lot Vehicle Detection Using Deep Learning

How AI Can Calculate Our Oil Surplus...From Space



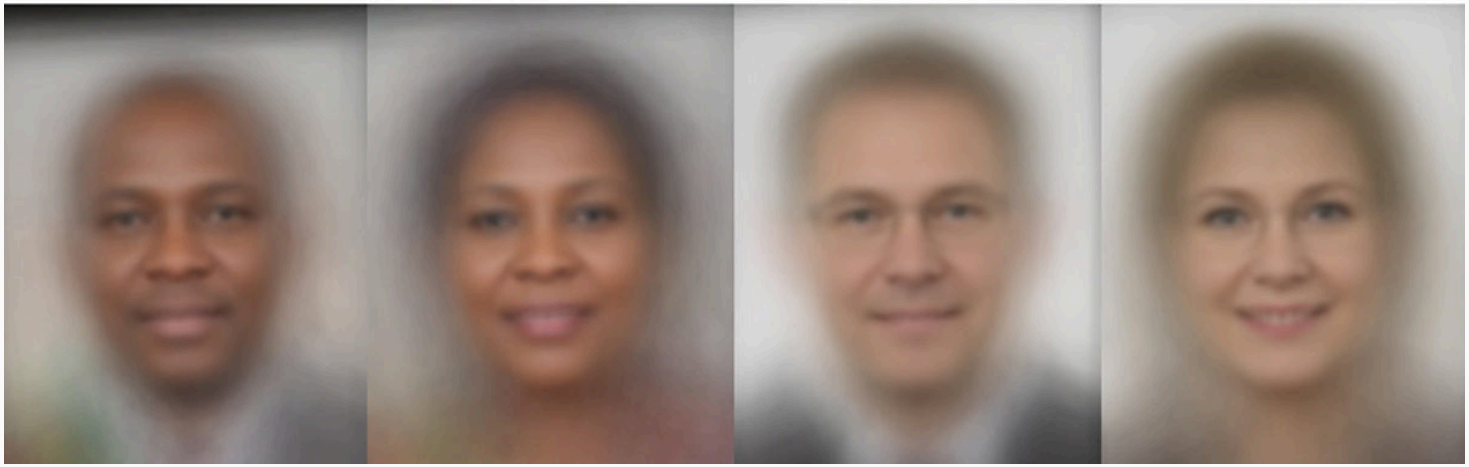
ORBITAL INSIGHT/DIGITALGLOBE



Monitoring methane emissions from gas pipelines


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%



And of course...

OpenAI and **ChatGPT**

Takeaways?

Any main takeaways/thoughts from these examples?

- Interactions and **nonlinearities** likely matter
- Features/variables can be important
  - We might not even know *which* are the features that matter
- Flexibility is huge, but we still want to avoid **overfitting**

Sources

Sources (articles) of images

- **Deep learning and radiology**
- **Parking lot detection**
- ***New Yorker* writing**
- **Oil surplus**
- **Methane leaks**
- **Gender Shades**

Types of ML Algorithms

We tend to break machine learning into two(ish) classes:

1. **Supervised learning** builds ("learns") a statistical model for predicting an **output** ($\mathbf{y}$) given a set of **inputs** ($\mathbf{x}_1, \ldots, \mathbf{x}_p$)

   i.e., we want to build a model/function f

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

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

2. **Unsupervised learning** learns relationships and structure using only **inputs** ($x_1, \ldots, x_p$) 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.

Output

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

1. **Classification tasks** for which the values of $\mathbf{y}$ are **discrete categories**
   E.g., race, sex, loan default, hazard, disease, flight status

2. **Regression tasks** in which $\mathbf{y}$ takes on **continuous, numeric values**.
   E.g., price, arrival time, number of emails, temperature

1 The use of *regression* differs from our use of *linear regression*.

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

† .[Q:] Where would you put responses to 5-item Likert scales?

Why *Learning*?

Q: What puts the "learning" in statistical/machine learning?

A: Most learning models/algorithms will **tune model parameters** based upon the observed dataset—learning from the data.

Primer on Machine Learning

This lecture we're going to do an overview of how to use machine learning for several types of tasks:

- **Model Selection and Regularization:** how do we choose what to throw on the RHS of a regression when economic theory doesn't tell us what the right controls or interactions are?
- **Classification:** Are there distinct groups within our highly multi-dimensional data?
- **Regression Trees and Forests:** Are there differences in conditional means (treatment effects) across the covariate space? What variable are most important for informing these differences?
- **Deep Learning:** Can we predict attributes of a document or class of images?

But first...

Terminology

Terminology

I'm following the notation of **ISL (great, free textbook)**

Data

n gives the number of observations

p represents the number of variables available for predictions

$\mathbf{X}$ is our $n \times p$ matrix of predictors

- Also known as **features**, inputs, independent/explanatory variables, ...
- $x_{i,j}$ is observation i (in $1, \ldots, n$) on variable j (for j in $1, \ldots, p$)

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix}$$

Dimensions of $\mathbf{X}$

Observation i is a p-length vector

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,p} \end{bmatrix}$$

Variable j is an n-length vector

$$\mathbf{x}_j = \begin{bmatrix} x_{1,j} \\ x_{2,j} \\ \vdots \\ x_{n,j} \end{bmatrix}$$

Dimensions of $\mathbf{X}$

Observation i is a p-length vector

Variable j is an n-length vector

Applied to R:

- `dim(x_df)` = n p
- `nrow(x_df)` $= n$; `ncol(x_df)` $= p$
- `x_df[1,]` $(i = 1)$; `x_df[,1]` $(j = 1)$

Outcomes

In supervised settings, we will denote our **outcome variable** as $\mathbf{y}$.

- **Synonyms:** output, outcome, dependent/response variable, ...

The outcome for our i^{th} observation is y_i. Together the n observations form

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

and our full dataset is composed of $\left\{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \right\}$

MSE

Mean squared error (MSE) is the most common[†] way to **measure model performance** in a regression setting.

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

Where $y_i - \hat{f}\left(x_i\right) = y_i - \hat{y}_i$ 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).

† *Most common* does not mean best—it just means lots of people use it.

MSE

Mean squared error (MSE) is the most common[†] way to **measure model performance** in a regression setting.

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

One potential alternative: **mean absolute error (MAE)**

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

† *Most common* does not mean best—it just means lots of people use it.

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.†

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).

† Recall R-squared weakly increasing in p.

Resampling

Resampling

Before we dive into our machine learning algorithms, we should take a moment to discuss **resampling**.

Resampling methods help us understand uncertainty in statistical modeling

- **Linear Regression:** How precise are your $\hat{\beta}_1$?
- **K-Means Clustering:** What choice of K minimizes out-of-sample error?

The Process Behind Resampling

Resampling methods largely follow the below steps:

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

Warning 1: resampling methods can be computationally intensive

Warning 2: certain methods only work in certain settings

Resampling Methods

We're going to focus on two common **resampling methods:**

1. **Cross validation** used to **estimate test error**, evaluating performance or selecting a model's flexibility

2. **Bootstrap** used to **assess accuracy**—parameter estimates or methods

Cross-Validation and Hold-out Methods

Hold-out methods like **Cross-validation** use the training data itself to estimate test performance

- **Holds out** a mini "test" sample of the training data that we use to estimate the test error.

Two approaches we'll see:

- **Leave-one-out cross-validation (LOOCV):** leave out one observation, train the model, estimate error, repeat over all observations
- **k-fold cross-validation:** split data into k groups (folds), leave out one fold and train the model, estimate error, repeat over all folds

Leave-one-out Cross Validation

Leave-one-out cross-validation (LOOCV) maximizes the available training data while still maintaining separation between training and validation subsets

1. **Reduces bias** relative to validation set methods by using n-1 (almost all) observations for training.
   - **Validation set:** reserve a subset (30%)
2. **Resolves variance**: it makes all possible comparisons (no dependence upon which validation-test split you make).

Leave-one-out Cross Validation

Steps:

1. **Leave out** one observation i
2. **Train** the model on the $n - 1$ observations
3. **Calculate** MSE_i
4. **Take Mean**

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

Leave-one-out Cross Validation

Observation 1's turn for validation produces MSE_1.

Leave-one-out Cross Validation

Observation 2's turn for validation produces MSE_2.

Leave-one-out Cross Validation

Observation 3's turn for validation produces MSE_3.

Leave-one-out Cross Validation

Observation 4's turn for validation produces MSE_4.

Leave-one-out Cross Validation

Observation 5's turn for validation produces MSE_5.

Leave-one-out Cross Validation

Observation n's turn for validation produces MSE_n.

k-fold Cross Validation

k-fold cross-validation (LOOCV) is less computationally demanding and has (generally) greater accuracy

- Somewhat **higher bias:** $n - 1$ vs. $(k - 1)/k$
- **Lower variance:** high degree of correlation in LOOCV MSE

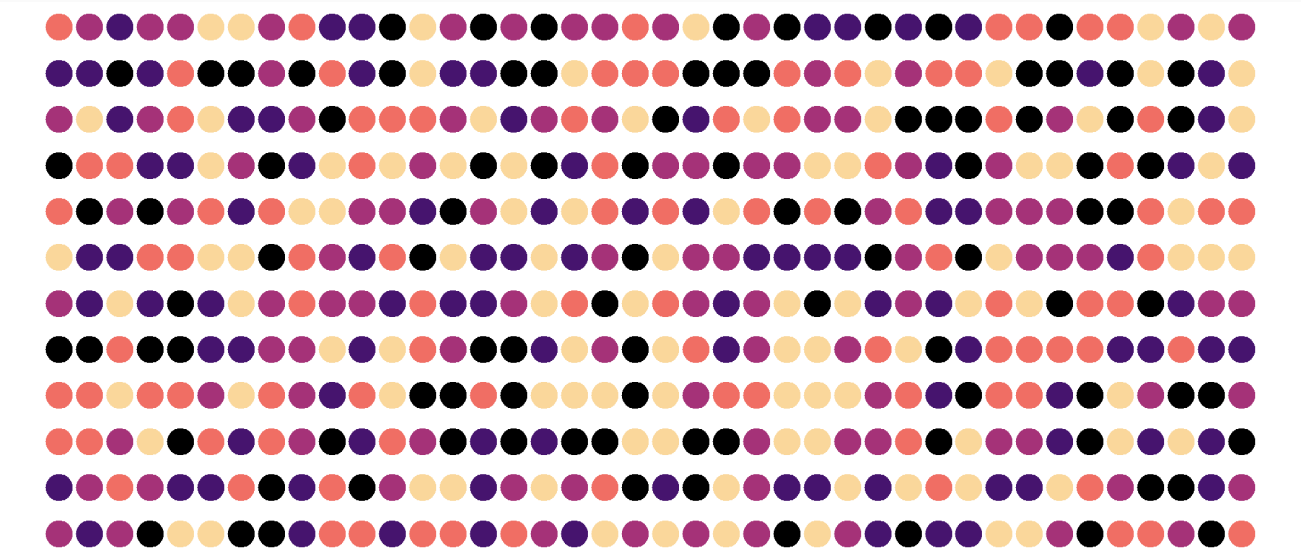
Steps:

1. **Divide** the training data into k equally sized groups (folds).
2. **Train** the model on the other $k - 1$ folds
3. **Repeat** for all folds
4. **Average** the folds' MSEs to estimate test MSE

k-fold Cross Validation

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

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

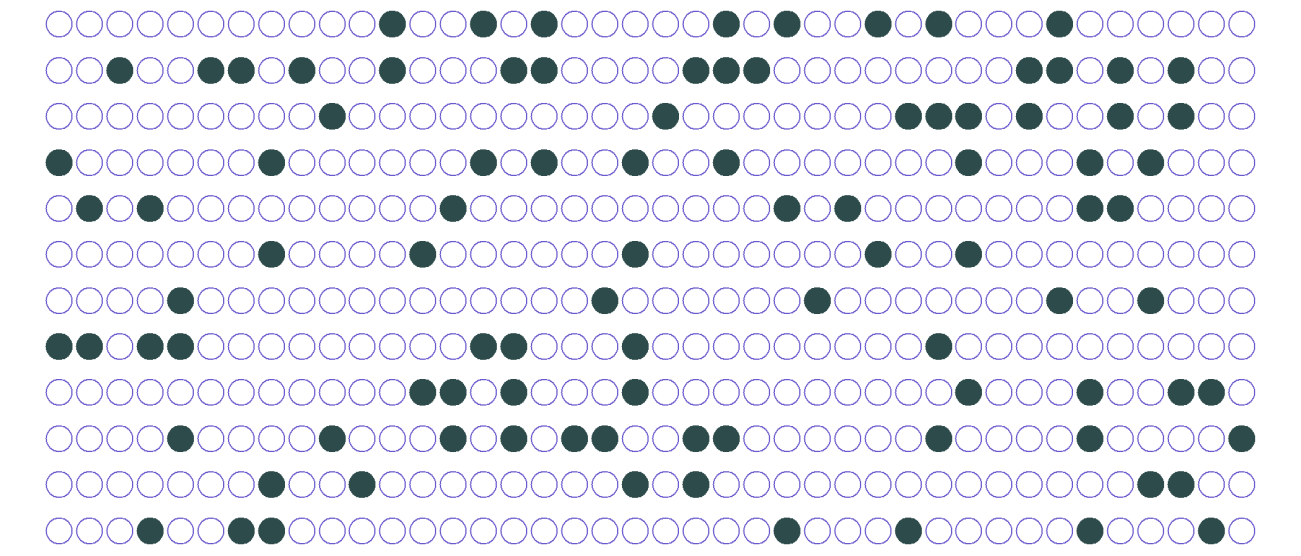


Our $k = 5$ folds.

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**.

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 $\text{MSE}_{k=1}$.

k-fold Cross Validation

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

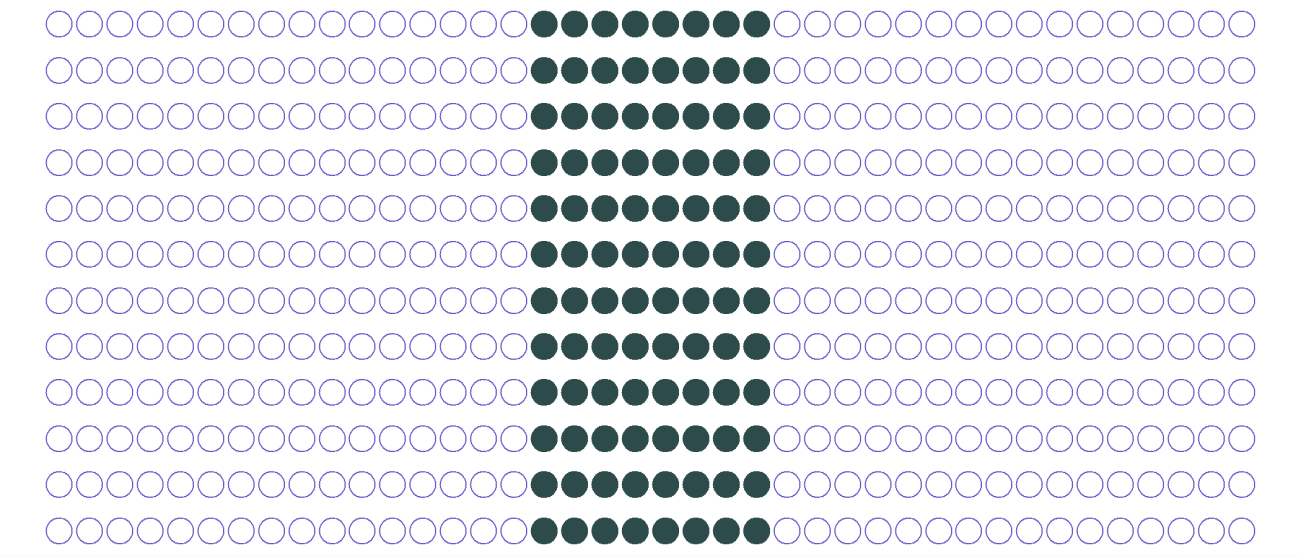
$$\text{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \text{MSE}_i$$

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

k-fold Cross Validation

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

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

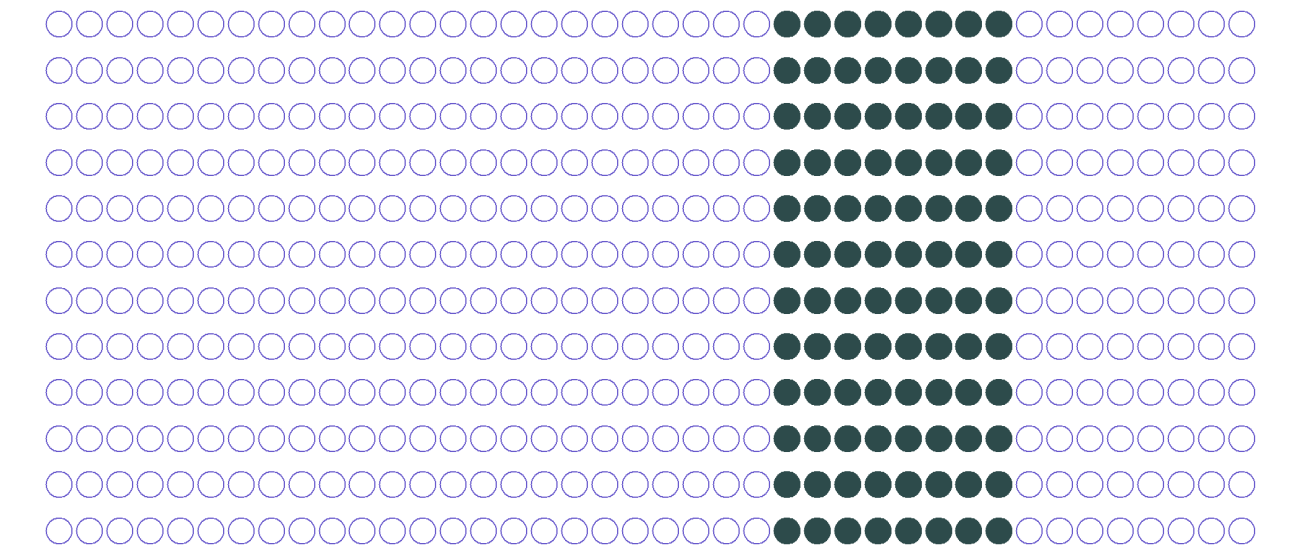


For $k = 5$, fold number 3 as the **validation set** produces $\text{MSE}_{k=3}$.

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 $\text{MSE}_{\text{k=4}}$.

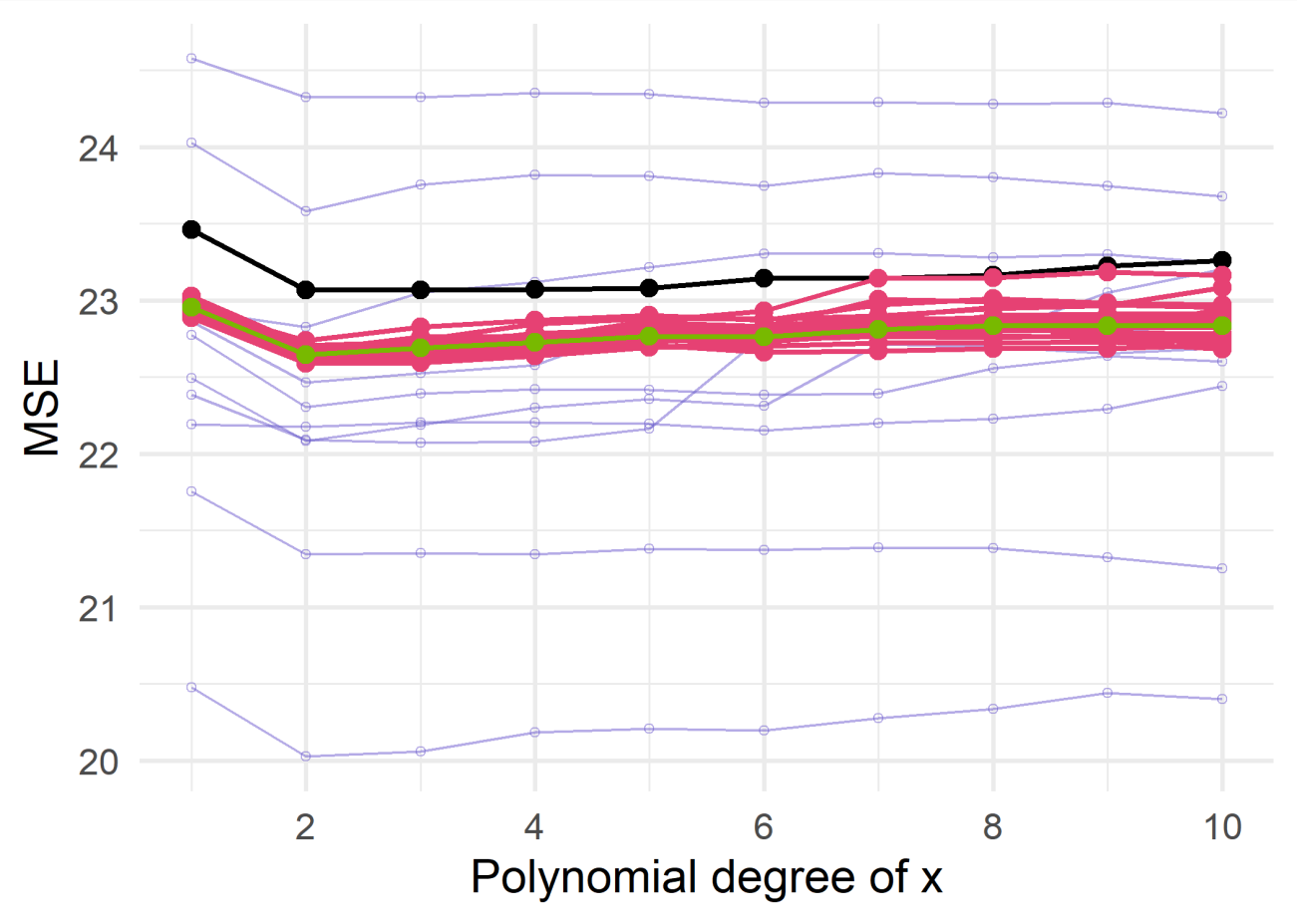
k-fold Cross Validation

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 $\text{MSE}_{\text{k=5}}$.

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

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

Caveat

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

The methods that we've defined assume this independence.

Make sure that you think about

- the structure of your data
- the goal of the prediction exercise

E.g.,

1. Are you trying to predict the behavior of existing or new customers?
2. Are you trying to predict historical or future recessions?

The Bootstrap

The Bootstrap

The **Bootstrap** is a resampling method often used to quantify the **uncertainty (variability) underlying an estimator** or learning method.

Hold-out methods

- Randomly divide the sample into training and validation subsets
- Train and validate ("test") model on **each subset/division**

Bootstrapping

- Randomly samples **with replacement** from the original sample
- Estimates model on each of the *bootstrap samples*

Why Bootstrap?

As we've seen, estimating an estimator's standard error involves assumptions and theory.

Sometimes this is straightforward to derive (e.g. OLS)

However, there are times this derivation is difficult or even impossible, *e.g.*,

$$\mathbf{Var}\left(\frac{\hat{\beta}_1}{1-\hat{\beta}_2}\right)$$

The bootstrap can help in these situations.

Rather than **deriving an estimator's variance**, we use bootstrapped samples to **build a distribution** and then learn about the estimator's variance.

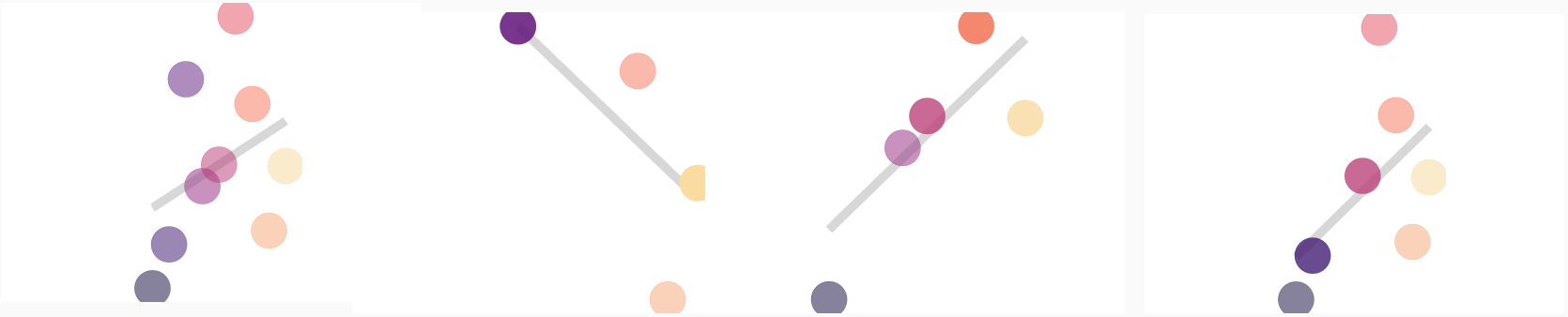
Intuition

Idea: Bootstrapping builds a distribution for the estimate using the variability embedded in the training sample.

Graphically

Z | $Z^{\star 1}$ | $Z^{\star 2}$ | ... | $Z^{\star B}$

$\hat{\beta} = 0.653$ | $\hat{\beta} = -0.96$ | $\hat{\beta} = 0.968$ | $\hat{\beta} = 0.978$



The Bootstrap

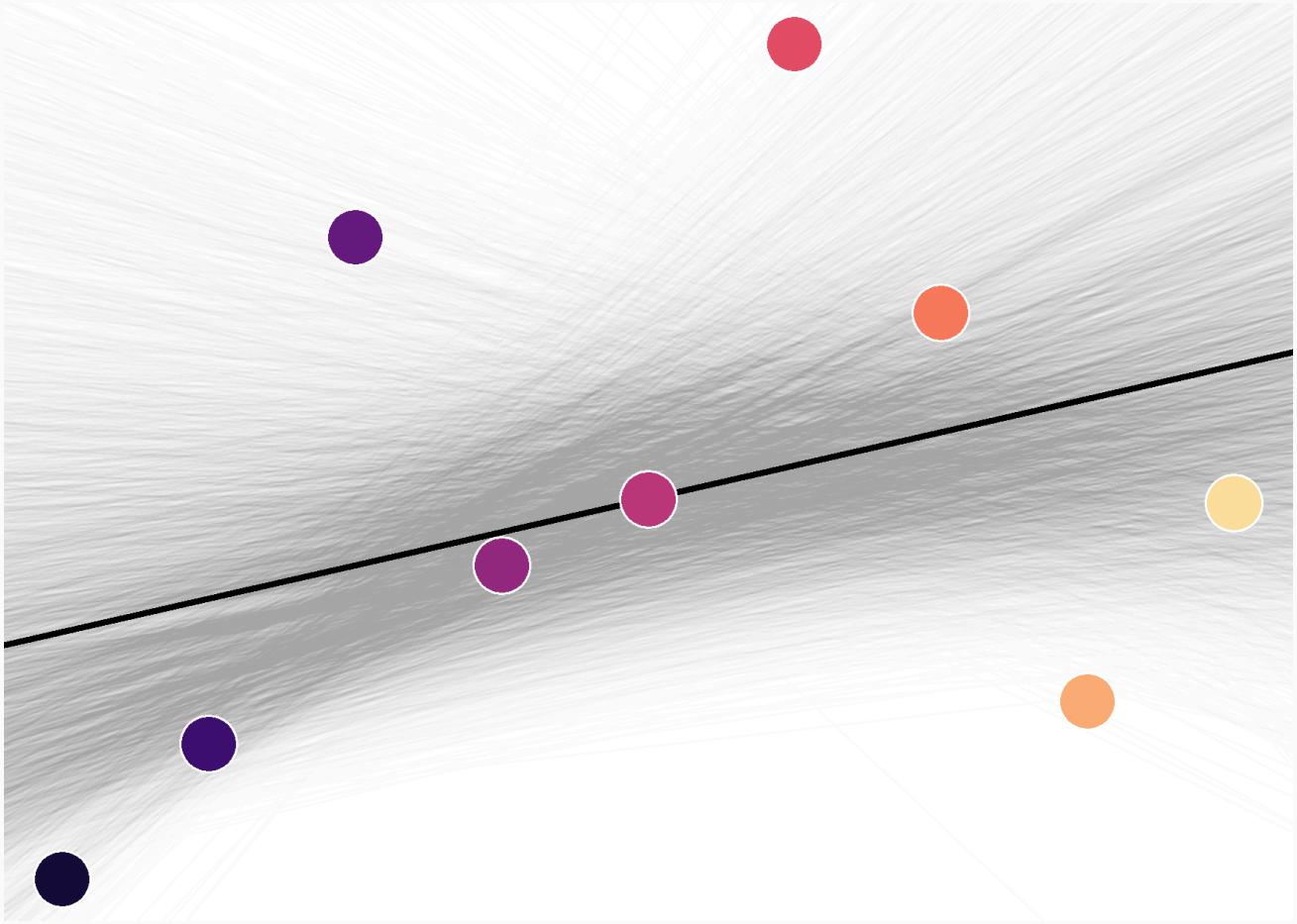
First, let's write a function to obtain one bootstrap estimate:

```
 # Write a bootstrap function
boot_est ← function(n) {
    # Estimates via bootstrap
    est ← lm(y ~ x, data = z[sample(1:n, n, replace = T), ])
    # Return a tibble
    data.frame(int = est$coefficients[1], coef = est$coefficients[2])
}

# Calculate a "safe" number of cores (allow for background processes)
n_cores = future::availableCores() - 2
```

The Bootstrap

Running this bootstrap 10,000 times in parallel

```
# Set the "plan"
plan(strategy = "multisession", # run in parallel in separate background R sessior
     workers = n_cores) # use the desired number of cores

# Set a seed
set.seed(123)
# Run the simulation 1e4 times
boot_df ← future_map_dfr(
  rep(n, 1e4),   # Repeat sample size 100 for 1e4 times
  boot_est, # our single bootstrap estimate function
  .options = furrr_options(seed = T)  # Let furrr know we want to set a seed
)
plan("sequential")
```



Comparison: Standard-error estimates

The bootstrapped standard error of $\hat{\alpha}$ is the standard deviation of the $\hat{\alpha}^{\star b}$

$$\mathrm{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B}\sum_{b=1}^{B}\left(\hat{\alpha}^{\star b} - \frac{1}{B}\sum_{\ell=1}^{B}\hat{\alpha}^{\star \ell}\right)^2}$$

This 10,000-sample bootstrap estimates $\mathrm{S.E.}\left(\hat{\beta}_1\right) \approx 0.77$.

If we go the old-fashioned OLS route, we estimate 0.673.

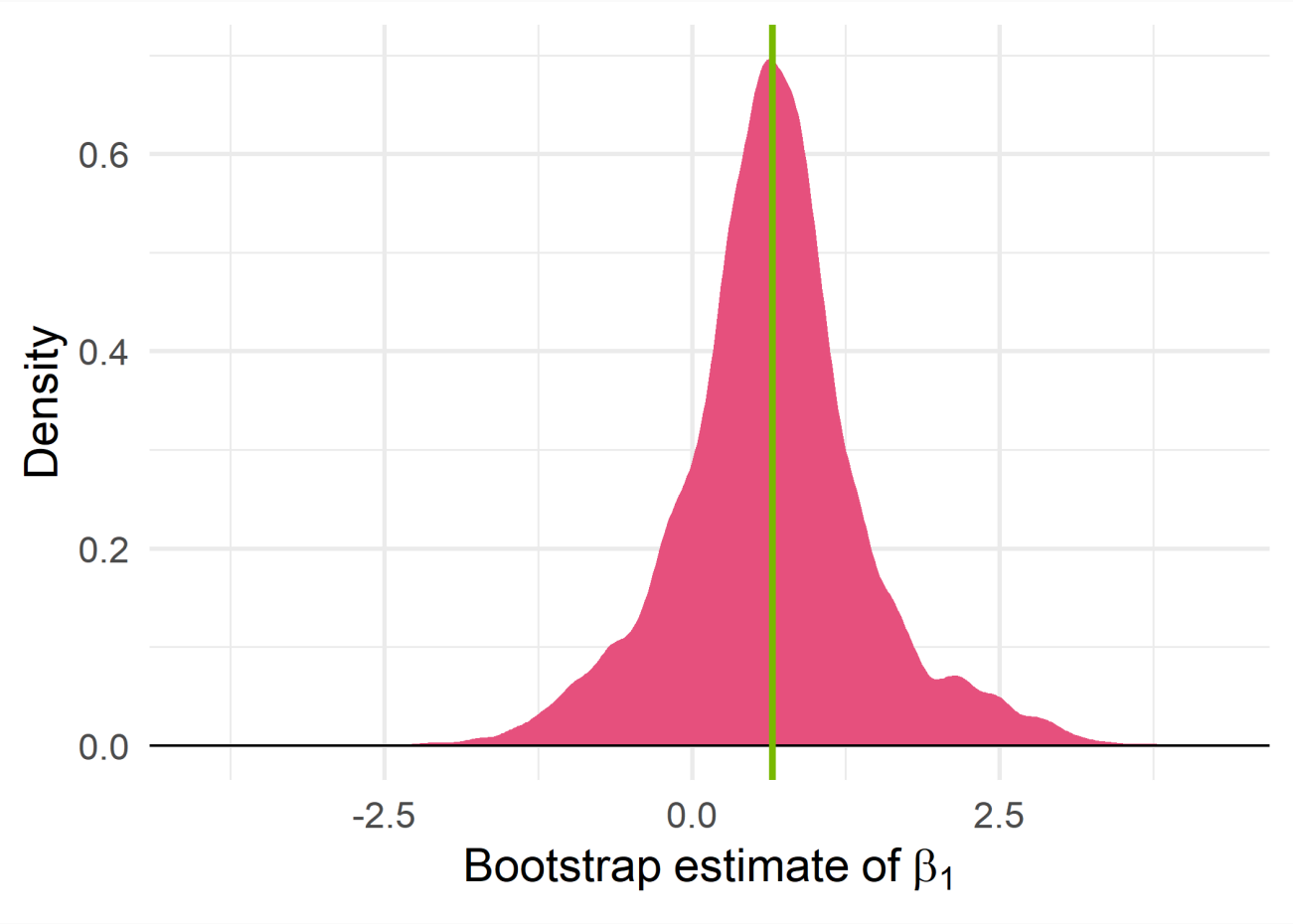

0.6
0.4
0.2
0.0
Density
-2.5
0.0
2.5
Bootstrap estimate of β_1

Table of Contents