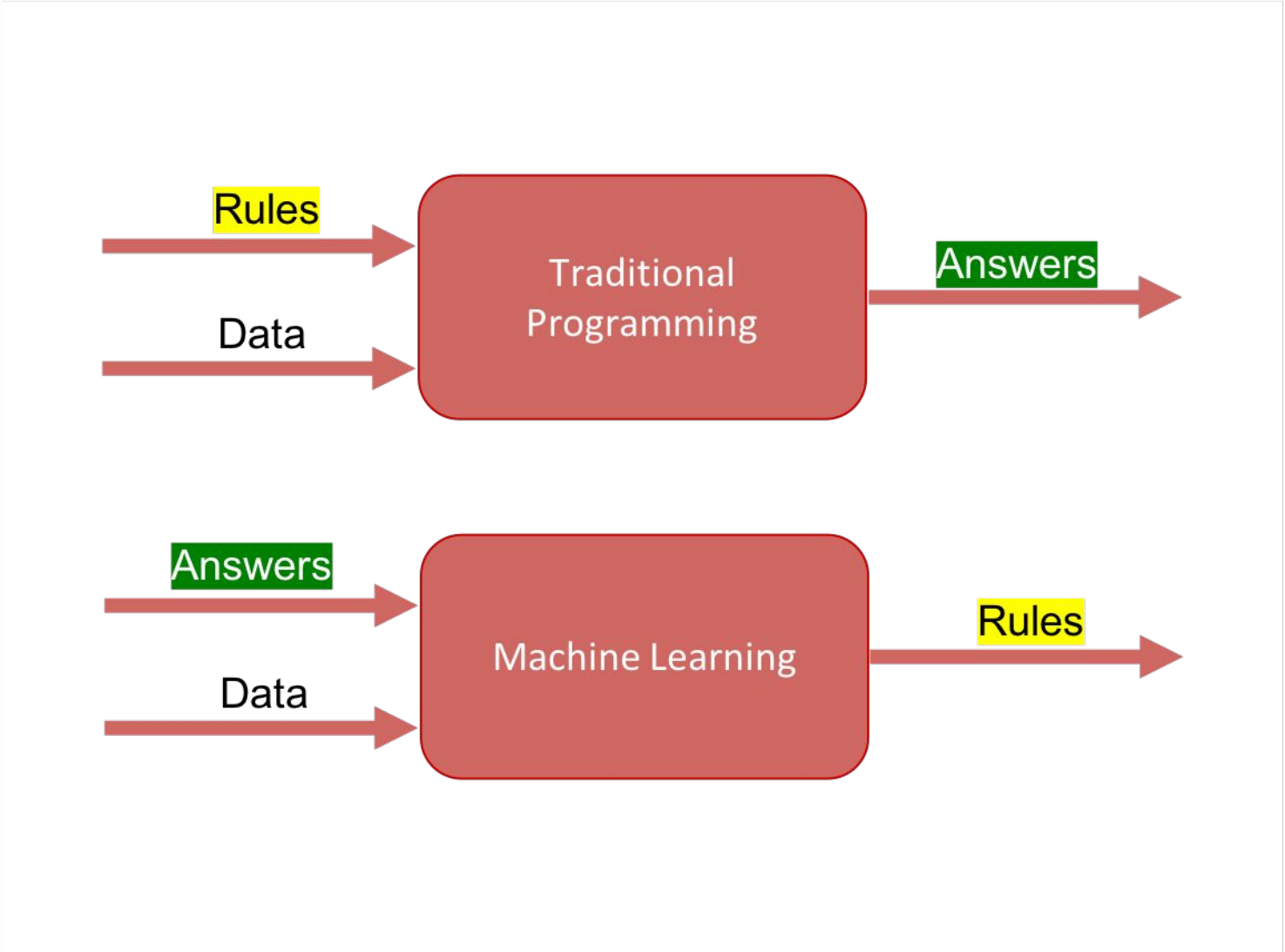
MACHINE LEARNING

Dr. Jayanti Dansana

Introduction to Machine Learning



Introduction to Machine Learning:

Machine Learning (ML) is a branch of artificial intelligence (AI) that focuses on building systems that learn from data, improve over time, and make predictions or decisions without being explicitly programmed.

What is Machine Learning?

- It is a method of data analysis that automates analytical model building.
- ML algorithms use computational methods to "learn" information directly from data without relying on a predetermined equation.
- It involves developing algorithms that can automatically detect patterns in data and use these patterns to predict future data or outcomes.

Why is Machine Learning Important?

Data Explosion: We generate an enormous amount of data every day, and analyzing it manually is impractical. ML helps in extracting useful insights.

Automation: ML algorithms can automate repetitive tasks like fraud detection, email classification, recommendation systems, and more.

Predictive Power: Machine learning models can predict future trends by recognizing patterns in past data.

Types of Machine Learning:

Supervised Learning:

Definition: Supervised learning is where the model is trained on a labeled dataset. The algorithm learns from the input-output pairs (known as training data) and tries to map inputs to outputs.

Example: Predicting house prices based on features like area, number of rooms, etc.

Key Algorithms:

- Linear Regression
- Logistic Regression
- Support Vector Machines (SVM)
- Decision Trees
- K-Nearest Neighbors (KNN)
- Neural Networks

Applications:

- Spam email detection
- Stock price prediction
- Disease prediction (e.g., cancer diagnosis)

Unsupervised Learning:

Definition: In unsupervised learning, the model is given data without labels (no predefined output). The model attempts to find hidden patterns or intrinsic structures in the data.

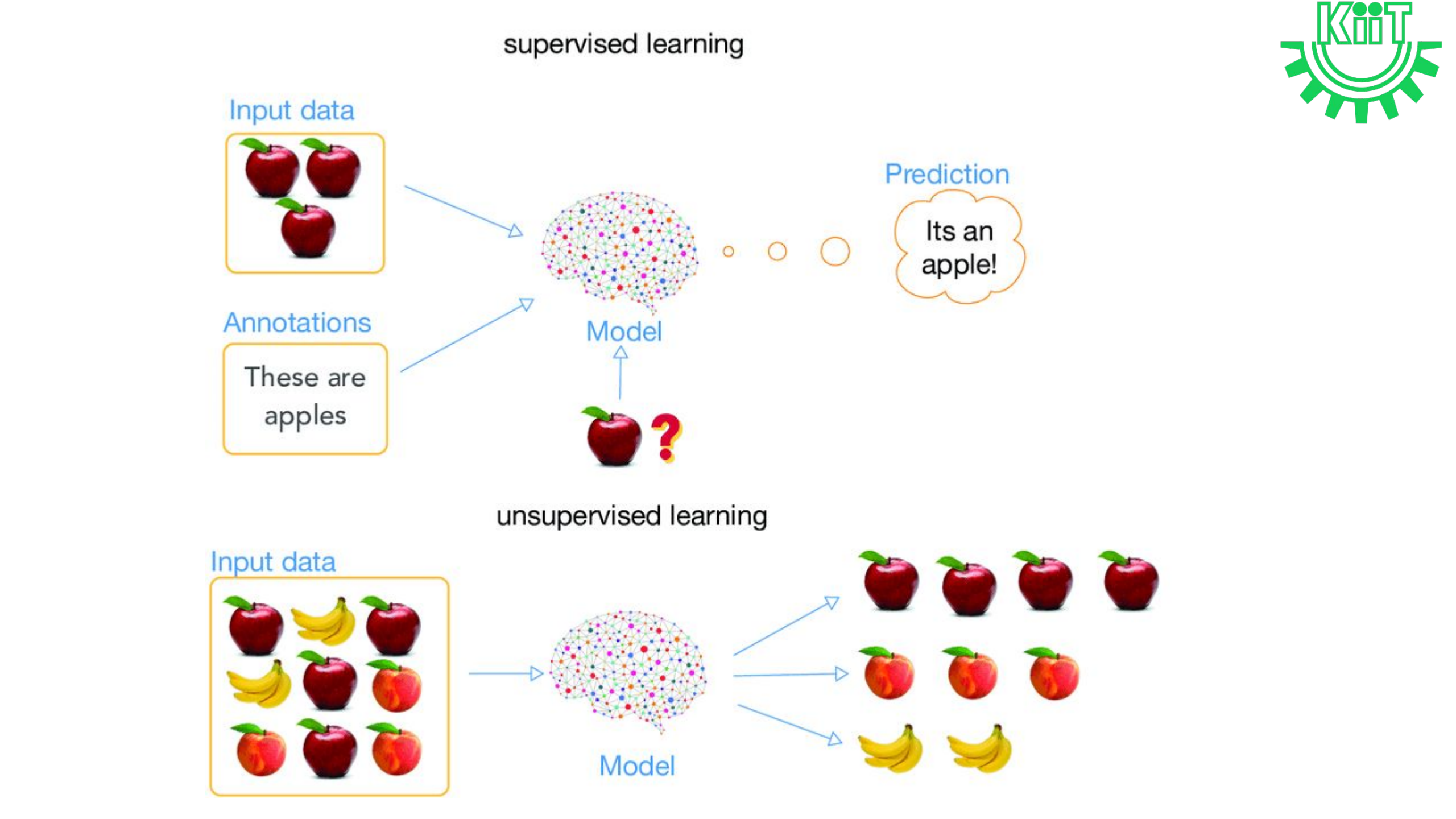
Example: Grouping customers based on purchasing patterns.

Key Algorithms:

- K-Means Clustering
- Hierarchical Clustering
- Principal Component Analysis (PCA)
- Gaussian Mixture Models (GMM)

Applications:

- Market segmentation
- Anomaly detection
- Dimensionality reduction


supervised learning
Input data
Annotations
These are apples
Model
Prediction
Its an apple!
unsupervised learning
Input data
Model

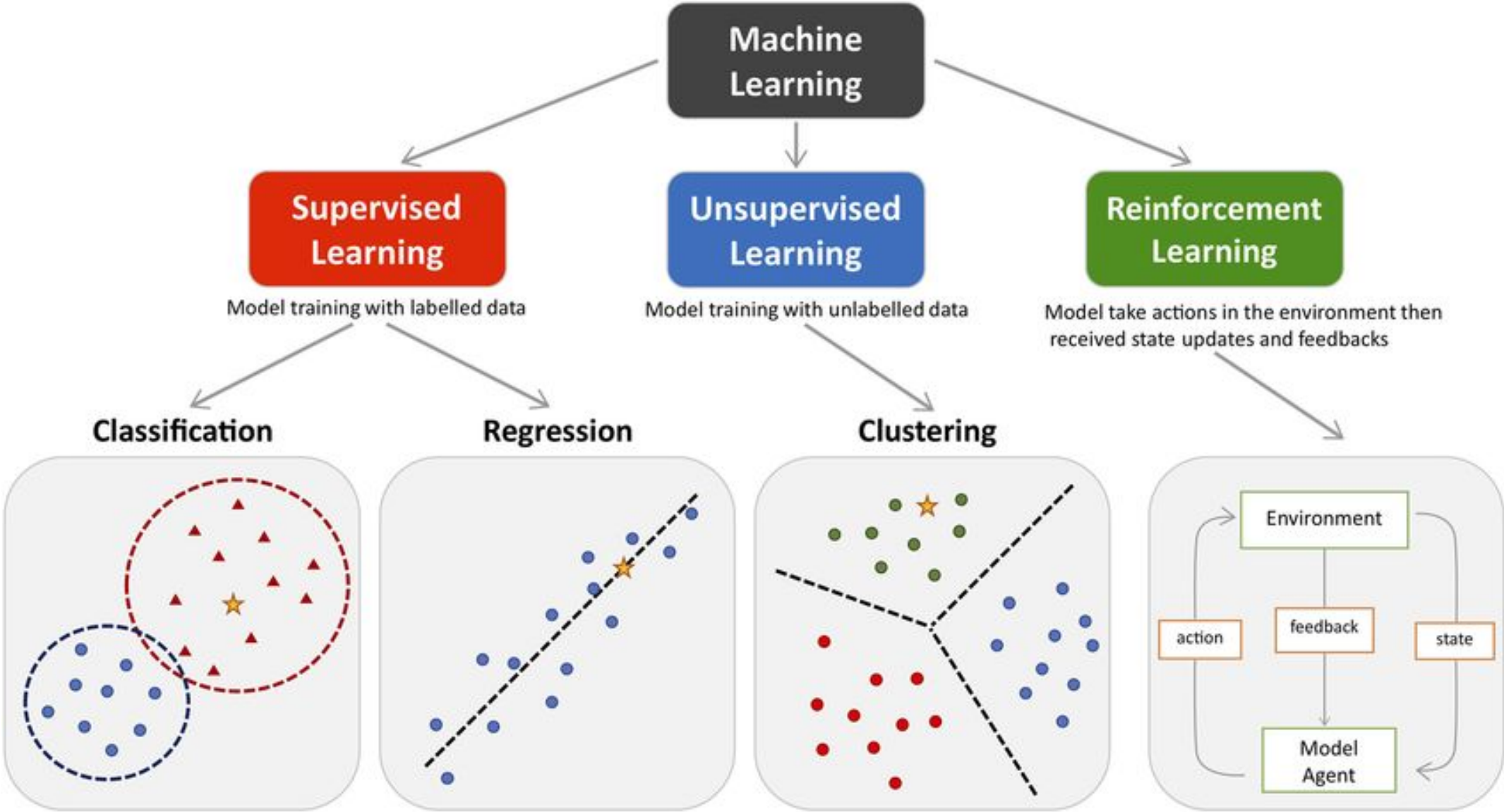
Reinforcement Learning

Definition: Reinforcement learning is a type of ML where an agent learns by interacting with the environment and receiving feedback through rewards or punishments.

Example: Training a robot to navigate through a maze.

Applications:

- Autonomous vehicles
- Game playing (e.g., AlphaGo)
- Robotics


Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Model training with labelled data
Model training with unlabelled data
Model take actions in the environment then received state updates and feedbacks
Classification
Regression
Clustering
Environment
action
feedback
state
Model Agent

The Curse of Dimensionality:

Definition: The **curse of dimensionality** refers to the challenges and issues that arise when analyzing and organizing data in high-dimensional spaces (**with many features or variables**).

Key Issues:

- **Data Sparsity:** As the number of features increases, the data becomes sparse, making it harder to find meaningful patterns.
- **Distance Metrics:** In high-dimensional spaces, the distance between data points becomes less informative, making clustering or classification less effective.
- **Overfitting:** Higher dimensions lead to overfitting since the model may fit the noise in the data.

Mitigation:

Dimensionality Reduction: Use techniques like **Principal Component Analysis (PCA)** to reduce the number of features while retaining the important information.

Overfitting and Underfitting:

Overfitting:

Definition: Overfitting occurs when a model is too complex and captures noise or random fluctuations in the training data. As a result, the model performs very well on the training data but poorly on unseen data (test set).

Symptoms:

- Very low error on the training set.
- High error on the test set.

Causes:

- Too complex model (e.g., too many parameters, high-degree polynomial regression).
- Insufficient training data.

Solutions:

- Use simpler models (e.g., reducing the number of features or parameters).
- Use regularization techniques (e.g., L1/L2 regularization).
- Increase the amount of training data.

Underfitting:

Definition: Underfitting occurs when the model is too simple to capture the underlying patterns in the data. The model performs poorly both on the **training set and the test set.**

Symptoms:

High error on both training and test sets.

Causes:

- Too simple model (e.g., linear regression when the data has non-linear relationships).
- Not enough features in the model.

Solutions:

- Use more complex models.
- Include more relevant features.

Model Selection:

Definition: Model selection is the process of choosing the best machine learning model for a given problem. It involves **selecting an algorithm** and **tuning its parameters** to achieve the best performance.

Approaches:

Cross-Validation: Use techniques like **k-fold cross-validation** to evaluate the model on different subsets of the data.

Hyperparameter Tuning: Use grid search or random search to find the best set of hyperparameters for the chosen model.

Bias-Variance Trade-off: Choose a model that strikes a balance between bias (underfitting) and variance (overfitting).

Error Analysis and Validation:

Error Analysis:

Analyze the errors of your model to understand where it is going wrong.

Common techniques include:

Confusion Matrix: Used for classification tasks to show the performance of a classification model.

Residual Analysis: For regression models, analyze the residuals (**the difference between the actual and predicted values**) to understand the model's performance.

Model Validation:

- **Hold-out Validation:** Split data into training and test sets. Train the model on the training set and validate it on the test set.
- **Cross-validation:** Split the data into multiple folds and train and validate the model on each fold.
- **Validation Set:** Set aside a portion of the training data to validate the model during training.

Parametric vs. Non-Parametric Models:

Parametric Models:

Definition: Parametric models **make assumptions about the underlying data distribution** and have a fixed number of parameters.

Examples: Linear Regression, Logistic Regression, Naive Bayes.

Advantages:

- Less computationally expensive.
- Easier to interpret.

Disadvantages:

May not perform well if the assumption about the data distribution is wrong.

Non-Parametric Models:

Definition: Non-parametric models **do not make strong assumptions about the data distribution** and do not have a fixed number of parameters. They are more flexible in terms of data modeling.

Examples: K-Nearest Neighbors (KNN), Decision Trees, Random Forests.

Advantages:

- Can model complex, non-linear relationships.
- Do not require assumptions about the data.

Disadvantages:

- More computationally expensive.
- Can suffer from overfitting if not tuned properly.

Example: Parametric Model: Linear Regression

Scenario: Predicting a student's exam score based on the number of hours they studied.

Assumption: There is a linear relationship between hours studied () and exam score ().

$$Y = 5X + 50$$

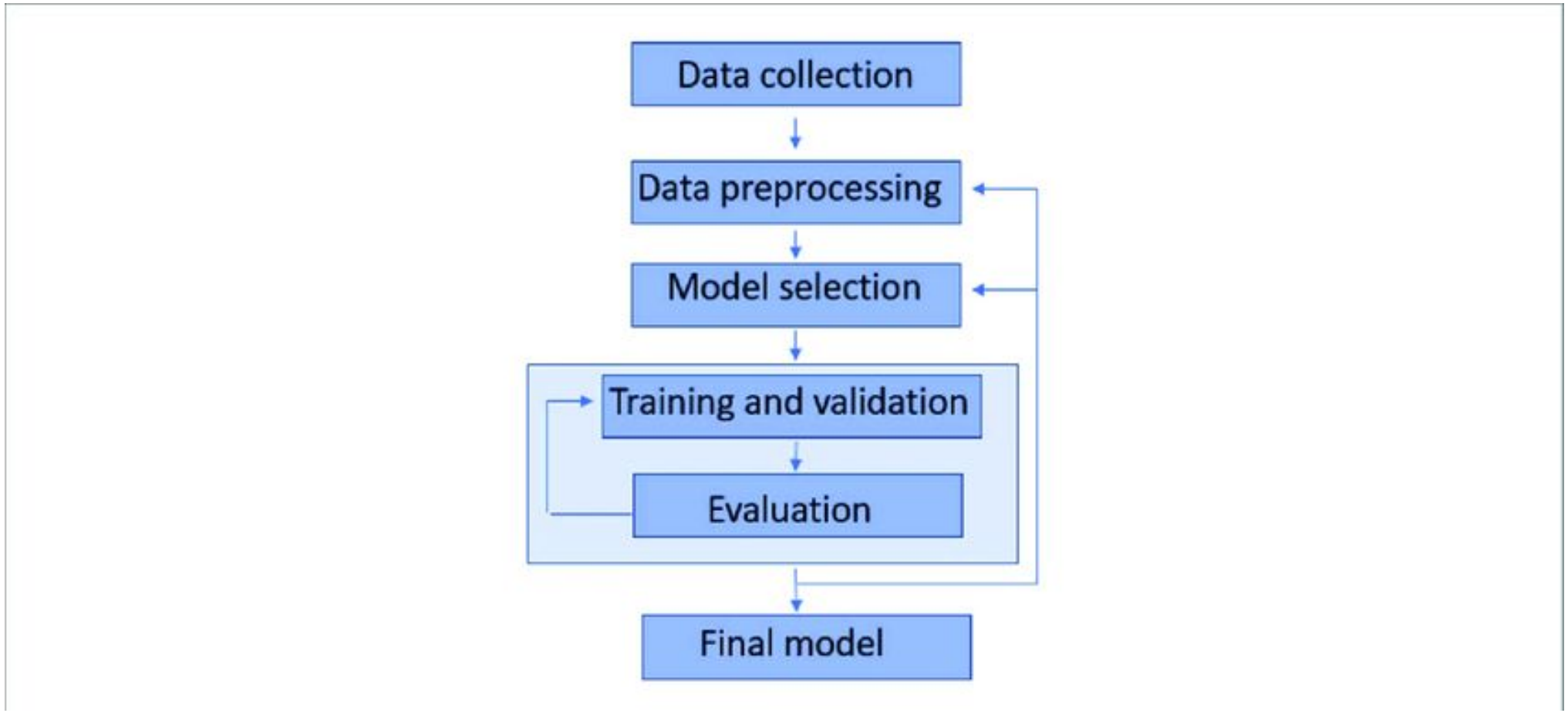
For every additional hour studied, the score increases by 5 points. Even with minimal data (e.g., 10 students), this model assumes the same relationship applies universally.

Example: Non-Parametric Model: k-Nearest Neighbors (k-NN)

Scenario: Predicting a student's exam score based on hours studied, but without assuming a specific form of the relationship.

Stores all the training data (e.g., scores of 50 students with their study hours). To predict a new student's score, it looks at the scores of the $k$closest students (e.g., those who studied a similar number of hours).

Example:
A new student studied for 4 hours.
The model looks at the **3 students who studied closest to 4 hours** and predicts the score based on their average.


Data collection
Data preprocessing
Model selection
Training and validation
Evaluation
Final model

Regression

What is Regression?

- Regression is a **supervised learning technique** used to model the relationship between input features (independent variables) and a continuous target variable (dependent variable).

- It predicts a numerical outcome based on the given **input data**.

Example: Predicting house prices based on features like size, location, and number of rooms.

Linear Regression

Definition:

- Linear regression is one of the simplest and most widely used regression techniques.

- It models the relationship between the dependent variable **Y** and one or more independent variables **X** by fitting a linear equation:

$$y = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

- $w_1, w_2, \ldots, w_n$: Weights or coefficients of the features.
- b: Bias term (intercept).
- $x_1, x_2, \ldots, x_n$: Features or predictors.

Types:

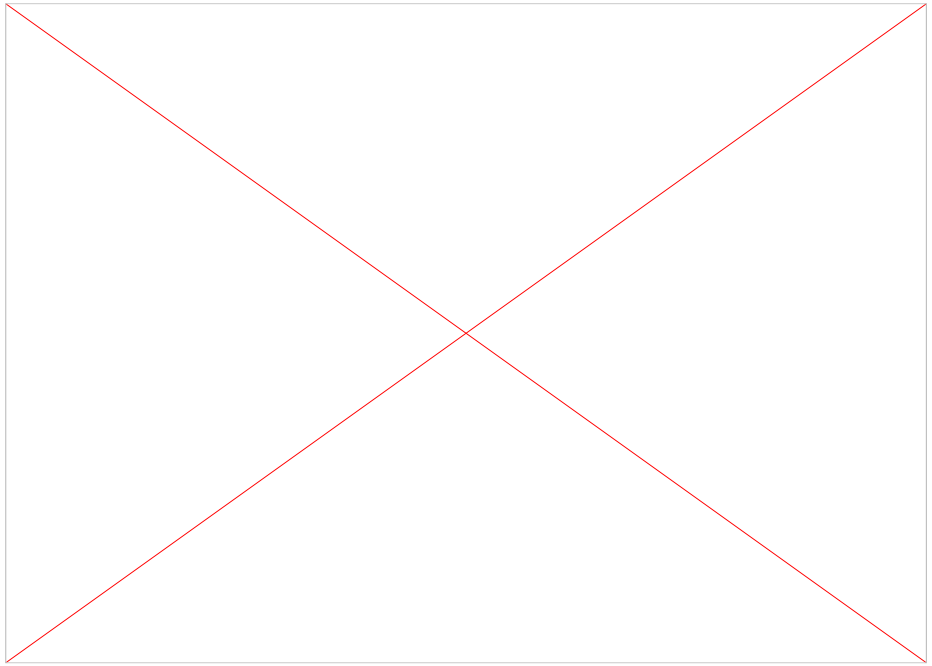
Simple Linear Regression:

Involves a single independent variable.

$$y = wx + b$$

Multiple Linear Regression:

Involves multiple independent variables.



Intuition Behind Linear Regression

- The goal is to find the best-fitting line (or hyperplane in higher dimensions) that minimizes the error between the **predicted values Y-** and the **actual values Y**

- The **"best fit"** is achieved by optimizing the weights ***w*** and bias ***b*** to minimize the cost function.

$$y = wx + b$$

Example:

Suppose you have data of house sizes **X** and their prices **Y**:

A linear regression model predicts house prices based on size by finding a straight line that best matches the data.

Cost Function

Purpose:

- The cost function measures the error or difference between the predicted values **Y-** and the actual values **Y.**

- In linear regression, we use the **Mean Squared Error (MSE)** as the cost function.

Mean Squared Error (MSE):

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

Where:

- m: Number of training examples.
- $\hat{y}_i$: Predicted value for the i^{th} training example.
- y_i: Actual value for the i^{th} training example.

Why Use MSE?

- Squaring the errors ensures they are positive, avoiding cancellations between positive and negative errors.
- Larger errors are penalized more heavily, encouraging the model to minimize big mistakes.

Minimizing the Cost Function:

- The goal of training the model is to find **w** and ***b*** such that the cost function ***J(w,b)*** is minimized.
- Optimization techniques like **Gradient Descent** are used for this purpose.

Example Workflow of Linear Regression

- Collect training data ***(X, y).***
- Initialize ***w*** and ***b*** randomly.
- Compute the cost function ***J(w,b)*** using **MSE**
- Update **w** and **b** iteratively to reduce the cost function (using **Gradient Descent**).
- Evaluate the model using new data.

Gradient Descent for Linear Regression

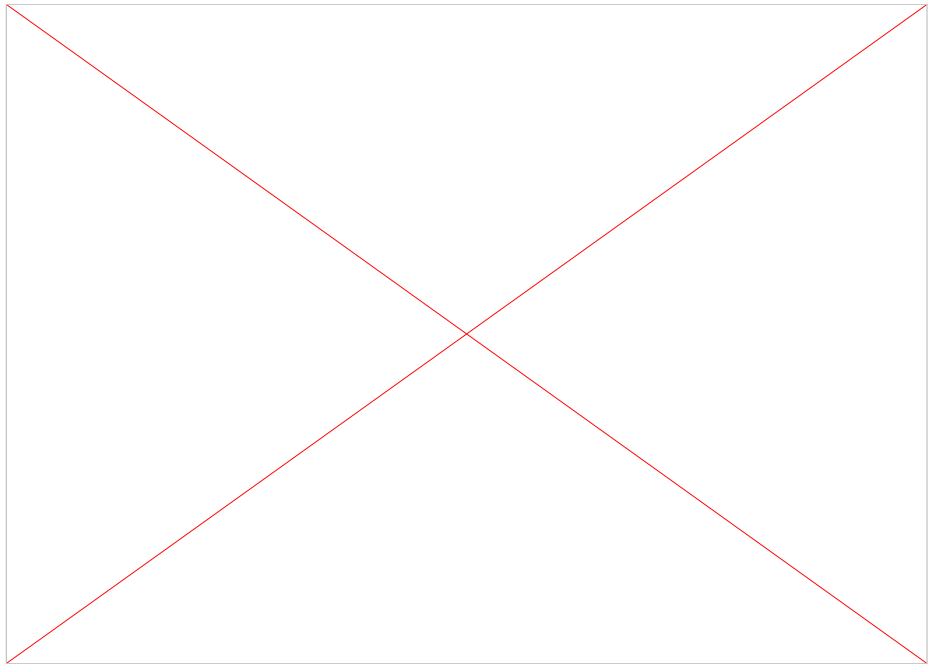
$Y_i = \beta_0 + \beta_1 X_i$

where Y_i = Dependent variable,

β_0 = constant/Intercept,

β_1 = Slope/Intercept,

X_i = Independent variable.

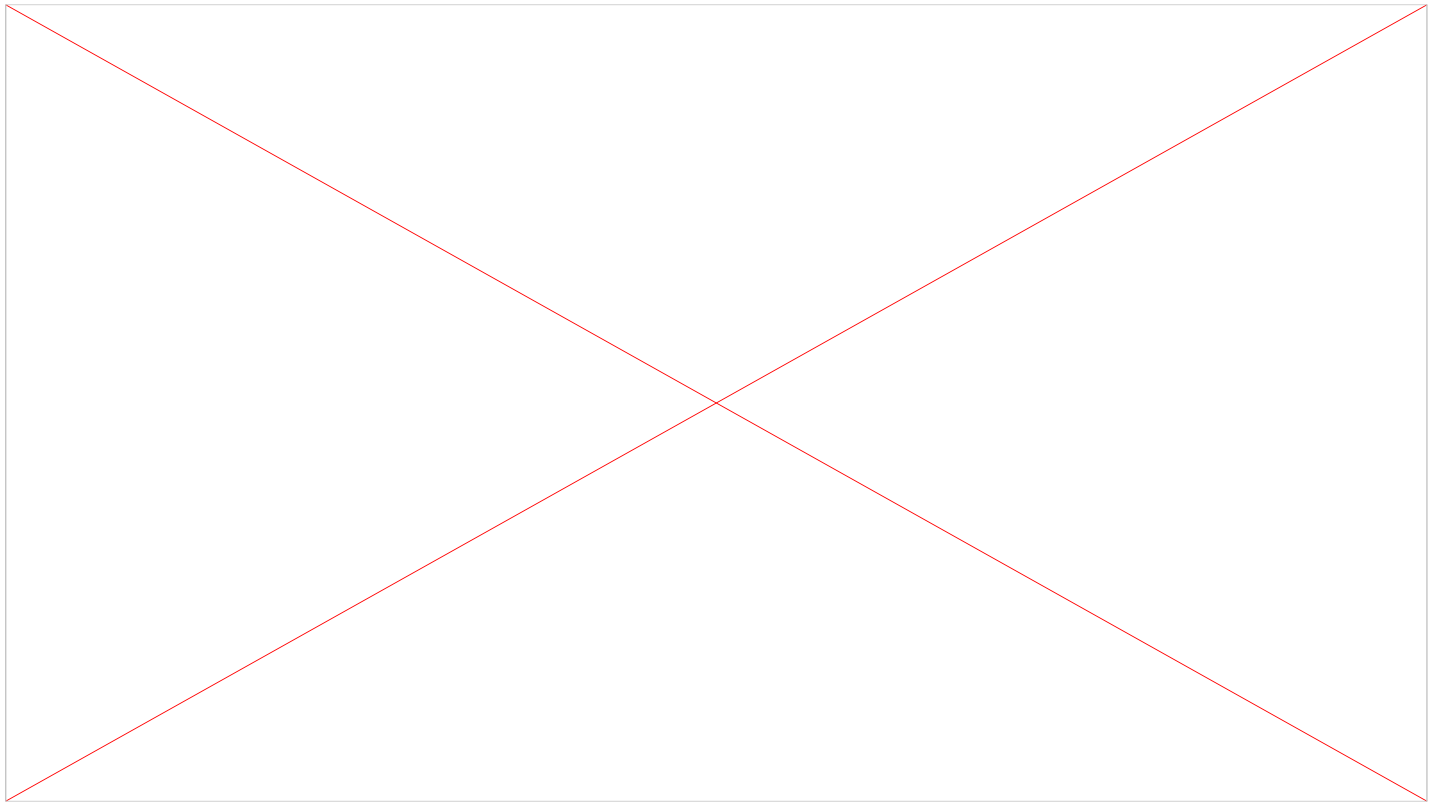


The goal of the linear regression algorithm is to get the best values for **B 0** and **B 1** to find the best-fit line.

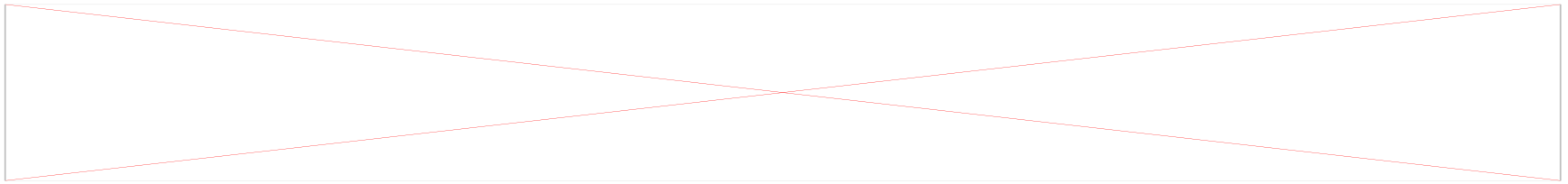
The best-fit line is a line that has the **least error** which means the error between predicted values and actual values should be minimum.

But how the linear regression finds out which is the best fit line?

- The goal of the linear regression algorithm is to get the best values for **B0 and B1** to find the best fit line.
- The **best fit line** is a line that has the least error which means the error between predicted values and actual values should be minimum.



We calculate MSE using the simple linear equation : **$Y_i = \beta_0 + \beta_1 X_i$**

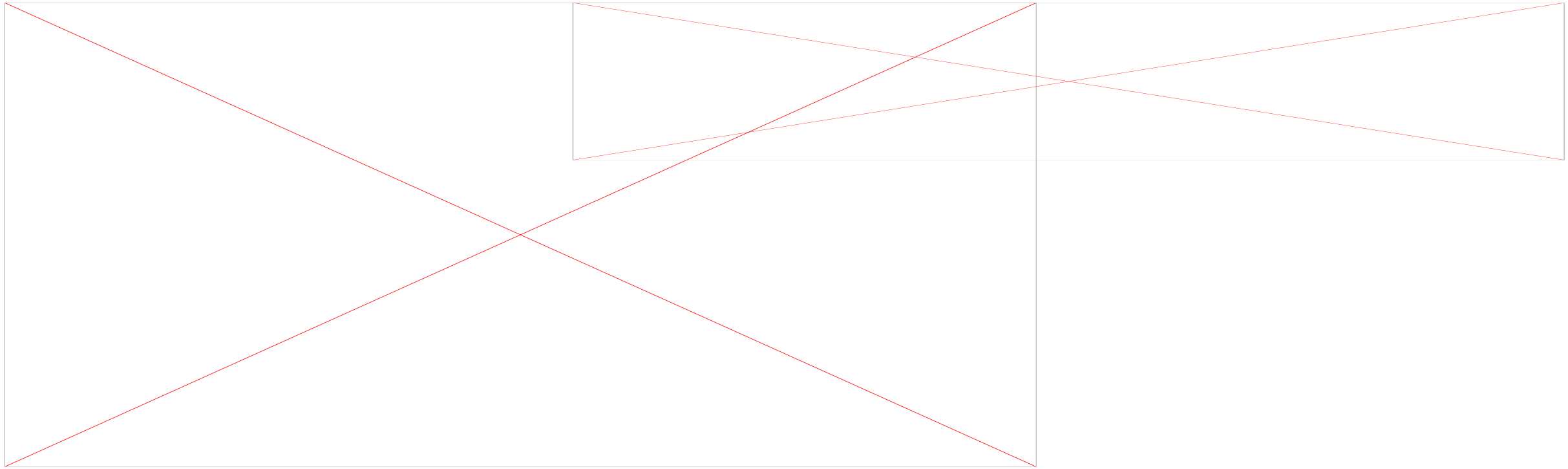


- Using the MSE function, we'll update the values of **B_0** and **B_1** such that the **MSE value settles at the minima.**

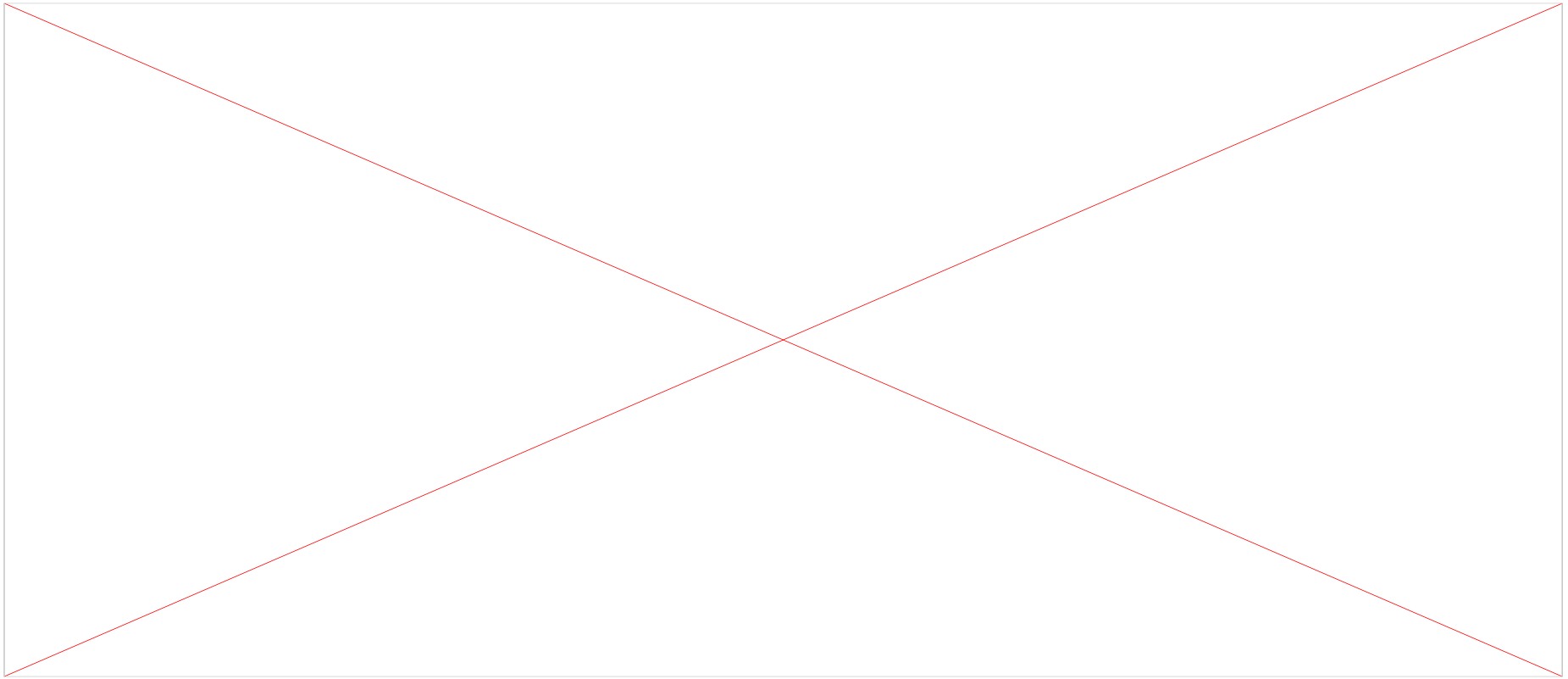
- These parameters can be determined using the gradient descent method such that the value for the cost function is minimum.

- **Gradient Descent** is one of the optimization algorithms that optimize the cost function (objective function) to reach the optimal minimal solution.

- To find the optimum solution, we need to reduce the **cost function (MSE)** for all data points. This is done by updating the values of the **slope coefficient (B1) and the constant coefficient (B0)** iteratively until we get an optimal solution for the linear function.



A **regression model** optimizes the gradient descent algorithm to update the coefficients of the line by reducing the cost function by randomly selecting coefficient values and then iteratively updating the coefficient values to reach the minimum cost function.



In the gradient descent algorithm, the number of steps you're taking can be considered as the **learning rate**, and this decides how fast the algorithm converges to the minima.

Problem Statement:

A company collects data about the number of hours employees work (`X`) and their productivity score (`Y`). The goal is to predict the productivity score for a new employee who works 7 hours using a simple linear regression model.

Data:

Hours Worked (X)	Productivity Score (Y)
1	2
2	4
3	5
4	4
5	5

The regression equation is:

$$Y = mX + c$$

Where:

- m = slope of the line
- c = y-intercept

2. Calculate m (Slope):

The formula for slope is:

$$m = \frac{N\sum(XY) - \sum X \sum Y}{N\sum(X^2) - (\sum X)^2}$$

- $N = 5$ (Number of data points)
- $\sum X = 1 + 2 + 3 + 4 + 5 = 15$
- $\sum Y = 2 + 4 + 5 + 4 + 5 = 20$
- $\sum (XY) = (1 \cdot 2) + (2 \cdot 4) + (3 \cdot 5) + (4 \cdot 4) + (5 \cdot 5) = 2 + 8 + 15 + 16 + 25 = 66$
- $\sum (X^2) = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 1 + 4 + 9 + 16 + 25 = 55$

Substitute into the formula:

$$m = \frac{5(66) - (15)(20)}{5(55) - (15)^2}$$

$$m = \frac{330 - 300}{275 - 225}$$

$$m = \frac{30}{50} = 0.6$$

The formula for intercept is:

$$c = \frac{\sum Y - m \sum X}{N}$$

Substitute values:

$$c = \frac{20 - 0.6(15)}{5}$$

$$c = \frac{20 - 9}{5} = \frac{11}{5} = 2.2$$

4. Final Regression Equation:

$$Y = 0.6X + 2.2$$

5. Predict Productivity for $X = 7$:

Substitute $X = 7$ into the equation:

$$Y = 0.6(7) + 2.2$$

$$Y = 4.2 + 2.2 = 6.4$$

Derivation of m and c in Linear Regression

The regression equation is:

$$Y = mX + c$$

where m is the slope and c is the intercept. These are derived using the **least squares method**, which minimizes the sum of squared errors (SSE).

Objective

The objective is to minimize:

$$SSE = \sum_{i=1}^{N} \left(Y_i - \hat{Y}_i\right)^2$$

where:

$$\hat{Y}_i = mX_i + c$$

Expanding:

$$SSE = \sum_{i=1}^{N} (Y_i - (mX_i + c))^2$$

Minimizing SSE

To find the values of m and c, take partial derivatives of SSE with respect to m and c, and set them to zero.

Step 1: Minimize with respect to c

$$\frac{\partial SSE}{\partial c} = \sum_{i=1}^{N} -2\left(Y_i - mX_i - c\right)$$

Set $\frac{\partial SSE}{\partial c} = 0$:

$$\sum_{i=1}^{N} \left(Y_i - mX_i - c\right) = 0$$

$$\sum_{i=1}^{N} Y_i = m \sum_{i=1}^{N} X_i + cN$$

Rearranging:

$$c = \frac{\sum Y - m \sum X}{N}$$

Step 2: Minimize with respect to m

$$\frac{\partial SSE}{\partial m} = \sum_{i=1}^{N} -2X_i \left(Y_i - mX_i - c\right)$$

Set $\frac{\partial SSE}{\partial m} = 0$:

$$\sum_{i=1}^{N} X_i \left(Y_i - mX_i - c\right) = 0$$

$$\sum_{i=1}^{N} X_i Y_i = m \sum_{i=1}^{N} X_i^2 + c \sum_{i=1}^{N} X_i$$

Substitute $c = \frac{\sum Y - m \sum X}{N}$ into the equation and simplify. After solving, the formula for m is:

$$m = \frac{N \sum (X_i Y_i) - \sum X_i \sum Y_i}{N \sum (X_i^2) - (\sum X_i)^2}$$

Step 3: Solve for c

Using the value of m, substitute it back into:

$$c = \frac{\sum Y - m \sum X}{N}$$

Final Formulas

The slope m and intercept c are:

$$m = \frac{N \sum (X_i Y_i) - \sum X_i \sum Y_i}{N \sum (X_i^2) - (\sum X_i)^2}$$

$$c = \frac{\sum Y - m \sum X}{N}$$

Multiple Linear Regression (MLR)

Introduction

- **Multiple Linear Regression** (MLR) is a statistical technique used to model the relationship between **one dependent variable (target) and two or more independent variables (predictors).**

- The goal is to find the **linear equation that best predicts the dependent variable** using the independent variables.

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n + \epsilon$$

Where:

- Y: Dependent variable (response).
- $X_1, X_2, \ldots, X_n$: Independent variables (predictors).
- b_0: Intercept (value of Y when all $X_i = 0$).
- $b_1, b_2, \ldots, b_n$: Coefficients of the independent variables.
- ϵ: Error term (captures the deviations of observed values from predicted values).

Assumptions of MLR

Linearity: The relationship between dependent and independent variables is linear.

Independence: Observations are independent of each other.

Homoscedasticity: The variance of residuals is constant across all levels of the independent variables.

No Multicollinearity: Independent variables are not highly correlated with each other.

Normality: Residuals (errors) are normally distributed.

Applications

- Predicting housing prices based on features like size, location, and number of rooms.
- Forecasting sales using advertising spend on different platforms.
- Estimating the impact of multiple factors on employee performance

Types of Gradient Descent

Batch, Stochastic, and Mini-Batch Gradient Descent

Gradient Descent is an optimization algorithm used to minimize the cost function by iteratively moving in the direction of the steepest descent (negative gradient).

Types of Gradient Descent

1. Batch Gradient Descent

2. Stochastic Gradient Descent (SGD)

3. Mini-Batch Gradient Descent

Batch Gradient Descent

Uses the entire dataset to compute the gradient.

- Advantages: Stable updates, converges to global minimum.
- Disadvantages: Slow for large datasets, high memory usage.
- Best for small datasets.

Stochastic Gradient Descent (SGD)

Uses one random sample to compute the gradient per iteration.

- Advantages: Faster updates, suitable for large datasets.
- Disadvantages: Noisy updates, may oscillate around minimum.
- Best for online learning and large-scale data.

Mini-Batch Gradient Descent

Splits the dataset into small batches for gradient computation.

- Advantages: Faster convergence, balanced approach.
- Disadvantages: Requires batch size tuning, slight oscillations.
- Most commonly used in deep learning.

Comparison of Gradient Descent Types

- Batch GD:
- Update: Once per epoch
- Stable but slow.

- SGD:
- Update: Once per sample
- Fast but noisy.

- Mini-Batch GD:
- Update: Once per batch
- Balanced and efficient.

Applications

- Batch GD: Small datasets (e.g., linear regression)
- SGD: Online learning (e.g., recommender systems).
- Mini-Batch GD: Deep learning models.

Normalization and Standardization

Normalization:

Rescales the values of a dataset to a range of [0, 1] or [-1, 1].

$$x_{\text{normalized}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Standardization:

Transforms data to have a mean (μ) of 0 and a standard deviation (σ) of 1.

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

Why Are These Important?

Normalization:

- Useful for algorithms like k-Nearest Neighbors (k-NN) and Neural Networks that are sensitive to the scale of data.
- Ensures all features contribute equally to the model.

Standardization:

- Essential for models assuming Gaussian distribution or sensitive to feature magnitude, such as Support Vector Machines (SVM), Principal Component Analysis (PCA), and Linear Regression.
- Improves convergence speed during optimization in Gradient Descent.

Normalization Example:

Feature 1 (Height in cm): [150, 160, 170, 180, 190]
Feature 2 (Weight in kg): [50, 60, 70, 80, 90]

Normalized Data:

Height: [0, 0.25, 0.5, 0.75, 1]
Weight: [0, 0.25, 0.5, 0.75, 1]

Standardization Example:

Feature (Scores): [50, 60, 70, 80, 90]
Mean = 70
Standard Deviation = 15.81

Standardized Data:

[−1.26,−0.63,0,0.63,1.26]
[−1.26,−0.63,0,0.63,1.26]

Useful for SVM or PCA as it makes the feature mean-centered.

PYTHON
CODE OF LINEAR REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

X = np.array([1, 2, 3, 4, 5,7,8,9,10,15,17,19,21]).reshape(-1, 1)
y = np.array([2, 4, 5, 4, 5,7,9,10,12,15,18,20,25])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=32)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

plt.scatter(X, y, color="blue", label="Data Points")
plt.plot(X, model.predict(X), color="red", label="Best Fit Line")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Linear Regression: Best Fit Line")
plt.legend()
plt.show()

new_data = np.array([[30]])  # Replace with your input
predicted_output = model.predict(new_data)
print(f"Predicted output for {new_data.flatten()}: {predicted_output[0]:.2f}")
```

Introduction to Overfitting and Underfitting

- Overfitting and underfitting are common problems in machine learning that affect model performance.
- They arise from the trade-off between a model's complexity and its ability to generalize to new, unseen data.

Underfitting:

A model is underfitting when it is too simple to capture the underlying patterns in the data.

Symptoms:

- High training error.
- High testing error.

Causes:

- Model lacks complexity.
- Insufficient training.
- Features are not informative enough.

Example: Using a linear regression model to fit non-linear data

Overfitting:

A model is overfitting when it learns not only the patterns but also the noise in the training data.

Symptoms:

- Low training error.
- High testing error.

Causes:

- Model is too complex.
- Insufficient training data for the model's complexity.
- No regularization applied.

Example: A decision tree with very deep splits that memorizes the training data.

Example predicting house prices:

Underfitting: Using only the size of the house as a feature while ignoring other factors like location, number of bedrooms, etc.

Overfitting: Including irrelevant details like the color of the walls, which does not generalize to new data.

What is Regularization?

- Regularization is a technique to **prevent overfitting in machine learning models**.
- It adds a penalty term to the loss function to shrink the magnitude of coefficients, ensuring the model generalizes well to new data.

Lasso Regularization

- Least Absolute Shrinkage and Selection Operator.
- Penalty Term: Adds the L1 norm (sum of absolute values of coefficients) to the loss function.

Formula:

$$\text{Loss Function} = \text{MSE} + \alpha \cdot \sum |\beta|$$

Where:

- α: Regularization strength (higher α = stronger penalty).
- β: Model coefficients.

Key Features:

- Shrinks coefficients.
- Feature Selection: Some coefficients are reduced to exactly zero, effectively removing irrelevant features.

When to Use:

When you suspect some features are irrelevant and want to automatically select the most important ones.

Ridge Regularization

Penalty Term: Adds the L2 norm (sum of squares of coefficients) to the loss function.

Formula:

$$\text{Loss Function} = \text{MSE} + \alpha \cdot \sum \beta^2$$

Key Features:

- Shrinks coefficients but does not make them zero.
- Retains all features and is better for handling multicollinearity (correlated features).

When to Use:

When all features are expected to contribute to the target variable, even if minimally.

Lasso Regularization: Feature Selection in Predictive Modeling

Example:

A healthcare company is building a predictive model to estimate a patient's risk of developing diabetes based on features like age, BMI, cholesterol levels, glucose levels, and hundreds of genetic markers.

Problem:

- Many features (like certain genetic markers) might have little to no effect on the outcome.
- Including all features increases model complexity and could lead to overfitting.

Solution:

- Apply **Lasso regularization** to automatically reduce irrelevant features' coefficients to 0.
- The model keeps only the most important features, making it simpler and more interpretable for clinicians.

Ridge Regularization: Improving Predictions in Multicollinear Data

Example:

An e-commerce company wants to predict product demand based on factors like price, discount offered, advertising spend, and competitor pricing.

Problem:

- Some features (e.g., "price" and "discount offered") are highly correlated, leading to multicollinearity.
- Standard linear regression struggles in such scenarios, resulting in unstable coefficient estimates.

Solution:

- Use **Ridge regularization**, which penalizes large coefficients without dropping any features.
- Ridge shrinks the correlated feature coefficients, stabilizing predictions while retaining all information.

Dataset: Predict house prices based on features like size, location, and age.

Lasso: Automatically drops less relevant features like age if size and location explain most of the variance.

Ridge: Keeps all features and distributes importance across them, even if age has minimal impact.

Lasso = Simpler, interpretable models (use when feature selection is needed).

Ridge = Stable, robust predictions (use when all features are useful).

Real-Time Example: Lasso Regularization

We aim to predict house prices (y) using the following features:

- **Feature 1:** Size of the house (in square feet).
- **Feature 2:** Age of the house (in years).
- **Feature 3:** Number of bedrooms.

The coefficients for the features are:

$$\beta = [100, 50, 30]$$

This means:

- For every square foot, the price increases by 100.
- For every year of age, the price decreases by 50.
- For every bedroom, the price increases by 30.

Step 1: Linear Regression (No Regularization)

The **Mean Squared Error (MSE)** for predicting house prices is given by:

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

Where:

- y_i: Actual price of house i.
- $\hat{y}_i = X_i \cdot \beta$: Predicted price of house i based on the features.

Linear regression minimizes only the **MSE** without considering the size of the coefficients β.

Step 2: Lasso Regression (With Regularization)

Lasso modifies the loss function by adding the **L1 penalty**:

$$\text{Loss} = \text{MSE} + \alpha \cdot \sum |\beta_i|$$

Assume:

- $\alpha = 0.1$ (regularization strength),
- $\beta = [100, 50, 30]$.

Penalty Calculation:

$$\sum |\beta_i| = |100| + |50| + |30| = 180$$

$$\text{Penalty Term} = \alpha \cdot \sum |\beta_i| = 0.1 \cdot 180 = 18$$

The total loss is:

$$\text{Loss} = \text{MSE} + 18$$

Step 3: Effect of Lasso on Coefficients

During training, Lasso tries to reduce the total loss. This can shrink some coefficients towards zero if they are less important. For example:

- Suppose the coefficient for **Age** ($\beta_2 = 50$) contributes little to reducing the MSE.
- Lasso shrinks β_2 to **0**, effectively removing it from the model.

The updated coefficients become:

$$\beta = [100, 0, 30]$$

Numerical Example

Dataset:

Size (sq ft)	Age (years)	Bedrooms	Actual Price (y)
1000	10	3	150,000
1500	20	4	200,000
1200	5	3	180,000

Without Regularization (Linear Regression):

The predicted price is:

$$\hat{y} = 100 \cdot \text{Size} + 50 \cdot \text{Age} + 30 \cdot \text{Bedrooms}$$

For the first house:

$$\hat{y} = 100 \cdot 1000 + 50 \cdot 10 + 30 \cdot 3 = 15\,\text{[illegible]}00$$

1000590

With Lasso Regularization:

If Lasso determines that $\beta_2 = 50$ (Age) contributes minimally, it will shrink β_2 to **0**. The new prediction formula becomes:

$$\hat{y} = 100 \cdot \text{Size} + 30 \cdot \text{Bedrooms}$$

For the first house:

$$\hat{y} = 100 \cdot 1000 + 30 \cdot 3 = 15\,\,090$$

100090

Real-Time Example: Ridge Regularization

We aim to predict house prices (y) using the following features:

- **Feature 1:** Size of the house (in square feet).
- **Feature 2:** Age of the house (in years).
- **Feature 3:** Number of bedrooms.

The coefficients for the features are:

$$\beta = [100, 50, 30]$$

This means:

- For every square foot, the price increases by 100.
- For every year of age, the price decreases by 50.
- For every bedroom, the price increases by 30.

Step 2: Ridge Regression (With Regularization)

Ridge modifies the loss function by adding the **L2 penalty**:

$$\text{Loss} = \text{MSE} + \alpha \cdot \sum \beta_i^2$$

Penalty Calculation:

- Assume $\alpha = 0.1$ (regularization strength).
- $\beta = [100, 50, 30]$.

$$\sum \beta_i^2 = (100)^2 + (50)^2 + (30)^2 = 10000 + 2500 + 900 = 13400$$

$$\text{Penalty Term} = \alpha \cdot \sum \beta_i^2 = 0.1 \cdot 13400 = 1340$$

The total loss is:

$$\text{Loss} = \text{MSE} + 1340$$

Numerical Example

Dataset:

Size (sq ft)	Age (years)	Bedrooms	Actual Price (y)
1000	10	3	150,000
1500	20	4	200,000
1200	5	3	180,000

Without Regularization (Linear Regression):

The predicted price is:

$$\hat{y} = 100 \cdot \text{Size} + 50 \cdot \text{Age} + 30 \cdot \text{Bedrooms}$$

For the first house:

$$\hat{y} = 100 \cdot 1000 + 50 \cdot 10 + 30 \cdot 3 = 15\,\,00$$

1000590

With Ridge Regularization:

If Ridge reduces the coefficients to $\beta = [95, 45, 28]$, the new prediction formula becomes:

$$\hat{y} = 95 \cdot \text{Size} + 45 \cdot \text{Age} + 28 \cdot \text{Bedrooms}$$

For the first house:

$$\hat{y} = 95 \cdot 1000 + 45 \cdot 10 + 28 \cdot 3 = 150[illegible]84$$

95534

Bias, Variance, and Tradeoff

Bias is when the model is too simple and cannot capture the pattern in the data properly. This leads to underfitting.

Example

Imagine you are trying to guess someone's age??

Instead of analyzing details like height, hair color, and clothing

you always guess the same average age for everyone (e.g., 25 years).

This is a high bias guess because you ignore specific details and make overly simplistic assumptions.

Variance is when the model is too complex and tries to learn even the smallest details (noise) in the data. This leads to overfitting.

You try to guess someone's age again??

But this time you overanalyze everything:

1. She has long hair, so she must be 20.
2. Oh wait, she's wearing glasses, so maybe she's 22.
3. Wait, a slight wrinkle? Maybe 28.

This is high variance because you are too sensitive to small details that don't really matter.

Bias-Variance Tradeoff (Simple Analogy of Exam)

High Bias (Underfitting):

- You only study one chapter for the exam and ignore everything else.
- **Result:** You don't perform well because you miss most questions.
- ***Model is too simple, unable to capture the full syllabus.***

High Variance (Overfitting):

- You try to memorize every single word from the textbook, notes, and examples—even unnecessary details.
- **Result:** You get confused in the exam because you overanalyzed and didn't focus on what's important.
- ***Model is too complex and learns unnecessary details.***

Balanced Study (Optimal Tradeoff):

- You study all chapters thoroughly but focus on understanding the **key concepts** and main topics.
- **Result:** You perform well on the exam because you have the **right balance of preparation**.

Bias = Too simple → Underfitting → Misses the important details.

Variance = Too complex → Overfitting → Gets distracted by unnecessary details.

Goal: Find the right balance (Bias-Variance Tradeoff) so the model can generalize well to new data.