LEVEL 2: Introduction to Machine Learning

Day 2

Course Outline

- Linear Regression
- Logistics Regression
- Classification Metrics
- Support Vector Machine (SVM)
- Decision Trees
- Random Forests
- Gradient Boosting

Learning Objectives

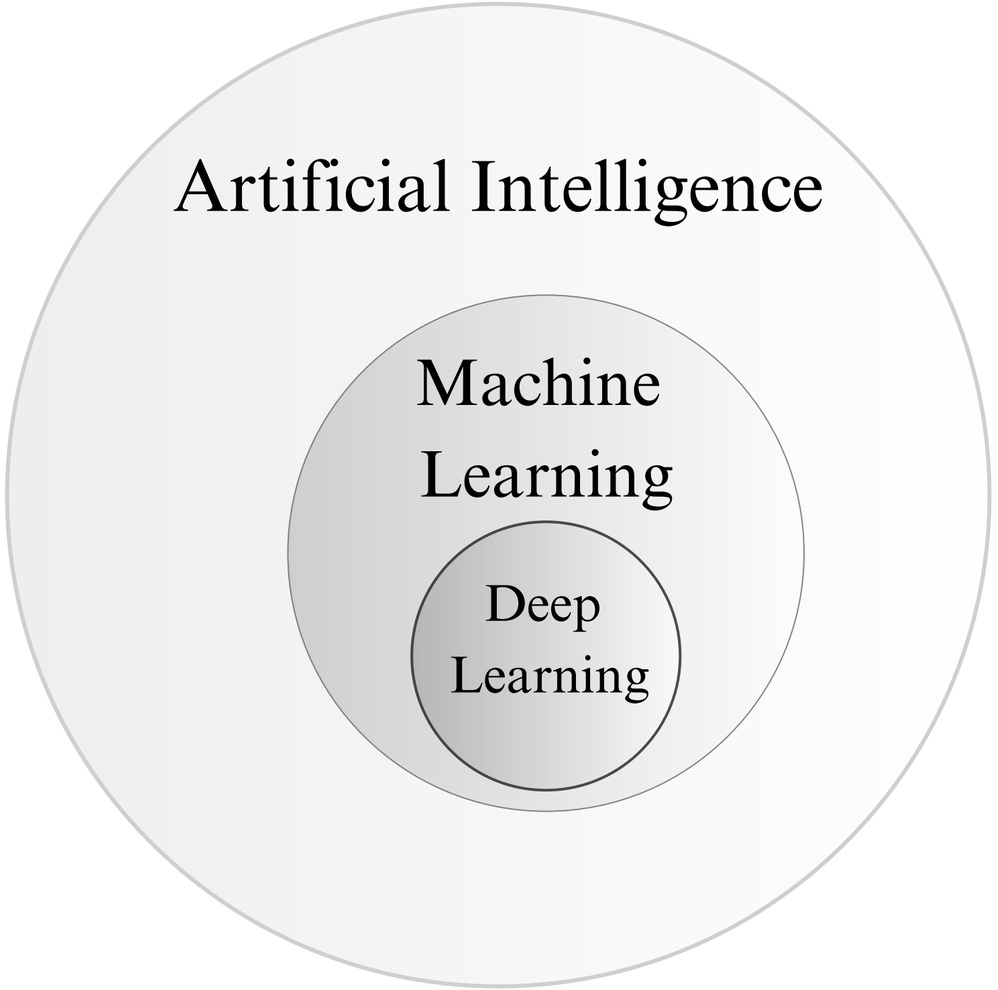
- Understand and apply linear and logistic regression for regression and classification tasks.
- Explain how probabilistic outputs and loss functions are used in classification models.
- Evaluate classification models using appropriate performance metrics beyond accuracy.
- Understand the principles of margin-based learning in Support Vector Machines (SVM).
- Build and interpret tree-based models, including decision trees and random forests.
- Explain the concept of ensemble learning and boosting and compare different models.

Machine Learning

Recap

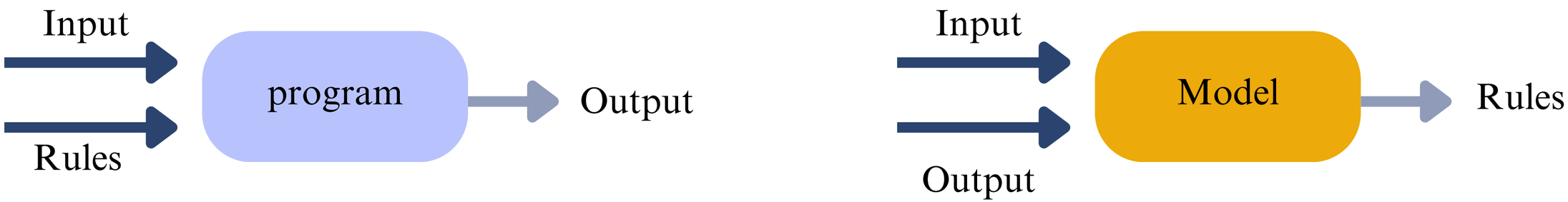
Machine Learning

- Machine Learning is a branch of Artificial Intelligence that enables computers to learn patterns from data and make predictions or decisions without being explicitly programmed.



Machine Learning

- Old programming takes input and rules to generate output.



- Machine Learning takes input and output examples to **learn** the rules

Machine Learning

Supervised Machin Learning

Problem 1:

Predicting home price

Regression

An ML task where the goal is to predict a continuous numerical value.

Problem 2:

Classify the customer segment

Classification

An ML task where the goal is to assign each input to a predefined category.

Unsupervised Machin Learning

Problem 3:

Discovering groups of buildings based on sustainability usage

Clustering

An ML task where the goal is to discover natural patterns or groups

Machine Learning

Linear Regression

Linear Regression

- Motivation

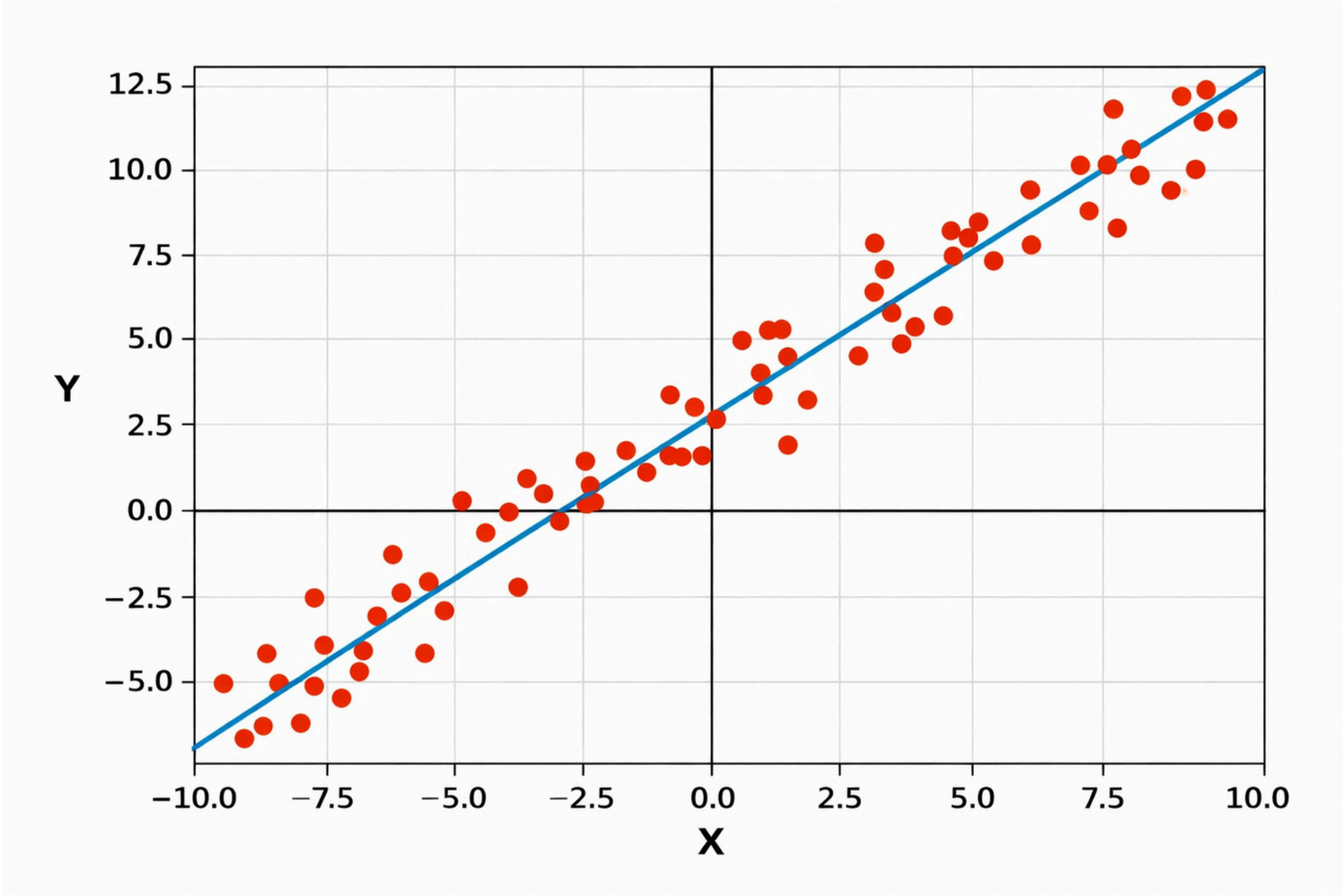
- Many real-world problems involve understanding how one variable changes with another.

Linear Regression

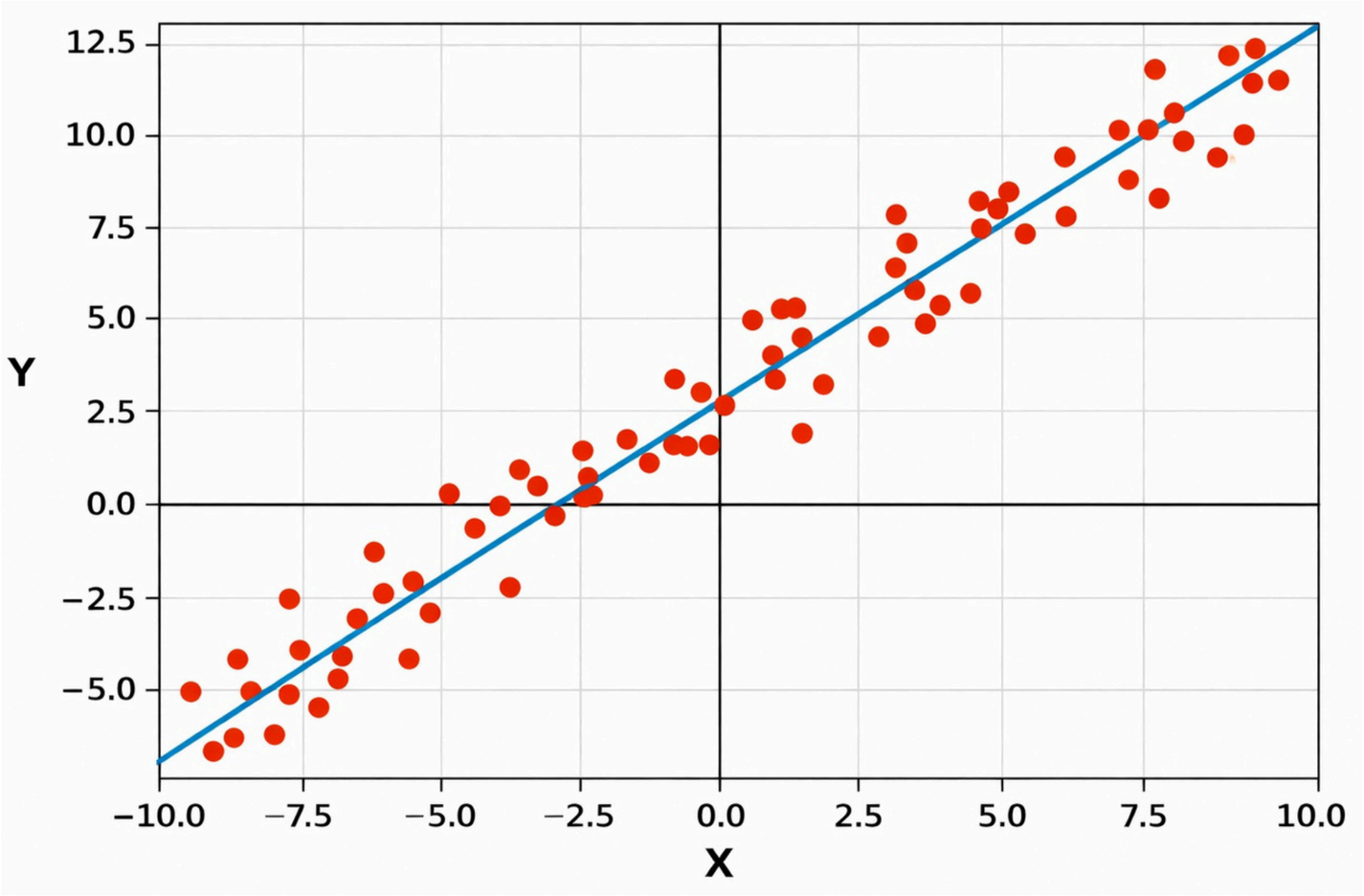
Motivation

- Linear regression answers key questions:
  - Does X affect Y?
  - How strong is the effect?
  - In which direction does Y change when X increases?

Linear Regression



Linear Regression



$$Y = mX + b$$

Y: Response Variable (Dependent Variable / Target)

X: Covariate / Independent Variable (Regressor)

m: Slope (Weight / Coefficient)

b: Bias (Intercept)

Linear Regression

- Model (Hypothesis):

$$\hat{y}_i = mx_i + b$$

Linear Regression

- Model (Hypothesis):

$$\hat{y}_i = mx_i + b$$

- Data:

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Linear Regression

- Model (Hypothesis):

$$\hat{y}_i = mx_i + b$$

- Data:

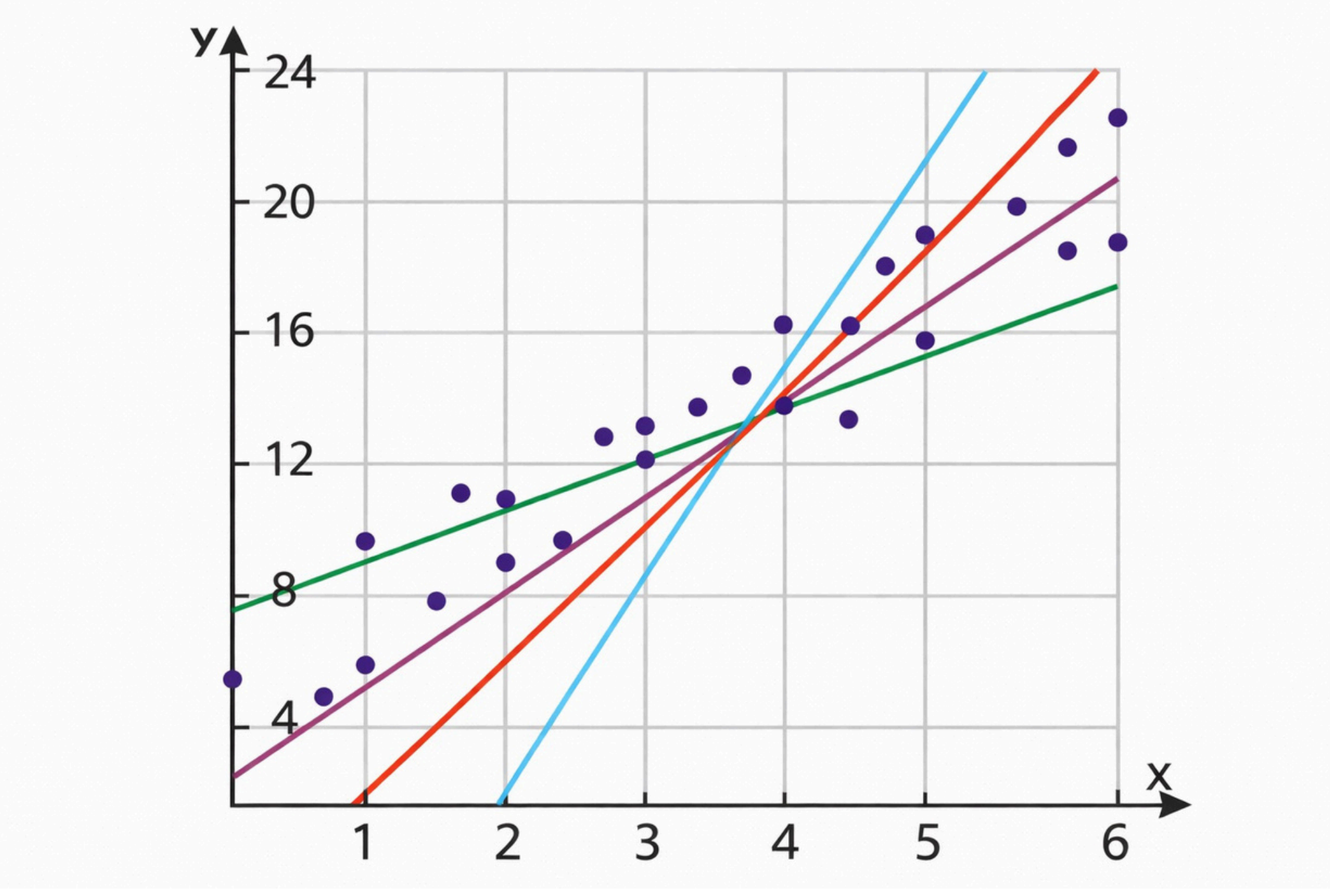
$$\{(x_i, y_i)\}_{i=1}^{N}$$

- Objective:

$$\text{Learn } (m, b) \text{ such that } \hat{y}_i \approx y_i$$

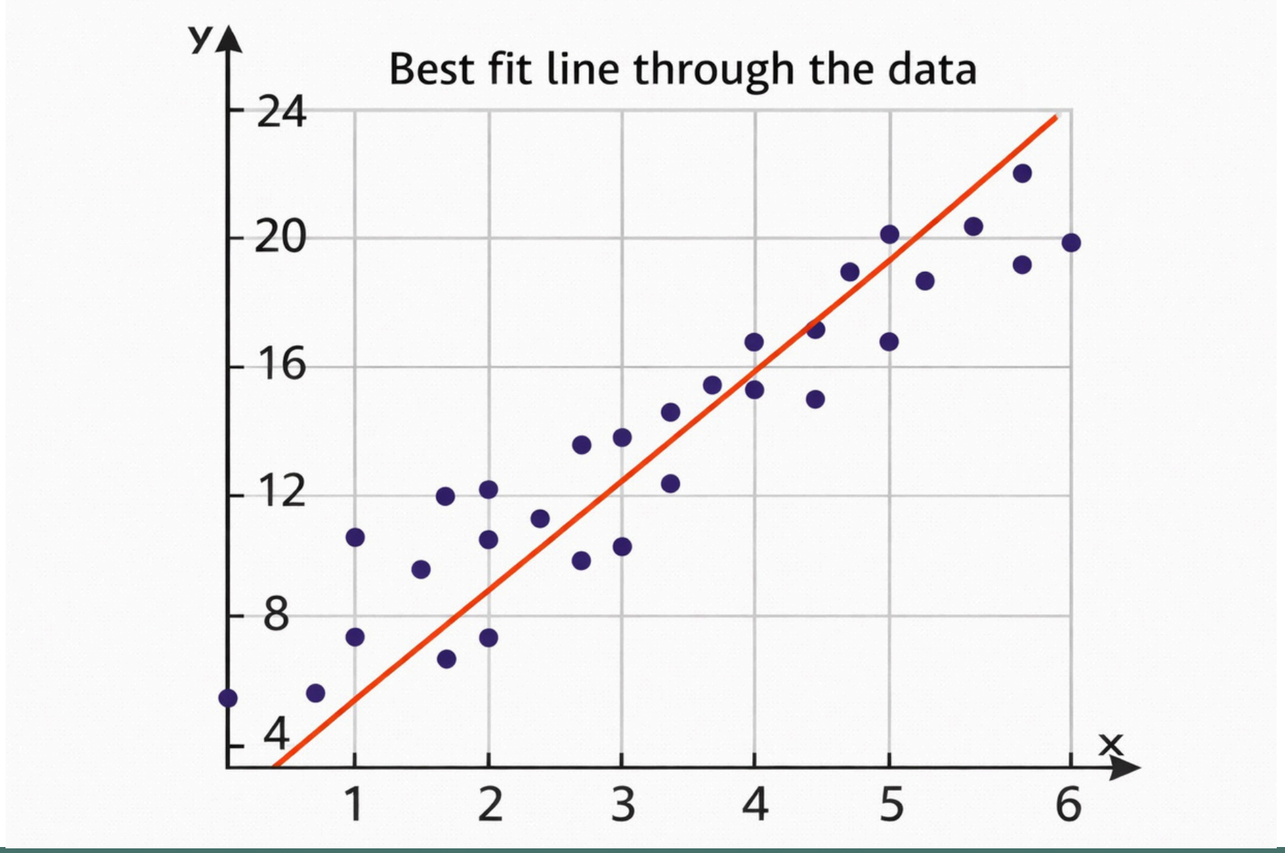
Linear Regression

- There are infinitely many possible lines that can fit the data.



Linear Regression

- The goal is to find an average line that represents the data.

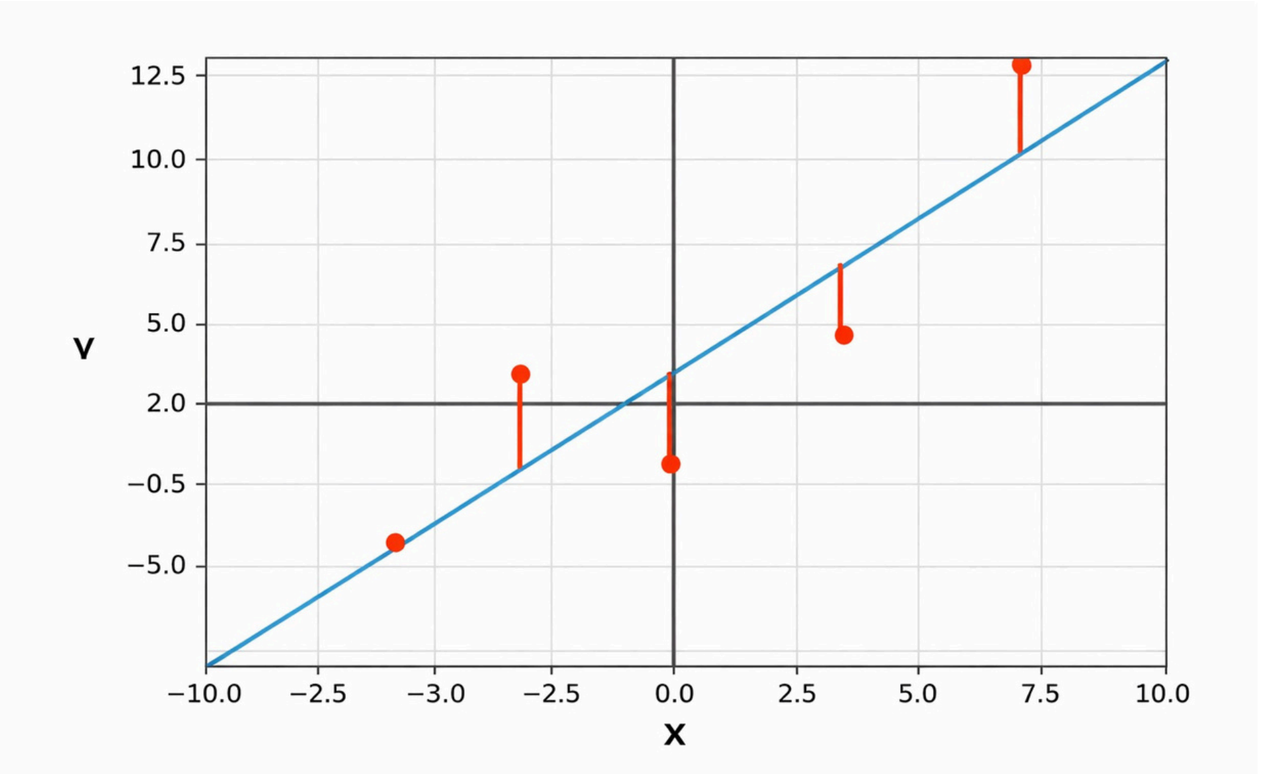


How?

Linear Regression

Optimization

To find the best line that minimizes the distances between predictions and data points.



Linear Regression

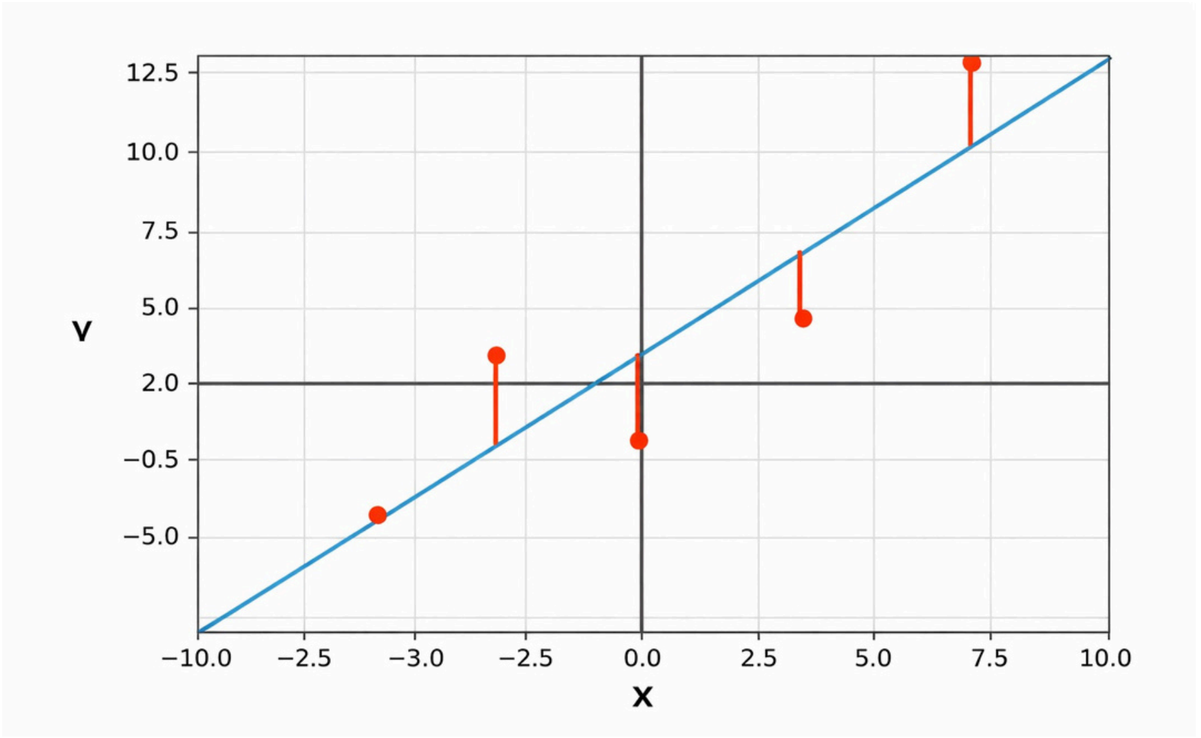
Loss Function

Error for a Single Data Point

$$y_i - \hat{y}_i$$

Squared Error

$$(y_i - \hat{y}_i)^2$$



Linear Regression

Loss Function

- Error for a Single Data Point

$$y_i - \hat{y}_i$$

- Squared Error

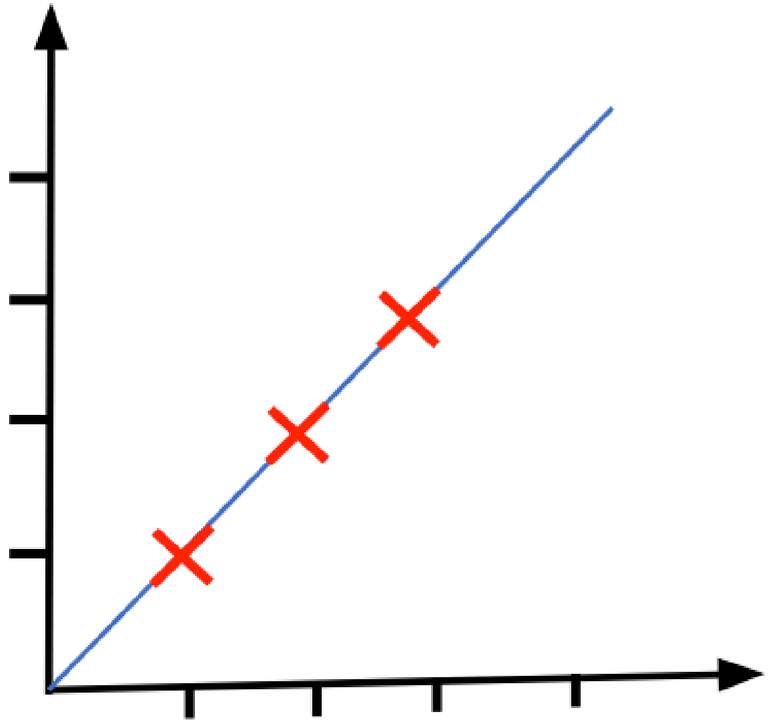
$$(y_i - \hat{y}_i)^2$$

- Error Over the Entire Dataset

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

Linear Regression

Model Hypothesis



- Some hypotheses fit the data better than others.

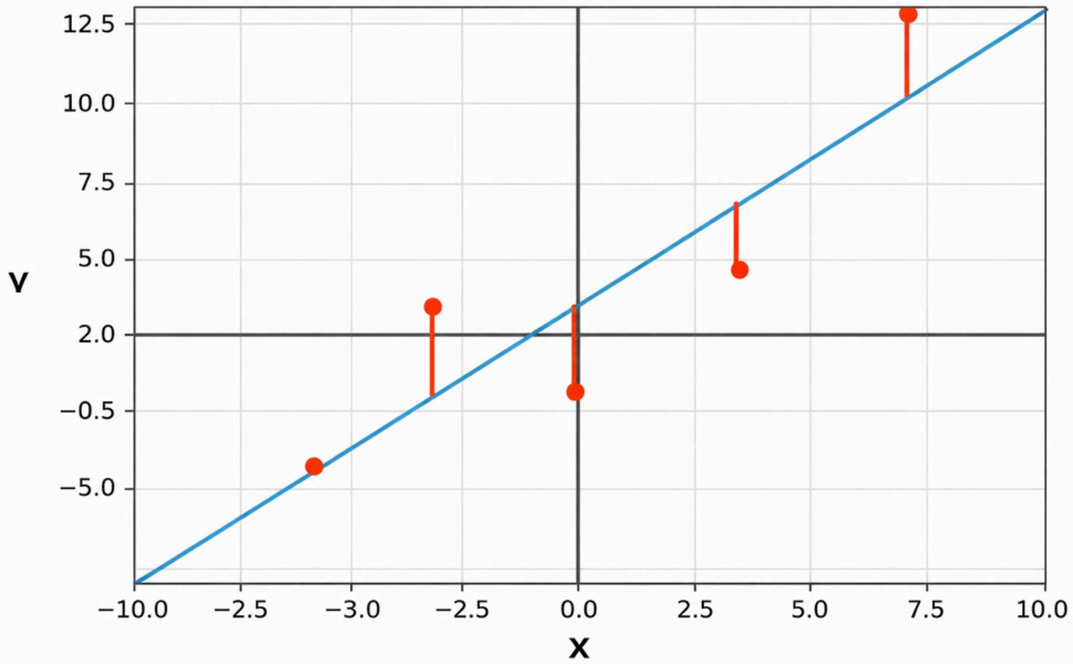
$$h(x) = mx$$

Linear Regression

Prediction Error

- Error is the vertical distance between a point and the line.

$$J(m) = \sum_{i=1}^{N} (y_i - mx_i)^2$$



Linear Regression

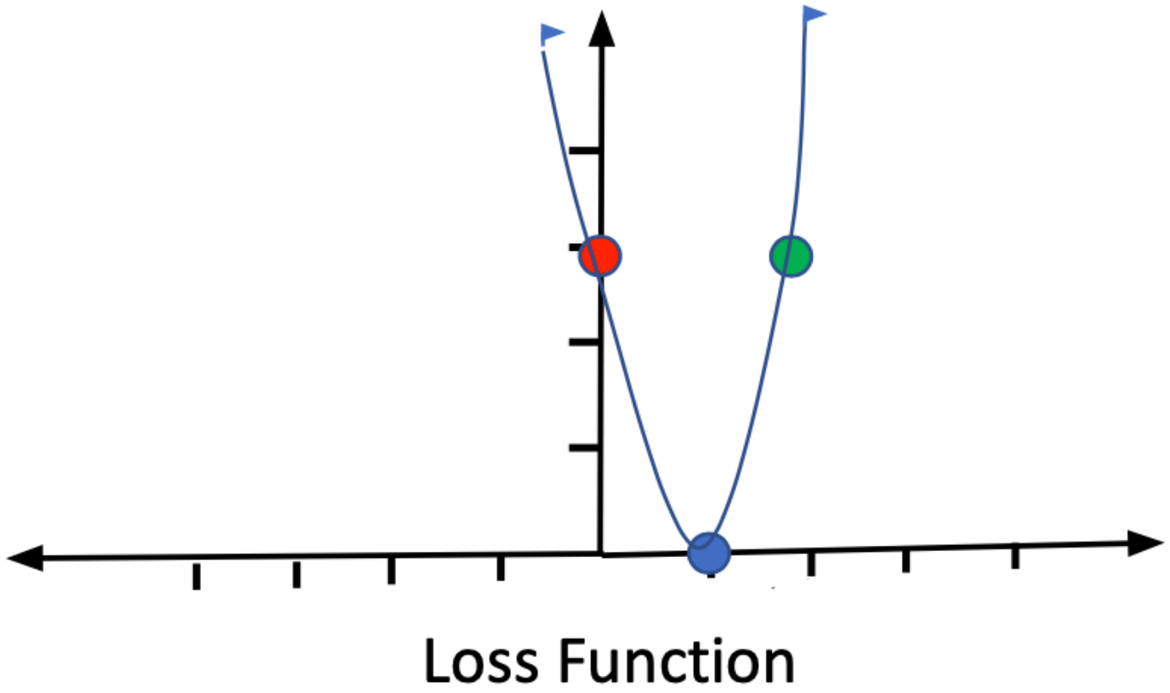
Prediction Error

$$J(m) = \sum_{i=1}^{3} (y_i - mx_i)^2$$

$$J(m=0) = 14$$

$$J(m=1) = 0$$

$$J(m=2) = 14$$



Linear Regression

- Optimization aims to find parameters that minimize a function.

$$J(m) = \sum_{i=1}^{N} (y_i - mx_i)^2$$

  - Exact (Closed-Form Solution)

$$f'(x) = 0$$

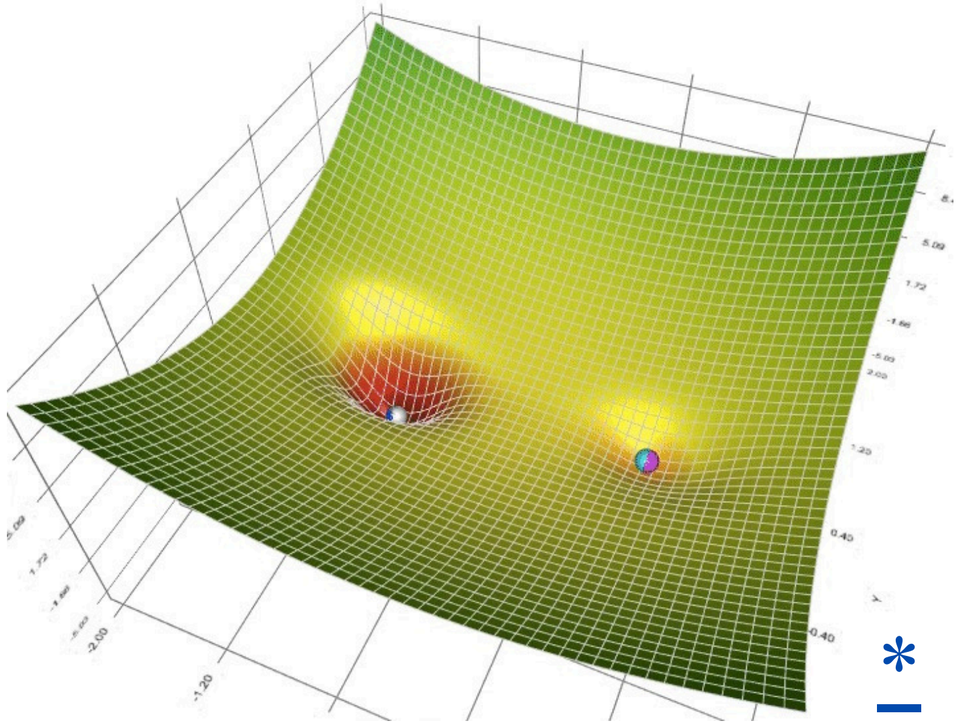
Works only for simple models and limited cases

Linear Regression

- Optimization aims to find parameters that minimize a function.

$$J(m) = \sum_{i=1}^{N} (y_i - mx_i)^2$$

- Approximation (Iterative Methods)
  - Start with an initial guess
  - Improve the solution step by step
  - Uses gradients to move toward the minimum

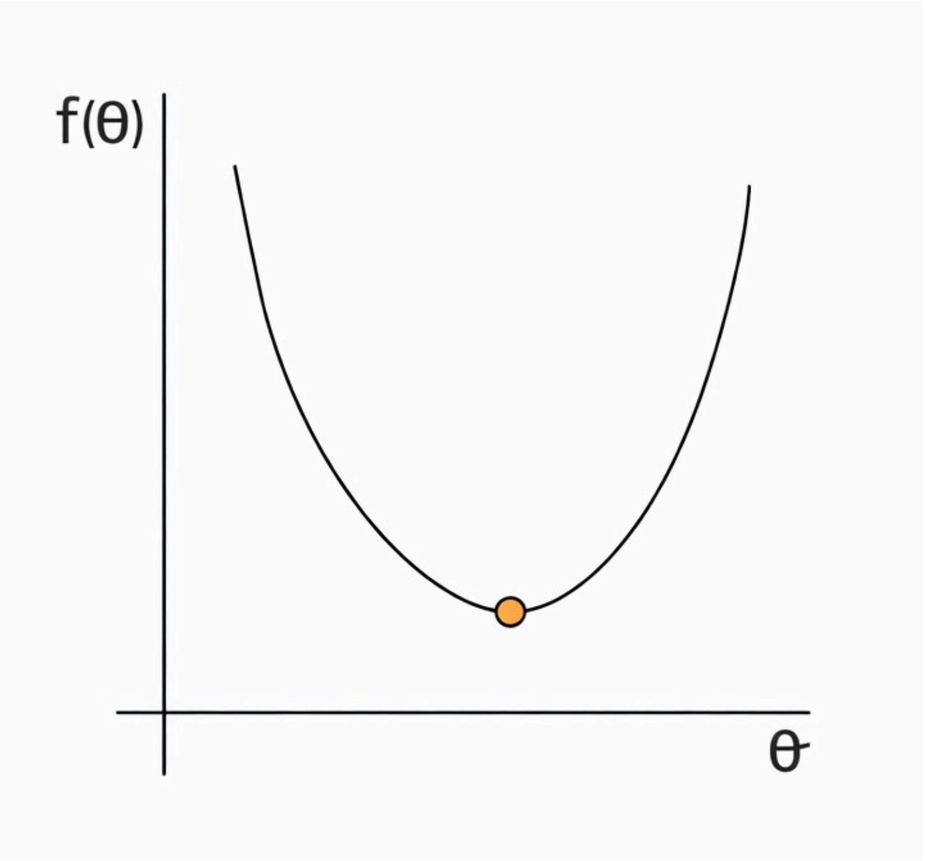


Linear Regression

- Optimization aims to find parameters that minimize a function.

$$Example: y = x^2$$

Minimum occurs at $x = 0$



- Exact (Closed-Form Solution)

$$x = 0$$

Linear Regression

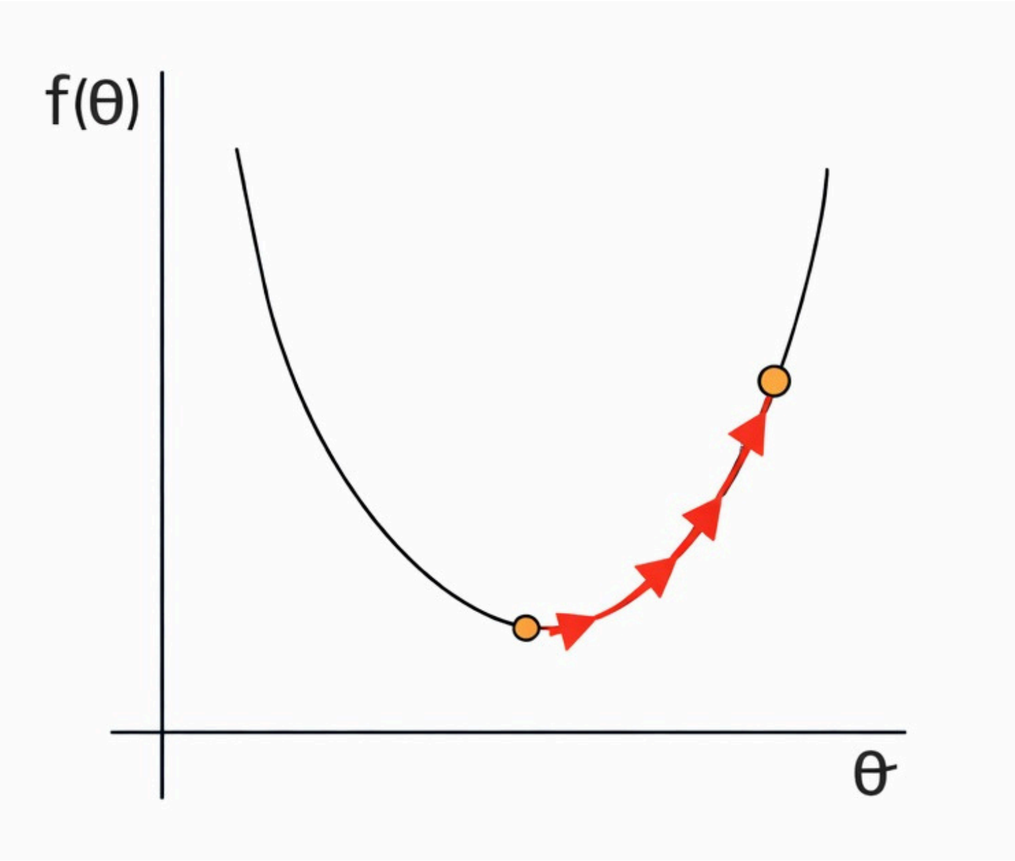
- Optimization aims to find parameters that minimize a function.

$$Example: y = x^2$$

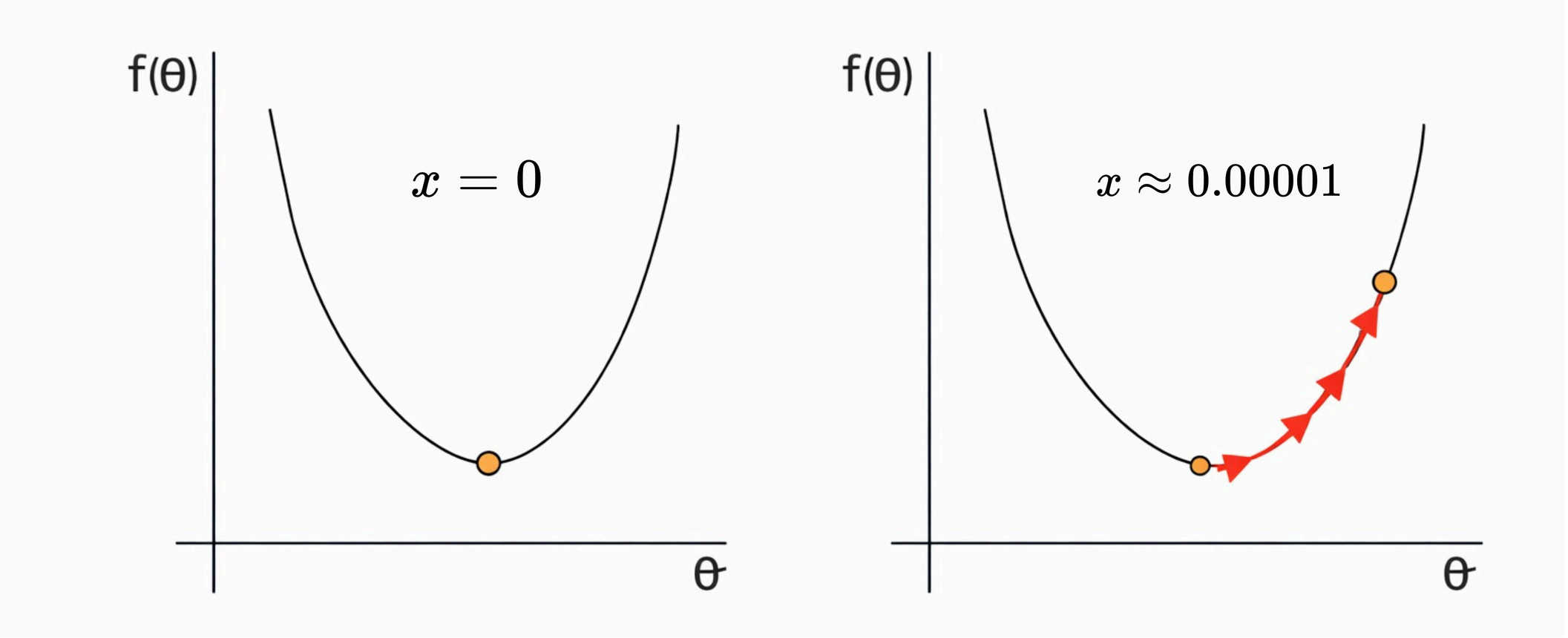
Minimum occurs at $x = 0$

- Approximation (Iterative Methods)

$$x \approx 0.00001$$



Linear Regression



Closed-form solutions are exact, iterative solutions are approximate.

Linear Regression

- Using closed-form: $\hat{y} = mx$
- Loss Function:

$$J(m) = \sum_{i=1}^{N} (y_i - mx_i)^2$$

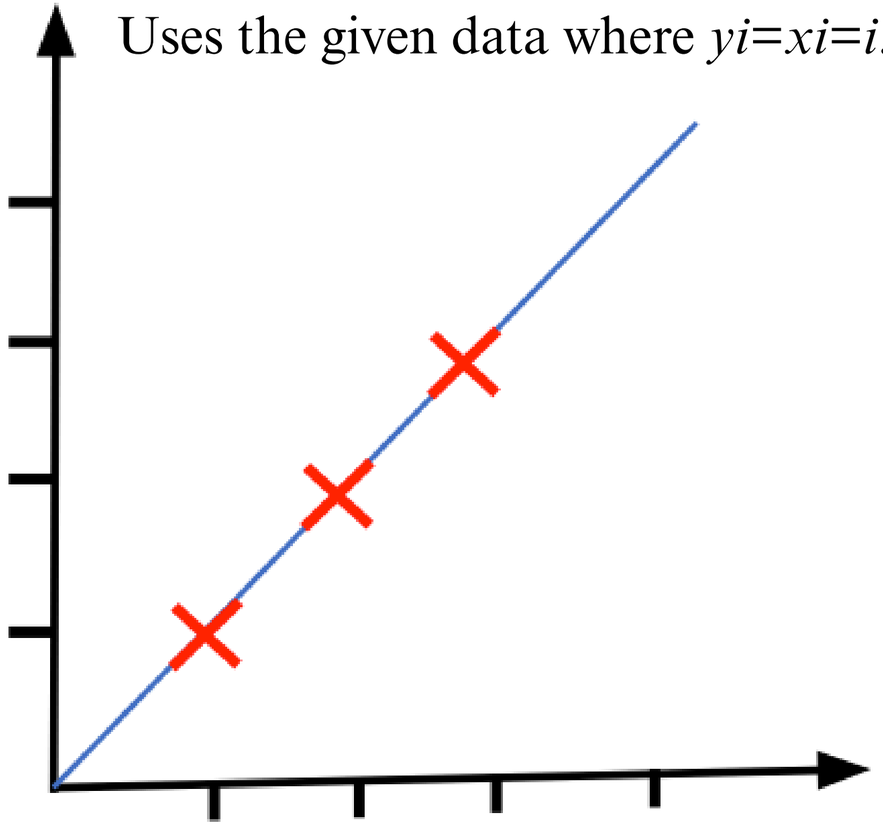
- Substitute the Data

$$J(m) = \sum_{i=1}^{3} (i - mi)^2$$

- Differentiate the Loss

$$\frac{dJ(m)}{dm} = \frac{d}{dm} \sum_{i=1}^{3} (i - mi)^2$$

Uses the given data where $yi=xi=i$.



Linear Regression

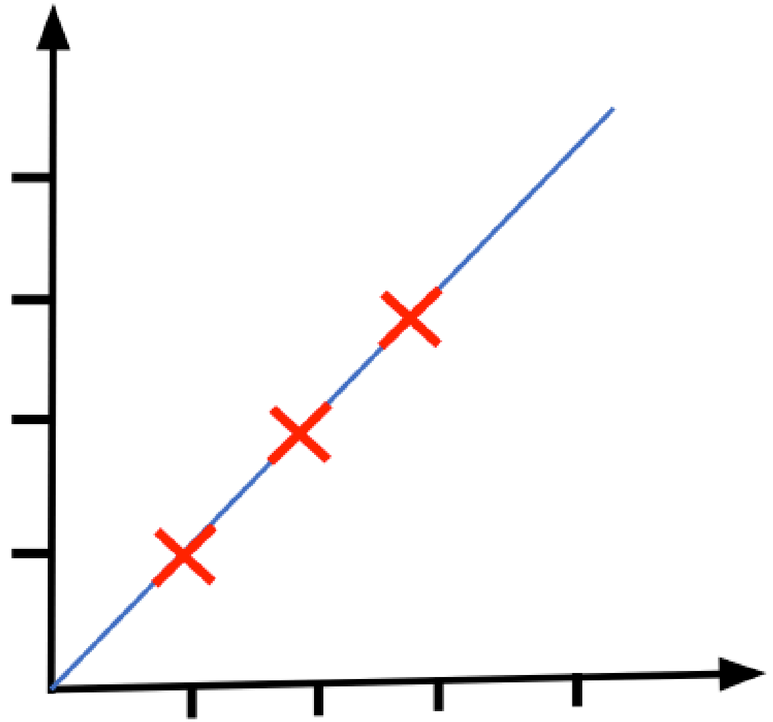
$$\frac{dJ(m)}{dm} = \sum_{i=1}^{3} \frac{d}{dm}(i - mi)^2$$

- Chain Rule

$$\frac{dJ(m)}{dm} = \sum_{i=1}^{3} -2i(i - mi)$$

$$\frac{dJ(m)}{dm} = -2\sum_{i=1}^{3} i^2 + 2m\sum_{i=1}^{3} i^2$$

Uses the given data where $yi=xi=i$.



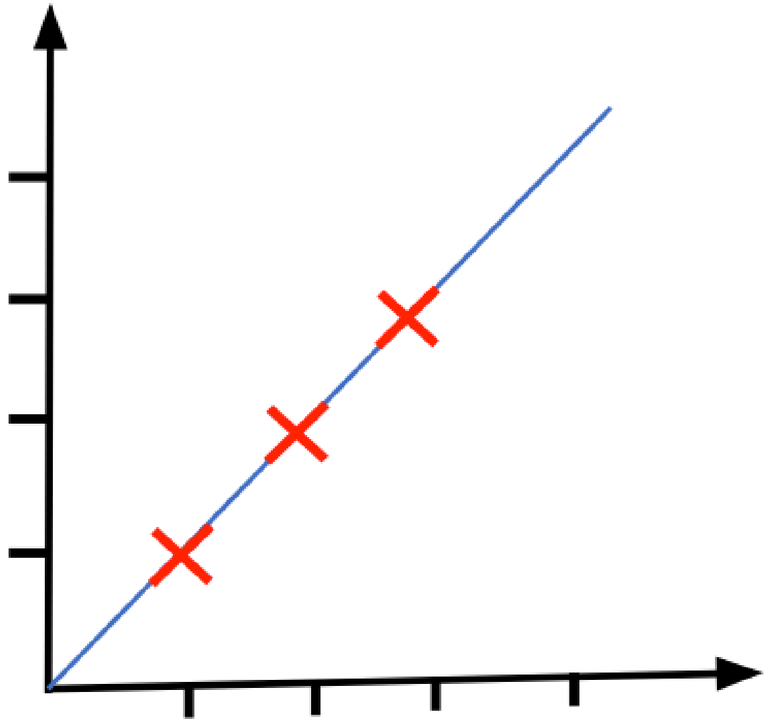
Linear Regression

$$-2\sum_{i=1}^{3} i^2 + 2m\sum_{i=1}^{3} i^2 = 0$$

- This value minimizes the loss.

$$m = 1$$

$$J(m = 1) = \min J(m)$$



- The model fits the data best when the slope is 1.

Linear Regression

Why Not Always Use Closed-Form?

- Closed-form solutions require simplifying assumptions
- They become complex with many features
- They are expensive for large datasets
- Some models have no closed-form solution

Linear Regression

- Gradient of the loss function
  - Gradient descent finds the minimum iteratively instead of solving equations directly.

- Model

$$\hat{y}_i = mx_i + b$$

- Loss Function

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

- The loss directly in terms of m and b.

$$J(m,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (mx_i + b)\right)^2$$

Linear Regression

- Gradient of the loss function
  - Gradient descent finds the minimum iteratively instead of solving equations directly.

- Model

$$\hat{y}_i = mx_i + b$$

- Loss Function

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

- The loss directly in terms of m and b.

$$J(m,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (mx_i + b)\right)^2$$

Linear Regression

- Gradient w.r.t. m

$$\frac{\partial J}{\partial m} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial m} \left(y_i - (mx_i + b)\right)^2$$

  - Chain Rule

$$\frac{\partial}{\partial m} \left(y_i - (mx_i + b)\right)^2 = 2 \left(y_i - (mx_i + b)\right) \cdot (-x_i)$$

  - Gradient for m is a weighted sum of errors by xi.

$$\frac{\partial J}{\partial m} = -\frac{2}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i\right) x_i$$

Linear Regression

- Gradient w.r.t. b

$$\frac{\partial J}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial b} \left(y_i - (m x_i + b) \right)^2$$

  - Chain Rule

$$\frac{\partial}{\partial b} \left(y_i - (m x_i + b) \right)^2 = 2 \left(y_i - (m x_i + b) \right) \cdot (-1)$$

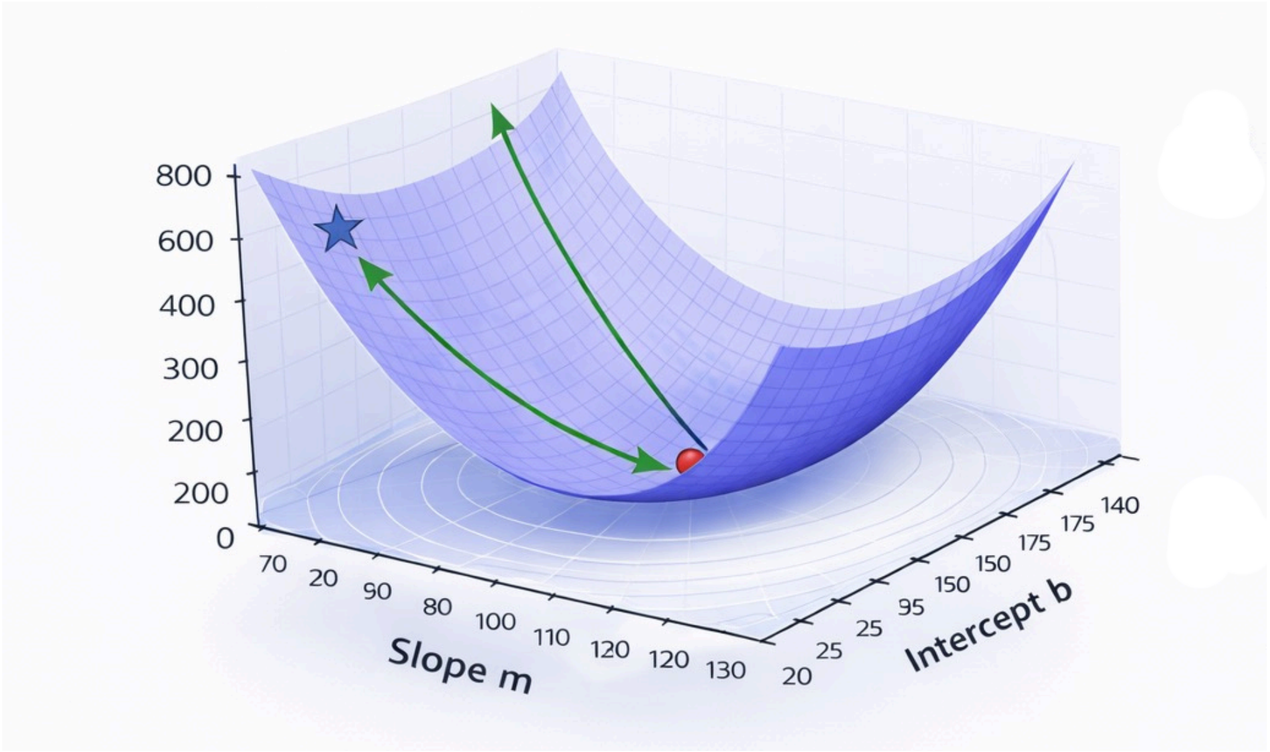
  - Gradient for b is proportional to the sum of errors.

$$\frac{\partial J}{\partial b} = -\frac{2}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i \right)$$

Linear Regression

- The gradient collects partial derivatives for all parameters.

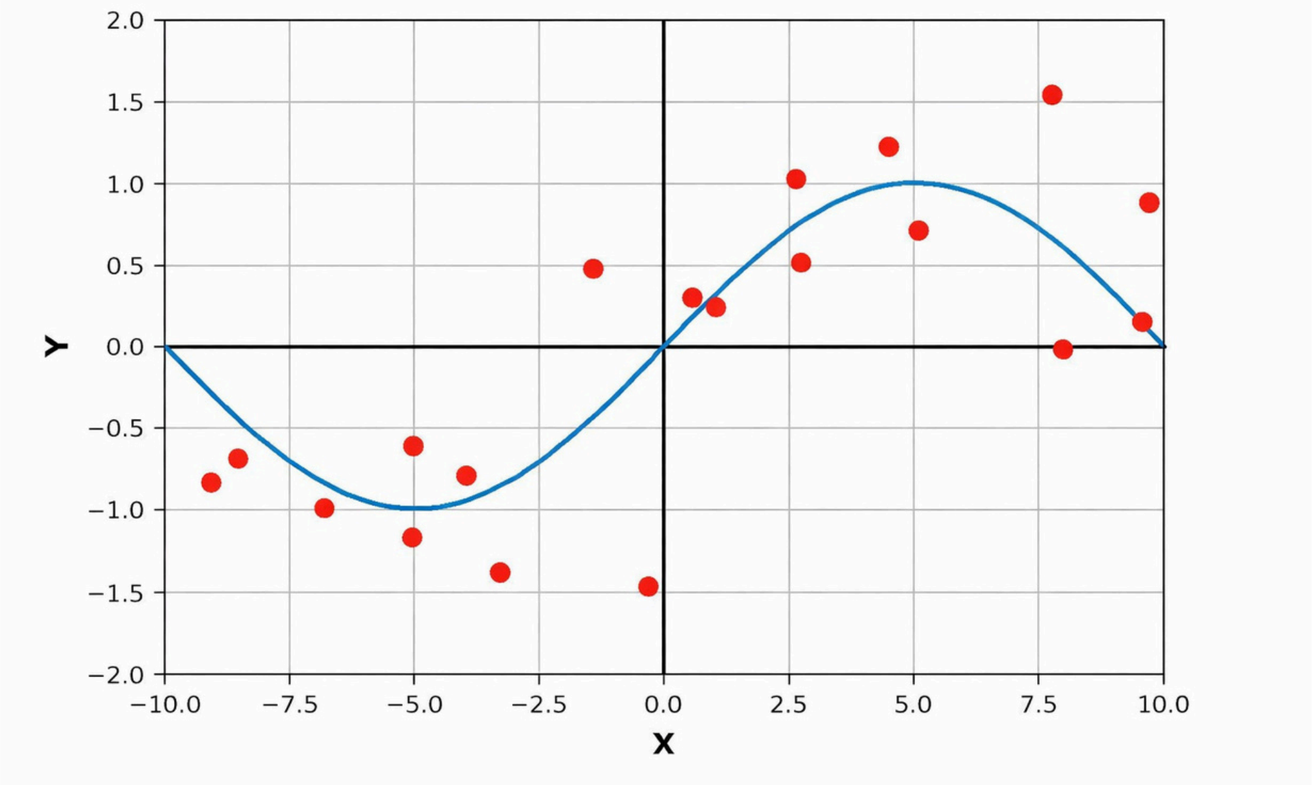
$$\nabla J(m,b) = \left[\frac{\partial J}{\partial m}, \frac{\partial J}{\partial b}\right]$$



Linear Regression

Fitting Non-Linear Data

What if y is a Non-Linear Function of x?



Linear Regression

- Original Feature Vector

$$x_i = (x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \ldots, x_i^{(m)})$$

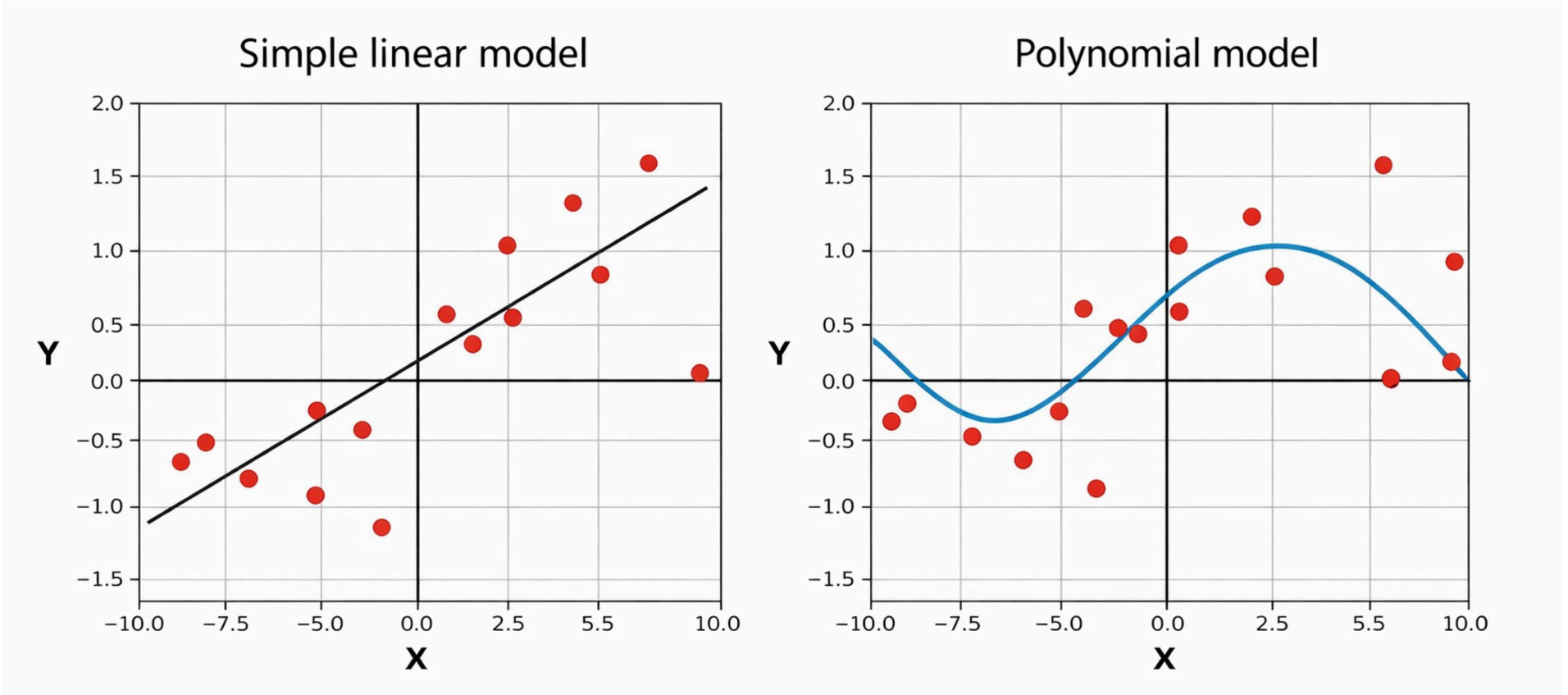
- Non-Linear Feature Mapping

$$\phi : \mathbb{R}^m \rightarrow \mathbb{R}^M$$

- Polynomial Feature Expansion (Degree k)

$$\phi(x_i) = \left[1, x_i^{(1)}, (x_i^{(1)})^2, \ldots, (x_i^{(1)})^k, \ldots, x_i^{(m)}, (x_i^{(m)})^2, \ldots, (x_i^{(m)})^k\right]$$

Linear Regression



$$h(x) = \theta_0 + \theta_1 x$$

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Linear Regression

- Original Feature Vector

$$x_i = \left(x_i^{(1)}, x_i^{(2)} \right)$$

- Engineered Feature (Area)

$$x_i^{(3)} = x_i^{(1)} \times x_i^{(2)}$$

- Extended Feature Vector

$$\tilde{x}_i = \left(x_i^{(1)}, x_i^{(2)}, x_i^{(3)} \right)$$

Linear Regression

- Original Feature Vector

$$x_i = \left(x_i^{(1)}, x_i^{(2)} \right)$$

- Engineered Feature (Area)

$$x_i^{(3)} = x_i^{(1)} \times x_i^{(2)}$$

- Extended Feature Vector

$$\tilde{x}_i = \left(x_i^{(1)}, x_i^{(2)}, x_i^{(3)} \right)$$

Linear Regression

- Gradient of the Loss Function
  - What We Have So Far (One Feature)

$$\frac{\partial J}{\partial m} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\,x_i$$

  - Now Assume We Have Many Features

$$\hat{y}_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_{100} x_{i,100}$$

  - We must compute one gradient per parameter

$$\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \cdots, \frac{\partial J}{\partial \theta_{100}}$$

Linear Regression

Gradient of the Loss Function

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i1}$$

$$\frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i2}$$

$$\vdots$$

$$\frac{\partial J}{\partial \theta_{100}} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i,100}$$

Linear Regression

Gradient of the Loss Function

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i1}$$

$$\frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i2}$$

$$\vdots$$

$$\frac{\partial J}{\partial \theta_{100}} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i,100}$$

Each gradient has the same structure

$$\left(y_i - \hat{y}_i\right)x_{ij}$$

Linear Regression

Gradient of the Loss Function

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i1}$$

$$\frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i2}$$

$$\vdots$$

$$\frac{\partial J}{\partial \theta_{100}} = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)x_{i,100}$$

Each gradient has the same structure

$$\left(y_i - \hat{y}_i\right)x_{ij}$$

Linear Regression

Gradient of the Loss Function

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\, x_{i1}$$

$$\frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\, x_{i2}$$

$$\vdots$$

$$\frac{\partial J}{\partial \theta_{100}} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\, x_{i,100}$$

Each gradient has the same structure

$$(y_i - \hat{y}_i)\, x_{ij}$$

This is equivalent to a dot product between:

- an error vector
- a feature vector

Linear Regression

Gradient of the Loss Function

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\, x_{i1}$$

$$\frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\, x_{i2}$$

$$\vdots$$

$$\frac{\partial J}{\partial \theta_{100}} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\, x_{i,100}$$

Each gradient has the same structure

$$(y_i - \hat{y}_i)\, x_{ij}$$

This is equivalent to a dot product between:

- an error vector
- a feature vector

Can we compute all gradients at once, instead of one by one?

Linear Regression

Vectorization

- Now, the equation is aggregated across all N data points.

$$
\begin{aligned}
y_1 &= \hat{y}_1 + \epsilon_1 \\
y_2 &= \hat{y}_2 + \epsilon_2 \\
&\vdots \quad \vdots \quad \vdots \\
y_N &= \hat{y}_N + \epsilon_N
\end{aligned}
$$

Linear Regression

- Vectorization
  - Replacing the values of $\hat{y}$, we obtain:

$$
\begin{aligned}
y_1 &= w_0 + w_1 x_1^{(1)} + w_2 x_1^{(2)} + \cdots + w_M x_1^{(M)} + \epsilon_1 \\
y_2 &= w_0 + w_1 x_2^{(1)} + w_2 x_2^{(2)} + \cdots + w_M x_2^{(M)} + \epsilon_2 \\
&\vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\
y_N &= w_0 + w_1 x_N^{(1)} + w_2 x_N^{(2)} + \cdots + w_M x_N^{(M)} + \epsilon_N
\end{aligned}
$$

Linear Regression

- Vectorization
  - Collecting the equations in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(M)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(M)} \\ 1 & x_3^{(1)} & x_3^{(2)} & \cdots & x_3^{(M)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(M)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Linear Regression

- Vectorization
  - Note that each row of the matrix on the right represents a data sample.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \cdots & \mathbf{x}_3 & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_N & \cdots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Linear Regression

- Vectorization
  - D represents the entire dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \cdots & x_1 & \cdots \\ \cdots & x_2 & \cdots \\ \cdots & x_3 & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & x_N & \cdots \end{bmatrix} \boldsymbol{\theta} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Linear Regression

Vectorization

- The Mean Squared Error measures the average squared prediction error.

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

- Using the prediction error $\epsilon_i = y_i - \hat{y}_i$.

$$J = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\epsilon_i^2$$

- The cost function can be expressed compactly using vector notation.

$$J = \frac{1}{N}\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$$

Linear Regression

- Optimization
  - The goal is to find model parameters that minimize the squared error.

$$\min_{\boldsymbol{\theta}} \ \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

$$\min_{\boldsymbol{\theta}} \ \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = \min_{\boldsymbol{\theta}} \left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\right)^T \left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\right)$$

  - The cost function can be expressed compactly using vector notation.

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = \frac{dJ}{d\boldsymbol{\epsilon}} \nabla_{\boldsymbol{\theta}} \boldsymbol{\epsilon}$$

Linear Regression

- Optimization
  - The gradient of the cost function with respect to the parameters is:

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = -2\,\mathbf{X}^T\left(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\right)$$

  - Setting the gradient equal to zero yields the optimal parameters:

$$\boldsymbol{\theta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Closed-form solution for Linear Regression

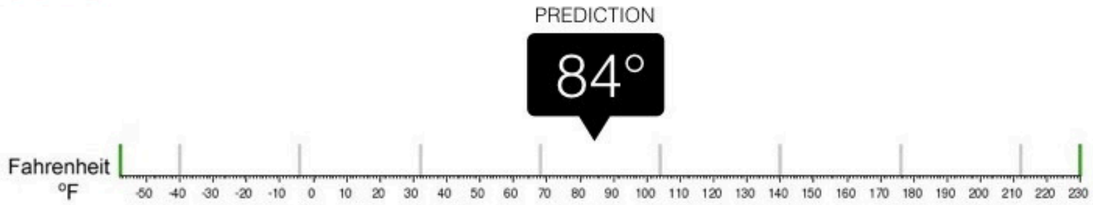
Machine Learning

Classification

Classification

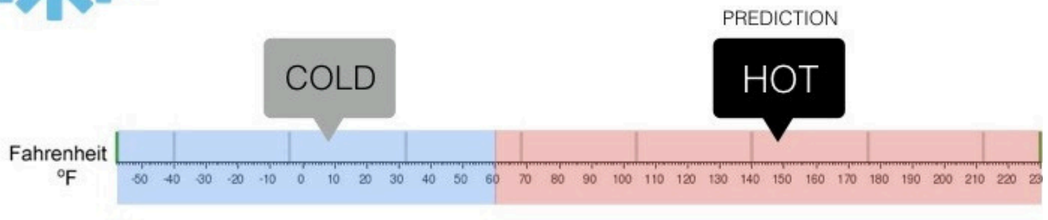
Regression

What is the temperature going to be tomorrow?



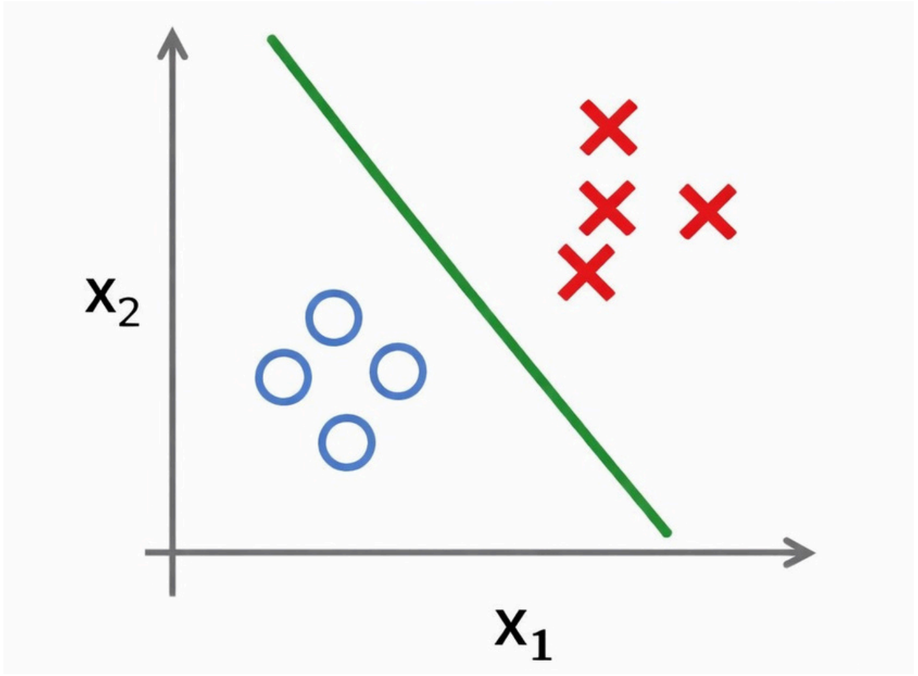
Classification

Will it be Cold or Hot tomorrow?

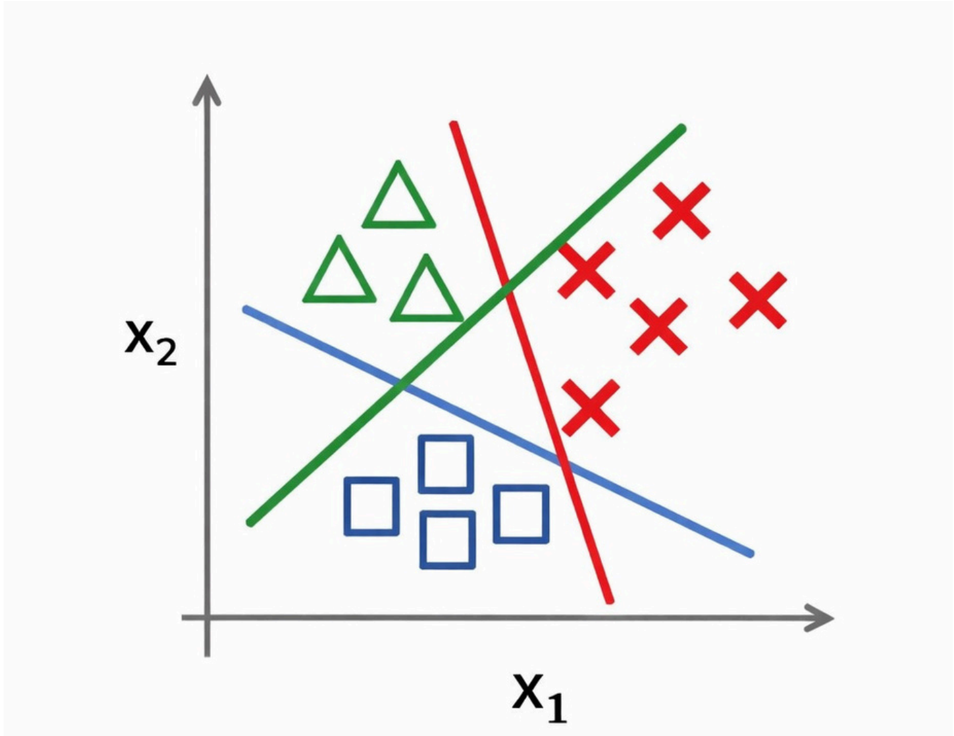


Classification

- Classification Types



- Binary Classification



- Multiclass Classification

Classification

- Classification models often rely on the same linear structure used in regression.
  - The model computes a linear score from the input features.

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

  - The weight vector controls the orientation of the decision boundary.

$$\mathbf{w} = [w_0, w_1, \ldots, w_m]^T$$

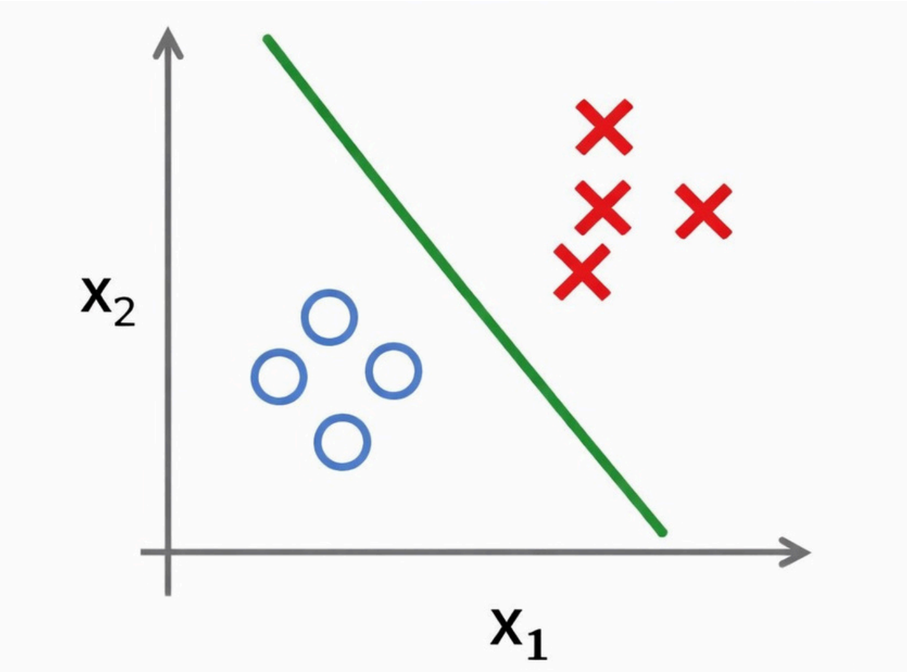
  - The constant 1 allows the model to learn a bias term.

$$\mathbf{x} = [1, x^1, \ldots, x^m]^T$$

Classification

- How can continuous predictions be mapped to binary labels?

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \quad \mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \{0, 1\}$$



Classification

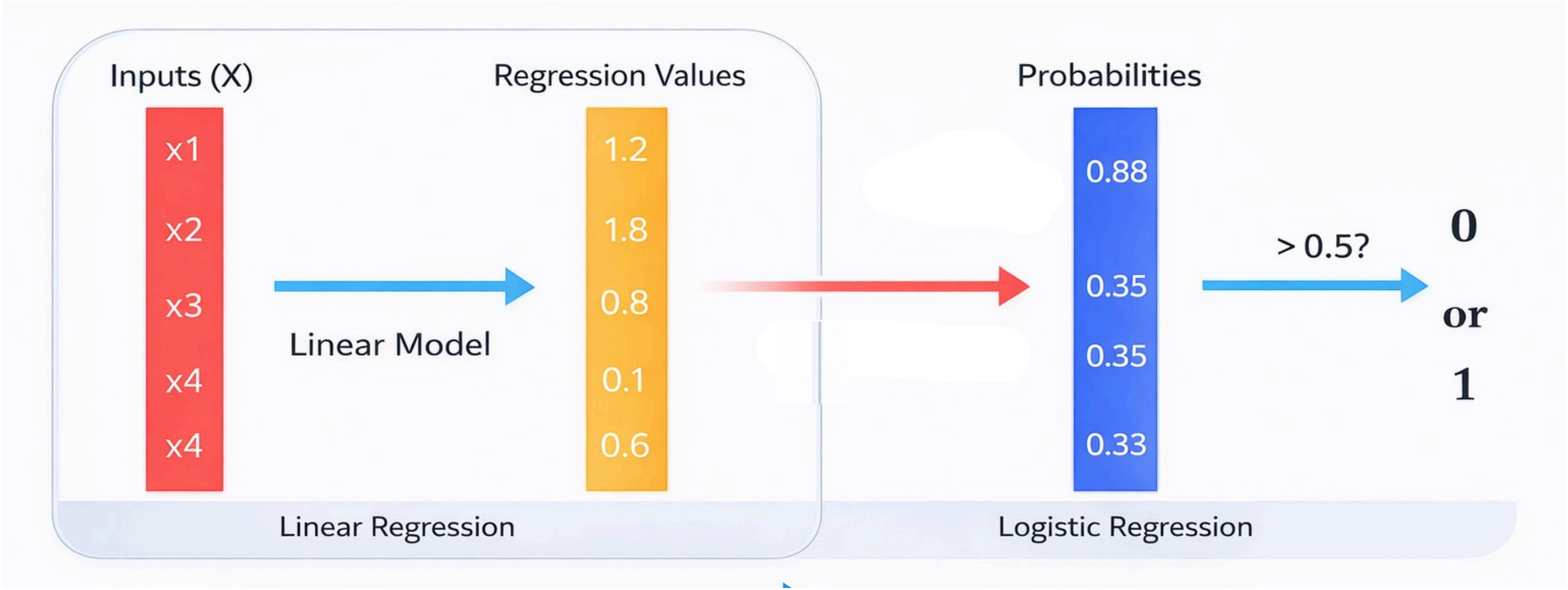
- Linear Classifier
  - A linear classifier builds on linear regression with two essential modifications.

  - Linear scores are passed through a nonlinear function to produce probabilities.

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x})$$

Logistic Regression

Logistic Regression



Logistic Regression

Logistic Regression

We want to transform raw model outputs into probabilities.

- Map any real value to [0,1]

Interpretable as probabilities

- Smooth and differentiable

Enables gradient-based optimization

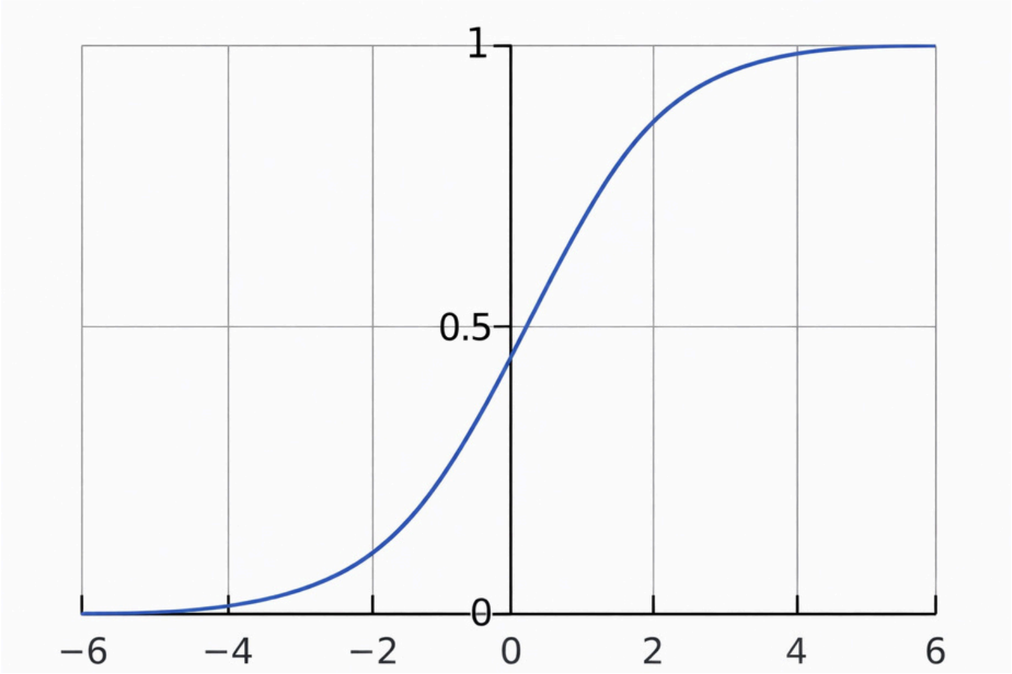
Logistic Regression

- Sigmoid Function
  - Sigmoid maps real values to probabilities

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  - Suitable for gradient descent

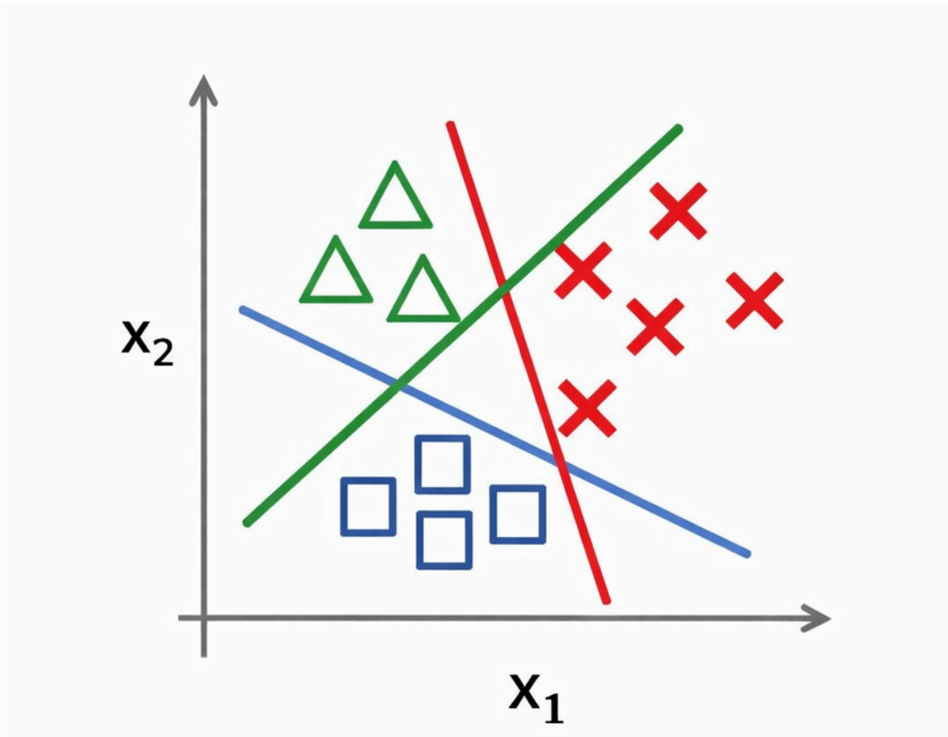
$$\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right)$$



$$\lim_{z \to -\infty} \sigma(z) = 0 \qquad \lim_{z \to +\infty} \sigma(z) = 1$$

Logistic Regression

- But what if we have multiclass problem?



Logistic Regression

- Softmax Function
  - Converts raw class scores into a probability distribution over K classes.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

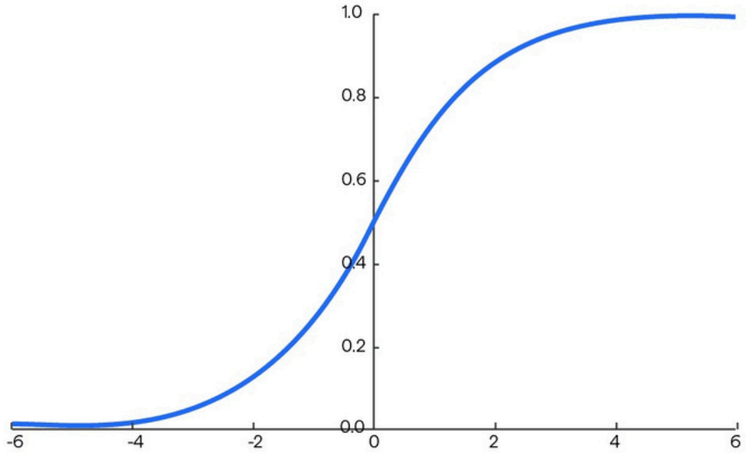
  - Each class receives a probability, and all probabilities sum to 1.

Logistic Regression

Softmax Function

Numerator: e^{z_i} (exponential of the score for the ith class)

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Denominator: $\sum_{j=1}^{n} e^{z_j}$ (sum of exponentials over all classes)

Logistic Regression

Softmax Example

$$z = [2.0,\ 1.0,\ 0.1] \quad (\text{cat, dog, deer})$$

Softmax Example

$$\text{Softmax}(z) = \left[\frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}},\ \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}},\ \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}} \right]$$

$$\text{Softmax}(z) \approx [0.66,\ 0.24,\ 0.10]$$

Logistic Regression

- Classification Loss

We need a loss function that compares predicted probabilities with true labels.

Probabilistic	Confidence-Sensitive	Differentiable
Works on probabilities, not raw scores.	Penalizes confident wrong predictions heavily	Smooth and optimizable using gradient descent.

Logistic Regression

- Binary Cross-Entropy

$$\ell(y, p) = -\Big[y \log(p) + (1 - y) \log(1 - p)\Big]$$

- Penalizes the model based on the probability it assigns to the true class.

$$\ell(y, p) = \begin{cases} -\log(p), & \text{if } y = 1 \\ -\log(1 - p), & \text{if } y = 0 \end{cases}$$

Logistic Regression

- One-sample loss

$$\ell(y, p) = -\Big[y \log(p) + (1 - y) \log(1 - p)\Big]$$

- Average over samples

$$\mathcal{L}(Y, P) = -\frac{1}{N} \sum_{i=1}^{N} \Big[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\Big]$$

Y: Represents the true labels.

P: Represents the predicted probabilities for the positive class.

$$y_i \in \{0, 1\}, \quad p_i \in (0, 1)$$

Logistic Regression

- Optimization

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log\left(\sigma(\mathbf{w}^T\mathbf{x}_i)\right) + (1-y_i)\log\left(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\right)\right]$$

How do we minimize this loss with respect to w?

Logistic Regression

- Optimization

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log\left(\sigma(\mathbf{w}^T\mathbf{x}_i)\right) + (1-y_i)\log\left(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\right)\right]$$

How do we minimize this loss with respect to w?

$$\nabla_{\mathbf{w}} J(\mathbf{w}) \neq 0 \quad \text{(no closed-form solution)}$$

We must use iterative optimization methods

Logistic Regression

- Optimization
  - The sigmoid makes the optimization non-linear.

$$\mathbf{w}^{k+1} = \mathbf{w}^{k} - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^{k})$$

Sigmoid converts score to probability: $p = \sigma(\mathbf{w}^T \mathbf{x})$

Binary Cross-Entropy measures prediction error

Gradient Descent updates parameters iteratively

Logistic Regression

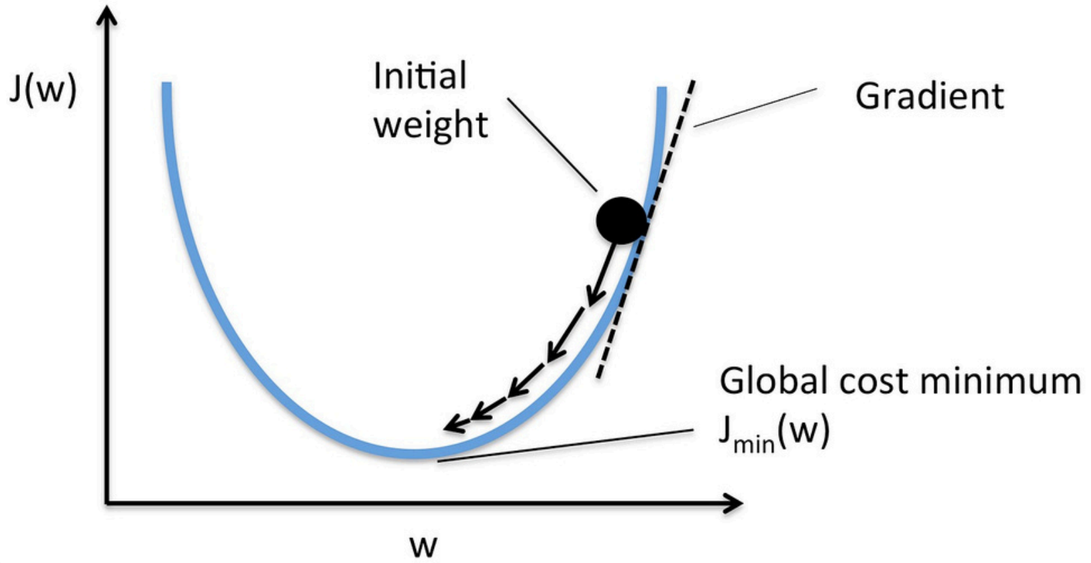
- Optimization
  - The sigmoid makes the optimization non-linear.

$$\mathbf{w}^{k+1} = \mathbf{w}^{k} - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^{k})$$

Sigmoid converts score to probability: $p = \sigma(\mathbf{w}^T \mathbf{x})$

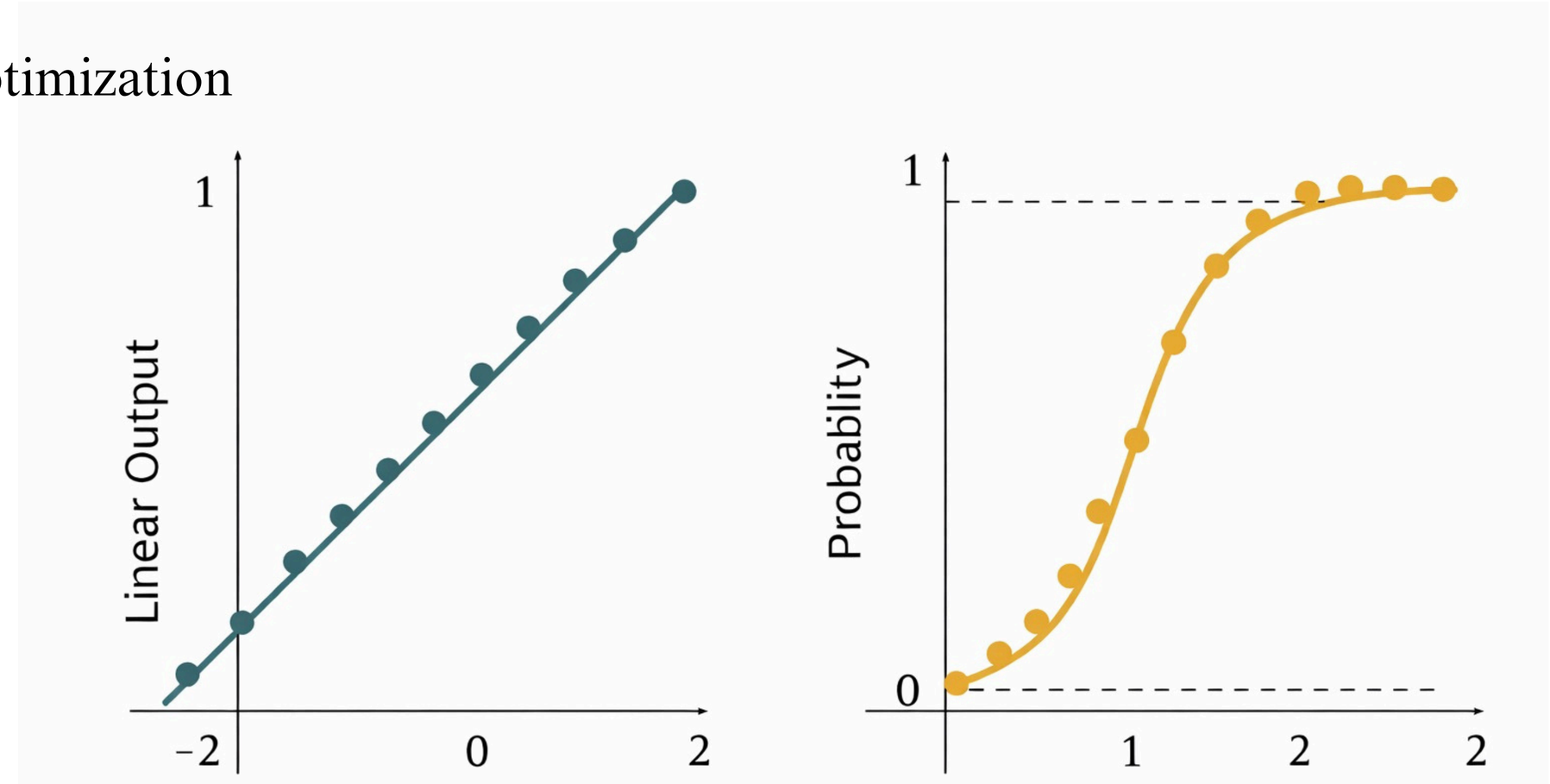
Binary Cross-Entropy measures prediction error

Gradient Descent updates parameters iteratively



Logistic Regression

- Optimization



Linear Regression

Logistic Regression

Machine Learning

Classification Metrics

Classification Metrics

- Accuracy
  - Accuracy measures how many predictions the model gets exactly right.

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\hat{y}_i = y_i)$$

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}}$$

Classification Metrics

- Accuracy
  - Accuracy measures how many predictions the model gets exactly right.

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\hat{y}_i = y_i)$$

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}}$$

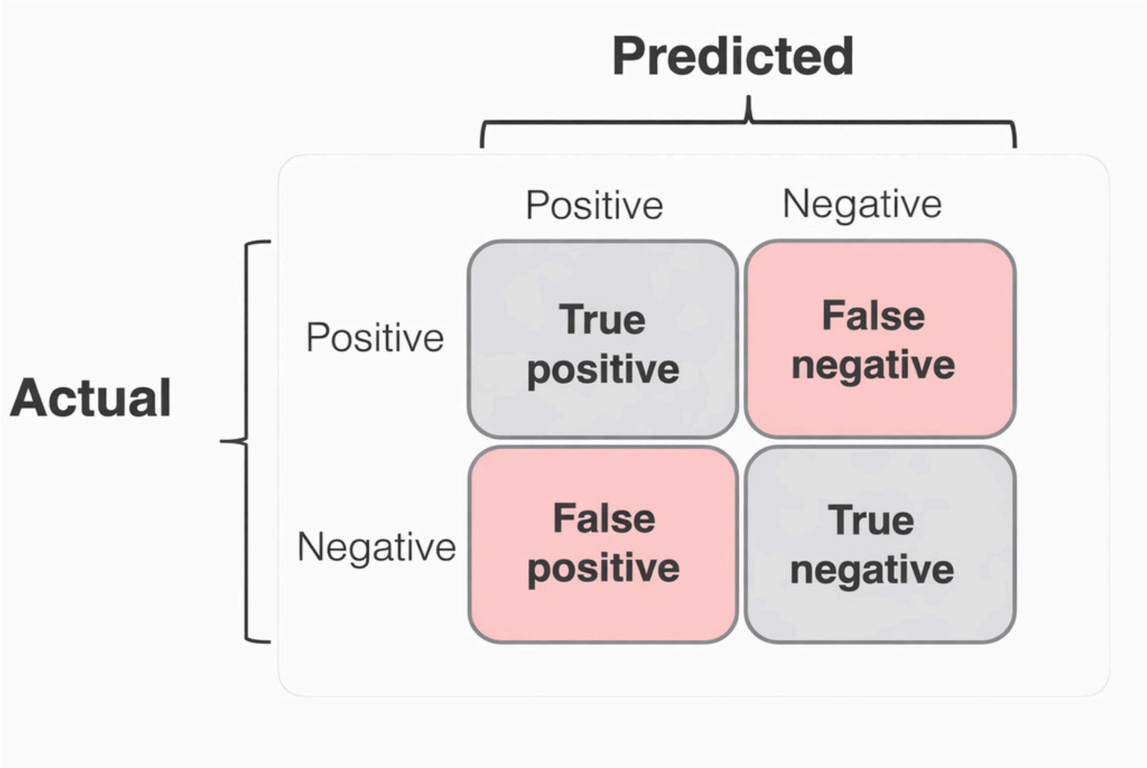
Why don't we optimize accuracy directly during training?

Classification Metrics

- Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TP : True Positives
TN : True Negatives
FP : False Positives
FN : False Negatives



Classification Metrics

- Accuracy

Assume we are working with the following dataset:

$$\begin{aligned} \text{Total Emails} &= 100 \\ \text{Spam Emails} &= 5 \\ \text{Not Spam Emails} &= 95 \end{aligned}$$

Classification Metrics

- Accuracy

Assume we are working with the following dataset:

$$\begin{aligned} \text{Total Emails} &= 100 \\ \text{Spam Emails} &= 5 \\ \text{Not Spam Emails} &= 95 \end{aligned}$$

The model predicts "Not Spam" for every email.

Classification Metrics

- Accuracy

Assume we are working with the following dataset:

$$\begin{aligned} \text{Total Emails} &= 100 \\ \text{Spam Emails} &= 5 \\ \text{Not Spam Emails} &= 95 \end{aligned}$$

The model predicts "Not Spam" for every email.

	Predicted Spam	Predicted Not Spam
Actual Spam	0	5
Actual Not Spam	0	95

Classification Metrics

Accuracy

Assume we are working with the following dataset:

The model predicts "Not Spam" for every email.

	Predicted Spam	Predicted Not Spam
Actual Spam	0	5
Actual Not Spam	0	95

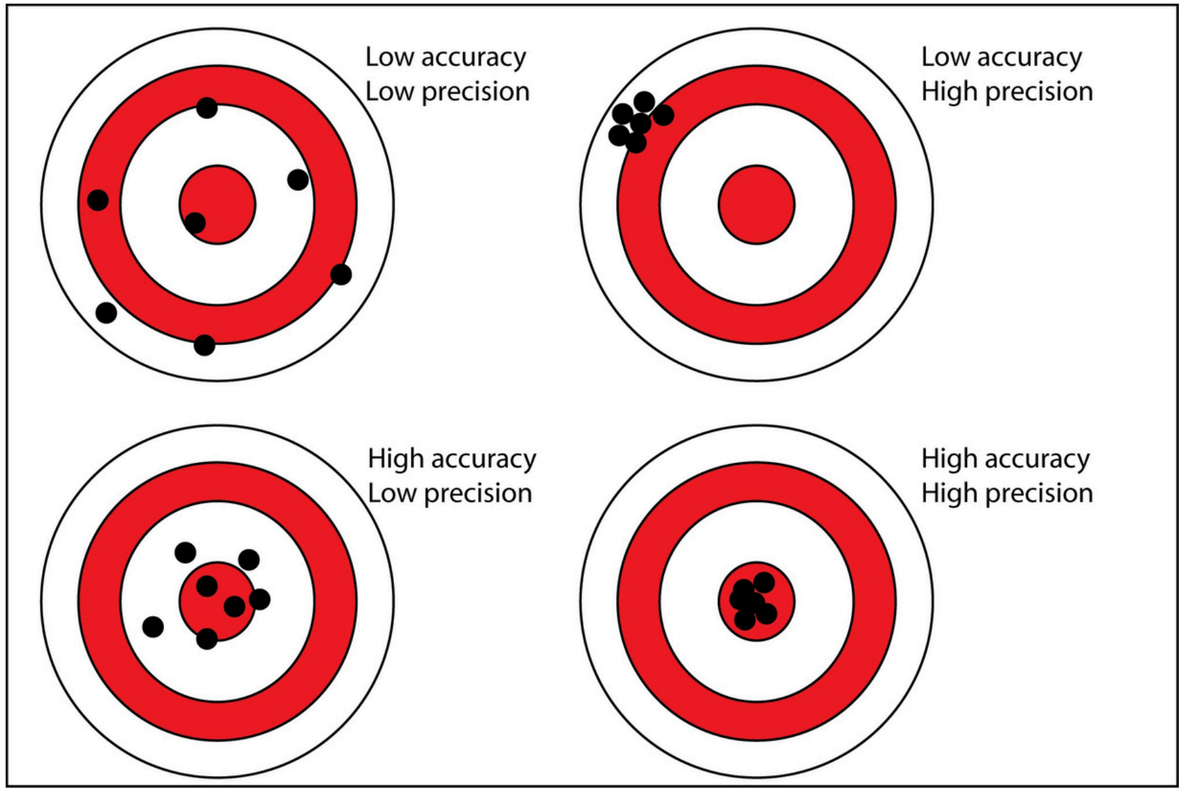
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{0 + 95}{100} = 95\%$$

95% accuracy, yet the model fails to detect any spam emails.

Classification Metrics

- Precision
  - Among all the samples predicted as positive, how many are actually positive?

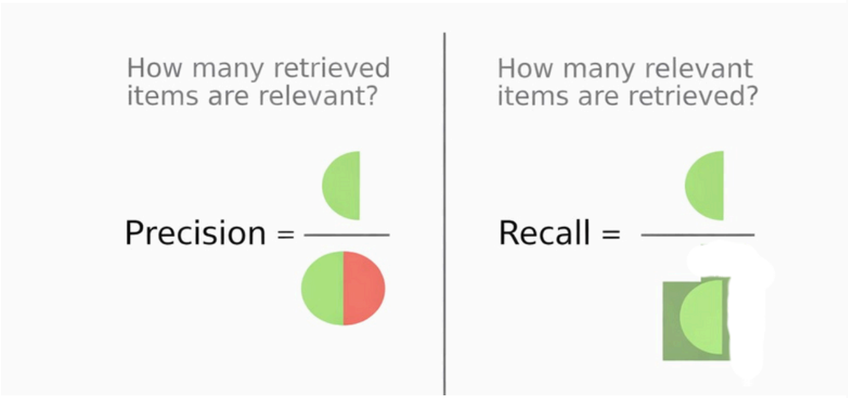
$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

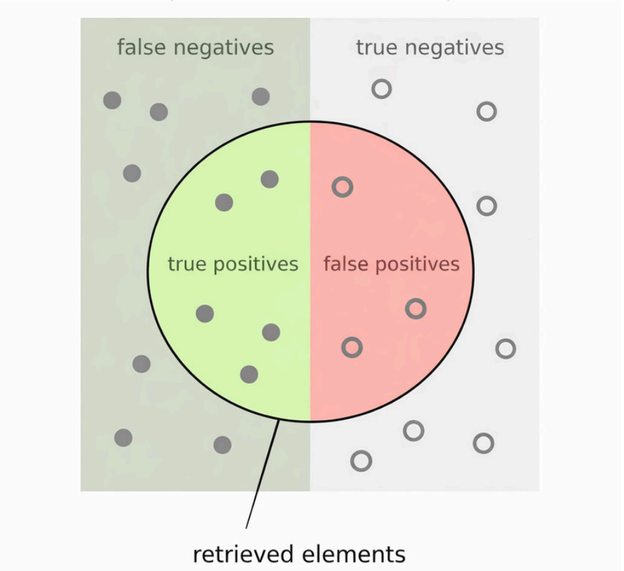


Classification Metrics

- Recall
  - Among all the actual positive samples, how many did the model correctly identify?

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

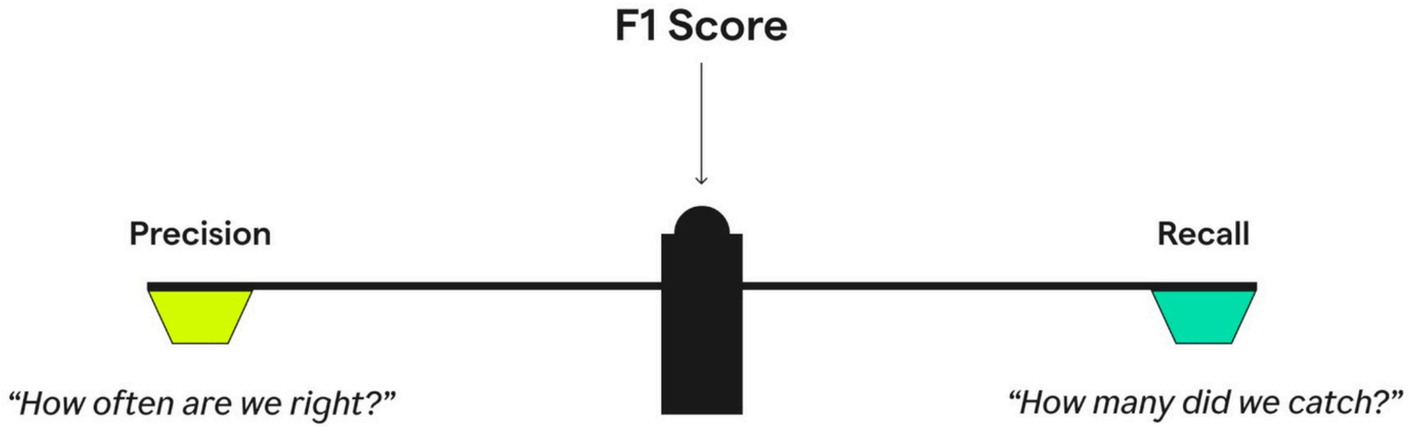




Classification Metrics

- F1 score
  - The F1 score is the harmonic mean of precision and recall.

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$



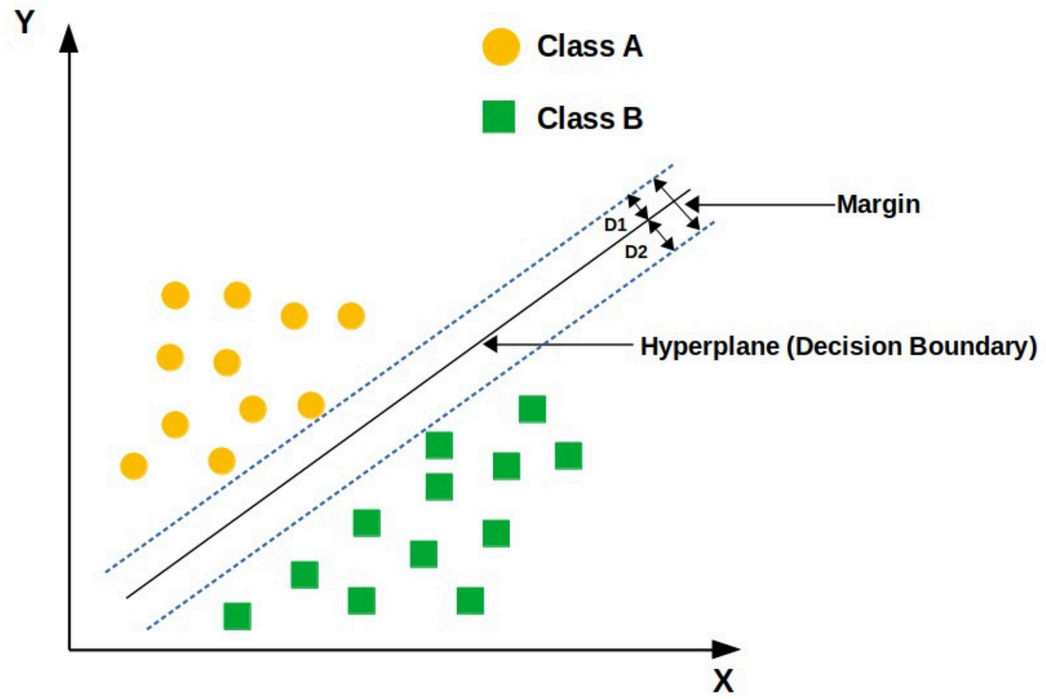
Machine Learning

Classification Models

Classification Metrics

- Support Vector Machine (SVM)
  - SVM finds the decision boundary that maximizes the margin between classes.

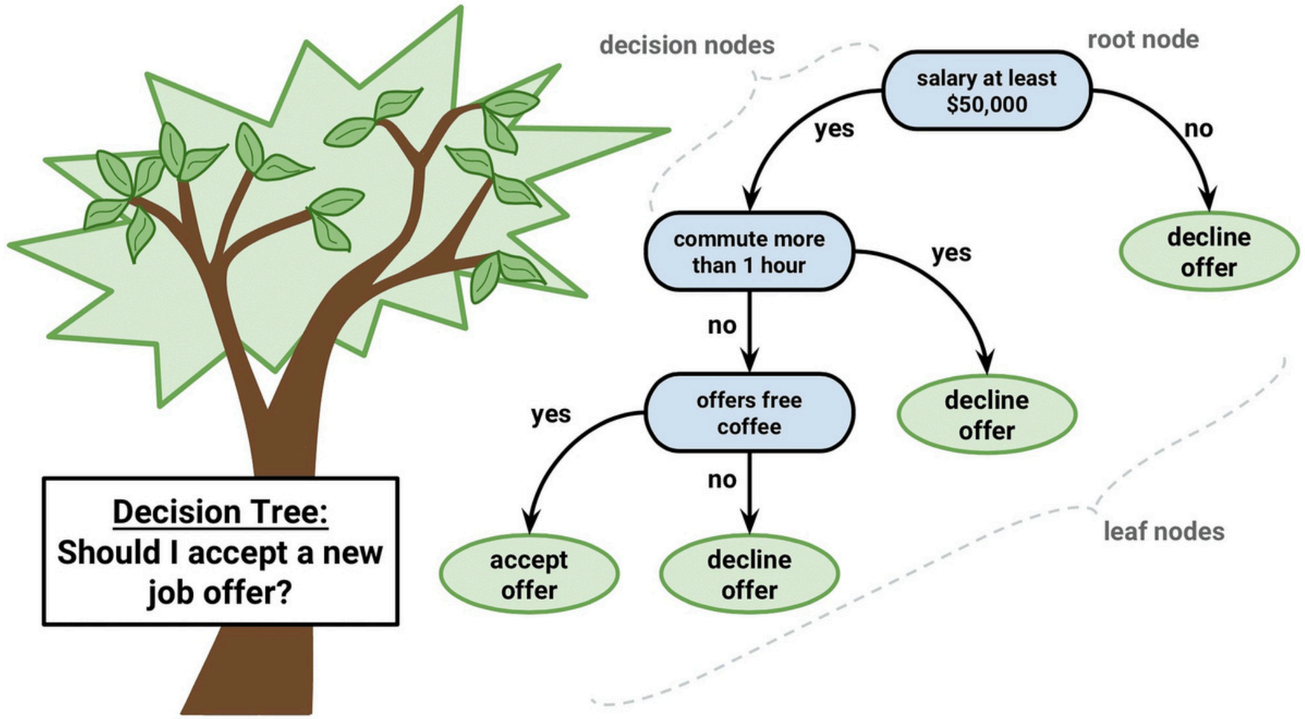
- **Hyperplane:** the decision boundary separating classes
- **Margin:** distance between the boundary and the closest data points
- **Support Vectors:** data points that define the margin



Classification Metrics

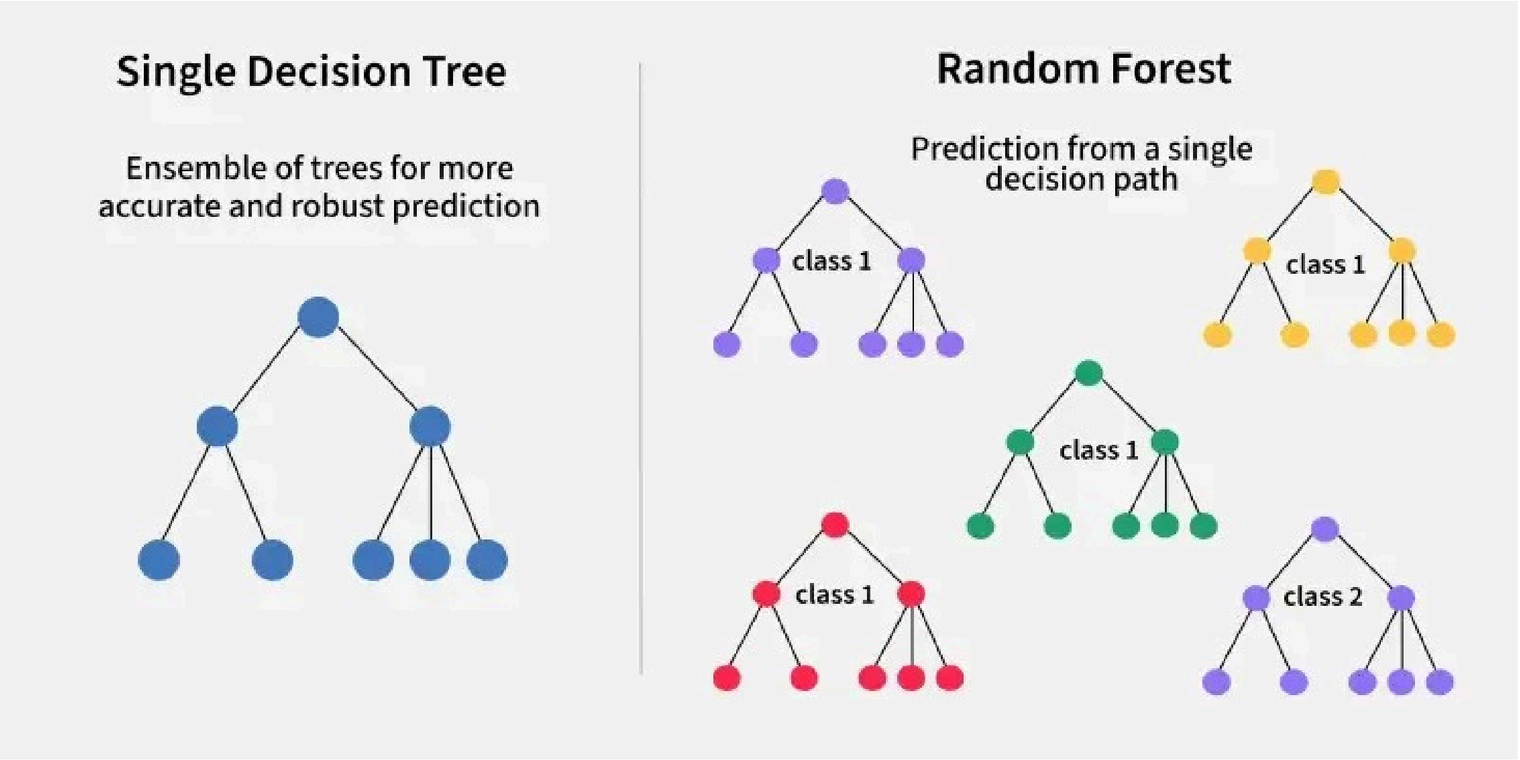
- Decision Trees
  - A decision tree classifies data by asking a sequence of simple if–else questions.

- Each internal node represents a feature test
- Each branch represents an outcome of the test
- Each leaf node represents a class label



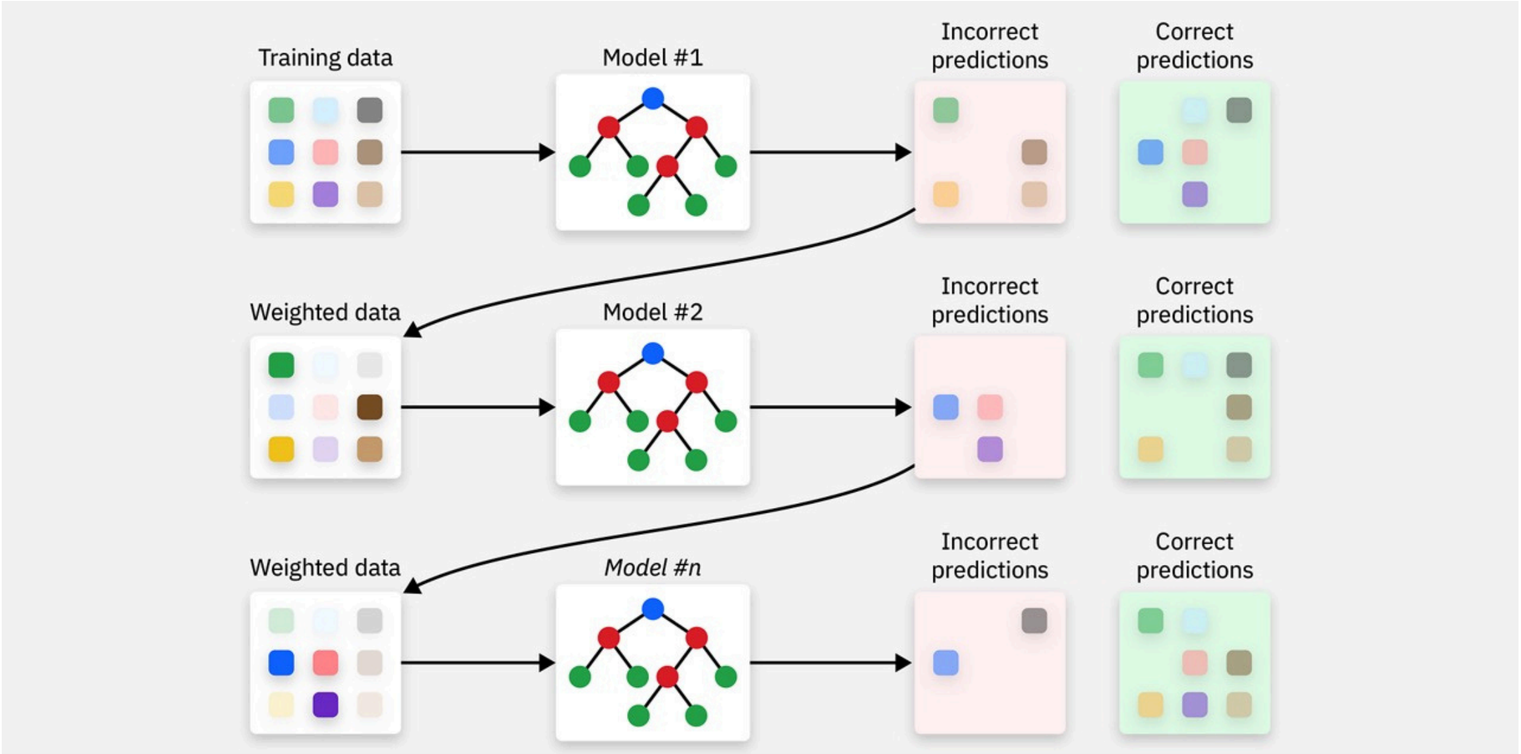
Classification Metrics

- Random Forest
  - Random Forest combines many decision trees to produce a more accurate and robust classifier.



Classification Metrics

- Gradient Boosting
  - Gradient Boosting builds models sequentially, where each new model corrects the mistakes of the previous ones.



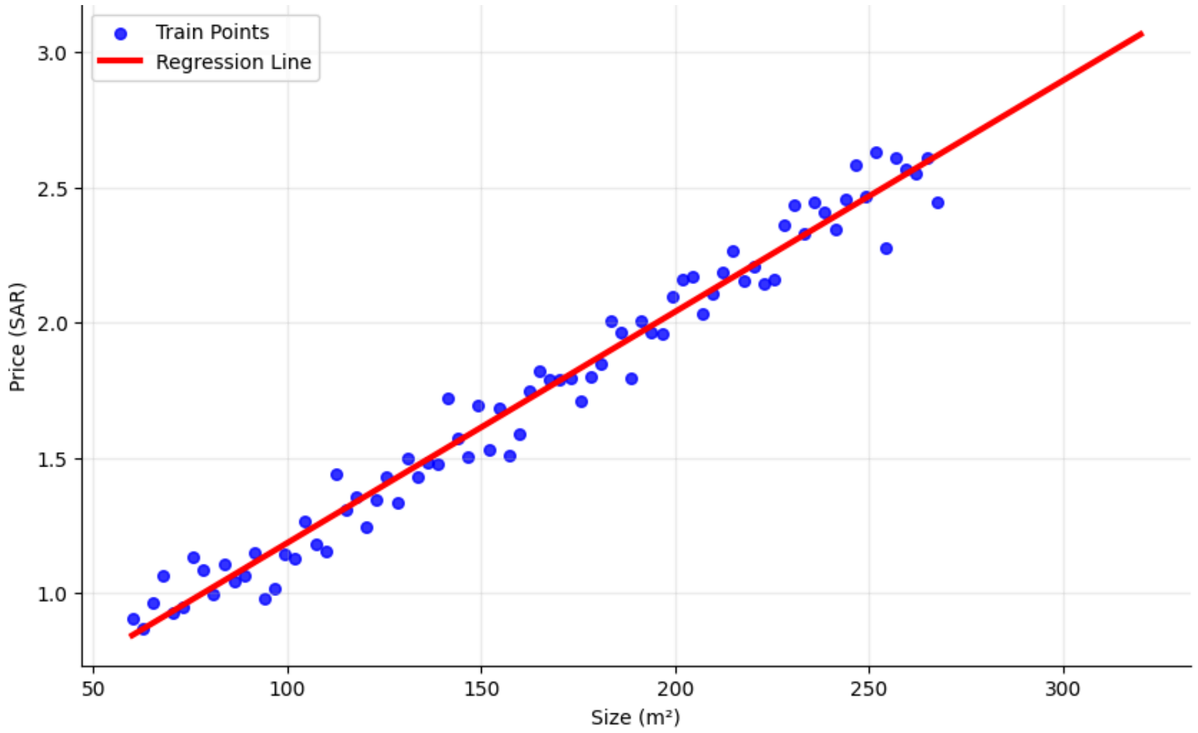
Machine Learning

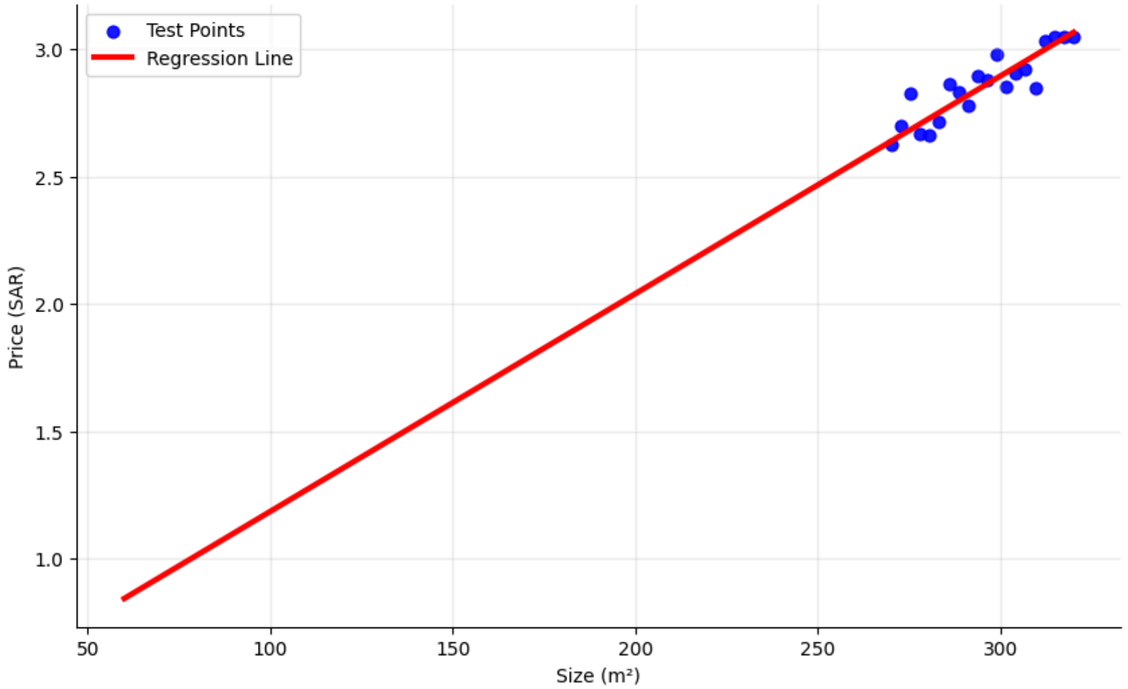
Bias variance trade off

Model Training

- Model Performance

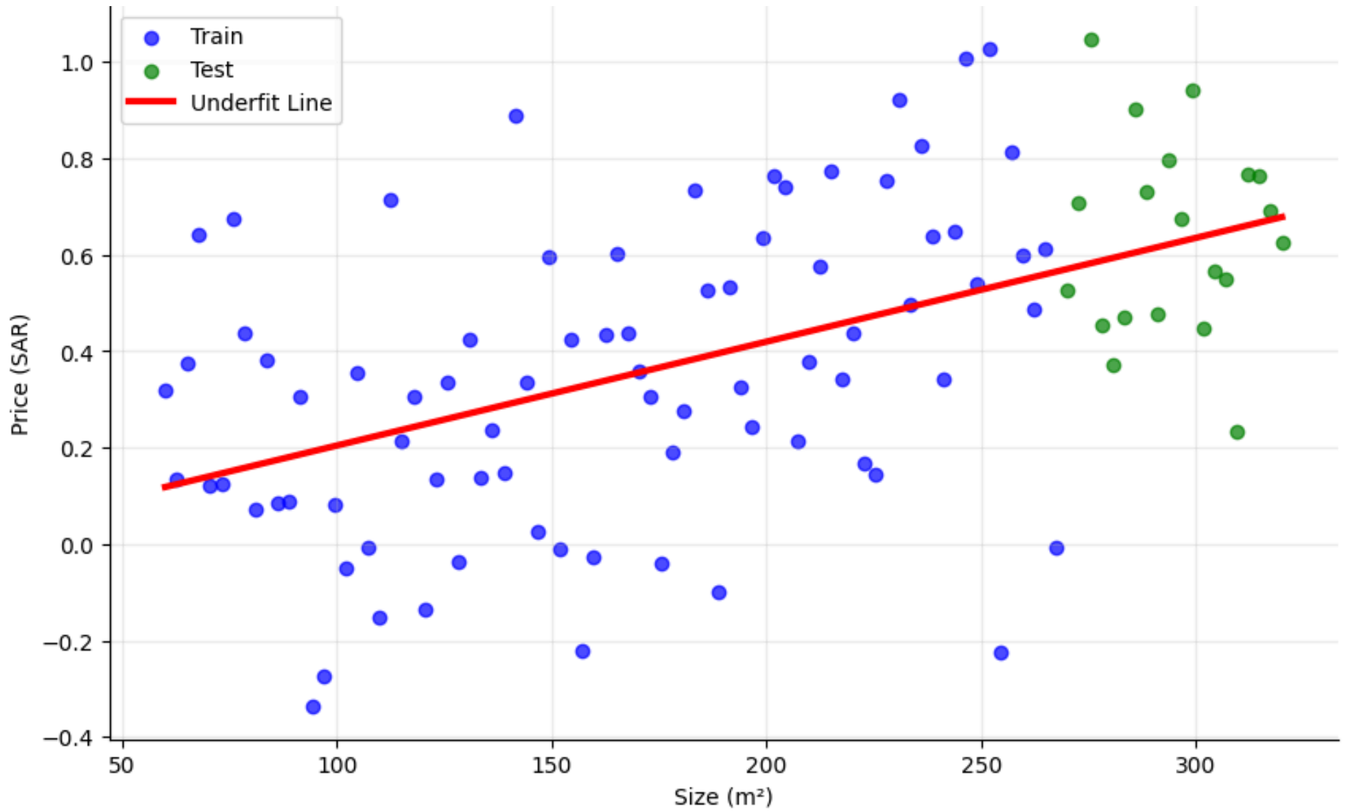
just fit





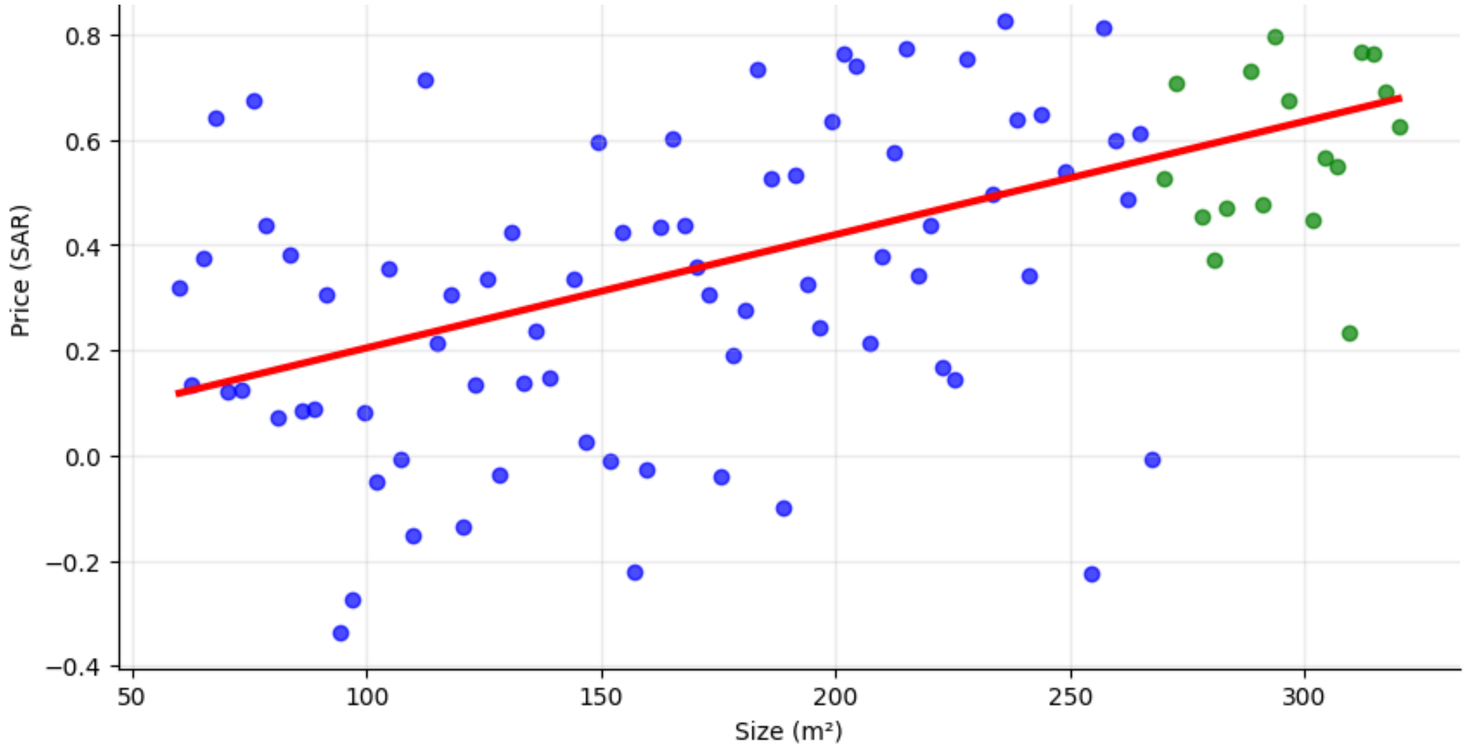
Model Training

Poor train + poor test = ?



Model Training

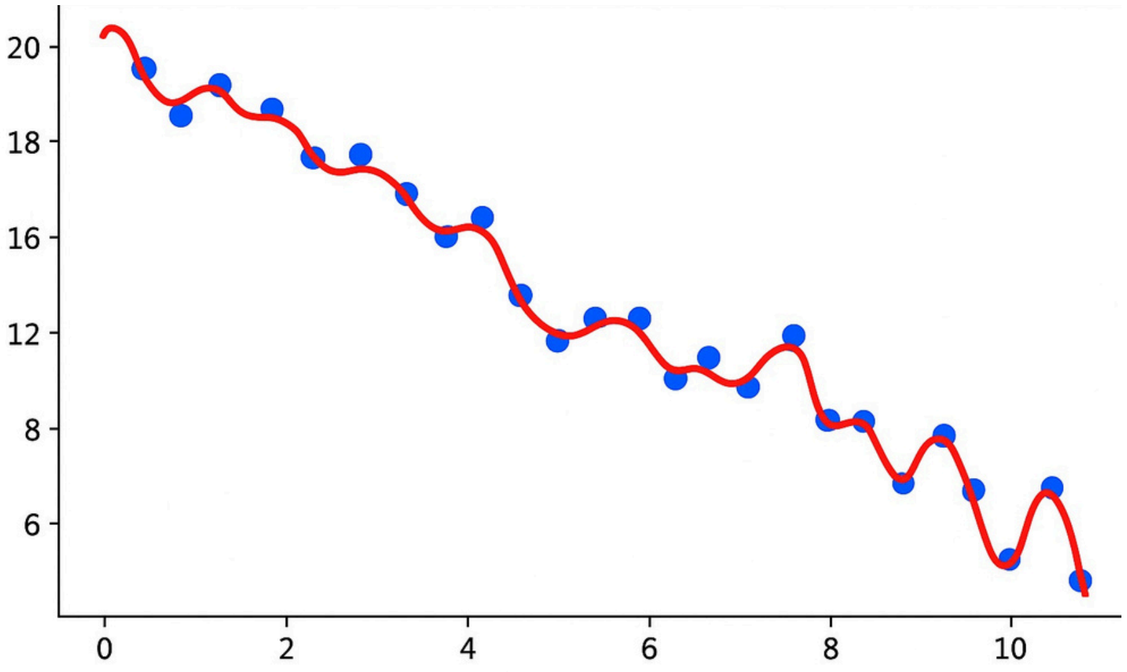
Poor train + poor test = Underfitting

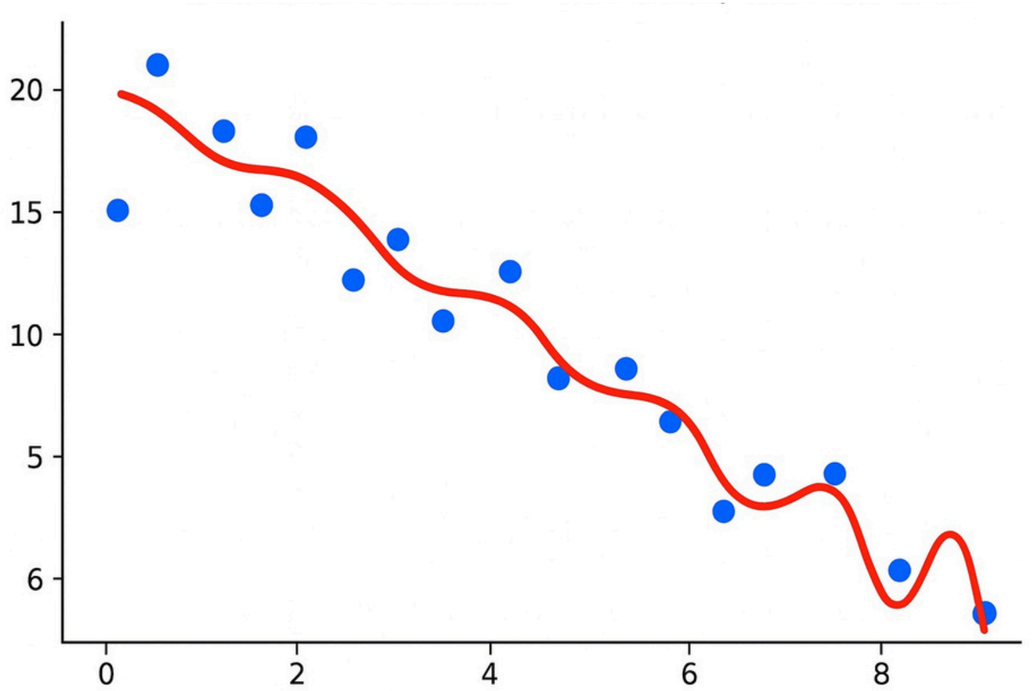


Underfitting → High Bias

Model Training

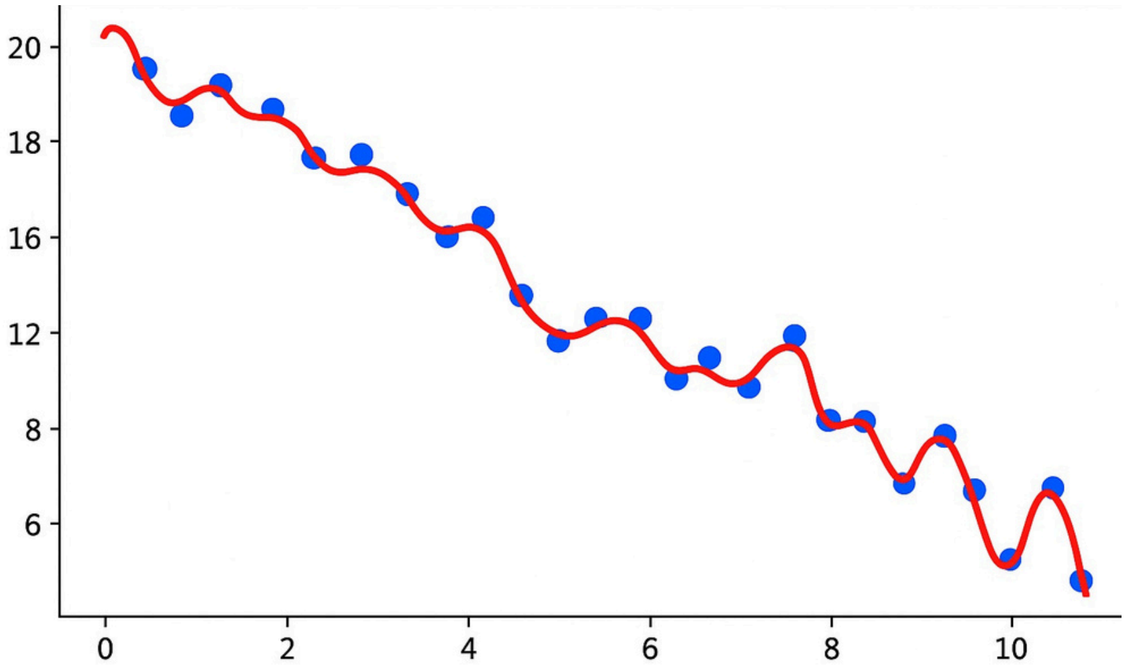
Good train + poor test = ?



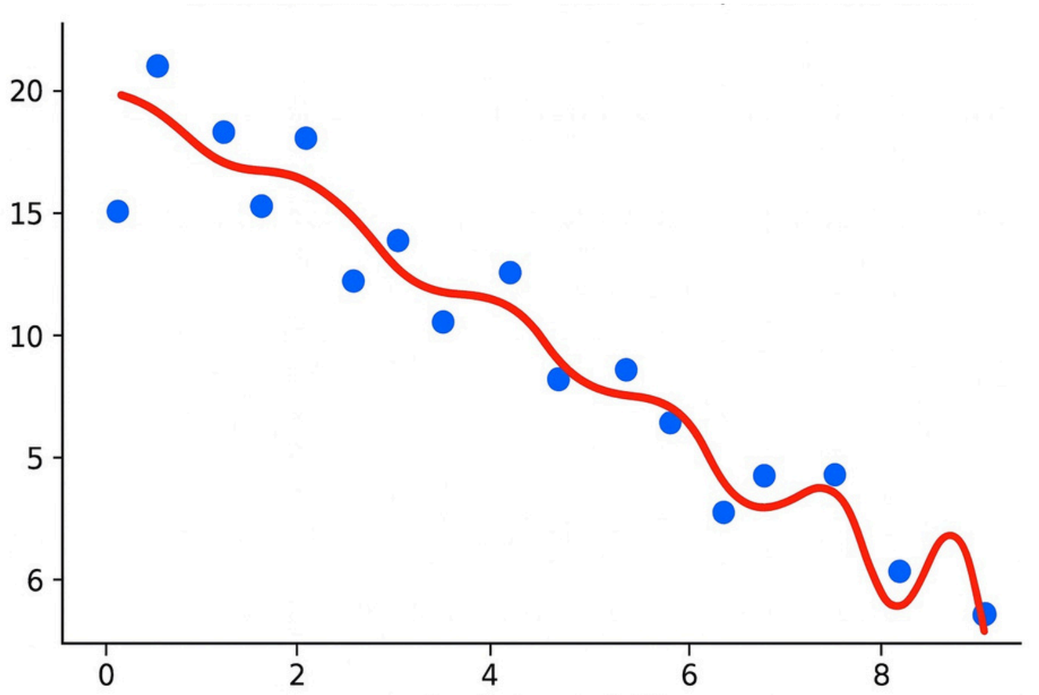


Model Training

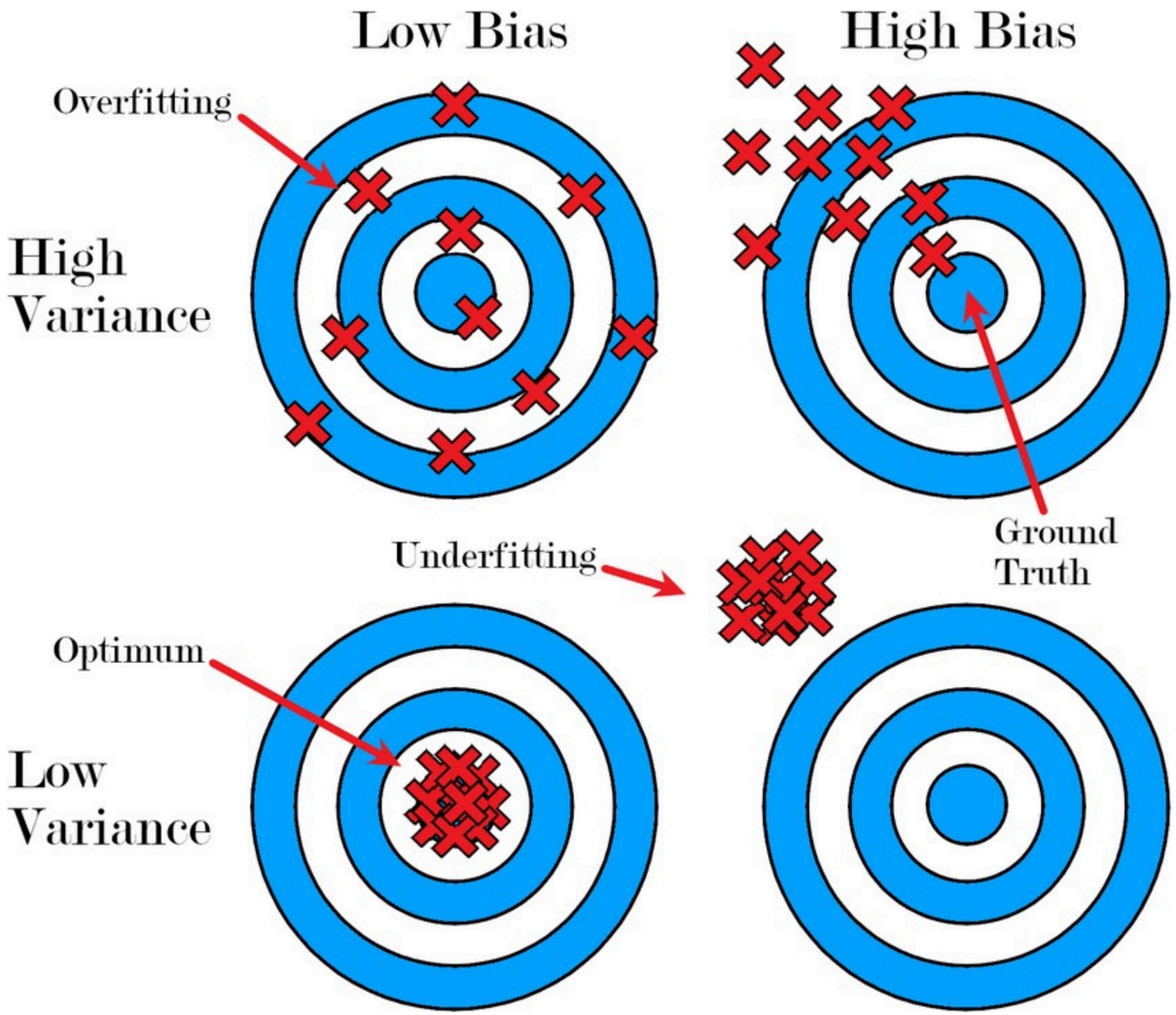
Good train + poor test = Overfitting



Overfitting → High Variance



Model Training




Q
A

Thank You