Artificial Intelligence and Machine Learning

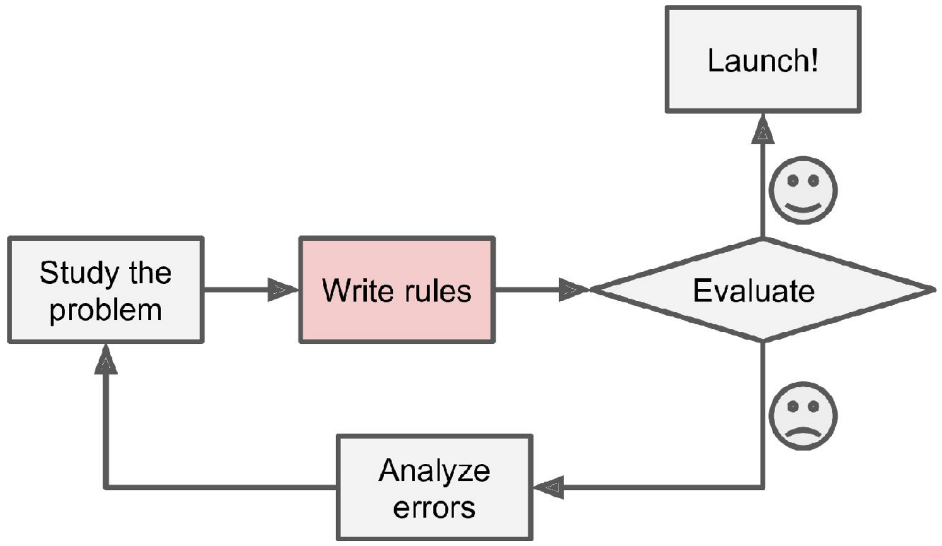
Linear Regression

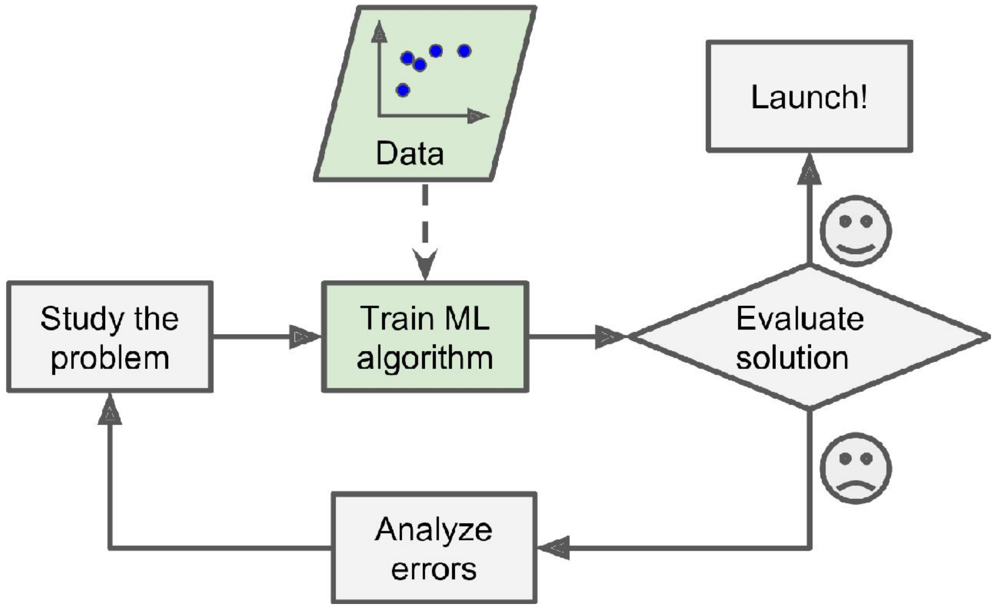
Outline

- Introduction to ML
- Linear Regression
- Optimization
- Bias-Variance Tradeoff
- Regularization

Introduction to ML

- **Machine Learning** is the science (and art) of programming computers so they can learn from data.





The traditional approach ***The Machine Learning approach***

Data Types

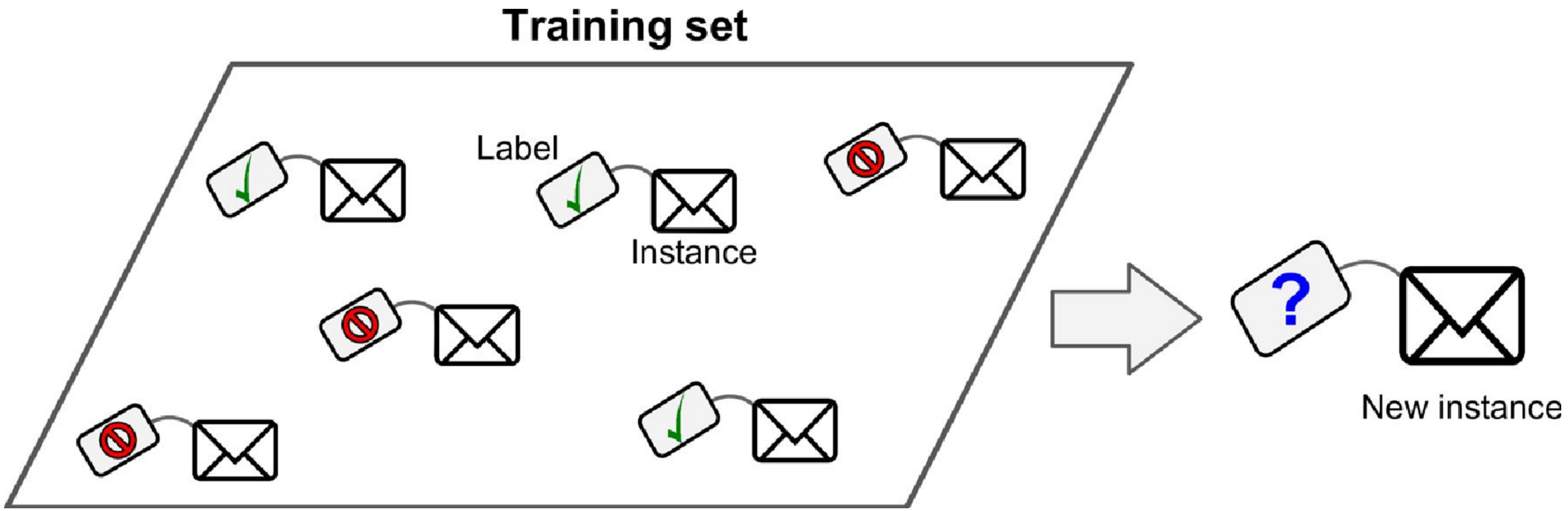
- **Tabular Data** (e.g., spreadsheets, databases)
  - Note: Columns are called **Features**. Rows are called **Samples.**
- **Time-Series Data** (e.g., stock prices, weather forecasts, IoT sensor data)
- **Text Data** (Natural Language Processing, e.g., emails, social media posts, documents)
- **Images and Videos** (Computer Vision, e.g., medical imaging, surveillance, facial recognition)
- **Audio Data** (Speech Recognition, Music Processing, e.g., voice commands, podcasts, sound classification)

ML Algorithms Types

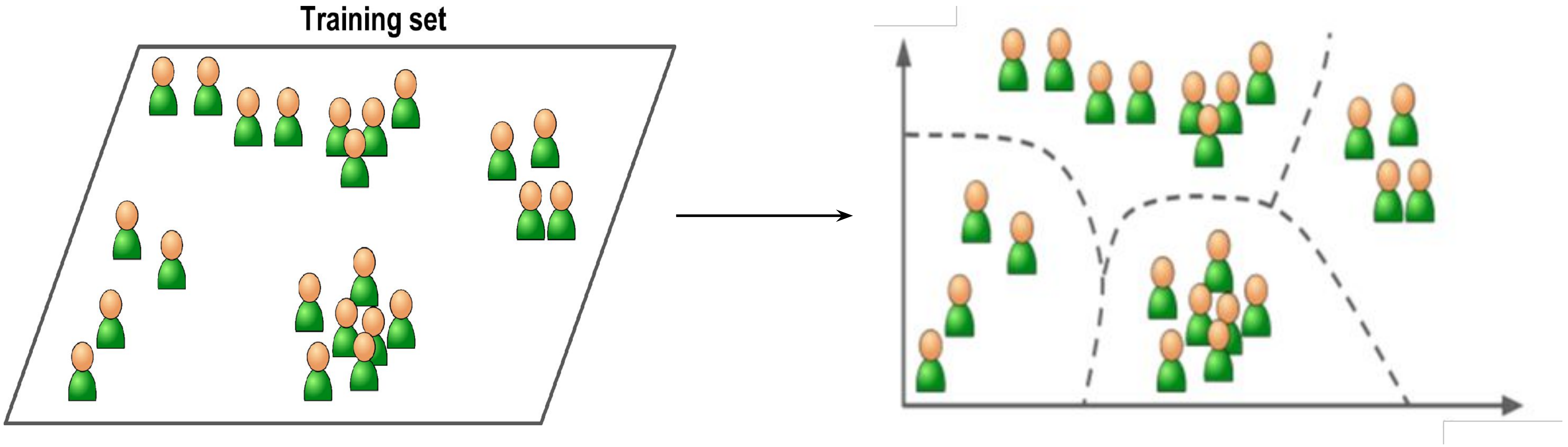
- **Supervised**: The algorithm learns from labeled data.
  - **Regression**: Predict continuous value (e.g. house prices).
  - **Classification**: Predict discrete value (e.g. spam/not-spam).
- **Unsupervised**: The algorithm works on unlabeled data. We are interested in things like:
  - **Clustering**: Grouping
  - **Dimensionality Reduction**: Reducing the Dimensions
  - **Anomaly Detection**: Detecting outliers

ML Algorithms Types cont.

- **Reinforcement Learning**: involves learning to make decisions by interacting with an environment.
  - **Reward Signal:** The agent receives feedback in the form of rewards or penalties, guiding its learning.
  - **Policy:** A strategy the agent learns to decide actions based on the current state.
  - **Value Function:** An estimate of the expected cumulative reward from a state or state-action pair.
  - **Exploration vs Exploitation:** The agent balances exploring new actions to discover rewards and exploiting known actions to maximize them.
  - Really popular in video games and robotics!(Also recently in LLMs, see RLHF)



An example of Supervised Learning: Spam Classification



An example of Unsupervised Learning: Clustering

How Does ML Work?

- Most of the ML systems consist of three main components:

  Hypothesis (Model): The function that approximates the target.

  - E.g. Linear Regression, Logistic Regression, SVM, Decision Trees, NN,…

  Optimizer: The mechanism for improving predictions of our model.

  Loss Function: The measure of how wrong the predictions are.

How Does ML Work?

- How are they related to each other?🤔

How Does ML Work?

- We firstly define our task (classification/regression) then choose an appropriate **model.**
- We will use an **optimization method** to minimize the **loss function**.
- Reached a minima?

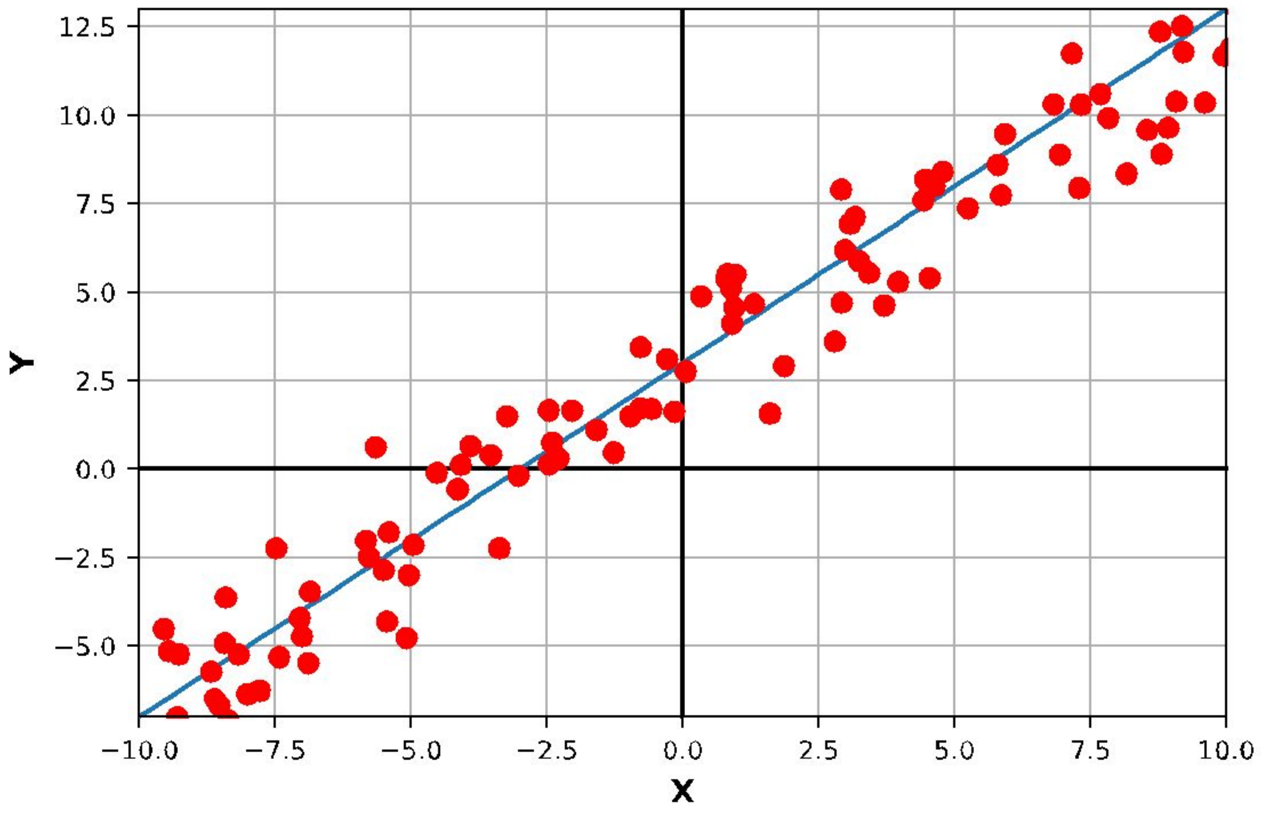
  = Model is making the least possible number of mistakes.

  = **Model trained**🎉.

Linear Regression: Motivation

- Linear Regression is “still” one of the more widely used ML/DL Algorithms
- Easy to understand and implement
- Efficient to Solve
- We will use Linear Regression to Understand the concepts of:
  - Data
  - Models
  - Loss
  - Optimization

Simple Linear Regression



Model (*Linear*) $Y = mX + b$

Y: Response Variable

X: Covariate / Ind., var/Regressors

m: slope

b: bias $\theta = \{m, b\}$

Simple Linear Regression

- **Hypothesis**:

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

- **Input**: data (x_i, y_i), $i \in \{1,2,\ldots,N\}$
  - (e.g., house size x and price y)

- **Goal**: learn values of variable (m, b)

Notation

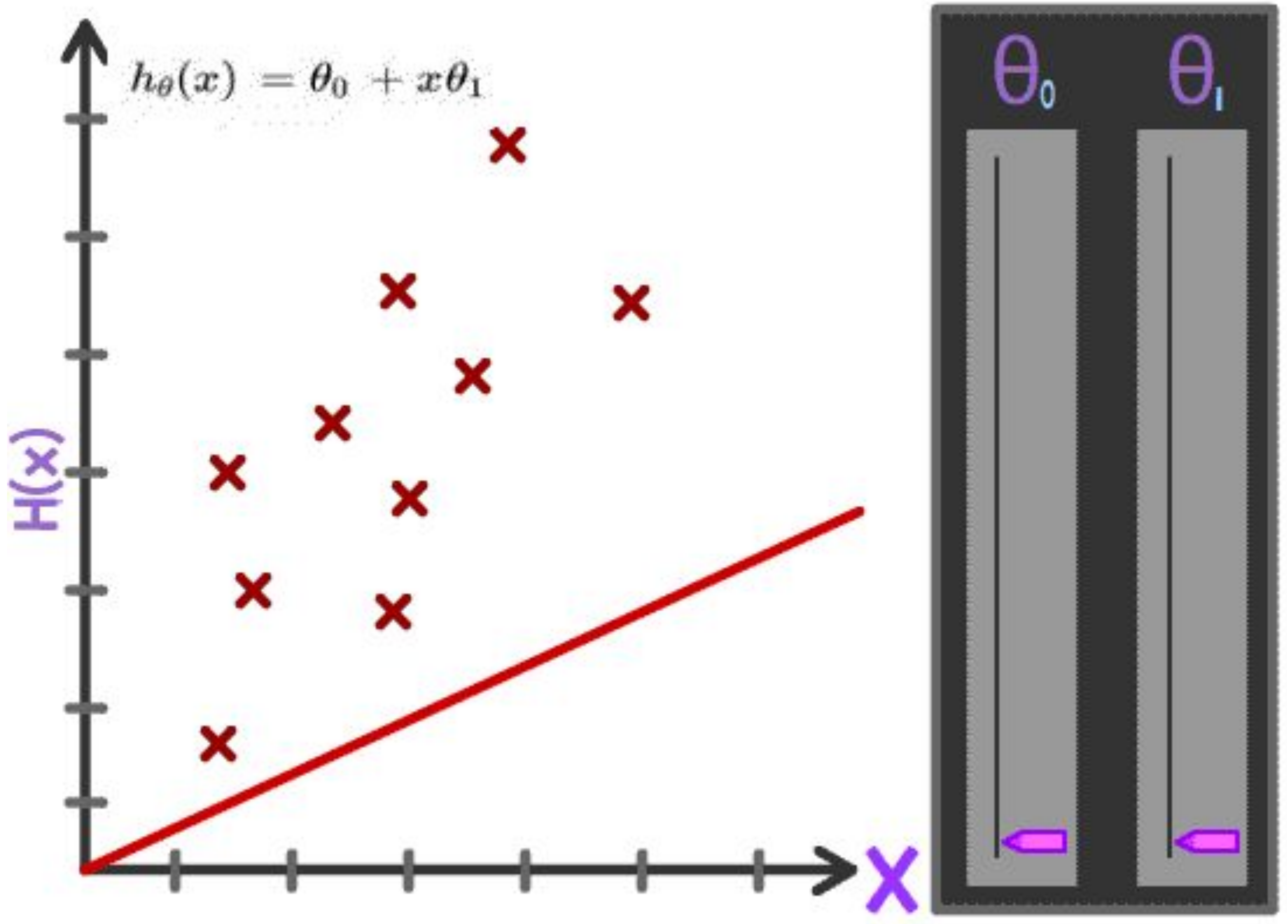
- Some clarification about the notation we will use for this course

$$X_i^{j,[k]}$$

- i is the index of the data, j is the feature number, and k is the power.

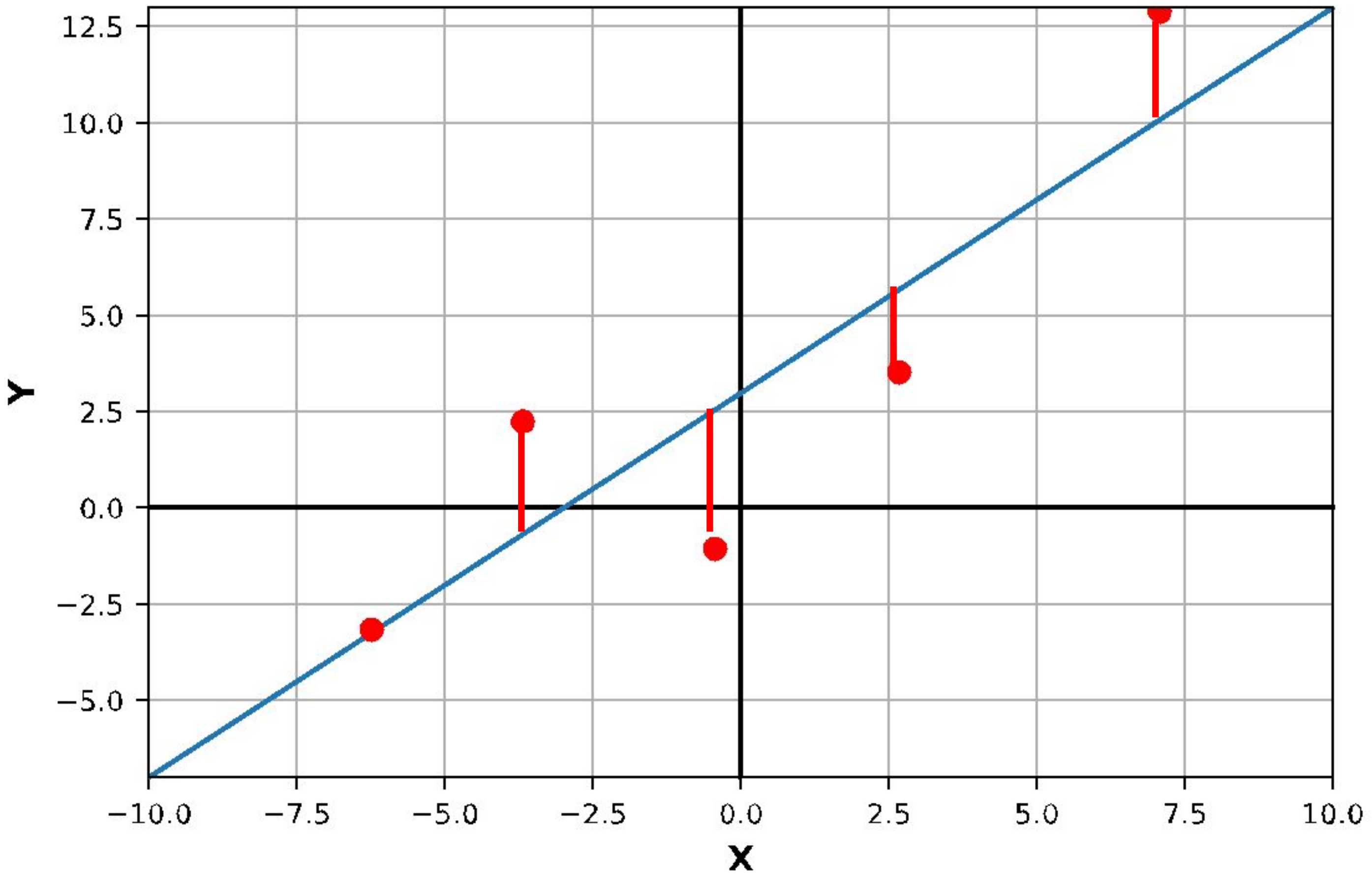
Solution Strategy for Solving the Problem

- There are countless possible lines.
- We want a line which is in some sense the "average line" that represents the data.
- Any ideas as to how we can do it?



Optimization

- To find the "best line," we should minimize the distances between our line’s predictions and all the data points.
- How to define that mathematically?



Loss Function

- For one sample, this can be represented mathematically by:

$$(y_i - \hat{y}_i)$$ (Error)

- But this could result in negative value if $\hat{y} > y$. Let's square it to remove the negative sign:

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

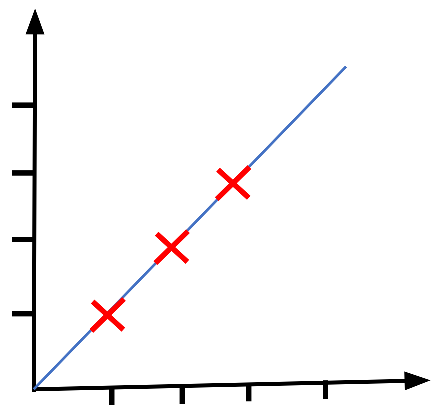
- But we have N samples, not only one. So, let's sum the errors and take the average:

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

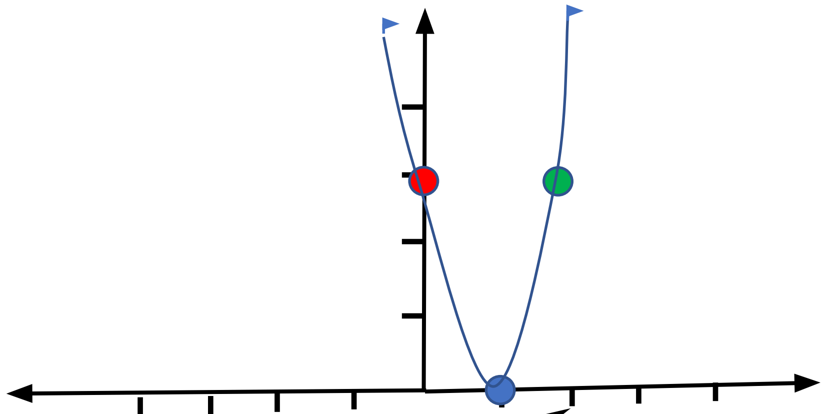
Intuition of Loss Function

$$h(x) = mx$$

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



Hypothesis



Loss Function

Notice: Lower is better.

$$\boxed{J(m=0)=14}$$

$$\boxed{J(m=1)=0}$$

$$\boxed{J(m=2)=14}$$

How to find minima of a function (Review):

- There are two approaches to find the minima:

  - **Exact (Closed-form)**: Directly calculates the solution mathematically by solving for $f'(x) = 0$.

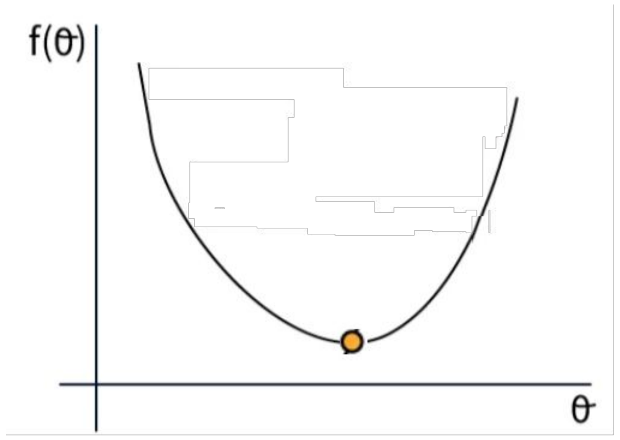
    Important Note: can be used only with a very limited number of algorithms.

  - **Approximation (Iterative approach):** Gradually improves the solution step by step.

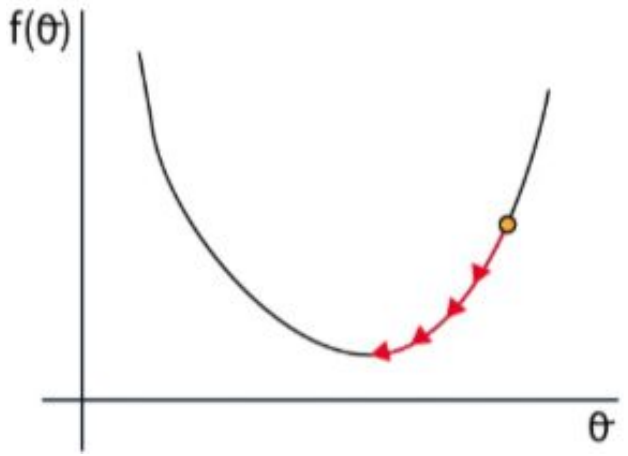
    Done by optimizers (e.g. Gradient Descent, ADAM,…etc).

How to find minima of a function (Review):

Closed-form:



Iterative:



-
  - Example: $y = x^2$ (Solution: $x = 0$)
    - Closed-form Final Result: $x = 0$
    - Iterative Final Result: $x = 0.00001$ (close enough)

أكاديمية كاوست
KAUST ACADEMY

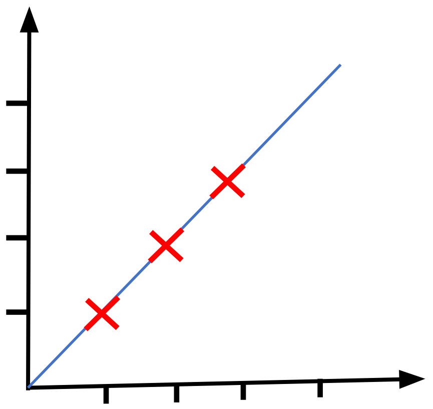
أكاديمية كاوست
KAUST ACADEMY

How to find minima of a function (Review):

- Let's try to solve this using the closed-form here (Assume $\hat{y} = mx$):

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

(Notice that $y = x$ for our 3 points).



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

$$\frac{dJ(m)}{dm} = \sum_{i=1}^{3} -2i(i - mi) \quad -2\sum_{i=1}^{3} i^2 + 2m \sum_{i=1}^{3} i^2 = 0 \quad m = 1$$

Hypothesis Function with 2 Variables

- Let's setup regression for linear function in two variables:
- The hypothesis function is:

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

- Similar to the previous problem our loss function is:

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

- Let's calculate the partial derivatives of the loss function w.r.t. m, b

Gradient of the loss function

- We get the following expressions for the gradient of the cost function

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

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

Gradient of the loss function

- Simplifying the above expressions, we get:

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

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

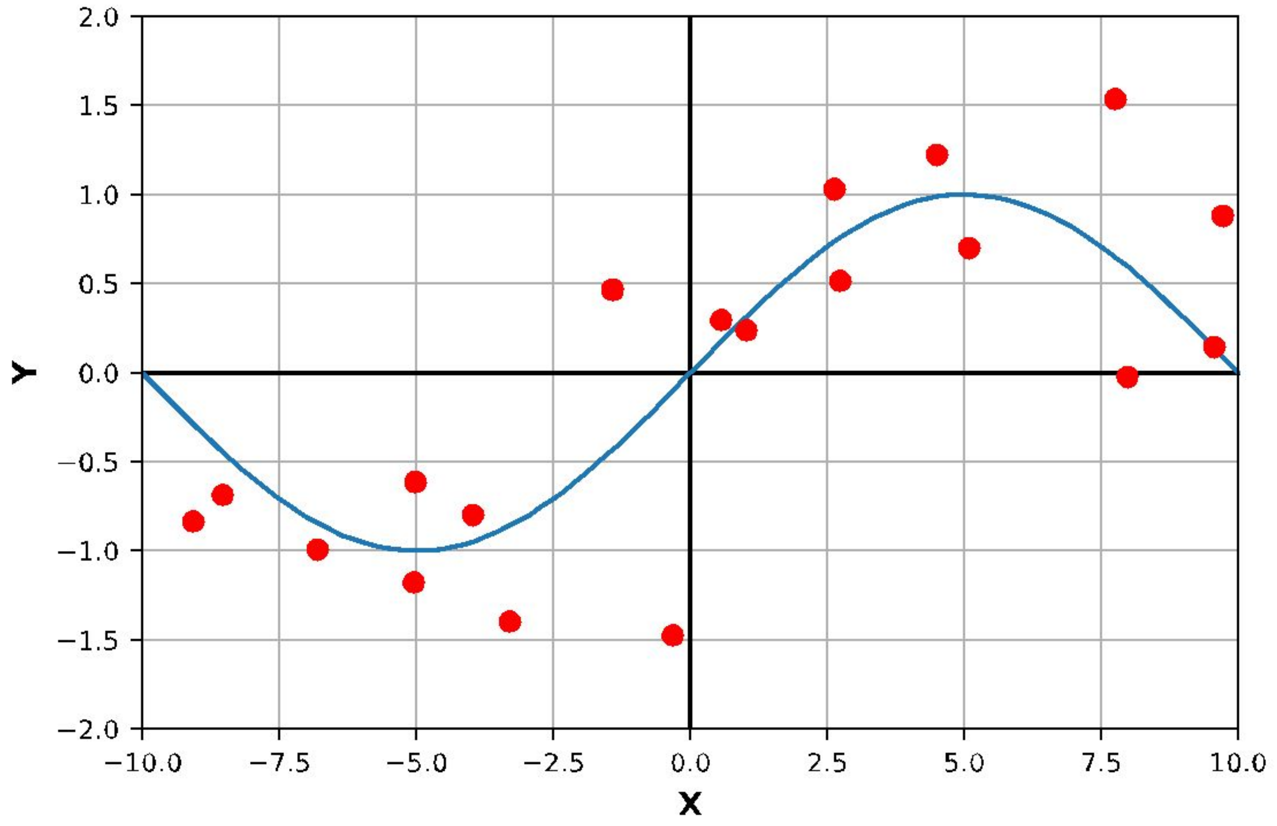
Gradient of the loss function

- Setting the Gradient equal to 0, and solving for m and b, we get

$$\underbrace{\begin{bmatrix} \frac{\sum_i x_i^2}{N} & \frac{\sum_i x_i}{N} \\ \frac{\sum_i x_i}{N} & 1 \end{bmatrix}}_{} \underbrace{\begin{bmatrix} m \\ b \end{bmatrix}}_{} = \underbrace{\begin{bmatrix} \frac{\sum_i x_i y_i}{N} \\ \frac{\sum_i y_i}{N} \end{bmatrix}}_{}$$

Fitting Non-linear Data

- What if y is a non-linear function of x, will this approach still work?



Transforming the Feature Space (Feature Engineering)

- We can transform features x_i

$$x_i = (x_i^1, x_i^2, x_i^3, \dots, x_i^m)$$

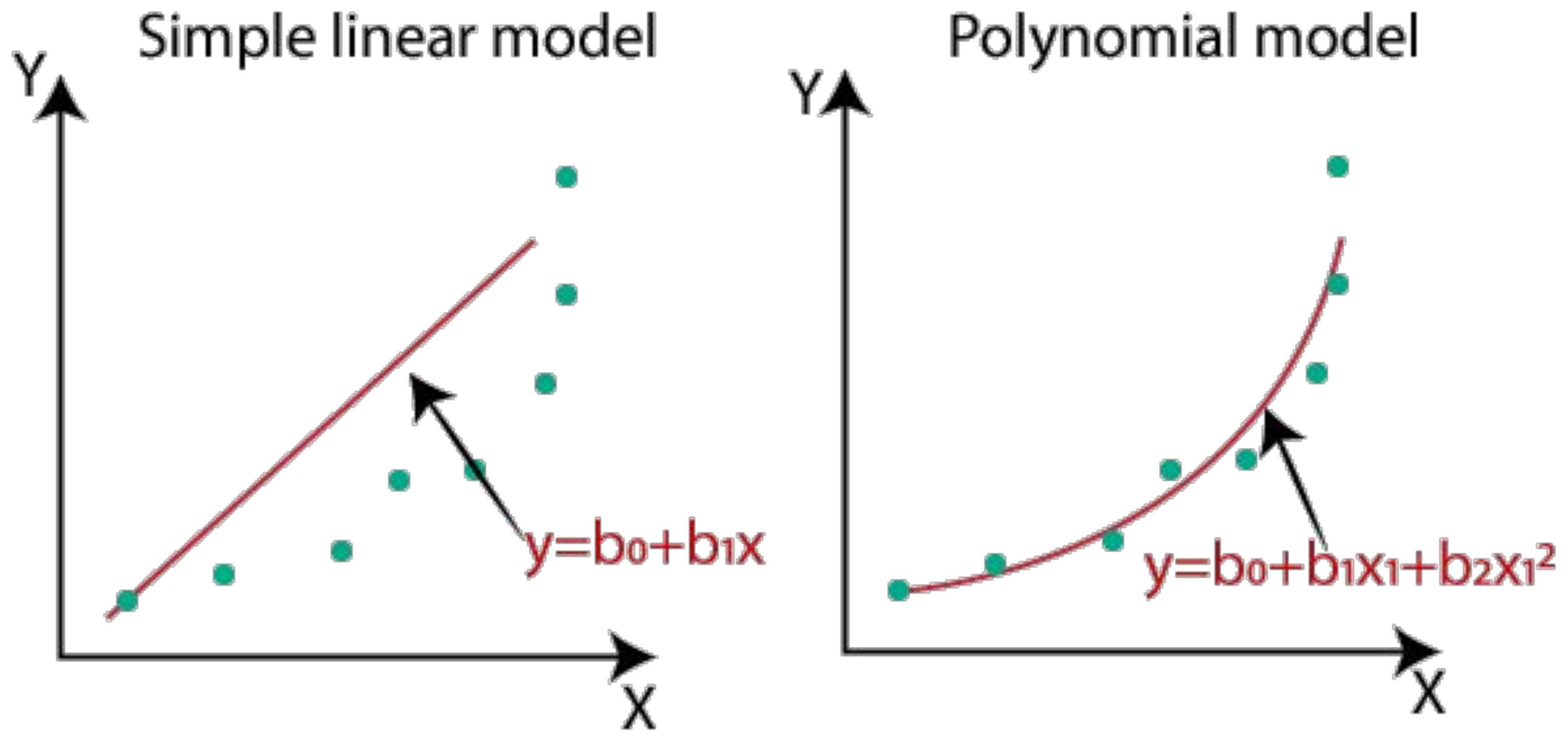
- We will apply some non-linear transformation ϕ:

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

- For example, Polynomial transformation:

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

Transforming the Feature Space (Feature Engineering)



Transforming the Feature Space (Feature Engineering)

- Example: assume you have:

$$x_i^1 : Length$$
$$x_i^2 : Width$$

You can add $x_i^3 : Area = x_i^1 * x_i^2$ to the dataset.

Other types:

- Cosine, splines, radial basis functions, etc.
- Encoding (Label encoding, One-hot,…)
- Domain-related features (e.g. financial measures)
- Time-related features (Day, month, year,…)
- Group-level features (e.g., average income per household, total sales per region, median age per team). Often called “Aggregation features”.

Gradient of the loss function

- Let’s get back to the gradients…

Issues with the Approach

- Assume we have 100 variables instead of 2.
- Calculating gradients like this can quickly become tedious
- **Notice**: Each term on either side of the experssion can be written as a dot product of two vectors (maybe we can calculate it more efficiently)?

- Let’s explore if we can do something better through **vectorization** (Writing equations as matrices).

Vectorization

- To truly appreciate the power of vectorization. Let's make the problem a little more complex. The hypothesis function is now

$$\hat{y}_i = w_0 + w_1 x_i^1 + w_2 x_i^2 + \cdots + w_M x_i^M$$

- Where w_j $(j = 0,1, \dots, M)$ are the unknown weights of the data, and x_i^j is the jth feature of the ith input.
- Next, we denote the discrepency between y_i and $\hat{y}_i$ as ϵ_i

$$y_i = \hat{y}_i + \epsilon_i$$

Vectorization

- Now let’s collect the above equation for all N datapoints

$$y_1 = \hat{y}_1 + \epsilon_1$$
$$y_2 = \hat{y}_2 + \epsilon_2$$
$$\vdots$$
$$y_N = \hat{y}_N + \epsilon_N$$

Vectorization

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

$$y_1 = w_0 + w_1 x_1^1 + w_2 x_1^2 + \ldots + w_M x_1^M + \epsilon_1$$

$$y_2 = w_0 + w_1 x_2^1 + w_2 x_2^2 + \ldots + w_M x_2^M + \epsilon_2$$

$$\vdots$$

$$y_N = w_0 + w_1 x_N^1 + w_2 x_N^2 + \ldots + w_M x_N^M + \epsilon_N$$

Vectorization

- Collecting the equations in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^M \\ 1 & x_2^1 & x_2^2 & \dots & x_2^M \\ 1 & x_3^1 & x_3^2 & \dots & x_3^M \\ & . & & & \\ & . & & & \\ & . & & & \\ 1 & x_N^1 & x_N^2 & \dots & x_N^M \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ . \\ . \\ . \\ \epsilon_N \end{bmatrix}$$

Vectorization

- Notice the rows of the matrix on the right are data samples:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_N \end{bmatrix} = \begin{bmatrix} \dots & \mathbf{x_1} & \dots \\ \dots & \mathbf{x_2} & \dots \\ \dots & \mathbf{x_3} & \dots \\ & . & \\ & . & \\ & . & \\ \dots & \mathbf{x_N} & \dots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ . \\ . \\ . \\ \epsilon_N \end{bmatrix}$$

Vectorization

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

- Let's formalize some notations:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_N \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \dots & \mathbf{x_1} & \dots \\ \dots & \mathbf{x_2} & \dots \\ \dots & \mathbf{x_3} & \dots \\ & . & \\ & . & \\ & . & \\ \dots & \mathbf{x_N} & \dots \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_M \end{bmatrix} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ . \\ . \\ . \\ \epsilon_N \end{bmatrix}$$

$$\boldsymbol{y} = \boldsymbol{X\theta} + \boldsymbol{\epsilon}$$

Cost function for the Vectorized form

- Notice that we are using the MSE cost function:

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

- Using the defintion of epsilon we can write the above as:

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

- Using the definition of dot product the above can be written as:

$$J = \frac{1}{N}\sum_{i=1}^{N}(\epsilon_i)^2 = \frac{1}{N}\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$$

Optimization

- The optimization problem is now:

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

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

- We will use chain rule to calculate the gradient of the cost function:

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

Linear Least Squares

- We get:

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

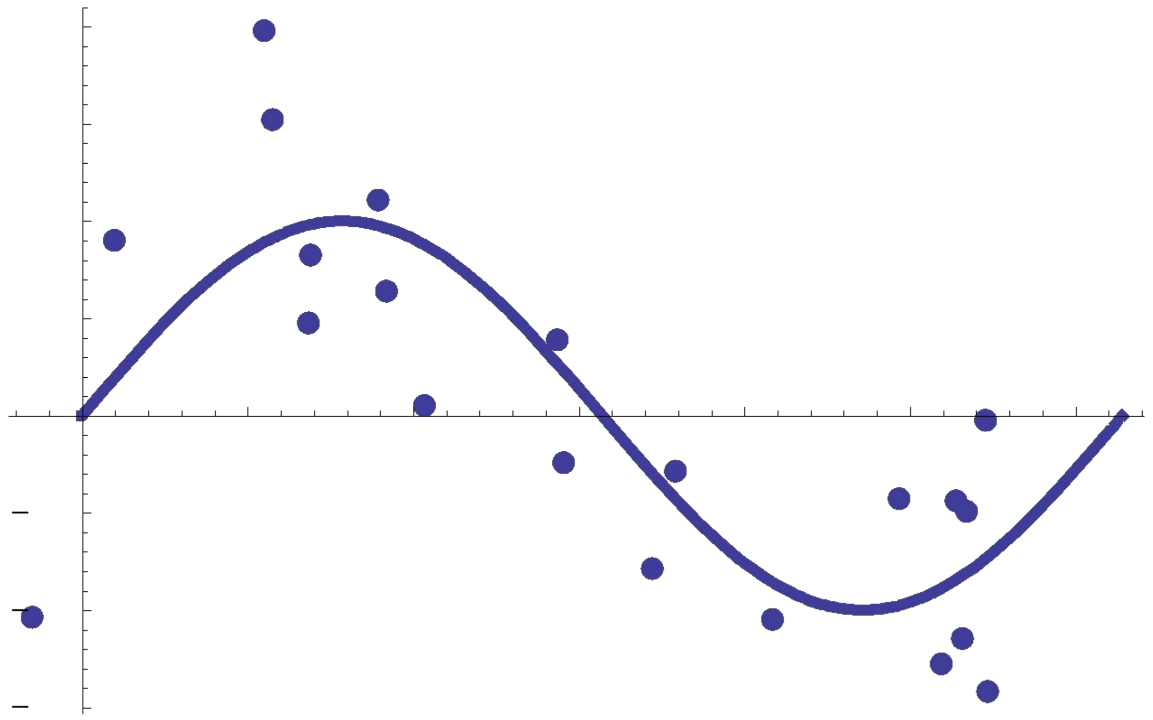
- Setting it equal to zero we can solve for $\boldsymbol{\theta}$:

$$\boxed{\boldsymbol{\theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}}$$

Closed-form solution for Linear Regression

Bias and Variance

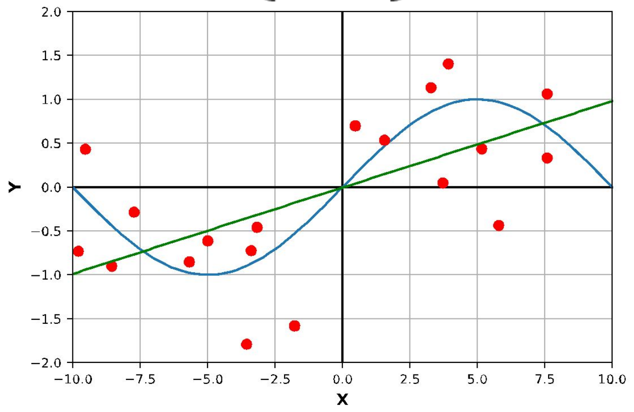
- What if Y has a non-linear response?



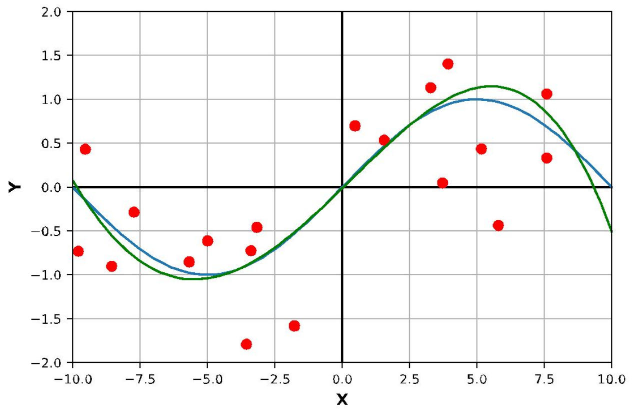
- Can we still use a linear model?

What is Bias and Variance?

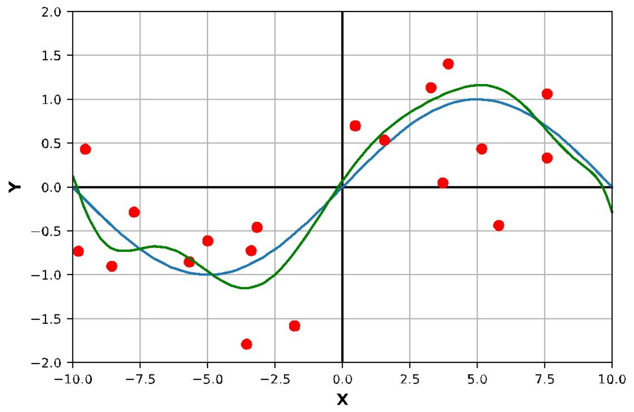
$\{1, x\}$



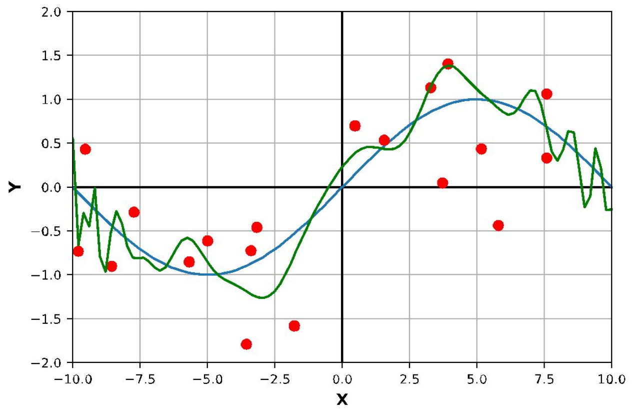
- $\{1, x, x^{[2]}, x^{[3]}, x^{[4]}\}$



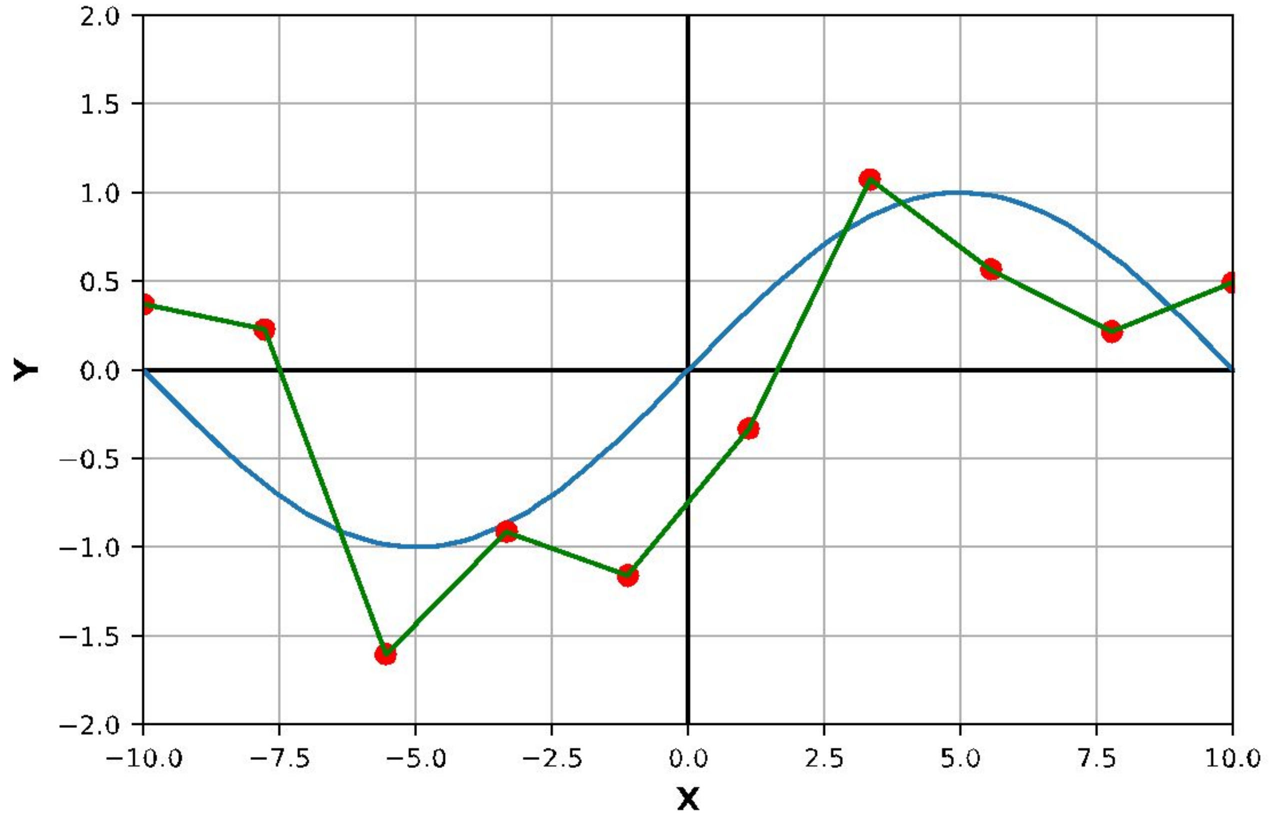
$\{1, x, x^{[2]}, \dots, x^{[9]}, x^{[10]}\}$



$\{1, x, x^{[2]}, \dots, x^{[99]}, x^{[100]}\}$



Real Bad Overfit?



Bias-Variance Tradeoff

- So far, we have minimized the error (loss) with respect to **training data**
  - Low training error does not imply good expected performance: **over-fitting**
- We would like to reason about the **expected loss (Prediction Risk)** over:
  - Training Data: $\{(y_1, x_1), \ldots, (y_n, x_n)\}$
  - Test point: (y_*, x_*)
- We will decompose the expected loss into:

$$\mathbf{E}_{D,(y_*,x_*)}\left[(y_* - f(x_*|D))^2\right] = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

Bias Variance Plot

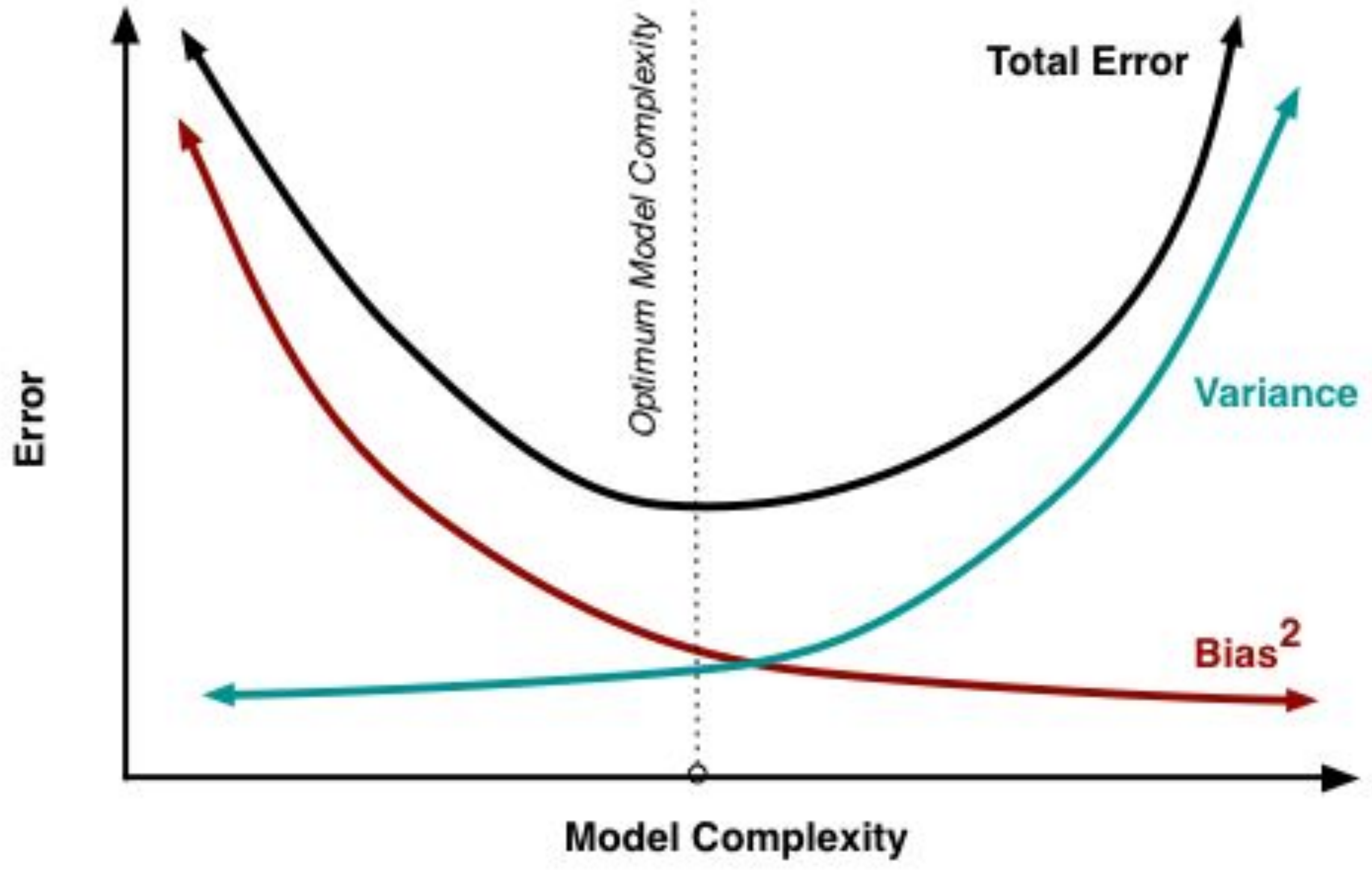


Image from http://scott.fortmann-roe.com/docs/BiasVariance.html

Evaluating models and Improving them

R-squared (R^2 or r^2) — “Coefficient of Determination”

- A **statistical metric** used to measure how well the independent variables explain the variability in the dependent variable.
- It provides a measure of the goodness-of-fit (**performance**) for a regression model:

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}}$$

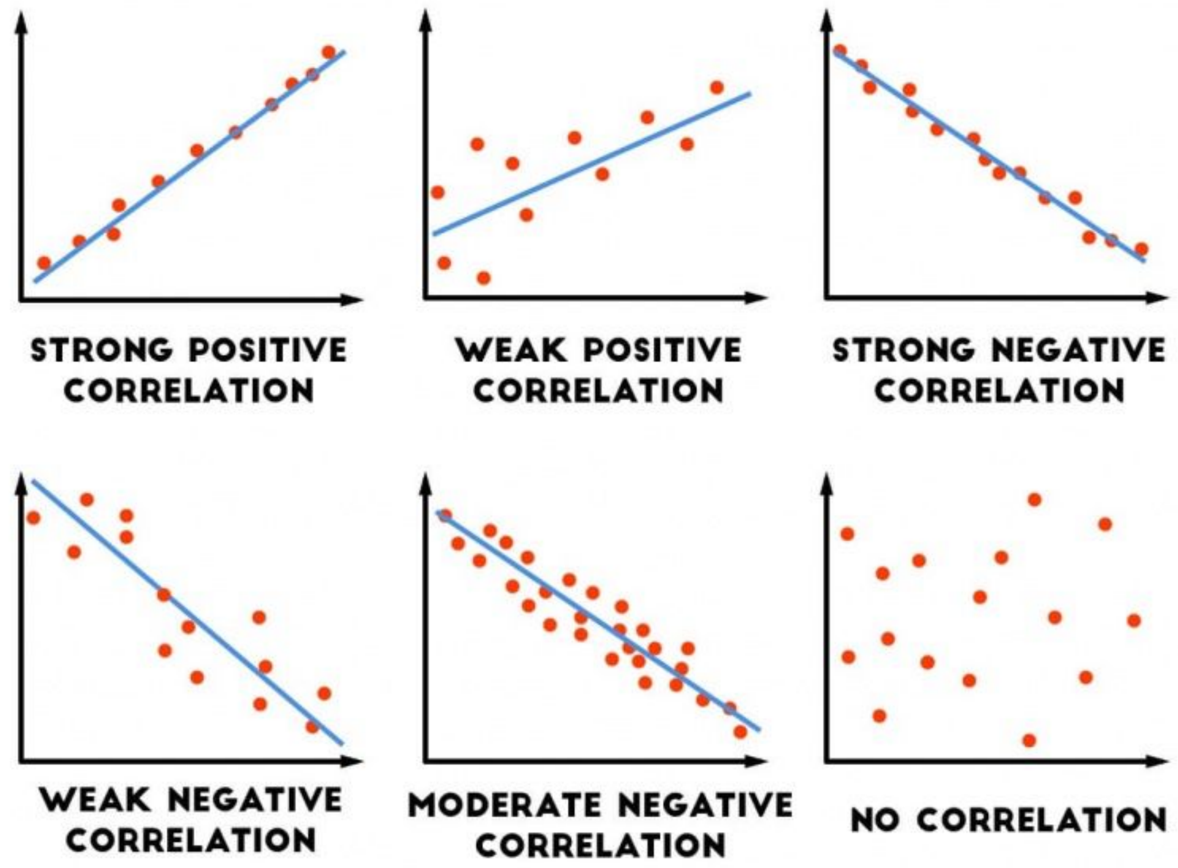
Where:

$\mathrm{SS}_{\mathrm{residual}} = \sum (y_i - \hat{y}_i)^2$: The sum of squared residuals (difference between observed and predicted values).

$\mathrm{SS}_{\mathrm{total}} = \sum (y_i - \bar{y})^2$: (Total Sum of Squares): The total variation in the data around the mean.

Learn more!

Visualisation of R^2



Interpretation:

- R^2=1: The model perfectly explains the data.
- R^2=0: The model explains none of the variability (as good as guessing the mean).
- R^2<0: The model performs worse than a simple horizontal line at the mean of the target variable.

Properties of R^2

Range:

- $0 \leq R^2 \leq 1$
  - R^2=1: Perfect model; predictions perfectly match the observations.
  - R^2=0: Model does no better than the mean of the dependent variable.

Interpretability:

- R^2 indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.

Limitations:

- R^2 increases with the addition of independent variables, even if they don't improve model prediction significantly.
- Does not account for overfitting or the complexity of the model.

Negative Values:

- In rare cases, R^2 can be negative when the model fits the data worse than a horizontal line representing the mean of the dependent variable.

Adjusted R^2:

- Adjusted R^2 penalizes for the addition of non-significant predictors and is often a better metric for comparing models (here ***n*** is the number of data points and ***p*** is the number of predictors):

$$R^2_{\text{adj}} = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$$

R-squared vs. Cost Function:

- **Cost Function**: A mathematical function used to optimize a model during training by minimizing the prediction error (e.g., Mean Squared Error or MSE).
  - Example: Gradient Descent minimizes MSE for linear regression.
- **R-squared**: A metric to evaluate how well the model fits the data **after training**. It is not used during model optimization.

Limitations of R^2:

1. **Doesn't Indicate Causation**: A high R^2 doesn’t mean the predictors cause the dependent variable.
2. **Sensitive to Overfitting**: A complex model might have a high R^2 but poor generalization.
3. **Adjusted R^2**: For models with many predictors, use adjusted R^2, which accounts for the number of predictors to avoid overestimating performance.

Data Split

- To ensure your model doesn’t overfit to the training data, you should have another subset called **testing data.**
- You will evaluate your model against this subset, and based on its **metric score (e.g. accuracy)** you will decide if it’s overfitting or not.

- But how should I split my data?

Data Split

- **Hold-out set**:
  - A portion of the dataset set aside and not used during training.
  - E.g. 80% for training and 20% for testing.

Issues:

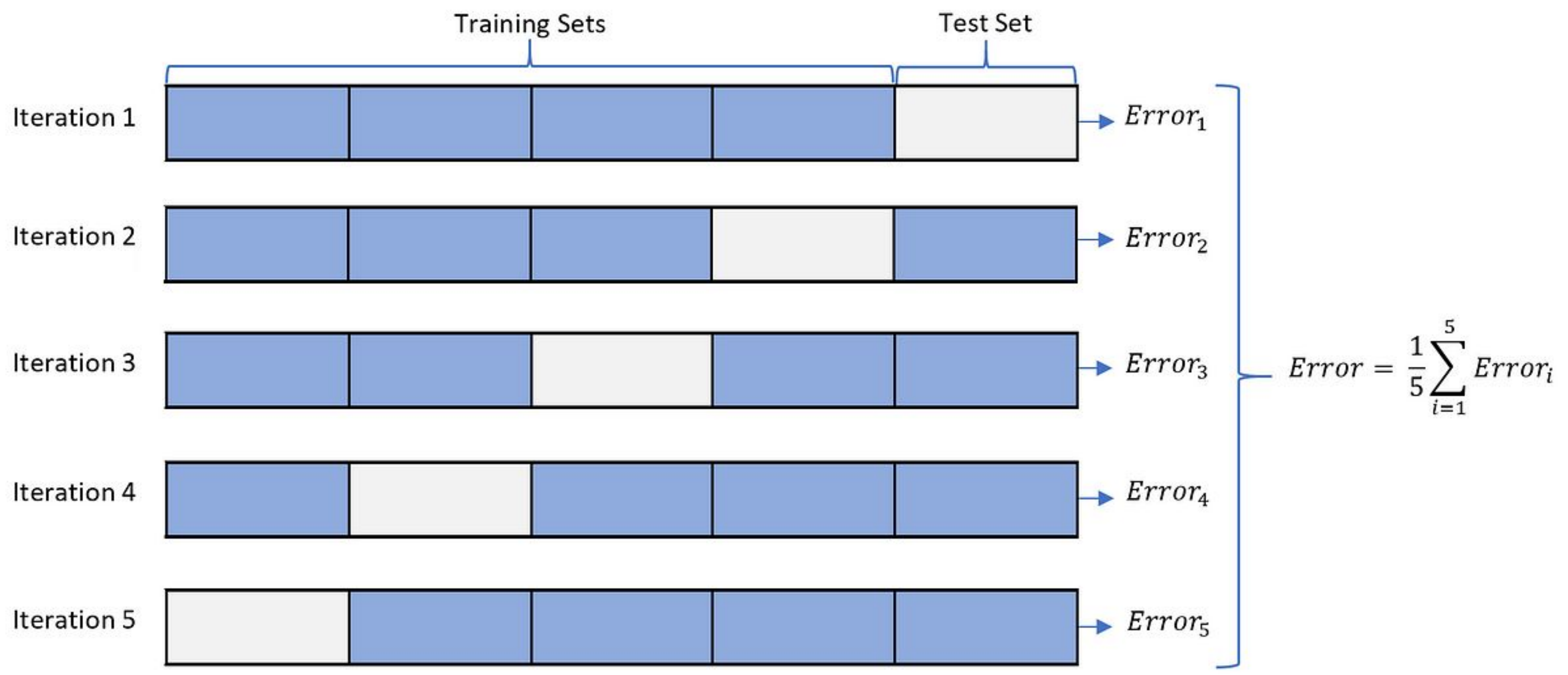
- Imagine you have these labels: [1, 1, 2, 2, 2, 3, 3, 3, 3, 3] and you took last 30% as test: [3,3,3]. <u>You didn’t include 1 and 2 in test</u>!

  Solution: Always shuffle before split: [3, 2, 3, 1, 3, 2, 3, 1, 3, 2] □ test: [1,3,2]
- My <u>dataset is small</u>. Taking 20% as test would not be representative!

  Solution: Use KFold.

Data Split

- **K-Fold Cross Validation (CV)**:
  - Split data into k parts (folds), trains on $k-1$ folds, test on the remaining fold, and repeats k times then average the scores.


Training Sets
Test Set
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
$Error_1$
$Error_2$
$Error_3$
$Error_4$
$Error_5$
$Error = \frac{1}{5}\sum_{i=1}^{5} Error_i$