

# Computational Analytics

Dr. Arjun Anil Kumar

National Institute of Technology Calicut

April 6, 2025



# Outline

## 1 Introduction

## 2 Unsupervised Learning

- Clustering Algorithms
- Association Rules
- Recommendation System
- Word2Vec
- Principal Component Analysis

## 3 Supervised Learning

- Bayes Classification
- Linear Regression
- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machine SVM
- Neural Networks
- Bias & Variance
- Bias & Variance in ML Models

## 4 AI in 21<sup>st</sup> Century

- AI - Applications

## 5 Problems



# **Introduction**



# Introduction

## Artificial Intelligence (AI) and Machine Learning (ML)

ARTIFICIAL INTELLIGENCE

MACHINE LEARNING

SUPERVISED  
LEARNING

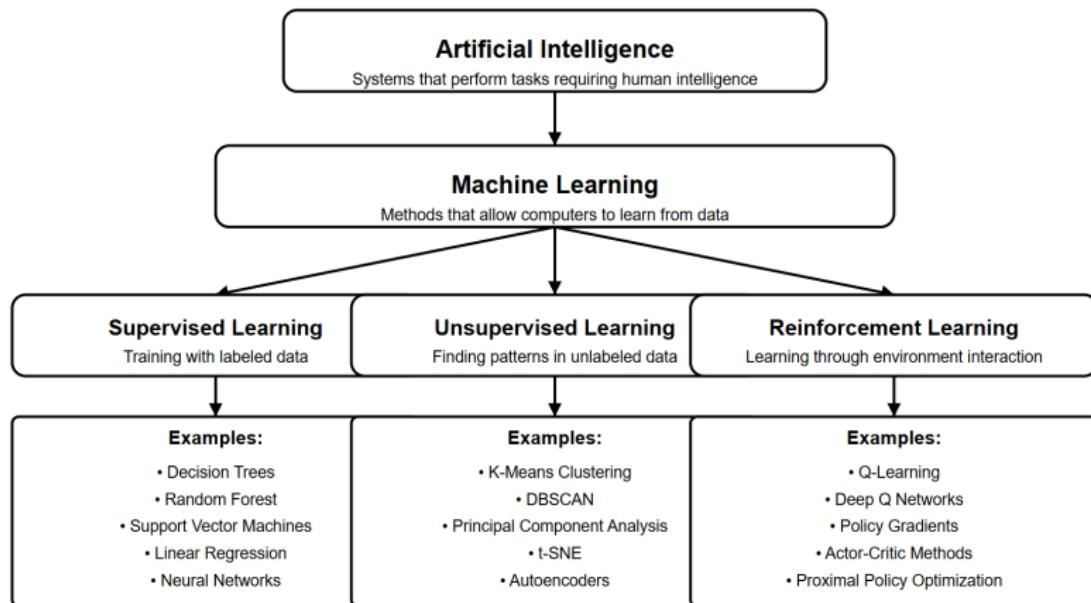
UNSUPERVISED  
LEARNING

REINFORCEMENT  
LEARNING



# Introduction

## Artificial Intelligence (AI) and Machine Learning (ML)



## Machine Learning Algorithms



# Introduction

## Machine Learning Definitions

### Machine Learning

Machine learning focuses on the use of **data** and **algorithms** to **imitate** the way that humans learn, gradually improving its accuracy.

### Machine Learning - Tom Mitchell

A computer program is said to learn from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with **experience E**.

### Machine Learning

A machine learning (ML) model is a mathematical **algorithm** that is **trained** on data to recognize patterns and make predictions or decisions without being explicitly programmed to perform the task. It **learns** from the input data and improves its performance over time as it is exposed to more data.

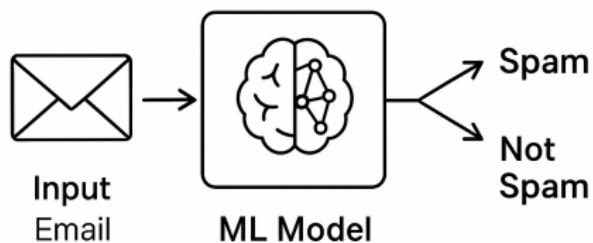


# Introduction

Machine Learning Isn't Rule Based!

## Spam Detection Problem

"What are the 200 rules to detect a spam mail?" Not practical!. ML learns patterns from data instead of rules from humans. For example : By training a ML model on 10,000 emails of spam and non-spam types, we can build a reasonable spam detection model that can predict without needing us to spell out all the rules.



Neural Networks for Spam Detection

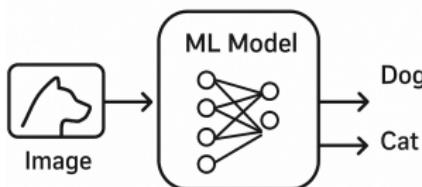


# Introduction

Machine Learning Isn't Rule Based!

## Image Recognition Problem

"What are the 100,000 rules to detect a cat in an image?" Not practical!. ML learns patterns from data instead of rules from humans. For example : By training an ML model on 10,000 pictures of cats and dogs, we can build a reasonable image recognition model that can predict without having to spell out all the rules.



Neural Networks for Dog/Cat Detection



# Introduction

## Supervised, Unsupervised and Reinforcement Learning

- **Supervised Learning** : In supervised learning, the model learns from labeled data, meaning each training example has an input (features) and a corresponding correct output (label or target). The goal is to learn a mapping from inputs to outputs. Eg : Customer Churn Prediction, Fraud Detection, Demand Forecasting.
- **Unsupervised Learning** : In unsupervised learning, the model is trained on unlabeled data, meaning there are no predefined correct outputs. The goal is to uncover patterns, relationships, or structures in the data. Eg : Customer Segmentation, Market Basket Analysis, Topic Modelling for Business Insights.
- **Reinforcement Learning<sup>1</sup>** : In reinforcement learning (RL), an agent learns by interacting with an environment and receiving rewards or penalties based on its actions. The goal is to maximize cumulative rewards over time. Eg : Dynamic Pricing based on Demand and Competition.

---

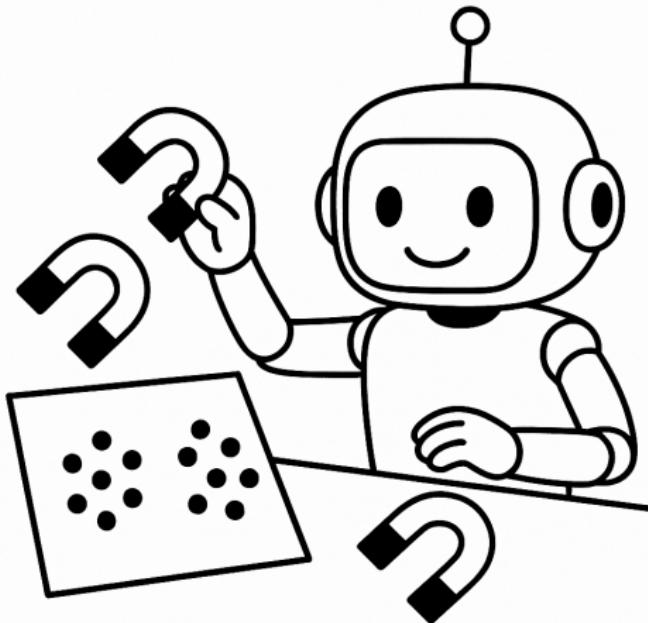
<sup>1</sup>Will not be discussed in this course



# **Unsupervised Learning**



# Unsupervised Learning



Unsupervised Learning



# Unsupervised Learning

## Feature Matrix X - Representation

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \in \mathbb{R}^{n \times m} = [X_{\text{features}}]$$

- The feature matrix  $X \in \mathbb{R}^{n \times m}$ , the unlabelled data set, represents the training samples X.
- $X_{\text{features}} \in \mathcal{R}^{m \times n}$  refers to feature data set, where each training sample (row) has m features.
- m indicates the number of features and n indicates the number of samples.
- Feature vector  $X_i$  indicates the  $i^{\text{th}}$  feature, where  $i \in [1, m]$ .

$$X_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{ni} \end{bmatrix} \in \mathbb{R}^m$$

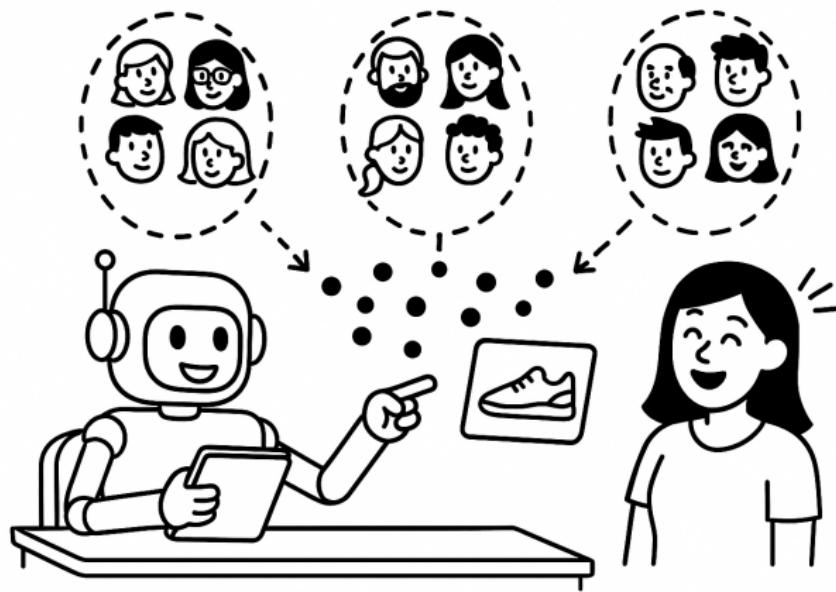


# **Clustering Algorithms**



# Clustering Algorithms

Application : Customer Segmentation



**Customer segmentation**

Customer Segmentation



# Clustering Algorithms

## Types of Clustering Algorithms

- **Partition Clustering** methods, like K-means clustering divide the dataset into a predefined number of clusters, minimizing within or intra-cluster variance.
- **Hierarchical Clustering** methods build a hierarchy of clusters either by merging smaller clusters (agglomerative) or splitting larger clusters (divisive).
  - 1 **Agglomerative Clustering:** Starts with each point (training instance) as its own cluster and merges clusters iteratively.
  - 2 **Divisive Clustering:** Starts with all points (training instances) in one cluster and splits them iteratively.



# Clustering Algorithms

## Advanced Clustering Algorithms

- **Density-based Clustering** methods, such as DBSCAN, OPTICS, HDBSCAN, form clusters based on the density of points in a region. They excel in finding clusters of arbitrary shapes and are **robust to noise and outliers**.
- **Model-based Clustering** methods, such as Gaussian Mixture Model (GMM), assume that data is generated from a mixture of probability distributions and estimate the parameters of these distributions.
- **Grid-based Clustering** methods, such as STING, CLIQUE, divide the data space into a finite number of cells or grids and then form clusters based on the density of points within each grid.
- **Spectral Clustering** uses the eigenvalues of a similarity matrix (derived from the data) to perform dimensionality reduction, followed by traditional clustering methods such as K-means clustering.



# K-means Clustering



# K-means Clustering

## K-means Clustering

### K-means Clustering

K-means clustering, an unsupervised learning method, aims to partition a feature dataset of  $n$  training samples into exact  $k$  distinct clusters, where each training sample belongs to the cluster with the nearest centroid.

The notations for K-means Clustering are

- $n$ : Number of training samples.
- $m$ : Number of features in each sample.
- $k$ : Number of clusters pre-defined by the user
- $X \in \mathcal{R}^{n*m}$ : Feature Matrix.
- $\mu_j \in \mathcal{R}^m$ : Centroid of cluster  $j$ , where  $j \in \{1, 2, \dots, k\}$ .



# K-means Clustering

## Objective

### K-means Clustering

The objective of K-means Clustering is to find the cluster centres  $\mu_j$  that minimizes the **within cluster variance or sum of squares (WCSS)**:

$$J = \sum_{j=1}^k \sum_{x^{(i)} \in C_j} \|x^{(i)} - \mu_j\|^2 \quad (1)$$

where:

- $C_j$ : The set of sample points belonging to cluster  $j$ ,  $j \in \{1, 2, \dots, k\}$ .
- $\|x^{(i)} - \mu_j\|^2$ : Squared Euclidean distance between a sample  $x_i \in C_j$  and the centroid  $\mu_j$ , where  $j \in \{1, 2, \dots, k\}$ .
- Minimizing WCSS does not guarantee maximization of between cluster sum of squares (BCSS)<sup>2</sup>.

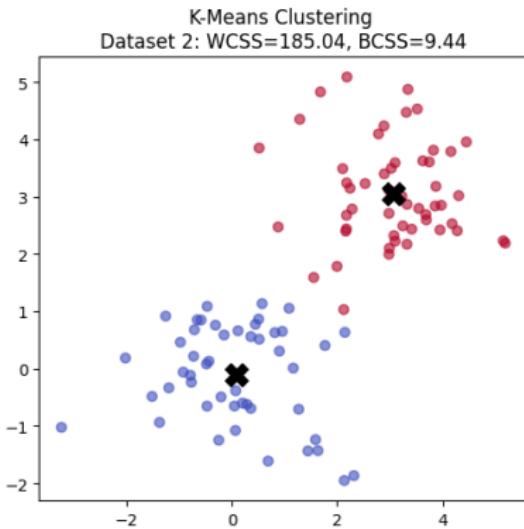
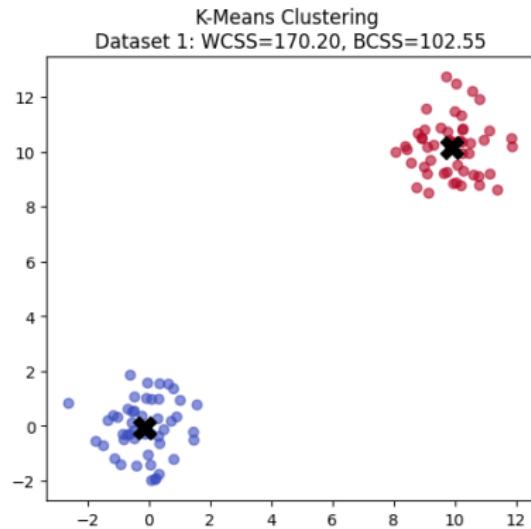
---

<sup>2</sup>Between cluster sum of squares BCSS is denoted by  $\sum_{j=1}^k n_j \|\mu_j - \mu\|^2$ , where  $n_j$  refers to the number of samples in each cluster  $j$  and  $\mu$  is the overall mean



# K-means Clustering

Illustration with 2-D data : WCSS & BCSS



K-Means Clustering with varying BCSS



# K-means Clustering

## K-means Algorithm

- 1 Initialize  $k$  centroids randomly, where each centroid  $\mu_j \in \mathbb{R}^m$ ,  $j \in \{1, 2, \dots, k\}$ .
- 2 Assign each data point  $x^{(i)}$  to the nearest cluster  $j$  based on the squared Euclidean distance  $\|x^{(i)} - \mu_j\|^2$ ,  $j \in \{1, 2, \dots, k\}$ .

$$j = \arg \min_j \|x^{(i)} - \mu_j\|^2 \quad (2)$$

- 3 Update each centroid  $\mu_j$  with the mean of all points assigned to cluster  $j$ , where  $|C_j|$  is the number of elements in the cluster:

$$\mu_j = \frac{1}{|C_j|} \sum_{x^{(i)} \in C_j} x^{(i)} \quad (3)$$

- 4 Repeat steps 2 and 3, until convergence of the cluster center(i.e., when the centroids no longer change significantly or a maximum number of iterations,as specified by the user, is reached).



# K-means Clustering

## K-means Clustering Illustration

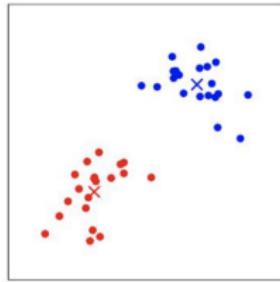
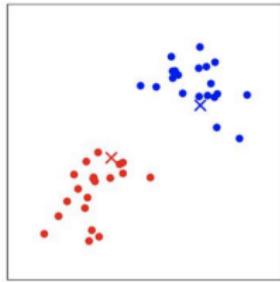
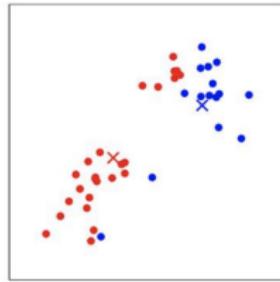
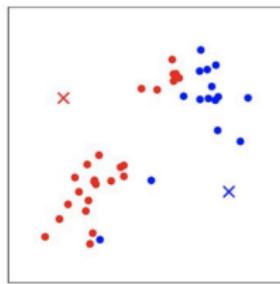
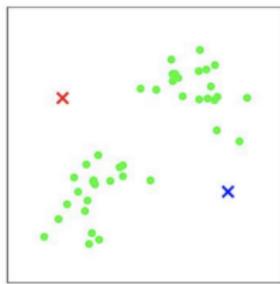
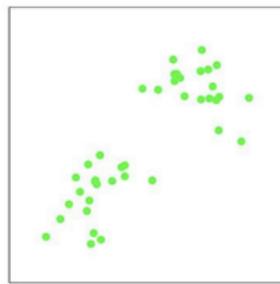


Illustration - K-means clustering for  $K = 2$



# K-Means Clustering

## Convergence

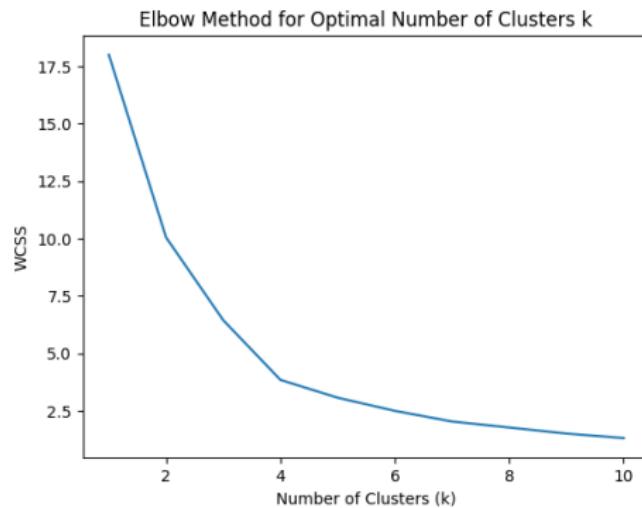
- K-means clustering is guaranteed to converge, but it does not always converge to the global optimum (i.e., the best clustering).
- The final cluster structure in K-means depends heavily on the choice of the initial centroids.
- The K-means algorithm would converge when there is no significant difference in the centroids in different iterations or a maximum number of iterations, as specified by the user, is reached.
- $n_{init}$  is a parameter that controls the number of times the algorithm will run with different initializations of centroids. Default value is 10.
- K-means clustering is performed  $n_{init}$  times and the initial centroid is chosen based on the the lowest within-cluster sum of squares (WCSS).
- Elbow method works by plotting the within-cluster sum of squares (WCSS), representing the total distance of points within a cluster to their centroids, for different values of k.



# K-means Clustering

## Deciding the Number of Clusters k - Elbow Heuristic Method

- As you increase k, the WCSS decreases, but at some point, the rate of decrease slows down, forming an "elbow".



Elbow Method - Choose Number of Clusters : Elbow Point = 4



# K-Means Clustering

## Deciding the Number of Clusters k - Silhouette Method

- Silhouette Score  $s(x^{(i)})$ , computed for each sample  $x^{(i)}$ , is a metric for evaluating the quality of clustering.
- The average Silhouette Score  $S$  is obtained by computing the mean of Silhouette Scores of all the samples  $s(x^{(i)})$  or  $S = \frac{\sum_{i=1}^n s(x^{(i)})}{n}$
- Silhouette Score for a sample  $x^{(i)}$

$$s(x^{(i)}) = \frac{b(x^{(i)}) - a(x^{(i)})}{\max(a(x^{(i)}), b(x^{(i)}))}$$

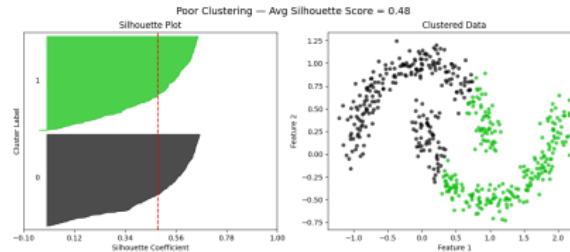
where  $a(x^{(i)})$  is the mean distance between  $x^{(i)}$  and all other points in the same cluster (intra-cluster distance) and  $b(x^{(i)})$  is the mean distance between  $x^{(i)}$  and all points in the nearest neighboring cluster (inter-cluster distance).

- Silhouette Score ranges from -1 to 1
  - Silhouette Score close to 1 => Data Points are Well Clustered
  - Silhouette Score close to 0 => Not Well-Clustered and Data Points are close to the boundary
  - Silhouette Score close to -1 => Data Points are Likely Misclassified.

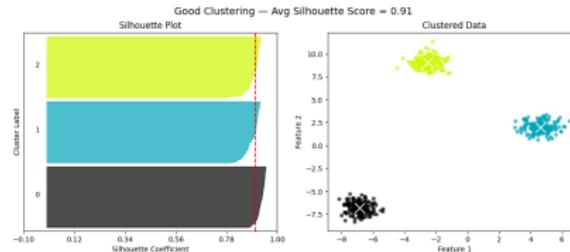


# K-Means Clustering

Visualizing Clustering Quality : Silhouette Score



Bad Cluster k=2



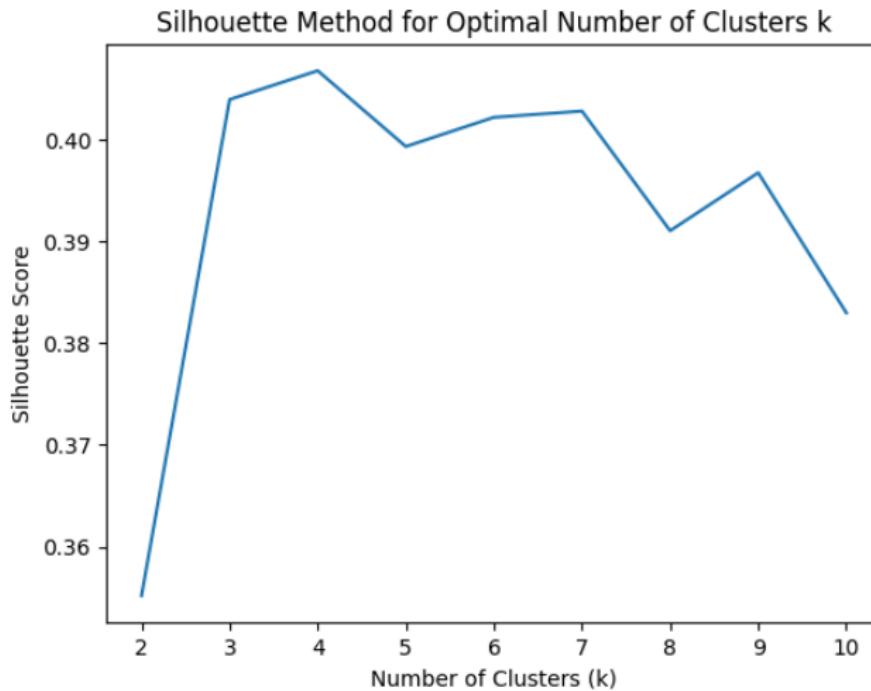
Good Cluster k=3

Bad Cluster & Good Cluster



# K-Means Clustering

Deciding the Number of Clusters k - Silhouette Method



Silhouette Method - Choose Number of Clusters : Silhouette Point = 4



# K-Means Clustering

Comparison : Elbow Method and Silhouette Method

Aspect	Elbow Method	Silhouette Method
Measure	Compactness	Compactness and Separation
Data	Spherical well-separated clusters	Arbitrary shaped clusters with any densities
Computations	Low	High
Interpretability	Subjective	Clear
Outliers	Sensitive to Outliers.	Robust to Outliers

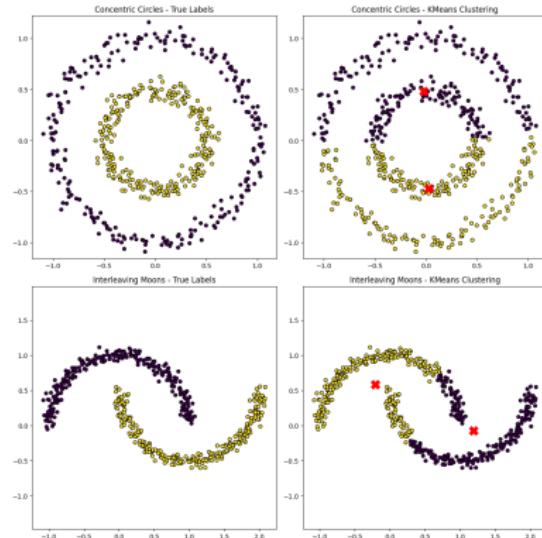
Comparison of Elbow Method and Silhouette Method



# K-means Clustering

## Why K-means fail

- K-Means works best for well-separated, spherical clusters or convex boundaries.



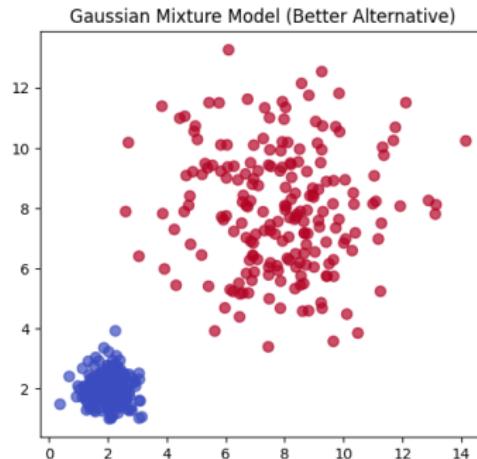
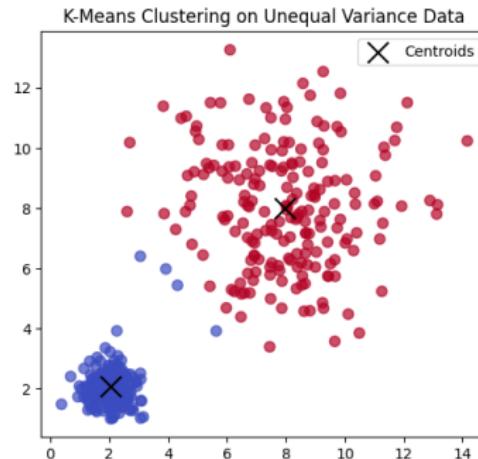
K-means fails for Non-Convex Data Patterns



# K-Means Clustering

## Why K-means fail

- K-Means assumes that the clusters have roughly the same variance and density.



K-means fails for Varying Density Clusters



# K-Means Clustering

## Distance Metrics

### Similarity & Distance

For any two samples  $x^{(i)}$  and  $x^{(j)}$ , the distance will be smaller if they are identical.

#### Continuous Data

Distance	Expression
Manhattan Distance $L_1$	$d(x^{(i)}, x^{(j)}) = \sum_{k=1}^m  x_{ik} - x_{jk} $
Euclidean Distance $L_2$	$d(x^{(i)}, x^{(j)}) = \sqrt{\sum_{k=1}^m (x_{ik} - x_{jk})^2}$
Cosine Distance $1 - \text{Cos}(\theta)$	$d(x^{(i)}, x^{(j)}) = 1 - \frac{x^{(i)} \cdot x^{(j)}}{\ x^{(i)}\  \ x^{(j)}\ }$

#### Binary Data

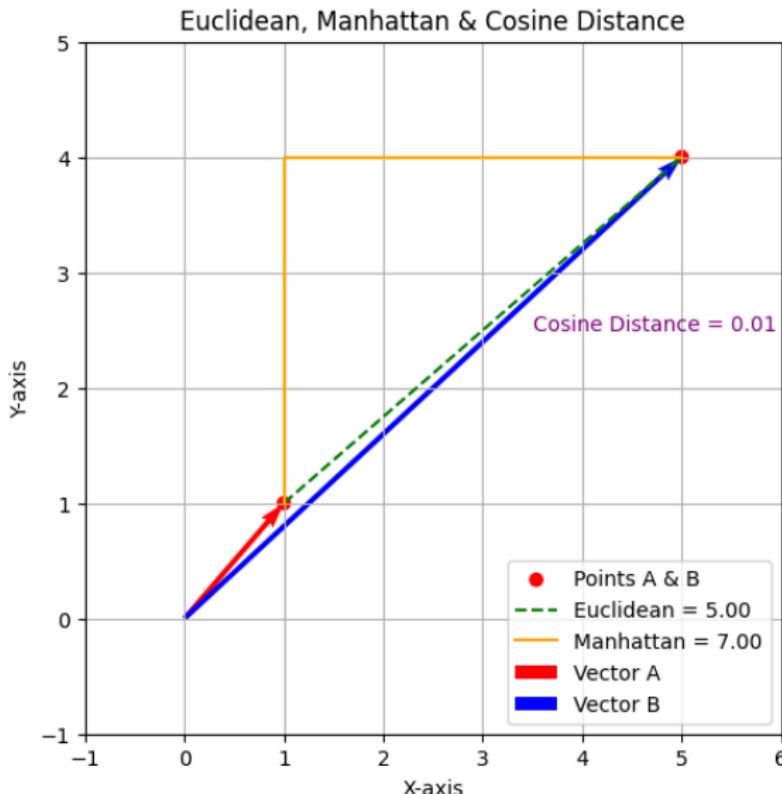
Hamming Distance	$d(x^{(i)}, x^{(j)}) = \sum_{i=1}^n \delta(x_i \neq y_i)$
------------------	---

### Distance Metrics



# K-Means Clustering

## Distance Measures - Visualization



# Hierarchial Clustering



## Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters either using a bottom-up approach (agglomerative clustering) or a top-down approach (divisive clustering).



# Hierarchical Clustering

## Agglomerative Hierarchical Clustering Algorithm

Agglomerative clustering is a bottom-up hierarchical clustering algorithm where

- Step 1 : At start, each data sample is considered as a cluster, resulting in  $n$  independent clusters  $C_1, C_2, C_3, \dots, C_n$
- Step 2 : Compute the distance between every pair of clusters  $C_i, C_j$ , where  $i \neq j, i, j \in \{1, 2, \dots, n\}$ .
- Clusters are iteratively merged based on the **linkage** criterion, until a single cluster containing all points is formed or a stopping criterion (e.g., a fixed number of clusters) is reached.
- Step 3 : Identify the pair of clusters  $(C_i, C_j)$ , with the smallest distance  $d(C_i, C_j)$  or smallest increase in within cluster variance  $\Delta E$  and merge  $C_i$  and  $C_j$  into a new cluster  $C_{ij}$ .
- Step 4 : Recompute distances between the new cluster  $C_{ij}$  and all other clusters.
- Repeat the step 2,3,4 until a predefined number of clusters  $k$  is reached, or the distance between clusters exceeds a threshold.



# Hierachial Clustering

## Linkage Criterion in Hierachial Agglomerative Clustering

### Linkage Criterion

The linkage criterion determines the distance between two clusters.

- **Single Linkage or Minimum pairwise distance:** The distance between two clusters is the minimum distance between any two points in the clusters:

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \quad (4)$$

- **Complete Linkage or Maximum pairwise distance:** The distance between two clusters is the maximum distance between any two points in the clusters:

$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \quad (5)$$

- **Average Linkage or Average pairwise distance:** The distance between two clusters is the average distance between all pairs of points in the clusters:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y) \quad (6)$$



# Hierachial Clustering

## Linkage Criterion in Hierachial Agglomerative Clustering

- **Centroid Linkage:** The distance between two clusters is the difference between the cluster centroids

$$d(C_i, C_j) = \|\mu_{C_i} - \mu_{C_j}\| \quad (7)$$

- **Wards Linkage or Increase in variance (SSE) :** Merge clusters that result in the smallest increase in within-cluster variance  $\Delta E$

$$\Delta E = \frac{|C_i||C_j|}{|C_i + C_j|} \|\mu_{C_i} - \mu_{C_j}\|^2 \quad (8)$$

where :

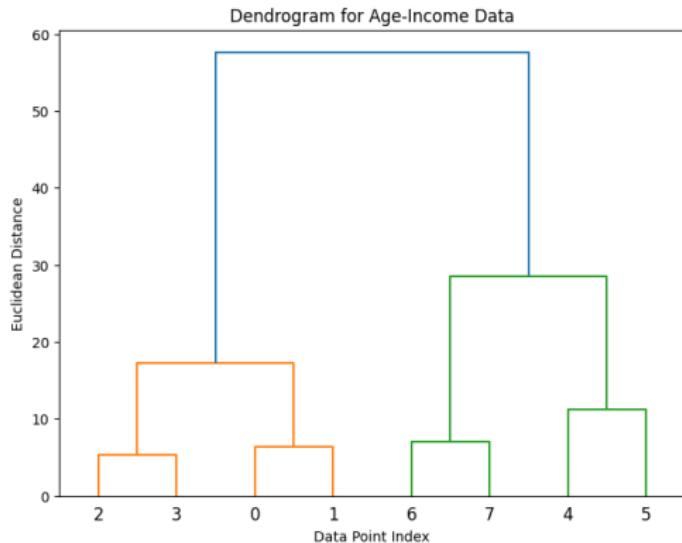
- $|C_i|$  represents the number of data points in Cluster i.
- $|C_j|$  represents the number of data points in Cluster j.
- $\mu_{C_i}$  represents the centroid of Cluster i.
- $\mu_{C_j}$  represents the centroid of Cluster j.



# Hierarchical Clustering

## Agglomerative Hierarchical Clustering

### 8 Sample Data with Age and Income - Agglomerative Hierarchical Clustering



Agglomerative Clustering

Sample	Age	Income
0	25	50
1	30	54
2	35	58
3	40	60
4	45	80
5	50	90
6	55	70
7	60	65

Age-Income Data



# Hierarchical Clustering

## Agglomerative Hierarchical Clustering - Illustration

### Linkage Matrix

$i$	$j$	$d$	$n$
2.	3.	5.39	2.
0.	1.	6.4	2.
6.	7.	7.07	2.
4.	5.	11.18	2.
8.	9.	17.26	4.
10.	11.	28.5	4.
12.	13.	57.64	8.

- The linkage matrix is an  $(n-1) \times 4$  matrix, where  $n$  is the number of data samples.
- Each row of the linkage matrix consists of  $[i,j,d,n]$ , where  $i$  and  $j$  are index of the clusters,  $d$  is the distance between the clusters, determined by linkage method and  $n$  is the number of points in the new cluster.
- If the distance between clusters is restricted within 11.19, then there are four clusters formed using Agglomerative Hierarchical Clustering.



# Hierarchical Clustering

## Divisive Hierarchical Clustering

- Divisive Hierarchical clustering is a top-down clustering approach where all data points start in a single cluster and iteratively split into smaller clusters.
- The goal of divisive clustering is to iteratively divide a cluster into two subclusters to optimize a criterion, such as minimizing the within-cluster sum of squares (WCSS).

$$\text{WCSS} = \sum_{i=1}^2 \sum_{x^{(j)} \in C_i} \|x^{(j)} - \mu_i\|^2$$

where:

- $C_i$  is the  $i$ -th cluster,
- $\mu_i$  is the centroid of cluster  $C_i$ ,
- $\|x^{(j)} - \mu_i\|^2$  is the squared Euclidean distance between a sample  $x^{(j)}$  and its cluster centroid.



# Hierarchical Clustering

## Divisive Hierarchical Clustering

- Step 1 : Start with all data points in a single cluster  $C = \{x_1, x_2, \dots, x_n\}$ .
- Step 2 : Partition  $C$  into two subclusters  $C_1$  and  $C_2$  such that the chosen criterion (e.g., WCSS) is optimized. Optimization requires finding a split that minimizes:

$$\text{WCSS}_{\text{split}} = \text{WCSS}_{C_1} + \text{WCSS}_{C_2}$$

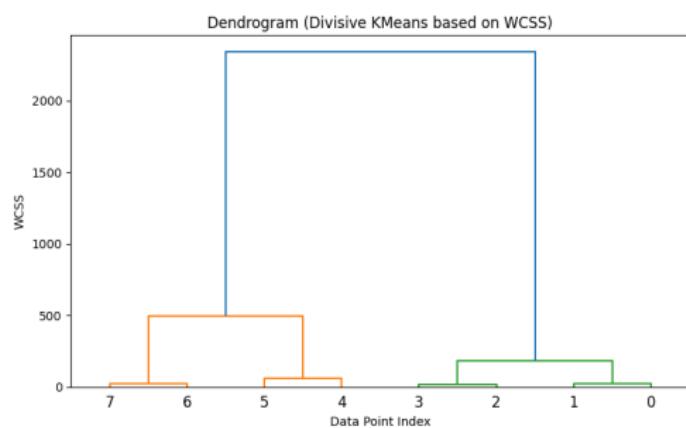
- The split could be achieved by K-Means Clustering with  $K=2$ .
- Select the next cluster to split (based on a heuristic, such as the largest WCSS) and repeat until a termination condition is met (e.g., a predefined number of clusters or threshold on WCSS).



# Hierarchical Clustering

## Divisive Hierarchical Clustering

### 8 Sample Data with Age and Income - Divisive Hierarchical Clustering



Divisive Clustering

Sample	Age	Income
0	25	50
1	30	54
2	35	58
3	40	60
4	45	80
5	50	90
6	55	70
7	60	65

Age-Income Data



# Hierarchical Clustering

## Divisive Hierarchical Clustering - Illustration

### Linkage Matrix

$i$	$j$	$d$	$n$
10	13	2338	8
8	9	493	4
11	12	184	4
5	4	62	2
7	6	25	2
1	0	20	2
3	2	14	2

- The linkage matrix is an  $(n-1) \times 4$  matrix, where  $n$  is the number of data samples.
- Each row of the linkage matrix consists of  $[i,j,d,n]$ , where  $i$  and  $j$  are index of the clusters after split,  $d$  is the WCSS and  $n$  is original number of data points in the cluster.



# **Association Rules**



## Association Rules

Association Rule Mining is a fundamental technique in data mining used to uncover interesting relationships (or associations) between items in large datasets.

- Association Rules was introduced in the landmark paper - "Mining Association Rules between Sets of Items in Large Databases" by Rakesh Agrawal, Tomasz Imieliński, and Arun Swami.
- The Apriori algorithm for mining frequent itemsets was later introduced in 1994 by Agrawal and Ramakrishnan Srikant.
- **Market Basket Analysis** examines customer purchase data to find associations between items.
- The goal of Market Basket Analysis is to discover which products are frequently bought together, allowing retailers to make more informed business decisions.



# Association Rules

## Market Basket Analysis

Transaction	Items
Transaction 1	
Transaction 2	
Transaction 3	
Transaction 4	
Transaction 5	
Transaction 6	
Transaction 7	
Transaction 8	

Transactions Data for Market Basket Analysis



# Association Rules

## Market Basket Analysis

- Itemset - A collection of one or more items.
- Support  $S(X)$  represents the proportion of transactions in the dataset that contain a particular itemset( $X$ ).

$$S(X) = p(X) = \frac{\text{Transactions containing } X}{\text{Total transactions } N} \quad (9)$$

For example : Support of beer, representing the fraction of transactions where atleast one or more beer is sold  $\rightarrow \frac{6}{8}$ .  $S(X \Rightarrow Y)$  represents the proportion of transactions containing both items  $X$  and  $Y$ .

- Confidence  $C(X \Rightarrow Y)$  represents how often an item  $Y$  appears in transactions containing an item  $X$ , represented by  $p(\frac{Y}{X})$ .

$$C(X \Rightarrow Y) = p(\frac{Y}{X}) = \frac{p(X \cap Y)}{p(X)} = \frac{S(X \Rightarrow Y)}{S(X)} \quad (10)$$

What is the probability that a transaction containing apple will also contain beer  $\rightarrow \frac{3}{4}$ .



# Association Rules

## Market Basket Analysis

- Lift measures how much more likely Y is to appear when X is present, compared to when X and Y are independent.
- Lift of association rule  $L(\{X \rightarrow Y\})$  is given by

$$L(X \Rightarrow Y) = \frac{P(X \cap Y)}{P(X).P(Y)} \quad (11)$$

- Lift  $> 1$  : Positive correlation (items occur together more often than expected).
- Lift  $= 1$  : No correlation
- Lift  $< 1$  : Negative Correlation (items occur together less often than expected).
- Lift can also be considered as a scaled version of confidence  $C(X \Rightarrow Y)$ .

$$L(X \Rightarrow Y) = \frac{C(X \Rightarrow Y)}{p(Y)} = \frac{p(Y|X)}{p(Y)} = \frac{p(X \cap Y)}{p(X).p(Y)} = \frac{C(X \Rightarrow Y)}{S(Y)} \quad (12)$$

How likely is chicken to appear with beer?  $\rightarrow \frac{4}{3}$ .



## Apriori Algorithm

The Apriori algorithm is one of the most popular ARM techniques, introduced by R. Agrawal and R. Srikant in 1994. It is based on the Apriori principle, which states:

- If an itemset is frequent, then all its subsets must also be frequent.
  - If an itemset is infrequent, then all its subsets must also be infrequent.
- 
- Find Frequent Itemsets
    - Identify itemsets that meet the minimum support threshold.
    - Use the Apriori property (if an itemset is frequent, its subsets are also frequent) to reduce the search space
  - Generate Association Rules:
    - For each frequent itemset L generate rules of the form  $X \Rightarrow Y$ , where  $X, Y \subseteq L$  and  $X \cap Y = \emptyset$
    - Calculate confidence and lift for each rule.
    - Retain rules that meet the minimum confidence threshold.
  - Prune and Optimize: Remove infrequent itemsets in each iteration to reduce computational complexity



# Apriori Algorithm : Example

Illustration of Apriori Algorithm

Transaction ID	Items Purchased
1	Milk, Bread, Butter
2	Bread, Butter, Tea
3	Milk, Bread
4	Milk, Butter, Eggs
5	Bread, Butter, Tea
6	Milk, Bread, Sugar
7	Bread, Butter
8	Tea, Sugar, Eggs
9	Milk, Bread, Butter, Eggs
10	Milk, Sugar

Transaction Dataset



# Apriori Algorithm

## Illustration of Apriori Algorithm

Step 1: Find the frequent items based on the Support Condition. Support Calculation for Single Items The frequency of individual items:

Item	Count	Support (%)
Milk	6	60%
Bread	7	70%
Butter	6	60%
Tea	3	30%
Eggs	3	30%
Sugar	3	30%

Support Calculation for Single Items

Assuming a minimum support threshold of 40%, we remove Tea, Eggs, and Sugar.



# Apriori Algorithm

## Illustration of Apriori Algorithm

### Step 2: Frequent Itemsets of Size 2

Itemset	Count	Support (%)
(Milk, Bread)	4	40%
(Milk, Butter)	3	30%
(Bread, Butter)	5	50%

Frequent Itemsets of Size 2

Since  $S(\text{Milk, Butter})$  has support  $< 40\%$ , we discard it.



# Apriori Algorithm

## Illustration of Apriori Algorithm

### Step 3: Frequent Itemsets of Size 3

Itemset	Count	Support (%)
(Milk, Bread, Butter)	2	20%

### Frequent Itemsets of Size 3

Since (Milk, Bread, Butter) has support < 40%, we discard it.

### Step 4: Association Rule Generation

Rules from (Milk, Bread):

- Milk  $\Rightarrow$  Bread (Confidence =  $\frac{4}{6} = 66.7\%$ )
- Bread  $\Rightarrow$  Milk (Confidence =  $\frac{4}{6} = 66.7\%$ )

Rules from (Bread, Butter):

- Bread  $\Rightarrow$  Butter (Confidence =  $\frac{5}{7} = 71.4\%$ )
- Butter  $\Rightarrow$  Bread (Confidence =  $\frac{5}{6} = 83.3\%$ ) (**Strongest Rule**)



# Apriori Algorithm

## Illustration of Apriori Algorithm

### Final Frequent Itemsets and Strongest Rule

- **Frequent Itemsets:**

- (Milk, Bread) (40%)
- (Bread, Butter) (40%)

- **Strongest Rule:** Butter  $\Rightarrow$  Bread (83.3% confidence)



# **Recommendation System**



## Recommendation System

A system that suggests relevant items (products, movies, articles) to users based on their preferences, behavior, or other contextual information. These systems are widely used in e-commerce, streaming platforms, and other domains to enhance user experience and engagement.

Recommendation systems are primarily based on three approaches

- **Content-based Filtering**
- **Collaborative Filtering**
  - User-based
  - Item-based
  - Matrix Factorization
- **Hybrid Filtering**

Filtering means selecting or recommending only those items that match a user's revealed interests.



# Recommendation System

## Key Idea & Applications

- A user  $U$  whose movie preferences are known will be recommended movies based on his preferences - Content Based Filtering based on user preferences.
- If User  $U_1$  and User  $U_2$  have rated  $N-1$  movies similarly, recommend a movie  $M_N$  highly rated by User  $U_1$  to User  $U_2$ , when User  $U_2$  has not watched the movie  $M_N$  - Collaborative Filtering based on Similar Users.
- If Movie  $M_1$  and Movie  $M_2$  have been rated similarly by  $N-1$  users, recommend movie  $M_1$  to user  $U_N$  when the user  $U_N$  has rated the movie  $M_2$  with a high rating - Collaborative Filtering based on Similar Items.

## Applications

- E-commerce (Amazon, Flipkart)
- Streaming services (Netflix, Spotify)
- Online learning (Coursera, Udemy)
- Healthcare (Personalized Treatment Recommendations)
- Newspaper (Offering articles based on interests)



# **Content Based Filtering**



# Content Based Filtering

## Content Based Filtering

Content-based filtering is a recommendation system technique that suggests an item (movie) to a user (reviewer) after learning the preferences of the user.

Customer	Action	Comedy
C1	5	2
C2	2	5

User Profile

Movie	Action	Comedy
Movie 1	5	1
Movie 2	1	5
Movie 3	4	1
Movie 4	2	4

Item Profile

Customer C1 would be recommended Movie 1 and Movie 3.

Customer C2 would be recommended Movie 2 and Movie 4.



# Content Based Filtering

- A **user profile** represented as a vector  $U \in \mathbb{R}^{m \times p}$ , where m refers to the number of users and p refers to the number of user features.
- A set of **item profiles** represented as a feature matrix  $X \in \mathbb{R}^{n \times p}$ , where n is the number of items and p refers to the number of features representing the items.
- The core idea in Content-based filtering is to compute the similarity between the user profile and item profiles. Commonly used similarity measures are
  - Cosine Similarity  $\text{Cos}(\theta)$ <sup>3</sup>
  - Euclidean Distance<sup>4</sup>
  - Pearson Correlation Coefficient  $\rho$ <sup>5</sup>
- A item  $X_k \in \mathbb{R}^p$  would be recommended to user with profile U, if the  $S(U, X_k) > S(U, X_i)$ , for  $k \neq i$ .
- For a given user profile  $U$ , recommend the top  $q$  most similar items based on the similarity scores, where  $q \ll n$ .

---

<sup>3</sup>Distance  $\downarrow \rightarrow \text{Cos}(\theta) \uparrow \rightarrow \text{Similarity} \uparrow$

<sup>4</sup>Distance  $\downarrow \rightarrow \text{Similarity} \uparrow$

<sup>5</sup>Distance  $\downarrow \rightarrow \rho \uparrow \rightarrow \text{Similarity} \uparrow$



# Content Based Filtering

## Similarity Measures - Cosine Similarity

- **Cosine similarity** -  $S(U, X_i)$  between an user U and an item  $X_i$  is given by

$$S(U, X_i) = \frac{U \cdot X_i}{\|U\| \|X_i\|} \quad (13)$$

where  $U \cdot X_i$  is the dot product of vectors  $U$  and  $X_i$ , and  $\|U\|$  and  $\|X_i\|$  is the Euclidean norm of user U and item  $X_i$ .

- A item  $X_k \in \mathbb{R}^P$  would be recommended to user with profile U, if the  $S(U, X_k) > S(U, X_i)$ , for  $k \neq i$ .
  - If  $S(U, X_i)$  close to 1, U and  $X_i$  are similar.
  - If  $S(U, X_i)$  close to -1, U and  $X_i$  are opposite.
  - If  $S(U, X_i) = 0$ , U and  $X_i$  are unrelated.
- $-1 \leq S(U, X_i) \leq 1$



# Content Based Filtering

Similarity Measures - Cosine Similarity

User	Movie A	Movie B	Movie C	Movie D
User1	5	3	0	1
User2	4	0	0	1
User3	1	1	0	5
User4	1	0	0	4
User5	0	1	5	4

User-Item Rating Matrix for 5 Users and 4 Movies

- Movies - 4, Users - 5
- User Cosine Similarity Matrix - Order (5\*5)
- Movie Cosine Similarity Matrix - Order (4\*4)
- User 1 and User 2 - Similar
- Movie A and Movie B - Similar



# Content Based Filtering

Similarity Measures - Cosine Similarity

	User1	User2	User3	User4	User5
User1	1.000000	0.860916	0.422890	0.368964	0.182574
User2	0.860916	1.000000	0.420084	0.470588	0.149696
User3	0.422890	0.420084	1.000000	0.980196	0.623610
User4	0.368964	0.470588	0.980196	1.000000	0.598785
User5	0.182574	0.149696	0.623610	0.598785	1.000000

Cosine Similarity between Users based on Ratings

	Movie A	Movie B	Movie C	Movie D
Movie A	1.000000	0.735681	0.000000	0.357365
Movie B	0.735681	1.000000	0.301511	0.471041
Movie C	0.000000	0.301511	1.000000	0.520756
Movie D	0.357365	0.471041	0.520756	1.000000

Cosine Similarity between Movies based on User Ratings



# Content Based Filtering

## Similarity Measures - Euclidean Distance

- Similarity between an user  $U$  and an item  $X_i$  can be computed using **Euclidean distance**  $S(U, X_i)$

$$S(U, X_i) = \|U - X_i\|_2 = \sqrt{\sum_{k=1}^p (U_k - X_{ik})^2} \quad (14)$$

where a lower Euclidean distance indicates higher similarity.

- A item  $X_k \in \mathbb{R}^p$  would be recommended to user with profile  $U$ , if the  $S(U, X_k) < S(U, X_i)$ , for  $k \neq i$ .
- $S(U, X_i) \geq 0$



# Content Based Filtering

Similarity Measures - Euclidean Distance

	User1	User2	User3	User4	User5
User1	0.000000	<b>3.162278</b>	6.000000	5.830952	7.937254
User2	<b>3.162278</b>	0.000000	5.099020	4.242641	7.141428
User3	6.000000	5.099020	0.000000	1.414214	5.196152
User4	5.830952	4.242641	1.414214	0.000000	5.196152
User5	7.937254	7.141428	5.196152	5.196152	0.000000

Euclidean Distance between Users based on Ratings

	Movie A	Movie B	Movie C	Movie D
Movie A	0.000000	<b>4.690416</b>	8.246211	8.124038
Movie B	<b>4.690416</b>	0.000000	5.099020	6.782330
Movie C	8.246211	5.099020	0.000000	6.633250
Movie D	8.124038	6.782330	6.633250	0.000000

Euclidean Distance between Movies based on User Ratings



# Content Based Filtering

## Similarity Measures - Manhattan Distance

- Similarity between an user  $U$  and an item  $X_i$  can be computed using **Manhattan distance**  $S(U, X_i)$ ,

$$S(U, X_i) = \|U - X_i\|_1 = \sum_{k=1}^p |U_k - X_{ik}| \quad (15)$$

where a lower Manhattan distance indicates higher similarity.

- A item  $X_k \in \mathbb{R}^p$  would be recommended to user with profile  $U$ , if the  $S(U, X_k) < S(U, X_i)$ , for  $k \neq i$ .
- $S(U, X_i) \geq 0$



# Content Based Filtering

## Similarity Measures - Manhattan Distance

User-User Manhattan Distance

	User1	User2	User3	User4	User5
User1	0.0	4.0	10.0	10.0	15.0
User2	4.0	0.0	8.0	6.0	13.0
User3	10.0	8.0	0.0	2.0	7.0
User4	10.0	6.0	2.0	0.0	7.0
User5	15.0	13.0	7.0	7.0	0.0

Movie-Movie Manhattan Distance

	Movie A	Movie B	Movie C	Movie D
Movie A	0.0	8.0	16.0	18.0
Movie B	8.0	0.0	8.0	14.0
Movie C	16.0	8.0	0.0	12.0
Movie D	18.0	14.0	12.0	0.0



# Content Based Filtering

## Similarity Measures - Pearson Correlation Coefficient

- For normalized data, the similarity between an user  $U$  and an item  $X_i$  could be represented using **Pearson's Correlation Coefficient**  $\rho(U, X_i)$ ,

$$\rho(U, X_i) = \frac{E[(U - \bar{U})(X_i - \bar{X}_i)]}{\sigma_U \cdot \sigma_{X_i}} \quad (16)$$

where  $\bar{U}, \bar{X}_i$  and  $\sigma_U, \sigma_{X_i}$  are the means and standard deviations of  $U, X_i$ , respectively.

- If  $\rho(U, X_i)$  is close to 1,  $U$  and  $X_i$  are similar.
- If  $\rho(U, X_i)$  is close to -1,  $U$  and  $X_i$  are opposite.
- If  $\rho(U, X_i)$  is close to 0,  $U$  and  $X_i$  are unrelated.



# Content Based Filtering

Similarity Measures - Pearson Correlation Coefficient

	User1	User2	User3	User4	User5
User1	1.000000	0.774291	-0.186441	-0.178683	-0.978839
User2	0.774291	1.000000	0.019854	0.162791	-0.628768
User3	-0.186441	0.019854	1.000000	0.972828	0.221028
User4	-0.178683	0.162791	0.972828	1.000000	0.258904
User5	-0.978839	-0.628768	0.221028	0.258904	1.000000

User-User Pearson Correlation

	Movie A	Movie B	Movie C	Movie D
Movie A	1.000000	0.470777	-0.567282	-0.924588
Movie B	0.470777	1.000000	0.000000	-0.327327
Movie C	-0.567282	0.000000	1.000000	0.298807
Movie D	-0.924588	-0.327327	0.298807	1.000000

Movie-Movie Pearson Correlation



# Content Based Filtering

User Preference Matrix + Item Feature Matrix → Recommendation Scores

Item	Action	Comedy	Drama
Movie A	1	0	1
Movie B	1	1	0
Movie C	0	1	1
Movie D	0	0	1

Item Feature Matrix (Movie)

User	Action	Comedy	Drama
User1	0.9	0.1	0.7
User2	0.2	0.8	0.1

User Preference Matrix

User	Movie A	Movie B	Movie C	Movie D
User1	1.6	1.0	0.8	0.7
User2	0.3	1.0	0.9	0.1



## Content-based Filtering -Pros and Cons <sup>6</sup>

- Pros 1 - The content-based filtering model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users, provided it is easy to obtain all the preferences of the users towards all items.
- Pros 2 - The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in. Eg: An award film, made in Vietnam might be suggested to a user based on his interest.
- Cons 1 - Since the feature representation of the items are hand-engineered to some extent, this technique requires a lot of domain knowledge. Therefore, the model can only be as good as the hand-engineered features.
- Cons 2 - The model can only make recommendations based on existing interests of the user or Overspecialization. In case, the interests changes, the recommendations would not be handy.



---

<sup>6</sup><https://developers.google.com/machine-learning/recommendation/content-based/summary>

# **Collaborative Filtering**



## Collaborative Filtering

Collaborative filtering (CF) is a recommendation technique that predicts user preferences based on similarities between users or items.

There are three approaches to Collaborative Filtering.

- User-Based Collaborative Filtering - Based on similar users.
- Item-Based Collaborative Filtering - Based on similar items.
- Matrix Factorization - Discover hidden patterns based on latent features.



# Collaborative Filtering

## User-Based Collaborative Filtering

User/Item	Movie A	Movie B	Movie C	Movie D	Movie E
Alice	5	4	?	2	?
Bob	5	4	4	3	2
Charlie	2	1	3	5	4
David	3	2	4	5	4

User-Item Rating Matrix for User-Based Collaborative Filtering

### User-Based Collaborative Filtering

Do you recommend movie C to Alice?

Do you recommend movie E to Alice?

In User-based Collaborative Filtering, the recommendations are made based on the similarity of users.



# Collaborative Filtering

User-Based Collaborative Filtering - Weighted User Similarity Approach with Bias Adjust

User-based collaborative filtering predicts a missing rating  $\hat{R}_{u,i}$  for user  $u$  and item  $i$  using similar users' ratings:

$$\hat{R}_{u,i} = \bar{R}_u + \frac{\sum_{v \in N(u)} S(u, v)(R_{v,i} - \bar{R}_v)}{\sum_{v \in N(u)} |S(u, v)|} \quad (17)$$

where:

- $\hat{R}_{u,i}$  Predicted rating of user  $u$  for item  $i$ .
- $N(u)$  is the set of users similar to  $u$  who rated item  $i$ .
- $S(u, v)$  is the similarity measure between users  $u$  and  $v$  (e.g., Pearson correlation, Cosine Similarity).
- $\bar{R}_u$  is the average rating of user  $u$ .
- $R_{v,i}$  is the rating given by user  $v$  to item  $i$ .
- $\bar{R}_v$  is the average rating of user  $v$ .

Bias adjustment in collaborative filtering refers to correcting for users' and items' individual rating tendencies before making predictions.



# Collaborative Filtering

User-Based Collaborative Filtering - Weighted User Similarity Approach with no Bias Adjust

User-based collaborative filtering predicts a missing rating  $\hat{R}_{u,i}$  for user  $u$  and item  $i$  using similar users' ratings:

$$\hat{R}_{u,i} = \frac{\sum_{v \in N(u)} S(u, v)(R_{v,i})}{\sum_{v \in N(u)} |S(u, v)|} \quad (18)$$

where:

- $\hat{R}_{u,i}$  Predicted rating of user  $u$  for item  $i$ .
- $N(u)$  is the set of users similar to  $u$  who rated item  $i$ .
- $S(u, v)$  is the similarity measure between users  $u$  and  $v$  (e.g., Pearson correlation, Cosine Similarity).
- $R_{v,i}$  is the rating given by user  $v$  to item  $i$ .



# Collaborative Filtering

## Item-Based Collaborative Filtering

Item/User	Alice	Bob	Charlie	David
Movie A	5	5	2	3
Movie B	4	4	1	2
Movie C	?	4	3	4
Movie D	2	3	5	5

Item-User Rating Matrix for Item-Based Collaborative Filtering

### Item-Based Collaborative Filtering

Do you recommend movie C to Alice?

In Item-based Collaborative Filtering, the recommendations are made based on the similarity of items.



# Collaborative Filtering

## Item-Based Collaborative Filtering - Weighted Item Similarity Approach

Item-based collaborative filtering predicts a missing rating  $R_{u,i}$  for user  $u$  and item  $i$  using similar items ratings:

$$\hat{R}_{u,i} = \frac{\sum_{j \in N(i)} S(i,j) R_{u,j}}{\sum_{j \in N(i)} |S(i,j)|} \quad (19)$$

where:

- $\hat{R}_{u,i}$  is the predicted rating of user  $u$  for item  $i$ .
- $j \in N(i)$  is the set of items similar to  $i$ .
- $S(i,j)$  is the similarity between items  $i$  and  $j$  (e.g., Pearson correlation, Cosine Similarity).
- $R_{u,j}$  is the rating of user  $u$  for item  $j$ .

Item-based Collaborative Filtering is more stable than User-based Collaborative Filtering as items change less frequently than users.



# Collaborative Filtering

Training Data : Implicit and Explicit

Item/User	Alice	Bob	Charlie	David
Movie A	5	5	2	3
Movie B	4	4	1	2
Movie C	?	4	3	4
Movie D	2	3	5	5

Explicit : User-Item Rating Matrix for Item-Based CF

Item/User	Alice	Bob	Charlie	David
Movie A	1	1	0	1
Movie B	1	1	0	0
Movie C	?	1	1	1
Movie D	0	1	1	1

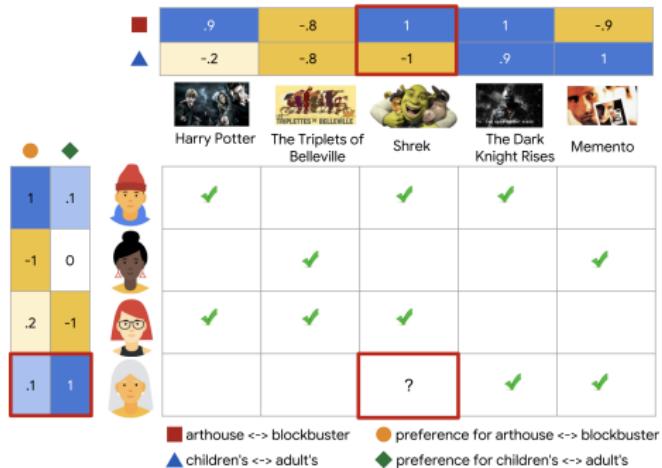
Implicit : User-Item Rating Matrix for Item-Based CF



# Collaborative Filtering with Matrix Decomposition

## Collaborative Filtering with Matrix Decomposition

Given the implicit feedback data matrix, can we predict the rating that would be given by the fourth customer for the movie "Shrek" - Yes!



Collaborative Filtering Training Feedback Data Matrix<sup>7</sup>

<sup>7</sup><https://developers.google.com/machine-learning/recommendation/collaborative/basics>



## Collaborative Filtering - Matrix Decomposition

- Implicit Feedback Matrix  $X_{(m*n)} \approx U_{(m*p)} \cdot V_{(p*n)}^T$  can be decomposed into User Embedding Matrix  $U_{(m*p)}$  and Item Embedding Matrix  $V_{(n*p)}$ , where m refers to number of users, n refers to number of items and p refers to the latent dimensions.
- Given the feedback matrix  $X_{(m*n)}$ , it learns the User Embedding Matrix  $U_{(m*p)}$  and Item Embedding Matrix  $V_{(n*p)}$ .
- Collaborative Filtering yields us the embedding representations of users and items, where the length of the embedding is p.
- The unknown cells in the sparse feedback matrix  $X_{m*n}$  can be filled with the approximation matrix  $X_{approx(m*n)} = U_{(m*p)} \cdot V_{(p*n)}^T$ .

$$X_{m*n} \approx X_{approx(m*n)} = U_{(m*p)} \cdot V_{(p*n)}^T$$



# Collaborative Filtering

## Matrix Decomposition and Matrix Reconstruction



## Matrix Decomposition - Collaborative Filtering<sup>8</sup>

Embedding representations of users and items are also shown

- Given the sparse matrix  $X_{m*n}$  with Users - 4, Movies - 5 ( $U_{4*2}, V_{5*2}$ )
- Latent Dimensions - 2
- Reconstructed Complete Matrix  $X_{approx(m*n)} = U_{(m*p)} \cdot V_{(p*n)}^T$

<sup>8</sup><https://developers.google.com/machine-learning/recommendation/collaborative/basics>



# Collaborative Filtering with SVD

## Singular Value Decomposition

### Singular Value Decomposition

SVD decomposes the rectangular user-item matrix  $A$  into three matrices  $U, \Sigma$  and  $V$ .

$$A_{(m*n)} = U_{(m*m)} \Sigma_{(m*n)} V^T_{(n*n)}$$

where:

- $U$  (User Profile Matrix): Left-Singular Orthogonal Matrix representing user embeddings or latent features.
- $\Sigma$  (Singular Values): Diagonal matrix  $\Sigma$  consists of singular values representing the most important features arranged in the descending order,  $\sigma_1 \geq \sigma_2 \geq \dots \sigma_r \geq 0$ , where  $r$  is the rank of the matrix  $A$ .
- $V^T$  (Item Profile Matrix): Right-Singular Orthogonal Matrix representing item embeddings or latent features.
- $m$  is the number of users.
- $n$  is the number of items.



# Collaborative Filtering with SVD

## Singular Value Decomposition

### Singular Value Decomposition (SVD)

$$A = U \Sigma V^T$$

The diagram illustrates the Singular Value Decomposition (SVD) of a matrix A. Matrix A is shown as a blue rectangle labeled 'A' with dimensions '(m x n)'. To its right is an equals sign. Following the equals sign are three components: matrix U (red rectangle), matrix Sigma (green rectangle), and matrix V^T (purple rectangle). Matrix U is labeled '(m x m) Orthogonal'. Matrix Sigma is labeled '(m x n) Diagonal'. Matrix V^T is labeled '(n x n) Orthogonal'. The components are arranged horizontally, separated by small gaps.

A: Original data matrix

U: Left singular vectors (output space)

$\Sigma$ : Singular values (strengths)

$V^T$ : Right singular vectors (input space)

*Singular values  $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$  represent importance of each dimension*

### Singular Value Decomposition

By choosing the top  $p$  singular values i.e Retain  $p$  columns of matrix  $U$  and  $p$  rows of matrix  $V^T$ , we can reconstruct the matrix  $A$

$$\hat{A} = U_{(m*p)} \Sigma_{(p*p)} V^T_{(p*n)}$$

(20)



# Collaborative Filtering with SVD

## Singular Value Decomposition

### Personalized recommendations using SVD

To predict missing values for user  $i$  for movie  $k$ , we approximate unknown cells  $\hat{R}$ :

$$\hat{R}_{1*1} = U_i \Sigma_{p*p} V_k^T$$

- **User profile**  $U_i$  ( $1*p$ ) is a row vector with length  $p$  representing the latent user preferences of user  $i$ .
- **Singluar value matrix**  $\Sigma_p$  ( $p*p$ ) with a diagonal square matrix of order  $p*p$ .
- **Item profile vector**  $V_k^T$  ( $p*1$ ) is a column vector with length  $p$  representing latent item features of item  $k$ .
- $p$  is the reduced number of dimensions and  $p \leq r$ ,  $r$  is the rank of the matrix  $A$ .



# Singular Value Decomposition

## Structure of User-Item Rating Matrix A

The sparse user-item matrix  $A$  is given by:

$$A_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & ? & \cdots & a_{2n} \\ a_{31} & ? & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

where,

- Each row represents a user.
- Each column represents an item.
- A cell represents a rating given by a user  $i$  for a movie  $j$ , where  $i \in [1, m]$  and  $j \in [1, n]$ .
- $A$  is a sparse user-item matrix as most of the cell values are unknown or  $a_{ij} = ?$ .
- SVD needs a completely filled matrix for decomposition.



# Singular Value Decomposition

## Singular Value, Eigenvalue, Eigenvector

- The singular values of a rectangular matrix A are the square roots of the n eigenvalues of the symmetric matrix  $A^T \cdot A$  or

$$\boxed{\text{Singular}(A) = \sqrt{\text{EigenValues}(A^T \cdot A)}}$$

- The singular value  $\sigma_i$  satisfy the condition  $\sigma_i = \sqrt{\lambda_i}$ , where  $\lambda_i$  refers to the eigen values of symmetric matrix  $A^T \cdot A$ ,  $i \in [1, n]$ .
- The eigen values of symmetric matrix  $S_1 = A \cdot A^T$  and  $S_2 = A^T \cdot A$  are the same, except for the possibly differing number of zero eigen values.
- The total number of eigen values of symmetric matrices  $S_1 = A \cdot A^T$  and  $S_2 = A^T \cdot A$  are m and n respectively.

## Left Singular Matrix U and Right Singular Matrix V

- The columns of the left singular<sup>9</sup> orthogonal matrix  $U_{(m*m)}$  represent the m eigen vectors of the square symmetric matrix  $S_1 = A \cdot A^T$ .
- The rows of the right singular orthogonal matrix  $V_{(n*n)}^T$  represent the n eigen vectors of the square symmetric matrix  $S_2 = A^T \cdot A$ .

<sup>9</sup>A singular matrix is a  $p*p$  square matrix that does not have an inverse or  $\det(A) = 0$  or  $\text{rank}(A) < p$ .



# Singular Value Decomposition

User Profile Matrix U

$$U_{(m*m)} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \\ \uparrow & \uparrow & & \uparrow \end{bmatrix}$$

$$U_{(m*m)} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1m} \\ u_{21} & u_{22} & u_{23} & \cdots & u_{2m} \\ u_{31} & u_{32} & u_{33} & \cdots & u_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{m1} & u_{m2} & u_{m3} & \cdots & u_{mm} \end{bmatrix}$$

$$\mathbf{u}_i = \begin{bmatrix} u_{1i} \\ u_{2i} \\ \vdots \\ u_{mi} \end{bmatrix}$$

$\mathbf{u}_i$  is one of the m Eigenvectors of symmetric matrix  $S_1 = A \cdot A^T$



# Singular Value Decomposition

Item Profile Matrix V

$$V_{(n \times n)} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ \uparrow & \uparrow & & \uparrow \end{bmatrix}$$

$$V_{(n \times n)} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & \cdots & v_{1n} \\ v_{21} & v_{22} & v_{23} & \cdots & v_{2n} \\ v_{31} & v_{32} & v_{33} & \cdots & v_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & v_{n3} & \cdots & v_{nn} \end{bmatrix}$$

$$v_i = \begin{bmatrix} v_{1i} \\ v_{2i} \\ \vdots \\ v_{ni} \end{bmatrix}$$

$v_i$  is one of the n Eigenvectors of symmetric matrix  $S_2 = A^T \cdot A$



# Singular Value Decomposition

## Diagonal Matrix and Rank

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix}$$

$\sigma_i$  are the singular values of the rectangular matrix A, where  $A = U.\Sigma.V^T$ , where  $i \in [1, r]$ , r is the rank of the matrix A.

- The rank of the matrix A, defined as the number of independent columns of A or the number of independent rows of A, is the total number of non-zero singular values in matrix  $\Sigma$ .

- **Rank(A)  $\leq \min(m,n)$**

- If A is full rank<sup>10</sup> and  $m > n$  or A is tall, then maximum value of  $\text{rank}(A)$  is n.
- If A is full rank and  $m < n$  or A is wide, then maximum value of  $\text{rank}(A)$  is m.
- If  $\text{Rank}(A) = n$ , the matrix A has n independent columns and if  $\text{Rank}(A) = m$ , the matrix A has n independent rows.

---

<sup>10</sup>A matrix is full rank if it has no redundant (linearly dependent) rows or columns.



# Singular Value Decomposition

## Eigenvalue

### Eigenvalue

For a square matrix  $S$  of size  $n \times n$ , an eigenvalue  $\lambda$  satisfies:

$$S\mathbf{x} = \lambda\mathbf{x}, \quad \text{where } \mathbf{x} \neq 0. \quad (21)$$

For Finding Eigenvalues, solve the characteristic equation:

$$\det(S - \lambda I) = 0. \quad (22)$$

- In Collaborative Filtering using SVD, there are two possible square symmetric matrices  $S_1 = A \cdot A^T$  and  $S_2 = A^T \cdot A$ .
- The eigen values of the symmetric matrices  $S_1 = A \cdot A^T$  and  $S_2 = A^T \cdot A$  are real due to symmetric nature of matrices.
- The non-zero eigenvalues of  $S_1 = A \cdot A^T$  and  $S_2 = A^T \cdot A$  are the same.



# Singular Value Decomposition

## Properties of Eigenvalues

- **Sum of Eigenvalues:**  $\sum \lambda_i = \text{trace}(S) = \sum S_{ii}$ .
- **Product of Eigenvalues:**  $\prod \lambda_i = \det(S)$ .
- **Eigenvalues of  $S^T$ :** Same as eigenvalues of  $S$ .
- **Eigenvalues of  $S^{-1}$  (if  $S$  is Singular):**  $\lambda_i(S^{-1}) = \frac{1}{\lambda_i(S)}$ .
- **Eigenvalues of a Diagonal Matrix:** The diagonal entries are the eigenvalues.
- **Eigenvalues of a Symmetric Matrix:** Always real.
- **Eigenvalues of  $S_1 = A.A^T$  and  $S_2 = A^T.A$ :** The non-zero eigen values are the same.
- For a symmetric matrix  $S_1 = A.A^T$ , the number of non-zero eigenvalues among its  $m$  eigenvalues, determine the rank of the matrix  $S_1$ .
- For a symmetric matrix  $S_2 = A^T.A$ , the number of non-zero eigenvalues among its  $n$  eigenvalues, determine the rank of the matrix  $S_2$ .



# Singular Value Decomposition

Reconstruction : SVD

## Singular Value Decomposition

Singular Value Decomposition (SVD) is a fundamental matrix factorization technique in linear algebra. Given an  $m \times n$  matrix  $A$ , SVD decomposes it into three matrices:

$$\tilde{A}_{(m*n)} = U_{(m*p)} \Sigma_{(p*p)} V^T_{(r*n)} \quad (23)$$

where:

- $\tilde{A}_{(m*n)}$  is the reconstructed user-item matrix.
- $U$  is an  $m \times p$  **orthogonal** matrix ('p' left singular vectors).
- $\Sigma$  is an  $p \times p$  **diagonal** matrix with top 'p' singular values, where  $p \leq \text{rank } A$  and  $r \leq \min(m,n)$ .
- $V^T$  is an  $r \times n$  **orthogonal** matrix ('r' right singular vectors).



## Collaborative Filtering - Pros and Cons

- Pros 1 : No Need for Domain Knowledge - Collaborative filtering relies on user behavior and preferences, not the actual content of the items.
- Pros 2 : Can Capture Complex Patterns - Identifies latent preferences and unexpected relationships among users or items.
- Pros 3 : Personalized Recommendation - Provides personalized recommendations by leveraging similar users' choices or similar items.
- Cons 1 : Cold Start Problem - Struggles with new users or new items due to lack of data.
- Cons 2 : Data Sparsity - User-item matrices are often sparse (most users rate few items), leading to poor similarity estimation.
- Cons 3 : Scalability - User-user and item-item similarity calculations can become expensive in large systems.
- Cons 4 : Popularity Bias - Tends to recommend already popular items more frequently, ignoring niche interests.
- Cons 5 : Gray Sheep Problem - Difficult to recommend for users whose tastes don't consistently align with any group.



# Word2Vec



## Word2Vec - Introduction

- Word2Vec is a shallow, two-layer neural network developed by Tomas Mikolov (Google) that learns vector representations of words (word embeddings) from a large corpus of text.
- Traditional representations like one-hot encoding or bag-of-words have limitations: They create high-dimensional, sparse vectors.
  - They create high-dimensional, sparse vectors.
  - They do not capture relationships between words (e.g., "king" and "queen" are unrelated in one-hot encoding).
- Word2Vec solves this by learning dense, low-dimensional (e.g., 100–300D) vector representations that capture semantic relationships.
- Word2Vec uses a shallow neural network to learn word embeddings. There are two main approaches:
  - Continuous Bag of Words (CBOW)
  - Skip-Gram Model
- Word2Vec is a self-supervised learning model because it learns meaningful word representations without requiring labeled data. Instead, it generates its own "labels" from the raw text using a predictive task.



# Word2Vec

## Continuous Bag of Words (CBOW)

- Predicts a target word based on surrounding context words.
- Works well for smaller datasets and is faster.
- In the CBOW (Continuous Bag of Words) model, the window size determines how many context words are used to predict a target word.
- Window size is a critical factor that contributes to the performance of CBOW model.
  - Smaller window size → more local context (useful for syntactic similarity).
  - Larger window size → captures broader meaning (useful for semantic relationships).



- For a window size of 1, each target word is predicted using one word before and one word after.
- For the sentence - "The cat sits on the mat" with a window size 1.

Context words	Target word
(The, sits)	cat
(cat, on)	sits
(sits, the)	on
(on, mat)	the
(the, )	mat
(,cat)	The

Training Data : Window size = 1



# Word2Vec

## Continuous Bag of Words (CBOW)

- For a window size of 2, each target word is predicted using two words before and two words after.
- For the sentence - "The cat sits on the mat" with a window size 2.

Context words	Target word
(The, cat, on, the)	sits
(cat, sits, the, mat)	on
(,The,sits, on)	cat
(sits,on,mat,)	the
(on, the, , ,)	mat

Training Data : Window size = 2



- The Continuous Bag of Words (CBOW) model in Word2Vec involves a neural network with one hidden layer. The CBOW model consists of:

- Input Layer: Takes the one-hot encoded context words.
- Hidden Layer: A weight matrix (embedding matrix) converts one-hot vectors into dense embeddings.
- Output Layer: A softmax layer predicts the target word.
- Backpropagation: Updates weights using gradient descent.

- Notations

- $V$ : Vocabulary size
- $d$ : Embedding vector size
- $N$ : Context window size
- $X$ : One-hot input matrix
- $X_i$  : Column vector of length  $V$  representing the one-hot representation of words, where  $i \in [1, V]$
- $W \in \mathbb{R}^{d \times V}$ : Input-to-hidden weight matrix (word embeddings)
- $W' \in \mathbb{R}^{V \times d}$ : Hidden-to-output weight matrix
- $Y$ : Output vector (softmax probabilities)



# CBOW Model

## Input Layer

- The context words  $X^{(i)}$ , where i ranges from 1 to V, are represented using one-hot encoding.
- One hot encoded context words  $X^{(i)}$  have dimension (V\*1).

		$X^{(1)}$	$X^{(2)}$	$X^{(3)}$	$X^{(4)}$	$X^{(5)}$
	the	the	cat	sits	on	mat
the	1	0	0	0	0	0
cat	0	1	0	0	0	0
sits	0	0	1	0	0	0
on	0	0	0	1	0	0
mat	0	0	0	0	0	1

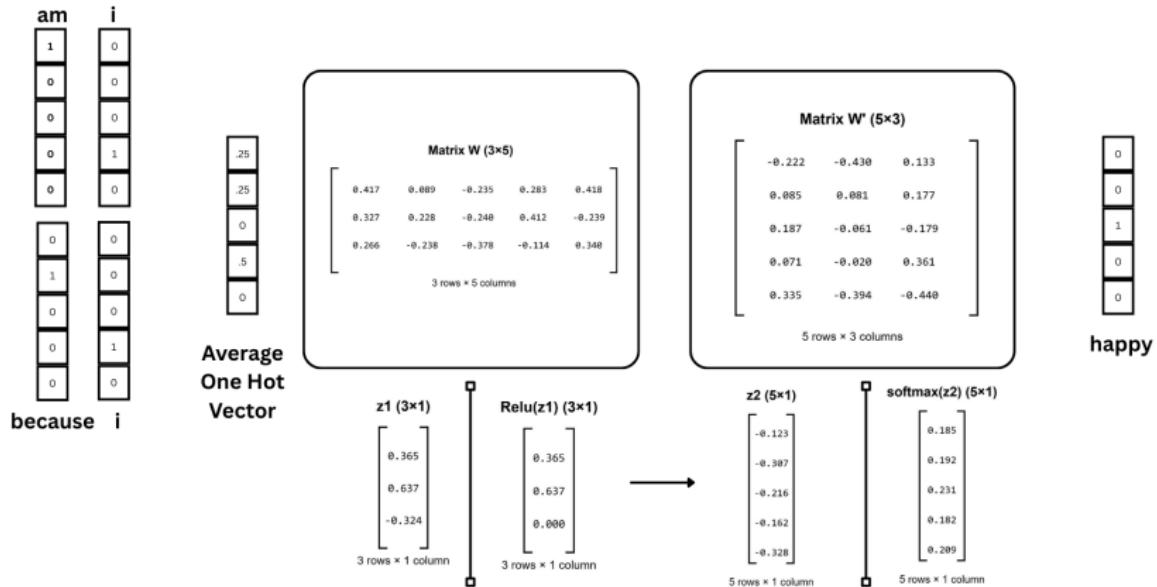
One Hot Encoding

- In One hot encoding, the presence of the word is indicated by a 1. The length of one hot encoded word is equal to the length of the vocabulary (V).



# CBOW Model

## CBOW Illustration



CBOW with vocabulary size of 5 and embedding size of 3



# CBOW Model

## Hidden Layer

- Input layer is the one-hot representation of the word  $X^{(i)}$ , where i ranges from 1 to V.
- The Input-to-Hidden Weight Matrix ( $W_{d*V}$ ) converts context words from the one-hot input representation into a continuous vector space of length V, where V is the vocabulary size or  $h_{d*1}(X^{(i)}) = W_{d*V} \cdot X_{V*1}^{(i)}$
- The hidden layer output  $h_{d*1}(X^{(i)})$  or the embedding for each input word  $X_i$  is of order d\*1.
- For a set of context words determined by the window size, the net hidden layer representation is obtained by averaging the embeddings of 'N' one hot encoded context words.

$$h_{d*1} = \left( \frac{1}{N} \sum_{i=1}^N W X^{(i)} \right) \quad (24)$$



# CBOW Model

## Output Layer

- The Hidden-to-Output Weight Matrix  $W'_{(V*N)}$  transforms the hidden layer representation into a probability distribution over the vocabulary V using softmax, where d is the embedding vector size and V is the vocabulary size.

$$u_{V*1} = W'_{(V*d)} h_{(d*1)} \quad (25)$$

- $u_{V*1}$  is represented as  $[u_1 \ u_2 \ ..u_V]^T$
- The final output probability distribution  $\hat{y}_{(V*1)}^{(i)}$  is computed using the softmax function:

$$\hat{y}^{(i)} = \text{Softmax}(u_{V*1}) = \left[ \frac{e^{u_1}}{\sum_j e^{u_j}}, \frac{e^{u_2}}{\sum_j e^{u_j}}, \dots, \frac{e^{u_V}}{\sum_j e^{u_j}} \right]^T \quad (26)$$

- In short, for each instance of one hot encoded context words, chosen based on the window size, a target vector  $\hat{y}^{(i)}$  is predicted and compared with the reference target  $y^{(i)}$  in one hot encoded format.



## CBOW Model - Linear Softmax

- $|V|$ : Vocabulary size
- $d$ : Embedding dimension
- $N$ : Context size
- $W_E \in \mathbb{R}^{|V| \times d}$ : Embedding matrix
- $W_O \in \mathbb{R}^{d \times |V|}$ : Output weight matrix
- $w_1, w_2, \dots, w_N$ : Context word indices
- $y \in \{1, \dots, |V|\}$ : Target word index
- $\mathbf{1}_i \in \mathbb{R}^{|V|}$ : One-hot vector for word  $i$
- Linear Softmax has only averaging and softmax.



# CBOW Model - Linear Softmax

## Forward Pass

- 1. Embedding Lookup and Averaging

$$x = \frac{1}{N} \sum_{i=1}^N W_E^\top \mathbf{1}_{w_i} \in \mathbb{R}^d$$

- 2. Output Scores

$$z = W_O^\top x \in \mathbb{R}^{|V|}$$

- 3. Softmax Prediction

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} \quad \text{for } j = 1, \dots, |V|$$

- 4. Loss (Cross-Entropy)

$$\mathcal{L} = -\log(\hat{y}_y)$$



# CBOW Model - Linear Softmax

## Gradient Computation

- 1. Gradient w.r.t. Output Scores

$$\frac{\partial \mathcal{L}}{\partial z} = \hat{y} - y_{\text{true}} \in \mathbb{R}^{|V|}$$

- 2. Gradient w.r.t. Output Weights

$$\frac{\partial \mathcal{L}}{\partial W_O} = x \cdot (\hat{y} - y_{\text{true}})^\top$$

- 3. Gradient w.r.t. Averaged Embedding

$$\frac{\partial \mathcal{L}}{\partial x} = W_O \cdot (\hat{y} - y_{\text{true}})$$

- 4. Gradient w.r.t. Embedding Matrix

$$\frac{\partial \mathcal{L}}{\partial W_E[w_i]} = \frac{1}{N} \cdot \frac{\partial \mathcal{L}}{\partial x} \quad \text{for each } w_i$$



## CBOW Model - Linear Softmax

### Weight Updation

- Updating the Output Weight Matrix  $W_O$  :

$$W_O \leftarrow W_O - \eta \cdot \frac{\partial \mathcal{L}}{\partial W_O}$$

- Updating the Embedding Matrix  $W_E$  :

$$W_E[w_i] \leftarrow W_E[w_i] - \eta \cdot \left( \frac{\partial \mathcal{L}}{\partial W_E[w_i]} \right) \quad \forall i = 1, \dots, N$$

$$W_E[w_i] \leftarrow W_E[w_i] - \eta \cdot \left( \frac{1}{N} \cdot \frac{\partial \mathcal{L}}{\partial x} \right) \quad \forall i = 1, \dots, N$$



## CBOW Model - Linear Softmax

$$\frac{\partial \mathcal{L}}{\partial z} = \hat{y} - y_{\text{true}}$$

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} \quad \text{for } j = 1, \dots, |V|$$

$$\mathcal{L} = -\log(\hat{y}_y) = -\log\left(\frac{e^{z_y}}{\sum_k e^{z_k}}\right) = -z_y + \log\left(\sum_{k=1}^{|V|} e^{z_k}\right)$$

Case 1:  $j = y$

$$\frac{\partial \mathcal{L}}{\partial z_j} = \frac{\partial}{\partial z_j} \left( -z_y + \log\left(\sum_{k=1}^{|V|} e^{z_k}\right) \right) = -1 + \frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} = \hat{y}_j - 1$$

Case 2:  $j \neq y$

$$\frac{\partial \mathcal{L}}{\partial z_j} = \frac{\partial}{\partial z_j} \left( \log\left(\sum_{k=1}^{|V|} e^{z_k}\right) \right) = \frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} = \hat{y}_j$$

$$\frac{\partial \mathcal{L}}{\partial z_j} = \hat{y}_j - y_j$$

where  $y_j = 1$  if  $j = y$ , and 0 otherwise.



## CBOW Model - Linear Softmax

$$\boxed{\frac{\partial \mathcal{L}}{\partial W_O} = x \cdot (\hat{y} - y_{\text{true}})^T}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \hat{y} - y_{\text{true}} \in \mathbb{R}^{|V|}$$

$$z = W_O^\top x \Rightarrow z_j = x^\top w_{o,j}$$

where  $w_{o,j}$  is the  $j$ -th column of  $W_O$ .

$$\frac{\partial z_j}{\partial w_{o,j}} = x$$

Using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial W_O} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial W_O}$$

Since  $z = W_O^\top x$ , we use matrix calculus:

$$\frac{\partial \mathcal{L}}{\partial W_O} = x \cdot (\hat{y} - y_{\text{true}})^T$$



## CBOW Model - Linear Softmax

$$\frac{\partial \mathcal{L}}{\partial x} = W_O(\hat{y} - y_{\text{true}})$$

$$\frac{\partial \mathcal{L}}{\partial z} = \hat{y} - y_{\text{true}} \quad \text{and} \quad z = W_O^\top x$$

$$\frac{\partial \mathcal{L}}{\partial x} = W_O(\hat{y} - y_{\text{true}})$$

Using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial x} = (\hat{y} - y_{\text{true}})^\top W_O^\top = W_O (\hat{y} - y_{\text{true}})$$



## CBOW Model - Linear Softmax

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_i}} = \frac{1}{N} W_O (\hat{y} - y_{\text{true}})} \quad \text{for } i = 1, \dots, N$$

Since the average is a linear operation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_i}} = \frac{\partial \mathcal{L}}{\partial x} \cdot \frac{\partial x}{\partial \mathbf{v}_{w_i}} = \frac{1}{N} \cdot \frac{\partial \mathcal{L}}{\partial x}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_i}} = \frac{1}{N} W_O (\hat{y} - y_{\text{true}})} \quad \text{for } i = 1, \dots, N$$



# CBOW Model with Hidden Layer

## Training - Forward Pass

- Vocabulary size:  $|V|$
- Context size:  $N$
- Embedding dimension:  $d$
- Hidden layer size:  $h$
- Hidden activation: Sigmoid
- Output activation: Softmax

$$x = \frac{1}{N} \sum_{i=1}^N W_E^\top w_i \in \mathbb{R}^d$$

$$z_1 = W_1^\top x + b_1$$

$$h = \sigma(z_1) = \frac{1}{1 + e^{-z_1}} \in \mathbb{R}^h$$

$$z_2 = W_2^\top h + b_2 \in \mathbb{R}^{|V|}$$

$$\hat{y} = \text{Softmax}(z_2)$$

$$\mathcal{L} = -\log(\hat{y}_y)$$



# CBOW Model with Hidden Layer

## Training - Computation of Gradients

### ■ Output Layer :

$$\delta^{(2)} = \hat{y} - y_{\text{true}} \in \mathbb{R}^{|V|}$$

$$\nabla W_2 = h \cdot (\delta^{(2)})^\top \in \mathbb{R}^{h \times |V|}$$

$$\nabla b_2 = \delta^{(2)} \in \mathbb{R}^{|V|}$$

### ■ Hidden Layer: Sigmoid Activation

$$\sigma'(z_1) = \sigma(z_1) \cdot (1 - \sigma(z_1)) = h \cdot (1 - h)$$

$$\delta^{(1)} = (W_2 \cdot \delta^{(2)}) \odot \sigma'(z_1) = (W_2 \cdot \delta^{(2)}) \odot h \cdot (1 - h)$$

$$\nabla W_1 = x \cdot (\delta^{(1)})^\top \in \mathbb{R}^{d \times h}$$

$$\nabla b_1 = \delta^{(1)} \in \mathbb{R}^h$$

### ■ Embedding Layer

$$\delta^{(\text{embed})} = W_1 \cdot \delta^{(1)} \in \mathbb{R}^d$$

$$\nabla W_E[w_i] = \frac{1}{N} \cdot \delta^{(\text{embed})} \quad \text{for each } w_i \in \text{context}$$



# CBOW Model with Hidden Layer

## Training - Weight Updation

Let  $\eta$  be the learning rate.

- Output Layer :

$$\begin{aligned}W_2^{(t+1)} &= W_2^{(t)} - \eta \cdot \nabla W_2 \\b_2^{(t+1)} &= b_2^{(t)} - \eta \cdot \nabla b_2\end{aligned}$$

- Hidden Layer :

$$\begin{aligned}W_1^{(t+1)} &= W_1^{(t)} - \eta \cdot \nabla W_1 \\b_1^{(t+1)} &= b_1^{(t)} - \eta \cdot \nabla b_1\end{aligned}$$

- Embedding Matrix :

$$W_E[w_i]^{(t+1)} = W_E[w_i]^{(t)} - \eta \cdot \left( \frac{1}{N} \cdot \delta^{(\text{embed})} \right)$$



# Principal Component Analysis



# Principal Component Analysis

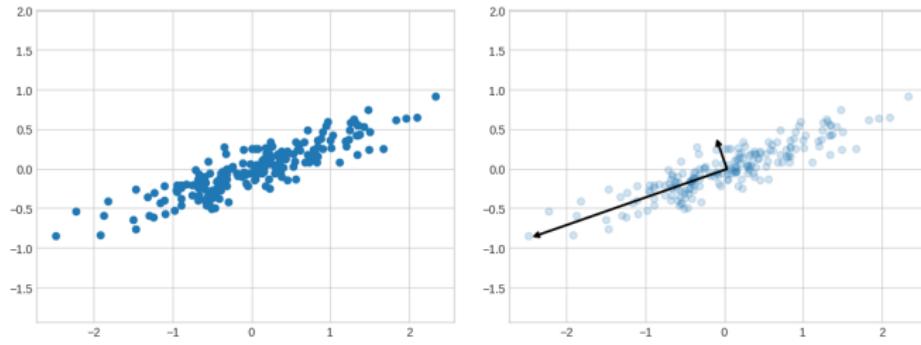
## Introduction

- Principal Component Analysis (PCA) is a dimensionality reduction ML technique to transform a large set of correlated variables into a smaller set of uncorrelated variables called principal components.
- These principal components retain most of the original dataset's variance, making PCA useful for simplifying complex datasets while minimizing information loss.
- Key Aspects : PCA
  - **Dimensionality Reduction:** Reduces dimensions of the data without losing significant information, making it easier to visualize and analyze.
  - **Feature Extraction:** Identifies the most important features (principal components) that explain the most variance in the data.
  - **Orthogonal Transformation:** The principal components are orthogonal to each other, ensuring no dependence with each other.
  - **Independence:** PCA transforms the original variables into a new set of uncorrelated variables (the principal components).
- Applications in ML include feature extraction, visualization of high dimensional data and factor analysis.



# Principal Component Analysis

## PCA Illustration 2D⇒1D



Dimensionality Reduction 2D⇒1D

- The black arrows represent the Eigen vectors.
- The length of the arrow  $i$  or the eigen value represents the proportion of variance of original data captured by the principal component  $i$ , where  $i \in [1, 2]$ .
- 2-D data can have maximum 2 principal components.
- The 2-D data projected on the eigen vector, corresponding to the highest eigen value, represents the 1-D reduced data.



# Principal Component Analysis (PCA)

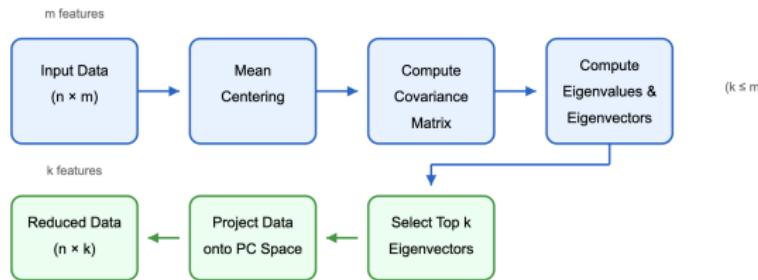
## PCA Definition

### PCA

- Method for Dimensionality Reduction as it reduces the  $m$  observed features to  $k$  reduced features, where  $k \leq m$ .
- The  $k$  principal components are independent and are expressed as a linear function of observed dependent features  $X_i$ , where  $i$  ranges from 1 to  $m$  and  $n$  is the number of samples.

Principal Component Analysis (PCA) Block Diagram

Dimensionality Reduction from  $m$  to  $k$  features



Principal Component Analysis  $m \rightarrow k$  Dimensionality Reduction



# Principal Component Analysis

## PCA Computation - 6 Steps

- Data Standardization ( $X \Rightarrow Z$ )
  - The idea of PCA is based on capturing the directions in which variance is maximum and variance is sensitive to the scale of the data.
  - Standardization (mean-centering (necessary) and scaling (optional)) ensures that the principal components are not biased towards the features with high variance.
- Covariance Matrix Computation  $C$  represents the Covariance of the standardized matrix.
- Eigen Values  $\lambda$  and Eigen Vectors  $v$  (Principal Component) are computed from Covariance Matrix  $C$  to find the directions in which the variance is maximum.
- Sort Eigen Vectors (Principal Components) based on Eigen Values in descending order.
- Project Data to New Basis defined by Eigen Vectors (Principal Components) and obtain the Principal Component Scores.
- The importance of the principal components are assessed with respect to the percentage of variance in the data explained by the principal components.



# Principal Component Analysis

## PCA - Data Standardization

- **Data Standardization** : The data matrix  $X_{n*m}$  with  $n$  samples and  $m$  features is standardized to obtain the matrix  $Z$ .

$$X_{n*m} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \Rightarrow \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{21} & z_{22} & \cdots & z_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nm} \end{bmatrix}$$

The standardized representation of each feature  $j$ ,  $z_{ij} \sim \mathcal{N}(0, 1)$  , is given by:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

- $\mu_j$  is the mean of feature  $j$ :

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$$

- $\sigma_j$  is the standard deviation of feature  $j$ :

$$\sigma_j = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \mu_j)^2}$$



# Principal Component Analysis

## PCA Computation

- The symmetric covariance matrix  $C_{m*m}$  of the standardized matrix Z captures the relationship between different 'm' features.

$$C_{m*m} = \frac{Z_{n*m}^T \cdot Z_{n*m}}{n} \quad (27)$$

- The eigen values of the symmetric covariance matrix  $C_{m*m}$  satisfies the following condition.

$$Cv_i = \lambda_i v_i \quad (28)$$

where  $v_i$  represents the eigen vector of C corresponding to eigen-value  $\lambda_i$ .

The square matrix  $C_{m*m}$  can have maximum 'm' eigen vectors corresponding to sorted eigen values ( $\lambda_1 > \lambda_2 \dots > \lambda_m$ ), where an eigen vector  $v_i$  is of dimension  $m*1$ , i ranging from 1 to m.

- The Projection Matrix  $V$  of order  $m*k$  is formed by picking the top k Eigen Vectors corresponding to the largest 'k' Eigen values and ( $\lambda_1 > \lambda_2 \dots > \lambda_m$ )

$$V_{m*k} = [v_1, v_2, \dots, v_k] \quad (29)$$



# Principal Component Analysis

## PCA Computation

- The projection matrix  $V_{m*m}$ , with 'm' eigen column vectors, is given by

$$V_{m*m} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mm} \end{bmatrix}$$

- The PCA principal component scores  $Y_{n*k}$  of order  $n*k$  is obtained by projecting the Standardized Data Z of order  $n*m$  onto the Projection Matrix V of order  $m*k$  to obtain

$$Y_{n*k} = Z_{n*m} \cdot V_{m*k} = [y_1 \dots y_2 \dots \dots y_k] \quad (30)$$

where the 'k' columns of the PCA output  $Y_{n*k}$  represent the principal component scores.

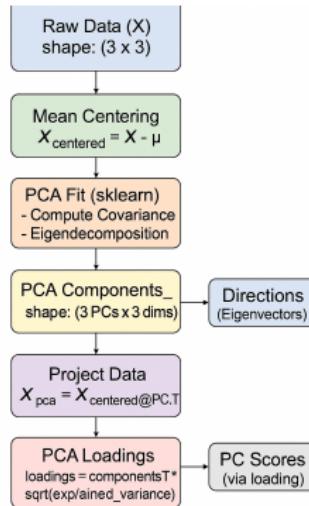
- Variance explained by the  $i^{th}$  principal component is given by

$$\text{Var}(Y_i) = \frac{\lambda_i}{\sum_{i=1}^m \lambda_i} \quad (31)$$



# Principal Component Analysis

## PCA - Steps



Dimensionality Reduction with PCA



# Principal Component Analysis

## PCA Mathematical Model

Let  $\mathbf{X} \in \mathbb{R}^{n \times m}$  be a mean-centered data matrix with n samples and m features.

- The sample covariance matrix is:

$$\mathbf{C}_{m \times m} = \frac{1}{n-1} \mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{m \times m}$$

- Let  $\mathbf{V} \in \mathbb{R}^{m \times m}$  be the matrix of eigenvectors of  $\mathbf{C}$ , such that:

$$\mathbf{C}\mathbf{V} = \Lambda\mathbf{V}$$

where  $\Lambda \in \mathbb{R}^{m \times m}$  is a **diagonal** matrix of eigenvalues.

- $C.V_i = \lambda_i.V_i$ , where i ranges from 1 to m. Since the matrix C is symmetric, the eigen values  $\lambda_i$  of the matrix C would be real.
- The principal component scores matrix  $\mathbf{Y} \in \mathbb{R}^{n \times m}$  is given by:

$$\boxed{\mathbf{Y}_{n \times m} = \mathbf{X}_{n \times m} \mathbf{V}_{m \times m}}$$

- To retain only the top  $k$  components, let  $\mathbf{V}_k \in \mathbb{R}^{m \times k}$  be the matrix of the top  $k$  eigenvectors. Then:

$$\boxed{\mathbf{Y}_k = \mathbf{X}\mathbf{V}_k, \quad \mathbf{Y}_k \in \mathbb{R}^{n \times k}}$$



# Principal Component Analysis

## PCA Model - Matrix Notations

The goal of PCA analysis is to reduce the dimension of the data set from m to k.  $X_{n*m}$  is the standardized data set and  $C_{m*m}$  is the covariance of the standardized dataset  $X_{n*m}$ .

$$X_{n*m} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}$$

$$C_{m*m} = Cov(X) = \begin{bmatrix} var(x_1) & c_{12} & c_{1m} \\ c_{21} & var(x_2) & c_{2m} \\ \dots & \dots & \dots \\ c_{m1} & c_{m2} & var(x_m) \end{bmatrix}$$

$c_{ij}$  represents the covariance between the feature  $Z_i$  and  $X_j$ , where  $i, j \in [1, m]$

$$V_{m*m} = EigenVectors(C) = \begin{bmatrix} v_{11} & v_{12} & v_{1m} \\ v_{21} & v_{22} & v_{2m} \\ \dots & \dots & \dots \\ v_{m1} & v_{m2} & v_{mm} \end{bmatrix}$$



# Principal Component Analysis

PCA Model - Principal Components as Linear Combination of Input Features

The principal components  $y_i$ , i ranging from 1 to k, are linear combinations of the m input features or variables  $x_i$ , i ranging from 1 to m or  $\mathbf{Y}_k = \mathbf{X}\mathbf{V}_k$ .

$$Y_1 = v_{11}X_1 + v_{21}X_2 + \dots v_{m1}X_m + \epsilon_1$$

$$Y_2 = v_{12}X_1 + v_{22}X_2 + \dots v_{m2}X_m + \epsilon_2$$

$$Y_3 = v_{13}X_1 + v_{23}X_2 + \dots v_{m3}X_m + \epsilon_3$$

$$Y_4 = v_{14}X_1 + v_{24}X_2 + \dots v_{m4}X_m + \epsilon_4$$

$$Y_5 = v_{15}X_1 + v_{25}X_2 + \dots v_{m5}X_m + \epsilon_5$$

$$Y_6 = v_{16}X_1 + v_{26}X_2 + \dots v_{m6}X_m + \epsilon_6$$

$$Y_k = v_{1k}X_1 + v_{2k}X_2 + \dots v_{mk}X_m + \epsilon_k$$

Note that the covariance matrix  $C_{m*m}$  can be expressed as a function of eigen vector matrix  $V_{m*k}$  and diagonal eigen value matrix  $\Phi_{k*k}$

$$C_{m*m} \approx V_{m*k} \cdot \Phi_{k*k} \cdot V_{k*m}^T \quad (32)$$



# Principal Component Analysis

## PCA Interpretation

- The first principal component  $y_1$  is the linear combination of X-variables that has maximum variance (among all linear combinations).
- The second principal component  $y_2$  is the linear combination of X-variables that accounts for as much of the remaining variation as possible and the second principal component is independent of the first component
- The  $m^{th}$  principal component  $Y_m$  is the linear combination of x-variables that accounts for as much of the remaining variation not explained by  $(m-1)$  components and the  $m^{th}$  principal component is independent of all the other  $(m-1)$  principal components.

### Loadings of PCA

The correlation between the feature  $X_i$ , i ranging from 1 to m, and the principal component  $Y_j$ , j ranging from 1 to m, is called the loading  $l_{ij}$ . The loading matrix L is given by

$$L_{m*k} = V_{m*k}.D_{k*k} \quad (33)$$

where  $D_{k*k}$  is a diagonal square matrix with elements  $(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \dots, \sqrt{\lambda_k})$ .



# Principal Component Analysis

## PCA Interpretation

- The total variance of the data is equal to the sum of diagonal elements of Covariance Matrix  $C$ .

$$V(X) = \text{Trace}(C) = \lambda_1 + \lambda_2 + \dots + \lambda_m \quad (34)$$

- Variance of the data explained by principal component  $Y_i$  is equal to the corresponding eigen value or  $V[Y_i] = \lambda_i$ , where  $(\lambda_1 > \lambda_2 > \lambda_m)$
- Percentage of variance explained by the  $i^{th}$  principal component is given by  $\frac{\lambda_i}{\sum_{i=1}^k \lambda_i}$ .
- Efficiency of PCA  $\eta$  reflecting the ability of the  $k$  principal components to explain the variance of the dataset  $X_{n*m}$ .

$$\eta = \frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^m \lambda_i} \quad (35)$$

- The principal component  $Y_i$ ,  $i$  ranging from 1 to  $m$ , are also called as the latent factors or hidden features.



## PCA Illustration

$$X_{3 \times 3} = \begin{bmatrix} 2 & 0 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 2 \end{bmatrix}$$

$$X_{std} = \begin{bmatrix} 1.33333333 & 0 & -1.66666667 \\ -1.66666667 & 1 & 0.33333333 \\ 0.33333333 & -1 & 1.33333333 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.74743799 & -0.16636188 & -0.64316419 \\ -0.40442388 & 0.65412482 & -0.63918859 \\ 0.52704628 & 0.73786479 & 0.42163702 \end{bmatrix}$$

$$\lambda_1 = 3.55, \lambda_2 = 2.10, \lambda_3 = 0$$

$$Y = \begin{bmatrix} 2.0685243 & 0.52608247 & 0 \\ -1.62647993 & 1.11510176 & 0 \\ -0.44204437 & -1.64118423 & 0 \end{bmatrix}$$

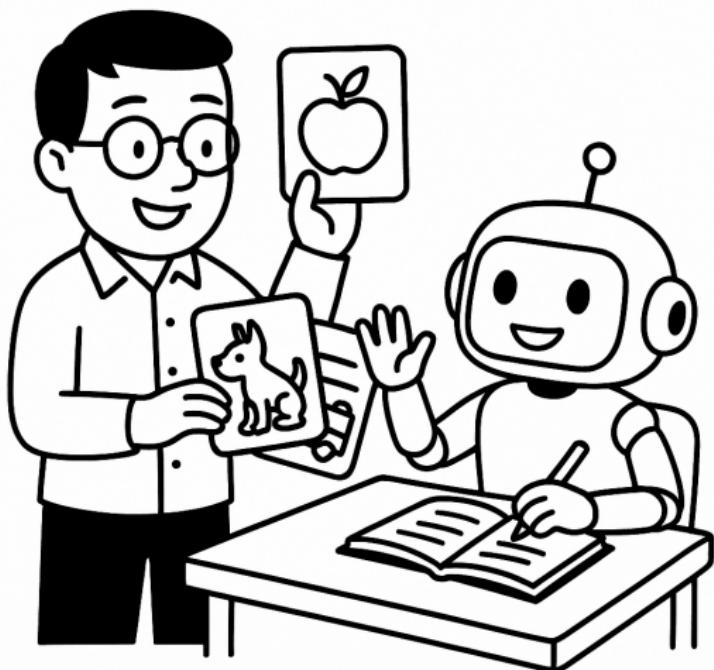
$$Y_{3 \times 2} = \begin{bmatrix} 2.0685243 & 0.52608247 \\ -1.62647993 & 1.11510176 \\ -0.44204437 & -1.64118423 \end{bmatrix}$$



# **Supervised Learning**



# Supervised Learning



Supervised Learning



# Supervised Learning

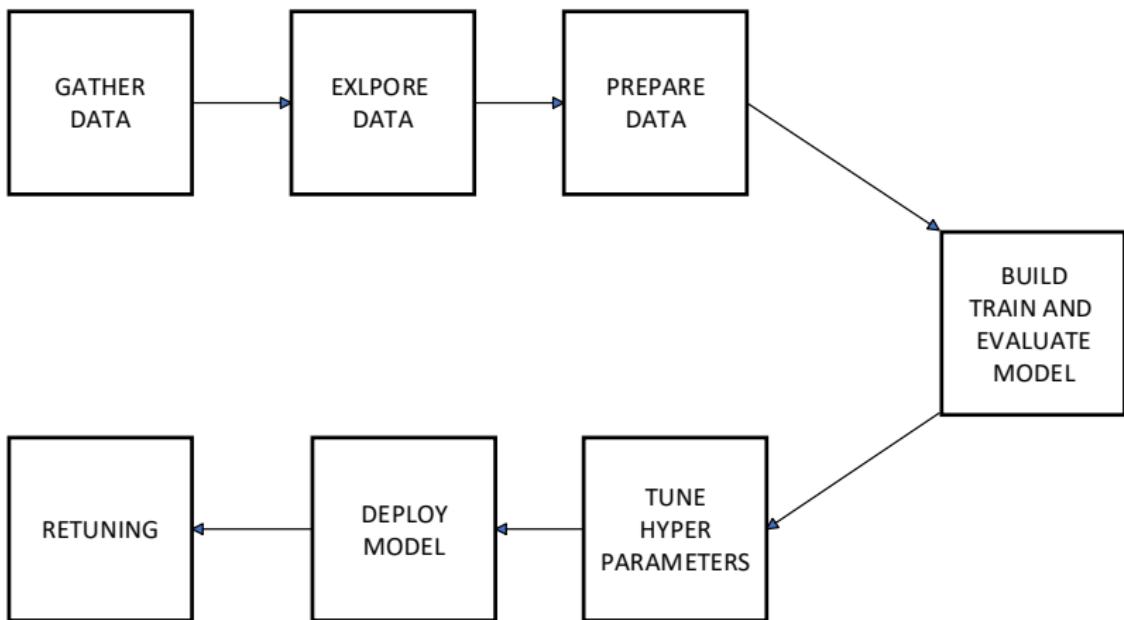
Labelled Matrix  $D=X|Y$  - Representation

$$D = \left[ \begin{array}{cccc|c} x_{11} & x_{12} & \cdots & x_{1m} & y_1 \\ x_{21} & x_{22} & \cdots & x_{2m} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} & y_n \end{array} \right] \in \mathbb{R}^{n \times (m+1)} = [X_{\text{features}} \quad | \quad Y_{\text{target}}]$$

- The data matrix  $D \in \mathbb{R}^{n \times (m+1)}$ , the labelled data set, represents the training samples X and the target variable Y.
- $X_{\text{features}} \in \mathcal{R}^{m \times n}$  refers to feature data set, where each training sample has m features.
- $Y_{\text{training}} \in \mathcal{R}^n$  refers to the target variable.
- m indicates the number of features and n indicates the number of samples.



# Supervised Learning



Steps to Solve a Supervised ML Problem



# Types of Supervised Learning

Supervised  
Learning  
Models

Regression  
Models

Classification  
Models

Types of Supervised Learning Models : Regression and Classification



# **Bayes Classification**



## Naive-Bayes Classifier

Naive-Bayes Classifier is a supervised learning algorithm used for text classification, recommendation systems and spam filtering applications.

$$P\left(\frac{Y = C_i}{X}\right) = \frac{P\left(\frac{X}{Y=C_i}\right) \cdot P(Y = C_i)}{P(X)} \quad (36)$$

where:

- $P\left(\frac{Y=C_i}{X}\right)$  is called Posterior Probability.
- $P\left(\frac{X}{Y=C_i}\right)$  is called Likelihood Probability.
- $P(Y=C_i)$  is called the Prior Probability.
- $P(X)$  is called the Marginal Probability or Evidence. (Same across classes)
- $X$  refers to the features and  $C_i$  refers to the class,  $i \in [1, k]$ ,  $k$  is number of classes.



# Bayes Classifier

## Naive-Bayes Gaussian Classifier

- The goal of the Bayes Theorem is to predict the probability that the dependent variable  $Y$  can belong to class  $C_i$ , given the  $n$  **independent** features  $X = [X_1, X_2, X_3, X_n]$ .
- Naive-Bayes Classifier is a generative classifier as it can generate data by modelling the class conditional joint density function  $P\left(\frac{X_1, X_2, X_n}{Y=y}\right)$ .

$$P(X|y) = \prod_{i=1}^n P(X_i|y) \quad (37)$$

- The  $x_1, x_2, x_3, x_n$  can be binary variable or continuous variable.
- When the features  $x_1, x_2, x_3, x_n$  are continuous and independent, and follows a normal distribution  $X_i \sim \mathcal{N}(\mu, \sigma^2)$ , the Bayes Classifier is called a Gaussian Naive Bayes Classifier.

$$P(X_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(X_i - \mu_y)^2}{2\sigma_y^2}\right) \quad (38)$$

where  $\mu_y$  and  $\sigma_y^2$  are the mean and variance of feature  $x_i$  for class  $y$ .



# Bayes Classifier

## Naive-Bayes Gaussian Classifier

- The Bayes rule compute the posterior probability as a function of likelihood, prior probability and marginal probability.

$$P\left(\frac{Y}{X}\right) = \frac{P\left(\frac{X}{Y}\right) \cdot P(Y)}{P(X)} \quad (39)$$

$$P\left(\frac{Y}{X}\right) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(X_i - \mu_y)^2}{2\sigma_y^2}\right) \cdot P(Y) \quad (40)$$

- Since  $P(X)$  is the same for all classes,  $P(X)$  is ignored in the posterior probability computation.
- The Bayes classifier assigns the class with the highest posterior probability to the target variable Y.

$$\hat{Y} = \arg \max_{C_i} P(Y = C_i | X) = \arg \max_{C_i} P(X | Y = C_i) \cdot P(Y = C_i) \quad (41)$$



# Bayes Classifier

Illustration of Gaussian Bayes Classifier

Task - Classifying emails as **Spam** or **Not Spam** using 2 features,  $x_1$ : Number of special characters (e.g., !, \$, @),  $x_2$ : Proportion of capital letters in the text.

Class	$P(y)$	$\mu_1$	$\sigma_1^2$	$\mu_2$	$\sigma_2^2$
Spam	0.4	10	4	0.6	0.01
Not Spam	0.6	3	1	0.2	0.02

Given a new email arrives with the following feature values:

$$x_1 = 8, \quad x_2 = 0.5$$

, what is the chance that the email is spam or non-spam?

$$P(y | x_1, x_2) \propto P(y) \cdot P(x_1 | y) \cdot P(x_2 | y)$$

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right)$$

Choose the class with the higher total log score:

$$\hat{y} = \arg \max_{C_i} \log P(Y = C_i | x_1, x_2)$$



# Bayes Classifier

## Illustration of Gaussian Bayes Classifier

$$\log P(\text{Spam}) = \log(0.4) = A_{\text{spam}}$$

$$\log P(x_1 = 8 | \text{Spam}) = \log \left( \frac{1}{\sqrt{2\pi \cdot 4}} \exp \left( -\frac{(8 - 10)^2}{2 \cdot 4} \right) \right) = B_{\text{spam}}$$

$$\log P(x_2 = 0.5 | \text{Spam}) = \log \left( \frac{1}{\sqrt{2\pi \cdot 0.01}} \exp \left( -\frac{(0.5 - 0.6)^2}{2 \cdot 0.01} \right) \right) = C_{\text{spam}}$$

$$\text{Total Score (Spam)} = A_{\text{spam}} + B_{\text{spam}} + C_{\text{spam}}$$

$$\log P(\text{Not Spam}) = \log(0.6) = A_{\text{notspam}}$$

$$\log P(x_1 = 8 | \text{Not Spam}) = \log \left( \frac{1}{\sqrt{2\pi \cdot 1}} \exp \left( -\frac{(8 - 3)^2}{2 \cdot 1} \right) \right) v = B_{\text{notspam}}$$

$$\log P(x_2 = 0.5 | \text{Not Spam}) = \log \left( \frac{1}{\sqrt{2\pi \cdot 0.02}} \exp \left( -\frac{(0.5 - 0.2)^2}{2 \cdot 0.02} \right) \right) = C_{\text{notspam}}$$

$$\text{Total Score (Not Spam)} = A_{\text{notspam}} + B_{\text{notspam}} + C_{\text{notspam}}$$



## Bayes Classifier : Fruit Problem

Given the class of the fruit, generate the joint probability distribution of the features of the fruit

Given the features of the fruit, give the probability that the fruit belongs to a class

Y Fruit	X1 Yellow	X2 Sweet	X3 Long	Total
Mango	350	450	0	650
Banana	400	300	350	400
Others	50	100	50	150
Total	800	850	400	1200

Frequency Table

Given that the fruit is Mango, what is the chance that the fruit is yellow,sweet and long? **0**

Given that the fruit is Banana, what is the chance that the fruit is yellow,sweet and long ? **.6562**

Given that the fruit is Others, what is the chance that the fruit is yellow,sweet and long ? **.07742**



## Bayes Classifier : Fruit Problem

All the features are Independent : Naive

$$P\left(\frac{X_1, X_2, X_3}{Y}\right) = P\left(\frac{X_1}{Y}\right).P\left(\frac{X_2}{Y}\right).P\left(\frac{X_3}{Y}\right)$$

$$P\left(\frac{X_i}{Y}\right) = \frac{P(X_i \cup Y)}{P(Y)}$$

$$P\left(\frac{X_i}{Y}\right) = \frac{P\left(\frac{Y}{X_i}\right).P(X_i)}{P(Y)}$$

i is index ranging from 1 to 3

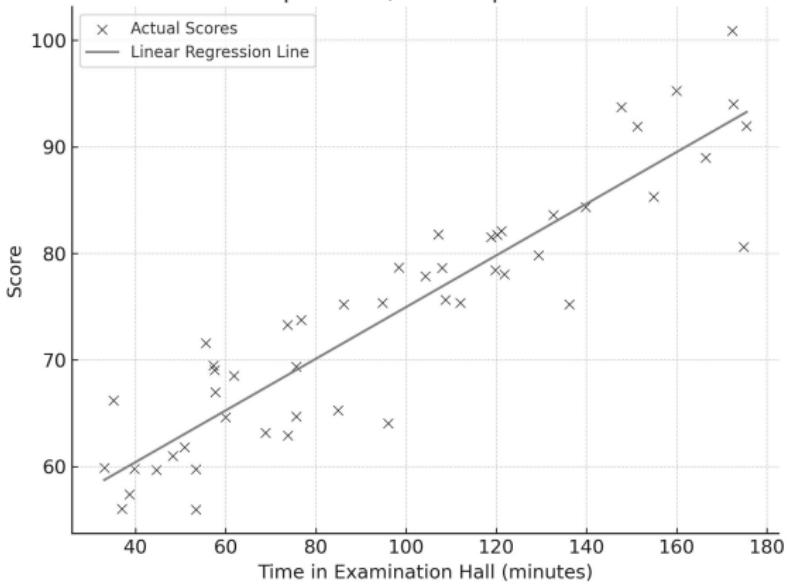


# Linear Regression



# Linear Regression

Linear Regression: Score vs Time in Exam Hall  
Slope: 0.24, Intercept: 50.71



Linear Regression : Exam Score and Time



## Linear Regression

Linear regression models the linear relationship between a set of  $m$  dimensional input features  $x_i$  (predictors) and an output target variable  $y_i$  (response).

- For  $n$  data samples, each with input vector  $x_i \in \mathbb{R}^m$ ,  $i = 1, 2, \dots, n$ , target variable  $y_i \in \mathbb{R}$  and predicted output  $\hat{y}_i \in \mathbb{R}$ , the linear regression model is given as

$$y_i = \hat{y}_i + \epsilon_i = \mathbf{w}^\top \mathbf{x}_i + b + \epsilon_i \quad (42)$$

where:

- $\mathbf{w} \in \mathbb{R}^m$ :  $m$  dimensional weight vector (model parameters).
- $b \in \mathbb{R}$ : bias term (intercept),  $\epsilon_i$  is the error.
- $\mathbf{w}^\top \mathbf{x}_i$ : dot product of the weights and input vector.



# Linear Regression

## Matrix Notation of Predicted Output

The predicted regression outputs  $\hat{y}$  for all  $n$  samples with  $m$  features are:

$$\hat{y}_{(n \times 1)} = X_{(n \times m)} \mathbf{w}_{(m \times 1)} + \mathbf{b} \cdot \mathbf{1}_{(n \times 1)}$$

$$y = \hat{y} + \epsilon$$

where

- $X \in \mathbb{R}^{n \times m}$  is the Design matrix , where the  $i$ -th row is  $x_i^\top$ .
- $\hat{y} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n]^\top \in \mathbb{R}^n$  is the predicted output.
- $y = [y_1, y_2, \dots, y_n]^\top \in \mathbb{R}^n$  is the target output.
- $\mathbf{w} \in \mathbb{R}^m$  is the weight vector.
- $\mathbf{1} \in \mathbb{R}^n$  is a column vector of 'n' ones.
- The total number of parameters to be estimated is  $n+1$ , where  $n$  is the number of weights and  $b$  is the bias or intercept term.
- $\epsilon = [\epsilon_1, \epsilon_2, \dots, \epsilon_n]^\top \in \mathbb{R}^n$  is the error in model.



# Linear Regression

## Loss Function - MSE

The goal is to minimize the loss function  $L(\mathbf{w}, \mathbf{b})$ , Mean Squared Error (MSE), that represents the deviation between predicted and target outputs.

$$\min_{\mathbf{w}, b} L(\mathbf{w}, \mathbf{b}) = \text{MSE} = \frac{1}{n} \sum_{i=1}^n e_i^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{\|y - \hat{y}\|^2}{n}$$

where

- $n$  is the total number of samples
- $y_i$  - Observed output for sample  $i$ ,  $i \in [1, n]$
- $\hat{y}_i$  - Target output for sample  $i$ ,  $i \in [1, n]$

Substituting  $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i + \mathbf{b}$ , the MSE becomes:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i - \mathbf{b})^2 \quad (43)$$



# Linear Regression

## Parameter Estimation

Linear Regression, modelling the relationship between a dependent variable  $y$  and independent variables  $X$ , in matrix notation is given by:

$$y_{(n+1)} = \tilde{X}_{n*(m+1)} \tilde{w}_{(m+1)*1} + \varepsilon_{(n+1)} \quad (44)$$

where:

- $\tilde{X}$  is the  $n*(m+1)$  feature matrix containing the  $m$  features.<sup>11</sup>
- $\tilde{w}$  is the  $m+1$  parameter vector that includes the parameters for the features and the intercept term  $b$  or  $\tilde{w} = [w^\top, b]^\top$
- $\varepsilon$  represents the error term.

The objective is to search for the weights  $w$  and bias  $b$  that minimizes the Mean Squared Error (MSE) cost function:

$$\min_{w,b} L(w, b) = J(\tilde{w}) = \frac{1}{2} \|X\tilde{w} - y\|^2 \quad (45)$$

---

<sup>11</sup>Last column is full of 1 for the intercept term



# Linear Regression

## Parameter Estimation

Taking the gradient of  $J(\tilde{w})$  with respect to  $\tilde{w}$ :

$$\nabla J(\tilde{w}) = X^T(X\tilde{w} - y) \quad (46)$$

Setting the derivative to zero for minimization:

$$\tilde{X}^T\tilde{X}\tilde{w} = \tilde{X}^Ty \quad (47)$$

Solving for  $\tilde{w}$  gives the **Normal Equation**:

$$\tilde{w} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^Ty \quad (48)$$

- If the matrix  $\tilde{X}^T\tilde{X}$  is singular or  $|\tilde{X}^T\tilde{X}| = 0$ , the linear regression does not have a solution.



# Linear Regression

## Assumptions

- Linear Relationship :

$$y_i = \hat{y}_i + \epsilon_i = \mathbf{w}^\top \mathbf{x}_i + b + \epsilon_i \quad (49)$$

- Independence of Errors

$$\text{Cov}(\epsilon_i, \epsilon_j) = 0 \quad (50)$$

- Normality of Errors

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2) \quad (51)$$

- Homoskedasticity or Constant Variance

$$\text{Var}(\epsilon) = \sigma^2 \quad (52)$$

- No Multi-Collinearity

$$|X^\top X| \neq 0 \quad (53)$$



# Linear Regression

## Coefficient Estimates

- The least squares estimate of  $\tilde{w}$  is given by  $(\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$ .
- Intercept  $b$  is the expected value of  $y_i$  when all  $m$  features  $x_{ij} = 0$ , where  $j$  ranges from 1 to  $m$ .
- Slope  $w_i$ : Change in  $y_i$  for a one-unit increase in  $x_i$ , holding all other variables  $x_j$  ( $j \neq i$ ) constant.
- The variance-covariance matrix of  $\tilde{w}$  is:

$$\text{Var}(\tilde{w}) = \sigma^2 (\tilde{X}^T \tilde{X})^{-1} \quad (54)$$

where  $\sigma^2$  is the variance of residuals:

- Variance of Residuals  $\sigma^2$

$$\sigma^2 = \frac{\sum (y_i - \hat{y}_i)^2}{n - (p + 1)} \quad (55)$$

- The standard error (SE) of  $w_j$  is:

$$SE(\hat{w}_j) = \sqrt{\sigma^2 [(\tilde{X}^T \tilde{X})^{-1}]_{jj}} \quad (56)$$



# Linear Regression

## Coefficient Estimates

- Standard error in coefficient estimates measure the uncertainty in estimated coefficients  $\hat{\mathbf{w}}$ .

$$\hat{\mathbf{w}} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

- Confidence Interval of Linear Regression Estimates (CI)

$$\hat{w}_j \pm t_{\frac{\alpha}{2}, n-(p+1)} \cdot SE(\hat{w}_j) \quad (57)$$

where

■  $\hat{w}_j$  - Coefficient Estimates

■  $t_{\frac{\alpha}{2}, n-(p+1)}$  - Critical value from the t-distribution with  $n-(p+1)$  degrees of freedom.

- A narrower CI suggests a more precise estimate of  $w_j$ , whereas a wider CI indicates more uncertainty.



## Coefficient of Determination $R^2$

- The coefficient of determination  $R^2$  measures the proportion of the variance in the dependent variable  $y_i$ , that is explained by the linear regression model with independent variables  $x_{i1}, x_{i2} \dots x_{in}$ .
- Total Sum of Squares (SST): Measures the total variance in  $y_i$ :

$$SS_{total} = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (58)$$

- Explained Sum of Squares (SSE): Measures the variance explained by the model  $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i + b$ :

$$SS_{reg} = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 \quad (59)$$

- The coefficient of determination  $R^2$  is defined as

$$R^2 = \frac{SS_{reg}}{SS_{total}} \quad (60)$$



## Adjusted $R^2$

- In multiple linear regression, using  $R^2$  alone can be misleading because adding more predictors always increases  $R^2$ .
- To correct for this, we use **Adjusted  $R^2$** :

$$R_{\text{adj}}^2 = 1 - \left( \frac{(1 - R^2)(n - 1)}{n - p - 1} \right) \quad (61)$$

where:

- $p$  = number of predictors
- $n$  = number of observations

- Adjusted  $R^2$  accounts for the number of predictors, making it a better measure for comparing models.
- $R_{\text{adj}}^2$  is always lesser than or equal to  $R^2$ .



## Variance Inflation Factor

- The Variance Inflation Factor (VIF) measures how much the variance of a regression coefficient is inflated due to multi-collinearity among independent variables.
- For each predictor variable  $x_i$ , the VIF is given by:

$$\boxed{\text{VIF}(x_i) = \frac{1}{1 - R_i^2}} \quad (62)$$

where  $R_i^2$  is the coefficient of determination obtained by regressing  $x_i$  on all other predictors in the model.

- VIF = 1: No multicollinearity (ideal scenario).
- VIF > 1 and < 5: Moderate correlation, generally acceptable.
- VIF > 5: High multicollinearity, may cause instability in regression coefficients.
- VIF > 10: Severe multicollinearity, strong indication that predictor variables are highly correlated.

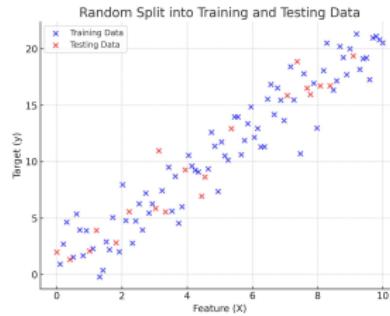


# Cross Validation

## Hold-out validation

Cross-validation is a statistical technique used to assess the generalization ability of a machine learning model by partitioning the dataset into multiple subsets for training and validation.

- The dataset is randomly split into a training set (80%) and a test set (20%).
  - Model is trained on the training set and evaluated on the test set.
  - Performance of model dependent on the chosen split (may lead to bias or variance issues).



Training-Testing



# Cross Validation

## K-fold validation

- K-fold validation

- The dataset is divided into  $k$  equal-sized folds (e.g.,  $k=5$ ).
- The model is trained on  $k-1$  folds and tested on the remaining fold.
- The process repeats  $k$  times, with a different test fold each time.
- The final performance metric is the average score across all  $k$  iterations, also called cross validation score.
- Reduces dependency on a single train-test split.
- Computationally Expensive for large data sets.

- Stratified K-fold validation

- The dataset is divided into  $k$  equal-sized folds (e.g.,  $k=5$ ), where each fold maintains the same proportion of classes as in the full dataset.
- Useful for classification problems with class imbalance

- Leave-One-Out Cross-Validation (LOOCV)

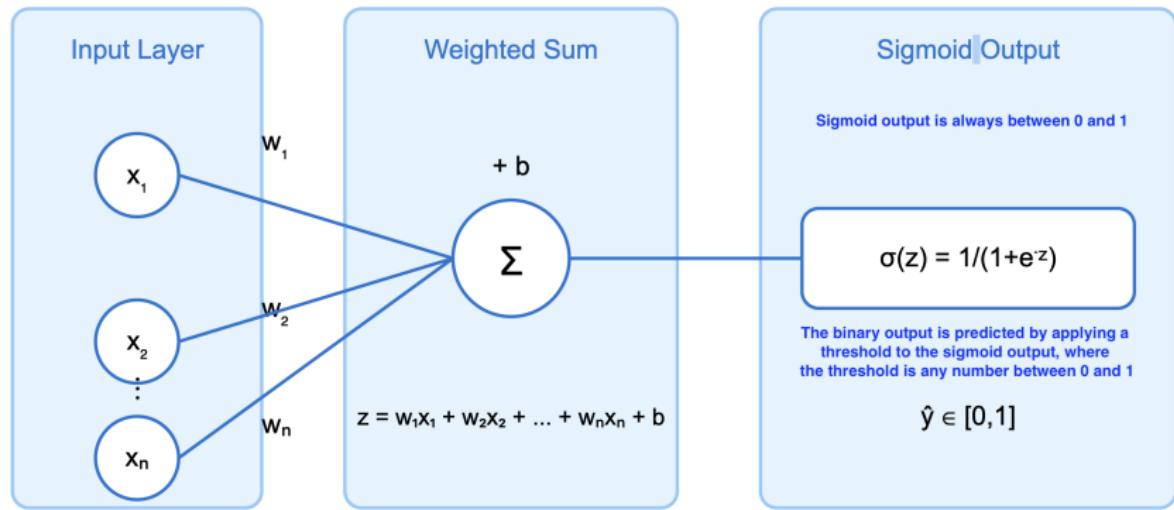
- A special case of k-Fold where  $k = \text{number of samples (n)}$ .
- Each sample is used as a test set once, while all other samples are used for training.
- Computationally expensive for large datasets.



# **Logistic Regression**



# Logistic Regression



Logistic Regression - Weighted inputs and Bias bias is given to the Logistic function to predict the output



# Logistic Regression

- Logistic Regression is a machine learning method for solving binary **classification** problems like spam filtering, fraud detection, credit card approval etc.
- The logistic regression model is based on the sigmoid function  $\sigma(z)$ , defined as:

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

where  $z_i$  is a linear combination of the input features:

$$z_i = w^T x_i + b$$

where  $w$  is weight vector,  $x_i$  is input Feature vector,  $b$  is bias term.

- The sigmoid function outputs a probability of outcome  $y$  given  $x$ .

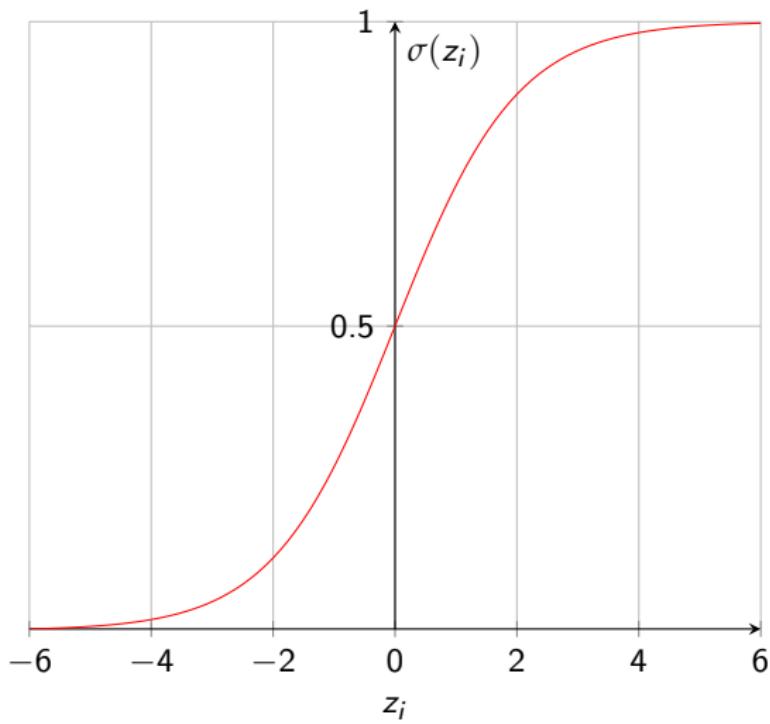
$$P(y = 1|x_i) = \sigma(z_i) = \sigma(w^T x_i + b)$$

The decision rule for binary classification in Logistic Regression is:

If  $\sigma(z_i) \geq 0.5$ , predict  $y_i$  1; otherwise, predict  $y_i$  0.



# Logistic Function



# Logistic Regression

## Odds

- The odds of an event occurring is defined as:

$$\text{Odds} = \frac{p}{1-p},$$

where  $p$  is the probability of the event.

In logistic regression, the probability  $p_i$  is modeled as:

$$p_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}, \quad z_i = \mathbf{w}^\top \mathbf{x}_i + b.$$

$$\text{Odds} = e^{z_i}$$

The log of the odds, known as the logit, is a linear function of the predictors:

$$\log\left(\frac{p_i}{1 - p_i}\right) = z_i = \mathbf{w}^\top \mathbf{x}_i + b.$$



# Logistic Regression

## Odds Ratio

- The odds ratio represents the multiplicative change in odds for a one-unit increase in one predictor variable  $x_j$ , keeping all other predictor variables unchanged . It is calculated as:

$$\text{Odds Ratio (OR)} = e^{w_j},$$

where  $w_j$  is the coefficient of  $x_j$  in the logistic regression model.

- $OR > 1$ : A one-unit increase in  $x_j$  increases the odds of the outcome being 1.
- $OR < 1$ : A one-unit increase in  $x_j$  decreases the odds of the outcome being 1.
- $OR = 1$ : A one-unit increase in  $x_j$  has no effect on the odds of the outcome being 1.



# Parameter Estimation : Logistic Regression Model

## Log-Likelihood Estimation

- Parameter Estimation involves finding the weight vector  $w = [w_1, w_2 \dots w_n]^T$  using a part of data  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ .
- The likelihood of the observed data is:

$$L(\mathbf{w}, b) = \prod_{i=1}^n P(y_i | \mathbf{x}_i) = \prod_{i=1}^n \sigma(z_i)^{y_i} (1 - \sigma(z_i))^{1-y_i}.$$

- For a training sample  $x_i$  whose label  $y_i$  is 1, the probability  $\sigma(w^T \cdot x_i + b)$  should be close to 1 and for a training sample  $x_i$  whose label  $y_i$  is 0, the probability  $\sigma(w^T \cdot x_i + b)$  should be close to 0.
- The log-likelihood function is given by:

$$\ell(\mathbf{w}, b) = \sum_{i=1}^n [y_i \log \sigma(z_i) + (1 - y_i) \log(1 - \sigma(z_i))]$$



## Parameter Estimation : Logistic Regression Model

- The goal is to maximize the log-likelihood  $\ell(\mathbf{w}, b)$  with respect to  $\mathbf{w}$  and  $b$ , which is equivalent to minimizing the negative log-likelihood:

$$\text{Min}_{\mathbf{w}, b}[-\ell(\mathbf{w}, b)]$$

- Gradient Descent** is commonly used to optimize  $\mathcal{L}$ . The gradients are computed as:

$$\nabla_{\mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^n (\sigma(z_i) - y_i) \mathbf{x}_i,$$

$$\nabla_b = \frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^n (\sigma(z_i) - y_i).$$



# Parameter Estimation : Logistic Regression Model

## Gradient Descent Update Rule

- Initialize the weights  $\mathbf{w}$  and bias  $b$  (e.g., to zeros or small random values).
- For each iteration:
  - 1 Compute predictions:  $\sigma(z_i) = \frac{1}{1+e^{-z_i}}$ .
  - 2 Calculate the gradients:

$$\nabla_{\mathbf{w}} = \sum_{i=1}^n (\sigma(z_i) - y_i) \mathbf{x}_i, \quad \nabla_b = \sum_{i=1}^n (\sigma(z_i) - y_i).$$

- 3 Update the weights and bias:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}}, \quad b \leftarrow b - \eta \nabla_b,$$

where  $\eta > 0$  is the **learning rate**.



## Parameter Estimation : Logistic Regression Model

### Gradient Descent Update Rule

- **Learning Rate ( $\eta$ )**: Determines the step size in gradient descent. A small  $\eta$  leads to slower convergence, while a large  $\eta$  can cause divergence.
- **Stopping Criteria**: The algorithm stops when:
  - The change in the loss function between iterations is below a threshold, as decided by the user.
  - A maximum number of iterations, as decided by the user, is reached.



## Logistic Regression in Practice

- Logistic regression assumes a **linear relationship** between the independent variables (features) and the log-odds of the outcome.
- It may not perform well if the actual relationship between variables is non-linear unless **feature transformations** are applied.
- While LGR can be extended to multi-class problems (e.g., multinomial logistic regression), other models like decision trees or neural networks are often more effective for complex tasks.
- If **multi-collinearity** exists (high correlation between predictors), the model may produce unreliable estimates for coefficients.
- Logistic regression can be sensitive to **outliers** in the data, as extreme values can disproportionately affect the model's parameters.
- With a large number of predictors compared to the number of observations, logistic regression is prone to overfitting unless **regularization techniques** (e.g.,  $\ell_1$  or  $\ell_2$ ) are applied.



# Logistic Regression

Why Logistic Regression is a Linear Classifier

- Logistic regression models the log-odds ( $\log \frac{p_i}{1-p_i}$ ) as a linear combination of the input features  $x_i$ :

$$\log \frac{p_i}{1-p_i} = \mathbf{w}^\top \mathbf{x}_i + b,$$

where:

- $p_i$  is the probability of the outcome being 1 ,
- $\mathbf{w}$  is the weight vector,
- $\mathbf{x}_i$  is the feature vector, and
- $b$  is the bias term.

The decision boundary for classification is determined when  $p = 0.5$ , which corresponds to:

$$\mathbf{w}^\top \mathbf{x}_i + b = 0.$$

This equation represents a hyperplane (a straight line in 2D, a plane in 3D, etc.), which divides the feature space into two regions.



# Logistic Regression

## Performance Metrics

- The **accuracy** of a logistic regression model is the proportion of correctly classified instances (both positive and negative) out of the total instances. It is defined as:

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

- A **confusion matrix** is a table used to describe the performance of a classification model. For binary classification, the confusion matrix is a  $2 \times 2$  matrix.

	Predicted 0	Predicted 1
Actual 0	TN (True Negative)	FP (False Positive)
Actual 1	FN (False Negative)	TP (True Positive)

- TN:** True Negatives (correctly predicted as class 0),
- FP:** False Positives (incorrectly predicted as class 1),
- FN:** False Negatives (incorrectly predicted as class 0),
- TP:** True Positives (correctly predicted as class 1).



# Logistic Regression

## Performance Metrics

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

However, accuracy can be misleading when the classes are **imbalanced** (e.g., if one class significantly outnumbers the other). In such cases, other evaluation metrics like precision, recall, and F1-score are often more informative.

- **Precision:** The proportion of true positive predictions among all positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

- **Recall (Sensitivity) or True Positive Rate (TPR):** The proportion of actual positives correctly identified by the model. High sensitivity means fewer false negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$



# Logistic Regression

## Performance Metrics

- **F1-Score:** The harmonic mean of precision and recall. It balances the trade-off between precision and recall, especially in imbalanced datasets.

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

- **Specificity or True Negative Rate (TNR)** measures how well the classifier identifies negative instances:

$$\text{Specificity} = \text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

High specificity means fewer false positives.



# Logistic Regression

## Performance Metrics

- **False Positive Rate (FPR)** is the proportion of actual negatives that are incorrectly classified as positive:

$$FPR = \frac{FP}{FP + TN} = 1 - \text{Specificity}$$

- Summary of Performance Metrics other than accuracy.

Metric	Formula
<b>Sensitivity (TPR) (Recall)</b>	$\frac{TP}{TP+FN}$
<b>False Positive Rate (FPR)</b>	$\frac{FP}{TN+FP}$
<b>Specificity (TNR) (1-FPR)</b>	$\frac{TN}{TN+FP}$
<b>F1-score</b>	$HM(\text{Precision}, \text{Recall})$
<b>Precision</b>	$\frac{TP}{TP+FP}$



# Logistic Regression

## Receiver Operating Characteristic ROC Curve

- The Receiver Operating Characteristic (ROC) curve is a graphical representation of a binary classifier's performance across different threshold values.
- ROC plots the True Positive Rate (TPR) (also known as Sensitivity) in Y-axis against the False Positive Rate (FPR) in X-axis at various threshold settings.
  - The diagonal line of RUC represents a classifier that randomly guesses labels, thereby producing a curve along the diagonal line (from (0, 0) to (1, 1)).
  - A perfect classifier would have a point at (0, 1), meaning an FPR of 0 and a TPR of 1, corresponding to 100% sensitivity and specificity.
  - The area under the ROC curve (AUC-ROC) is a single scalar value summarizing the classifier's performance. An AUC of:
    - 1.0 indicates perfect classifier
    - .5 indicates random classifier
    - Value closer to 1.0 indicates greater performance of the classifier.

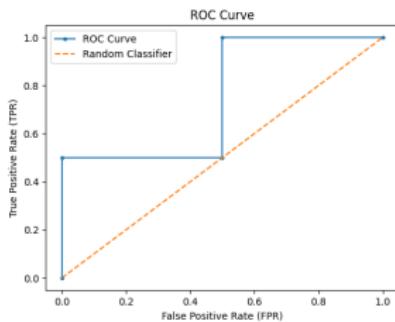


## ROC curve - Single Classifier vs Random Classifier

$y_{true} = [0, 0, 1, 1]$ , threshold = [inf, 0.8 , 0.4 , 0.35, 0.1 ],  $y_{scores} = [0.1, 0.4, 0.35, 0.8]$ .

$y_{scores}$  represents the probability estimates of the positive class.

$$AUC = .75$$



ROC Curve of Binary Classifier

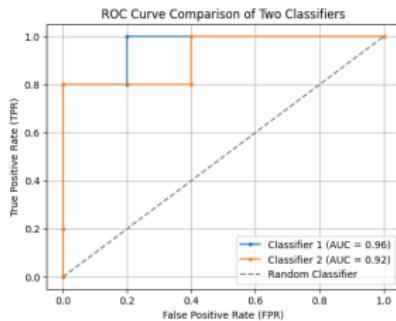


## ROC curve - Two Classifiers vs Random Classifier

$y_{true} = [0, 0, 1, 1, 0, 1, 0, 1, 1, 0]$ ,  $threshold_1 = [\inf 0.9 0.75 0.4 0.35 0.05]$ ,  
 $threshold_2 = [\inf 0.9 0.7 0.3 0.25 0.05]$ ,  $y_{score1} = [0.1, 0.4, 0.35, 0.8, 0.2, 0.75, 0.3, 0.9, 0.85, 0.05]$ ,  $y_{score2} = [0.05, 0.3, 0.25, 0.9, 0.15, 0.7, 0.4, 0.85, 0.8, 0.1]$ .

$y_{scores}$  represents the probability estimates of the positive class.

$AUC1 = .92, AUC2 = .96$



## ROC Curves of Binary Classifiers



# Logistic Regression

## Hyper-Parameters

### Hyper-Parameters

- **Hyperparameters** are configuration settings used to control the training process of a machine learning model.
- Unlike model parameters (which are learned from the data during training), hyperparameters are specified before training begins and influence the behavior and performance of the model.

Hyperparameter	Logistic Regression
Regularization	penalty: (l1,l2, elasticnet)
Optimization Algorithm	(liblinear, lbfgs, saga, newton-cg, sag)
Class Weight	Weights for imbalanced classes

Hyperparameters of Logistic Regression

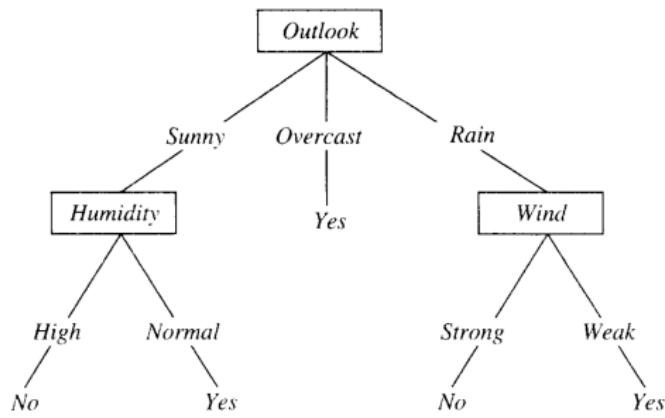


# **Decision Trees**



# Decision Tree

To predict whether to play tennis or not depending on features such as outlook, humidity and wind



Decision Tree [?]



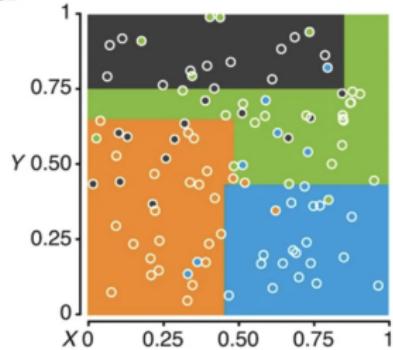
## Decision Tree

- A decision tree is a **supervised** machine learning algorithm used for **classification** and **regression** tasks.
- Decision tree splits the original dataset recursively into subsets using a feature that maximises the reduction in **impurity** of a dataset.
- A decision tree is built using nodes where:
  - Root Node: Represents the entire dataset.
  - Internal Nodes: Represent conditions based on features to split the dataset.
  - Leaf Nodes: Represent the final output (class label or regression value).
- A decision tree is a non-linear classifier unlike a logistic regression model, that is a linear classifier.
- The importance of each feature is computed based on how much it reduces the Gini Impurity (classification problems) or how much it decreases the variance (regression problems) across the tree.



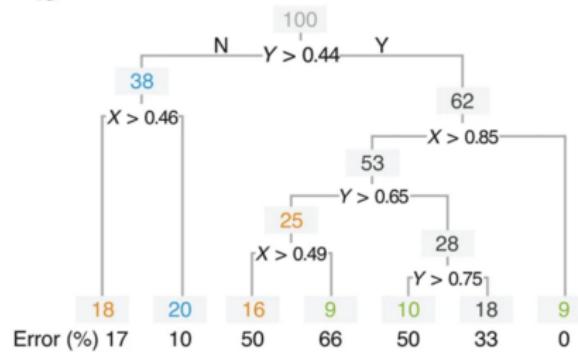
# Decision Tree

**a** Partitioning of two predictor variables



**b**

Decision tree



Decision Tree as a Non-linear classifier [?]



# Decision Tree

## Notations

- $D = \{(x_i, y_i)\}_{i=1}^n$ : Training dataset with  $n$  samples.
- $D_j$  represents the partitions of a dataset D, where j range from 1 to k.
- $x_i$ : Feature vector of the  $i$ -th sample.
- $y_i$ : Target value of the  $i$ -th sample (class label or regression value).
- $f(x)$ : A feature used for splitting.
- $T$ : The decision tree.
- $N$ : A node in the tree.
- $\mathcal{A}$ : Set of all features available for splitting.
- $|D_j|$  represents the number of elements in Subset  $D_j$ , j range from 1 to k.
- $|D|$  represents the number of elements in Dataset D.
- $k$  is the number of classes.



# Decision Tree

## Concept of Entropy

Entropy is a measure of impurity or non-homogeneity.



# Decision Tree

## Entropy & Information Gain for Classification Tasks

- **Entropy** of a dataset  $D$  is a measure of randomness or non-homogeneity.

$$H(D) = - \sum_{i=1}^k p_i \log_2(p_i),$$

where  $p_i$  is the proportion of samples in class  $i$ .

- The entropy ranges from 0 to 1. When a data set is heterogeneous, the entropy is close to 1. When a data set is homogeneous, the entropy is close to 0.
- **Information Gain** from splitting on feature  $f$  with threshold  $t$ :

$$IG(D, f, t) = H(D) - \sum_{j=1}^k \frac{|D_j|}{|D|} H(D_j),$$

where  $D_j$ : Subsets created by the split  $f(x) \leq t$  and  $f(x) > t$ .



# Decision Tree

## Gini Impurity for Classification Tasks

- **Gini Impurity** of a dataset  $D$ :

$$G(D) = 1 - \sum_{i=1}^k p_i^2.$$

- When Gini Impurity is close to 0, data is homogenous and when Gini Impurity is higher than zero, data is heterogeneous.
- The split minimizes:

$$G_{\text{split}}(D, f, t) = \sum_{j=1}^k \frac{|D_j|}{|D|} G(D_j).$$



# Decision Tree

## Variance Reduction for Regression

- **Variance** of the target values in  $D$ :

$$\sigma^2(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - \bar{y})^2,$$

where  $\bar{y}$  is the mean of  $y$  in  $D$ <sup>12</sup>.

- **Variance Reduction** for a split:

$$\Delta\sigma^2(D, f, t) = \sigma^2(D) - \sum_{j=1}^k \frac{|D_j|}{|D|} \sigma^2(D_j).$$

---

<sup>12</sup>MSE heavily penalizes large errors, so the presence of outliers in the data can disproportionately influence the model's performance



# Decision Tree

## Recursive Partitioning

- **Objective:** At each node  $N$ , find the feature  $f^*$  and threshold  $t^*$  that optimize the chosen splitting criterion:

$$(f^*, t^*) = \arg \max_{f \in \mathcal{A}, t} \text{Criterion}(D, f, t),$$

where Criterion could be  $IG(D, f, t)$ ,  $G(D, f, t)$ , or  $\Delta\sigma^2(D, f, t)$ .

- **Partitioning:** Split  $D$  into subsets  $D_1, D_2, \dots, D_k$  based on  $f^*$  and  $t^*$ :

$$D_j = \{x \in D : \text{Condition on } f^* \text{ at } t^*\}.$$

- Repeat the process recursively for each subset  $D_j$  until a stopping criterion is met.



# Decision Tree

## Stopping Criteria

The recursion stops when one of the following conditions is met:

- 1 Purity:** All samples in  $D(N)$  belong to the same class (classification) or the variance is below a threshold (regression).
- 2 Maximum Depth:** The tree reaches a predefined maximum depth `max_depth`.
- 3 Minimum Samples:** The number of samples in  $D(N)$  is below a threshold.
- 4 Pruning:** Use pruning techniques to remove branches.



# Decision Tree

## Prediction

- Decision Tree for Classification : Traverse the tree from the root to a leaf node based on the splitting conditions. At the leaf node:

$$\hat{y} = \arg \max_i p_i,$$

where  $p_i$  is the proportion of samples in class  $i$ .

- Decision Tree for Regression : At the leaf node, predict the mean of the target values:

$$\hat{y} = \frac{1}{|D|} \sum_{i=1}^{|D|} y_i$$



# Decision Tree

## Feature Importance

- Total Feature Importance : The total importance of a feature is the sum of the impurity reductions across all nodes where that feature is used for splitting. The total importance for a feature  $f$  is given by:

$$\text{Importance}(f) = \sum_{\text{nodes where } f \text{ is used}} \text{Impurity Reduction}_f$$

- Normalization of Feature Importance - To normalize the feature importance values, we divide each feature's importance by the sum of all feature importances:

$$\text{Normalized Importance}(f) = \frac{\text{Importance}(f)}{\sum_{f'} \text{Importance}(f')}$$

where the sum in the denominator is over all features.



# Decision Tree

## Performance Metrics

- $R^2$  (R-squared) measures the proportion of variance in the target variable that is explained by the model and informs how well the model fits the data.

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

Where:

- $N$  is the number of samples,
- $y_i$  is the actual value of the target variable for the  $i$ -th sample,
- $\hat{y}_i$  is the predicted value for the  $i$ -th sample,
- $\bar{y}$  is the mean of the actual target values ( $y_i$ ) across all the samples.
- The numerator  $\sum(y_i - \hat{y}_i)^2$  is the *residual sum of squares* (RSS), which measures the unexplained variation or error in the model.
- The denominator  $\sum(y_i - \bar{y})^2$  is the *total sum of squares* (TSS), which measures the total variance of the target variable in the dataset.



# Decision Tree

## Performance Metrics

- $R^2 = 1$ : The model explains 100% of the variance in the target variable. The model's predictions match the actual values perfectly.
- $R^2 = 0$ : The model does not explain any of the variance in the target variable. The model is no better than simply predicting the mean value for every observation.
- $R^2 < 0$ : The model performs worse than simply predicting the mean of the target variable. This could happen if the model is overfitting the training data, and its predictions are worse than a baseline model (the mean value).



# Decision Tree

## Performance Metrics

- In regression decision trees,  $R^2$  can be used both during model evaluation and model selection:
  - **Model Evaluation:** After training the tree,  $R^2$  can be computed on a test set to assess how well the model generalizes to new data. A higher  $R^2$  indicates a better fit of the model to the data.
  - **Model Selection:** While building a decision tree, you might use splits that maximize  $R^2$  at each node. However, decision trees typically focus on minimizing *Mean Squared Error (MSE)* during training. But once the tree is trained,  $R^2$  can provide an additional measure of goodness-of-fit.
- **Overfitting Warning:** A decision tree with a high  $R^2$  on training data but low  $R^2$  on test data is a sign of overfitting. Pruning or limiting tree depth can help address this issue.



# Decision Tree

## Hyper-Parameters

Hyperparameter	Decision Tree
<b>Split Criterion</b>	Metric to measure split quality (gini, entropy, log.loss)
<b>Maximum Depth</b>	Maximum depth of the tree to control overfitting
<b>Minimum Samples to Split</b>	Minimum samples required to split an internal node
<b>Minimum Samples per Leaf</b>	Minimum samples required in a leaf node
<b>Maximum Features</b>	Number of features to consider for the best split
<b>Class Weight</b>	Weights for imbalanced classes
<b>Splitter Strategy</b>	Split strategy (best, random)
<b>Maximum Leaf Nodes</b>	Maximum number of leaf nodes to control tree complexity

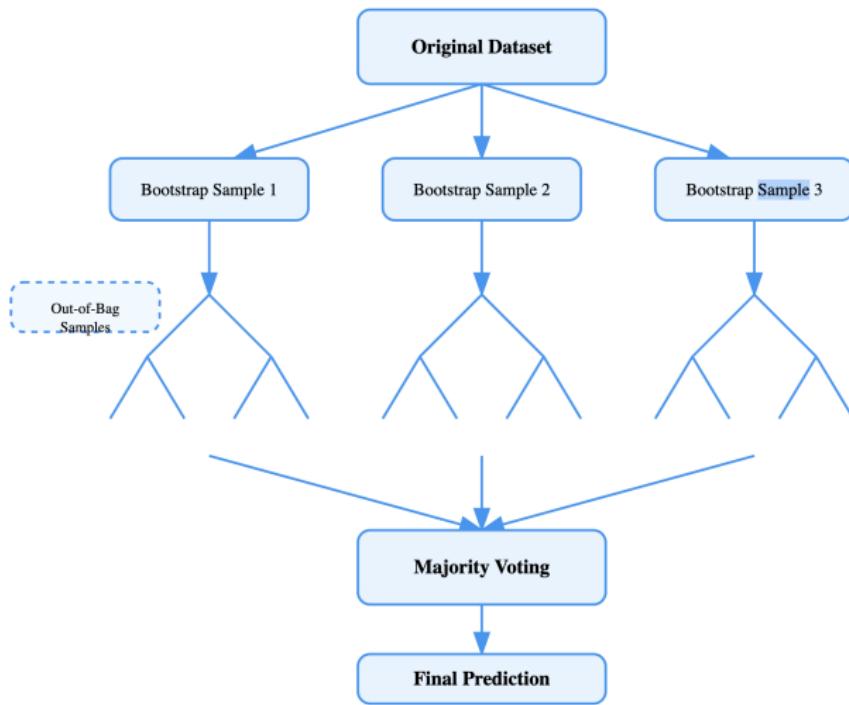
## Hyperparameters of Decision Tree



# **Random Forest**



# Random Forest



Random Forest



## Random Forest

Random Forest is an ensemble learning algorithm that operates by constructing multiple decision trees during training and outputs the mode (classification) or mean (regression) of the predictions made by each individual tree., using random subsets of data and features.

- It is widely used due to its robustness, ability to handle overfitting, and high accuracy in predictive tasks.
- Random Forest is based on the principle of ensemble learning, where the idea is to combine multiple models (in this case, decision trees) to improve the overall performance.
- Each tree contributes to the final decision, and by aggregating these decisions, Random Forest aims to reduce **variance** and **bias**.



# Random Forest

## Random Forest and Bootstrapping

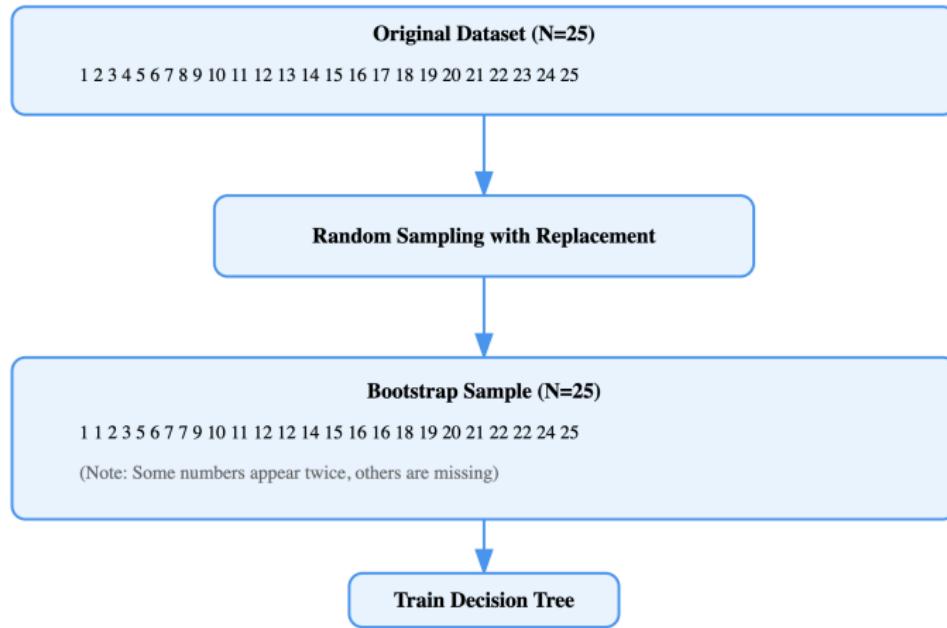
The process of training a Random Forest involves the following steps:

- Bootstrap Sampling - For each tree in the forest, a random subset of the training data is selected using *bootstrap sampling*, where a sample of  $N$  data points is drawn randomly with replacement from the original dataset.
  - Let the original dataset be  $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , where  $x_i$  are the ' $m$ ' dimensional features and  $y_i$  are the corresponding labels.
  - A bootstrap sample  $\mathcal{D}_b$  is drawn by selecting  $N = n$  samples with replacement from  $\mathcal{D}$ .
  - This means some data points may appear multiple times in  $\mathcal{D}_b$ , and some may be left out.
  - Since sampling is done with replacement, approximately 63% of the original data points are included in the bootstrap sample for a given tree.
  - The remaining 37% of the data points, which are not included in the bootstrap sample, are called Out-of-Bag (OOB) samples for that tree.



# Random Forest

Decision Tree with Bootstrapped Samples



Decision Tree Training using Bootstrapped Samples



# Random Forest

## Random Forest and Bootstrap Aggregation

■ **Bagging or Bootstrap Aggregating** : Bootstrap samples are used to train different decision trees and their results are aggregated to make predictions.

- For each bootstrap sample, create a classifier i.e. decision tree using the bootstrapped sample as training data.
- For training  $T$  independent decision trees trained, we have  $T$  bootstrap samples.
- For each decision tree, there will be out of bag samples and bootstrapped samples.



# Random Forests

## Random Forest Training

- **Building Decision Tree** - For each bootstrap sample, a *decision tree* is trained.
  - During the construction of each tree, at each node, a **random** subset of features  $x_j$ , where  $j \subseteq \{1, 2, 3..m\}$  is selected to find the best split, rather than using all features (which is done in traditional decision trees).
  - This random feature selection is the key to ensuring that the trees are diverse and independent of each other.
  - Let  $m$  be the total number of features, and at each node,  $m_r$  features are randomly chosen, where  $m_r \ll m$ .
  - The split criterion (e.g., Gini impurity or Entropy for classification, or variance reduction for regression) is applied to the randomly chosen features.
- **Growing Trees** - Each tree is grown to its full size (without pruning) in the training process.
  - This allows the model to capture complex patterns in the data.
  - Trees are built independently and will differ due to the random sampling of data and features.



# Random Forests

## Random Forest and Prediction

- **Prediction with Random Forest** - Once the forest of trees is built, predictions are made by aggregating the predictions from each tree.
  - In **classification** problems, each tree in the forest makes a prediction, which is a class label. The final prediction is made by *voting*:

$$\hat{y} = \text{mode}(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_T)$$

where  $\hat{y}_i$  is the predicted class by tree  $i$  and  $T$  is the number of trees in the forest. The class that receives the most votes is selected as the final prediction  $\hat{y}$ .

- In **regression** problems, each tree predicts a continuous value,  $\hat{y}_i$ . The final prediction is the *mean* of the predictions from all the trees:

$$\hat{y} = \frac{1}{T} \sum_{i=1}^T \hat{y}_i$$

where  $\hat{y}_i$  is the predicted value from tree  $i$  and  $\hat{y}$  is the final prediction.



## Out of Bag Error

An important feature of Random Forest is the ability to estimate the model's performance without needing a separate validation set. This is done using *out-of-bag* samples.

- Since the bootstrap sample for each tree is drawn with replacement, few data points are not exposed to each tree during the training phase.
- These are called *out-of-bag* samples.
- These OOB samples are used to test the performance of the model as the trees are being trained, providing an internal estimate of model error (out-of-bag error).
- The Random Forest Classifier is trained using bootstrap aggregation where each new tree is fitted from a bootstrap sample of the training observations  $\mathcal{D}_b$ .



# Out of Bag Error

## OOB Error - Classification

- If a particular sample  $(x_i, y_i) \in \mathcal{D} - \mathcal{D}_b$  is not seen by  $N_{oob}(x_i, y_i)$  number of trees, then the out of bag error represents the error made by the  $N_{oob}$  trees in predicting the result  $y_i$ , where  $i$  ranges from 1 to  $T$  trees.
- $OOB_{error}$  is the average error between the decision tree output  $\hat{y}_i$  obtained for the bootstrapped sample  $(x_i, y_i)$  and the real output  $y_i$ .
- The OOB error estimate for classification is given by:

$$OOB\_error(x_i, y_i) = \frac{1}{N_{oob}(x_i, y_i)} \sum_{i=1}^{N_{oob}(x_i, y_i)} \mathbb{I}(\hat{y}_i \neq y_i)$$

where  $N_{OOB}(x_i, y_i)$  is the number of trees that have not seen the out of bag sample  $(x_i, y_i)$ ,  $\hat{y}_i$  is the predicted class for the sample  $i$  by decision tree  $i$ , where  $i$  ranges from 1 to  $N_{oob}(x_i, y_i)$ , and  $y_i$  is the actual class as per the training data.



# Out of Bag Error

## OOB Error - Regression

- For regression, the OOB error is computed as the mean squared error (MSE) between the actual and predicted values for the out-of-bag samples.
- The OOB error estimate for regression is given by:

$$\text{OOB\_error}(x_i, y_i) = \frac{1}{N_{oob}(x_i, y_i)} \sum_{i=1}^{N_{oob}(x_i, y_i)} (\hat{y}_i - y_i)^2$$

where  $N_{\text{OOB}}(x_i, y_i)$  is the number of trees that have not seen the out of bag sample  $(x_i, y_i)$ ,  $\hat{y}_i$  is the predicted class for the sample  $i$  by decision tree  $i$ , where  $i$  ranges from 1 to  $N_{oob}(x_i, y_i)$ , and  $y_i$  is the actual class as per the training data.



# Random Forest

## Advantages and Disadvantages

- **Robustness:** By averaging the results from multiple decision trees, Random Forest reduces the variance compared to individual decision trees, making it more robust to overfitting.
- **Handles Non-linearity:** Random Forest can handle non-linear relationships in the data without requiring any assumptions about the functional form.
- **Works with Missing Data:** It can handle missing values by averaging over multiple trees, making it a versatile model.
- **Internal Cross-Validation:** OOB error provides an internal validation measure.



# Random Forest

## Disadvantages

- **Interpretability:** Unlike a single decision tree, a Random Forest model is less interpretable due to the large number of trees.
- **Computational Complexity:** Training many decision trees can be computationally expensive, especially with large datasets.



# Random Forest

## Hyper-Parameters

Hyperparameter	Description
No Estimators	Number of trees in the forest
Max Depth	Maximum depth of each tree
Min Samples Split	Minimum number of samples for splitting an internal node
Min Samples Leaf	Minimum number of samples at a leaf node.
Max Features	Number of sampled features
Bootstrap	Bootstrap Enable
Random State	For repeatable results
Oob Score	Oob score
No jobs	No of processors to use
Warm Start	Reusability of previous trees
Class Weight	Weights of classes

Hyperparameters of Random Forest



# **Support Vector Machine SVM**



- SVM is a supervised learning method.
- It's considered sometimes the best supervised learning method.
- SVM is an optimal marginal classifier as it is based on the concept of margins.
- The problem at hand is to predict the class of the dependent variable  $y^{(i)}$  given the features  $x^i$ .
- Assume  $y^{(i)}$  takes values -1 or 1 or  $y^{(i)} \in \{-1, 1\}$ .
- The logistic classifier with parameters  $\theta$  is given as

$$h(x; \theta) = g(\theta^T \cdot x) \quad (63)$$

$$h(x; \theta_0, \theta_1^T) = g(\theta_0 + \theta_1^T \cdot x) \quad (64)$$

$$h(x; w, b) = g(b + w^T \cdot x) \quad (65)$$

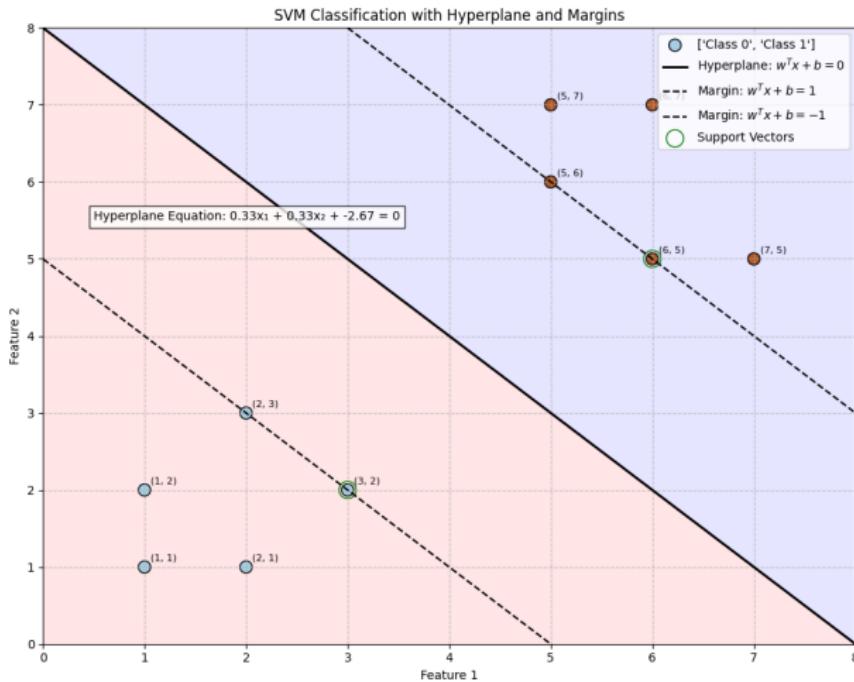
where

$$g(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases} \quad (66)$$



# SVM

## SVM for Classification



## SVM for Classification



## Functional Margin

Functional margin : Given the training sample  $x^{(i)}, y^{(i)}$ , the functional margin of a sample  $x^{(i)}, y^{(i)}$  is given by

$$\gamma_f^{(i)} = y^{(i)} \cdot (w^T \cdot x^{(i)} + b) \quad (67)$$

Given a training set  $S = \{x^{(i)}, y^{(i)}\}$ ,  $i = 1, 2, 3, 4, \dots, m$ , the functional margin of the training set  $S$  is defined by

$$\min_{i=1,2,\dots,m} \gamma_f^{(i)} \quad (68)$$

- A large functional margin  $\gamma_f^{(i)}$  represents a correct and confident prediction.
- If  $y^{(i)} == 1$ , then  $(w^T \cdot x^{(i)} + b)$  has to be very large positive to ensure a large functional margin.
- If  $y^{(i)} == -1$ , then  $(w^T \cdot x^{(i)} + b)$  has to be a large negative number to ensure a large functional margin.
- Essentially, given a training sample  $x^{(i)}, y^{(i)}$ , we can make correct and accurate predictions if functional margin is greater than zero or  $\gamma_f^{(i)} >> 0$



# Geometric Margin

## Geometric Margin

Geometric margin : Given the training sample  $x^{(i)}, y^{(i)}$ , the geometric margin of a training sample  $x^{(i)}, y^{(i)}$  is given by

$$\gamma_g^{(i)} = \frac{y^{(i)} \cdot (w^T \cdot x^{(i)} + b)}{\|w\|} \quad (69)$$

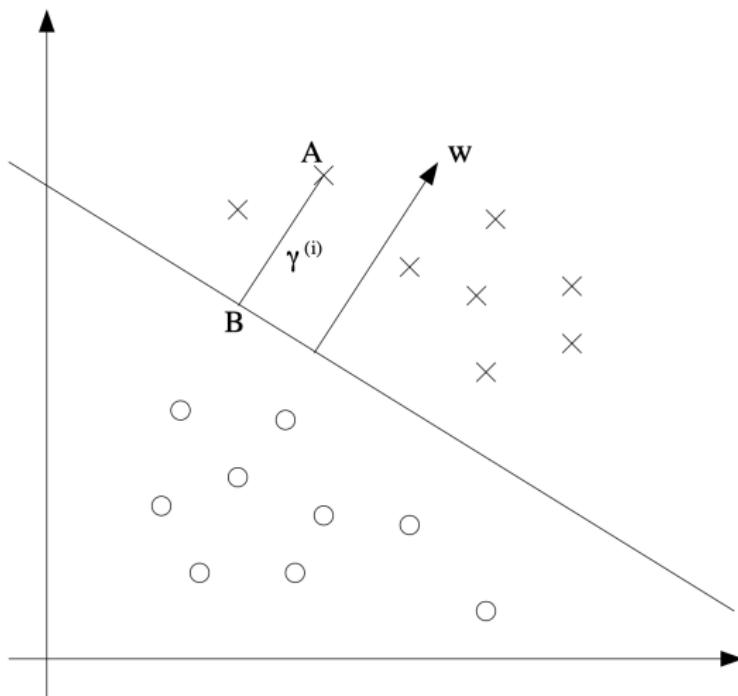
Given a training set  $S = \{x^{(i)}, y^{(i)}\}$ ,  $i = 1, 2, 3, 4, \dots, m$ , the geometric margin of the training set  $S$   $\gamma_{gs}$  is defined by

$$\gamma_{gs} = \min_{i=1,2,\dots,m} \gamma_g^{(i)} \quad (70)$$

- The functional margin could be changed by changing  $w$  and  $b$ , as the function margin  $\gamma_f^{(i)}$  is a function of  $(w^T \cdot x^{(i)} + b)$ .
- Note that the Geometric Margin is the distance of a point  $x^{(i)}$  to a hyperplane in  $n$ -dimensions denoted by  $w^T \cdot x^{(i)} + b = 0$



# Functional Margin Illustration



Functional Margin



# SVM

## SVM - Weight Computation

- SVM is an optimum linear classifier that tries to maximize the geometric margin for a given training data set S defined by  $x^{(i)}, y^{(i)}$ , where  $i \in [1, n]$ .
- The reason for maximizing the geometric margin is that it improves our confidence for making predictions.
- SVM assumes that the data pattern  $x^{(i)}, y^{(i)}$  has to be linearly separable.
- The objective function and constraints for the Optimum Margin Classifier, the SVM is

$$\max_{w,b} \gamma_g = \max_{w,b} \min_{i=1,2,\dots,n} \gamma_g^{(i)} \quad (71)$$

such that  $\frac{y^{(i)}.(w^T.x^{(i)}+b)}{\|w\|} \geq \gamma_g$  and  $\|w\| = 1$ .

- This optimization problem cannot be solved by plugging the equations and constraints into a solver, as the constraint  $\|w\| = 1$  is a non-convex constraint.



# SVM

## SVM - Weight Computation

- The above optimization problem is slightly transformed to the following.

$$\max_{w,b} \frac{\gamma_f}{\|w\|} \quad (72)$$

such that  $\frac{y^{(i)} \cdot (w^T \cdot x^{(i)} + b)}{\|w\|} \geq \frac{\gamma_f}{\|w\|}$ . Although non-convexity in constraints is solved, we have a non-convex objective function.

- Set functional margin  $\gamma_f = 1$ , the optimization function becomes,

$$\max_{w,b} \frac{1}{\|w\|} \quad (73)$$

such that  $y^{(i)} \cdot (w^T \cdot x^{(i)} + b) \geq 1$ .

- The max optimization function can be represented as a min optimization function.

$$\begin{aligned} & \min_{w,b} \|w\|^2 \\ \text{s.t. } & y^{(i)} \cdot (w^T \cdot x^{(i)} + b) \geq 1 \end{aligned} \quad (74)$$

This is a convex quadratic objective function with linear constraints that can be solved using a quadratic programming code.



- To solve this constrained optimization problem, we introduce \*\*Lagrange multipliers\*\*  $\alpha_i$  (one for each data point) and convert the problem into its \*\*dual form\*\*:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i^T x_j) \quad (75)$$

subject to:

$$\sum_{i=1}^n \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad \forall i. \quad (76)$$

- $\alpha_i$  represents the influence of the  $i$ -th training point.
- The solution to SVM is determined entirely by the support vectors, which are the points for which  $\alpha_i > 0$ .
  - $\alpha_i = 0$ : The point is correctly classified and far from the margin. It does not contribute to the decision boundary.
  - $\alpha_i > 0$ : The point lies exactly on the margin. These points are called \*\*support vectors\*\* and they define the margin.
  - Since it's a hard-margin SVM, no points can violate the margin, so there are no  $\alpha_i$  values between 0 and  $C$  (as in soft-margin SVM).



## SVM Weights

The SVM optimum weight vector  $w_*$  depends on all the training samples where the scaling constants  $\alpha_i$  (lagrange multipliers) are greater than zero.

$$w_* = \sum_{i=1}^n \alpha_i \cdot y^{(i)} \cdot x^{(i)} \quad (77)$$

- The scaling constants  $\alpha_i$  are non-zero only on the support vectors that are the training points that lie on the boundary of the margin
- Intercept of SVM

$$b_* = - \left[ \frac{\max_{y^{(i)}=-1} w_*^T \cdot x^{(i)}}{2} + \frac{\max_{y^{(i)}=1} w_*^T \cdot x^{(i)}}{2} \right] \quad (78)$$



## SVM Prediction for testing data point z

$$y_{pred} = g(w_*^T \cdot z + b_*) \quad (79)$$

$$y_{pred} = g\left(\sum_{i=1}^n \alpha_i \cdot y^{(i)} \cdot x^{(i)}^T \cdot z + b_*\right) \quad (80)$$

$$y_{pred} = g\left(\sum_{i=1}^n (\alpha_i \cdot y^{(i)} \cdot \langle x^{(i)}, z \rangle) + b_*\right) \quad (81)$$

- We observe that even if we have a large training data set or m is large, the solution depends only on the support vectors and hence is computationally efficient.
- The term  $\langle x^{(i)}, z \rangle$  is called the inner product of the training sample i and new sample for which prediction has to be made, where the  $x^{(i)}$  are effectively support vectors.



# SVM

## Hard Margin SVM

- In hard-margin SVM, we require that each training point is correctly classified and at least a unit distance from the decision boundary  $(w^T \cdot x^{(i)} + b) = 0$ . This constraint is written as

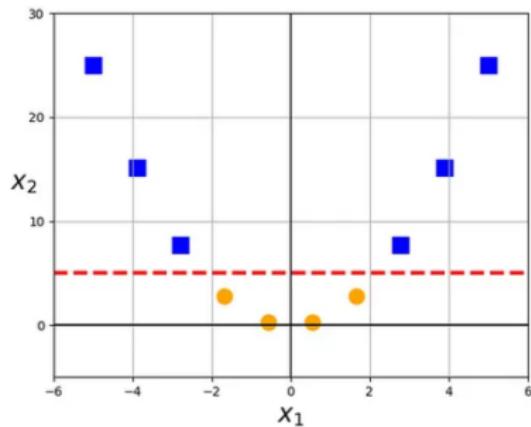
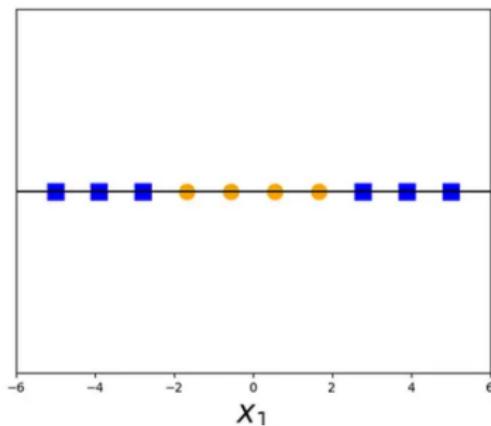
$$y^{(i)} \cdot (w^T \cdot x^{(i)} + b) \geq 1 \quad \forall i \quad (82)$$

- If  $y_i = 1$  is correctly classified,  $(w^T \cdot x^{(i)} + b) > 1$ .
- If  $y_i = -1$  is correctly classified,  $(w^T \cdot x^{(i)} + b) < -1$ .
- The margin itself is determined by the closest points satisfying the equality,  $(w^T \cdot x_s^{(i)} + b) = 1$  which defines the support vectors, where  $x_s^{(i)}$  are called the support vectors.
- For a SVM with hard margin, no data points would be located within the margin.
- Since the SVM optimization function  $\|w\|^2$  is convex, it will always converge to a global minimum.
- Hard Margin SVM assumes that all real world data is linearly separable or there exists a hyperplane that can separate two classes of points without any mis-classification → Unrealistic → Kernel, Soft Margin SVM.



# Kernel Transformation $\phi(x) = x^2$

## Kernel Functions for Non-linear Data



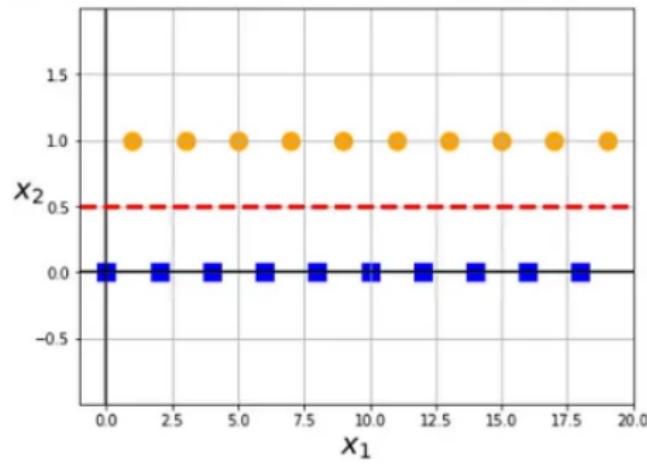
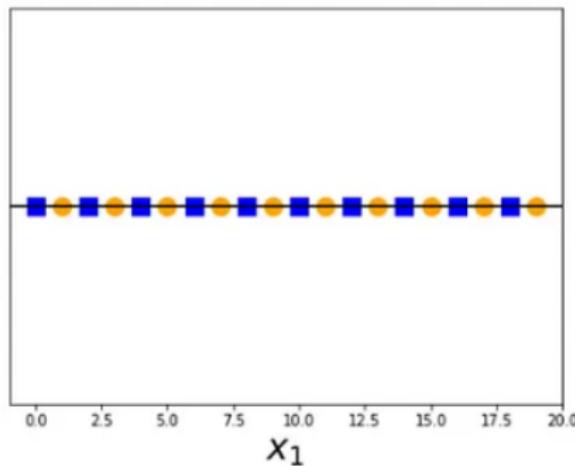
Low Dimension to High Dimension - Kernel Transformation

Data becomes linearly separable after a Kernel Transformation is applied.



# Kernel Transformation $\phi(x) = x.mod(2)$

## Kernel Functions for Non-linear Data



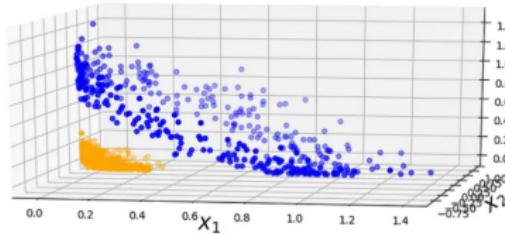
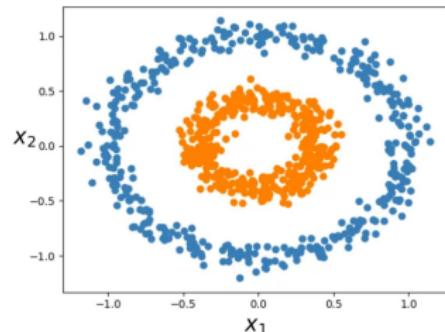
Low Dimension to High Dimension - Kernel Transformation



# Kernel Transformation

## Kernel Functions for Non-linear Data

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$



Low Dimension to High Dimension - Kernel Transformation



# Kernel Function

## Kernel Functions for Non-linear Data

Kernel Function  $K(x^{(i)}, z)$

$$K(x^{(i)}, z) = \phi(x^{(i)})^T \cdot \phi(z)$$

$$y_{pred} = g\left(\sum_{i=1}^m \alpha_i \cdot y^{(i)} \cdot \phi(x^{(i)})^T \cdot \phi(z) + b_*\right) \quad (83)$$

- For computing the predictions for data that is non-linearly separable, kernel functions are applied on the support vectors  $x^{(i)}$ ,  $\alpha_i \neq 0$  and the testing data point  $z$ .
- The Kernel function  $K(x, z)$  is easy to compute by taking the product of  $\phi(x^{(i)})$  and  $\phi(z)$ .
- However, it is difficult to compute the Kernel function in this way, when the dimension of  $\phi$  is extremely high.
- But using "Kernel Trick", we can compute the kernel function without explicitly computing  $\phi(x^{(i)})$  and  $\phi(z)$ .



# Kernel Function

## Kernel Functions for Non-linear Data

- Kernel functions are mathematical functions that transform the input data into a higher-dimensional feature space, where the data points may become more linearly separable or have more complex relationships.
- Kernel functions allow SVMs to handle non-linear and complex data without explicitly computing the features, which can be computationally expensive or impossible.



## Soft Margin SVM

- In real-world datasets, perfect separation is rare, and strict enforcement of a margin may lead to infeasibility.
- The soft-margin SVM allows points to violate the margin constraints to some extent, making the problem solvable.
  - Hard-margin SVM (Large C) is highly sensitive to outliers, as a single misclassified point can significantly shift the decision boundary.
  - Soft-margin SVM (Small C) reduces this sensitivity by allowing outliers to lie within or beyond the margin with a penalty.
- The soft-margin formulation introduces a regularization parameter (C), which controls the balance between generalization (maximising SVM margin) and training accuracy (minimize classification errors).
- $C \uparrow \Rightarrow \text{SVM margin} \downarrow \Rightarrow \text{Training Accuracy} \uparrow$ .
- How to use C?
  - Increase C when: You have high confidence in your training data and want to minimize training error.
  - Decrease C when: You suspect noise in training data and want to prevent overfitting.



# C - SVM

Regularization Parameter C

## Soft Margin SVM

The \*\*soft-margin SVM\*\* optimization problem is formulated as:

$$\min_{\mathbf{w}, b, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$$

where the first term  $\frac{1}{2} \|\mathbf{w}\|^2$  maximizes the margin and the second term  $C \sum \xi_i$  penalizes margin violations and misclassified points among the n training samples.

- $\xi_i = 0 \Rightarrow$  The point is **correctly classified** and lies **outside or on the margin**.
- $0 < \xi_i < 1 \Rightarrow$  The point is **within the margin** but **correctly classified**.
- $\xi_i \geq 1 \Rightarrow$  The point is **misclassified** (lies on the wrong side of the decision boundary).



# Lagrangian - SVM

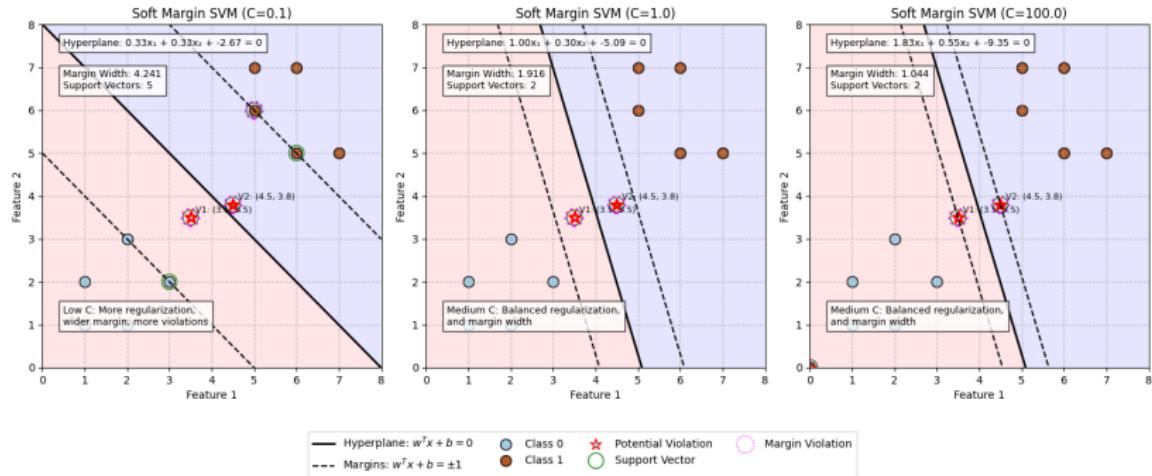
## Lagrangian & C

- Lagrange multipliers  $\alpha_i$  determine which points are support vectors.
- Support vectors inside the margin have  $0 < \alpha_i < C$
- Misclassified points or support vectors on the boundary have  $\alpha_i = C$
- Points outside the margin have  $\alpha_i = 0$  and do not contribute to the model.



# SVM

## SVM Illustration



## C and Margin Width of SVM



# **Neural Networks**



## Neural Networks

A neural network is a computational model inspired by the structure and function of biological neural networks. It consists of layers of interconnected nodes (neurons) that process data through weighted connections and activation functions. Neural networks are used for tasks such as classification, regression, and pattern recognition.

### A neural network consists of three layers:

- **Input Layer:** Receives raw data as input features.
- **Hidden Layers:** Perform transformations using weights and activation functions.
- **Output Layer:** Produces the final prediction.



# Neural Network

## Neural Networks for Handwritten Digit Recognition

### Neural Network for MNIST Digit Recognition

(784-16-16-10 Architecture)



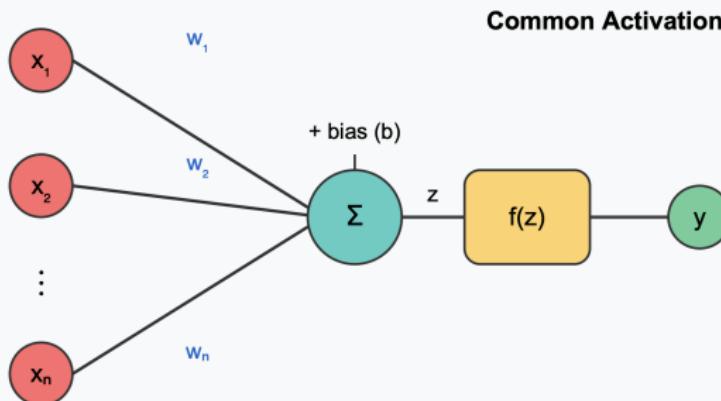
## Neural Network for Handwritten Digit Recognition



# Neural Network

## Structure of a Neuron

### Single Neuron with Multiple Inputs and Activation Function



#### Common Activation Functions

**ReLU:**  $f(z) = \max(0, z)$



**Sigmoid:**  $f(z) = 1/(1+e^{-z})$



**Tanh:**  $f(z) = (e^z - e^{-z})/(e^z + e^{-z})$



#### Summation:

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

#### Output:

$$y = f(z)$$

#### Final Output:

$$y = f(w_1x_1 + w_2x_2 + \dots + w_nx_n + b)$$

Single Neuron with  $n$  inputs and one output



# Neural Network

## Forward Propagation

**Neural networks process data in two steps:**

### 1. Weighted Sum:

$$z = \sum_{i=1}^n w_i x_i + b \quad (84)$$

### 2. Activation Function:

$$y = \sigma(z) \quad (85)$$

where

- $w_i$  refer to the weights of the inputs
- $x_i$  refer to the inputs
- $b$  refer to the bias
- $n$  refers to the number of inputs



### Sigmoid Activation Function

The **Sigmoid** function is commonly used for binary classification problems. It squashes input values into the range  $(0,1)$ , making it useful for probabilistic interpretations.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (86)$$

#### Properties:

- Range of Sigmoid:  $(0,1)$
- Nonlinear and differentiable.
- Can cause **vanishing gradient** problems in deep networks.



### RELU Activation Function

**ReLU** is the most widely used activation function in deep learning. It introduces non-linearity by outputting zero for negative inputs.

$$\sigma(z) = \max(0, z) \quad (87)$$

#### Properties:

- Range of RELU:  $[0, \infty)$
- Computationally efficient.
- Solves the vanishing gradient problem but may suffer from **dying ReLU** (neurons stuck at zero).



### Hyperbolic Tangent

The **Tanh** function is similar to Sigmoid but outputs values in the range (-1,1), centering data around zero.

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (88)$$

### Properties:

- Range of Tanh: (-1,1)
- Zero-centered (better than Sigmoid for deep networks).
- Still suffers from **vanishing gradient** problems.



### Softmax

The Softmax activation function is used to convert raw scores (logits) into probabilities. It ensures that all output values are in the range (0, 1) and the sum of all outputs is 1, making it suitable for multi-class classification.

Given a vector  $\mathbf{z} = [z_1, z_2, \dots, z_n]$ , the Softmax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}, \quad \text{for } i = 1, \dots, n$$

#### Properties:

- Used in the last layer of neural networks.
- Suitable for multi-class classification problems where the input belongs to only one class out of many possible classes. Eg. Handwritten Digit Recognition

Subtracting the maximum of the input vector helps prevent overflow in the exponential function. This is a standard practice for numerical stability:

$$\text{Softmax}(z_i) = \frac{e^{z_i - \max(z)}}{\sum_j e^{z_j - \max(z)}}$$



## Training a Neural Network

Training a neural network refers to the process of adjusting its weights and biases so that it can learn patterns from data and make accurate predictions.

- 1 Forward Propagation:** Input data is propagated through the neural network which uses weights, biases, and activation functions to compute predictions.
- 2 Loss Computation:** The difference between the predicted output and the actual target is measured using a loss function (e.g., Mean Squared Error, Cross-Entropy Loss).
- 3 Backpropagation:** The gradient of the loss function with respect to each weight is computed using the chain rule and is used to update the weights.
- 4 Weight Update:** The gradients are used to update weights using an optimizer like Stochastic Gradient Descent (SGD) or Adam. The learning rate  $\eta$  controls the size of weight updates.
- 5 Iterations until Convergence:** The process is repeated for multiple epochs until the loss function reaches a minimum or the network performance stabilizes.



## Training

- Forward Propagation : Given Input Vector  $x$ , Weight Matrix  $W$ , Bias Vector  $b$ , the output  $\hat{y}$  of the neural network is computed as:

$$\hat{y} = f(Wx + b) \quad (89)$$

- Loss Function Computation : The difference between the predicted output and the actual target is measured using a loss function. For example, in regression, the Mean Squared Error (MSE) loss is given by:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (90)$$

For classification, Cross-Entropy Loss is commonly used:

$$\mathcal{L}(y, \hat{y}) = - \sum_{i=1}^n y_i \log \hat{y}_i \quad (91)$$



## Training

- Gradient Computation : The gradient of the loss function with respect to each weight is computed using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{W}} \quad (92)$$

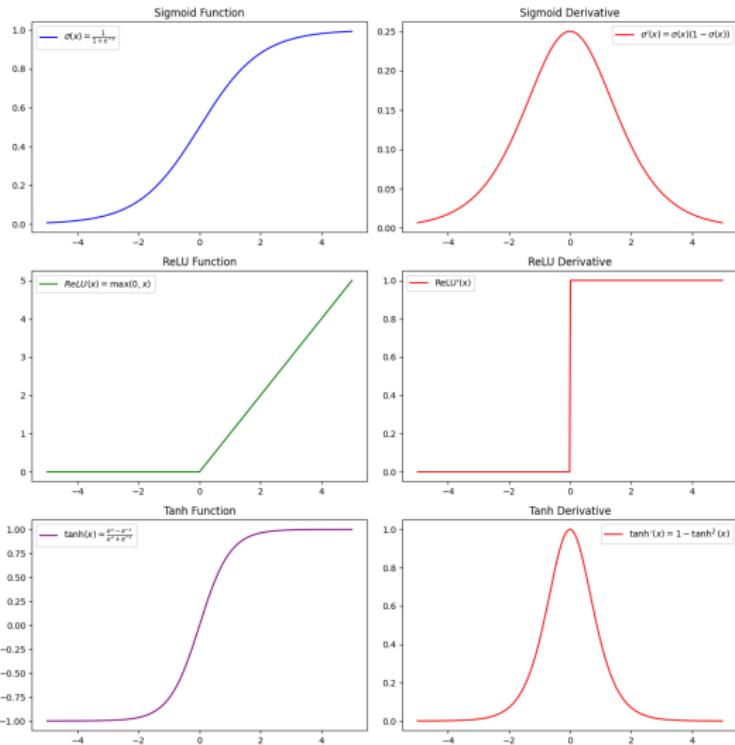
- Weight Updation : Weights are updated using an optimization algorithm such as Stochastic Gradient Descent (SGD):

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \quad (93)$$

where  $\eta$  is the learning rate.



# Activation Functions



Activation Functions - Sigmoid, Relu, Tanh



# Neural Networks

## Types of Neural Networks

### Different architectures for various tasks:

- **Perceptron Model** : Best for classification tasks
- **Feedforward Neural Networks (FNNs)**: Best for prediction tasks.
- **Convolutional Neural Networks (CNNs)**: Best for image processing.
- **Recurrent Neural Networks (RNNs)**: Used for sequential data like text and speech.
- **Transformer Networks**: Used in NLP (e.g., GPT, BERT).



# Neural Network

## Perceptron Model (Frank Rosenblatt)

### Perceptron Model

A **Perceptron** is the simplest type of artificial neural network used for binary classification. It is a single-layer neural network that consists of a **linear combination of inputs followed by an activation function**.

- Given an input vector  $\mathbf{x}$  with  $n$  features:  $\mathbf{x} = [x_1, x_2, \dots, x_n]$
- Each input has an associated weight  $\mathbf{w}$  and a bias term  $b$ :  
$$\mathbf{w} = [w_1, w_2, \dots, w_n]$$
- The output of the perceptron is computed as:  
$$z = \mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^n w_i x_i + b$$
- The activation function used in a perceptron can be **step function**, also called the Heaviside function or threshold function.

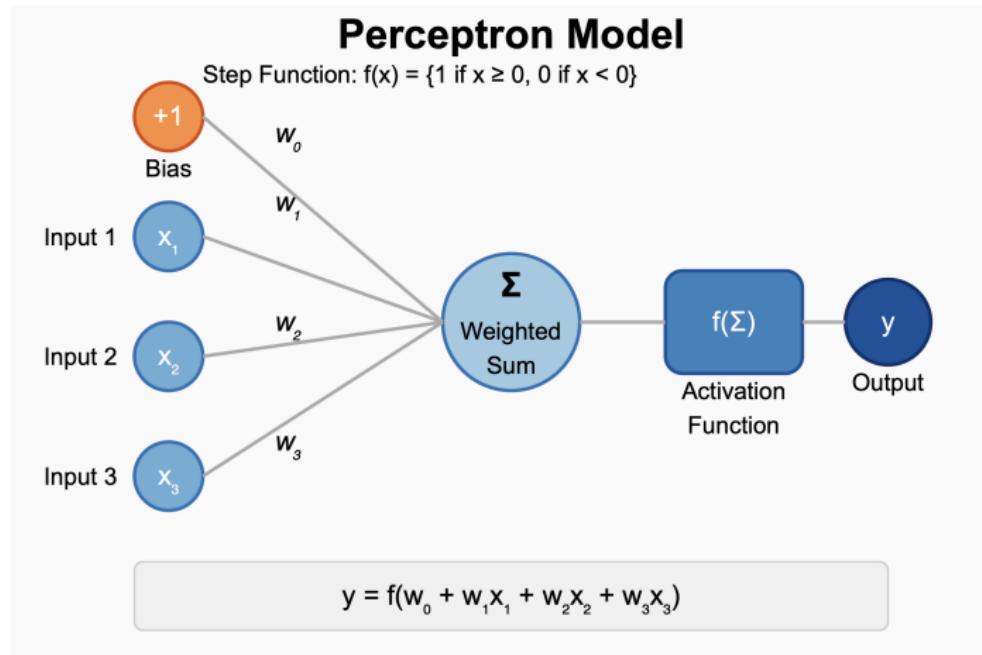
$$y = f(z) = \text{step}(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \quad (94)$$

This function classifies inputs into one of two categories '0' & '1'.



# Neural Network

## Feed-Forward Neural Network - Perceptron Model



3 Input Perceptron Model with Step Function Activation Function



# Neural Network

## Training Perceptron Model

- Only works for **linearly separable** data.
- Cannot handle XOR-type problems.
- However, the Perceptron model has a high degree of explainability compared to more complex models like deep neural networks.
- The Perceptron learns a simple linear decision boundary.
- Each weight  $w_i$  represents the importance of the corresponding feature  $x_i$ .
- The perceptron model converges only if data is linearly separable.
- The perceptron model does not capture the interaction effects of the input variables.
- More complex models (like decision trees or neural networks) capture interactions between features, making them harder to interpret.
- Does not support multi-class classification (requires extensions like **Multi-Layer Perceptron (MLP)**).

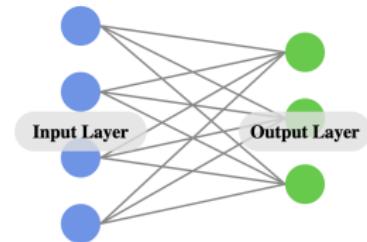


# Neural Network

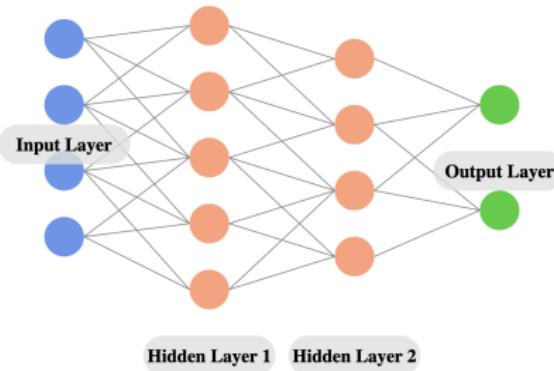
## Single and Multi-layer Perceptron

### Single-Layer vs Multi-Layer Perceptron

Single-Layer Perceptron



Multi-Layer Perceptron



● Input Neurons      ● Hidden Neurons      ● Output Neurons      ——— Connections (Weights)

Single Layer and Multi Layer Perceptron Model



# Neural Network

## Perceptron and Cosine Similarity

- Perceptron output depends on the dot product of the weight vector and the training data  $x$

$$y = f(z) = \text{step}(w_{\text{net}} \cdot x) = \begin{cases} 1, & w_{\text{net}} \cdot x \geq 0 \\ 0, & w_{\text{net}} \cdot x < 0 \end{cases} \quad (95)$$

- **The perceptron decision is based on cosine similarity**, as it checks whether the angle  $\theta$  between  $w$  and  $x$  is **less than or greater than  $90^\circ$** .
- The cosine similarity between two vectors  $w$  and  $x$  is given by:

$$\cos \theta = \frac{w_{\text{net}} \cdot x}{||w_{\text{net}}|| ||x||}$$

In the perceptron model:

- If  $\cos \theta > 0$  (angle  $< 90^\circ$ ), the perceptron predicts  $+1$  (same direction).
- If  $\cos \theta < 0$  (angle  $> 90^\circ$ ), the perceptron predicts  $-1$  (opposite direction).
- If  $\cos \theta = 0$  (angle  $= 90^\circ$ ), the point lies exactly on the decision boundary.
- The best set of weights  $w$  would ensure that Class 1 points would have Cosine Similarity greater than 0 and Class 2 points would have Cosine Similarity lesser than 0.



### Back Propogation

Backpropagation is a key algorithm for training neural networks, allowing them to learn by adjusting weights based on the error. It uses the chain rule of differentiation to propagate the gradient of the loss function backward through the network.

The key steps involved in training a neural network are

- 1 Forward pass: The forward pass is the process where input data is passed through the neural network to compute the output. Each layer processes the input using weights, biases, and activation functions.
- 2 Backward pass: The backward pass calculates the gradient of the loss function with respect to weights and biases using the chain rule of differentiation. This helps adjust the network parameters to minimize the error.
- 3 Weight update: Once gradients are computed, weights and biases are updated using gradient descent:

**Repeat until convergence!**



# Neural Network

## Forward Pass

For a neural network with  $L$  layers, the forward pass refers to the transition from input  $x$  in layer zero to output  $\hat{y} = a^{(L)}$  in the  $L^{th}$  layer.

- $x$  is the input
- $W^{(l)}$  be the weight matrix for layer  $l$
- $b^{(l)}$  be the bias for layer  $l$
- $z^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$  is the weighted sum before activation
- $\mathbf{a}^{(l)} = f(z^{(l)})$  is the activation function output.
- For the first layer  $a^{(0)} = x$ , where  $x$  is the input.
- For the output layer  $L$ , the prediction is  $\hat{y} = \mathbf{a}^{(L)}$ .
- The loss function  $J$  measures how far the predicted  $\hat{y}$  is from the true observed  $y$ . The loss function is given by

$$J = \frac{1}{2} ||\hat{\mathbf{y}} - \mathbf{y}||^2 \quad (96)$$



# Neural Network

## Backward Pass & Gradient Computation

The primary objective is to update the weights using the gradient descent.

$$\mathbf{W}^{(I)} \leftarrow \mathbf{W}^{(I)} - \eta \frac{\partial J}{\partial \mathbf{W}^{(I)}}$$
$$\mathbf{b}^{(I)} \leftarrow \mathbf{b}^{(I)} - \eta \frac{\partial J}{\partial \mathbf{b}^{(I)}}$$

where  $\eta$  is the learning rate.

$$\frac{\partial J}{\partial W^{(L)}} = \delta^{(L)} (a^{(L-1)})^T = \left( \frac{\partial J}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \right) \cdot \left( \frac{\partial z^{(L)}}{\partial W^{(L)}} \right) \quad (97)$$

- Step 1 : The loss function is defined as  $J = \frac{1}{2} ||\hat{\mathbf{y}} - \mathbf{y}||^2$ .

The error at the output layer L is defined as

$$\delta^{(L)} = \frac{\partial J}{\partial z^{(L)}} = \left( \frac{\partial J}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \right) = \frac{\partial J}{\partial a^{(L)}} \odot f'(\mathbf{z}^{(L)})$$

- Step 2 : For a hidden layer I, the error propagates backwards.

$$\delta^{(I)} = (\mathbf{W}^{(I+1)T} \delta^{(I+1)}) \odot f'(\mathbf{z}^{(I)})$$

where  $\odot$  indicates element-wise multiplication.



# Neural Networks

## Gradient Computation & Weight Updation

- Step 3 : Compute weight gradient  $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$  and bias gradient  $\frac{\partial J}{\partial \mathbf{b}^{(l)}}$ .

### Weight Gradient:

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} \mathbf{a}^{(l-1)T}$$

### Bias Gradient:

$$\frac{\partial J}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

- Step 4 : The weights and biases are updated based on the gradient descent rule.

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial J}{\partial \mathbf{W}^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial J}{\partial \mathbf{b}^{(l)}}$$

where  $\eta$  is the learning rate.



## Solving AND gate Problem : Neural Network

The AND gate has two inputs ( $x_1, x_2$ ) and one output ( $y$ ). The truth table is:

$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

We solve this using a two-layer neural network:

- Input Layer: 2 neurons ( $x_1, x_2$ )
- Hidden Layer: 2 neurons with sigmoid activation
- Output Layer: 1 neuron with sigmoid activation



# Solving AND gate Problem : Neural Network

## Initialization

Random initialization of weights and biases:

For input to hidden layer:

$$W^{(1)} = \begin{bmatrix} 0.2 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \quad (98)$$

For hidden to output layer:

$$W^{(2)} = [0.6 \quad 0.8], \quad b^{(2)} = 0.3 \quad (99)$$

Activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \sigma'(z) = \sigma(z)(1 - \sigma(z)) \quad (100)$$

Learning rate:

$$\eta = 0.5 \quad (101)$$



# Solving AND gate Problem : Neural Network

## Forward Pass

For input  $(1, 1)$  with expected output  $y = 1$ :

Hidden layer:

$$z^{(1)} = W^{(1)}x + b^{(1)} = \begin{bmatrix} 0.7 \\ 1.2 \end{bmatrix} \quad (102)$$

Applying sigmoid:

$$a^{(1)} = \sigma(z^{(1)}) = \begin{bmatrix} 0.668 \\ 0.768 \end{bmatrix} \quad (103)$$

Output layer:

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} = 1.315 \quad (104)$$

Applying sigmoid:

$$\hat{y} = a^{(2)} = \sigma(1.315) = 0.788 \quad (105)$$

Error Calculation

$$Error = \hat{y} - y = (a^{(2)} - y) = -0.212 \quad (106)$$

Loss Function

$$J = \frac{(\hat{y} - y)^2}{2} = \frac{(a^{(2)} - y)^2}{2} \quad (107)$$



# Solving AND gate Problem : Neural Network

## Backpropagation

Output layer error:

$$\delta^{(2)} = (a^{(2)} - y) \cdot \sigma'(z^{(2)}) \quad (108)$$

$$\sigma'(1.315) = 0.167 \quad (109)$$

$$\delta^{(2)} = (-0.212) \times 0.167 = -0.0354 \quad (110)$$

Hidden layer error:

$$\delta^{(1)} = (W^{(2)T} \delta^{(2)}) \odot \sigma'(z^{(1)}) \quad (111)$$

$$W^{(2)T} = \begin{bmatrix} .6 \\ .8 \end{bmatrix} \quad (112)$$

$$\sigma'(z^{(1)}) = \sigma'(W^{(1)}x + b^{(1)}) = \begin{bmatrix} \sigma(0.7) \cdot (1 - \sigma(0.7)) \\ \sigma(1.2) \cdot (1 - \sigma(1.2)) \end{bmatrix} = \begin{bmatrix} .221 \\ .177 \end{bmatrix} \quad (113)$$

$$\delta^{(1)} = \begin{bmatrix} -0.0047 \\ -0.0050 \end{bmatrix} \quad (114)$$



# Solving AND gate Problem : Neural Network

## Weight Updates

Updating output layer weights:

$$W^{(2)} = W^{(2)} - \eta \cdot \delta^{(2)} \cdot a^{(1)} \quad (115)$$

$$W^{(2)} = \begin{bmatrix} 0.612 \\ 0.814 \end{bmatrix} \quad (116)$$

Updating hidden layer weights:

$$W^{(1)} = W^{(1)} - \eta \cdot \delta^{(1)} \cdot a^{(0)} = W^{(1)} - \eta \cdot \delta^{(1)} \cdot x \quad (117)$$

$$W^{(1)} = \begin{bmatrix} 0.202 & 0.402 \\ 0.302 & 0.702 \end{bmatrix} \quad (118)$$

Repeating this over multiple epochs<sup>13</sup> will allow the network to converge to the correct function.



---

<sup>13</sup>1 Epoch means that the neural network has seen the entire training data once

## Forward Pass with Updated Weights

Performing forward pass with updated weights:

$$z^{(1)} = W^{(1)}x + b^{(1)} = \begin{bmatrix} 0.704 \\ 1.204 \end{bmatrix} \quad (119)$$

Applying sigmoid:

$$a^{(1)} = \sigma(z^{(1)}) = \begin{bmatrix} 0.669 \\ 0.769 \end{bmatrix} \quad (120)$$

Output layer:

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} = 1.322 \quad (121)$$

Applying sigmoid:

$$a^{(2)} = \sigma(1.322) = 0.789 \quad (122)$$

$$\text{Error} = (a^{(2)} - y) = -0.211 \quad (123)$$

Repeating the backpropagation steps again will further update the weights, reducing the error iteratively. Note that the reduction in error is 0.001.



### Loss Functions

Loss functions measure the difference between the predicted output and the actual target, guiding the optimization process.

- Loss functions are defined based on whether the problem is a prediction problem or a classification problem
- For prediction problem, the preferred loss functions are
  - Mean square error (MSE) - Large errors are penalized heavily compared to small errors, Sensitive to outliers.
  - Mean absolute error (MAE) - Treats all errors equally, Less sensitive to outliers.
  - Log Cosh loss (LCL) - Less sensitive to outliers
- For a classification problem, the preferred loss functions are
  - Binary Cross-Entropy Loss - Binary Cross-Entropy (BCE) is a loss function commonly used in binary classification tasks, where the goal is to predict one of two possible classes.



### ■ Loss Functions for Prediction Problem

$$J_{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \quad (124)$$

$$J_{MAE} = \frac{\sum_{i=1}^n |\hat{\mathbf{y}}_i - \mathbf{y}_i|}{n} \quad (125)$$

- Loss Functions - Classification Problem For a single data point with a true label  $y \in \{0, 1\}$  and predicted probability  $\hat{y}$  (output of a sigmoid activation function), the binary cross-entropy loss is defined as:

$$J_{BCE}^{(i)} = -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (126)$$

For a dataset of  $n$  samples, the total loss is given by:

$$J_{BCE} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (127)$$



# Neural Networks

## Loss Functions

- If the actual label  $y = 1$ , the loss simplifies to:

$$J_{BCE} = -\log(\hat{y}) \quad (128)$$

This means that if  $\hat{y}$  is close to 1, the loss is small; if  $\hat{y}$  is close to 0, the loss is large.

When the true label  $y=1$ , the loss follows  $-\log(\hat{y})$ .

- If the actual label  $y = 0$ , the loss simplifies to:

$$J_{BCE} = -\log(1 - \hat{y}) \quad (129)$$

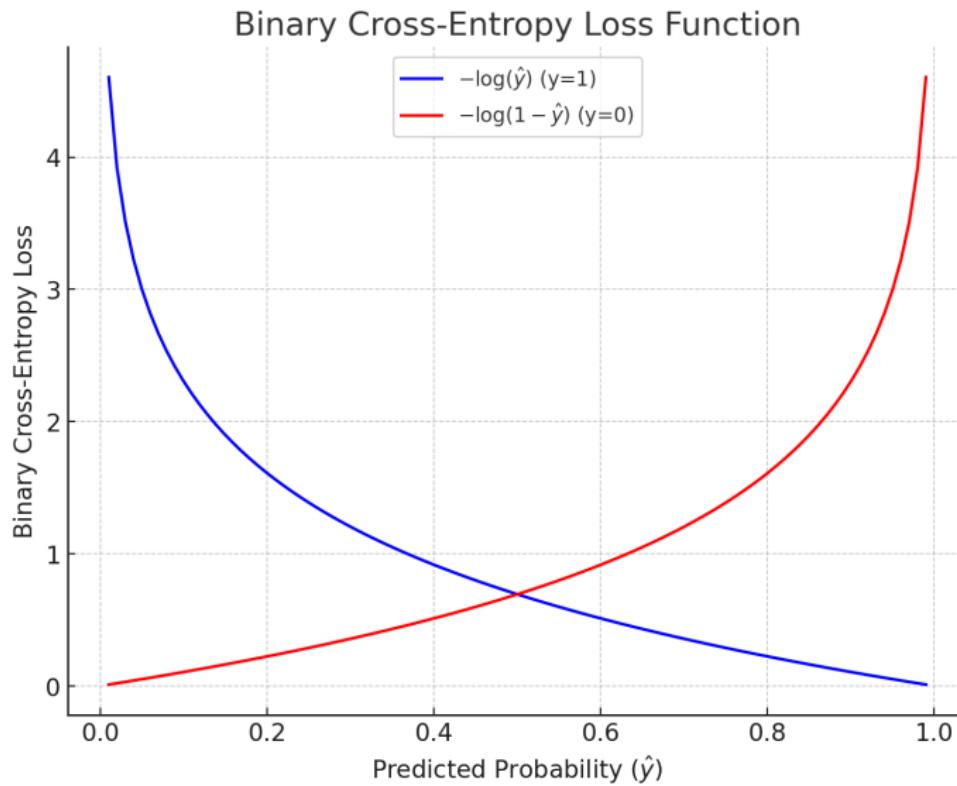
This means that if  $\hat{y}$  is close to 0, the loss is small; if  $\hat{y}$  is close to 1, the loss is large.

When the true label  $y=0$ , the loss follows  $-\log(1 - \hat{y})$ .



# Neural Networks

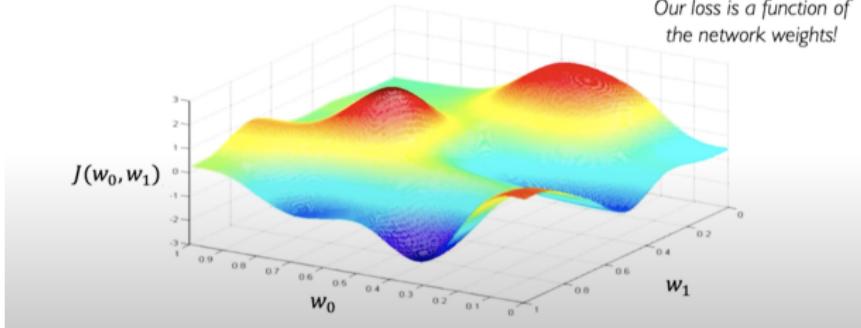
## Loss Functions



## Loss Optimization

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} J(\mathbf{W})$$

Remember:  
Our loss is a function of  
the network weights!

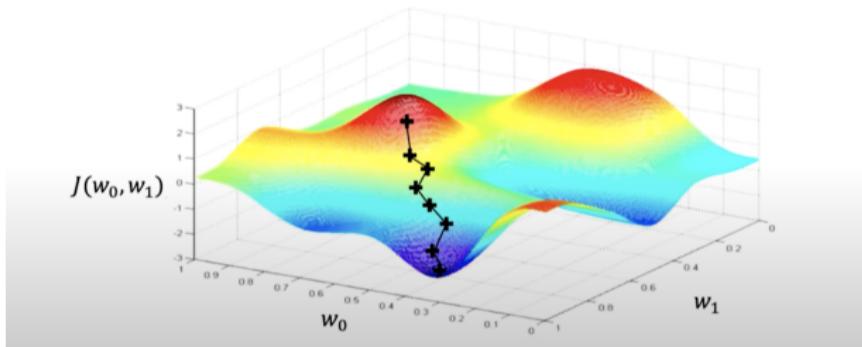


Identify Weights that Optimizes Total Loss  $J$



## Gradient Descent

Repeat until convergence



Gradient Descent Algorithm - Identify Weights that Optimizes Total Loss  $J$



### Gradient Descent

Gradient Descent is an optimization algorithm used to minimize a loss function by updating the model parameters iteratively. The key difference between **Batch Gradient Descent (BGD)**, **Stochastic Gradient Descent (SGD)**, and **Mini-Batch Gradient Descent (MBGD)** lies in *how much data* is used to compute the gradient at each step.

- Batch Gradient Descent Algorithm computes the gradient using the entire dataset before updating weights.

$$w = w - \eta \cdot \frac{1}{N} \sum_{i=1}^N \nabla L_i \quad (130)$$

where  $N$  is the total number of data points, and  $\eta$  is the learning rate.

- Stochastic Gradient Descent Algorithm computes the gradient and updates weights using only a **single random sample** at each iteration.

$$w = w - \eta \cdot \nabla L_i \quad (131)$$

where  $L_i$  is the loss of a single data point.



# Neural Networks

## Gradient Descent

- Mini-Batch Gradient Descent Algorithm computes the gradient using a **small random subset (mini-batch) of data** at each iteration.

$$w = w - \eta \cdot \frac{1}{m} \sum_{i=1}^m \nabla L_i \quad (132)$$

where  $m$  is the mini-batch size (typically 32, 64, 128, etc.).

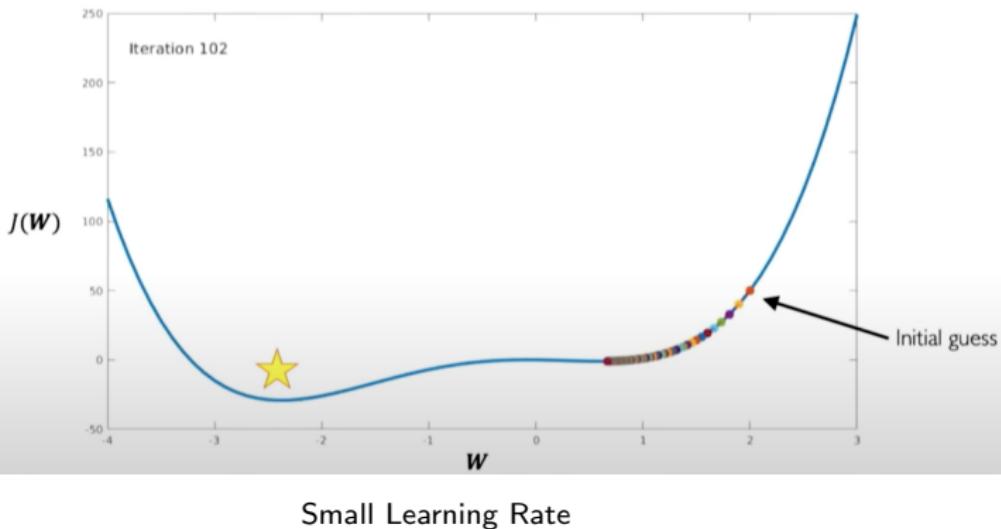
Method	Data Used Per Step	Convergence	
		Speed	Stability
Batch GD	Entire dataset	Slow	Stable
SGD	1 sample	Fast	Noisy
Mini-Batch GD	Small batch (e.g., 32, 64)	Moderate	Good balance

Comparison of Gradient Descent Methods



# Setting the Learning Rate

**Small learning rate** converges slowly and gets stuck in false local minima



# Neural Networks

## Convolutional Neural Networks CNN

### CNN

Convolutional Neural Networks (CNNs) are specialized neural networks designed for image processing and spatial data analysis.

- Stage 1 : Input Image → A grayscale or RGB image is fed into the network.
- Stage 2 : Feature Extraction (Convolution → ReLU → Pooling) → Filters detect edges, textures, and objects.
- Stage 3 : Classification (Fully Connected Layer + Softmax/Sigmoid) → Assigns probabilities to different classes.

### Applications

- Image Classification (e.g., ResNet, VGG, AlexNet)
- Object Detection (e.g., YOLO, Faster R-CNN)
- Facial Recognition (e.g., FaceNet)
- MRI Scan Analysis



# Neural Networks

## Convolutional Neural Networks CNN Architecture

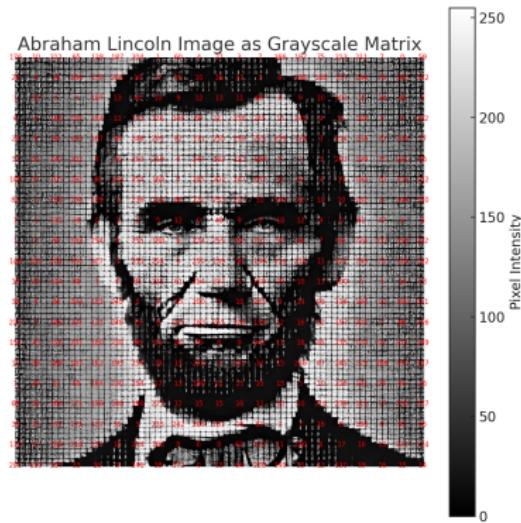
The architecture of CNNs for classification tasks is discussed below.

- Convolution Layer : Performs convolution operation using a filter (kernel) and extracts spatial features such as edges, textures, and patterns.
- Activation Layer : Introduces non-linearity after convolution. RELU is generally preferred.
- Pooling Layer : Downsamples feature maps to reduce dimensionality and computation. MaxPooling and Average Pooling are different pooling techniques.
- Fully Connected Neural Network : Flattens the feature maps, connects them to neurons and performs the final classification.
- SoftMax/Sigmoid Output Layer : SoftMax/Sigmoid Output Layers are used for multi-classification/binary classification problems.



# Neural Networks

## Convolutional Neural Networks CNN Architecture



Abraham Lincoln - Gray Scale Image

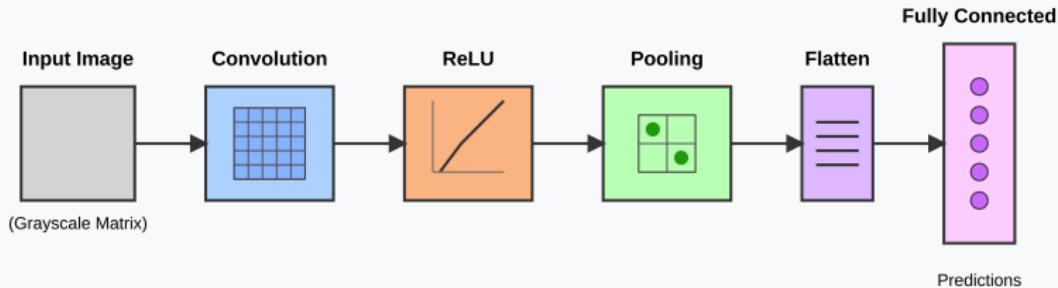
CNNs are superior to Fully Connected Neural Networks as they capture the spatial relationships in the data and can be implemented with less number of parameters.



# Neural Networks

## Convolutional Neural Networks CNN Architecture

### Convolutional Neural Network Architecture



CNN - Architecture for Classification

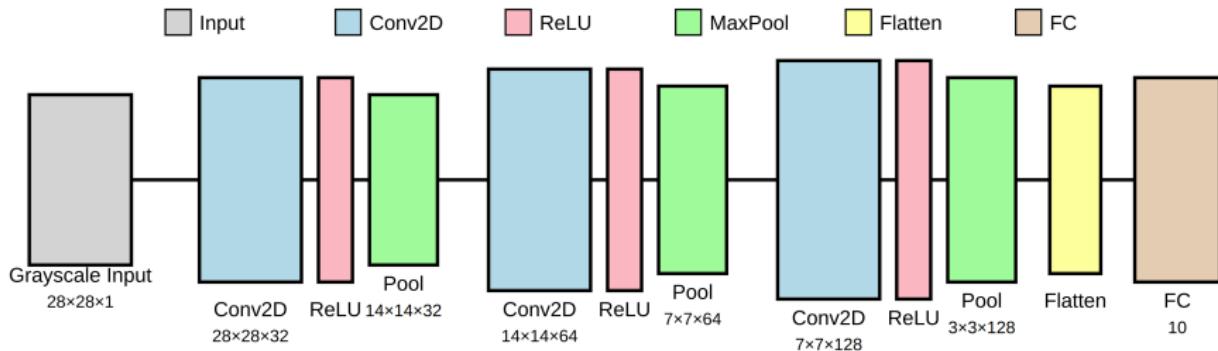
CNNs self-learn features and use those features as training inputs for the fully-connected neural network layers.



# Neural Networks

## Convolutional Neural Networks CNN Architecture

### Multi-Layer CNN Architecture for Grayscale Images



Multi-Layer CNN



# Neural Networks

## Convolutional Neural Networks CNN Architecture

### Convolution

- Convolution is the fundamental operation in Convolutional Neural Networks (CNNs), used for feature extraction from images.
- It helps detect low-level features of an image such as edges, textures, and patterns by applying a small filter (kernel) over an input image.
- It also helps in reducing the computations in CNN.

#### The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The diagram illustrates the convolution operation. On the left, an input matrix of size 5x5 is shown with values: 1, 1, 1, 0, 0; 0, 1, 1, 1, 0; 0, 0, 1<sub>v1</sub>, 1<sub>v0</sub>, 1<sub>v1</sub>; 0, 0, 1<sub>v0</sub>, 1<sub>v1</sub>, 0<sub>v0</sub>; 0, 1, 1<sub>v1</sub>, 0, 0. A 3x3 filter is shown in the middle with values: 1, 0, 1; 0, 1, 0; 1, 0, 1. A multiplication symbol ( $\otimes$ ) is placed between the input and the filter. To the right of the filter is an equals sign (=). To the right of the equals sign is a feature map of size 3x3 with values: 4, 3, 4; 2, 4, 3; 2, 3, 4. Below the feature map is the label "feature map".

Convolution Operation - 5\*5 Input Matrix, 3\*3 Filter, 3\*3 Feature Map



### Convolution Filters

Filters (also called kernels) in Convolutional Neural Networks (CNNs) are small matrices that slide over an image to detect patterns. They learn from the data using backpropagation and gradient descent.

#### What gets Captured by CNN Filters?

- First Layer → Simple edges, corners, and textures.
- Middle Layers → Complex shapes like eyes, wheels, or letters.
- Last Layers → High-level patterns like faces, objects, or digits

#### How Filters are Updated using Back Propogation

- Backpropagation starts from the FC layer and propagates backward to the convolutional layers.
- Pooling layers do not update parameters but pass gradients to previous layers.
- Filters (kernels) in CNN are updated using gradient descent, just like fully connected layers



# Neural Networks

## Filters - Sobel Filter for Edge Detection

### Sobel Filter

The **Sobel filter** detects edges by computing the **gradient**(change) of pixel intensity in an image. It consists of two filters:

- **Vertical Sobel Filter ( $G_x$ )** – Detects **horizontal** edges.
- **Horizontal Sobel Filter ( $G_y$ )** – Detects **vertical** edges.

#### 1. Vertical Sobel Filter ( $G_x$ )

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

#### 2. Horizontal Sobel Filter ( $G_y$ ) = $G_x^T$

After applying  $G_x$  and  $G_y$  separately, the final **edge strength** is computed as:

$$G = \sqrt{G_x^2 + G_y^2}$$

This combines the **horizontal and vertical edges** into a single gradient magnitude image.



# Neural Networks

## Filters - Sobel Filter for Edge Detection

- A large G value  $\Rightarrow$  Strong edge (significant intensity change).
- A small G value  $\Rightarrow$  Weak edge or smooth region (little intensity change).
- $G=0 \Rightarrow$  No edge at that pixel

Original Image



Edge Detected Image (Sobel Filter)



Vertical Sobel Filter



# Neural Networks

## Filters - Laplacian Filter for Sharpening

### Laplacian Filter

A sharpening kernel is a filter used in image processing to enhance edges and fine details. The most common sharpening kernel is derived from the Laplacian filter, which highlights regions of rapid intensity change.

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (133)$$

Original Image



Sharpened Image

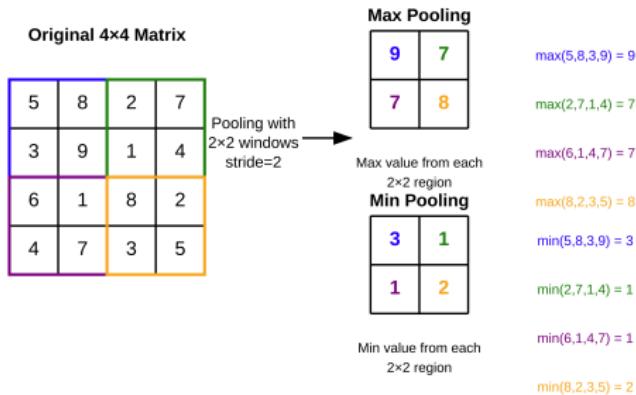


# Neural Networks

## Convolutional Neural Networks CNN Architecture

### Pooling

Pooling is a downsampling operation in Convolutional Neural Networks (CNNs) that reduces the spatial dimensions of feature maps while retaining important information. This helps in making the model computationally efficient and robust to small changes in the input image.



### Max Pooling and Average Pooling



# Neural Network

## Tensor & Tensorflow

### Tensor & Tensorflow

A tensor is a generalization of scalars, vectors, and matrices to higher dimensions. It is a multidimensional array used in deep learning and scientific computing. TensorFlow is an open-source machine learning framework that efficiently handles tensors and builds computational graphs. Tensors are optimized for parallel processing on GPUs & TPUs for high-performance deep learning.

Type	Example	Shape
Scalar (0D)	5	()
Vector (1D)	[1, 2, 3]	(3,)
Matrix (2D)	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	(2,2)
Tensor (3D+)	$[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]$	(2,2,2)

Examples include grayscale image stored as a 2D tensor (height  $\times$  width), colored image (RGB) stored as a 3D tensor (height  $\times$  width  $\times$  channels), a batch of images is stored as a 4D tensor (batch size  $\times$  height  $\times$  width  $\times$  channels).



# Bias and Variance in ML Models

## Bias

**Bias** in machine learning refers to systematic errors that cause a model's predictions to be consistently different from the true values.

$$\text{Bias} = \mathbb{E}[\hat{f}(x)] - f(x) \quad (134)$$

## Variance

**Variance** refers to the error introduced by the model's sensitivity to small fluctuations in the training dataset. A model with high variance pays too much attention to the training data and models the noise, leading to overfitting. Variance measures the variability of the model's prediction for different training sets:

$$\text{Variance} = \mathbb{E} \left[ (\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2 \right] \quad (135)$$



# Bias & Variance in ML Models

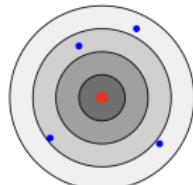


# Bias & Variance in ML Models

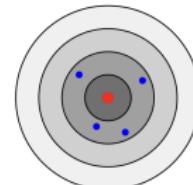
## Bias-Variance Tradeoffs

### Bias-Variance Combinations in Machine Learning

High Variance →

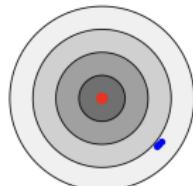


High Bias, High Variance  
(Poor Model)



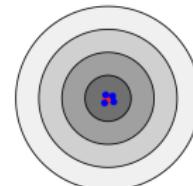
Low Bias, High Variance  
(Overfitting)

High Bias ↓



High Bias, Low Variance  
(Underfitting)

Low Variance →



Low Bias, Low Variance  
(Optimal Model)

Low Bias ↓

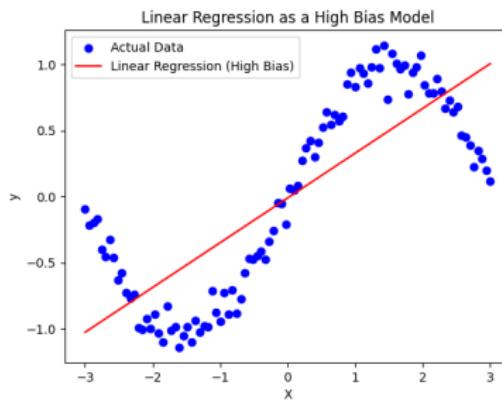
### Bias-Variance Tradeoff



# Bias & Variance in ML Models

## Linear Regression as a high Bias Model

Using linear regression to model a highly non-linear relationship results in a high-bias as the model is too simplistic to capture the true patterns in the data, leading to underfitting.



Linear Regression - High Bias

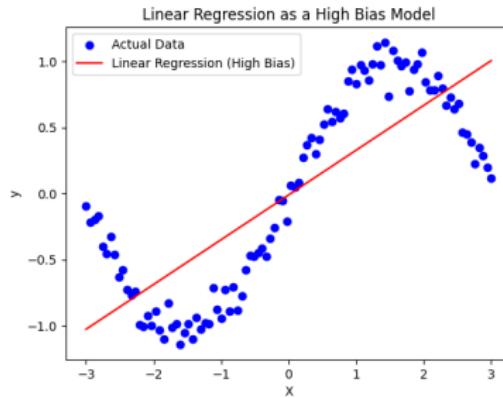
$$y = \alpha_0 + \alpha_1 \cdot x + \epsilon \quad (136)$$



# Bias & Variance in ML Models

## Polynomial Regression as a low Bias Model

Using polynomial regression with degree 5 to model a highly non-linear relationship results in a low-bias as the model is complex to capture the true patterns in the data, leading to overfitting.



## Polynomial Regression - Low Bias

$$y = \alpha_0 + \alpha_1.x + \alpha_2.x^2 + \alpha_3.x^3 + \alpha_n.x^n + \epsilon \quad (137)$$



# Bias & Variance in ML Models

## Bias-Variance Trade-Off

- The bias-variance tradeoff is a fundamental concept in machine learning that helps understand a model's ability to generalize to new data.
- High Bias (Underfitting): The model is too simple and cannot capture patterns in the data.
- High Variance (Overfitting): The model is too complex and captures noise in the training data, leading to poor generalization.
- For capturing simple patterns, models with lower variance (e.g., Linear Regression, Random Forest) are preferable.
- For capturing complex patterns, low-bias models like deep neural networks or SVM with kernels perform better but require regularization.
- Regularization techniques (L1/L2 regularization, pruning, dropout, cross-validation) help balance bias and variance.



# Bias & Variance in ML Models

Model Type	Bias	Variance
Linear Regression	High	Low
Polynomial Regression (High Degree)	Low	High
Decision Trees (Deep Trees)	Low	High
Pruned Decision Trees	Moderate	Moderate
Random Forest	Low	Low to Moderate
Support Vector Machines (SVM) with Linear Kernel	High	Low
SVM with RBF Kernel	Low	High
k-Nearest Neighbors (k-NN) (Low k value)	Low	High
k-NN (High k value)	High	Low
Neural Networks (Shallow)	High	Low
Neural Networks (Deep, Overtrained)	Low	High
Regularized Neural Networks (Dropout, Early Stopping)	Low	Moderate

Bias-Variance Tradeoff in Machine Learning Models



## Bias & Variance in ML Models

- Linear Regression → Assumes a linear relationship, often too simple for complex patterns.
- Polynomial Regression → Fits the training data well but may overfit if the degree is too high.
- Decision Trees → Can learn complex rules, but deep trees overfit easily.
- Pruned Decision Tree → A balanced tree avoids overfitting by reducing complexity.
- Random Forest → Aggregation of trees reduces variance while maintaining predictive power.
- SVM → Works well when data is linearly separable but struggles with complex data.
- SVM with Kernels → Can capture complex patterns but may overfit on small datasets.
- k-Nearest Neighbors (k-NN) → Very flexible, but small k values lead to overfitting. Large k smooths predictions, reducing variance but increasing bias.
- Neural Networks : Shallow Neural Networks has limited capacity to learn complex features. Deep Neural Networks can overfit if not regularized properly (e.g., no dropout).
- 



# Bias & Variance in ML Models

## Managing Bias and Variance

### Techniques to Manage Bias and Variance

Problem	Solution
High Bias	Use more complex models (e.g., increase model capacity). Reduce regularization (if applied too strongly).
High Variance	Use simpler models (e.g., reduce model capacity). Increase regularization (e.g., L1/L2 regularization). Use more training data. Apply ensemble methods (e.g., bagging, Random Forest).



# Bias & Variance in ML Models

## Regularization

### Regularization

Regularization is a technique used in machine learning to prevent overfitting by adding a penalty to the model's complexity during training. It helps to improve the model's ability to generalize to new, unseen data by discouraging it from fitting the noise or small fluctuations in the training dataset.

Regularization helps construct generalizable models by → preventing over-fitting with the addition of penalty to model complexity.

Types of Regularization.

- L1 Regularization
- L2 Regularization
- Elastic Regularization
- Drop-Out
- Early Stopping



# Bias & Variance in ML Models

## Types of Regularization

### ■ L1 Regularization (Lasso):

- Penalizes the sum of the absolute values of model coefficients.
- Mathematical Formulation:

$$\text{Loss} = \text{Loss Function} + \lambda \sum_{i=1}^n |w_i|$$

- Encourages **sparsity** in the model, meaning some coefficients are pushed to zero.
- Useful for **feature selection** since it tends to eliminate irrelevant features.

### ■ L2 Regularization (Ridge):

- Penalizes the sum of the squared values of model coefficients.
- Mathematical Formulation:

$$\text{Loss} = \text{Loss Function} + \lambda \sum_{i=1}^n w_i^2$$

- Encourages **small but non-zero coefficients**.
- Works well for models where features are strongly correlated but should not be eliminated.



# Bias & Variance in ML Models

## Types of Regularization

### ■ Elastic Net Regularization:

- A combination of both **L1** and **L2** regularization.
- **Penalizes** both the absolute values and the squared values of the coefficients.
- **Mathematical Formulation:**

$$\text{Loss} = \text{Loss Function} + \lambda_1 \sum_{i=1}^n |w_i| + \lambda_2 \sum_{i=1}^n w_i^2$$

- Combines the strengths of Lasso and Ridge regularization.
- Useful when there are many correlated features, and both feature selection and regularization are needed.

### ■ Dropout Regularization (Neural Networks):

- **Randomly drops** (sets to zero) a fraction of neurons during training.
- Prevents overfitting by ensuring the model does not rely too heavily on any particular neuron or feature.
- Especially effective in **deep neural networks**.

### ■ Early Stopping (Neural Networks):

- **Stops training early** if the validation loss does not improve for a certain number of epochs.
- Prevents overfitting by halting training before the model starts to memorize the training data.
- This is a form of implicit regularization, often combined with other techniques like dropout.

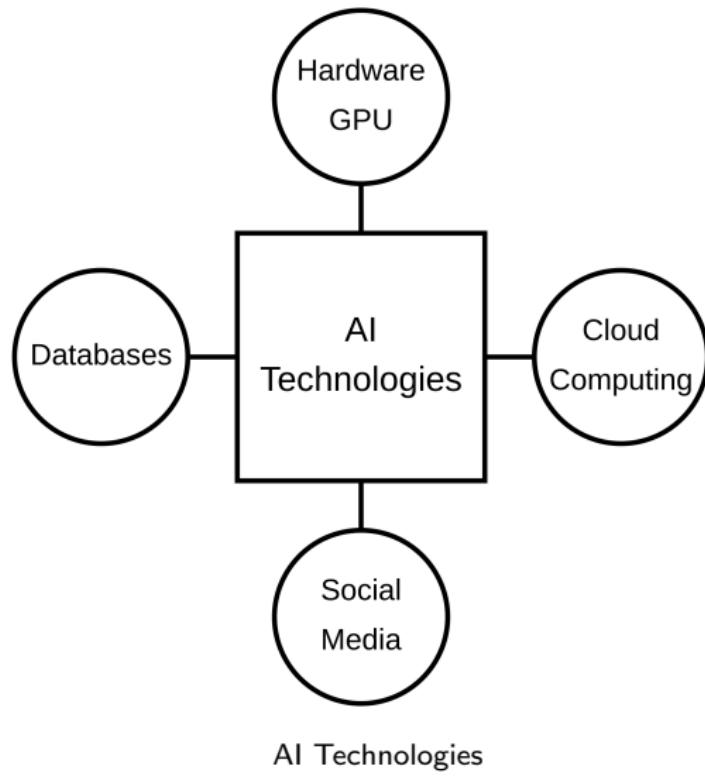


# **AI in 21<sup>st</sup> Century**



# Introduction

## Artificial Intelligence Ecosystem



# Introduction

AI and Hardware - Nvidia ,AMD, Intel

- GPUs (Graphics Processing Units) are the backbone of AI models as it reduces the training time.
- The latest AI advancements rely on GPUs for deep learning (large scale matrix multiplication) and high-performance computing.
- Unlike CPUs which have a few powerful cores, GPUs have thousands of smaller cores designed for **parallel** computations.
- GPUs have high memory bandwidth memory (HBM).

## CPU & GPU

CPUs are great for handling a few complex tasks quickly. GPUs are built to handle many simple tasks at once, reducing waiting time (latency) when processing large amounts of data.

- CPU - Sequential Processing with 4 to 64 cores with little parallelism, GPU - Thousands of simple cores for parallel processing.
- CPU - Excellent at handling complex diverse tasks, GPU - Excels at performing the same operation.
- CPU - Lower Memory Bandwidth 50-100 GB/s, GPU - Higher Memory Bandwidth 1-2 TB/s.



# Introduction

AI and Hardware - Nvidia ,AMD, Intel



CPU & GPU

Feature	CPU (Fine Dining Chef)	GPU (Fast-Food Kitchen)
Cores	Few, powerful chefs	Many, simple workers
Latency	High (each task takes time)	Low (many tasks done in parallel)
Throughput	Low (few meals per hour)	High (many meals per hour)
Best For	Complex, sequential tasks	Massive parallel tasks

Comparison of CPU and GPU - Chef Analogy



# Introduction

AI and Hardware - Nvidia ,AMD, Intel

- Physical CPU: This is the actual hardware component, often referred to as the processor, that performs the computations in a computer or effectively the brain.
- Core: A core is a smaller processing unit within the physical CPU. Modern CPUs have multiple cores, allowing them to perform multiple tasks simultaneously.
- Thread: A thread is the smallest unit of processing that can be scheduled by an operating system. Each core can handle multiple threads through a process called multi-threading, which allows for more efficient use of the CPU's resources.

Term	Meaning	Analogy	Example (AMD)
Physical CPU	Entire processor	The whole kitchen	1 CPU (single-core) or 2 CPUs (dual-core)
Core	Processing unit inside CPU	A chef in the kitchen	64 cores
Thread	Task that a core handles	The hands of the chef	128 threads

Physical CPU, Cores, and Threads - Chef Analogy



# Introduction

AI and Hardware - Nvidia ,AMD, Intel

- AI models like GPT, DALL-E, and AlphaFold require thousands of GPUs for training. Companies like NVIDIA provide AI-specific architectures like Tensor Cores and Hopper GPUs to accelerate processing. Eg : H100 Tensor Core GPU powers ChatGPT.
- AI Workstations & Data Centers – DGX systems (by NVIDIA) and Radeon Instinct (by AMD) power AI-driven workloads in research, self-driving cars, and healthcare.
- Optimized AI Frameworks – CUDA, cuDNN, ROCm and AI-specific libraries optimize neural networks for GPUs.
- CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA that allows developers to utilize GPUs (Graphics Processing Units) for general-purpose computing. It is widely used for AI, deep learning, scientific computing, and high-performance applications.



# Introduction

AI and the World's Largest GPU Companies - Nvidia ,AMD, Intel

- GPUs (Graphics Processing Units) are the backbone of AI models as it reduces the training time.
- The latest AI advancements rely on GPUs for deep learning (large scale matrix multiplication) and high-performance computing.
- Unlike CPUs which have a few powerful cores, GPUs have thousands of smaller cores designed for parallel computations.
- AI models like GPT, DALL-E, and AlphaFold require thousands of GPUs for training. Companies like NVIDIA provide AI-specific architectures like Tensor Cores and Hopper GPUs to accelerate processing. Eg : H100 Tensor Core GPU powers ChatGPT.
- AI Workstations & Data Centers – DGX systems (by NVIDIA) and Radeon Instinct (by AMD) power AI-driven workloads in research, self-driving cars, and healthcare.
- Optimized AI Frameworks – CUDA, cuDNN, ROCm and AI-specific libraries optimize neural networks for GPUs.
- CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA that allows developers to utilize GPUs (Graphics Processing Units) for general-purpose computing. It is widely used for AI, deep learning, scientific computing, and high-performance applications.



# Introduction

AI and the World's Largest Cloud Computing Companies -  
Amazon Web Services (AWS), Microsoft Azure, Google Cloud, IBM Cloud

- AI-as-a-Service (AlaaS) – Platforms like AWS SageMaker, Google Vertex AI, and Azure Machine Learning provide pre-trained AI models and tools for businesses.
- Scalability & Distributed AI Training – Cloud GPUs (NVIDIA A100 in Google Cloud, AWS EC2 P4d instances) enable companies to train AI models faster.
- Serverless AI & Edge Computing – AI services now run at the edge for low-latency applications (e.g., Autonomous Vehicles & IoT).
- OpenAI's GPT-4 runs on Microsoft Azure's cloud infrastructure, leveraging its AI supercomputing resources.
- Google Cloud has been used by Google DeepMind and Gemini models.
- Netflix uses Amazon Simple Storage Service S3 for storing terabytes of movies and TV shows.



# Introduction

AI and the World's Largest Database Companies - Oracle, Microsoft (SQL Server), Google (BigQuery), Amazon (AWS RDS)

AI integration in databases supports real-time fraud detection, customer insights, and predictive analytics.

- AI-Powered Query Processing – AI enhances databases by predicting optimal query execution plans (e.g., Oracle Autonomous Database uses ML for self-optimization).
- Data Warehousing for AI – Platforms like Google BigQuery enable AI models to process large-scale structured and unstructured data efficiently.
- Google BigQuery – A Fully Managed Data Warehouse Google BigQuery is a serverless, highly scalable, and cost-effective cloud data warehouse offered by Google Cloud Platform (GCP). It is designed for analyzing huge datasets using SQL-like queries with lightning-fast performance.
  - Serverless & Scalable
  - Support SQL queries
  - Built-in Machine Learning (Big Query ML)
  - Columnar Database
  - Integrates with AI & BI tools.
- AWS RDS (Amazon Relational Database Service) is a managed database service that makes it easy to set up, operate, and scale relational databases in the cloud.



# Artificial Intelligence

AI and the World's Largest Social Media Companies - Meta (Facebook, Instagram), Twitter, TikTok, LinkedIn, Snapchat

- Recommendation Engines – AI personalizes user feeds by analyzing behavior.
- AI-Powered Content Moderation – AI detects and removes hate speech, misinformation, and inappropriate content in real-time (Meta's AI tools analyze billions of posts).
- Deepfake Detection & AI Ethics – AI models help combat deepfakes and ensure ethical AI usage in social media.
- Size of the data generated is huge. Facebook processes 4+ petabytes of data daily. Twitter/X generates 500M+ tweets per day ( 12TB of data daily).
- Facebook & Instagram (Meta) use Deep Learning for personalized feeds (Engagement-based ranking).
- Graph Neural Networks (GNNs) has been used by facebook for friend recommendations.





AI in Healthcare

- AI-Powered Diagnostics : Detecting diseases like cancer using deep learning (e.g., Google's DeepMind).
- Personalized Medicine : AI-driven drug discovery and patient-specific treatment plans (e.g., IBM Watson Health).
- Medical Image Analysis : AI accurately interprets X-rays, MRIs, and CT scans faster than radiologists.
- Predicting Disease Outbreaks : AI + Big Data predict COVID-19 and other pandemics using global health records.



# AI in Banking & Finance



AI in Finance & Banking

- Fraud Detection : AI detects anomalies in financial transactions (e.g., PayPal, Mastercard).
- Algorithmic Trading : AI-driven hedge funds like Renaissance Technologies outperform traditional traders.
- Risk Assessment & Credit Scoring : Banks use AI to evaluate loan eligibility beyond credit scores.
- Chatbots & Customer Service : AI-powered assistants handle banking queries (e.g., Erica by Bank of America).





AI in Retail

- Recommendation Systems : Netflix, Amazon, and Spotify use AI to suggest products & content.
- Dynamic Pricing : AI adjusts prices based on demand, inventory, and competitor pricing.
- Supply Chain Optimization : AI predicts demand and automates inventory management.
- Customer Sentiment Analysis : Brands analyze social media & reviews using NLP.



# AI in Transportation



AI in Transportation

- Self-Driving Cars : Tesla, Waymo, and Uber use AI for autonomous driving.
- AI Traffic Management : Smart traffic lights optimize flow & reduce congestion (e.g., Google's AI traffic prediction).
- Predictive Maintenance : AI detects faults in aircraft & trains before failures occur.
- Route Optimization : Google Maps and Uber AI optimize travel routes.



# AI in Agriculture



AI in Agriculture

- Precision Farming : AI-driven drones & sensors optimize irrigation & fertilization.
- Crop Disease Prediction : AI detects diseases early to prevent food shortages.
- Automated Harvesting : Robots pick crops to address labor shortages (e.g., John Deere AI tractors).
- Food Supply Chain Optimization : AI reduces food waste & predicts demand.



# AI in Cyber Security



AI in Cyber Security

- **AI-Powered Threat Detection :** AI detects cyberattacks before they happen (e.g., Darktrace).
- **Identity Verification :** AI-powered biometrics (facial recognition, fingerprint scanning).
- **Automated Incident Response :** AI responds to cyber threats in real time.



# AI in Smart Cities & Environment



AI in Smart Cities & Environment

- Energy Grid Optimization – AI reduces power waste & integrates renewable energy.
- Wildlife Conservation – AI tracks endangered species & prevents poaching.
- Disaster Prediction – AI forecasts earthquakes, floods, and hurricanes using satellite data.
- Waste Management – AI-driven recycling robots sort trash efficiently.



# AI in Education



AI in Education

- AI Tutors & Personalized Learning – Platforms like Duolingo and Coursera adapt lessons based on performance.
- Automated Grading – AI speeds up grading of assignments & exams.
- Academic Research – AI accelerates scientific discoveries (e.g., AI-designed protein structures).
- Cheating Detection – AI prevents exam fraud using proctoring software.



# Problems



# Collaborative Filtering

## Problem 1

Assume we have three users ( $u_1, u_2, u_3$ ) and five movies ( $m_1, m_2, m_3, m_4, m_5$ ). Users rate movies on a scale from 1 to 5, and missing values indicate that the user hasn't rated that movie.

User	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$
$u_1$	5	4	3	?	2
$u_2$	4	?	5	3	1
$u_3$	?	4	4	2	3

## Collaborative Filtering

Hint!

$$R_{u,i} = \bar{R}_u + \frac{\sum_{v \in N(u)} S(u, v)(R_{v,i} - \bar{R}_v)}{\sum_{v \in N(u)} |S(u, v)|} \quad (138)$$

$$S(u_1, u_2) = (5, 3, 2) \cdot (4, 5, 1) = .93$$

$$S(u_1, u_3) = (4, 3, 2) \cdot (4, 4, 3) = .99$$

$$\bar{R}_u = \frac{14}{4} = 3.5, R_{u2, m4} - \bar{R}_{u2} = 3 - \frac{13}{4}, R_{u3, m4} - \bar{R}_{u3} = 3 - \frac{13}{4}$$

$$R_{u, m4} = ?$$



## Collaborative Filtering with SVD

Users	m1	m2	m3	m4	m5
u1	5	4	3	?	2
u2	4	?	5	3	1
u3	?	4	4	2	3

Step 1:

Users	m1	m2	m3	m4	m5	Row Mean
u1	5	4	3	3.5	2	3.5
u2	4	3.25	5	3	1	3.25
u3	3.25	4	4	2	3	3.25

$$R = \begin{bmatrix} 5 & 4 & 3 & 3.5 & 2 \\ 4 & 3.25 & 5 & 3 & 1 \\ 3.25 & 4 & 4 & 2 & 3 \end{bmatrix} \quad (139)$$



## Collaborative Filtering with SVD

Step 2: Apply SVD on R =>  $R = U.\Sigma.V^T$

$$U = \begin{bmatrix} -0.71 & 0.52 & -0.47 \\ -0.62 & -0.78 & 0.06 \\ -0.32 & 0.35 & 0.88 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 10.2 & 0 & 0 \\ 0 & 2.4 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.56 & -0.45 & -0.55 & -0.34 & -0.29 \\ 0.19 & -0.05 & -0.39 & -0.8 & 0.41 \\ -0.46 & 0.39 & -0.1 & 0.46 & 0.67 \end{bmatrix}$$



## Collaborative Filtering with SVD

Step 3 : Select only the top  $k = 2$  singular values:

$$U_2 = \begin{bmatrix} -0.71 & 0.52 \\ -0.62 & -0.78 \\ -0.32 & 0.35 \end{bmatrix}$$

$$\Sigma_2 = \begin{bmatrix} 10.2 & 0 \\ 0 & 2.4 \end{bmatrix}$$

$$V_2^T = \begin{bmatrix} -0.56 & -0.45 & -0.55 & -0.34 & -0.29 \\ 0.19 & -0.05 & -0.39 & -0.8 & 0.41 \end{bmatrix}$$

Step 4 : To predict missing values such as movie rating for  $m_4$  by  $u_4$ :

$$\hat{R}_{u1,m4} = [-0.71, 0.52] \begin{bmatrix} 10.2 & 0 \\ 0 & 2.4 \end{bmatrix} \begin{bmatrix} -0.34 \\ -0.8 \end{bmatrix} \quad (140)$$

$$\hat{R}_{u1,m4}?$$



# Association Rules

## Market Basket Analysis

Transaction	Items
T1	Bread, Milk
T2	Bread, Diaper
T3	Milk, Diaper
T4	Bread, Milk
T5	Bread, Milk, Diaper

Transaction dataset

$$\text{Support}(\text{Bread} \cap \text{Milk}) = \frac{3}{5} = 0.6$$

$$\text{Support}(\text{Bread}) = \frac{4}{5} = 0.8$$

$$\text{Support}(\text{Milk}) = \frac{4}{5} = 0.8$$

$$\begin{aligned}\text{Confidence}(\text{Bread} \Rightarrow \text{Milk}) &= \frac{\text{Support}(\text{Bread} \cap \text{Milk})}{\text{Support}(\text{Bread})} \\ &= \frac{0.6}{0.8} = 0.75\end{aligned}$$

$$\begin{aligned}\text{Lift}(\text{Bread} \Rightarrow \text{Milk}) &= \frac{\text{Confidence}(\text{Bread} \Rightarrow \text{Milk})}{\text{Support}(\text{Milk})} \\ &= \frac{0.75}{0.8} = .9375\end{aligned}$$

