First Step in Data Mining

What You Need to Know About Data Mining

From Basics → Classification → Clustering → Parallel Computing with MPI

Author:
ADEL AZZI

Edition: 2025

Institution / University

Student, University of Science and Technology Houari Boumediene (USTHB)

Index

❖ **Introduction to the bookμ**

❖ **Part I: Foundational Principles of Data Mining**

- **Chapter 1: Introduction to Data Mining**
  - **Definition and Core Concepts**
  - **The KDD Process (Knowledge Discovery in Databases)**
  - **Comparison: Data Mining, Machine Learning, and Statistics**
  - **Categorization of Data Mining Tasks**
  - **Key Applications and Tools**
- **Chapter 2: Data Preprocessing**
  - **Data Quality: Cleaning, Integration, and Transformation**
  - **Data Normalization and Standardization (Z-Score)**
  - **Strategies for Handling Missing Values and Outliers**
- **Chapter 3: Distance and Similarity Measures**
  - **Distance Metrics: Euclidean, Manhattan, and Minkowski Distances**
  - **Similarity Measures: Cosine Similarity**
  - **Practical Applications in Clustering and Classification**

❖ **Part II: Supervised Learning Techniques**

- **Chapter 4: K-Nearest Neighbors (KNN)**
  - **Theoretical Foundation and Intuition**
  - **Distance Metrics and the Selection of *k***
  - **Practical Implementation in Python**
- **Chapter 5: Decision Trees**
  - **Splitting Criteria: Entropy, Information Gain, and Gini Index**

    - **Algorithms: ID3, C4.5, and CART**
    - **Mitigating Overfitting with Pruning**
- **Chapter 6: Naive Bayes**
    - **Probabilistic Framework: Bayes' Theorem and Conditional Independence**
    - **Types and Applications (Gaussian, Multinomial, Bernoulli)**
    - **Text Classification with Naive Bayes**
- **Chapter 7: Logistic Regression**
    - **The Sigmoid Function and Decision Boundaries**
    - **Multiclass Strategies (One-vs-Rest, Softmax)**
    - **Cross-Entropy Loss and Feature Interpretation**
- **Chapter 8: Support Vector Machines (SVM)**
    - **Core Principle: Margin Maximization (Hard vs. Soft Margin)**
    - **Handling Non-linear Data with the Kernel Trick (RBF, Polynomial)**
    - **Hyperparameter Tuning (C and Gamma)**
- **Chapter 9: Introduction to Neural Networks**
    - **Fundamental Structure and Conceptual Overview**
    - **Typical Use Cases and Inherent Limitations**

Part III: Unsupervised Learning

- **Chapter 10: K-Means Clustering**
    - **Centroid-based Clustering**
    - **Step-by-Step Algorithm**
    - **Initialization Strategies for K-Means**
- **Chapter 11: K-Medoids (PAM)**
    - **Distinctions from K-Means**
    - **Differences Between K-Means and K-Medoids**

- **Chapter 12: Hierarchical Clustering**
    - **Agglomerative (AGNES) vs. Divisive (DIANA) Approaches**
    - **Dendrogram Visualization**
    - **Linkage Methods: Single, Complete, and Average**
- **Chapter 13: DBSCAN and Density-Based Clustering**
    - **Density Concepts and Parameter Tuning (Epsilon, MinPts)**
    - **Anomaly and Noise Detection**

Part IV: Advanced Concepts & High-Performance Data Mining

- **Chapter 14: Introduction to Parallel Computing**
    - **Introduction**
    - **What is Parallel Computing?**
    - **Why Parallel Computing?**
    - **Types of Parallelism**
    - **Parallel Architectures**
    - **Challenges in Parallel Computing**
- **Chapter 15: Introduction to MPI (Message Passing Interface)**
    - **Fundamentals: MPI_Send, MPI_Recv, and Collective Operations**
    - **Communicators, Rank, and Size**

- **Challenges**
- **Useful URLs**

{ Introduction to the Book }

Introduction to the Book

Welcome to this comprehensive guide on **Data Mining and High-Performance Clustering Techniques**. Whether you are a student, researcher, or data enthusiast, this book is designed to help you build a strong theoretical foundation while also mastering practical skills through hands-on examples.

To ensure a well-rounded learning experience, **each chapter** in this book is divided into **three structured sections**:

🔹 1. Course Part (Theoretical Concepts)

This section provides clear and concise **theoretical explanations** of the topic. You will explore key definitions, formulas, algorithms, and examples — designed to help you understand the core principles behind each method or model.

🔹 2. TD Part (Directed Exercises)

The TD (Travaux Dirigés) section includes a series of **guided exercises** to reinforce the theoretical knowledge. These problems range from simple to advanced levels, helping you apply what you've learned and prepare for exams or interviews.

🔹 3. TP Part (Practical Labs / Code Implementation)

The TP (Travaux Pratiques) section focuses on **hands-on programming** using tools like **Python**, **scikit-learn**, **mpi4py**, and more. You will learn how to implement algorithms, visualize results, and evaluate performance through real datasets.

Structure of the Book

The book is divided into **four main parts**:

- ❖ **Part I**: Fundamentals of Data Mining
- ❖ **Part II**: Supervised Learning
- ❖ **Part III**: Unsupervised Learning
- ❖ **Part IV**: Parallel and High-Performance Data Mining

Each part builds progressively, helping you move from basic concepts to advanced implementations using parallel architectures such as **MPI**.

Objective of the Book

- To provide a solid understanding of **data mining theory**
- To bridge the gap between **mathematical models** and **real-world coding**
- To introduce **parallel computing principles** applied to data science
- To prepare you for academic projects, research, or industrial roles

Target Audience

- Master's and engineering students in computer science, data science, or HPC
- Educators and researchers seeking a structured, practical resource
- Developers or analysts transitioning into machine learning or big data

We hope this book empowers you to **think critically**, **code confidently**, and **analyze data at scale**.

Let's begin your journey into data mining!

{ Introduction to Data Mining }

Chapter 1: Introduction to Data Mining

1.1 What is Data Mining?

Data mining is the **process of discovering patterns, correlations, anomalies, and useful information** from large volumes of data using statistical, machine learning, and computational techniques.

It is one of the core steps of the **Knowledge Discovery in Databases (KDD)** process and plays a crucial role in decision-making across industries.

In simple terms:

Data Mining = Extracting Knowledge from Data

It is not just about collecting data, but **analyzing it intelligently** to find **hidden relationships** or **predict future outcomes**.

1.2 The KDD Process

The term **KDD (Knowledge Discovery in Databases)** refers to the complete process of converting raw data into useful knowledge. It involves several key steps:

KDD Steps:

1. **Selection**: Choosing the relevant dataset or features.
2. **Preprocessing**: Cleaning data (handling missing values, removing noise).
3. **Transformation**: Converting data into a suitable format.
4. **Data Mining**: Applying algorithms to extract patterns.
5. **Evaluation & Interpretation**: Making sense of the patterns and using them.

- **Data mining is just one part of the KDD process**, but it is the most critical phase.

1.3 Data Mining vs. Machine Learning vs. Statistics

Although often used interchangeably, these fields have different focuses:

Aspect	Data Mining	Machine Learning	Statistics
Goal	Discover patterns from data	Make predictions or decisions	Analyze and model relationships
Data Orientation	Often large-scale historical data	Focused on prediction & generalization	Typically small to medium structured data
Main Techniques	Clustering, Association Rules, etc.	Supervised/Unsupervised algorithms	Probability, Hypothesis Testing
Tools	Weka, RapidMiner, SQL	Scikit-learn, TensorFlow, PyTorch	R, SAS, SPSS

➢ **Summary**:

- **Data mining** is more application-driven.
- **Machine learning** is model/prediction-driven.
- **Statistics** is theory/interpretation-driven.

1.4 Types of Data Mining Tasks

Data mining tasks are generally classified into two broad categories:

Supervised Learning (Labeled data)

- **Classification**: Predicts a categorical label (e.g., spam or not spam)
- **Regression**: Predicts a numeric value (e.g., price of a house)

Unsupervised Learning (No labels)

- **Clustering**: Grouping similar items together (e.g., customer segmentation)
- **Association Rule Mining**: Discovering rules in data (e.g., "people who buy X also buy Y")

Other Tasks

- **Anomaly Detection**: Identifying outliers or rare events (e.g., fraud detection)
- **Dimensionality Reduction**: Reducing feature space while keeping information (e.g., PCA)

1.5 Applications and Tools

Applications of Data Mining

- **Healthcare:** Data mining helps medical professionals by analyzing patient data to identify patterns that can assist in **diagnosing diseases** and grouping patients with similar conditions for more effective treatment.
- **Banking:** In the financial sector, it's a key player in **detecting fraudulent transactions** and creating **credit scores** to assess a person's credit risk.
- **Retail:** Retailers use it to understand customer behavior, allowing them to create **customer segments** for targeted marketing and build **recommendation systems** (like "customers who bought this also liked...").
- **Telecommunications:** Telecom companies use data mining to predict **customer churn**, helping them identify customers who are likely to leave so they can try to retain them.
- **IoT and High-Performance Computing (HPC):** In these advanced fields, data mining enables the **real-time detection of patterns** in the massive streams of data generated by connected devices.

Popular Tools & Languages

- **Python:** This is the most popular language for data mining. It has an extensive ecosystem of libraries like **Scikit-learn** for machine learning, **pandas** for data manipulation, and **NumPy** for numerical operations.
- **R:** This language is a favorite among statisticians and data scientists for its robust capabilities in **statistical analysis** and data **visualization**.
- **Weka:** This is a good choice for those who prefer a more visual approach. It's a **GUI-based tool** that provides a collection of machine learning algorithms for data mining tasks.
- **RapidMiner:** Another user-friendly, visual tool, RapidMiner allows you to build complex data mining **workflows using a drag-and-drop interface**.
- **SQL:** While not a data mining tool itself, **Structured Query Language (SQL)** is essential for **querying databases** and preparing data before it's used in any data mining process. It's often the first step in the workflow.

➢ TD

○ Exercise 1: Definition & Understanding

1. Define **data mining** in your own words.
2. What does the acronym **KDD** stand for? Describe each step in order.
3. Explain why **data mining** is considered the most critical step in the KDD process.

○ Exercise 2: Comparison Table Completion

Fill in the missing cells in the following table comparing **Data Mining**, **Machine Learning**, and **Statistics**:

Aspect	Data Mining	Machine Learning	Statistics
Main Goal	Discover patterns	?	?
Typical Data Used	Large-scale, historical	?	?
Common Techniques	Clustering, Association Rules	?	?
Tools/Platforms	Weka, RapidMiner	Scikit-learn, TensorFlow	R, SAS

○ Exercise 3: Classification Practice

Classify each of the following problems as **Supervised**, **Unsupervised**, or **Other** data mining tasks:

Scenario	*Task Type*
Predicting the price of a house based on its features	
Grouping customers based on purchasing behavior	
Detecting abnormal network activity in a security system	
Recommending products based on past purchases	
Reducing the number of features in a dataset with minimal information loss	

Exercise 4: Real-World Applications (Short Answer)

For each domain below, describe **one real-world application of data mining**:

1. Healthcare
2. Finance
3. Retail
4. Telecom
5. Internet of Things (IoT) or HPC

Exercise 5: Tool Identification

Match the following tools with their correct category or description:

TOOL	CATEGORY
WEKA	A. Statistical Language
SCIKIT-LEARN	B. Visual Workflow Tool
R	C. GUI-Based Mining Software
RAPIDMINER	D. Python ML Library
SQL	E. Query Language

Chapter 2: Data Preprocessing

"Garbage in, garbage out."

This popular phrase highlights the importance of data preprocessing. Before applying any mining or machine learning technique, the **quality** and **format** of your data must be addressed.

➢ Cours

2.1 Data Cleaning, Integration, and Transformation

Data Cleaning

This step removes **errors, inconsistencies, and noise** in data.
Tasks include:

- **Removing duplicates**
- **Correcting typos or formatting issues**
- **Filling missing values**
- **Filtering out noisy records**

Data Integration

Combining data from **multiple sources** into one consistent format.

Sources can include:

- Databases
- CSV files
- APIs
- Web scraping

Challenges:

- Schema mismatch
- Duplicate records across sources
- Conflicting values (e.g., birthdate differences)

Data Transformation

Prepares the data for mining through:

- **Normalization/Standardization**
- **Discretization** (converting continuous to categorical)
- **Feature construction** (e.g., combining year and month into a timestamp)
- **Encoding categorical data** (e.g., one-hot encoding)

2.2 Normalization & Standardization

Normalization

Used to **rescale values** between a range (usually 0 and 1).
Essential for algorithms like KNN or K-Means, which rely on **distance metrics**.

Formula (Min-Max Scaling):

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Example:
Original: [10, 20, 30]
Normalized: [0.0, 0.5, 1.0]

Standardization (Z-score Scaling)

Centers data around 0 with a standard deviation of 1.

Formula (Z-score):

$$Z = \frac{x - \mu}{\sigma}$$

Where:

- **X** is the value
- **μ** mu is the mean
- **σ** sigma is the standard deviation

Standardized values make it easier for **gradient-based models** to converge efficiently.

2.3 Z-Score Calculation (Worked Example)

Given:

$$X = 72 , \mu = 60 , \sigma = 8$$

Z-score:

$$z=\frac{72-60}{8} =\frac{12}{8}=1.5$$

Interpretation:
The value 72 is **1.5 standard deviations** above the mean.

Use cases:

- **Outlier detection**: Values with $|z| > 3$ are often outliers.
- **Feature scaling**: When variables have different units.
-

2.4 Handling Missing Values & Outliers

Missing Values

Approaches:

- **Ignore the row** (if very few)
- **Mean/Median Imputation**
- **Mode Imputation** (for categorical)
- **Advanced**: KNN Imputer, MICE (Multiple Imputation)

Outliers

Outliers are values that are **significantly different** from others.

Detection Methods:

- **Z-score Method**: If |z| > 3, consider as outlier
- **IQR Method**:

$$\mathbf{IQR= Q_3 - Q_1}$$

❖ Chapter Summary

- **Data Preprocessing:** Ensures high-quality, algorithm-compatible data for reliable model performance.
- **Feature Scaling:** Techniques like Z-score standardization and normalization enable fair feature comparisons by adjusting scales.
- **Data Cleaning:** Critical handling of missing values and outliers to prevent bias and improve model accuracy.

➢ TD

Exercise 1: Terminology Check

Match the following concepts to their correct definitions:

CONCEPT	DEFINITION
DATA CLEANING	A. Merging information from multiple sources into a unified format
DATA INTEGRATION	B. Replacing text labels with numeric codes
DATA TRANSFORMATION	C. Removing noise, correcting errors, and filling missing values
FEATURE ENCODING	D. Converting data into a suitable structure for mining

Exercise 2: Short Answer Questions

1. What are three common tasks performed during **data cleaning**?
2. List two challenges encountered during **data integration**.
3. Define **discretization** and give an example of when it might be useful.
4. Why is **data preprocessing** often said to be more time-consuming than model building?

Exercise 3: Normalization and Standardization Practice

Given the dataset:
X = [5, 15, 25, 35]

1. **Normalize** this data using Min-Max Scaling.
   (Hint: Rescale values between 0 and 1)
2. Calculate the **Z-score** for the value 25, given that:
   - Mean (μ) = 20
   - Standard Deviation (σ) = 10

Exercise 4: Z-Score Interpretation

You are given the following values:
$X = 88, \mu = 70, \sigma = 9$

1. Compute the **Z-score**.
2. Based on your result, determine if this value is an **outlier** using the Z-score method.

Exercise 5: Handling Missing Data

The following table contains missing values:

AGE	INCOME	GENDER
23	30000	Male
45	?	Female
34	50000	?
?	42000	Male

Questions:

1. What imputation method would you use for the missing **Income** value?
2. How would you fill in the missing **Gender** value?
3. Is it reasonable to drop the row with the missing **Age**? Why or why not?

Exercise 6: Outlier Detection with IQR

Given the following values:

[12, 15, 17, 21, 23, 25, 45]

1. Compute **Q1**, **Q3**, and **IQR**.
2. Use the IQR method to identify any **outliers**.

Exercise 7: Scenario-Based Application

Scenario: You are working on a telecom churn dataset with over 10,000 rows. Some values are missing and others are in different units (e.g., minutes vs. number of calls).

Answer the following:

1. What are **two preprocessing steps** you would apply first?
2. Would you use **normalization or standardization** before applying K-Means? Explain why.
3. How might outliers affect your clustering results?

➢ TP

Goal: Apply preprocessing techniques on a sample dataset using Python tools.
Tools: pandas, numpy, scikit-learn, matplotlib, seaborn

Step 0: Setup

```
pip install pandas numpy matplotlib seaborn scikit-learn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

Step 1: Load and Explore the Dataset

```
data = pd.DataFrame({
    'Age': [23, 45, 34, np.nan, 29, 67, 88],
    'Income': [30000, np.nan, 50000, 42000, 39000, 120000, 130000],
    'Gender': ['Male', 'Female', np.nan, 'Male', 'Female', 'Female', 'Male']
})

print("Dataset Preview:")
print(data)
```

Step 2: Data Cleaning

```
# Fill missing numerical values with mean
data['Age'].fillna(data['Age'].mean(), inplace=True)
data['Income'].fillna(data['Income'].median(), inplace=True)

# Fill missing categorical value with mode
data['Gender'].fillna(data['Gender'].mode()[0], inplace=True)

print("\nCleaned Dataset:")
print(data)
```

Step 3: Normalization (Min-Max Scaling)

```
scaler = MinMaxScaler()
data[['Age_normalized', 'Income_normalized']] = scaler.fit_transform(data[['Age',
'Income']])

print("\nAfter Normalization:")
print(data[['Age', 'Age_normalized', 'Income', 'Income_normalized']])
```

Step 4: Standardization (Z-score)

```
zscaler = StandardScaler()
data[['Age_zscore', 'Income_zscore']] = zscaler.fit_transform(data[['Age', 'Income']])

print("\nAfter Z-score Standardization:")
print(data[['Age', 'Age_zscore', 'Income', 'Income_zscore']])
```

Step 5: Outlier Detection (Z-score Method)

```
# Mark outliers where z-score > 3 or < -3
outliers = (np.abs(data['Income_zscore']) > 3)
print("\nOutlier Rows (Z-score Method):")
print(data[outliers])
```

Step 6: Outlier Detection (IQR Method)

```
Q1 = data['Income'].quantile(0.25)
Q3 = data['Income'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

iqr_outliers = data[(data['Income'] < lower_bound) | (data['Income'] > upper_bound)]

print("\nOutliers Detected via IQR:")
print(iqr_outliers)
```

Step 7: Visualization (Optional)

```
sns.boxplot(x=data['Income'])
plt.title("Income Distribution and Outliers")
plt.show()
```

Expected Learning Outcomes

By the end of this lab, you should be able to:

- ✓ Clean and fill missing data appropriately
- ✓ Apply normalization and standardization
- ✓ Detect outliers using both Z-score and IQR
- ✓ Understand how scaling affects data distribution

TP Coding Tasks

○ Exercise 1: Fill Missing Values

1. Fill missing numeric values in a DataFrame column using:
    - Mean
    - Median
2. Fill missing categorical values using the most frequent value (mode).

○ Exercise 2: Normalize and Standardize

1. Apply **Min-Max normalization** to two numeric columns.
2. Apply **Z-score standardization** to the same columns.

○ Exercise 3: Z-Score and Outlier Detection

1. Calculate the **mean** and **standard deviation** of a numeric list.
2. Compute the **Z-score** for each value.
3. Identify and print the values with absolute Z-score greater than 3.

○ Exercise 4: One-Hot Encoding

1. Apply **one-hot encoding** to a column containing city names in a DataFrame.

○ Exercise 5: Detect Outliers Using IQR

1. Calculate **Q1**, **Q3**, and the **Interquartile Range (IQR)** of a list of values.
2. Compute the **lower and upper bounds** for outlier detection.
3. Print values that fall outside these bounds as outliers

Chapter 3: Distance and Similarity Measures

➢ Cours

Many data mining and machine learning algorithms rely on the concept of **distance** or **similarity** to compare data points. Choosing the right metric can significantly affect model performance, especially in clustering and classification.

3.1 Euclidean Distance

The **Euclidean distance** is the most common and intuitive measure of distance in a **straight line** between two points in n-dimensional space.

Formula (2D case):

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

General form (n-dimensional):

$$\mathbf{d(p,q)= \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}}$$

Example:

Given:

$$p=(2,3),q=(6,7)$$

$$d = \sqrt{(6-2)^2 + (7-3)^2} = \sqrt{16+16} = \sqrt{32} = 5.66$$

Use cases: K-Means, KNN

3.2 Manhattan Distance

Also called **Taxicab** or **L1 distance**, it measures the total **absolute difference** across dimensions.

Formula:

$$\mathbf{d(p,q)= \sum_{i=1}^{n} |p_i - q_i|}$$

Example:

$$p=(1,2),q=(4,6)p = (1, 2)$$

$$d = |1 - 4| + |2 - 6| = 7$$

Use cases: Grid-based systems, text mining, decision trees

3.3 Minkowski Distance

This is a **generalization** of both Euclidean and Manhattan distances.
It introduces a parameter pp, which determines the type of distance:

Formula:

$$d(p, q) = (\sum_{i=1}^{n} | pi - qi |^{p})^{\frac{1}{p}}$$

- p=1p = 1: Manhattan distance
- p=2p = 2: Euclidean distance
- p→∞ : Chebyshev distance (max coordinate difference)

Useful when experimenting with different metrics in **KNN or clustering**.

3.4 Cosine Similarity

Instead of computing **distance**, cosine similarity measures the **angle** between two vectors. It is especially useful when magnitude is less important than orientation (e.g., text mining).

Formula:

$$\textbf{Cosine Similarity} = \frac{\boldsymbol{A} . \boldsymbol{B}}{||\mathbf{A}|| . ||\mathbf{B}||}$$

Where:

- A·B = dot product
- ||A|| = magnitude of vector A

Example:

$$A=(1,2),B=(2,4)A = (1, 2)$$

$$\text{Similarity} = \frac{(1 . 2 + 2 . 4)}{\sqrt{1^2 + 2^2} . \sqrt{2^2 + 4^2}} = \frac{10}{\sqrt{5} . \sqrt{20}} = 1$$

They are perfectly aligned ⇒ **Cosine similarity = 1**

Use cases: Text classification, document similarity (TF-IDF vectors)

3.5 Applications in Clustering and Classification

In Classification:

- **KNN**: Distance metric defines how neighbors are selected.
- **Anomaly Detection**: Outliers are far from normal clusters in terms of distance.
- **Content-based Filtering**: Cosine similarity helps match user preferences.

In Clustering:

- **K-Means**: Uses **Euclidean distance** to group similar points.
- **K-Medoids**: Can use **Manhattan** or **custom distances**.
- **DBSCAN**: Uses **distance threshold** to find dense regions.

Quick Comparison Table

Metric	Use Case	Strength	Weakness
Euclidean	K-Means, KNN	Simple, Intuitive	Affected by scale
Manhattan	Grids, Sparse data	Robust to outliers	Less natural in curved spaces

Minkowski	Flexible (p parameter)	Generalizes L1 and L2	Harder to interpret
Cosine Similarity	Text/data vectors	Works well with sparse vectors	Ignores magnitude

---

Chapter Summary

- Euclidean and Manhattan are the most used distance measures.
- Cosine similarity is ideal for text and vector space models.
- Choosing the right metric depends on your **data type**, **domain**, and **algorithm**.

- TD

Exercise 1: Conceptual Understanding

1. Define the following terms:
   - Euclidean Distance
   - Manhattan Distance
   - Cosine Similarity
2. What is the key difference between **distance** and **similarity**?
3. Which distance measure is more affected by outliers: Euclidean or Manhattan? Justify.

Exercise 2: Compute by Hand

Given the points:
A = (2, 4), B = (5, 8)

1. Calculate the **Euclidean Distance** between A and B.
2. Calculate the **Manhattan Distance** between A and B.
3. Calculate the **Minkowski Distance** with p=3p = 3.
4. Calculate the **Cosine Similarity** between A and B.

Exercise 3: Multiple Choice Questions

Q1. Which metric is least sensitive to feature scaling?

- A. Euclidean Distance
- B. Manhattan Distance
- C. Cosine Similarity
- D. Minkowski Distance with p = 2

Q2. Cosine similarity ranges from:

- A. [−1,1][-1, 1]
- B. [0,1][0, 1]
- C. [0,∞)[0, \infty)
- D. [−∞,∞][-∞, ∞]

Q3. Minkowski distance becomes Manhattan when:

- A. $p=0$p = 0
- B. $p=1$p = 1
- C. $p=2$p = 2
- D. $p=\infty$p = ∞

○ Exercise 4: Application Reasoning

1. Which distance would be better in a grid-like environment (e.g., city blocks)? Why?
2. Why is **normalization** often necessary before computing Euclidean distance?
3. In text mining, why is **cosine similarity** often better than Euclidean distance?

○ Exercise 5: Metric Comparison Table (Complete)

Fill in the missing information in the following table:

Metric	Main Advantage	Limitation
Euclidean	Simple and widely used	Sensitive to feature scale
Manhattan	Robust in grid-based models	May not work in all spaces
Cosine Similarity	Good for text/document vectors	Ignores vector magnitude
Minkowski (p=3)	Generalizes L1 & L2 distances	Harder to interpret

➢ TP

Data Set URL :

https://github.com/renatopp/arff-datasets/blob/master/classification/diabetes.arff

❖ Exercise 1: Euclidean and Manhattan Distance

Pick two patients (rows 0 and 1) and:

1. Extract their numerical features (preg, plas, ..., age)
2. Compute:
    - Euclidean distance
    - Manhattan distance
    - Minkowski distance with p=3

❖ Exercise 2: Cosine Similarity

1. Compute cosine similarity between the same two rows (patients 0 and 1).
2. What does the result tell you about their **similarity of profile**, regardless of magnitude?

❖ Exercise 3: Normalization and Standardization

1. Normalize the full dataset using Min-Max scaling.
2. Standardize it using Z-score (mean = 0, std = 1).
3. Compare the effect of normalization vs. standardization using the first 5 rows.

❖ Exercise 4: Outlier Detection

1. Calculate the Z-score for all numerical columns.
2. Identify rows where any feature has $|z|>3$.
3. Use IQR method on the insulin and mass columns:
    - Compute Q1, Q3, and IQR
    - Detect outliers

❖ Exercise 5: Distance After Preprocessing

1. Apply normalization to the dataset.
2. Recalculate Euclidean and Manhattan distances between rows 0 and 1 using the **normalized data**.
3. Compare results with the raw data version — what changes and why?

{ Part II: Supervised Learning }

Part II: Supervised Learning

❖ Classification in Supervised Learning

Classification is a central task in **supervised learning**, where the goal is to assign input data to one of several **predefined categories or labels**. It involves learning from a **labeled dataset**—each data point is associated with a known class—so that the model can predict the correct class for new, unseen data.

- **How Classification Works**

  During training, the model identifies patterns or decision boundaries that separate the classes. Once trained, it uses those patterns to classify new inputs.

  Examples:

  1. Email: spam or not spam
  2. Tumor: benign or malignant
  3. Image: cat, dog, or horse

Key Classification Algorithms (Brief Overview)

- ❖ K-Nearest Neighbors (KNN)
- ❖ Naive Bayes
- ❖ Decision Trees
- ❖ Support Vector Machines (SVM)

Comparison Table of Classification Algorithms

Algorithm	Training Time	Interpretability	Strengths	Weaknesses	Good Use Cases
KNN	Low	Low	Simple, no training phase	Slow prediction, sensitive to K	Image recognition, small datasets
Naive Bayes	Very Fast	Medium	Efficient, handles high dimensions	Assumes feature independence	Spam filtering, text classification
Decision Tree	Fast	High	Easy to understand and visualize	Prone to overfitting	Rule-based systems, finance
SVM	Moderate	Low	Effective in high-dimensional space	Hard to interpret, needs tuning	Bioinformatics, image classification

❖ Chapter 1: **Performance measures of a classifier**

In classification, we evaluate how well a model predicts **correct class labels**. There are several performance measures, each showing a different aspect of the model's behavior.

❖ Accuracy

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

- Easy to compute
- Not reliable with **imbalanced data**

❖ Precision (Précision)

$$\textbf{Precision} = \frac{TP}{TP+FP}$$

- **How many of the predicted positives are actually correct?**
- Important when **false positives** are costly (e.g., spam detection).

❖ Recall (Sensibilité ou Rappel)

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

- **How many actual positives were correctly identified?**
- Important when **false negatives** are dangerous (e.g., medical diagnosis).

❖ F1-Score

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

- Harmonic mean of **precision** and **recall**
- Best when we need a **balance** between precision and recall

❖ Confusion Matrix

	PREDICTED POSITIVE	PREDICTED NEGATIVE
ACTUAL POSITIVE	True Positive (TP)	False Negative (FN)
ACTUAL NEGATIVE	False Positive (FP)	True Negative (TN)

- Gives a **complete overview** of prediction results
- Useful for manually calculating other metrics

Summary: When to Use What?

Measure	Type	Use Case
Accuracy	Classification	Balanced datasets
Precision	Classification	High cost for false positives
Recall	Classification	High cost for false negatives
F1-Score	Classification	Need balance between precision & recall

❖ Chapter 2: K-Nearest Neighbors (KNN)

The **K-Nearest Neighbors (KNN)** algorithm is a fundamental method in machine learning, widely used for both **classification** and **regression** tasks. It is characterized by its **non-parametric** nature and **lazy learning** approach, making it simple yet powerful in a variety of applications.

➢ Cours

What is KNN?

KNN is a **memory-based algorithm** that does not learn an explicit model during training. Instead, it **stores the entire training dataset**, and defers computation to the **prediction phase**, where it relies on proximity-based reasoning.

Lazy Learning

Unlike eager algorithms that build a model during the training phase (e.g., Decision Trees, SVM), KNN is a **lazy learner**, meaning:

- It performs **no generalization** or abstraction during training.
- All computation is **postponed** until a prediction is required.
- As a result, **training is fast**, but **prediction can be computationally expensive**, especially with large datasets.

How KNN Works: Step-by-Step (Classification)

To classify a new instance xxx:

1. **Distance Calculation**: Compute the distance between xxx and all points in the training set using a chosen **distance metric**.
2. **Neighbor Selection**: Identify the **K closest neighbors** to xxx.
3. **Majority Voting**: Assign the class label that appears most frequently among the K neighbors.

For Regression:

Instead of majority voting, the output is typically the **mean** (or **weighted average**) of the numerical values of the K nearest neighbors.

❖ *Key Characteristics of KNN*

Property	Description
Non-parametric	Does not assume any specific functional form for the data distribution.
Instance-based	Stores all training examples for use during inference.
Local	Predictions are made based on a localized region in the feature space.
Versatile	Can handle both classification and regression tasks.

❖ *Real-World Analogy*

Imagine you've just moved to a new city and want to estimate the **rent of an apartment** in your neighborhood. You might ask several **nearby residents** about their rent prices. You'll assume that similar homes in the **same area** are likely to have similar prices. This is the essence of KNN: **"similar things exist close together."**

❖ *Distance Metrics in KNN*

The effectiveness of KNN greatly depends on the distance metric used to determine "closeness" between data points. Common choices include:

- **Euclidean Distance**
- **Manhattan Distance**
- **Minkowski Distance** (generalized)

- **Note**: Feature **normalization or standardization** is recommended when features are on different scales.

❖ *Choosing the Right Value of K*

The parameter **K** controls the number of neighbors considered during prediction, and it has a direct effect on the model's bias-variance tradeoff:

Value of K	Behavior	Consequence
Small (e.g., K = 1)	High variance, sensitive to noise	Can lead to **overfitting**
Large (e.g., K = 20)	High bias, smoother boundaries	Can lead to **underfitting**

To select the optimal K:

- Use **cross-validation** techniques.
- Visualize **error rates** for different K values (elbow method).

- Consider the **odd K** rule in binary classification to avoid ties.

❖ *Summary*

- **KNN** is a simple yet powerful algorithm that makes predictions based on the labels of the closest training examples.
- It is most effective when the dataset is **small to medium-sized** and well-structured.
- Its performance heavily depends on the choice of **distance metric**, the **value of K**, and proper **data preprocessing** (e.g., scaling).
- Though computationally expensive at prediction time, KNN remains a valuable tool in the machine learning toolbox due to its **interpretability**, **non-assumptive nature**, and **flexibility**.

❖ Python Implementation

Simple and efficient implementation using **scikit-learn**:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
```

➢ TD

Dataset Exercise

You are given the following labeled dataset:

INSTANCE	FEATURE 1 (X1)	FEATURE 2 (X2)	CLASS
A	1	2	Red
B	2	3	Red
C	3	3	Blue
D	5	1	Blue
E	3	2	Red

Q1. Using **K = 3**, predict the class of a new instance with coordinates:
X = (3, 1)
Use **Euclidean distance** to calculate the nearest neighbors.

Q2. Repeat the classification using **K = 1** and **K = 5**.
How does the choice of K affect the result?

Conceptual Questions

Q3. What happens if K is too small or too large? Explain the trade-off between bias and variance in KNN.

Q4. Why is KNN considered a *lazy learning* algorithm? What are the implications of that in terms of:

- Training time
- Prediction time

Q5. Which distance metrics can be used in KNN? Give at least three and explain when each might be preferred.

Q6. KNN assumes that “closer” points have similar classes. In which cases might this assumption fail?

Q7. How would KNN behave with:

- High-dimensional data?

- Data with irrelevant features?
- Data with noise

➢ TP

❖ Part 1: Dataset Creation

T1. Generate and Display Dataset

- Use **make_classification(n_samples=150, n_features=2, n_classes=3, n_clusters_per_class=1, n_redundant=0)** to create well-separated classes.
- Plot with **matplotlib.pyplot.scatter()**, coloring points by class labels.

T2. Train-Test Split

- Use train_test_split(X, y, test_size=0.3, random_state=42) (70% train, 30% test).

T3. Predict New Instance

- Manually add a point (e.g., [0.5, 0.5]) and predict its class using KNN (later steps).

❖ Part 2: Manual KNN Implementation

T4. Distance Functions

- Implement Euclidean: sqrt(sum((x1 - x2)^2)) and Manhattan: sum(|x1 - x2|).
- Test both to compare results.

T5. Custom KNN Predictor

For a test point:

- Calculate distances to all training points.
- Sort distances and pick top K neighbors.
- Majority vote (mode) decides the class.

T6. Optimal K Selection

- Loop over K (1 to 15), compute accuracy on test set for each.
- Plot accuracy vs. K; choose K with highest accuracy (often odd to avoid ties).

❖ Part 3: Comparison with scikit-learn

T7. Sklearn KNN Benchmark

- Use KNeighborsClassifier(n_neighbors=k):
- Time training with time.time().
- Compare accuracy (accuracy_score) and confusion matrix (confusion_matrix).

T8. Custom vs. Sklearn

- Check if predictions match (debug if not).
- Sklearn is faster (optimized Cython).
- Sklearn handles ties by default (prefers first class); weights by distance (weights='distance').

❖ Part 4: Advanced Tasks

T9. Noise & Irrelevant Features

- Add noise: Flip 10% labels randomly.
- Add 3 random noise columns. Observe accuracy drop (KNN is sensitive to irrelevant features).

T10. High-Dimensional Data

- Generate data with 10+ features. Note the "curse of dimensionality" (distances become meaningless).

T11. Imbalanced Classes

- Generate 90% class A, 10% class B. Risk: Bias toward majority class. Use metrics like F1-score, not just accuracy.

T12. Weighted KNN

- Modify your KNN: weight votes by 1 / (distance + epsilon) (closer neighbors matter more).

❖ Chapter 3: Naïve Bayes

➢ Cours

1. Introduction

Naïve Bayes is a family of probabilistic classifiers based on Bayes' Theorem with the "naïve" assumption that features are conditionally independent given the class label.
It is especially popular for:

- Text classification (spam detection, sentiment analysis)
- Document categorization
- Medical diagnosis
- Recommender systems

---

2. Bayes' Theorem Refresher

For events A (class) and B (features):

$$\mathbf{P(A \mid B) = \frac{P(B|A) \cdot P(A)}{P(B)}}$$

Where:

- P(A|B) → Posterior probability (probability of class A given features B)
- P(B|A) → Likelihood
- P(A) → Prior probability
- P(B) → Evidence (normalizing factor)

---

3. Naïve Independence Assumption

If we have features x1 , x2 , ..., xn :

$P(x1,x2,...,xn|C) = \prod_{i=1}^{n} P(xi \mid C)$

This greatly simplifies computation — we just multiply the likelihoods of individual features.

4. Variants of Naïve Bayes

- Gaussian Naïve Bayes — Assumes continuous features follow a normal distribution.

- Multinomial Naïve Bayes — Best for count-based features (word counts in text).
- Bernoulli Naïve Bayes — Best for binary/boolean features.

5. Example: Text Classification

Suppose we want to predict whether an email is spam based on the presence of certain keywords:

1. Compute P (spam) and P (not spam) from the dataset (priors) .
2. Compute P (word | spam) and P (word | not spam) for each word (likelihoods) .
3. Multiply and normalize using Bayes' theorem.
4. Pick the class with the highest posterior.

6. Laplace Smoothing

Avoids zero probability when a feature value never appears in a class:

$$P(x_i \mid C) = \frac{\text{count}\ (\text{xi}, C) + 1}{\text{count}\ (C) + m}$$

where mmm is the number of possible feature values.

7. Strengths

- Simple, fast, and efficient.
- Works well with high-dimensional data (e.g., text).
- Requires small training data to estimate parameters.

8. Weaknesses

- Assumes feature independence (often unrealistic).
- Performs poorly when features are highly correlated.
- Not suitable for continuous features without transformation.

9. Performance Measures

Same metrics as in KNN:

- Accuracy, Precision, Recall, F1-score.
- Confusion Matrix to analyze classification performance.

10. Summary Table

Property	Naïve Bayes
Type	Probabilistic, generative model
Assumption	Conditional independence
Data type	Categorical, discrete, or Gaussian
Speed	Very fast training and prediction
Best for	Text data, high-dimensional input

➢ TD

Datasets:

1. Dataset Weather Conditions for Tennis

Instance	Outlook	Temperature	Humidity	Windy	PlayTennis
1	Sunny	Hot	High	False	No
2	Sunny	Hot	High	True	No
3	Overcast	Hot	High	False	Yes
4	Rain	Mild	High	False	Yes
5	Rain	Cool	Normal	False	Yes
6	Rain	Cool	Normal	True	No
7	Overcast	Cool	Normal	True	Yes
8	Sunny	Mild	High	False	No

2. Dataset Buys Computer

INSTANCE	AGE	INCOME	STUDENT	CREDITRATING	BUYSCOMPUTER
1	<=30	High	No	Fair	No
2	<=30	High	No	Excellent	No
3	31-40	High	No	Fair	Yes
4	>40	Medium	No	Fair	Yes
5	>40	Low	Yes	Fair	Yes
6	>40	Low	Yes	Excellent	No
7	31-40	Low	Yes	Excellent	Yes
8	<=30	Medium	No	Fair	No
9	<=30	Low	Yes	Fair	Yes
10	>40	Medium	Yes	Fair	Yes
11	<=30	Medium	Yes	Excellent	Yes
12	31-40	Medium	No	Excellent	Yes
13	31-40	High	Yes	Fair	Yes
14	>40	Medium	No	Excellent	No

Questions:

Dataset 1

1. Calculate P(PlayTennis =Yes) and P (PlayTennis = No).
2. Compute P (Sunny | Yes), P (Sunny | No) , etc.
3. Predict PlayTennis for:
   Outlook = Sunny, Temperature = Cool , Humidity = High, Windy = True.
4. Repeat with and without Laplace smoothing — compare results.
5. Explain when Naïve Bayes would fail for this dataset.

Dataset 2

1. Calculate P(BuysComputer =Yes) and P(BuysComputer =No).
2. Compute P (Student | Yes), P (Student | No) , etc.
3. Predict BuysComputer for:
   Student = no , Age = <=30 , Income = High, CreditRating = Excellent.
4. Repeat with and without Laplace smoothing — compare results.
5. Explain when Naïve Bayes would fail for this dataset.

➢ TP

- **Part 1:** Dataset Creation
  - Use sklearn.datasets.fetch_20newsgroups for text classification (2 or 3 categories).
  - Preprocess using:
    - CountVectorizer (for Multinomial NB)
    - TfidfVectorizer (optional)
  - Split into train/test sets.
- **Part 2:** Manual Implementation
  - Implement Naïve Bayes formula manually for discrete counts.
  - Include Laplace smoothing in your implementation.
- **Part 3:** Using scikit-learn

  python

  ---

  from sklearn.naive_bayes import MultinomialNB

  from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

  ---

  - Compare accuracy with manual version.
  - Show confusion matrix.
- **Part 4:** Experiments
  - Test BernoulliNB for binary features.
  - Try GaussianNB on numeric datasets.
  - Compare accuracy and F1-score across types.
  - Add noise to the dataset and observe performance changes.
- **Part 5:** Challenge
  - Implement incremental learning with partial_fit (simulate streaming data).

❖ Chapter 4 : Decision Trees

➢ Cours

1. Introduction

Decision Trees are supervised learning algorithms used for both classification and regression. They work by recursively splitting datasets into smaller subsets based on feature values, creating a tree-like structure where:

- **Nodes** → represent tests on features.
- **Branches** → represent outcomes of those tests.
- **Leaves** → represent the final prediction (class label or value).

Advantages:

- Easy to interpret and visualize.
- Handle categorical and numerical data.
- Naturally model non-linear decision boundaries.

Disadvantages:

- Prone to overfitting.
- Can be unstable if small changes occur in data.

2. How Decision Trees Work

1. Select the best feature at each node using a purity measure.
2. Split the dataset accordingly.
3. Continue recursively until a stopping criterion (max depth, pure leaf, or minimum samples).

3. Purity Measures

❖ **Entropy**

$$H(S) = \sum_{i=0}^{n} pi \log2 (pi)$$

- p_i : proportion of class i .
- H (S) = 0 : perfectly pure**.**

Information Gain

$$IG(S, A) = H(S) - \sum_{k=1}^{k} \frac{|Sk|}{|S|} H(S_k)$$

- Measures how much entropy decreases after a split.
- The feature with highest IG is selected.

Gini Index

$$Gini(S) = 1 \sum_{i=1}^{n} p_i^2$$

- Simpler and faster than entropy.
- Used in **CART** algorithm.

4. Popular Algorithms

Algorithm	Criterion	Notes
ID3	Entropy & Information Gain	Works only with categorical features
C4.5	Gain Ratio	Handles continuous features & missing values
CART	Gini Index	Builds binary trees, supports regression

5. Overfitting & Pruning

❖ **Problem: Overfitting**

- A deep tree may perfectly fit training data but fail on test data.

❖ **Solutions**

- **Pre-Pruning:** Limit depth, min samples per split, early stopping.

- **Post-Pruning:** Fully grow tree, then remove branches with little impact using validation data.

6. Summary

- Decision Trees split data recursively using **Entropy, Information Gain, or Gini Index.**
- Popular algorithms**: ID3, C4.5, CART**.
- They are **interpretable but prone to overfitting.**
- **Pruning** is essential for generalization.

➢ TD

TD1: Entropy and Information Gain

Given the dataset:

Weather	Temperature	Play Tennis
Sunny	Hot	No
Sunny	Mild	No
Overcast	Hot	Yes
Rainy	Mild	Yes
Rainy	Cool	Yes
Overcast	Cool	Yes

Tasks:

1. Compute the entropy of the target variable Play Tennis.
2. Compute the entropy of the dataset when splitting by Weather.
3. Compute the Information Gain for Weather.
4. Which feature would be chosen as the root node?

TD2: Gini Index

For the same dataset, compute:

1. The **Gini Index** before any split.
2. The Gini Index for splitting on *Temperature*.
3. Compare results with entropy-based splitting.

TD3: Conceptual Questions

1. What is the main difference between Entropy and Gini Index?
2. Why might a tree built using ID3 overfit?

3. Explain pre-pruning vs post-pruning with examples.
4. Which algorithm (ID3, C4.5, CART) would you choose for mixed numerical + categorical data? Why?

TD4: Building a Small Decision Tree

Dataset:

Age	*Student*	*Buys Computer*
Young	**No**	**No**
Young	**Yes**	**Yes**
Middle	**No**	**Yes**
Senior	**Yes**	**Yes**
Senior	**No**	**No**

Tasks:

1. **Compute Entropy for *Buys Computer*.**
2. **Calculate Information Gain for both *Age* and *Student*.**
3. **Choose the best root attribute.**
4. **Build the first two levels of the decision tree.**

TD5: Practical Scenario Analysis

You are training a decision tree for loan approval prediction.
Features: Income (High/Low), Credit Score (Good/Bad), Employment (Stable/Unstable).
Target: Loan Approved (Yes/No).

Tasks:

1. **List possible overfitting risks when using decision trees in this scenario.**
2. **Propose two pre-pruning strategies and one post-pruning method to reduce overfitting.**
3. **Explain how imbalanced data (e.g., 90% "No" and 10% "Yes") would affect entropy and splitting.**

➢ TP

TP1: Dataset Creation

- Create a synthetic dataset with features (*Weather*, *Temperature*, *Humidity*) and a target (*Play Tennis*).
- Explain dataset parameters: size, number of features, target distribution.

Tasks:

1. Build a Decision Tree Classifier using **sklearn.tree.DecisionTreeClassifier**.
2. Visualize the tree (**plot_tree**) .
3. Interpret the root node and first split.

TP2: Splitting Criteria

- Train one tree using entropy, another using gini.

Tasks:

1. Compare the depth of both trees.
2. Evaluate accuracy on training & testing data.
3. Explain the differences in results.

TP3: Overfitting & Pruning

- Train a tree without restrictions (no max depth).
- Train another with constraints:
  - **max_depth = 3**
  - **min_samples_split = 5**

Tasks:

1. Compare performance (accuracy, precision, recall).
2. Visualize both trees.
3. Explain how pruning affects overfitting.

TP4: From-Scratch Implementation

- Implement your own decision tree splitting function:
    1. Compute entropy manually.
    2. Write a function to calculate Information Gain.
    3. Build a simple recursive decision tree.

Task: Compare your custom implementation with sklearn results.

TP5: Real Dataset Experiment

- Use a real dataset such as Iris or Titanic.

Tasks:

1. Train a decision tree classifier.
2. Visualize the decision tree structure.
3. Identify the most important features (using .feature_importances_).
4. Compare performance with a Logistic Regression model.

TP6: Hyperparameter Tuning

- Explore decision tree hyperparameters:
    - max_depth
    - min_samples_leaf
    - criterion (entropy/gini)

Tasks:

1. Use GridSearchCV to find the best parameters.
2. Report the best model parameters and accuracy.
3. Discuss the trade-off between bias and variance.

❖ Chapter 5 : Support Vector Machines (SVM)

➢ Cours

4. Introduction

Support Vector Machines (SVMs) are powerful supervised learning algorithms used for classification and regression.
The main idea: find the best hyperplane that separates data points of different classes with the maximum margin.

- Works well in high-dimensional spaces.
- Effective for datasets where classes are not linearly separable (via kernel trick).
- Robust against overfitting in many cases.

❖ Intuition of SVM

- Imagine two classes of points in a plane. Many lines could separate them.
- SVM selects the optimal line (or hyperplane) that maximizes the margin (the distance between the line and the closest data points of each class).
- These closest points are called support vectors — they "support" or define the decision boundary.

❖ Mathematical Formulation

➢ Linear SVM (Hard Margin)

Given training data (x_i , y_i) with $y_i \in \{-1,+1\}$:
We want to find a hyperplane:

$$\mathbf{W \cdot X + b = 0}$$

where w is the weight vector and b the bias.

The margin is:

$$\frac{\mathbf{2}}{\mathbf{|| w ||}}$$

Optimization problem:

$$\min \frac{1}{2} ||w||^2 \text{ subject to } y_i (W \cdot X_i + b) \geq 1$$

Soft Margin SVM

- Real-world data is not perfectly separable.
- Introduces slack variables ξi\xi_iξi to allow misclassifications.

$$\min \frac{1}{2} ||w||^2 + C \sum_i \xi_i$$

- C = penalty parameter (trade-off between margin maximization and error tolerance).

4. Kernel Trick

When data is not linearly separable, SVM projects it into a higher-dimensional space using kernels:

- Linear Kernel : $K (x , x') = x \cdot x'$
- Polynomial Kernel : $K (x , x') = (x \cdot x' + 1)^d$
- RBF (Gaussian) Kernel :

$$K (x , x') = \exp (-\gamma || x - x' ||^2)$$

Key Parameters:

- **C :** Controls trade-off between margin and misclassification.
- **γ (gamma) :** Defines influence of individual points in RBF kernel.

5. Applications of SVM

- Text classification (spam filtering, sentiment analysis).
- Image classification (e.g., handwritten digit recognition).
- Bioinformatics (gene classification, protein detection).

6. Advantages and Limitations

- **Advantages:**
  - Effective in high-dimensional spaces.
  - Works well with clear margin of separation.
  - Flexible with kernel functions.
- **Limitations:**
  - Computationally expensive for large datasets.
  - Choice of kernel and parameters is critical.
  - Less interpretable than Decision Trees.

❖ **Summary**

- SVM finds the **maximum-margin hyperplane** for separation.
- **Support vectors** are critical points defining the boundary.
- **Soft margin** handles noisy data.
- **Kernel trick** allows handling of non-linear problems.
- Key hyperparameters: **C , γ , and kernel type.**

➢ TD

TD1: Linear Separation

Given the dataset:

Point	x1	x2	Class
A	2	3	+1
B	3	3	+1
C	3	2	-1
D	2	1	-1

Tasks:

1. Draw the points on a 2D plane.
2. Find a separating line manually (by inspection).

3. Which points are support vectors?

TD2: Margin Computation

For a separating hyperplane $2x_1 + x_2 - 4 = 0$:

1. Compute the distance of point (2,3) from the hyperplane.
2. Compute the margin size.
3. Is the point correctly classified if its class is +1?

TD3: Conceptual Questions

1. Explain the difference between hard margin and soft margin SVM.
2. Why is the kernel trick important?
3. What is the role of the C parameter?
4. Compare SVM with Logistic Regression.

➢ TP

TP1: Linear SVM

- Create a synthetic 2D dataset (two separable classes).

Tasks:

1. Train a linear SVM (SVC (kernel = "linear")).
2. Visualize the decision boundary & support vectors.
3. Compute accuracy on training & test sets.

TP2: Soft Margin Effect

- Use same dataset but introduce noise (mislabel some points).

Tasks:

1. Train models with different C values (0.1, 1, 10, 100).
2. Plot decision boundaries.
3. Explain how C affects overfitting/underfitting.

TP3: Non-Linear SVM with Kernels

- Create a dataset where classes are not linearly separable (e.g., concentric circles).

Tasks:

1. Train SVM with **linear kernel** → observe poor performance.
2. Train with **RBF kernel** → visualize boundary.
3. Experiment with different **gamma values** .

TP4: Real-World Example

- Load the Iris dataset. *Data bases parts*

Tasks:

1. Train SVM with different kernels (linear, poly, RBF).
2. Compare performance using accuracy, precision, recall, F1-score.

3. Discuss which kernel works best and why.

TP5 : From Scratch

- Implement a simple linear SVM using optimization (gradient descent or quadratic programming).
- Compare results with scikit-learn implementation.

❖ Chapter 6: Neural Networks

➢ Cours

1. Introduction

Artificial Neural Networks (ANNs) are a family of **machine learning models** inspired by the structure of the human brain.
They consist of **layers of interconnected neurons** (nodes) that learn to transform inputs into meaningful outputs by adjusting **weights** during training.
Neural networks are extremely powerful for **non-linear problems** and form the basis of **deep learning**.

2. Structure of a Neural Network

A typical feed-forward neural network has:

- **Input Layer**: Takes features (e.g., pixels, measurements).
- **Hidden Layers**: Transform inputs through weighted connections and activation functions.
- **Output Layer**: Produces final predictions (class labels, regression values).

Each connection has:

- **Weight (w)**: Determines importance of input.
- **Bias (b)**: Shifts the activation.

3. Mathematical Model of a Neuron

For a neuron receiving inputs x1,x2,...,xn with weights w1,w2,...,wn :

$$\mathbf{Z} = \sum_{i=1}^{n} \boldsymbol{w_i}\, \boldsymbol{x_i} + \boldsymbol{b}$$

The output is obtained by applying an **activation function**:

$$\mathbf{a = f\,(\,z\,)}$$

4. Activation Functions

Activation functions introduce **non-linearity** into the network :

- **Sigmoid**:

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

Outputs between 0 and 1 (good for probabilities).

- **Tanh**:

$$\mathbf{Tanh\,(\,z\,) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}}$$

Outputs between -1 and 1.

- **ReLU (Rectified Linear Unit)**:

$$\mathbf{F(z) = \max(0, z)}$$

Efficient, prevents vanishing gradient problem.

- **Softmax**:
  For multiclass classification, outputs probability distribution:

$$\mathbf{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

5. Learning in Neural Networks

Training involves two main steps:

Forward Propagation

- Input data flows through the network.
- Activations are computed layer by layer.
- Output is compared with target label → loss computed.

Loss Functions

- **Mean Squared Error (MSE)** for regression.
- **Cross-Entropy Loss** for classification:

$$\mathbf{L} = -\sum y_i \log(\hat{y}_i)$$

Here's what each term means:

- $\mathbf{y_i}$ → The *true label* for class i.
  - In classification, this is usually a **one-hot vector**.
  - Example: If the true class is "cat" out of {cat, dog, horse}, then y=[1,0,0]y = [1, 0, 0]y=[1,0,0]
- $\hat{\mathbf{y}}_\mathbf{i}$ → The *predicted probability* for class i.
  - This comes from the model's output after applying **softmax** (for multi-class classification).
  - The softmax ensures all are $\hat{\mathbf{y}}_\mathbf{i}$ between 0 and 1 and sum to 1.
  - Example: If the model predicts probabilities

$$\hat{\mathbf{y}} = [0.7, 0.2, 0.1]$$

then it thinks "cat" with 70% probability, "dog" with 20%, "horse" with 10%.

➢ So in words: $\hat{\mathbf{y}}_\mathbf{i}$ is the **predicted probability** that the input belongs to class i.

Backpropagation

- Gradients of the loss w.r.t weights are computed using the **chain rule**.
- Weights are updated using **gradient descent**:

$$\mathbf{W := w - \eta \frac{\partial L}{\partial w}}$$

where η is the learning rate.

6. Important Hyperparameters

- **Learning Rate (η):** Step size in weight updates.
- **Number of Hidden Layers & Neurons:** Controls model capacity.
- **Batch Size:** Number of samples processed before updating weights.
- **Epochs:** Number of passes through the full dataset.

7. Advantages and Limitations

- **Advantages:**
  - Can model complex non-linear relationships.
  - Applicable to text, images, speech, time series.
  - Basis of deep learning (CNNs, RNNs, Transformers).
- **Limitations:**
  - Requires large amounts of data.
  - Training is computationally expensive.
  - Less interpretable than Decision Trees.
  - Sensitive to hyperparameter choices.

8. Applications of Neural Networks

- Image recognition (e.g., face detection).
- Natural language processing (sentiment analysis, translation).
- Medical diagnosis (disease prediction).
- Autonomous driving (object detection).

9. Summary

- Neural networks are built from **neurons arranged in layers**.
- Learning happens via **forward propagation + backpropagation**.
- Activation functions allow modeling **non-linear problems**.
- Hyperparameters strongly influence performance.
- Basis for modern **deep learning architectures**.

➢ TD

- **TD1: Single Neuron**

A neuron receives inputs $x_1 = 2$, $x_2 = -1$, with weights $w_1 = 0.5$, $w_2 = -0.3$, and bias $b=0.2$.

1. Compute $z = w_1 x_1 + w_2 x_2 + b$.
2. Apply **Sigmoid** activation.
3. Interpret the result if the neuron is used for binary classification.

- **TD2: Activation Functions**

1. Compare **Sigmoid vs ReLU**. Why is ReLU preferred in deep networks?
2. For $z = -2$, 0 , 2 compute outputs of Sigmoid, Tanh, and ReLU.
3. Which function is best for multiclass problems?

- **TD3: Backpropagation**

1. Explain why backpropagation requires the **chain rule**.
2. What happens if the learning rate is too high? Too low?
3. Why do deep networks suffer from the **vanishing gradient problem**?

➢ TP

○ TP1: Basic Neural Network

- Create a small dataset (e.g., XOR problem).

Tasks:

1. Implement a simple neural network with one hidden layer using **scikit-learn MLPClassifier**.
2. Compare results with Logistic Regression.
3. Plot decision boundary.

○ TP2: Hyperparameter Tuning

- Use a classification dataset (e.g., Iris).

Tasks:

1. Train networks with different **hidden layer sizes (5, 10, 50 neurons)**.
2. Experiment with different **activation functions** (sigmoid, tanh, ReLU).
3. Observe the effect of **learning rate** on convergence.

○ TP3: Overfitting and Regularization

- Use MNIST subset (digits 0–1).

Tasks:

1. Train a small neural network.
2. Observe overfitting with many hidden neurons.
3. Apply **regularization**:
    - Dropout
    - Early stopping
    - L2 penalty
4. Compare results.

○ TP4: From Scratch (Optional Challenge)

1. Implement forward and backward propagation manually (no ML libraries).
2. Train on a toy dataset (e.g., XOR).
3. Compare results with scikit-learn implementation.

❖ Summary of Part II:

Classification is a fundamental task in **supervised learning**, where the model learns from **labeled data** to distinguish between predefined categories. The goal is to build a predictive function that assigns unseen instances to their correct class.

To achieve this, various algorithms are used — each with different strategies, assumptions, and strengths. Some rely on **distance**, others on **probabilistic reasoning**, **tree structures**, or **optimization margins**. The performance of each algorithm depends on the **data characteristics**, such as dimensionality, feature correlation, noise level, and size of the dataset.

When choosing a classification algorithm, consider:

- **Interpretability** – Do you need to explain how decisions are made? (e.g., Decision Trees)
- **Scalability** – Is the dataset large or high-dimensional? (e.g., Naive Bayes, SVM)
- **Speed** – Do you need fast training or prediction?
- **Accuracy vs Simplicity** – Is model performance or simplicity more important?

In Practice:

- For **small and clean datasets**, KNN and Decision Trees work well.
- For **text and high-dimensional data**, Naive Bayes and SVM often perform best.
- For **interpretable decisions**, Decision Trees are preferred.
- For **complex patterns**, ensemble methods or Neural Networks may be required.

Ultimately, classification is not about finding *one perfect algorithm*, but rather understanding your data and choosing the model that best fits the problem — often validated using **cross-validation and performance metrics** like accuracy, F1-score, precision, and recall.

Algorithm	Key Use Case	Pros	Cons
KNN	Any simple classification	No training time, intuitive	Slow for big data
Decision Trees	Rule-based systems	Interpretable, handles mixed data	Prone to overfitting
Naive Bayes	Text classification	Fast, works on high-dim data	Strong independence assumption
Logistic Reg.	Binary/multiclass	Interpretable, fast	Assumes linear decision boundary
SVM	Margin-based classification	Effective in high-dim spaces	Sensitive to parameter tuning
Neural Networks	Complex pattern learning	Powerful, scalable	Less interpretable, requires tuning

{ Part II: Unsupervised Learning }

Part III: Unsupervised Learning

❖ Chapter 1 : Introduction to Unsupervised Learning and Clustering

1. What is Unsupervised Learning?

Unsupervised Learning is a branch of machine learning where the model learns from **unlabeled data**.

- Unlike **Supervised Learning**, where we have input-output pairs (features + labels), in **Unsupervised Learning** we only have inputs XXX.
- The goal is to discover **hidden structures, patterns, or groupings** in the data.

Example:

- In customer data with no labels, unsupervised learning can group similar customers together → useful for marketing segmentation.
- In text data, it can uncover hidden topics.

2. Main Tasks in Unsupervised Learning

There are two major families of unsupervised tasks:

1. **Clustering** → grouping similar data points.
    - Example: Market segmentation, anomaly detection.
2. **Dimensionality Reduction** → reducing the number of features while preserving structure.
    - Example: PCA (Principal Component Analysis), t-SNE for visualization.

Other tasks include **density estimation** and **representation learning**.

3. Clustering: Core Concept

Clustering is the most fundamental and widely used unsupervised technique.

- It groups data into **clusters**, where:
    - Points in the same cluster are **similar**.
    - Points in different clusters are **dissimilar**.

❖ **Visual Example**:
Imagine plotting 1000 points of customer data by age and income.

- Clustering might reveal 3 natural groups:
    - Young + low income.
    - Middle-aged + medium income.
    - Older + high income.

This structure is **not given by labels**—it's discovered.

4. Types of Clustering

Clustering methods vary in how they define similarity and form groups:

- **Partition-based** (e.g., K-Means):
  Divides data into kkk clusters by minimizing distance to cluster centers.
- **Hierarchical**:
  Builds a tree of clusters (dendrogram), either bottom-up (agglomerative) or top-down (divisive).
- **Density-based** (e.g., DBSCAN):
  Finds clusters of high density separated by sparse regions, good for non-spherical clusters.
- **Model-based** (e.g., Gaussian Mixture Models):
  Assumes data is generated from a mixture of probability distributions.

5. Challenges in Clustering

- **Number of clusters (k)** is often unknown.
- Results depend on distance metric (Euclidean, cosine, etc.).
- Sensitive to noise and outliers.
- No universal "best" clustering—depends on context.

6. Evaluation of Clustering

Since labels are missing, evaluating clustering is tricky:

- **Internal evaluation** (without labels):
    - Silhouette Score, Davies-Bouldin Index.
- **External evaluation** (if labels are available for testing):
    - Adjusted Rand Index, Mutual Information.

7. Applications of Clustering

Clustering is applied in many domains:

- **Business**: Market segmentation, customer profiling.
- **Biology**: Grouping genes with similar expression.
- **Healthcare**: Identifying disease subtypes.
- **Finance**: Fraud/anomaly detection.
- **NLP**: Topic discovery, document grouping.
- **Computer Vision**: Image segmentation.

8. Advantages and Limitations

❖ **Advantages**

- No need for labeled data.
- Helps discover hidden patterns.
- Useful for exploratory data analysis.

❖ **Limitations**

- Sensitive to algorithm choice and distance metrics.
- Often requires parameter tuning (e.g., number of clusters).
- Interpretability may be subjective.

9. Summary

- **Unsupervised learning** works with unlabeled data to reveal structure.
- **Clustering** is the most important unsupervised task, grouping data points by similarity.
- Different clustering approaches exist (partition, hierarchical, density, model-based).
- Evaluation is challenging, but internal/external metrics help.
- Applications span business, healthcare, biology, NLP, and computer vision.

Chapter 2: K-Means

➢ Cours

1. Introduction

K-Means is one of the most widely used **clustering algorithms** in unsupervised learning.
It partitions a dataset into **K clusters**, where each data point belongs to the cluster with the nearest mean (called the **centroid**).

- Goal: minimize the **intra-cluster variance** (distance within clusters).
- K-Means works best when clusters are **compact, spherical, and well-separated**.

2. Intuition

Imagine a set of points scattered on a 2D plane:

- If we choose K=3K = 3K=3, the algorithm will try to find **3 cluster centers**.
- Each point is assigned to the closest center.
- The centers are updated iteratively until they stabilize.

Example: Customers grouped into segments (low, medium, high spenders) based only on income and age.

3. The K-Means Algorithm

Steps

1. **Initialization**:
   Randomly select K points as initial centroids (e.g., C1 , C2 , ... , CK).
2. **Assignment Step**:
   Assign each data point x_i to the cluster of the closest centroid:

$$\textbf{Cluster (xi) = arg min}_j \textbf{ d}(x_i, C_j)$$

- **arg min** finds the argument that gives the minimum value. It answers the question: "Which input causes the function to be at its minimum?"

Example: arg min{5, 2, 9, 1} = 4 (the position or index of the smallest value, which is the 4th element). In mathematics, we often use the label itself. If these were distances to centroids C1=5, C2=2, C3=9, C4=1, then arg min would be C4 (the centroid itself, which is the argument), not the value 1.

- where **d (·)** is a distance metric (usually Euclidean).

3. **Update Step**:
   Recompute each centroid as the mean of its assigned points:

$$\mathbf{Cj} = \frac{\mathbf{1}}{\mathbf{|Sj|}} \sum_{x_i \in S_j} \boldsymbol{x_i}$$

   where S_j is the set of points in cluster j.

4. **Repeat** steps 2 and 3 until centroids don't change (convergence).

❖ **Objective Function**

K-Means minimizes the **Within-Cluster Sum of Squares (WCSS)**:

$$\mathbf{J} = \sum_{j=1}^{k} \sum_{x_i \in S_j} \| \mathbf{xi} - \mathbf{Cj} \|^2$$

4. Choosing K

Choosing the right number of clusters is critical.

- **Elbow Method**: Plot WCSS vs. KKK → choose KKK where the curve bends.
- **Silhouette Score**: Measures how well-separated the clusters are (ranges from -1 to 1).

5. Advantages and Limitations

❖ **Advantages**

- Simple and fast (scales well to large datasets).
- Easy to understand and implement.
- Works well when clusters are compact and spherical.

❖ **Limitations**

- Must specify K in advance.
- Sensitive to initialization (different runs may give different results).
- Struggles with non-spherical clusters or varying cluster sizes.
- Sensitive to outliers.

6. Applications of K-Means

- **Business**: Customer segmentation.
- **Computer Vision**: Image compression (color quantization).
- **Healthcare**: Grouping patients by symptoms.
- **NLP**: Document clustering.
- **Anomaly Detection**: Points far from centroids may be anomalies.

7. Summary

- K-Means is a partition-based clustering algorithm.
- Works by iteratively assigning points and updating centroids.
- Minimizes WCSS to create compact clusters.
- Requires careful choice of KKK and is sensitive to initialization.
- Widely used in practice due to simplicity and efficiency.

➢ TD

TD1: Theory

1. Explain in your own words how K-Means works.
2. Why do we need to choose KKK before running the algorithm?
3. Compare **Elbow Method** and **Silhouette Score** for choosing KKK.
4. What are the main limitations of K-Means?

TD2: Manual Calculation

Given the points:

A(1,1) , B(2,1) , C(4,3) , D(5,4)

- Apply **K-Means with K=2** by hand (first iteration only).
- Step 1: Assume AAA and DDD are initial centroids.
- Step 2: Assign each point to the closest centroid.
- Step 3: Recompute centroids.

TD3: Conceptual

- For each case, explain whether K-Means is suitable or not:
    1. Detecting irregular-shaped clusters (e.g., concentric circles).
    2. Segmenting customers by age and income.
    3. Anomaly detection in bank transactions.

➢ TP

Apply K-Means

Code Example: Basic K-Means with Scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Load dataset
iris = load_iris()
X = iris.data[:, :2]   # use only first 2 features for visualization
# Apply KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
# Predictions
y_kmeans = kmeans.predict(X)
# Plot clusters
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0],
        kmeans.cluster_centers_[:, 1],
        s=200, c='red', marker='X', label='Centroids')
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.title("K-Means Clustering (Iris Dataset)")
plt.legend()
plt.show()
```

TP1: Dataset Exploration

- Load the **Iris dataset (without labels)**.
- Plot the data using the first two features.
- Discuss how many clusters you expect.

TP2: Apply K-Means

- Apply **K-Means clustering with K=3** .
- Visualize the resulting clusters.
- Compare cluster assignments with true labels (just to evaluate, not for training).

TP3: Choosing K

- Apply K-Means for K=2,3,4,5K = 2, 3, 4, 5K=2,3,4,5.
- Plot the **Elbow curve (WCSS vs. K)**.
- Compute the **Silhouette Score** for each KKK.
- Discuss the best value of KKK.

TP4: Beyond Basics

- Try different initializations (random, k-means++).
- Introduce outliers into the dataset → observe their effect.
- Compare K-Means results with another clustering method (e.g., Hierarchical).

Chapter 3 : K-Medoids (PAM)

➢ Cours

1. Introduction

K-Medoids (also called **Partitioning Around Medoids, PAM**) is a clustering algorithm closely related to K-Means.
While K-Means represents each cluster by its **centroid (average point)**, K-Medoids represents clusters using **real data points (medoids)**.

- **Medoid**: The most centrally located data point within a cluster (minimizes the sum of distances to all other points in the cluster).
- K-Medoids is more **robust to noise and outliers** than K-Means, because medoids are always actual data points.

2. Intuition

- In K-Means, a cluster center may be an artificial point not present in the dataset.
- In K-Medoids, each cluster is represented by an existing point (the medoid).
- Example: In customer segmentation, instead of having an "average customer" as the centroid, we identify an actual customer who best represents the cluster.

3. The K-Medoids Algorithm (PAM)

Steps

1. **Initialization**
   Select K random data points as medoids.
2. **Assignment Step**
   Assign each data point to the nearest medoid based on a distance metric (e.g., Euclidean, Manhattan).
3. **Update Step**
   For each cluster:
   - Try swapping the medoid with a non-medoid point.
   - If the swap reduces the total cost, update the medoid.
4. **Repeat** until medoids no longer change (convergence).

Cost Function

K-Medoids minimizes the **total dissimilarity**:

$$\mathbf{J = \sum_{j=1}^{k} \sum_{x_i \in S_j} d\,(\,xi - mj\,)^2}$$

where:

- **Sj :** set of points in cluster j
- **mj :** medoid of cluster j
- **d(·) :** distance metric

4. Differences Between K-Means and K-Medoids

Feature	***K-Means***	***K-Medoids***
Cluster center	Centroid (average point)	Medoid (actual data point)
Robustness to outliers	Low	High
Distance metric	Usually Euclidean	Flexible (Euclidean, Manhattan, etc.)
Computational cost	Lower (faster)	Higher (slower, due to swaps)

5. Advantages and Limitations

❖ **Advantages**

- More robust to outliers and noise.
- Works with any distance metric (not just Euclidean).
- Produces interpretable clusters since medoids are actual data points.

❖ **Limitations**

- Computationally expensive for large datasets.

- Less scalable than K-Means.
- Like K-Means, requires K to be specified in advance.

6. Applications

- **Market segmentation**: Identifying representative customers.
- **Healthcare**: Grouping patients with similar conditions.
- **Computer vision**: Robust clustering of noisy image data.
- **Text mining**: Document clustering using similarity measures.

7. Summary

- K-Medoids is a clustering algorithm similar to K-Means but uses **medoids instead of centroids**.
- More **robust to noise/outliers**, but **computationally more expensive**.
- Useful when interpretability and robustness are more important than speed.

➢ TD

TD1: Theory

1. Explain the main difference between K-Means and K-Medoids.
2. Why is K-Medoids more robust to outliers?
3. In what case would you prefer K-Medoids over K-Means?

TD2: Manual Example

Consider the points:

A(1,1) , B(2,2) , C(3,3) , D(8,8) , E(9,9)

- Apply K-Medoids with K=2K=2K=2.
- Step 1: Choose AAA and EEE as initial medoids.
- Step 2: Assign points to nearest medoid.
- Step 3: Check if swapping medoids reduces cost.

TD3: Conceptual

For each scenario, explain if K-Medoids is suitable:

1. Clustering GPS coordinates of taxi pickup locations.
2. Clustering bank transactions with many outliers.
3. Clustering millions of high-dimensional images.

➢ TP

TP1: Dataset Preparation

- Use the **Iris dataset (without labels)**.
- Standardize the features.

TP2: Apply K-Medoids

- Use a clustering library (e.g., scikit-learn-extra or pyclustering).
- Run K-Medoids with K=3K=3K=3.
- Visualize clusters in 2D (using PCA for dimensionality reduction).

TP3: Compare with K-Means

- Apply K-Means with the same dataset.
- Compare cluster quality using **Silhouette Score**.
- Discuss differences in robustness to outliers.

TP4: Real Data Experiment

- Load a dataset with outliers (e.g., a customer dataset with abnormal values).
- Apply both K-Means and K-Medoids.
- Show how outliers affect results differently.

Chapter 4: Hierarchical Clustering

➢ Cours

1. Introduction

Hierarchical Clustering builds a hierarchy of clusters, visualized through a **dendrogram** (tree structure).
Instead of fixing the number of clusters in advance (like in K-Means), hierarchical methods let us **explore multiple levels of granularity**.

Think of it like organizing files on your computer:

- First, you group them by general topics (e.g., "School", "Work", "Photos").
- Inside "School", you further divide by "Math", "History", "Biology".
- Inside "Math", you might have "Assignments", "Exams", "Notes".

This **nested grouping** is exactly what hierarchical clustering produces.

2. Types of Hierarchical Clustering

There are **two main approaches**:

A. Agglomerative (AGNES – Agglomerative Nesting)

- **Bottom-up approach**.
- Start with each point as its own cluster.
- At each step, **merge the two closest clusters**.
- Continue until all points belong to one single cluster.
- Most widely used in practice.

❖ Imagine students in a classroom forming pairs, then small groups, then merging into bigger groups until the whole class is together.

B. Divisive (DIANA – Divisive Analysis)

- **Top-down approach**.
- Start with **all data in one big cluster**.
- Recursively **split clusters** into smaller groups.
- Continue until each observation is in its own cluster.

❖ Imagine starting with one big committee of people, then dividing it into sub-committees, then further splitting them into teams.

3. Distance Metrics Between Clusters

When merging or splitting, we need a **rule to measure similarity** between clusters:

- **Single Linkage (Minimum Distance)**

$$d(A, B) = \min_{x \in A, y \in B} d(x, y)$$

Tends to create long "chains" of clusters.

- **Complete Linkage (Maximum Distance)**

$$d(A, B) = \max_{x \in A, y \in B} d(x, y)$$

Produces compact, spherical clusters.

- **Average Linkage**

$$J = \frac{1}{|A||B|} \sum_{x \in A} \sum_{y \in B} d(x, y)$$

Balances between single and complete.

- **Ward's Method**
  Merges clusters that **minimize variance increase**.
  Often gives the best practical results.

4. Algorithm Steps (AGNES Example)

1. Start: each point = its own cluster.
2. Compute pairwise distances between all clusters.
3. Merge the two closest clusters.
4. Update the distance matrix using chosen linkage.
5. Repeat until only one cluster remains.
6. Cut the dendrogram at desired level to form k clusters.

5. Dendrogram

- A **tree diagram** that shows how clusters are merged/split.
- The **height** of a node = distance (or dissimilarity) at which merge occurred.
- By **cutting horizontally** at a chosen height, you decide the number of clusters.

❖ Example: Cutting higher gives fewer (larger) clusters. Cutting lower gives more (smaller) clusters.

6. Advantages and Limitations

❖ **Advantages**

- No need to predefine number of clusters.
- Dendrogram = interpretable and intuitive visualization.
- Captures nested structures.
- Flexible with different distance metrics.

❖ **Limitations**

- High computational cost (O(n2)O(n^2)O(n2) or more). Not suitable for very large datasets.
- Sensitive to noisy data and choice of distance measure.
- Once a merge/split is made, it cannot be undone (greedy).

7. Applications

- **Bioinformatics:** Grouping genes/proteins with similar behavior.
- **Marketing:** Customer segmentation by purchasing behavior.
- **Document analysis:** Organizing news/articles by topics.
- **Social sciences:** Building hierarchies of relationships between people.

8. Summary

- **Hierarchical clustering** organizes data into a **tree structure**.
- Two approaches:
  - **Agglomerative (AGNES)** → bottom-up (most common).
  - **Divisive (DIANA)** → top-down (less common).
- **Dendrograms** help visualize and choose number of clusters.
- Flexible but computationally expensive.

➢ TD

TD1: Conceptual Questions

1. Differentiate between AGNES and DIANA. Which is more widely used and why?
2. Compare Single Linkage and Complete Linkage in terms of cluster shapes.
3. Explain why Ward's method often gives better results.

TD2: Manual Example

Given points on a line:

A (1) , B (2) , C (6) , D (10)

1. Apply **AGNES** with **Single Linkage** step by step.
2. Draw the dendrogram.
3. If we cut at 2 clusters, which groups do we obtain?

TD3: Discussion

Why is hierarchical clustering often combined with **K-Means** (e.g., using hierarchical to choose k, then running K-Means)?

➢ TP

TP1: Using Iris Dataset

- Load **Iris dataset** without labels.
- Standardize features.
- Apply AgglomerativeClustering (with single, complete, average, ward).
- Compare number and quality of clusters.

TP2: Visualizing a Dendrogram

- Use scipy.cluster.hierarchy.dendrogram.
- Plot the dendrogram for Iris dataset.
- Show effect of cutting at different heights.

TP3: Comparing AGNES and DIANA

- Implement AGNES using AgglomerativeClustering.
- Implement DIANA manually (start with all points, split recursively).
- Compare their dendrograms.

TP4: Evaluation

- Use **Silhouette Score** to evaluate clusters at different cut levels.
- Compare with K-Means and K-Medoids.

TP5: Real-World Example

- Take a dataset of **text documents** (e.g., news articles).
- Use **TF-IDF + cosine similarity** as distance metric.
- Apply hierarchical clustering and visualize the dendrogram.
- Interpret the clusters (e.g., politics, sports, technology).

Chapter 5 : DBSCAN (Density-Based Clustering)

➢ Cours

1. Introduction

While **K-Means** and **Hierarchical Clustering** are powerful, they have some **limitations**:

- They require the number of clusters k to be known in advance.
- They assume spherical, equally sized clusters.
- They are sensitive to outliers.

DBSCAN solves many of these problems.
It is a **density-based clustering algorithm** that can:

- Discover clusters of **arbitrary shapes** (not just circles).
- Identify **noise points (outliers)**.
- Work without specifying the number of clusters.

❖ DBSCAN is widely used in spatial data (geography, image processing) and anomaly detection.

2. Key Concepts

DBSCAN relies on **two parameters**:

1. **ε (Epsilon)** – neighborhood radius.
   Defines how close points need to be to be considered neighbors.
2. **MinPts (Minimum Points)** – minimum number of neighbors required to form a cluster.
   Defines the "density" threshold.

Types of Points

- **Core Point**:
  Has at least **MinPts** neighbors within distance ε.
  Part of a dense region.
- **Border Point**:
  Has fewer than MinPts neighbors, but lies within ε of a **core point**.
  Attached to a cluster but not dense enough to be a core.

- **Noise Point (Outlier)**:
  Neither core nor border.
  Lies in sparse regions.

3. Algorithm Steps

1. Select a random point p.
2. If p is a **core point** (≥ MinPts within ε):
    - Form a new cluster.
    - Add all density-reachable points (neighbors within ε, recursively).
3. If p is a **border point**, assign it to a nearby cluster.
4. If p is noise, mark it as outlier.
5. Repeat until all points are visited.

4. Example (Intuition)

Imagine points scattered in a 2D plane:

- If many points are close together → cluster.
- If a point is isolated → noise.

❖ Unlike K-Means, DBSCAN can find clusters shaped like moons, spirals, or irregular blobs.

5. Choosing Parameters

- **Epsilon (ε):**
    - Too small → many small clusters + noise.
    - Too large → few big clusters, losing structure.
- **MinPts:**
    - Common rule: at least MinPts=2×dimension\text{MinPts} = 2 \times \text{dimension}MinPts=2×dimension.
    - Example: For 2D data → MinPts = 4.

❖ **K-distance graph method**:

- Plot distance to the k-th nearest neighbor for each point.
- Look for the "elbow" → good ε choice.

6. Advantages and Limitations

❖ **Advantages**

- No need to specify number of clusters.
- Detects clusters of arbitrary shape.
- Robust to outliers.

❖ **Limitations**

- Struggles with clusters of **varying densities**.
- Sensitive to choice of ε and MinPts.
- Computationally expensive for large datasets.

7. Applications

- **Geographic data**: Detecting densely populated cities.
- **Astronomy**: Finding star clusters.
- **Image processing**: Segmenting objects.
- **Anomaly detection**: Fraud detection, unusual patterns.

8. Summary

- DBSCAN groups points based on **density**.
- Requires two parameters: **ε** and **MinPts**.
- Identifies **core, border, and noise points**.
- Powerful for **arbitrary shapes** and **outlier detection**.
- Struggles with varying densities.

➢ TD

TD1: Conceptual Questions

1. Explain the difference between **core**, **border**, and **noise** points.
2. Why doesn't DBSCAN require the number of clusters in advance?
3. What happens if ε is too large? Too small?

TD2: Small Dataset Example

Consider points on a line:

$$X = \{ 1 , 2 , 2.1 , 2.2 , 8 , 8.1 , 8.2 , 20 \}$$

With parameters: ε = 0.5, MinPts = 2:

1. Identify core, border, and noise points.
2. Form the clusters step by step.

TD3: Discussion

Why is DBSCAN preferred over K-Means when dealing with clusters shaped like **half-moons** or **spirals**?

➢ TP

TP1: Applying DBSCAN on Synthetic Data

- Generate 2D blobs and moons (make_moons, make_circles from sklearn).
- Apply DBSCAN with different ε and MinPts.
- Visualize clusters and noise.

TP2: Iris Dataset

- Apply DBSCAN on the Iris dataset.
- Compare with K-Means and Hierarchical clustering.
- Evaluate with **Silhouette Score**.

TP3: Choosing Parameters

- Plot the **k-distance graph** for Iris dataset.
- Find the "elbow" to select ε.
- Show how different ε values change clustering.

TP4: Outlier Detection

- Add artificial outliers to dataset.
- Show how DBSCAN identifies them as noise.

TP5: Real-World Dataset

- Load a dataset (e.g., GPS coordinates of taxis in a city).
- Apply DBSCAN to find hotspots (e.g., busiest pickup locations).
- Interpret results.

Summary: Clustering in Unsupervised Learning

Clustering is one of the core techniques in **unsupervised learning**, where the task is to organize data into groups (clusters) without using predefined labels. The idea is simple: data points that look alike should belong to the same group, while points that are very different should end up in different clusters.

In other words, a good clustering achieves two things:

- **Similarity inside the cluster** = members of the same group are close to each other.
- **Dissimilarity between clusters** = groups are clearly separated from one another.

Why Clustering Matters

Unlike supervised learning, where the algorithm learns from labeled data, clustering helps us **discover hidden patterns** in completely unlabeled datasets. It is often used as the first step in exploring a new dataset: segmenting customers, identifying patterns in social networks, or even finding anomalies.

Different Families of Clustering Methods

Clustering is not a "one-size-fits-all" task. Different algorithms use different strategies:

1. **Partitioning Methods** – Divide data into a fixed number of clusters.
   Examples: K-Means, K-Medoids.
2. **Hierarchical Methods** – Build a hierarchy of clusters, either bottom-up (agglomerative) or top-down (divisive).
   Examples: AGNES, DIANA.
3. **Density-Based Methods** – Identify clusters as dense regions separated by sparse areas; good at detecting outliers.
   Examples: DBSCAN, OPTICS.

How Do We Judge a Clustering?

Because clustering works without labels, evaluation is not always straightforward. We usually rely on:

- **Internal measures** – how well clusters are formed (e.g., Silhouette Score, Davies-Bouldin Index).
- **External measures** – if we do have labels for comparison, we can check how well clusters match reality (e.g., Purity, Adjusted Rand Index).

Where Is Clustering Used?

Clustering has an incredibly wide range of applications:

- Businesses use it for **market segmentation** (grouping customers with similar behaviors).
- In computer vision, it helps with **image segmentation**.
- In cybersecurity, it supports **anomaly detection** (spotting unusual activity).
- In text mining, it allows us to discover **topics** hidden in large collections of documents.

Strengths and Limitations

❖ **Strengths**

- Finds hidden patterns without supervision.
- Works with many types of data.
- Can be used as a preprocessing step before other machine learning tasks.

❖ **Limitations**

- Sensitive to parameter choices (e.g., the number of clusters in K-Means).
- Results can vary a lot depending on the algorithm.
- High-dimensional data can make clustering difficult.
- No universal "right answer" — what counts as a good cluster often depends on the application.

Final Thoughts

Clustering is less about “getting the one correct answer” and more about **discovering structure** in data. It is a powerful tool for exploration and knowledge discovery, but it requires careful choice of algorithms, parameters, and evaluation methods.

By combining different clustering methods and validation techniques, we can gain deep insights into data that would otherwise remain hidden.

Real-Life Applications

- **K-Means**: Market segmentation, grouping users by behavior
- **K-Medoids**: Clustering of transaction data, gene sequences
- **Hierarchical**: Biological taxonomies, document clustering
- **DBSCAN**: Geospatial data, fraud detection

Part IV: Advanced Concepts & High-Performance Data Mining

Chapter 1 : Introduction to Parallel Computing

1. Introduction

As the size of data grows and computational problems become more complex, traditional **sequential computing** (executing one task at a time on a single processor) is no longer sufficient. To solve modern scientific, engineering, and data-intensive problems efficiently, we need to exploit the power of **parallel computing**.

Parallel computing is the practice of dividing a large problem into smaller tasks, distributing these tasks across multiple processors, and executing them **simultaneously**. The ultimate goal is to achieve **faster computation** and to handle **bigger, more complex problems** than what a single processor can manage.

2. What is Parallel Computing?

- **Sequential computing:** one processor executes one instruction at a time.
- **Parallel computing:** multiple processors execute multiple instructions at the same time, working together to solve a problem.

❖ In simple terms: *Instead of one person doing all the work, several people share the work and finish faster.*

3. Why Parallel Computing?

1. **Performance and Speed**
    - Large computations can be completed in less time.
2. **Scalability**
    - Parallel systems can scale from a few cores in a laptop to thousands of processors in supercomputers.
3. **Handling Big Data**
    - Necessary for modern applications like machine learning, weather forecasting, and genome sequencing.
4. **Energy Efficiency**

- Many small processors running in parallel can sometimes be more energy-efficient than a single powerful processor.

4. Types of Parallelism

Parallelism can be categorized at different levels:

4.1. Data Parallelism

- The same operation is applied simultaneously to different chunks of data.
- Example: processing multiple images at once.

4.2. Task Parallelism

- Different tasks are executed in parallel, possibly on different data.
- Example: one processor sorts numbers while another processor computes statistics.

4.3. Pipeline Parallelism

- Tasks are divided into stages, where each stage is processed in sequence but by different processors at the same time.
- Example: assembly line in a factory, where each worker (processor) performs one step.

5. Parallel Architectures

- **Shared Memory Systems**: All processors access the same memory.
- **Distributed Memory Systems**: Each processor has its own local memory; processors communicate by passing messages (→ this is where MPI comes in).
- **Hybrid Systems**: Combine both approaches.

6. Challenges in Parallel Computing

- **Communication Overhead**: Time spent coordinating between processors.
- **Load Balancing**: Ensuring all processors have roughly the same amount of work.
- **Data Dependencies**: Some tasks cannot proceed until others are finished.
- **Debugging Complexity**: Parallel programs are harder to design and test.

7. Real-Life Applications

- **Scientific Simulations** (climate models, physics experiments).
- **Big Data Analytics** (Hadoop, Spark).
- **Machine Learning** (training deep neural networks on GPUs).
- **Medical Imaging** (3D reconstruction, MRI analysis).
- **Finance** (risk modeling, fraud detection).

Summary of this Section

Parallel computing is a **fundamental paradigm** that allows us to solve large-scale problems more efficiently by dividing tasks across multiple processors. It underpins many modern technologies and is essential for handling today's massive datasets and computational challenges.

Chapter 2: Introduction to MPI (Message Passing Interface)

➢ Cours

1. Introduction to MPI

When working on **distributed memory systems**, each processor has its **own local memory**, and processors cannot directly access each other's data. To collaborate, they must exchange information explicitly through **messages**.

The **Message Passing Interface (MPI)** is the **de facto standard** for message-based communication in parallel computing. It provides a set of **functions and protocols** that allow multiple processes running on different processors (and possibly different machines) to communicate and synchronize efficiently.

2. What is MPI?

- **MPI = Message Passing Interface**
- A **library specification**, not a programming language.
- Provides standardized routines for:
    - **Point-to-point communication** (send & receive).
    - **Collective communication** (broadcast, scatter, gather, reduce).
    - **Synchronization** across processes.
- Implementations: **OpenMPI**, **MPICH**, Intel MPI, etc.

❖ MPI is widely used in **supercomputing, high-performance simulations, and scientific computing**.

3. How MPI Works

Each MPI program typically follows this structure:

1. **Initialization**
    - Start the MPI environment.
2. **Communication**
    - Processes exchange data through send/receive or collective operations.

3. **Computation**
    - Each process works on its portion of the data.
4. **Finalization**
    - Close the MPI environment.

MPI programs are usually written in **C, C++, or Fortran**, but bindings exist for Python (**mpi4py**) and other languages.

4. Basic MPI Operations

4.1. Initialization and Finalization

- MPI_Init: starts the MPI environment.
- MPI_Finalize: ends the MPI environment.

4.2. Process Information

- MPI_Comm_size: number of processes.
- MPI_Comm_rank: ID of each process (rank).

4.3. Point-to-Point Communication

- MPI_Send: send a message.
- MPI_Recv: receive a message.

4.4. Collective Communication

- MPI_Bcast: broadcast data from one process to all others.
- MPI_Scatter: split data across processes.
- MPI_Gather: collect data from all processes.
- MPI_Reduce: perform reduction (sum, max, min, average).

Advantages of MPI

- **Portability**: works on almost any distributed system.
- **Scalability**: runs efficiently on thousands of processors.
- **Flexibility**: fine-grained control over communication.
- **Efficiency**: optimized for high-performance computing (HPC).

Limitations of MPI

- **Complexity**: Writing MPI programs requires careful design.
- **Debugging Difficulty**: Errors in message passing are hard to trace.
- **Overhead**: Communication cost can dominate if not managed well.
- **No Shared Memory Abstraction**: Programmer must explicitly manage all communication.

5. Applications of MPI

- Weather and climate modeling.
- Computational fluid dynamics (CFD).
- Molecular dynamics simulations.
- Parallel graph processing.
- Training large AI models on distributed clusters.

➢ TD

TD1 – Understanding MPI Concepts

1. Explain the difference between **shared memory** and **distributed memory** systems. Why does MPI fit better in one than the other?
2. Define: MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Finalize.
3. Distinguish between **point-to-point communication** and **collective communication** with examples.

TD2 – Performance & Scalability

1. Suppose you run an MPI program on 4 processes. Each process handles 1/4 of the data but needs to exchange results at the end. Discuss the communication overhead.
2. Compare MPI with OpenMP in terms of memory usage and scalability.

➢ TP

TP1 – Getting Started with MPI

- Install **OpenMPI** or use a cluster environment.
- Write a simple MPI program where:
    - Each process prints: "Hello from process X of N".
    - Run with mpirun -np 4 ./program and observe the output.

TP2 – Data Distribution with Scatter & Gather

- Create an array of numbers.
- Use MPI_Scatter to distribute chunks of the array to each process.
- Each process computes the sum of its chunk.
- Use MPI_Gather to collect partial sums in the root process.
- Verify the total sum matches the sequential computation.

TP3 – Parallel Reduction

- Use MPI_Reduce to compute the **global maximum** and **global sum** across all processes.
- Compare execution time with sequential computation for large datasets.

TP4 – Custom Experiment

- Implement **matrix-vector multiplication** using MPI:
    - Distribute rows of the matrix across processes.
    - Perform multiplication locally.
    - Gather results in the root process.

❖ Summary of this Section

MPI is the **standard tool for distributed parallel programming**, allowing processes to communicate via messages. It is powerful, scalable, and widely used in high-performance computing but requires careful design to avoid communication bottlenecks.

Challenge

Part I: Foundational Principles of Data Mining

- **Chapter 1: Introduction to Data Mining**
    - **Challenge:** Instead of just defining the KDD process, you are tasked with analyzing a real-world business case. You must propose a complete KDD pipeline for a retail company that wants to increase online sales. For each stage (Selection, Preprocessing, etc.), describe the specific actions you would take, the type of data involved, and the business question you aim to answer at that stage.
- **Chapter 2: Data Preprocessing**
    - **Challenge:** You are given a raw dataset with mixed data types, missing values, and outliers. Your task is to apply a series of preprocessing steps and justify each choice. Use both the Z-score and IQR methods to detect and handle outliers, and compare their effectiveness on a specific feature. Afterward, explain which scaling method (Normalization vs. Standardization) is best suited for a K-Means clustering algorithm and why.
- **Chapter 3: Distance and Similarity Measures**
    - **Challenge:** You have a dataset of customer reviews. You need to perform two different analyses. First, use an appropriate distance metric to find the 5 most similar customers to a target customer. Second, use a different similarity measure to find the 5 most similar reviews to a specific product review. Justify your choice of metric for each task, explaining why one is a better fit than the other in a textual vs. numerical context.

Part II: Supervised Learning Techniques

- **Chapter 4: K-Nearest Neighbors (KNN)**
    - **Challenge:** You must implement a custom KNN classifier from scratch without using a pre-built library like scikit-learn. Your implementation must include the ability to switch between Euclidean and Manhattan distance. After building the model, you must evaluate its performance by plotting the accuracy curve as a function of the *k* value, then analyze and explain the trade-off between bias and variance that you observe.
- **Chapter 5: Decision Trees**
    - **Challenge:** Build a decision tree on a dataset and intentionally overfit the model. Then, apply both pre-pruning and post-pruning techniques to combat this overfitting. Visualize all three trees (overfitted, pre-pruned, and post-pruned) and provide a detailed analysis of how each method affects the tree's complexity, interpretability, and generalization performance on a holdout test set.

- **Chapter 6: Naive Bayes**
    - **Challenge:** You are given a text classification problem. Your task is to build a Naive Bayes model using one-hot encoded features and then a second model using TF-IDF vectors. Compare the performance (accuracy, precision, recall) of both models. Explain the theoretical reasons why one model might outperform the other, particularly in a sparse data environment.
- **Chapter 7: Logistic Regression**
    - **Challenge:** You are asked to build a multiclass classifier for a dataset with three classes. Implement both a One-vs-Rest (OvR) and a Softmax classifier from scratch (without using a library for the core algorithm). Compare their results, and discuss the computational and conceptual differences between the two approaches.
- **Chapter 8: Support Vector Machines (SVM)**
    - **Challenge:** You are given a dataset that is not linearly separable (e.g., concentric circles). You must first try to classify it with a linear SVM and then demonstrate its failure. Then, apply the kernel trick using the RBF kernel. Plot the decision boundary for both models. Finally, tune the hyperparameters *C* and *gamma* and explain how different values affect the margin and prevent overfitting or underfitting.
- **Chapter 9: Introduction to Neural Networks**
    - **Challenge:** Manually implement a single neuron with a Sigmoid activation function that can solve a simple binary classification problem (e.g., a single logical operation). Show the step-by-step calculations for forward propagation. Then, explain the role of a loss function and manually calculate the gradient for a single training instance, demonstrating the core principle of backpropagation.

Part III: Unsupervised Learning

- **Chapter 10: K-Means Clustering**
    - **Challenge:** Implement the K-Means algorithm from scratch. Your implementation must include the K-means++ initialization strategy. After implementing it, use the Elbow Method and Silhouette Analysis to determine the optimal number of clusters for a given dataset. Explain the limitations of these methods and propose an alternative strategy for a scenario where the data has no clear "elbow."
- **Chapter 11: K-Medoids (PAM)**
    - **Challenge:** Implement the K-Medoids (PAM) algorithm from scratch. Using a dataset with known outliers, compare the clustering results of your K-Medoids implementation with K-Means. Provide a quantitative analysis (e.g., using a silhouette score) and a qualitative one, explaining why K-Medoids is more robust to outliers than K-Means.
- **Chapter 12: Hierarchical Clustering**

- **Challenge:** Given a complex, non-spherical dataset, you must apply both single-linkage and complete-linkage hierarchical clustering. Plot the dendrograms for both methods. Analyze the visual differences in the resulting cluster structures and explain the theoretical reason behind these differences. Justify why one method might be a better fit for this specific dataset.

- **Chapter 13: DBSCAN and Density-Based Clustering**
  - **Challenge:** Using a dataset that contains clusters of different shapes and a significant amount of noise, apply DBSCAN. You must first use a method to programmatically determine the optimal epsilon and minPts parameters. Compare the resulting clusters with K-Means and Hierarchical Clustering, explaining why DBSCAN is uniquely suited for this type of data and how it successfully handles the noise points that would confuse other algorithms.

Part IV: Advanced Concepts & High-Performance Data Mining

- **Chapter 14: Introduction to Parallel Computing**
  - **Challenge:** You are given a large matrix and a vector. You need to implement a parallel matrix-vector multiplication using MPI. Your implementation should demonstrate the use of MPI_Scatter to distribute the matrix rows and MPI_Gather to collect the results. Compare the execution time of your parallel code with a sequential Python implementation and calculate the speedup and efficiency.

- **Chapter 15: Introduction to MPI (Message Passing Interface)**
  - **Challenge:** Implement a parallel word count program on a large text file using MPI. This task requires you to: 1) Distribute chunks of the file to different processes. 2) Have each process count the words in its chunk. 3) Use MPI_Reduce to aggregate the local counts into a final global word count on the root process. Analyze the communication overhead in your implementation.

- **Chapter 16: Parallelizing Other Clustering Algorithms**
  - **Challenge:** Design and pseudo-code a parallel version of the AGNES (Agglomerative Nesting) algorithm. Explain how you would handle the most significant challenge: the $O(n^2)$ pairwise distance matrix computation. Describe a strategy for distributing this computation and minimizing communication bottlenecks, explaining why a simple master-worker model might be inefficient for this specific algorithm.

Useful URLs

This section provides curated resources — official documentation, online books, tutorials, and communities — that readers can use to deepen their understanding of **Machine Learning, Parallel Computing, and MPI/OpenMP**.

- ❖ **Github Repo for code TP examples**
  - Github repo : **https://github.com/adelazzi/data-mining-codes-for-book**
- ❖ **General Machine Learning & AI**
  - Scikit-learn: https://scikit-learn.org/stable/
  - TensorFlow: https://www.tensorflow.org/
  - PyTorch: https://pytorch.org/
  - Google ML Crash Course: https://developers.google.com/machine-learning/crash-course
  - Kaggle: https://www.kaggle.com/
- ❖ **Online Books**
  - Elements of Statistical Learning (ESL): https://hastie.su.domains/ElemStatLearn/
  - Deep Learning Book (Goodfellow): https://www.deeplearningbook.org/
  - Pattern Recognition and Machine Learning (Bishop): https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/
  - Hands-On Machine Learning (O'Reilly): https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/
- ❖ **Clustering & Unsupervised Learning**
  - Scikit-learn Clustering Overview: https://scikit-learn.org/stable/modules/clustering.html
  - Hierarchical Clustering Tutorial: https://www.geeksforgeeks.org/hierarchical-clustering-in-machine-learning/
  - Visualizing K-Means: https://www.naftaliharris.com/blog/visualizing-k-means-clustering/
- ❖ **Parallel Computing & HPC**

- MPI Forum: https://www.mpi-forum.org/
- OpenMPI: https://www.open-mpi.org/
- MPICH: https://www.mpich.org/
- mpi4py (MPI for Python): https://mpi4py.readthedocs.io/en/stable/
- OpenMP Official: https://www.openmp.org/
- NVIDIA CUDA: https://developer.nvidia.com/cuda-zone

❖ **Online Courses**

- Machine Learning by Andrew Ng (Coursera): https://www.coursera.org/learn/machine-learning
- Deep Learning Specialization (Coursera): https://www.coursera.org/specializations/deep-learning
- Parallel Programming with MPI (Coursera): https://www.coursera.org/learn/parallel-programming-in-mpi
- HPC Carpentry: https://hpc-carpentry.github.io/

❖ **Communities & Forums**

- Stack Overflow: https://stackoverflow.com/
- Reddit MachineLearning: https://www.reddit.com/r/MachineLearning/
- ResearchGate: https://www.researchgate.net/
- HPC Wiki: https://hpc-wiki.info/