BPS5231

Artificial Intelligence

For

Sustainable

Building

Design

L2

Ang Yu Qian

Assistant Professor

No class next week (27 Aug)

Study Trip (3 Sep)

https://www.youtube.com/watch?v=3-M8C23SqUk

Bus leaves 2pm sharp
(be there at 1.50pm)
@ SDE3 Foyer Drop-off
(returns at 4.45pm)

you late, your problem

Start thinking about your projects

(may have interim submissions)

Flexibility

L02.1

Supervised 1

Linear Regression

Brute Force

Exact Method (Analytical)

Gradient Descent

Multilinear

Polynomial

L02.2

Supervised 2

Model Evaluation

Bias-Variance Trade-off

Parametric vs Non-Parametric

KNN

L02.3

Beyond Linearity

L02.1

Supervised 1

Linear Regression

Brute Force

Exact Method (Analytical)

Gradient Descent

Multilinear

Polynomial

L02.2

Supervised 2

Model Evaluation

Bias-Variance Trade-off

Parametric vs Non-Parametric

KNN

L02.3

Beyond Linearity

Today, we will cover

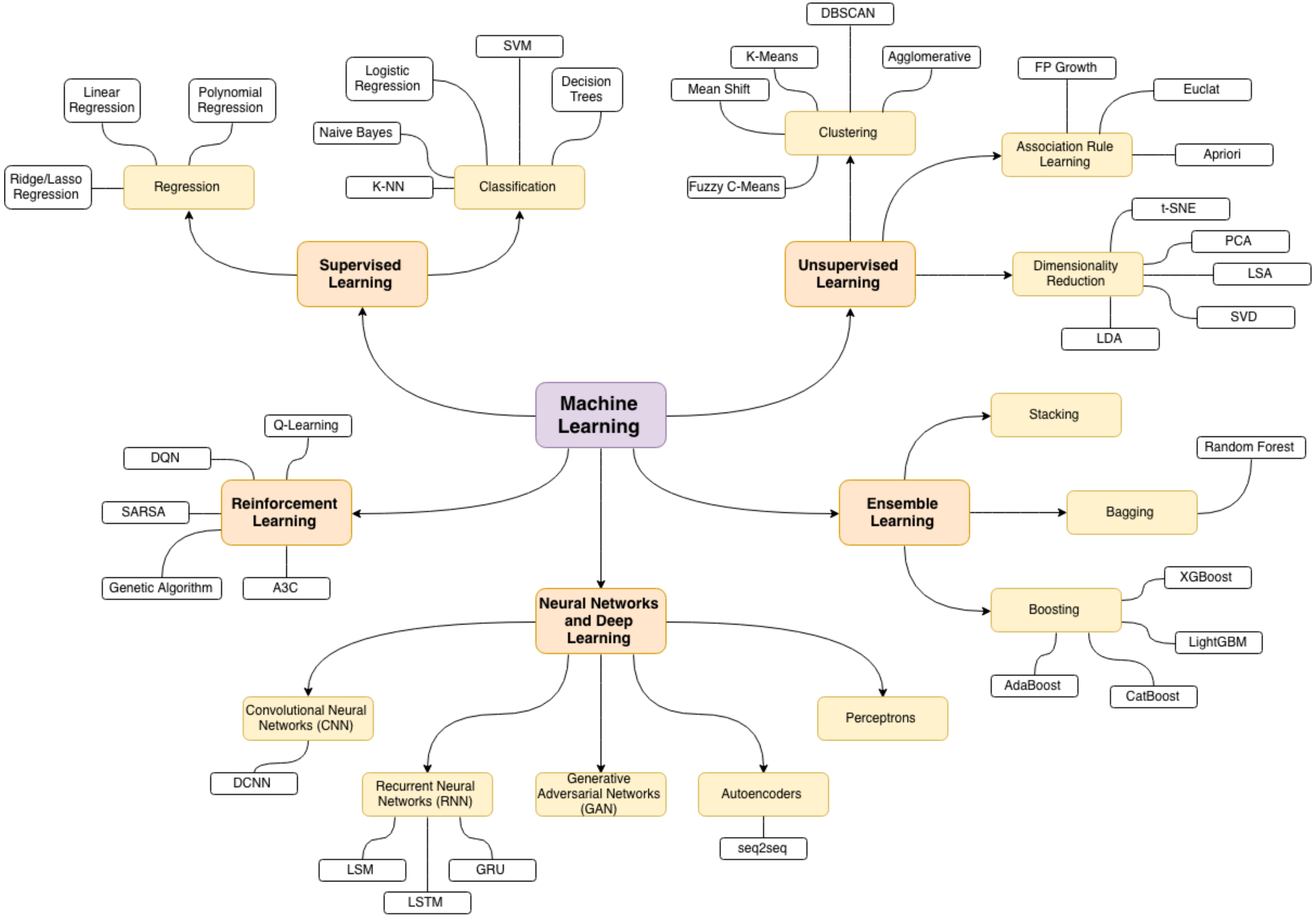


Image from Anil Jain, Google

Today, we will cover

Today →

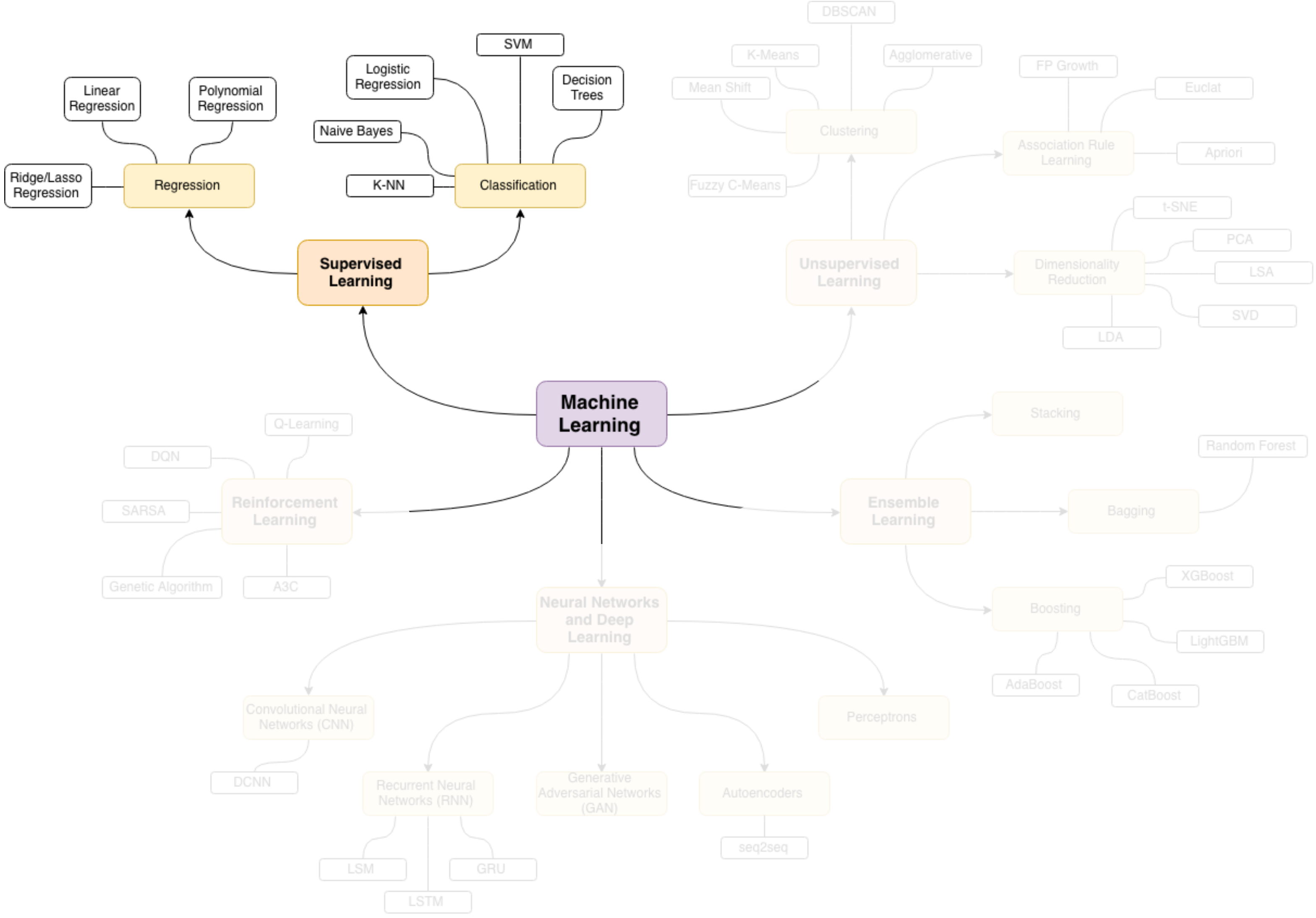


Image from Anil Jain, Google

$S_n = \{(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}), t = 1, \ldots, n\}$ = dataset with features and labels (ground truth examples)

$\mathcal{H}$ = hypothesis class (space of functions)

$\mathcal{L}$ = loss function (often squared error for regression) e.g. $\mathcal{L}(x_1, x_2) = (x_1 - x_2)^2$

Goal: pick a function $h \in \mathcal{H}$ that predicts labels y based on features x as accurately as possible

\>>> Optimization
ML algorithms always involve optimization (sometimes greedy). Formally:

observation

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \mathcal{L}(h(x_i), y_i)$$

prediction

Minimization over training data but we really want to do well over testing data!
(more on this next weeks)

Remember our simplest friend from previous class:

$$Y = f(X) = \beta_0 + \beta_1 X + \varepsilon$$

- Y: predicted output (e.g. building energy use)
- X: input/feature (e.g. outdoor temperature)
- β_0: the intercept (baseline value when $X = 0$)
- β_1: The slope/weight (how much Y changes per unit change in X)
- ε: The error term (the part not explained by the model)

we compute β_0 and β_1 using observations

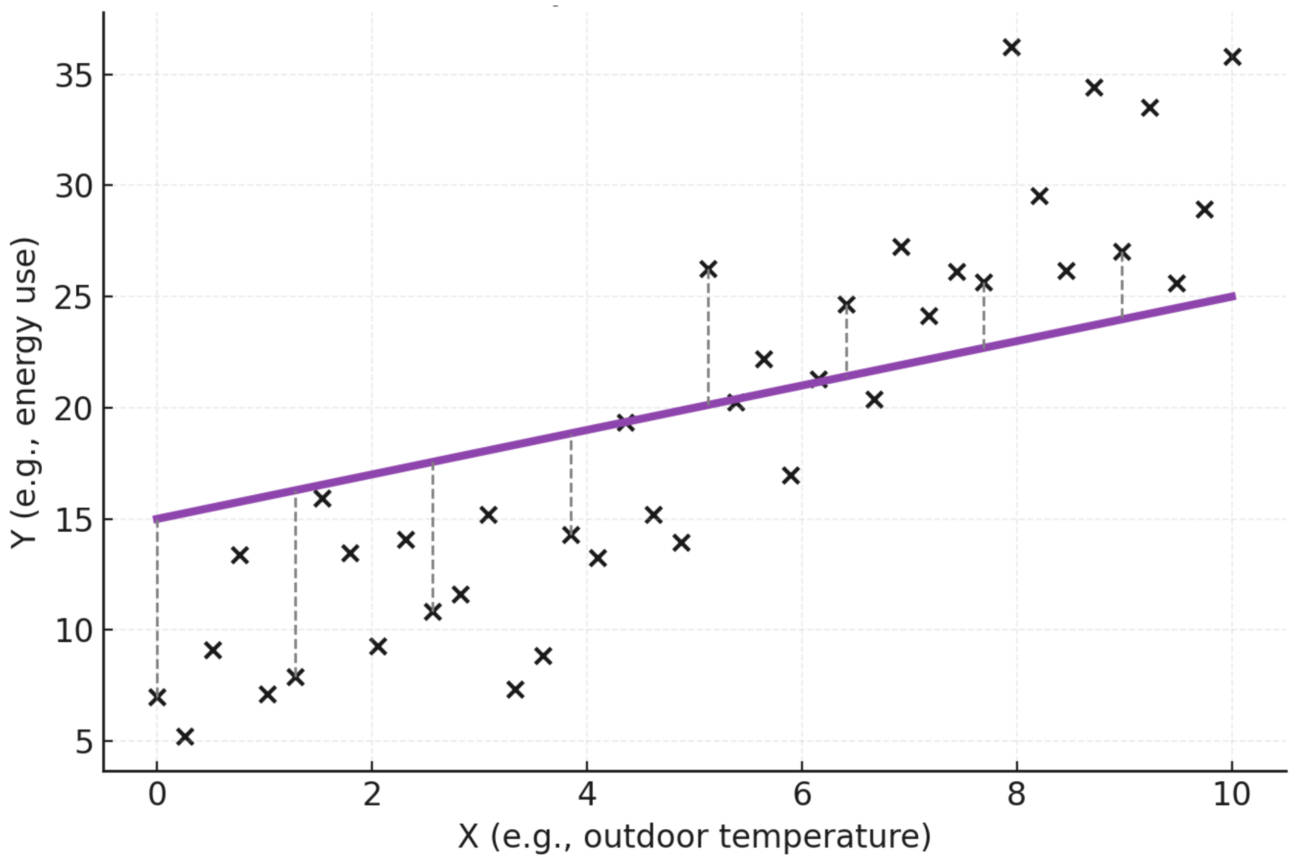
How do we estimate or find the regression coefficients?

1. Brute Force

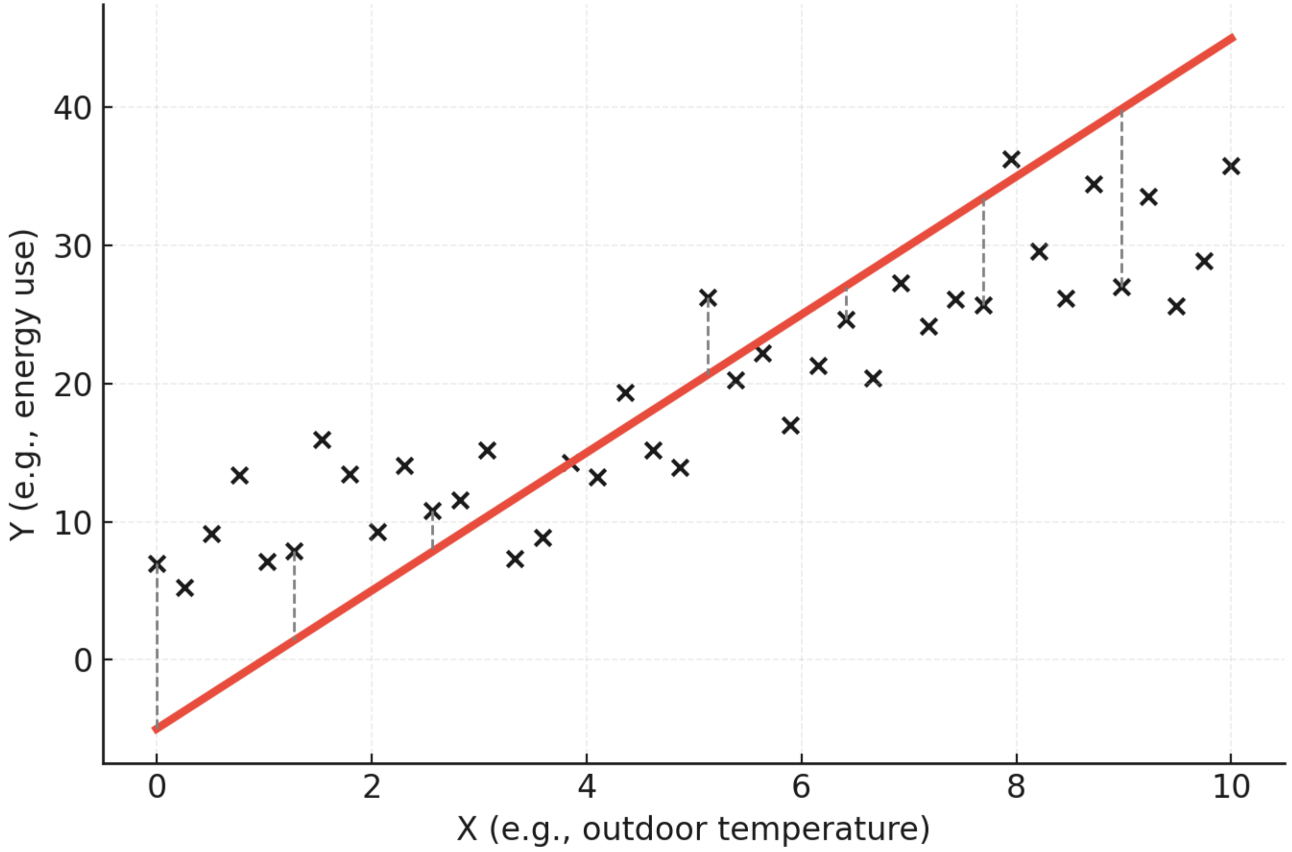
2. Exact Method (Analytical)

3. Gradient Descent (Optimization)

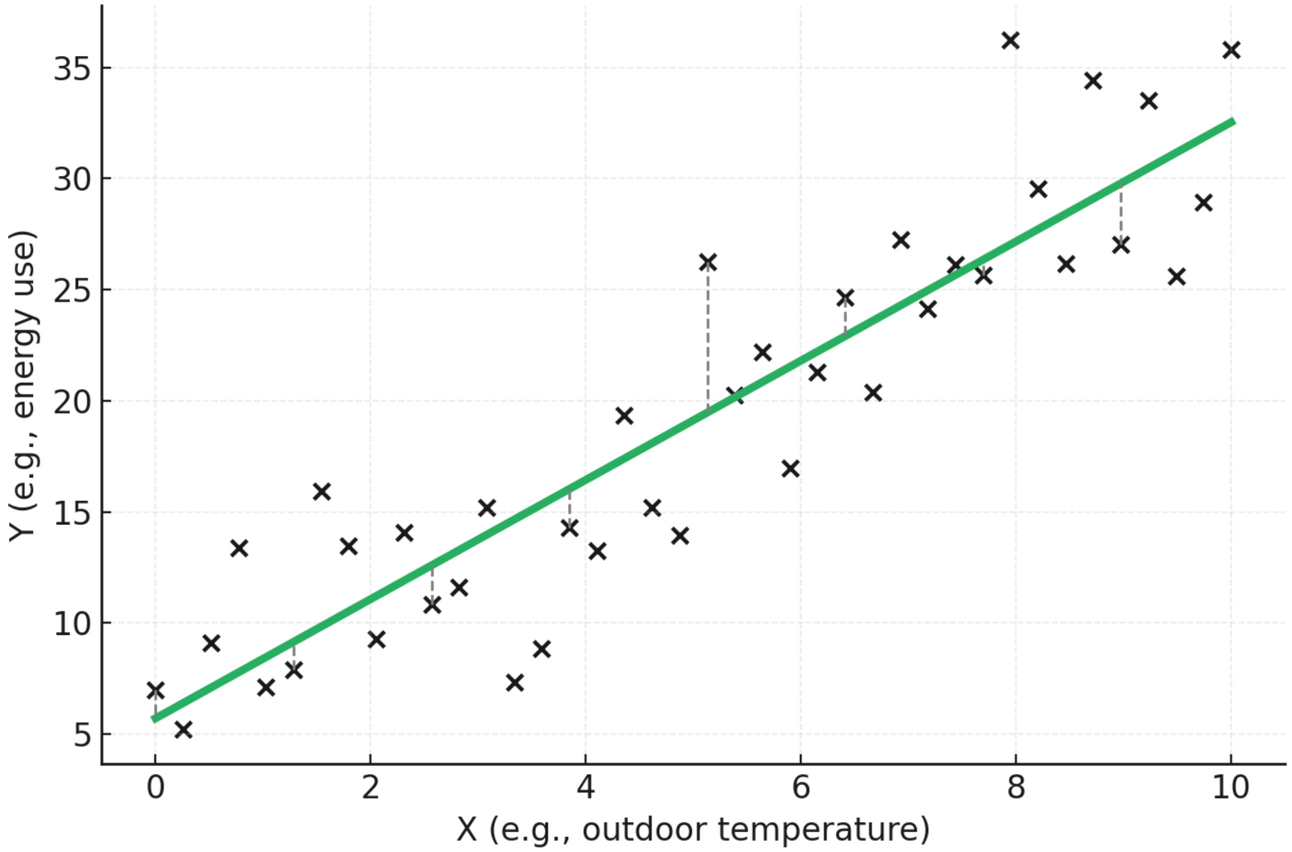
1 Brute Force
Candidate A: Y = 15 + X



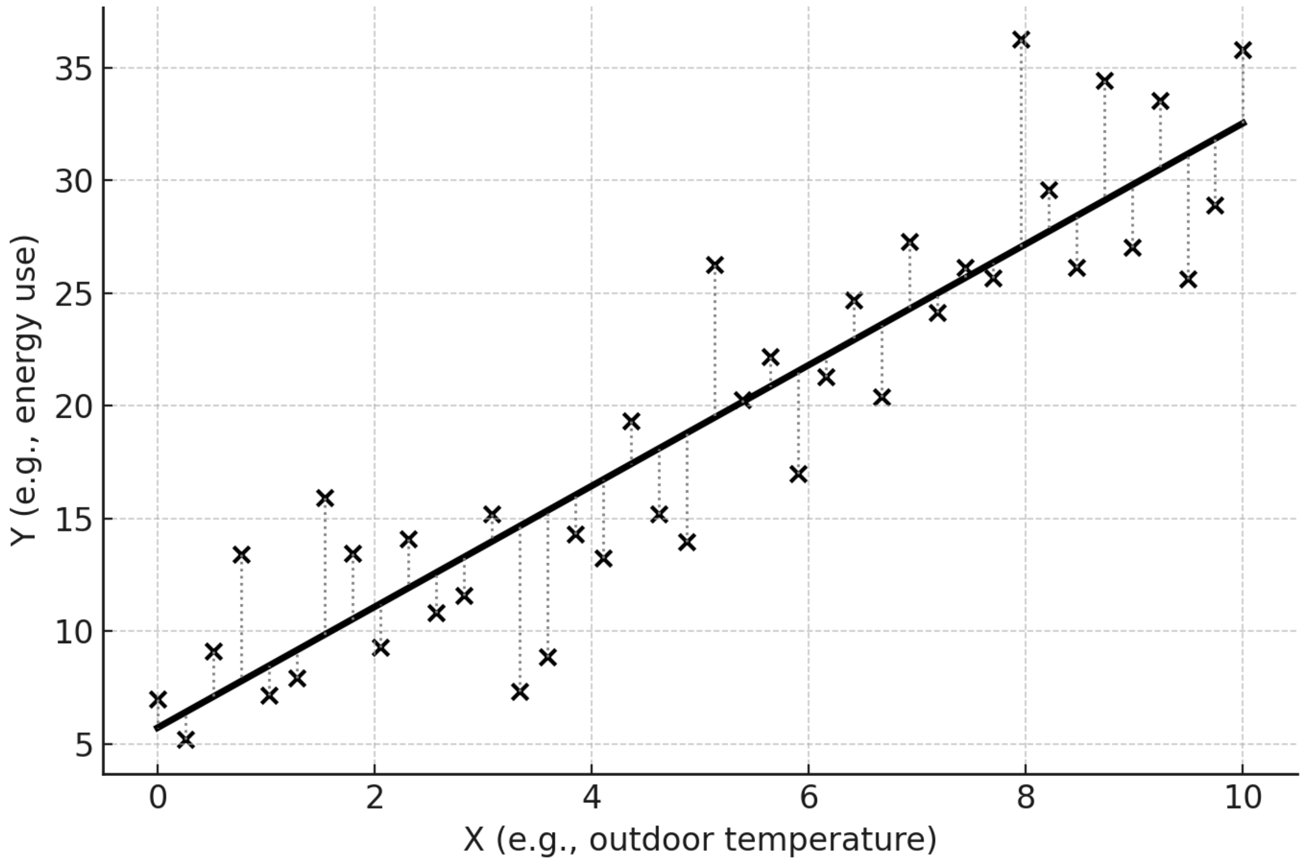
1 Brute Force
Candidate B: Y = -5 + 5X



1 Brute Force
Candidate C: Y = 5.72 + 2.68X



Question: Which line is the best?
First, calculate the residuals



Again, we use MSE as our loss function (our best friend from last week)

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2$$

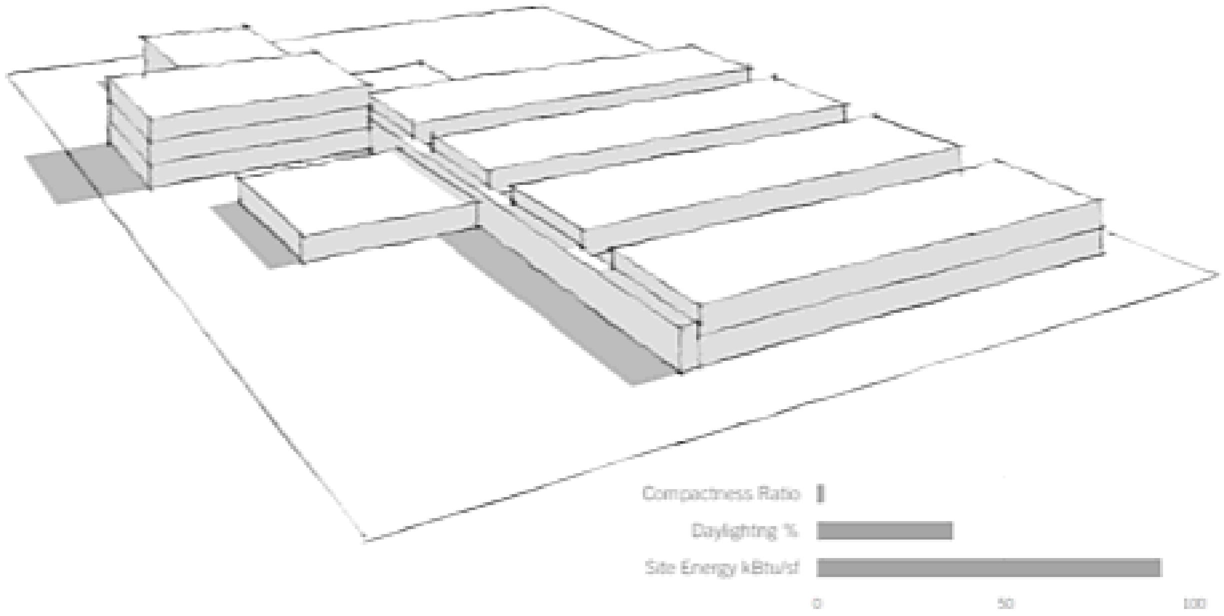
We choose β_0 and β_1 to minimize the predictive errors made by the model i.e., to minimize our loss function

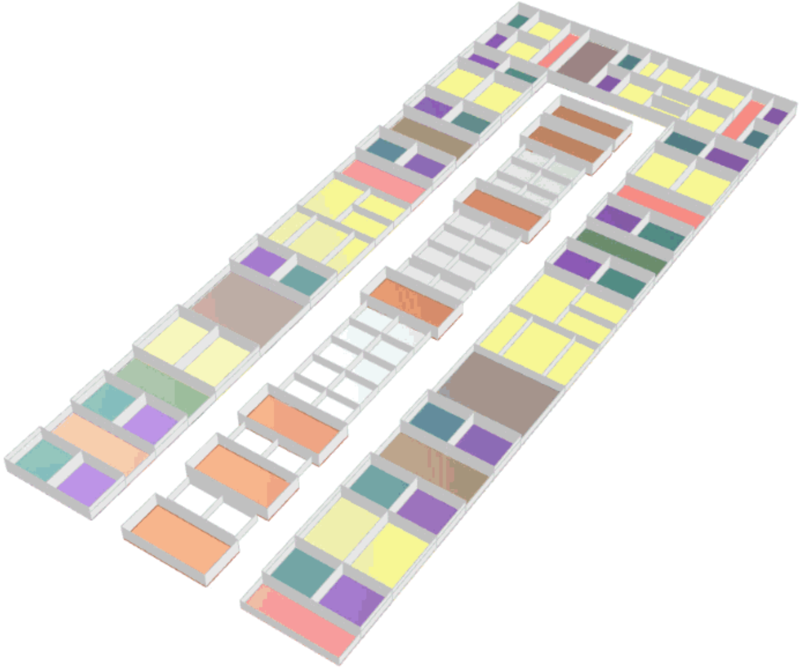
Then, optimal values should be:

$$(\hat{\beta}_0, \hat{\beta}_1) \; = \; \underset{\beta_0, \beta_1}{\arg\min} \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2$$

Again, we use MSE as our loss function (our best friend from last week)

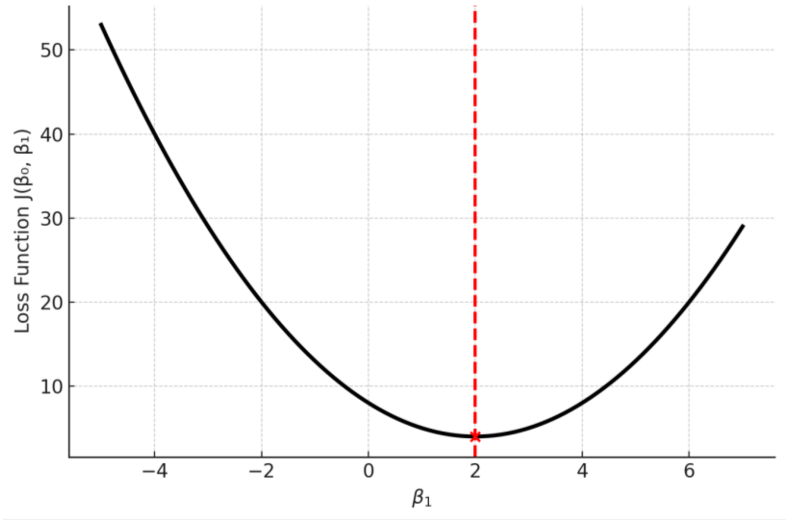
$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2$$



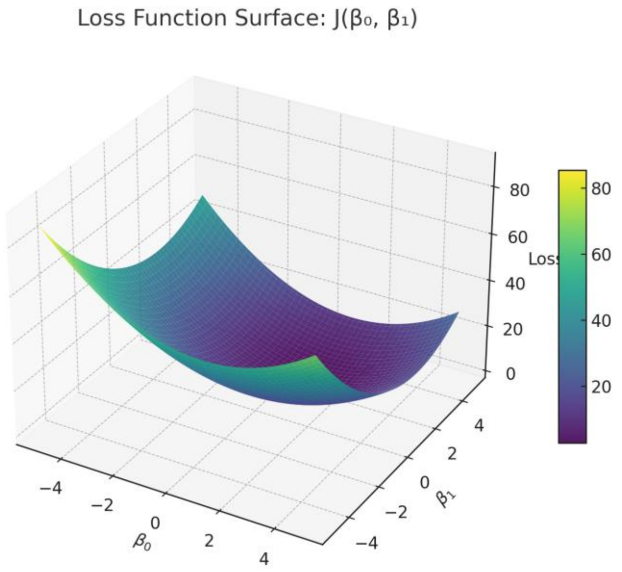


source: Rensselar Polytechnic Institute

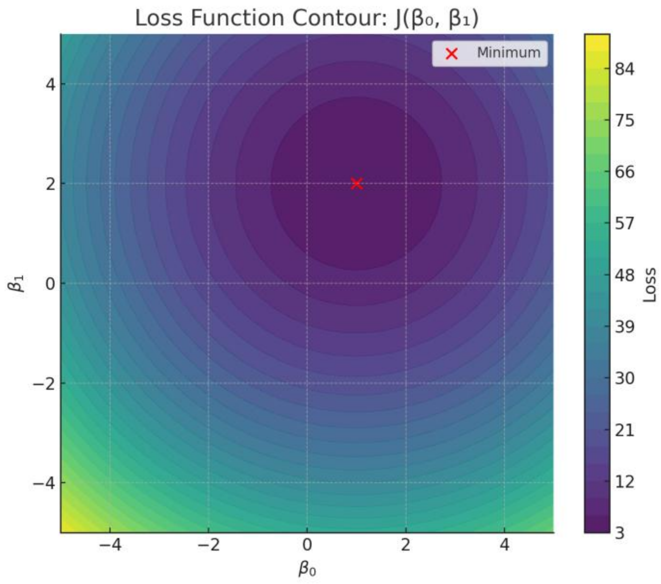
One way to estimate argmin β_0, β_1 is to calculate the loss function for every possible β_0 and β_1 , then find the β_0 and β_1 where loss function is minimum



1D



2D



2D

Google Colab

https://bit.ly/BPS5231-L2

2 Exact Method (Analytical)

Another way is to take the partial derivatives w.r.t β, set derivatives to zero, and solve for the coefficients (linear algebra)

We represent training data in terms of matrices:

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

- Each row of X is one training example
- First column corresponds to intercept (β_0)
- Y is column vector of target values

Matrix form of Linear Regression:

$$\hat{y} = X\beta$$

where

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

2 Exact Method (Analytical)
Another way is to take the partial derivatives w.r.t β, set derivatives to zero, and solve for the coefficients (linear algebra)

Close-Form Solution:
- Minimize Loss

$$L(\beta) = \frac{1}{2m}\|y - X\beta\|^2$$

- Expand loss, take partial derivatives w.r.t β, set derivatives to zero

$$X^T X\hat{\beta} = X^T y$$

- Solve for the coefficients

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

2 Exact Method (Analytical)

Another way is to take the partial derivatives w.r.t β, set derivatives to zero, and solve for the coefficients (linear algebra)

Explicit Formula
$$\hat{\beta}_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

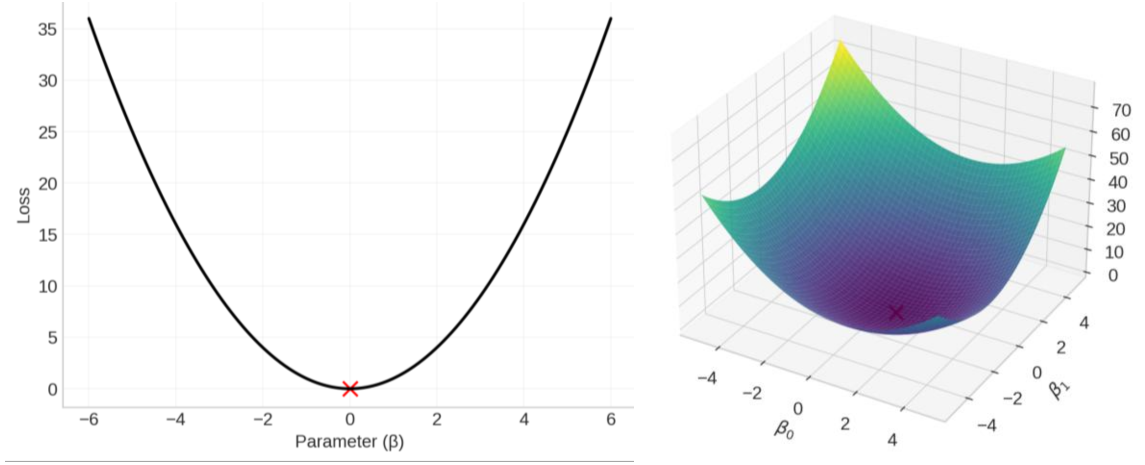
- Provides the exact best-fit parameters in one step.
- Intuitively: fitting a line = solving a system of equation

- Works when optimization problem is convex and has a nice mathematical structure
- But, for large datasets or many features, computing the matrices can be computationally expensive and numerically unstable.

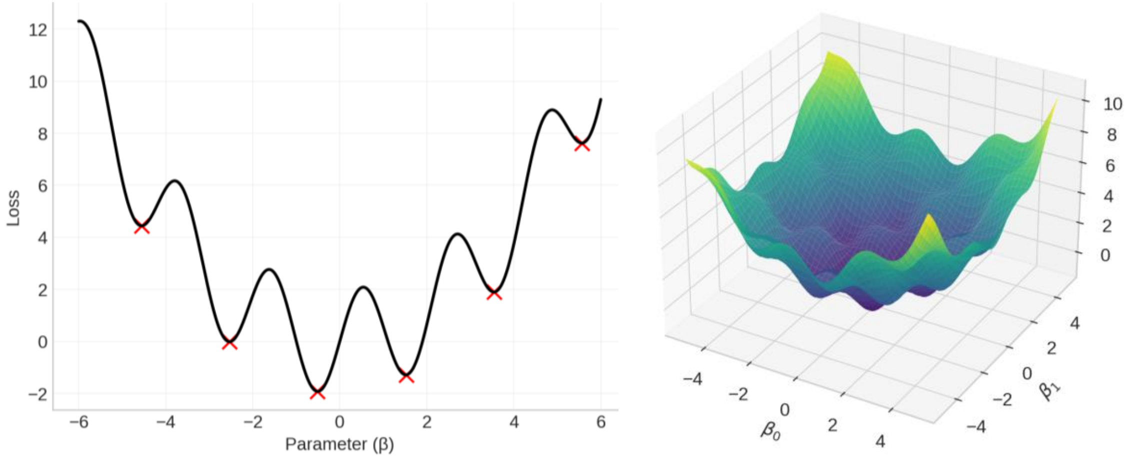
Complex real-world problems are non-convex (e.g. neural networks)
Non-convex landscape have multiple local minima, saddle points, highly irregular “loss surfaces”
Analytical solutions usually impossible

Here, we rely on iterative optimization methods (e.g., gradient descent)

Convex

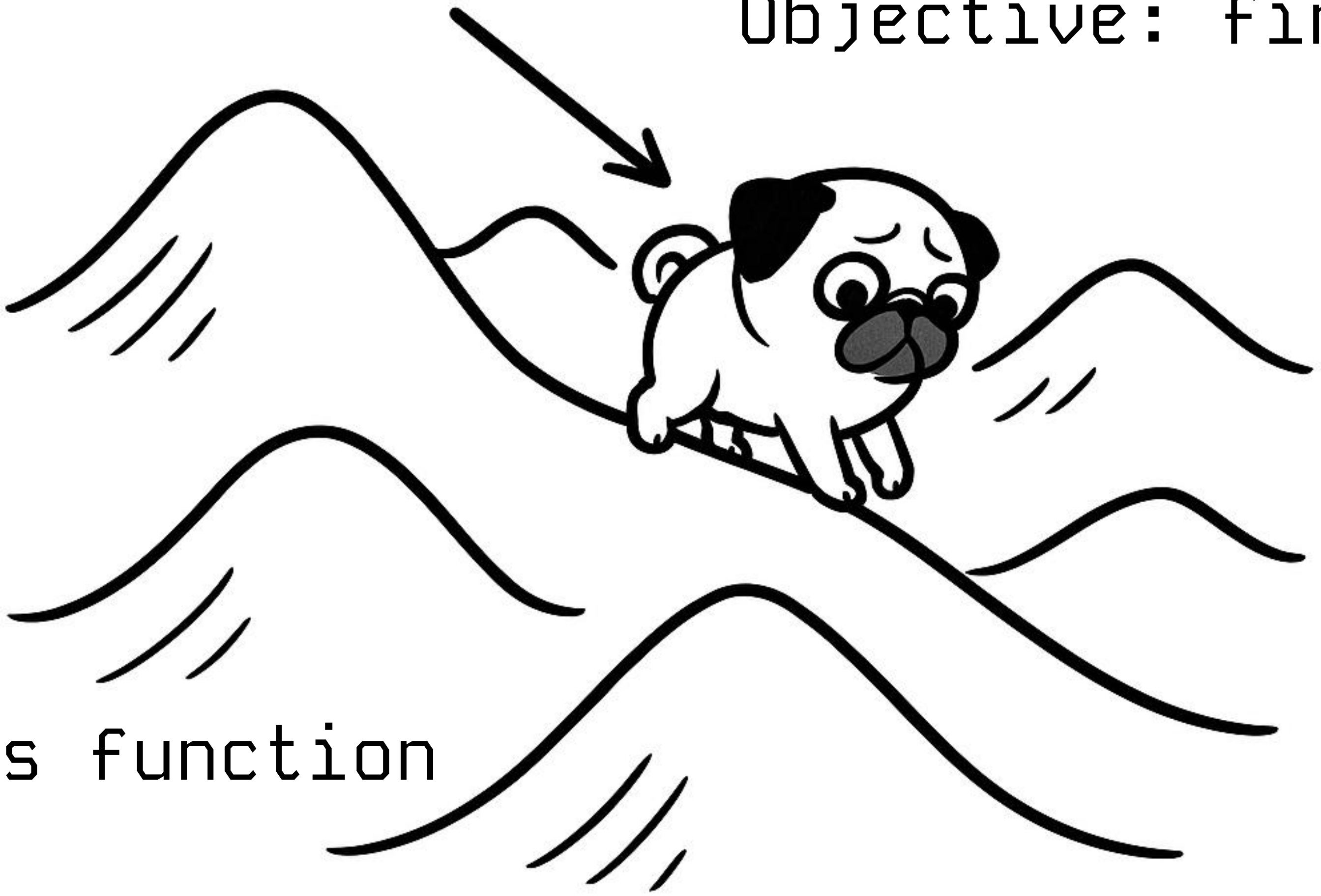


Non-Convex



2 Gradient Descent (Optimization)

Objective: find global minima



Loss function

?

2 Gradient Descent (Optimization)
What is the mathematical function that describes the slope?

Derivative

What do you think is a good approach for telling the model how to change (what is the step size) to become better?

If the step is proportionate to the slope, then you avoid overshooting

How to generalize to more than one predictor

Take derivative with respect to each coefficient and do the same sequentially

2 Gradient Descent (Optimization)

Idea: iteratively adjust parameters β_0 and β_1 to minimize the loss function
Think of it as walking downhill on the loss landscape until reach bottom
Works for both convex (easy) and non-convex (hard) problems

Equation (general form):

$$\theta := \theta - \alpha \cdot \nabla J(\theta)$$

where:

- θ = parameters (e.g., β_0, β_1)
- α = learning rate (step size)
- $\nabla J(\theta)$ = gradient of the loss function

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2$$

Gradients:

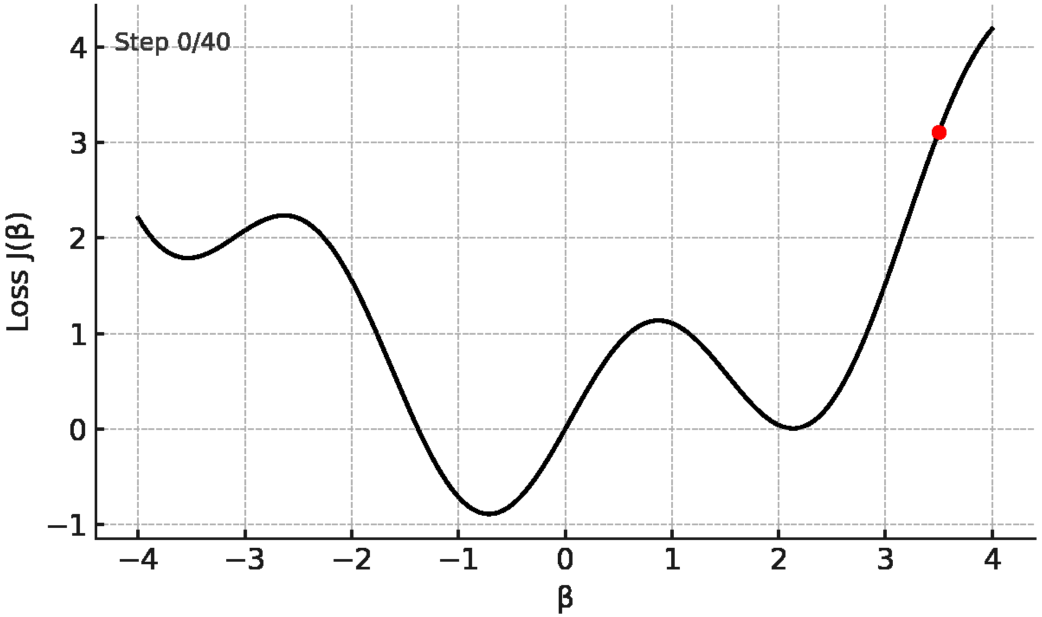
$$\frac{\partial J}{\partial \beta_0} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)$$

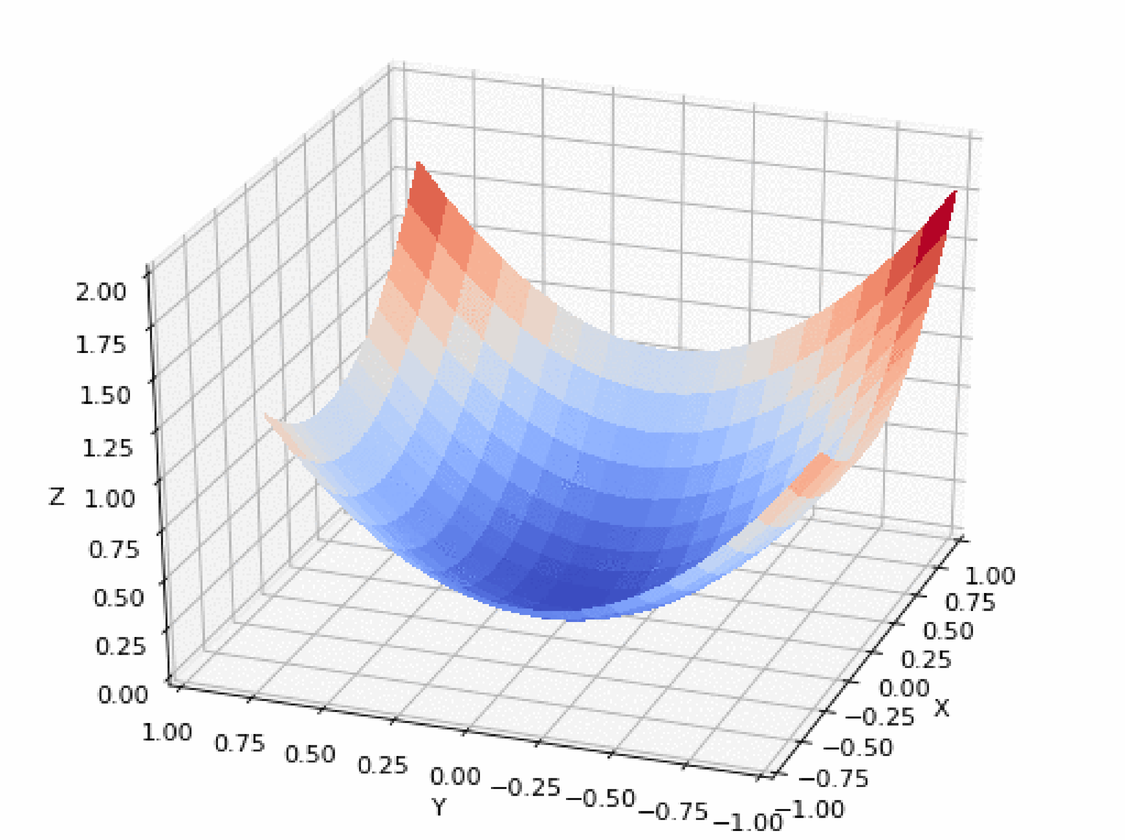
$$\frac{\partial J}{\partial \beta_1} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i) x_i$$

Update Rules:

$$\beta_0 := \beta_0 - \alpha \cdot \frac{\partial J}{\partial \beta_0}, \qquad \beta_1 := \beta_1 - \alpha \cdot \frac{\partial J}{\partial \beta_1}$$

2 Gradient Descent (Optimization)





2 Gradient Descent (Optimization)

We still need to compute derivatives

We need to set the learning rate

We need to avoid local minima

In Linear Regression, there are no local minimas because loss function is convex

We will talk about this again in later lectures (e.g. Neural Networks)

Google Colab

https://bit.ly/BPS5231-L2

Multiple variables
In practice, unlikely that any response variables Y depends solely on one predictor

Extends simple Linear Regression to more than one input variable

Instead of predicting using just one X, we use multiple features

$$Y = f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon$$

- Y: predicted output (e.g. building energy use)
- X_1: Outdoor temperature
- X_2: Floor area
- X_3: Number of occupants
- …

Multiple variables

Compact way to write using matrices

$$\hat{y} = X\beta$$

- X is matrix of inputs (rows = samples, columns = features)
- β is vector of coefficients
- Y is the vector of predicted outputs.

$$\hat{y} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

Still minimize MSE

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

Closed Form Solution

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

More variables → more expressive model
Risk of:
- Multicollinearity
- Overfitting
- Higher computational cost for large X

Polynomial

Linear Regression assumes a straight-line relationship
But most real-world problems are non-linear
Polynomial regression solves this by adding polynomial terms of the inputs

Degree *d* controls model flexibility

Univariate Case

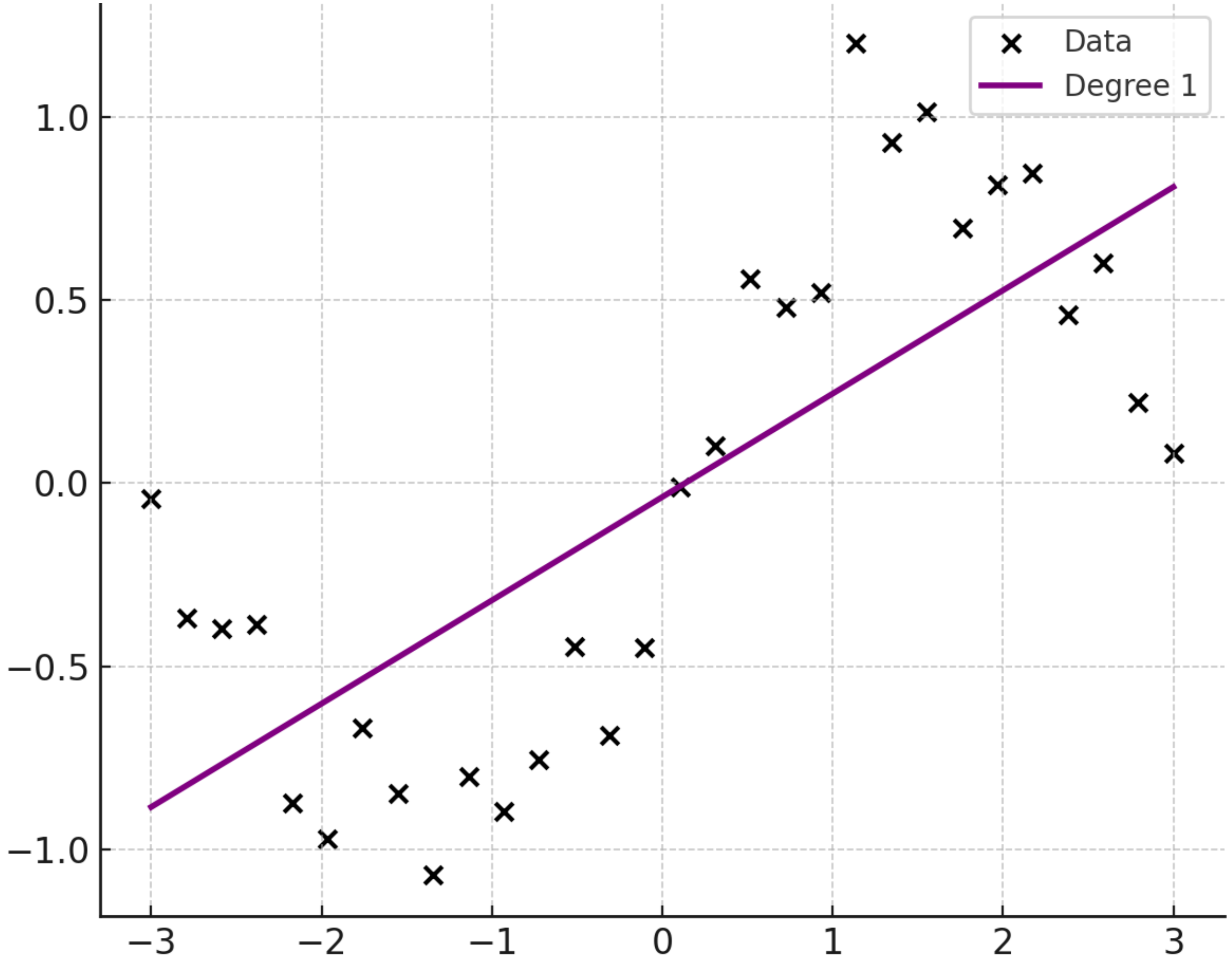
$$Y = f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \dots + \beta_d X^d + \varepsilon$$

Create new features by expanding powers of x

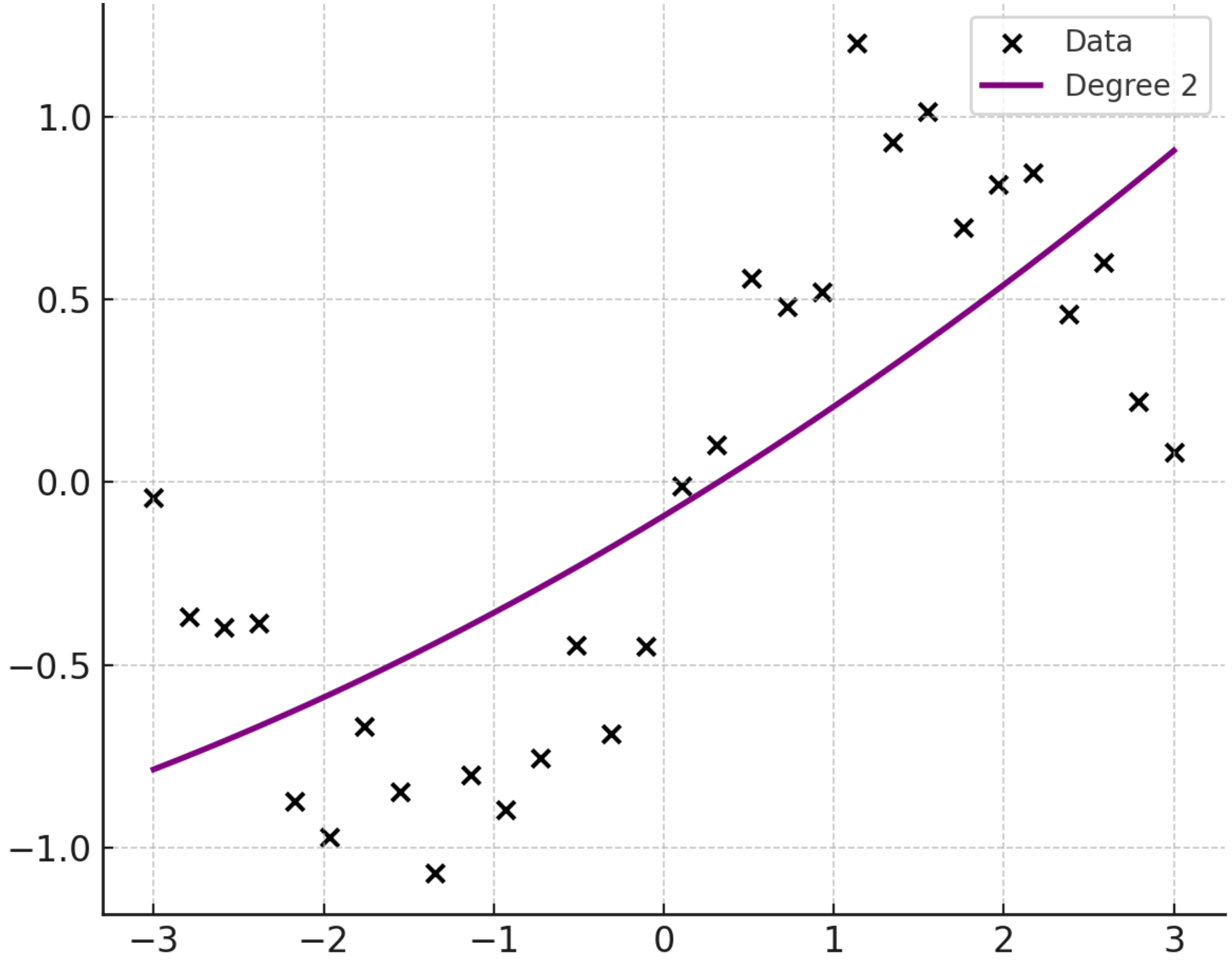
$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^d \end{bmatrix}$$

$$\hat{y} = X\beta$$

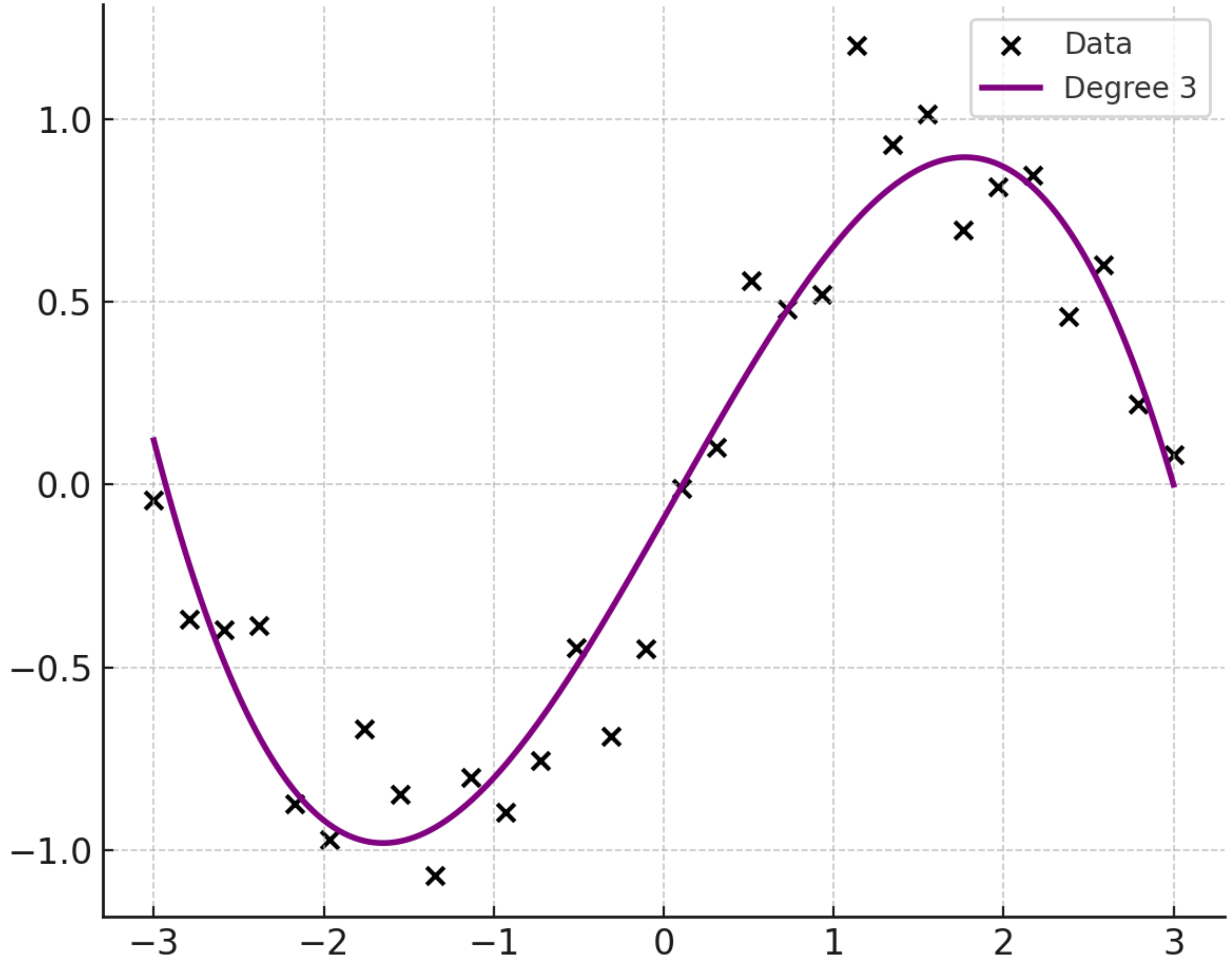
Polynomial Degree 1



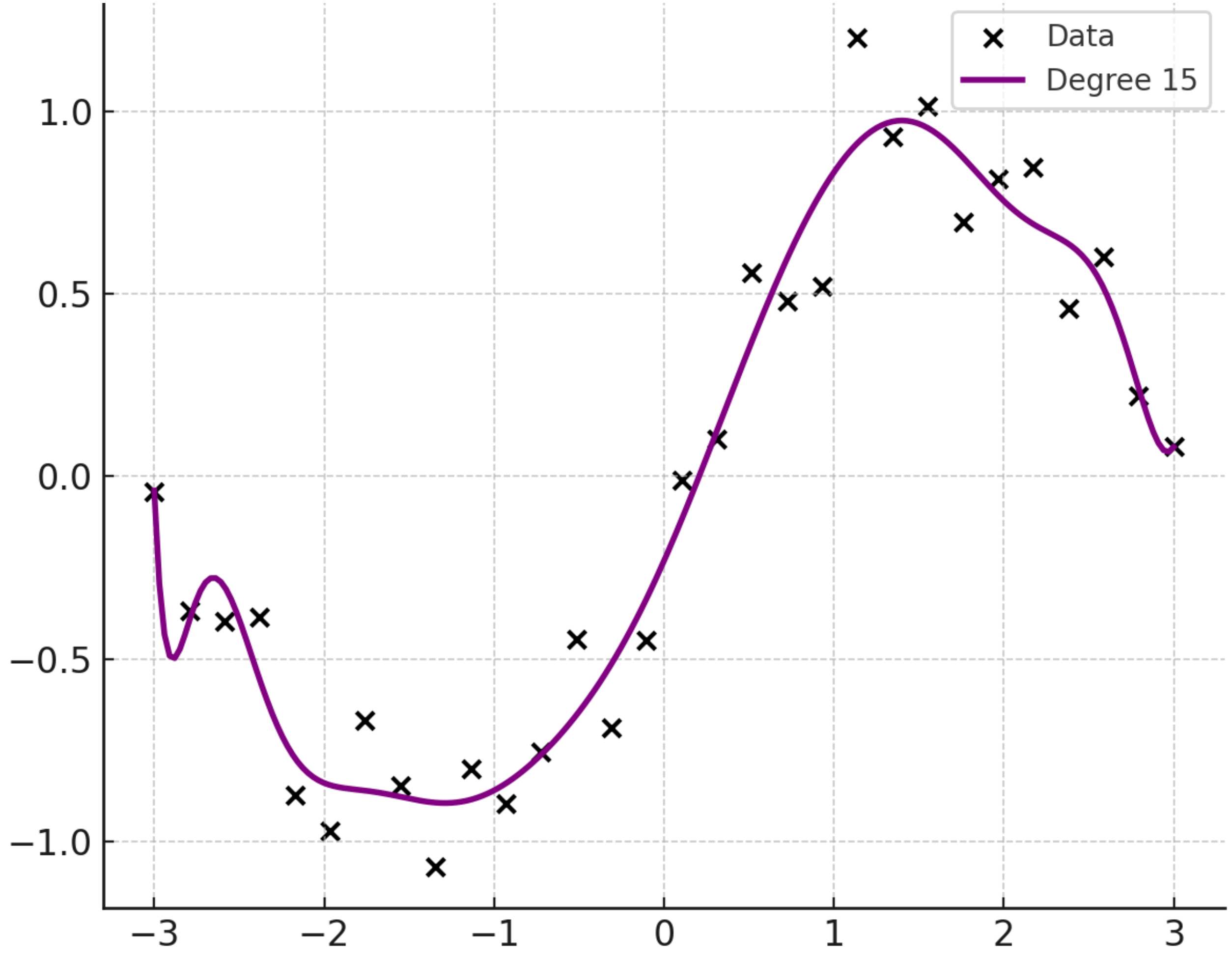
Polynomial Degree 2



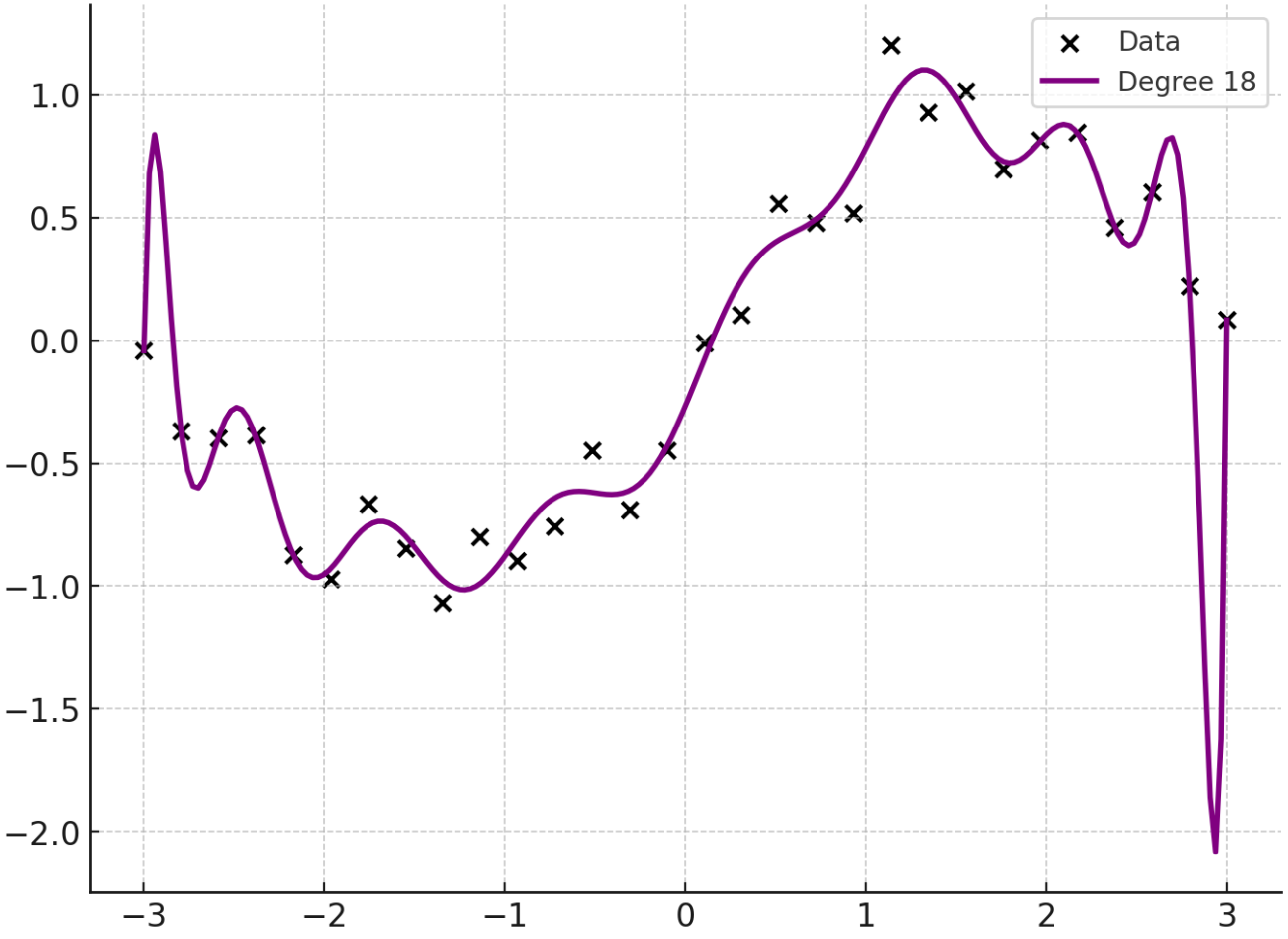
Polynomial Degree 3



Polynomial Degree 15



Polynomial Degree 18



Google Colab
https://bit.ly/BPS5231-L2

L02.1

Supervised 1

Linear Regression

Brute Force

Exact Method (Analytical)

Gradient Descent

Multilinear

Polynomial

L02.2

Supervised 2

Model Evaluation

Bias-Variance Trade-off

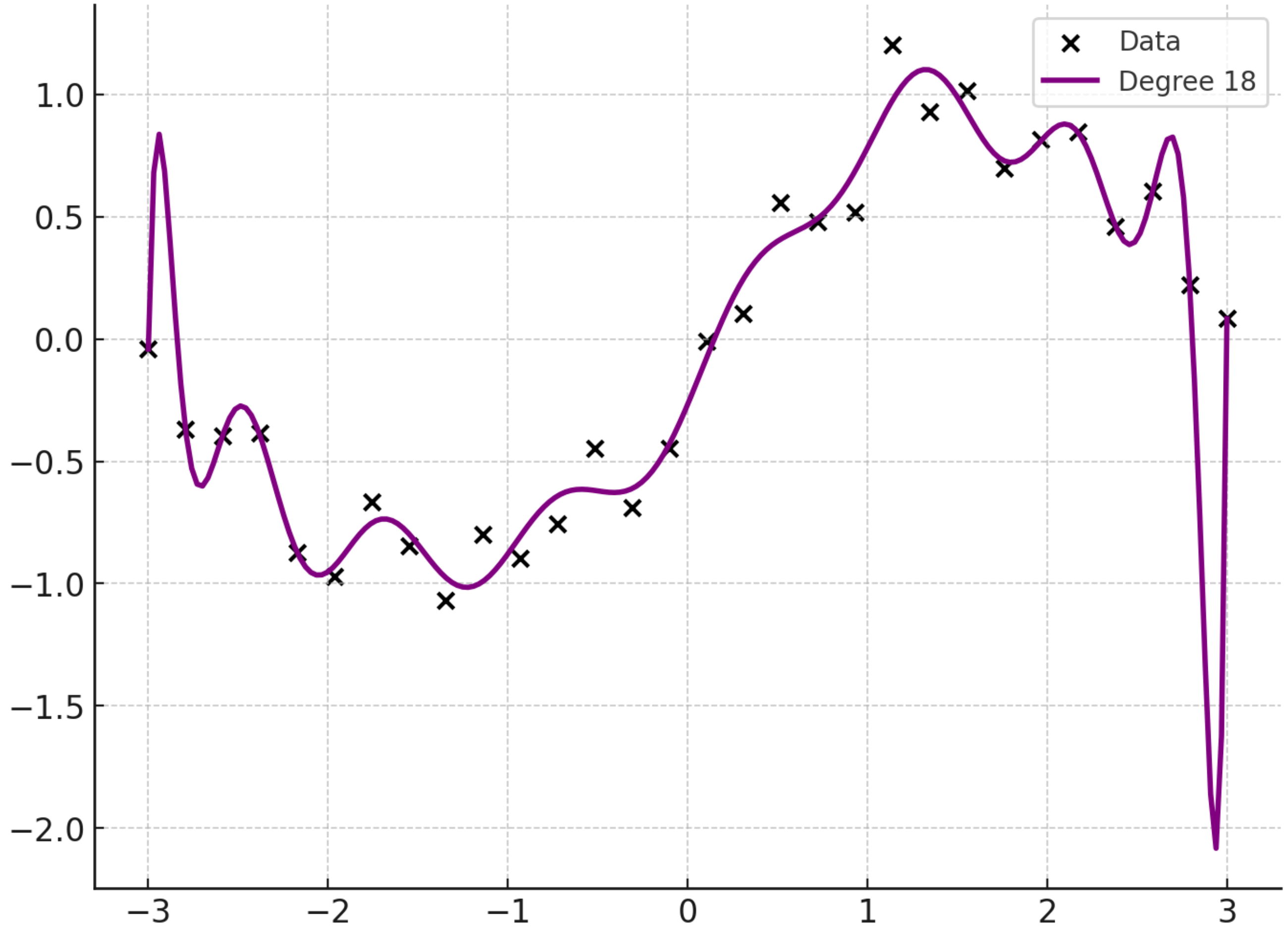
Parametric vs Non-Parametric

KNN

L02.3

Beyond Linearity

Recall our not-so-good 18 degree friend

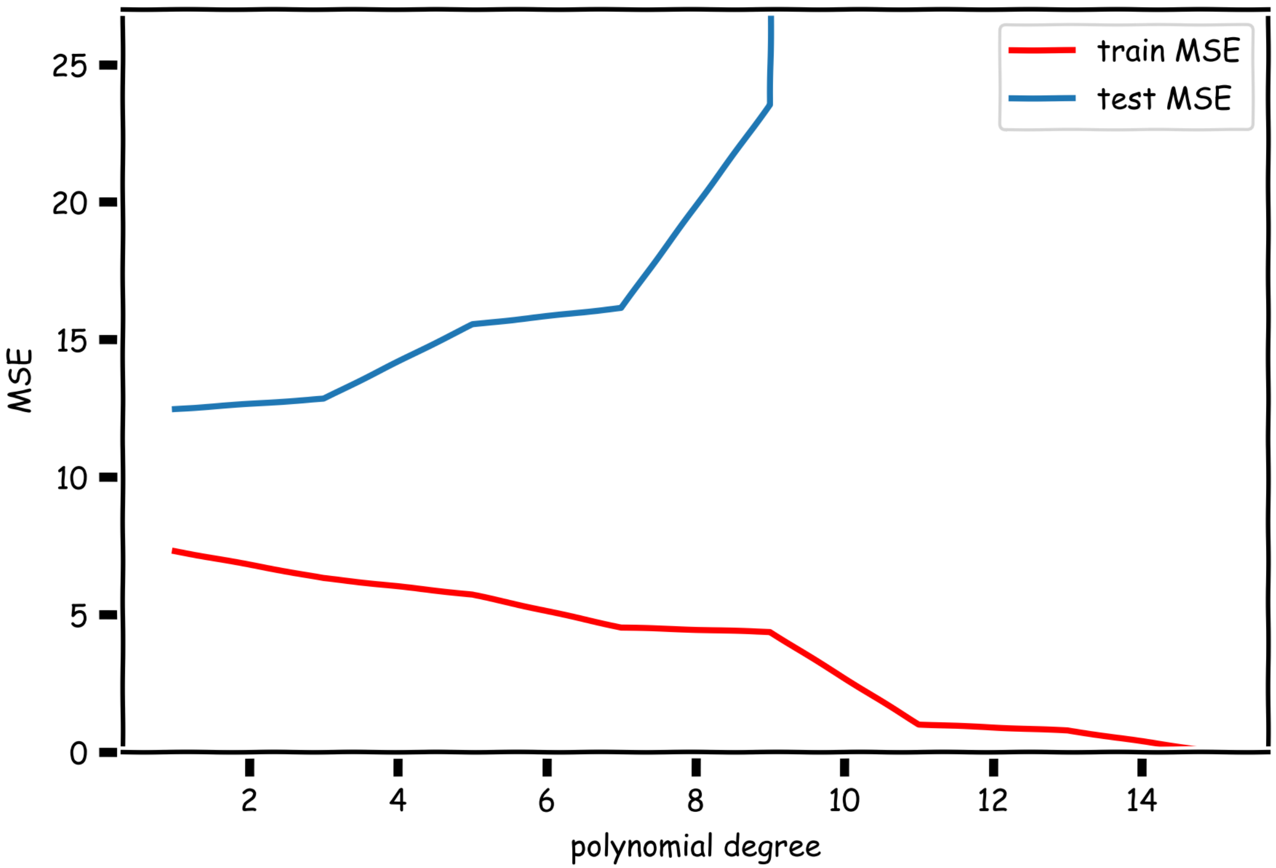


Polynomial Degree 18

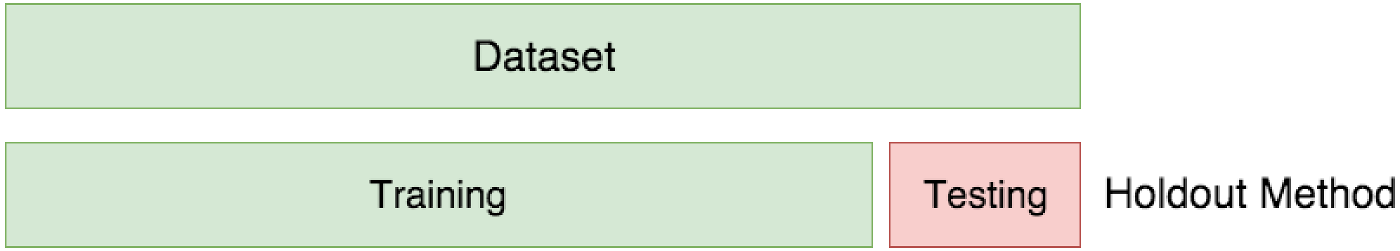
What is the problem?

Overfitting

We need to predict well on the test set – i.e. generalizability

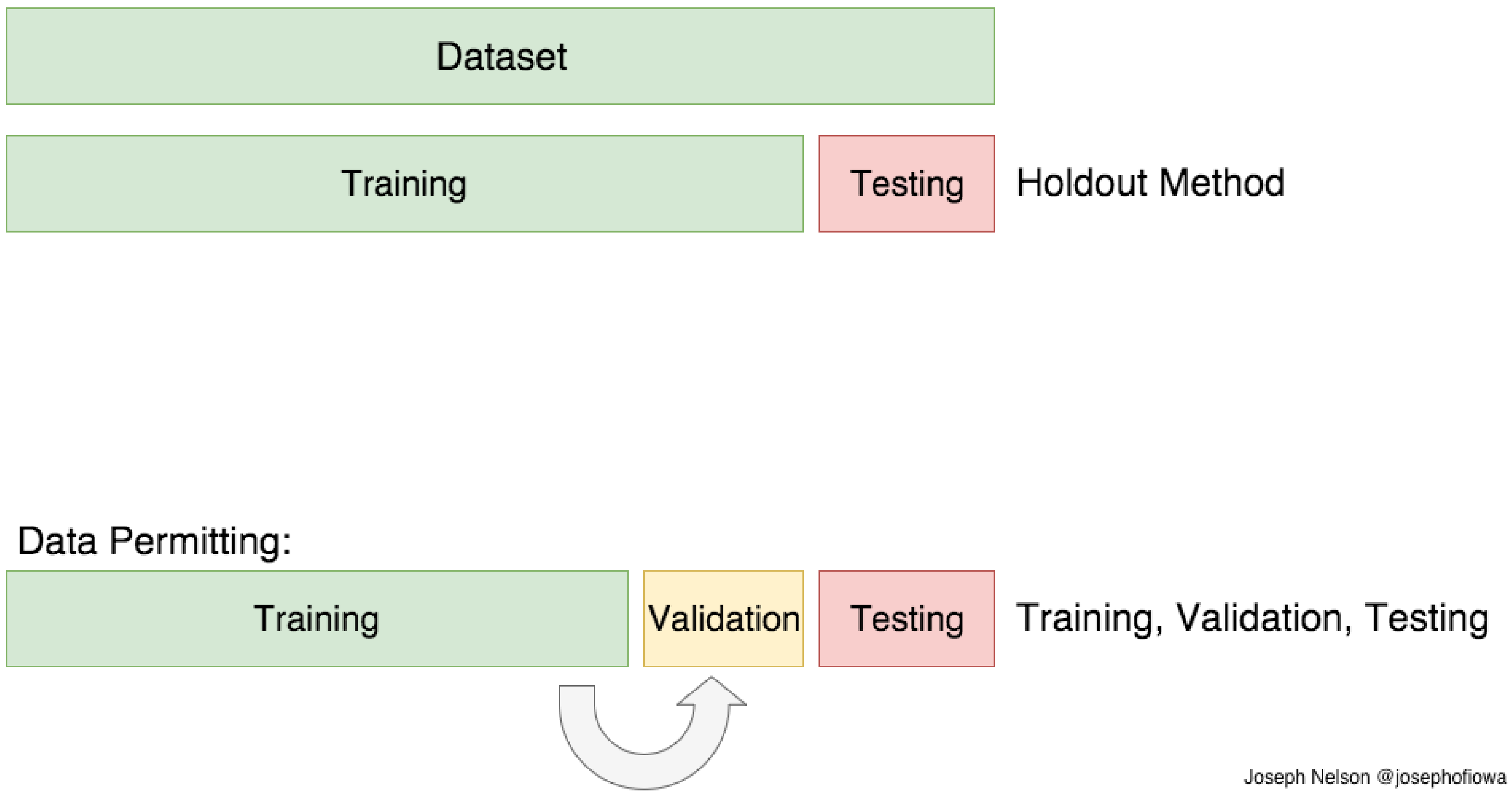


How would you report performance of model?

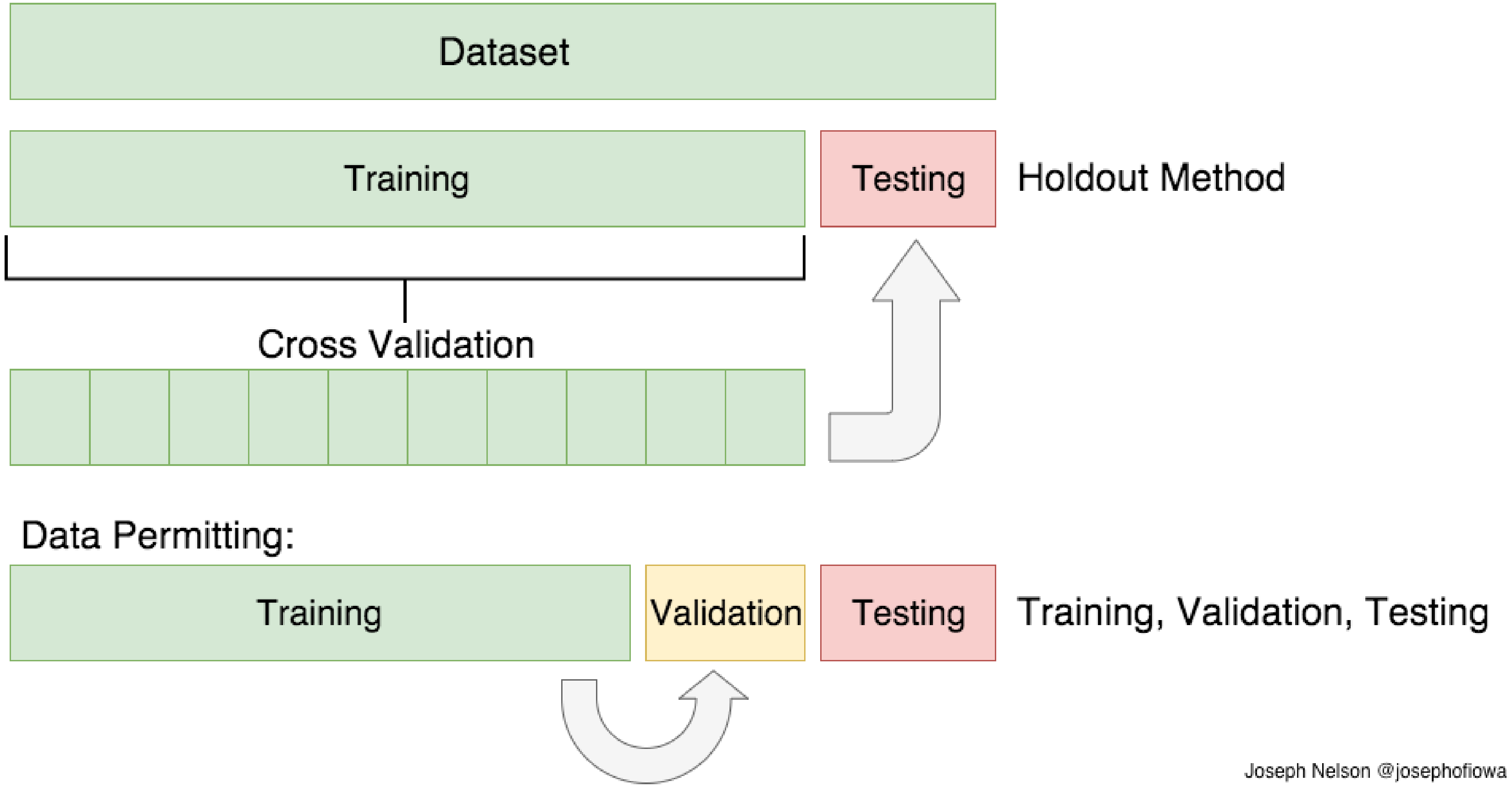


R^2/MSE_{train} R^2/MSE_{test}

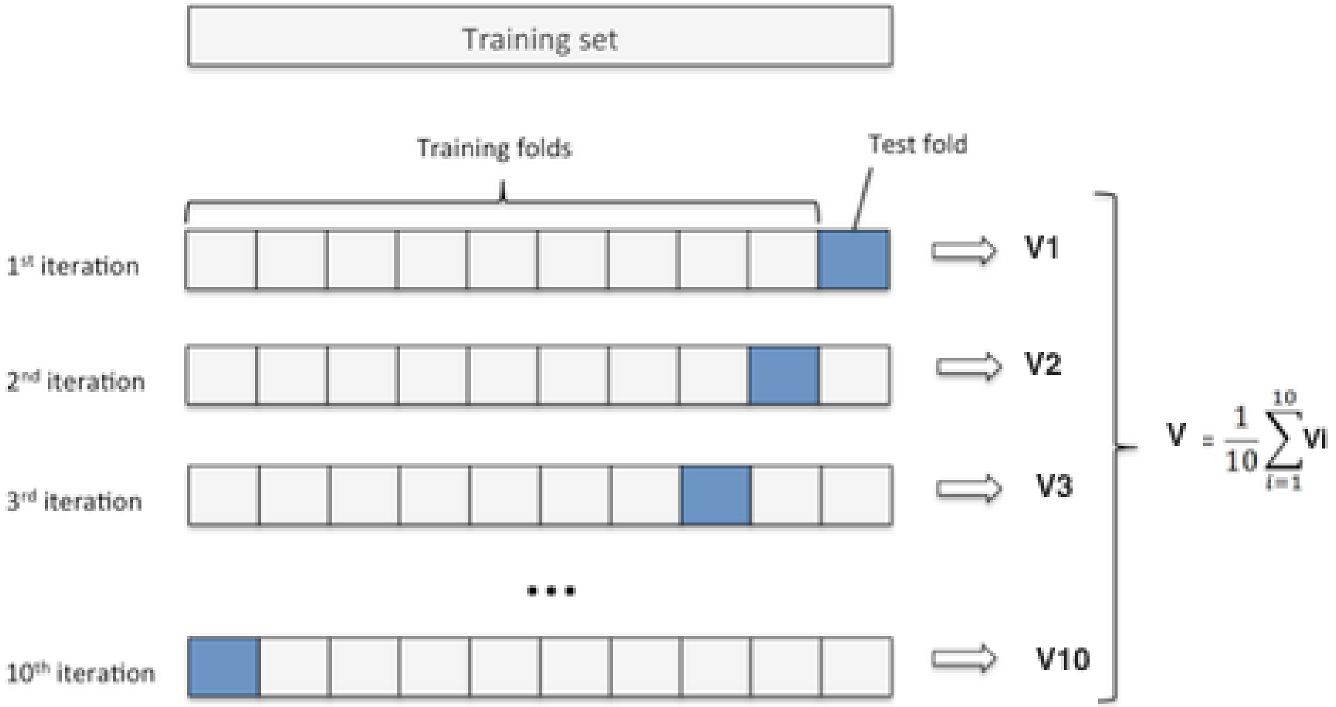
How would you report performance of model?



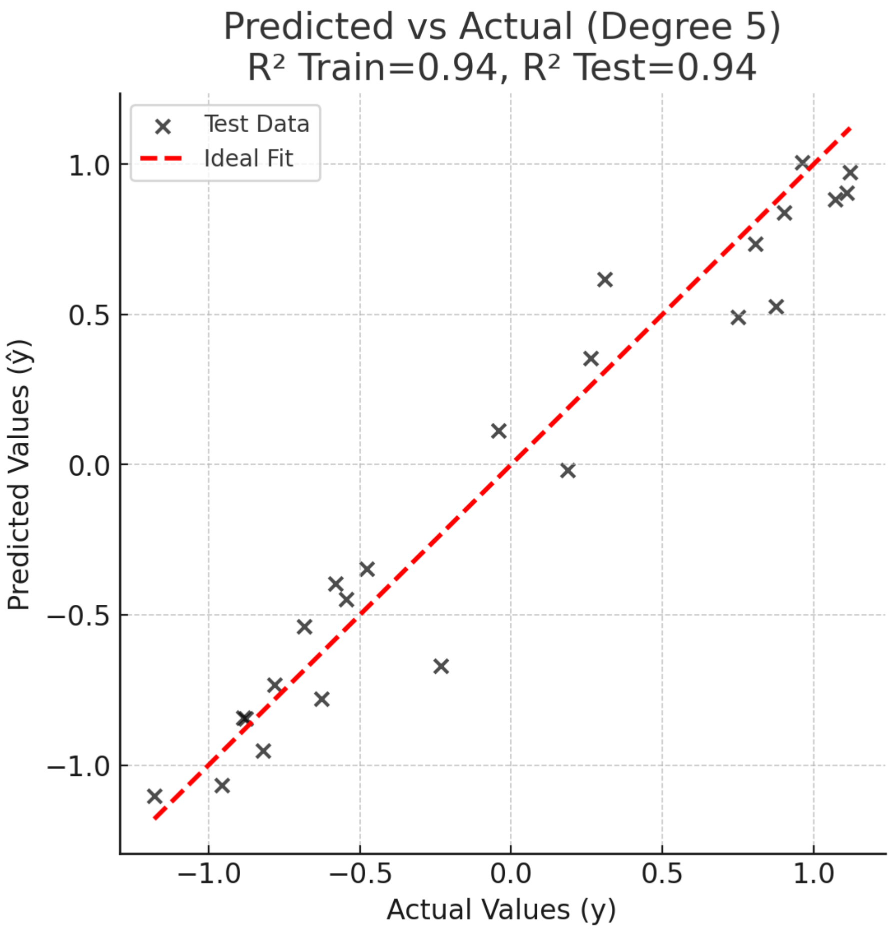
How would you report performance of model?



How would you report performance of model?



How would you report performance of model?



Theory

Bias

Amount that a model's prediction differs from target values

Simplifying assumptions in model to make approximating target function easier

Variance

Measures inconsistency of different predictions using different training sets

(not accuracy)

Amount that estimates of target function will change with different datasets

Why is there a trade-off?

Less variance algorithms tend to be less complex

(think linear regression)

Simple or rigid underlying structure

Low bias algorithms tend to be more complex, with flexible underlying structure

(e.g. trees)

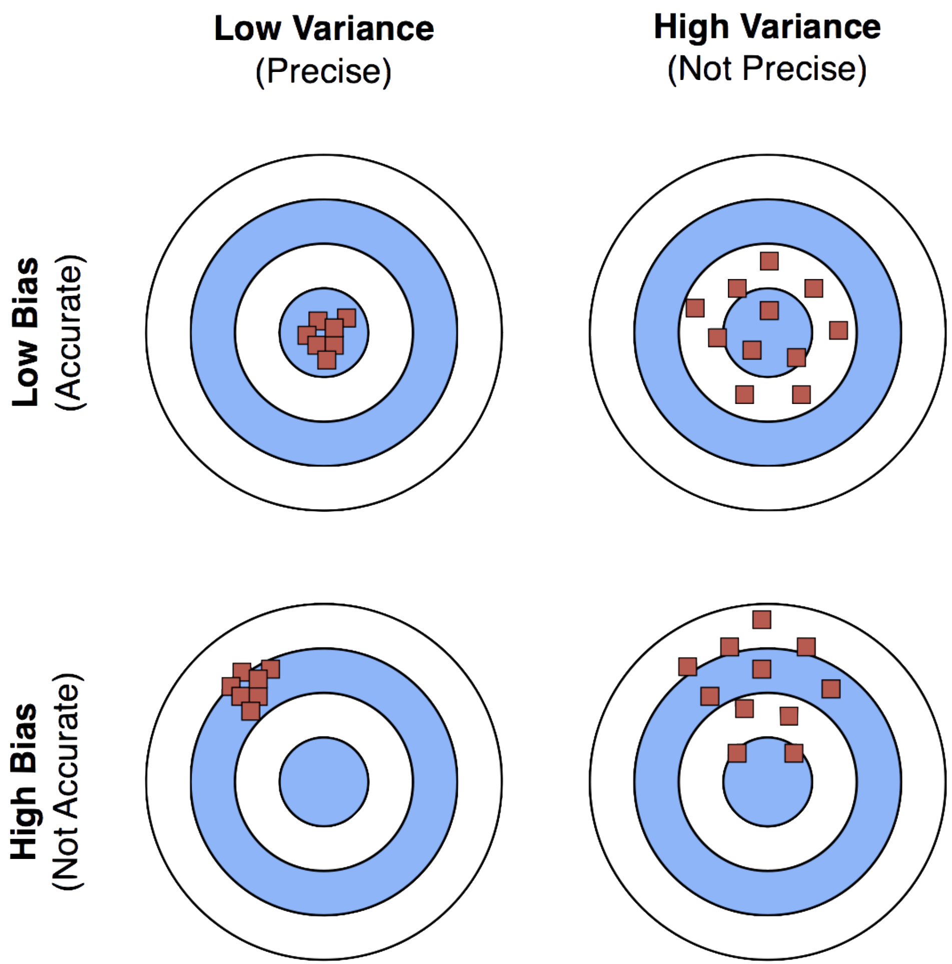
Easy to overfit

Bias + Variance + Irreducible Error

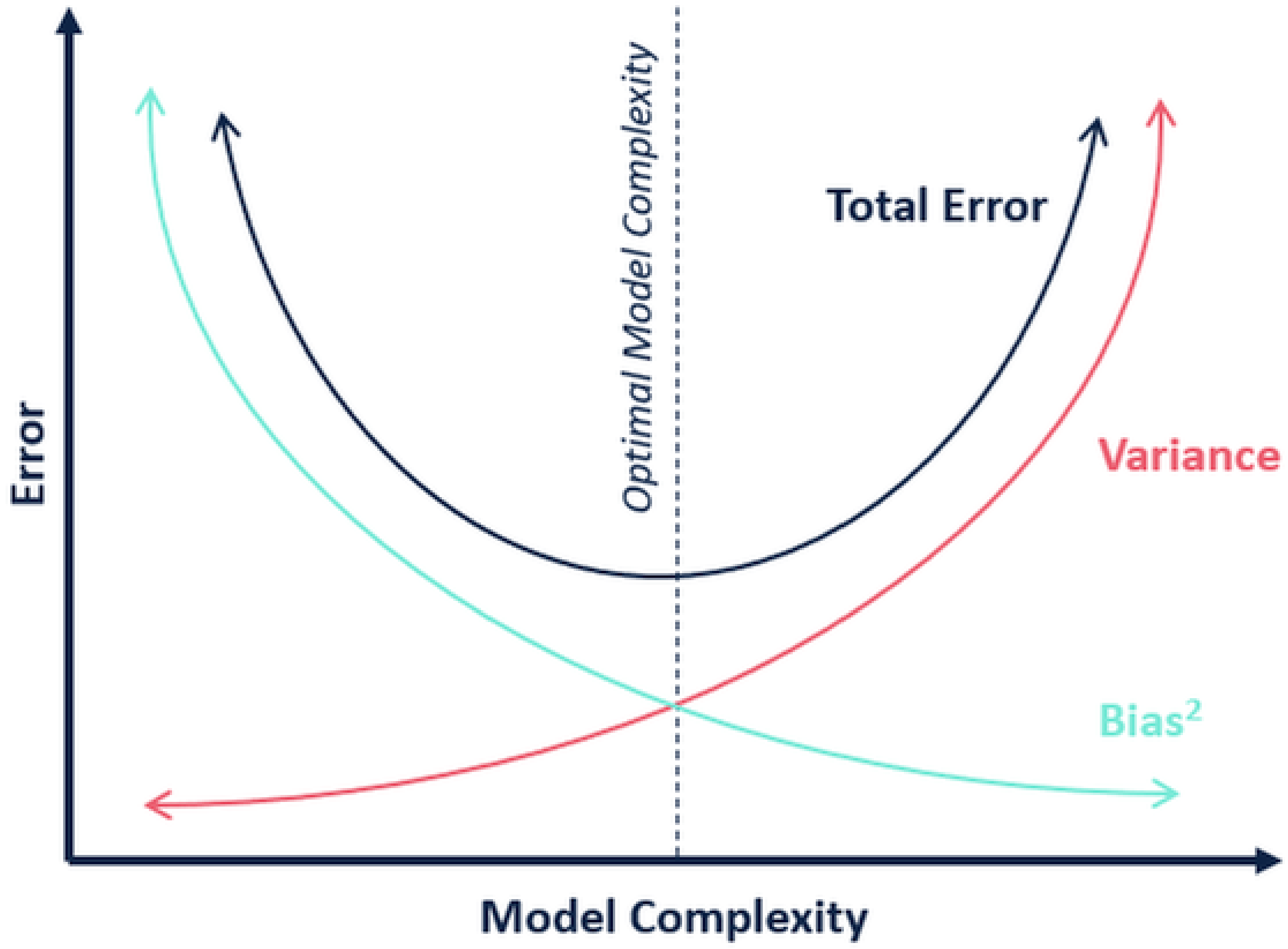
First two types usually stemmed from algorithm and hyperparameter choices

Can be reduced

Bias-Variance Trade-off



Bias-Variance Trade-off



Simple Situation 1 – High Bias

Cause of poor performance is high bias, model not sufficiently robust

With desire tolerance / error threshold ϵ

Symptoms

Training error higher than ϵ, test error usually bad as well

Remedies

Use more complex models (e.g. non-linear)

Add features

Boosting (will cover later)

Simple Situation 2 – High Variance

Cause of poor performance is high variance, model doesn't generalize well

With desire tolerance / error threshold ϵ

Symptoms

- Test error much higher than training error
- Training error lower than ϵ, test error higher than ϵ

Remedies

- Regularize
- Reduce model complexity
- Find more training data

Parametric vs Non-parametric

So far, we've seen parametric models (Linear Regression, Polynomial Regression).

These assume a fixed functional form (e.g., a straight line, polynomial) and estimate a finite set of parameters.

But what if:

The data is complex and doesn't fit any simple function?

We want predictions based purely on data similarity rather than formulae?

Parametric vs Non-parametric

Parametric Methods: They reduce the problem of estimating f(x) to finding a finite set of parameters
Example: In linear regression, we only need to find the intercept and the slope

Approach:
(1) Make some assumptions about the functional form of f (e.g., linear, quadratic, or …),
(2) Estimate the required parameters using training data

Nonparametric Methods: No assumption about the functional form of f

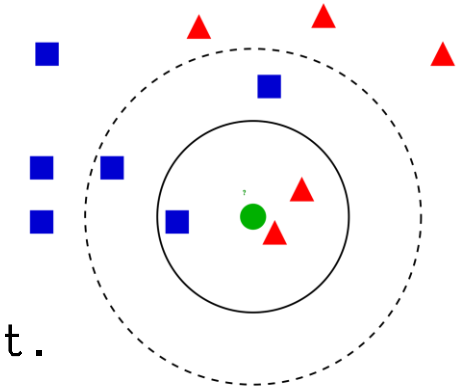
Advantage: They can fit a variety of different shapes of f to data

Disadvantages:
(1) Some harder than parametric methods,
(2) Require a good amount of data to accurately estimate f

K-Nearest Neighbors (KNN)

Idea: To predict the output for a new data point:

- Look at the K closest neighbors in the training data (using a distance metric, usually Euclidean).
- Aggregate their values:
  - Regression: average their target values.
  - Classification: majority vote of their labels.



Key property: No “training” in the classical sense — just storing the dataset.

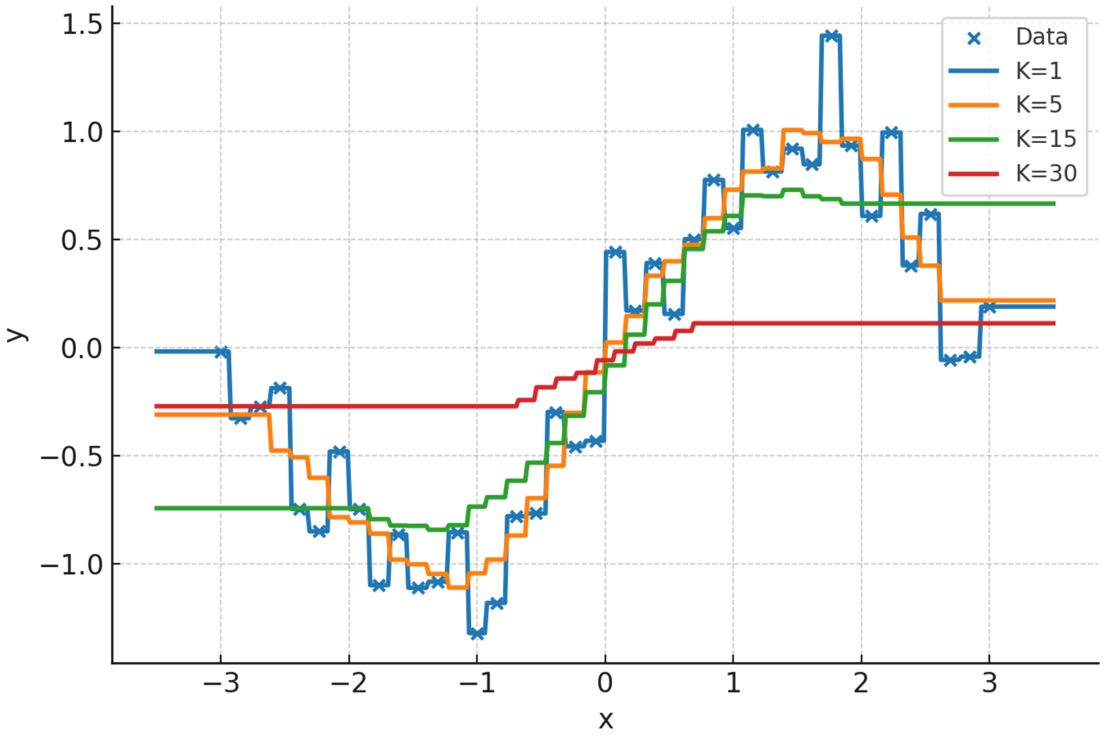
Prediction = similarity-based lookup.

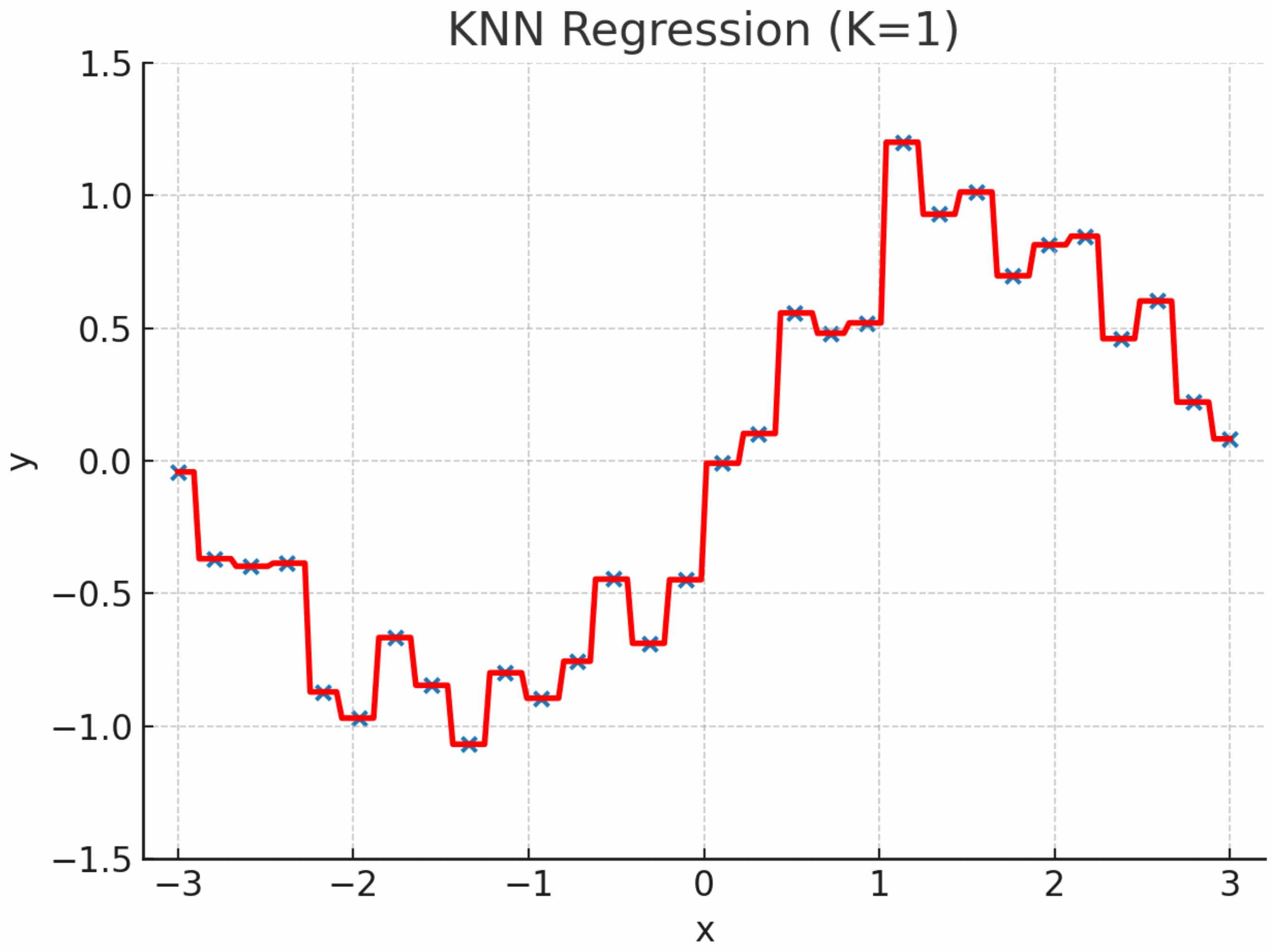
Intuition: *“Tell me who your neighbors are, and I’ll tell you who you are.”*

K-Nearest Neighbors (KNN)

K=1: Each point is classified by its single closest neighbor → very flexible but noisy (overfit)

K=5: Each point is classified by majority vote among its 5 closest neighbors → smoother




KNN Regression (K=1)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
−3
−2
−1
0
1
2
3
x

Beyond Linearity

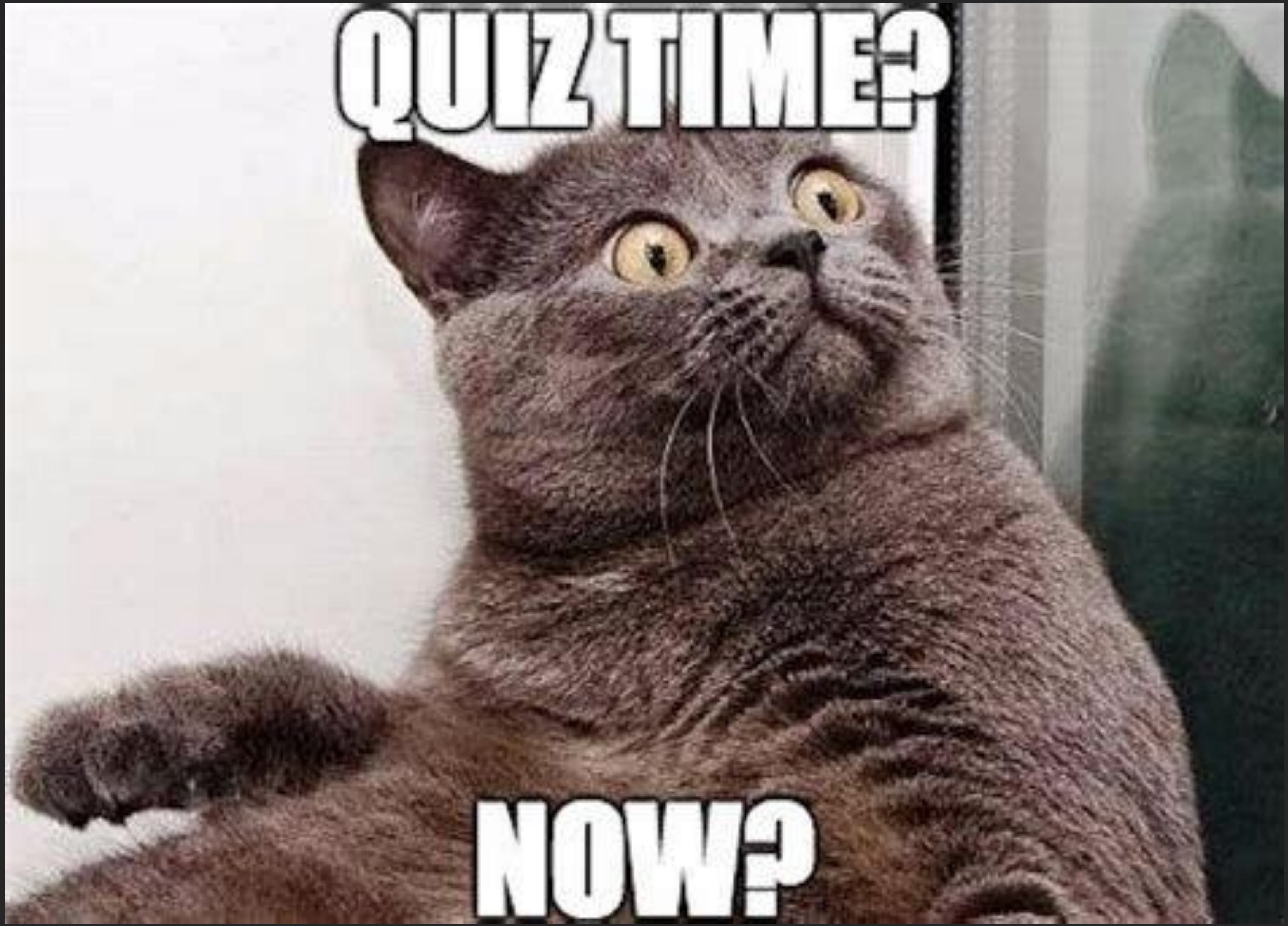
In-class activity

Google Colab
https://bit.ly/BPS5231-L2

Go To https://bit.ly/BPS5231-L2-BeyondLinearity

Please install Rhino by next class

(this is not a pure computer science course)


QUIZ TIME?
NOW?

Prize
AIRPODS ULTRA