الجمـــهورية الجزائريـــة الديمقراطيـــة الشعبيـــة
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH
UNIVERSITY OF 8th MAY, 1945
GUELMA
Faculty of Mathematics,
Computer Science and Material Sciences
Computer Science department

وزارة التعليم العالـي
و البحث العلمـي
جامعة 8 مـاي 1945
قالمـة
كليـة الرياضيـات، الإعـلام الآلـي
و علـوم المـادة
قسم: الإعلام الآلي

Course Handout

Data Mining

Intended for second-year students in the Professional Bachelor's program in Networks and Web Technologies.

Established by:

Dr. Aicha AGGOUNE

SYLLABUS

Teaching unit: UEF 4-1

Module: *Data mining*

Domain: Computer science, 2nd year Professional Bachelor « Network and Web Technologies »

Semester: 4 **College year**: 2024/2025

Credit: 3 **Coefficient**: 3

Total Weekly Hourly Volume: 4 H30 Hours

- Course (*Number of hours per week*): 1H30
- Practical work (*Number of hours per week*): 3H00

Education language: ENGLISH

Teacher responsible for the module: Dr. Aicha AGGOUNE, **Grade**: MCA

Office : No.02 (Teachers' room)

Email: aggoune.aicha@univ-guelma.dz

Consultation periods: Sunday 09H:00-11H:00

Description

Data mining is a process of extracting, from a large amount of data, interesting patterns that are non-trivial, hidden, new and potentially useful. It is a rapidly growing field and is becoming important because with the increasing quantity and variety of online data collections by many organizations and commercial enterprises, there is a high potential value of patterns discovered in those collections.

The overall aim of this module is to introduce students to modern data mining techniques and their use in business and other areas of applications.

- In particular, the module explores basic concepts, principles and techniques of data mining with emphasis on both the technical and the practical issues.
- The module provides students with an understanding in evaluating and comparing data mining solutions for effective use of the solutions in practice.
- The module also equips students with some hands-on experience and skills in conducting a data mining project using a data mining software tool.

Objectives

On successfully completing the module students will be able to:

1. Understand the motivation for data mining in the context of business and information technology
2. Know how data mining is used, such as marketing, sales and customer relationship management
3. Understand the concepts and main techniques in data mining

4. Be able to describe the differences between the major data mining tasks
5. Have an understanding of the knowledge discovery process
6. Understand the purpose of the main tasks involved in data preparation for mining
7. Gain hands-on experience in using a state-of-the-art data mining tool.

Pre-requisites: Introduction to Programming, Databases, basic probability and statistics

Course content

This course consists of five chapters:

Chapter 1: Introduction to Data mining

1. Data, information and knowledge
2. Definition of data mining
3. Types of data mining
4. Data mining tasks
5. Data mining process
6. Data mining applications
7. Data mining tools

Chapter 2: Exploratory data analysis and Data preprocessing

1. Types of variables
2. Techniques of Exploratory Data Analysis
3. Data cleaning
4. Data transformation
5. Data integration
6. Data reduction

Chapter 3: Association rules mining

1. Introduction to ARM
2. Basic concepts
3. Methods for finding frequent itemsets
4. Rule evaluation metrics
5. Applications

Chapter 4: Classification

1. Introduction to Machine learning
2. Basic concepts
3. Process of classification
4. Classification techniques
5. Evaluation and comparison of classifiers.

Chapter 5: Clustering

1. Introduction
2. Partitioning methods
3. Hierarchical methods

4. Density-based methods
5. Grid-based methods
6. Model-based methods

Assessment method

Assessment Instruments	Weight (%)
Practical work	50%
Project	50%
Total	100 %

References

- Aggarwal, C. C. (2015). Data mining: the textbook (Vol. 1, No. 3). New York: springer.
- Berry, M., and Linoff, G. (2012). Data Mining Techniques: For Marketing, Sales and Customer Relationship Management.
- Nasser, F. K., & Behadili, S. F. (2022). A review of Data Mining and knowledge discovery approaches for bioinformatics. Iraqi Journal of Science, 3169-3188.
- Zong, R. Xia, and J. Zhang. Text Data Mining. Springer Singapore, 2021.
- Lu, Y. J., Lee, W. C., & Wang, C. H. (2023). Using data mining technology to explore causes of inaccurate reliability data and suggestions for maintenance management. Journal of Loss Prevention in the Process Industries, 83, 105063.
- Durugkar, S. R., Raja, R., Nagwanshi, K. K., & Kumar, S. (2022). Introduction to data mining. Data Mining and Machine Learning Applications, 1-19.
- Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. SN computer science, 2(3), 160.

Preface

This course handout was prepared for the Data Mining module, intended for second-year students in the Professional Bachelor's Degree program in Networks and Web Technologies within the Department of Computer Science, Faculty of Mathematics, Computer Science, and Material Sciences, at the University of 8 May 1945, Guelma, Algeria.

The 'Data Mining' course is one of the modules in the Networks and Web Technologies (NWT) training program, offered within the framework of COFFEE (Co-construction d'une Offre de Formations à Finalité d'Employabilité Elevée), part of the Erasmus+ Capacity Building initiative. It is scheduled in the fourth semester of the NWT program, as part of the core teaching unit UEF4-1. The course carries a coefficient of 3, is worth 3 credits, and has a weekly workload of 4.30 hours, divided into two sessions: 1.30 hours of course and 3 hours of practical work (PW).

As data continues to grow in volume and complexity, the ability to extract meaningful insights has become an essential skill for professionals in the fields of networking and web technologies.

The primary goal of this course is to introduce students to the fundamental concepts, techniques, and tools of data mining. It covers both the theoretical foundations and practical applications, helping students develop a solid understanding of how data mining contributes to intelligent decision-making and predictive analysis in real-world scenarios.

These notes are structured to provide a clear and concise learning path, supported by illustrative examples. Whether students aim to pursue further studies in data science or apply data mining techniques in professional settings, this course serves as a valuable foundation.

The content of this course is structured into five chapters, each focusing on a key aspect of data mining and concluded with practical work using the Python programming language, which aims to reinforce theoretical concepts through hands-on implementation. These practical sessions guide students in applying data mining techniques to real or simulated datasets

Chapter 1: Introduction to Data Mining
Covers the basic concepts including data, information, and knowledge; the definition and types of data mining; core tasks and processes; applications; and commonly used tools.

Chapter 2: Exploratory Data Analysis and Data Preprocessing

Introduces types of variables, techniques for exploratory analysis, and essential preprocessing steps such as data cleaning, transformation, integration, and reduction.

Chapter 3: Association Rules Mining

Explores the principles of association rule mining (ARM), methods for discovering frequent itemsets, rule evaluation metrics, and real-world applications.

Chapter 4: Classification

Focuses on supervised learning, outlining the classification process, key techniques, and how to evaluate and compare classifiers.

Chapter 5: Clustering

Presents unsupervised learning methods, including partitioning, hierarchical, density-based, grid-based, and model-based clustering techniques.

These chapters are designed to build a progressive understanding of the field, supporting students in developing both analytical thinking and practical skills in data mining. The material serves as a foundation for more advanced studies and professional applications in the field of data science, web analytics, and network intelligence.

Author's note

These lecture notes for the Data Mining course were prepared by *Dr. Aicha AGGOUNE,* Associate Professor at the University of May 8, 1945, Guelma, and member of the LabSTIC research laboratory. Dr. Aicha AGGOUNE began teaching this course in the 2020/2021 academic year within the Department of Computer Science.

This material reflects several years of teaching experience in the domain of data analysis and is specifically tailored for second-year undergraduate students in the Networks and Web Technologies program. The course content aligns with the official curriculum and introduces students to the fundamental concepts, methods, and applications of data mining.

The lecture notes are organized into five chapters, each addressing a core aspect of data mining such as exploratory data analysis, association rule mining, classification, and clustering.

These notes aim to provide students with both a solid theoretical foundation and practical skills that are essential for working with data in today's digital landscape. They are intended not only for students but also for educators and professionals interested in acquiring hands-on experience in the field of data mining.

Dr. Aicha AGGOUNE

Table of Contents

CHAPTER 5. CLUSTERING

List of Figures

Chapter 1.

Introduction to Data mining

Introduction

The explosion in the amount of data stored, with the significant progress in processing speeds and digital storage has led users to accumulate more and more data. For example, companies need to enhance the value of the data they accumulate in their databases to best target their customers and meet their expectations.

Database management systems and information retrieval process are simply not enough anymore for decision-making. Confronted with huge collections of data, it is very important to set up techniques and tools for analyzing and exploring this quantity of data, for example, to identify customer profiles, discover relationships between customers, detect essential products and typical purchases, and predict future sales. The set of techniques allowing the user to extract of the essence of information stored is known as: Data Mining which is also referred to as knowledge discovery in databases (KDD).

1.1. Definition of data mining

Data mining is a multidisciplinary field drawing works from statistics, database technology, artificial intelligence, pattern recognition, machine learning, information theory, knowledge acquisition, information retrieval, high performance computing, and data visualization.

According to Marcel Holeshemier & Arno Siebe, "data mining is the search for relationships and global patterns that exist in large database but are ‘hidden’ among the vast amount of data, such as a relationship between patient data and their medical diagnosis. These relationships represent valuable knowledge about the database and the objects in the database and, if the database is a faithful mirror, of the real world registered by the database".

Data mining allows analysis of a large database to support decisions when traditional query languages are useless.

Data mining is a form of knowledge discovery essential for solving problems in a specific domain. There is confusion about the exact meaning of the terms “data mining” and knowledge discovery in databases “KDD.” Data mining emerged in the late 1980s, made great progress during the Information Age.

In 1990, the term "data mining" was not as widely known or established as it is today. The field of data mining was in its infancy, with researchers and practitioners exploring techniques to extract insights from data.

KDD was proposed in 1995 to describe the overall process of discovering useful and meaningful patterns, knowledge, and insights from large volumes of data. In this context, knowledge means relationships and patterns between data elements.

Data mining should be used exclusively for the discovery stage of the KDD process.

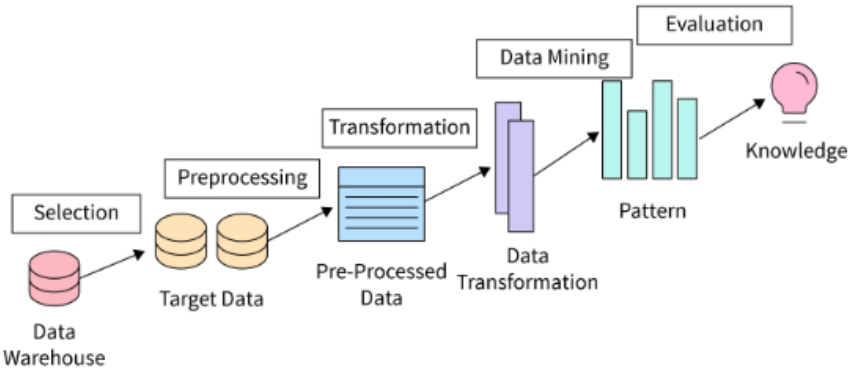


Figure 1.1. Process of KDD[1]

Data Selection: relevant data is selected from various sources for analysis. This involves identifying which data is relevant to the task at hand.

Data Preprocessing: the selected data frequently requires cleaning and transformation before it can be analyzed. This stage involves dealing with missing values, handling outliers, and normalizing data, among other tasks.

Data Transformation: Data is often transformed into a more suitable format or representation for analysis. This could involve aggregating data, creating new features, or converting data types.

Data Mining: This is where data mining techniques are used to identify patterns, relationships, correlations, classifications, and other information. These techniques are: Statistical techniques, Visualization, Decision trees, Association rules, etc.

Pattern Evaluation: The patterns or models generated by data mining algorithms are evaluated for their relevance and significance. This step involves assessing the quality and usefulness of the discovered knowledge.

Knowledge representation: to help stakeholders understand and make decisions, the discovered knowledges are represented in a more human-interpretable form by visualization techniques or summary reports.

The data to be mined should contain information worth mining: consistent, cleaned, and representative for the application. So, what kind of data can be mined? What is the data mining process?

1.2. Basic concepts

The digital world is rapidly expanding and increasing in complexity in terms of volume (terabytes to petabytes), variety (containing structured, unstructured and hybrid data) and velocity (with rapid growth). Data mining aims to efficiently extract form useful, task-oriented knowledge. Data collected for data mining process can indeed be stored in databases and other storage systems. Data come from simple numerical measurements and text

[1] https://www.scaler.com/topics/kdd-in-data-mining/

documents, to more complex information such as spatial data, multimedia channels, and hypertext documents.

1.2.1. Data, information and knowledge

Data: Data are any facts, text, or numbers without any context or interpretation and that can be processed by a computer. Data by itself does not convey any meaning. For example, a list of student grades. Metadata is data about the data itself such as dictionary definition. There are three types of data:

- Structured data: data organized and stored in a predefined format such as, tables in a relational database.
- Unstructured data: data does not have a predefined format or organization, such as document, text, images, videos. This type of data requires advanced research techniques like information retrieval and natural language processing.
- Semi-structured: data does not have a fixed format like structured data. This makes it more flexible, simpler to store, and easier to analyze than unstructured data. Example, NoSQL data, JSON document, XML, etc.

Information: Information is the result of processing and organizing data to give it context and meaning. For example, analysis the list of student grades can yield information on which students are serious.

Knowledge: Knowledge involves making connections between pieces of information, drawing conclusions, and deriving insights. It is the deeper understanding and insights derived from information and data. For example, if we analyze the list of student grades, we are building knowledge about the level of students or of the classroom against the previous years.

1.2.2. Data storage

Data mining is focused with uncovering hidden correlations in data to enable users to make decisions and forecasts for future usage. In most organizations, data we find really large databases and datasets. Data storage in organizations is often divided into two types: databases and data warehouses. Each of these modes has a distinct function and is intended to satisfy different data storage, management, and retrieval requirements.

1.2.2.1. Database

Databases are structured collections of data that enable for efficient storage (without duplication), retrieval, and modification. Databases are utilized for transactional and operational purposes, with data being regularly updated, added, or amended. Databases are geared for transaction processing and are built to manage real-time interactions. Hence, databases are based on OLTP database systems, which mean OnLine Transaction Processing to manage daily transactional operations of operational data, such as managing customer interactions, recording sales, etc.

Relational databases are one of the most widely used types of databases in various industries and applications for many decades due to their structured nature (tabular form) and ability to handle structured data efficiently using SQL (Structured Query Language).

Examples of relational database management systems include Oracle, MySQL, PostgreSQL, Microsoft SQL server, etc. Advanced databases are emerged such as, object-relational databases, object-oriented databases, multimedia databases, logical databases, distributed databases, etc.

In addition, with the increase of complexity and volume of data, these databases face challenges in handling very large datasets efficiently. An alternative of relational database, called NoSQL (Not Only SQL) to manage semi-structured data and provide high scalability (distribute data across multiple servers or clusters.) and flexibility (do not require a fixed schema). Types of NoSQL databases include key-value stores, document stores, column-family stores, and graph databases. Examples include MongoDB, Cassandra, Redis, and Neo4j.

1.2.2.2. Data warehouse

Data warehouse is enabled relational database systems designed to support very large amounts of historical data from various sources and are optimized for complex queries and data analysis. When relational databases are used for OLTP systems, data warehouses are based on OLAP for OnLine Analytical Processing, which are designed to support complex data analysis and reporting.

The data in data warehouse is organized around specific subjects and aggregated from multiple sources. Historical data is stored to support trend analysis and comparison over time. Thus, once data is loaded into the warehouse, it is not frequently updated, ensuring data integrity. Examples of data warehouse solutions include: IBM Netezza, Amazon Redshift, Google BigQuery, and Snowflake.

Data warehouses are often used for business intelligence and data mining purposes. Business intelligence (BI) includes technologies; processes used by enterprises for analyze data and make decisions. BI aims to turn raw data into actionable insights, helping organizations make informed choices to enhance their operations and strategic direction. Data warehouses allow analysts to access and analyze data from multiple databases and sources in a centralized and consistent manner.

In data warehousing, data is often structured in a multidimensional model, which can be represented as a cube. OLAP data cubes are valuable for complex analysis and business intelligence, offering a multidimensional view of data across factors like time, products, locations, and customers. Unlike simple data cubes, OLAP cubes enable in-depth insights. Moving forward, "data cube" refers to the OLAP model. For example, a sales data cube can present information across different dimensions, such as year, product category, locations, and customers.
A data mart is a subset of a data warehouse that focuses on specific business areas of the corporation. It is intended to make it simple to retrieve relevant and consolidated data for analytical and reporting needs.

1.2.3. Data scientist, data engineer and data analyst

Three roles collaborate closely within a data-centric team: Data engineers ensure that data is properly packaged and accessible, data scientists create knowledge from that data, and

data analysts assist in translating that knowledge into usable information for various stakeholders. The interaction of these roles is essential for organizations to make data-driven decisions.

Data Scientist: A data scientist is someone who analyzes and interprets large amounts of digital data. While there are multiple paths to becoming a data scientist, the most straightforward is to get sufficient experience and understand the various data scientist skills. These skills include extensive statistical analysis, machine learning, data conditioning, builds predictive models, and conducts advanced analyses to solve complex problems using data.

Data Engineer: A data engineer has the ability to create, develop, and maintain data pipelines, databases, and infrastructure to ensure data availability and accessibility.

Data Analyst: A data analyst understands data handling, modeling and reporting techniques along with a strong understanding of the business. It focuses on data interpretation, visualization, and basic statistical analysis to provide actionable insights for decision-makers.

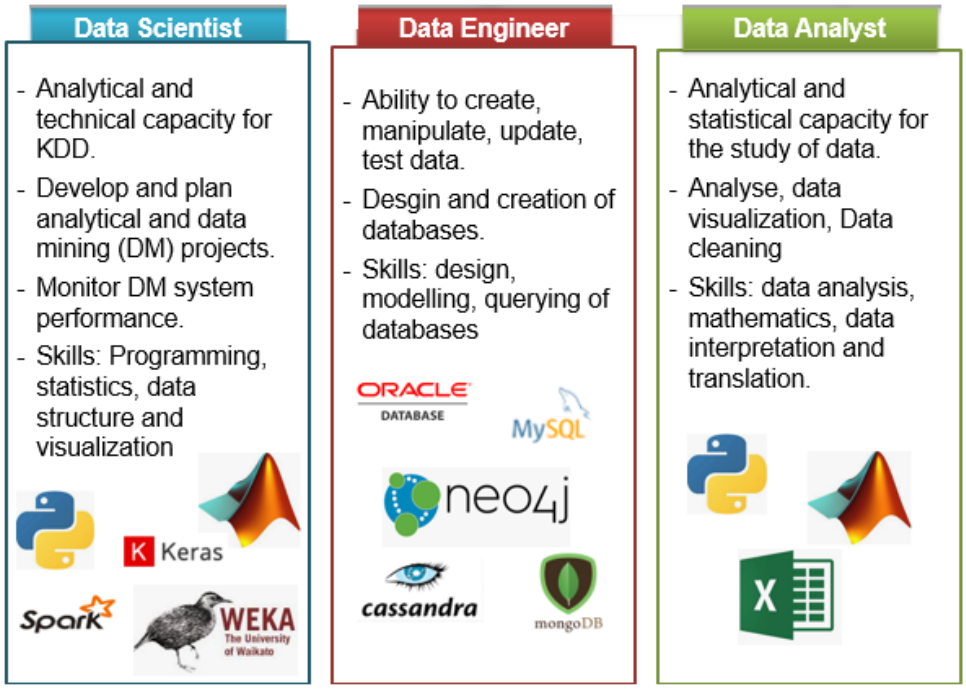


Figure 1.2. Data scientist, data engineer and data analyst

1.3. Types of data mining

According to the type of data to be mined, we distinguish four types of data mining: text mining, multimedia mining, web mining, and graph mining.

1.3.1. Text mining

Text mining involves extracting valuable information from text data. It is used to analyze sentiment, extract entities, summarize text, and perform other tasks. Techniques like sentiment analysis, topic modeling, and named entity recognition are part of text mining.

Text mining can be used to improve the comprehensiveness and relevance of information retrieved from databases. Examples of text mining tools, VantagePoint, which was developed by Search Technology, Inc., with the technology assistance of Dr. Alan Porter, the director of Technology Policy and Assessment Center (TPAC) at Georgia Tech.

1.3.2. Multimedia mining

Multimedia mining is critical for maximizing the value of varied data sources, allowing organizations to acquire deeper insights and make more educated decisions. Multimedia mining aims to bridge the gap between different types of media and extract meaningful insights by analyzing and correlating data from multiple sources. It involves applying data mining and machine learning techniques to multimedia data to discover hidden relationships and patterns.

1.3.3. Web mining

Web mining specializes in the field of customer profiling using Web log files, assessing customer trace and historical data with the purpose of increasing product sales on the Web site. There are three main categories of web mining: Web Usage Mining, Web Content Mining, and Web Structure Mining.

- **Web usage Mining:** focuses on analyzing user interactions and behaviors while they brow websites, for example log files
- **Web Content Mining:** focuses on extracting information and knowledge from the textual content present on web pages. Example, the content of web page.
- **Web Structure Mining:** focuses on analyzing the relationships and link structures among web pages. Example, use PageRank algorithm to assign a score to web pages based on the number and quality of links.

1.3.4. Graph mining

A graph mining aims to analyze relationships between elements in graph or network structures, such as social networks, citation networks, and recommendation networks. The data is structured as a collection of nodes and edges (or links or connections) that represent relationships between nodes. Graph mining aims to analyze nodes, edges, discovering frequent sub graphs or patterns within a larger graph, etc.

1.4. Data Mining Tasks

Data mining involves the process of discovering knowledge, patterns and associations from prepared datasets. There are several data mining tasks such as, summarization, classification, regression, clustering, association rules, deviation detection, trend analysis, pattern recognition, collaborative filtering, etc.

1.4.1. Summarization

The summarization aims to give a general overview of the data using descriptive statistical measures (max, min, mean, etc.), graphical presentation (histogram, scatter plot, etc.) or in the form of “If-then” rules.

1.4.2. Classification

The classification or categorization aims to determine the class of an object, based on its attributes. A classification model is constructed by analyzing the relationship between the attributes and the classes of the objects in the training dataset. It predicts the class label of new data instances. Example diagnoses a new patient's disease.

1.4.3. Regression

The regression aims to predict a continuous numerical value (or vector) based on input features or variables. It is utilized to establish relationships between independent variables (predictors or features) and the dependent variable (the target or outcome). Regression models can be simple or complex, and they come in various forms, for example linear regression where the relationship between the independent variables and the dependent variable is assumed to be linear.

1.4.4. Clustering

The clustering aims to identify classes, also called clusters or groups for a set of objects whose classes are unknown. In other words, it involves grouping similar instances together, where the intra-class similarities are maximized and the interclass similarities are minimized. Example, determine the different kinds of disease.

1.4.5. Association rules

Association rules task is the discovery of the relationship or connection between objects in the form of association rules. It is often used in market basket analysis, where you might identify that customers who buy item A for example coffee are likely to buy item B such as milk as well.

1.4.6. Deviation detection

Deviation detection aims to identify points that cannot be fitted into the segment and then explain whether each such point is noise or should be examined in more detail. It is used to find rare events or outliers in data, which could be indicative of fraud, errors, or unusual behavior.

1.4.7. Trend analysis

Trend analysis refers to sequence analysis and time series analysis. It aims to analyze and predict future values based on past patterns in an object's evolution. Example, estimate this year's profit for an organization based on its profits last year.

1.4.8. Pattern recognition

Pattern recognition is the ability of computers to identify patterns in data, and then use those patterns to make decisions or predictions. Pattern recognition can be miscellaneous information: text, images, emotions, sentiments, sounds, symbols, numbers, etc. It is based on statistics and machine learning techniques and it is very significant in a lot of domains such as multimedia, sonar, speech recognition, vision, agriculture, etc.

1.4.9. Collaborative filtering

Collaborative filtering aims to predict a user's interests by collecting preferences from many users. It is used in recommendation systems, which provide personalized suggestions or recommendations to users.

1.5. Data mining techniques

Data mining techniques have been used successfully in a variety of fields, including business, science, education, etc. There are different types of data mining techniques such as query and reporting, techniques for description-driven data mining, and techniques for discovery-driven data mining, such as machine learning techniques and deep learning.

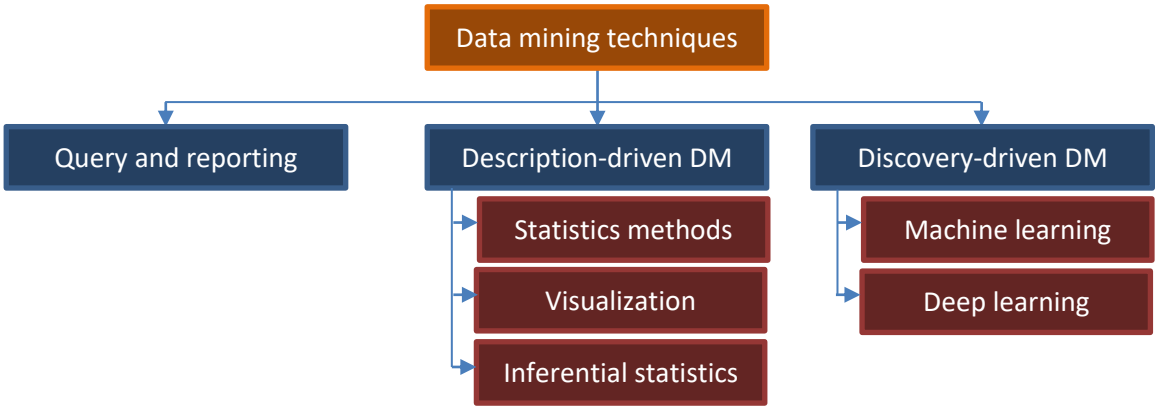


Figure 1.3. Data mining techniques

1.5.1. Query and reporting

Query and reporting technique, involves the creation of a set of queries that best express the stated hypothesis, posing the query to the database, and analyzing the returned data to explore relationships among a few variables and determine statistical significance against a population. Reports contain selected analysis results, presented in graphical, tabular, and textual form, and include a subset of the queries.

Query and reporting can be performed on various data sources, including databases, data warehouses, spreadsheets, etc.

1.5.2. Description-derived data mining

1.5.2.1. Statistical methods

Statistical methods play a fundamental role in data analysis and data mining. They are relied on mathematical analysis for resorting, classifying, organizing, analyzing, and interpreting numerical values. The main statistical metrics used to perform data analysis and data mining are:

- **Minimum and Maximum Values**: Relative to the minimum and maximum value of the sample under analysis.
- **Mean:** Indicates where the data of a distribution is centralized. It is measured by dividing the sum of all values and the number of instances.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- **Median:** indicates to the numerical value that separates the distribution into two equal halves in number of occurrences. It measures as follows:
  - The values are listed in ascending order.
  - If the number of occurrences is odd, the median is the middle.
  - If the number of occurrences is even, the median is the half-sum of the two middle values.
- **Mode:** the value in the sample with the highest number of observations, which is, the most frequent or most common.
- **Variance:** the dispersion of the sample values from the mean.

$$var(x) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2$$

- **Standard deviation:** Measure from how much each distribution value deviates from the mean.

$$\sigma(x) = \sqrt{var(x)}$$

- **Quartiles:** there are three main quartiles that divide the ordered data set into four equal parts. The first quartile corresponds to the value in the middle of the first half of the total sample, which is 25%. The second quartile corresponds to the median, the value that is in the middle. And the third quartile corresponds to the value found in the middle of the second half of the total sample, which is 75%
- **Covariance:** measures how much two variables X and Y change together. If the variables tend to increase or decrease together, the covariance is positive. If one tends to increase when the other decreases, the covariance is negative.

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1}$$

- **Regression**: aims to describe how a dependent variable varies depending on the change of an independent variable.

1.5.2.2. Visualization

Visualization techniques are a very useful method of discovering patterns in data sets and may be used at the beginning of a data mining process to get a rough feeling of the quality of the data set and where patterns are to be found. For example, the Scatter diagrams can be used to identify interesting subsets of the data sets so that we can focus on the rest of the data mining process.

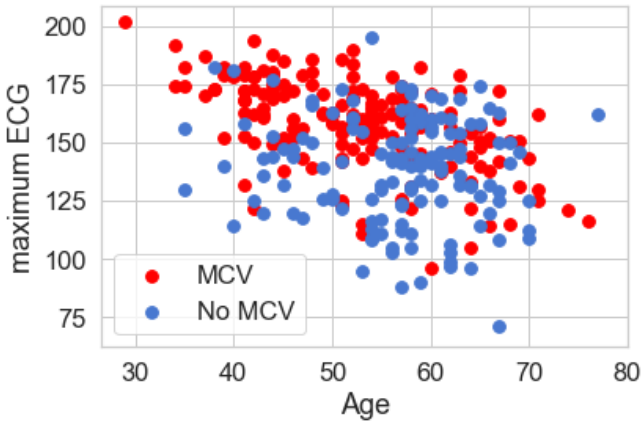


Figure 1.4. Example of data visualization using scatter plot

1.5.2.3. Inferential statistics

Inferential statistics are widely used in research, business, social sciences, and many other fields where making informed decisions about populations based on sample data from dataset to make estimates, decisions, forecasts, and other generalizations.

- **Sampling:** A representative sample is drawn from a larger population. This sample should ideally reflect the characteristics of the population to ensure that the inferences made are valid.
- **Estimation:** estimating population parameters, such as means, proportions, or variances based on the sample statistics.
- **Confidence Intervals:** A confidence interval provides a range within which the population parameter is likely to fall, given the sample data and a certain level of confidence.
- **Hypothesis Testing:** involves making decisions or drawing conclusions about a population based on the sample data. It aims to determine whether the observed differences or patterns in the sample are likely to exist in the population or are just due to chance.

1.5.3. Discovery-derived data mining

1.5.3.1. Machine learning techniques

Machine learning is a subset of artificial intelligence, based on the concept that systems can learn from data, and make predictions or decisions without being explicitly programmed. It is the study of making machines more human-like in their behavior and decisions by giving them the ability to learn and develop their own programs.

Machine learning is a powerful tool that can be used to solve a wide range of problems. Spam detection is one of the best and most common problems solved by Machine Learning, where the trained Machine Learning model (a mathematical representation of a real-world process) is used to identify all the spam emails based on common characteristics such as the email, subject, and sender content. Good quality data is fed to the machines, and different algorithms are used to build Machine Learning models to train the machines on this data.

There are three principal types of Machine Learning: supervised learning, unsupervised learning and reinforcement learning. A semi-supervised learning can be defined according to the two first types of machine learning.

Supervised learning

Supervised learning is a type of problems that uses a model to learn the mapping between the input and target variables. It is one where data is labeled to inform the machine about the exact patterns it should look for. Although the data needs to be labeled accurately for this method to work. There are basically two types of supervised problems:

- **Classification:** categories to data input based on their features. For example, the inputs are the factors of disease, and the output is a class label which identifies the type of disease into different classes.
- **Regression:** The difference between classification and regression is the fact that the dependent attribute is numerical for regression and categorical for classification. The regression predicts numerical values based on input features. For example, the inputs are the features of the house, and we predict the price of a house as in outputs.

Unsupervised learning

Unsupervised learning, as the name suggests, has no data labels. The goal here is to interpret the underlying patterns in the data in order to obtain more proficiency in the underlying data. Clustering is the operation to group similar data points together without predefined labels. The dimensionality reduction is considered an unsupervised learning technique to reduce the number of features in the dataset while preserving essential information. Principal Component Analysis (PCA) is an example of this technique.

A mix of labeled and unlabeled data used for training gives the semi-supervised learning. This approach can be beneficial when obtaining labeled data is expensive or time-consuming.

Reinforcement Learning

Reinforcement learning where an agent operates in an environment with no fixed training dataset. It is based on the feedback or reward given to the agent by the environment in which it is operating. An agent learns to take actions in an environment to maximize a reward signal. It used in applications like game playing and robotics.

1.5.3.2. Deep Learning

Deep learning is a subfield of machine learning that uses multi-layered artificial neural networks (they are inspired by the human brain) to learn complex patterns in data. Convolutional Neural Networks (CNNs) for image analysis, Recurrent Neural Networks (RNNs) for sequential data, and Transformers for natural language processing are examples.

1.6. Data mining process

Cross-Industry Standard Process for Data Mining "CRISP-DM" methodology defines a standard process of data mining in six phases from business understanding to deployment of results.

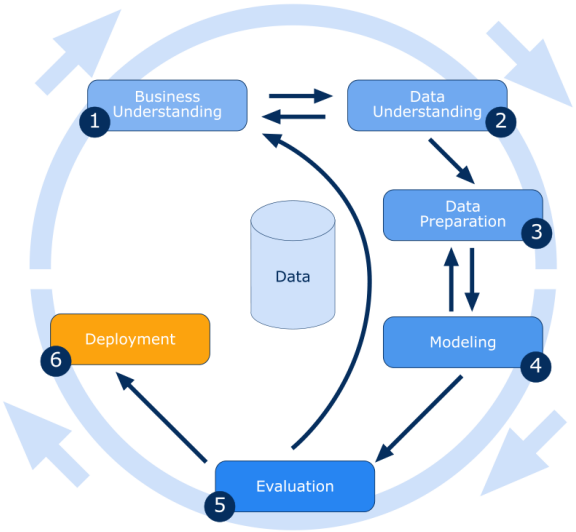


Figure 1.5. Data mining process[2]

1. **Business understanding:** It focuses on understanding the project objectives and needs from a business perspective and identifies a data mining challenge. It aims to create a preliminary project plan outlining key tasks, resources, and timelines. For example, to diagnose heart disease, we need to understand this disease and the important factors and tests such as Electrocardiogram ECG. With the business understanding, we can determine the data necessary for the exploration and correctly interpret the results obtained.
2. **Data understanding:** After a good understanding of problem domains and domain expertise, data understanding involves understanding the data before previous data is collected and proceeds. In this phase, we collect the data required for the project, and explore its structure and attributes. Continuing with the previous example, it may be important to understand the ECG trace.
3. **Data preparation:** Typically consumes about 90% of the time of the project. It includes all efforts for gathering data that will be supplied into modeling tools. The tasks will almost be repeated at this phase, such as handle outliers, missing values, selecting appropriate features, etc. To go to the next phase, the data must be divided into training, validation, and testing sets for model development and evaluation.
4. **Modeling:** During this phase, data mining techniques are chosen and utilized using the prepared data and fine-tuned model parameters. One or more models are created on the prepared data set. Example of data mining techniques: classification (decision trees, naïve Bayes classifier, k-nearest neighbor, neural networks), clustering (k-means, hierarchical clustering, density-based clustering), association (one-dimensional,

[2] https://barnraisersllc.com/2018/10/01/data-mining-process-essential-steps/

multidimensional, multilevel association, constraint-based association). Thus, improve outcomes by iterating on model creation and refining.

5. **Evaluation:** It involves the evaluation of the performance of the developed models using evaluation metrics such as, accuracy, precision, recall, F1-score. To establish alignment, compare the results to the initial business goals defined in the first phase.
6. **Deployment:** It involves the integration of the model into existing business processes, systems, or applications. This phase might be as simple as creating a report or as complicated as putting a repeatable data mining strategy in place.

1.7. Data mining application areas

Data mining domain has been applied in many areas such as:

- Market management. Target marketing, customer relationship management, market basket analysis, cross-selling, market segmentation.
- Health care, where the health care organizations have large collections of data about patients. Understanding relationships in this data is critical for a wide variety of problems.
- E-commerce which provides large datasets in which the analysis of marketing patterns and risks patterns is critical.
- Business transactions, where businesses have millions of customers and billions of their transactions, which need to understand risks.
- Agriculture for example, optimizing resources like water and fertilizers using machine learning.

1.8. Data mining tools and programming languages

There are several data mining tools, the most popular are:

Rapidminder is a free open-source data science platform that features hundreds of algorithms for data preparation, machine learning, deep learning, text mining, and predictive analytics. Its drag-and-drop interface and pre-built models allow non-programmers to intuitively create predictive workflows for specific use cases.

Weka is an open-source machine learning software with a vast collection of algorithms for data mining. It supports different data mining tasks, like preprocessing, classification, regression, clustering, and visualization, in a graphical interface that makes it easy to use. For each of these tasks, Weka provides built-in machine learning algorithms which allow you to quickly test your ideas and deploy models without writing any code.

Orange is a free, open-source data science toolbox for developing, testing, and visualizing data mining workflows. It is a component-based software, with a large collection of pre-built machine learning algorithms and text mining add-ons.

is an open-source platform for creating scalable applications with machine learning. Its goal is to help data scientists or researchers implement their own algorithms. Apache Mahout is free to use under the Apache licence and it's supported by a large community of users.

Enterprise Miner is an analytics and data management platform. Its goal is to simplify the data mining process to help analytics professionals turn large volumes of data into insights.

is a component of Oracle Advanced Analytics that enables data analysts to build and implement predictive models. It contains several data mining algorithms for tasks like classification, regression, anomaly detection, prediction, and more.

is a machine learning platform that specializes in text mining. Available in a user-friendly interface, you can easily integrate MonkeyLearn with your existing tools to perform data mining in real-time.

is a free, open-source platform for data mining and machine learning. Its intuitive interface allows you to create end-to-end data science workflows, from modeling to production. And different pre-built components enable fast modeling without entering a single line of code.

Statistica is a data analysis and visualization program. Its data analysis capabilities cover thousands of algorithms, functions, tests, and methods ranging from simple break-down tables to advance nonlinear modeling, generalize linear models, and time-series methods.

The programming languages suited for data mining are: python, R and Matlab.

Python: is the most popular programming language, open-source, powerful packages and rich ecosystem of libraries. The most used libraries for data science and machine learning purposes: NumPy for mathematical functions, pandas for performing all kinds of manipulation of data, Matplotlib for data visualization, scikit-learn for developing machine learning algorithms, TensorFlow for developing machine learning and deep learning algorithms, and Keras to train neural networks with high performance. Python is our focus in this course for developing data mining applications.

R: is an open-source, domain-specific language, explicitly designed for data science as well as statistical computing and machine learning. Like Python, R has a large community of users and a vast collection of specialized libraries, for example dplyr for data manipulation, and ggplot2 for data visualization.

Matlab: is a language mainly designed for numerical computing, based on matrices. It provides powerful tools to carry out advanced mathematical and statistical operations. Maltlab is a software platform optimized for solving scientific problems. In

Matlab calculation, visualization, and programming are integrated in an easy-to-use environment, where problems and solutions are expressed in familiar mathematical notation.

Conclusion

This chapter presents data mining as a process of extracting useful knowledge from large and complex datasets. It explains the Knowledge-Developmental Data (KDD) process, data types, and the differences between databases and data warehouses. The main types of data mining text, multimedia, web, and graph are presented, along with fundamental tasks such as classification, clustering, and association rules. The chapter also briefly touches on key techniques, including statistics, machine learning, and deep learning, and describes the CRISP-DM methodology used to guide data mining projects.

PRACTICE ONE

DATA MANIPULATION WITH PANDAS

1. Aim of the practice

- Discover the principal libraries and functions for data mining and data analysis
- Discover the different data structures with pandas.
- Data manipulation on DataFrame.

2. Libraries

Python libraries are collections of modules that contain useful codes and functions, eliminating the need to write them from scratch.

To import library, we use the following syntax:

Import name[.subpackage] [**as** variable]

Example:

```
import math
x=input("Enter a value for X: ")
x
R=math.sqrt(int(x))
print("The square root of ", x, " is ", R)
```

- To import some functions, we use the following syntax:
  From library name **Import** function name [**as** variable] or
  Import library name. function name [**as** variable]

2.1. NumPy

NumPy is a Python library that provides a multidimensional array object. It facilitates advanced mathematical and other types of operations on large numbers of data. It manipulates the matrix to easily improve machine learning performance.

import numpy as np

Exercise 01: Write the following code

```
import numpy as np
a= np.array([[1, 2, 3], [4, 5, 6]])
print(a)
```

Try with the following expressions: type(a), type([0, 1, 2]), np.shape(a), a[0], a[0,0], a[:, 0], np.amax(a), len(a), np.zeros([5, 3]), b=a[:, 1,2)], c=np.sin(a).

- Show the number of columns of the array “a”

Linespace () Return evenly spaced numbers over a specified interval.

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

Example:

Array with 7 equally spaced samples in the closed interval [2, 6]

```
np.linspace(2, 6, 7)
```

```
array([2.        , 2.66666667, 3.33333333, 4.        , 4.66666667,
       5.33333333, 6.        ])
```

Exercise 02:

Write a program to create the following array using a loop.

```
[[ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]
 [ 5 10 15]]
```

2.2. Matplotlib

Matplotlib is a Python library focused on data visualization and primarily used for creating beautiful graphs, plots, histograms, and bar charts. It is compatible for plotting data from SciPy, NumPy, and Pandas.

from matplotlib import pyplot as plt or **import matplotlib.pyplot as plt**

Example:

```
import matplotlib.pyplot as plt
x = np.arange(-10, 11)
y = x**2
plt.plot(x, y)
```

Exercise 03:

Use the previous function y=x**2 to show it using:

- The scatter plot “plt.scatter”.
- The bar chart “plt.bar’.

Generate line plot of cosinus and sinus function where the range is between -5 and 5 with 100 elements equally spaced. Use plt.xlabel, plt.ylabel, plt.title and plt.legend(loc='lower right') to show a complete plot.

2.3. Seaborn

Seaborn is another open-source Python library, one that is based on Matplotlib (which focuses on plotting and data visualization) but features Pandas’ data structures.

import seaborn as sns

2.4. Pandas

Pandas (Panel data) is the responsible for preparing high-level data sets for machine learning and training. It relies on two types of data structures, one-dimensional (series) and two-dimensional (DataFrame). **import pandas as pd**

3. Data structures with Pandas

Data Structures are a way of organizing data so that it can be accessed more efficiently depending upon the situation.

3.1. Series

Series can be created by:

- Directly Via S=Pd.Series(Data, Index, Dtype, Name, Copy)
- Passing in a List of values
- Passing in a Dictionary

Examples:

```
import pandas as pd
student= pd.Series(['Nacer', 19, '2RTW'],
                   index = ['Name', 'Age', 'Level'])
student
```

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
s
```

```
 dict={'Name':'Bader', 'Age':20, 'Level':'3RTW'}
ST= pd.Series(dict)
ST
```

Retrieve a single element using index label value, for example s[101].
Retrieve the first two elements of student: student[:2].
Retrieve the last element of ST: ST[-1:].

3.2. Data frame

The fundamental Pandas object is called a DataFrame. It is a 2-dimensional size-mutable, potentially heterogeneous, tabular data structure.
A DataFrame can be created by:

- Directly Via: Pandas.Dataframe(Data, Index, Columns, Dtype, Copy)
- Passing in a List of Lists
- Passing in a Dictionary
- Passing in a List of Series
- Reading Data from a CSV File.

Examples:

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
df
```

	a	b	c
0	1	2	NaN
1	5	10	20.0

```
import pandas as pd
Students=pd.DataFrame()
Students['Name']=['Ali', 'Omar', 'Sami', 'Sarah']
Students['Age']=[18, 19, 20, 18]
Students['Level']=['1Master', '3Licence', '2RTW', '1RTW']
Students
```

	Name	Age	Level
0	Ali	18	1Master
1	Omar	19	3Licence
2	Sami	20	2RTW
3	Sarah	18	1RTW

Exercise 04:

Create a dataframe df from a list of series. You can use the previous series Student and ST.

Reading Data from a CSV File:

CSV files are the Comma Separated Files. To access data from the CSV file, we require a function read_csv() from Pandas that retrieves data in the form of the data frame.

	A	B	C	D	E	F	G
1	age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,slope,ca,thal,target						
2	63,1,3,145,233,1,0,150,0,2.3,0,0,1,1						
3	37,1,2,130,250,0,1,187,0,3.5,0,0,2,1						
4	41,0,1,130,204,0,0,172,0,1.4,2,0,2,1						
5	56,1,1,120,236,0,1,178,0,0.8,2,0,2,1						
6	57,0,0,120,354,0,1,163,1,0.6,2,0,2,1						

Examples:

```
import pandas as pd
df=pd.read_csv("data.csv")
df
```

Save dataframe to CSV file:

```
Students.to_csv("stds.csv")
```

Question: show the content of stds.

4. Data Manipulation on Dataframe

- **Adding data in DataFrame using Append Function**

```
newStudent=pd.DataFrame({'Name':['Said'], 'Age':[19], 'Level':['2SIQ']})
Students.append(newStudent, ignore_index=True)
```

	Name	Age	Level
0	Ali	18	1Master
1	Omar	19	3Licence
2	Sami	20	2RTW
3	Sarah	18	1RTW
4	Said	19	2SIQ

- **Columns selection**

Students[['Name', 'Level']]

- **Columns Deletion**

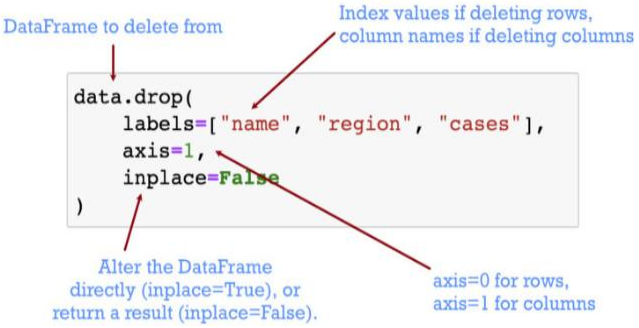
s.drop("Name", axis=1, inplace=True)

- **Rows selection**

lab=s.loc[:,"Age"]

- **Rows Deletion**

df = df.drop(labels=4, axis=0)



Exercise 05:

Use the heat.csv to manipulate data in dataframe and run the following commands:

1. df.head()
2. df.columns
3. df.shape
4. df.tail()
5. df.describe()
6. df.corr()
7. df1 = df.rename(columns={'cp': 'chest pain'})
8. df1.dtypes

Solution of Practice One

Chapter 2.

Exploratory data analysis and Data preprocessing

Introduction

The most time-consuming part of the data mining process is preparing data for analysis and mining. Many years of practice show that the successful application of data mining requires data preprocessing and good understanding of problem domains and domain expertise.

After a good understanding of problem domains in the first step of data mining process (Business understanding) and the collect of data, the data preprocessing (or data preparation) aims to manipulate or prepare raw data from one or more sources into a structured and clean data set for analysis.

A robust data preprocessing system is required in order to draw any kind of knowledge from even medium-sized data sets. The data must not only clean of errors and redundancy but organized in a fashion that makes sense for the problem. If this phase is not performed adequately, it is not possible for the mining algorithms to provide reliable results.

Exploratory Data Analysis (EDA) is the very first step for data understanding before you can perform any changes to the dataset or develop a statistical model to answer business problems. Experiment data are typically organized in a table format (such as a spreadsheet or database), usually with each row representing a different experimental subject and each column containing a subject identifier, outcome attribute or variable. It is important to understand the different variables before proceeding with EDA techniques.

2.1. Types of variables

A variable is a characteristic that can be measured and can take on different values, for example price of product, level of student, weight of person, etc. Variables can be divided into two primary categories: categorical and numeric. Each category is further divided into two subcategories: nominal or ordinal for categorical variables, and discrete or continuous for numeric variables.

2.1.1. Categorical variables

A categorical variable or qualitative variable refers to a characteristic that cannot be quantifiable. Categorical variables can be either nominal or ordinal.

Nominal variables: nominal variable is one that represents a name, label, or category without any inherent order. For example, sex is a nominal variable, which can be represented by categories such as male and female.

Ordinal variables: An ordinal variable is a variable whose values are defined by an ordered relationship between the different categories. Ordinal attributes can also be created by discretizing numeric variables, where the range of values is divided into a finite number of categories. For example, the level variable is ordinal because the category "Excellent" is considered better than "Very Good," which is better than "Good," and so on. Each ordinal value corresponds to a specific numeric range, such as "Excellent" representing a student grade greater than 18.

2.1.2. Numeric variables

A numeric variable or quantitative variable is a quantifiable characteristic whose values are numbers. Numeric variables may be either continuous or discrete.

Continuous variables: A variable is said to be continuous if it can assume an infinite number of real values within a given interval. For instance, the grade of a student is ranged between 0 and 20, but between 0 and 20, the number of possible values is theoretically infinite.

Discrete variables: A discrete variable can assume only a finite number of real values within a given interval. Example, the number of students cannot be a fraction of an integer.

2.2. Techniques of Exploratory Data Analysis

Exploratory Data Analysis allows understanding the dataset by giving an overview such as the size, variables, labels, and data types (categorical or numeric). It gives a descriptive statistics, data visualization, and identification of issues in data like missing values, outliers, etc. EDA is generally cross-classified in two ways: Descriptive Statistics involve calculation of summary statistics, and data visualization display the data in a diagrammatic or pictorial way.

2.2.1. Descriptive Statistics

Descriptive statistics represent the different metrics (see chapter 1), such as Minimum and Maximum values, Mean, Median, Mode, Variance, Standard deviation, Quartiles, Correlation, and Regression. These methods can be classified into two classes: Univariate methods for one variable at a time and multivariate methods look at two or more variables at a time to explore relationships.

2.2.1.1. Univariate measures

Univariate statistic EDA aims to better appreciate the sample distribution and also to make some tentative conclusions about what population distribution is compatible with the sample distribution. Depending on the type of variable (numeric or categorical), the univariate measures for a numeric variable include central tendency, spread, skewness, and kurtosis.

The central tendency: represents the location of a distribution by mean, median, and mode.

- Mean: is the average of data values.
- Median: the middle value after all of the values is put in an ordered list.
- Mode: represents the most frequently occurring value.

The spread of a distribution, including:

- Min, Max: lower and higher value of the variable.
- Variance: is a standard measure of spread, because the bigger the deviations from the mean, the bigger the variance gets.
- Standard deviation: is simply the square root of the variance. Therefore, it has the same units as the original data, which helps make it more interpretable.
- Quartiles and interquartile range (IQR): IQR = Q3 − Q1.

Skewness and Kurtosis: skewness measures the asymmetry of a distribution. It indicates whether the data is skewed to the right (positive skew) or left (negative skew). Kurtosis is a measure of peakedness of a distribution. The positive kurtosis is more peaked, it means there are more extreme values (outliers), and while the negative kurtosis (less than 3) means that the data has fewer outliers.

For the categorical variable, the statistical measure is simply the range of values and the frequency (or relative frequency) of occurrence for each value.

2.2.1.2. Multivariate measures

Multivariate measures generally show the relationship between two or more variables in the form of either cross-tabulation or statistics.

Cross-tabulation: For two variables, cross-tabulation is performed by making a two-way table with column headings that match the levels of one variable and row headings that match the levels of the other variable, then filling in the counts of all subjects that share a pair of levels.

The basic statistics of interest help in understanding how variables are related to each other are covariance and correlation.

Covariance is a measure that indicates the extent to which two variables change together, showing how much (and in which direction) one variable is likely to change when the other changes.

$$cov(x,y)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1}$$

The correlation coefficient (often referred to as Pearson's correlation coefficient) measures the strength and direction of the linear relationship between two variables. It is a normalized version of covariance, which makes it easier to interpret since it ranges between -1 and 1. The formula for the correlation coefficient r between two variables X and Y is:

$$r=\frac{Cov(X,Y)}{\sigma(X)\sigma(Y)}$$

The correlation coefficient r has the following interpretations:

- r=1: Perfect positive correlation.
- r=−1: Perfect negative correlation.
- r=0: No linear relationship.
- 0<r<1: Positive correlation (as one variable increases, the other tends to increase).
- −1<r<0: Negative correlation (as one variable increases, the other tends to decrease).

When we have many quantitative variables the most common statistical EDA technique is to calculate all of the pairwise covariances and correlations and assemble them into a matrix.

2.2.2. Data visualization

Statistical and visual methods complement each other. While the descriptive statistics are quantitative and objective, they do not give a full picture of the data; therefore, data visualization, which is more qualitative and involves a degree of subjective analysis, is also required.

2.2.2.1. Univariate plots

The univariate plots aim to look graphically of the single variable of data on n subjects (the sample size). The histograms are one of the best ways to quickly learn a lot about your data, including central tendency, spread, modality, shape and outliers. The only one of these techniques that makes sense for categorical data is the histogram.

The histogram is a bar plot in which each bar represents the frequency (count) or proportion of cases for a range of values. Figure 2.1 shows the histogram of the frequency of medical data for the values of cp variable (four values 0, 1, 2, and 3).

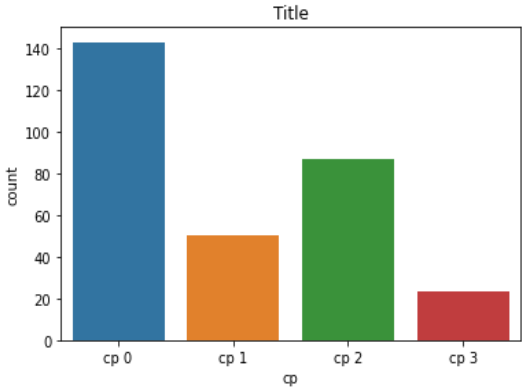


Figure 2.1. Histogram for the values of cp.

Boxplots are very good at presenting information about the central tendency symmetry and skew, as well as outliers, although they can be misleading about aspects such as multimodality.

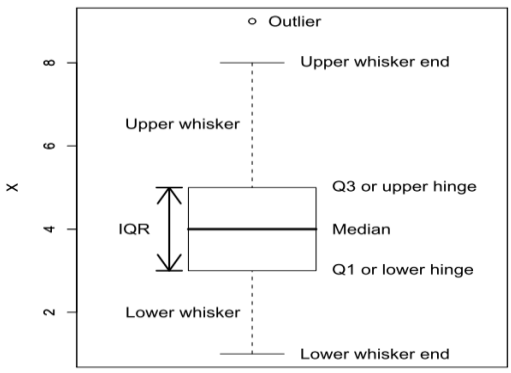


Figure 2.2. Boxplot representation[3].

[3] https://bookdown.org/steve_midway/DAR/exploratory-data-analysis.html

Quantile-Normal plots allow detection of non-normality and diagnosis of skewness and kurtosis. It determines how well a sample of data of size n matches a Gaussian distribution with mean and variance equal to the sample mean and variance. By examining the quantile-normal plot we can detect left or right skew, positive or negative kurtosis, and bimodality.

2.2.2.2. Multivariate plots

There are few useful techniques for visual EDA of two categorical random variables. The only one used commonly is a grouped barplot with each group representing one level of one of the variables and each bar within a group representing the levels of the other variable.

For two quantitative variables, the basic graphical EDA technique is the scatterplot which has one variable on the x-axis, one on the y-axis and a point for each case in your dataset.

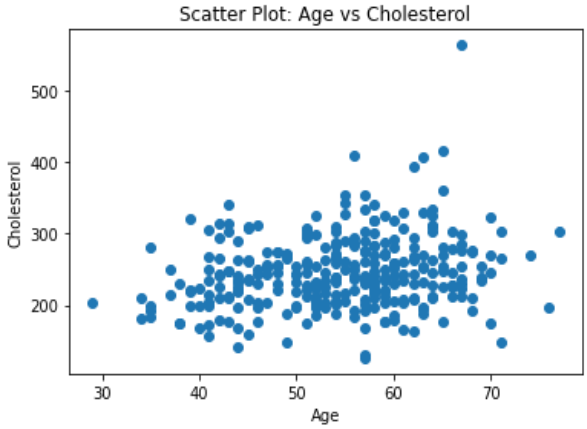


Figure 2.3. Example of scatter plot between two variables (Age and Cholesterol).

Correlation heatmap is a visual representation of a correlation matrix, where the individual correlation coefficients between variables are visualized as colors. The color intensity or gradient reflects the strength and direction of the relatSionship between the variables.

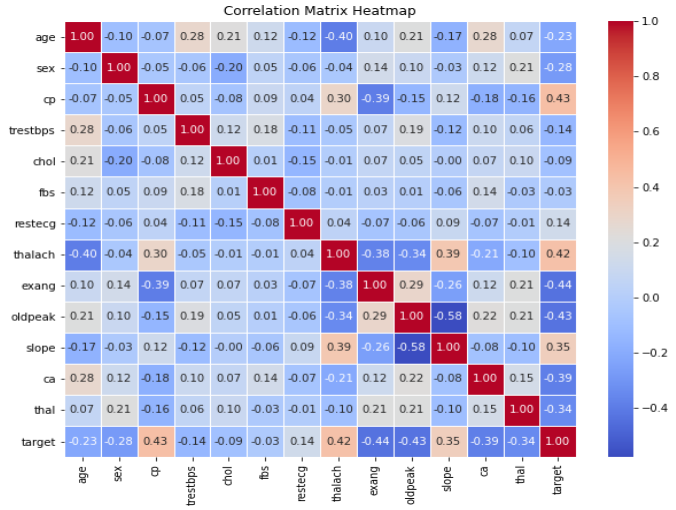


Figure 2.4. Example of correlation matrix heatmap

2.3. Data pre-processing tasks

Data is often incorrect and incomplete, distributed across many databases, organized using incompatible data models and formats, and using often-incomprehensible naming conventions. Hence, there is a need to preprocess the data to make it easier to mine.
Preprocessing tasks include the selection of appropriate data subsets for performance and consistency reasons, as well as complex data transformations to bridge the representation gap. They are classified into four categories: data cleaning, data transformation, data integration, and data reduction.

2.3.1. Data Cleaning

After data is collected or obtained, whether it is from databases, APIs, surveys, sensors, or other sources, the data cleaning is very important to deal with missing values, outliers, and noisy data.

The following are some of the advantages of data cleaning:

- It aids in the removal of erroneous data that could lead to poor decision-making.
- Use the accurate datasets.
- Companies can be sure of a high engagement rate when they work with the correct marketing database, giving them the optimal value for their money.
- Employees will spend less time contacting expired contacts or clients.

Cons of data cleaning:

- Data cleaning may require a lot of time, especially when working with a huge amount of data.
- The procedure can be costly depending on the size of the data and the degree of corrections required.
- When missing data and anomalies are deleted, it may make the problem worse.

2.3.1.1. Missing data

Data cleaning allows dealing with missing data. It allows identifying and handling missing or incomplete data in a dataset to improve its quality and reliability. Missing data can lead to biased analysis and inaccurate results. Example of missing value, the special values NULL, NaN (Not a number), NA (Not Available).
After identifying missing values by using a summary statistics or visualization from EDA techniques, we apply a strategy of handling missing data, which are: delete missing data, data imputation and create a missing data indicator variable.

Delete missing data (Listwise Deletion): data cleaning comprises the removing data with missing values of data that are missing or unknown. It involves removing entire rows or columns that contain missing values. The use of this strategy must be done with caution since it can lead to a loss of information, especially if missing data is not entirely random. So, it is not always the best approach.

Data imputation: another strategy to deal with missing data is to replace or input missing values with estimated values. The common imputation methods are based on other observations, such as the Mean, Median, mode or standard value of the observed data in

the same column, for example missing value of the temperature of a person, it can be replaced by the standard temperature of 37 degree Celsius.
Other methods of data imputation are:

- Linear Regression Imputation: Predict missing values using a linear regression model based on other available variables.
- K-Nearest Neighbors (KNN) Imputation: Replace missing values with values from the K-nearest neighbors in the dataset.
- Interpolation: Estimate missing values based on the values before and after the missing point, which is useful for time series data.

The data imputation is used with caution.

Create a missing data indicator variable: it aims to create a new binary variable that indicates whether a value was missing or not. This approach preserves the information about missingness.
The choice of a technique of handing of data missing may be depend on the nature of the data and the problem.

2.3.1.2. Noisy Data

Noisy data refers to data that contains errors, random variations, or irrelevant information. This noise can come from: Measurement errors, Sensor faults, Human input errors, or Environmental factors. It usually distorts the true pattern in the data and makes learning or analysis harder. For example, if someone's height is recorded as 1600 cm instead of 160 cm due to a typo.
Methods to handle noisy data:
Binning: This method is used to smooth or handle noisy data. First, the data is sorted, and then the sorted values are separated and stored in the form of bins. There are three methods for smoothing data in the bin. Smoothing by bin mean, smoothing by bin median, Smoothing by bin boundary.
Regression: This is used to smooth the data and will help to handle data when unnecessary data is present. For the analysis, purpose regression helps to decide the variable which is suitable for our analysis. Other techniques for handling the noisy data are: Clustering-based methods, Outlier detection and removal, Data transformation (log, normalization), Decision Tree Induction, etc.

2.3.1.3. Outliers

Outliers express deviation of data points in a dataset. They may indicate a rare event or another category of data. Understanding outliers is an essential part of data cleaning. Outliers can distort statistics, impact model training, and even lead to wrong insights, especially in algorithms sensitive to distance or mean values.

Different approaches to identify outliers for example, visualization techniques, probabilistic approaches by estimating the likelihood of each element in the data, and the clustering approach by partitioning the dataset on clusters and the anomalous data will be in the small clusters. An example of outliers: In a dataset of people's ages, most individuals may be

between 20 and 80 years old, but there could be outliers, such as centenarians who are over 100 years old.

To deal with outliers, we can use the following methods:

Remove outliers: remove outliers should be apply with caution since improper data entry, may help the performance of the data. This method is needed to determine the validity of that number. If an outlier proves to be irrelevant for analysis or is a mistake, consider removing it.

Data transformation: consists to alter the values of outliers. Winsorization is a data transformation method that replaces extreme values with less extreme values after identifying a threshold (e.g., 99th percentile). Other methods like log transformation and Square Root Transformation.

2.3.1.4. Data deduplication

Data deduplication expresses how duplicate records or rows are detected and removed. It aims to detect and eliminate duplicate records or rows to ensure each data point is unique.

Cleaning data takes a great deal of effort and time. In addition, many of these records were not in a form suitable for data mining; they had to be transformed to more meaningful attributes before mining could proceed.

Several techniques can be used to detect duplicate data. The simplest method is exact matching, which identifies perfectly identical records. Key-based detection uses unique fields such as email address or username, while composite key matching relies on multiple attributes like name and date of birth. When values are similar but not identical, string similarity techniques, such as Levenshtein or Jaro-Winkler, help identify spelling variations. Phonetic algorithms like Soundex detect names with identical pronunciations. Hashing can also be used to identify identical or nearly identical records, and block grouping speeds up comparisons by blocking potential matches. More advanced approaches include clustering, which groups similar records together, and machine learning deduplication, where a model predicts whether two records refer to the same entity.

2.3.2. Data Transformation

Data transformation is the process of converting data from one format or structure into another. Transformation processes can also be referred to as data mapping from one data form into another format for warehousing and analyzing.

Data transformation for constructing data warehouse or data integration by the ETL (Extraction, Transformation, and Loading) process. This process aims to extract and transform the data, which will be loaded in data warehouse.

After the data is extracted from sources, the transformation step aims to convert data before loading it into the central repository or warehouse. The transformation activities include: Conversion to a single format, sorting data, cleaning, pivoting, standardization, deduplication.

Data transformation enables datasets to be converted into different accessible formats. Transformation can be as simple as converting number to string or very complex involving

multiple data fields to be evaluated simultaneously and logic applied to determine the desired value.

Data transformation refers to data normalization or standardization to bring all features to a common scale. Without normalization, variables with high values dominate other variables, leading to biased or inaccurate results. It is essential in distance-based algorithms (KNN, K-Means), gradient-based models (regression, neural networks), regularization techniques, and PCA. Normalization improves model accuracy, learning speed, and fairness among variables. For example, when predicting the price of a house using large-scale variables like area and small-scale variables like the number of rooms, normalization ensures that both influence the model equally.

Data normalization: Data normalization is an essential preprocessing step that transforms raw data into a format suitable for machine learning algorithms. Several techniques can be used depending on the distribution and characteristics of the data:

- **Min-Max Scaling:** This method scales the data to a specific range, often [0, 1]. The normalized value of X is defined by X' as follows:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  Where X_{min} and X_{max} are the minimum and the maximum value in the feature (or attribute), respectively.

- **Z score standardization or zero-mean normalization:** is a standard scalar, which scales the data to have a mean (μ) of 0 and a standard deviation (σ) of 1, which makes it particularly useful when the data follows a normal distribution or when centering is required. The formula for z-score of X is:

$$X' = \frac{X - \mu}{\sigma}$$

- **Robust Scaling:** This method is similar to min-max scaling but uses the first quartile Q_1 and the third quartile Q_3 instead of the range (max-min). It is more robust to outliers. The formula for robust scaling is:

$$X' = \frac{X - Q_1}{Q_3 - Q_1}$$

- **Log Transformation:** This is used for highly deviated data to make the distribution more symmetric. It's often used when the range of values varies widely.

$$X' = \log(X)$$

- **Decimal Scaling:** Decimal scaling normalizes attribute values by moving the decimal point of values so that they fall within a small range, typically between –1 and 1. A value, v, of attribute A is normalized to v' by computing:

$$v' = \frac{v}{10^j}$$

  Where j: the smallest integer such that: $\max(|v'|) < 1$

The choice of normalization method depends on the nature of the data and the needs of the machine learning algorithm. For instance, Min-Max scaling is preferred when the data does not follow a normal distribution and we want to preserve its original shape, whereas Z-score

standardization is more suitable when the data is approximately normally distributed and centering is required.

Data discretization for converting continuous data into discrete intervals, for example converting age into age groups: 0–18, 19–35, 36–60, 60+

Encoding Categorical Variables:

- One-Hot Encoding: Converts categorical values into binary columns, for example cat, dog, and mouse can be encoded by 100, 010, and 001, respectively.
- Label Encoding: Assigns an integer to each category (for ordinal data)
- Target/Mean Encoding: Replaces categories with the mean target value (used with caution).

Feature engineering:

Where new features are created and applied to assist the mining process from the given set of features. This simplifies the original data and makes the mining more efficient. For example, extracting the date from timestamp. It refers to handling text data by tokenization, TF-IDF, etc. and imbalanced data by oversampling or under-sampling techniques.

2.3.3. Data integration

The intent of data integration is to enable the user to extract information from heterogeneous data sources using a unified model. The data integration system (DIS) is considered as an interoperable information system, which copes with heterogeneity and changeability in data sources to be integrated. The DIS can be classified into five categories:

1. Multi-base system allows users to access directly to the different data sources.
2. Federation system provides a federated schema for integrating heterogeneous databases called federated databases.
3. Mediator system provides a virtual integration of data at query time.
4. Data warehouse system is a materialized integration which feeds integrated data into data warehouses.
5. Hybrid system that supports the data integration using a hybrid of the virtual and materialized approaches.

Data integration can be as simple as:

- **Joining and merging** by using for example SQL joins or pandas merge of Python to combine datasets based on common keys or columns.
- **Concatenation:** Concatenate datasets vertically or horizontally when they have the same structure.
- **Data Linkage:** Use record linkage techniques to match records across datasets, especially when dealing with real-world data that might have inconsistencies or typos.

The materialized data integration involves the creation of data warehouse by ETL process while virtual data integration is based on layered architecture where the mediator layer (on top of the architecture) receives the user query and send the integrated data, wrappers

layer to transform the user query to rewrite the input query in terms of data source schema, and the data sources layer, which contains the different participant data sources.

The main functions of data integration are:

1. Data Source Heterogeneity:

Schema Mapping: DIS handles schema mismatches by mapping attributes from different data sources to a common schema. This involves transforming data types, resolving naming conflicts, and ensuring consistency.

2. Data Query and Retrieval:

Query Translation: DIS translates queries from a unified query language to the specific query languages used by different data sources. This translation ensures that queries are correctly executed on the respective sources.

Query Optimization: DIS optimizes queries to reduce the load on individual data sources and improve overall performance.

3. Data Transformation:

Data Format Transformation: DIS transforms data formats to ensure consistency. This includes date formats, units of measurement, and other data type conversions.

Data Cleaning: DIS can perform data cleaning operations, such as handling missing values, duplicates, and outliers, before integrating the data from various sources.

4. Data Aggregation and Fusion:

Data Aggregation: DIS aggregates data from multiple sources to create a consolidated view. Aggregation operations might include sum, average, count, etc.

Data Fusion: DIS merges data from different sources to create a unified record.

5. Data Security and Access Control:

Access Control: DIS enforces access control policies to ensure that users and applications only access the data they are authorized to view or modify.

Data Encryption: DIS can encrypt sensitive data to ensure secure transmission and storage.

6. Error Handling and Logging:

Error Detection: DIS detects errors during the integration process, such as failed queries or data inconsistencies.

Logging: DIS logs integration activities, errors, and transformations for auditing and troubleshooting purposes.

7. Scalability and Performance Optimization:

Load Balancing: DIS distributes queries across multiple data sources to balance the load and improve performance.

Caching: DIS can implement caching mechanisms to store frequently accessed data, reducing the need to query the original sources repeatedly.

8. Real-time Integration:

Streaming: For real-time data integration, DIS handles streaming data sources, ensuring that real-time data is integrated seamlessly with batch data.

2.3.4. Data reduction

Data reduction refers to techniques that reduce the volume of data while maintaining its integrity and meaningful patterns, so that analysis or modeling becomes faster, more efficient, and less memory-intensive. The reduction of the data may be in terms of the number of rows (records) or terms of the number of columns (dimensions). Data reduction strategies encompass both dimensionality reduction and numerosity reduction.

2.3.4.1. Dimensionality Reduction

Dimensionality Reduction refers to techniques that reduce the number of input variables (features) in a dataset while preserving as much information as possible. It is especially applied when we have high-dimensional data (many features), leading to improve model performance and reduce overfitting. There are four techniques of dimensionality reduction.

A. Wavelet Transform:

In the wavelet transform, suppose a data vector A is transformed into a numerically different data vector A' of wavelet coefficients, such that both A, A' vectors are of the same length. These coefficients capture the signal's content at various scales and positions. Once transformed, the data can be approximated by keeping only a small subset of the most significant coefficients (those coefficients whose absolute values exceed a certain threshold). The remaining (small) coefficients are set to zero. For example, Haar wavelet is the simplest wavelet which is discontinuous (or discrete wavelet transform), while all other Daubechies wavelets are continuous. Furthermore, all Daubechies wavelets are generators of orthogonal basis for function spaces $L^2(R^n)$, while spline wavelets generate unconditional basis rather than orthonormal basis, and some wavelets could only generate redundant frames rather than a basis. Figure 2.5 shows Haar and Daubechies wavelets

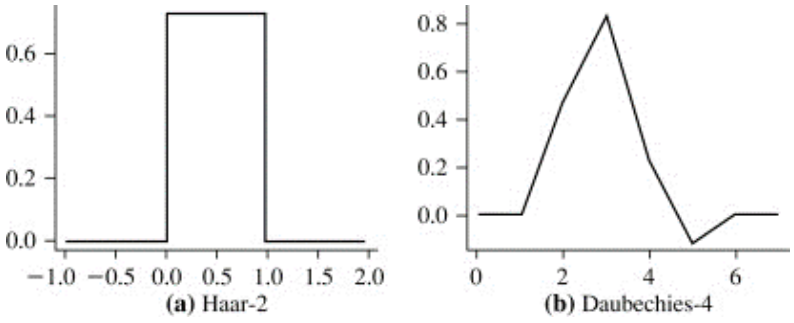


Figure 2.5. Examples of wavelet families[4].

[4] https://www.mathworks.com/help/wavelet/ug/wavelet-families-additional-discussion.html

Simply speaking, a mother wavelet is a function $\psi(x)$ such that $\{\psi(2^j x - k), i, k \in Z\}$ is an orthonormal basis of $L^2(R)$. The term mother implies that the functions with different regions of support that are used in the transformation process are derived by dilation and translation of the mother wavelet. To find the wavelets, we start with a function $\phi(x)$ that is made up of smaller version of itself. This is the refinement (or 2-scale, dilation) equation :

$$\phi(x) = \sum_{k=-\infty}^{\infty} a_k \phi(2x - k)$$

$a'_k s$ are called filter coefficients or masks and the function $\phi(x)$ is called the scaling function (or father wavelet). Under certain conditions, gives a wavelet:

$$\psi(x) = \sum_{k=-\infty}^{\infty} (-1)^k b_k \phi(2x - k) = \sum_{k=-\infty}^{\infty} (-1)^k \bar{a}_{1-k} \phi(2x - k)$$

The scaling function is chosen to preserve its area under each iteration, so $\int_{-\infty}^{\infty} \phi(x)dx = 1$. Integrating the refinement equation then:

$$\int_{-\infty}^{\infty} \phi(x)dx = \sum a_k \int_{-\infty}^{\infty} \phi(2x - k)dx = \frac{1}{2} \sum a_k \int_{-\infty}^{\infty} \phi(u)du.$$

Hence $\sum a_k = 2$. So, the stability of the iteration forces a condition on the coefficient a_k. Second, the convergence of wavelet expansion requires the condition $\sum_{k=0}^{N-1} (-1)^k k^m a_k = 0$ where $m = 0,1,2, \ldots, \frac{N}{2} - 1$ (if a finite sum of wavelets is to represent the signal as accurately as possible). Third, requiring the orthogonality of wavelets forces the condition $\sum_{k=0}^{N-1} a_k a_{k+2m} = 0$. Finally, if the scaling function is required to be orthogonal $\sum_{k=0}^{N-1} a_k^2 = 2$. To summarize, the conditions are

$$\begin{cases} \sum_{k=0}^{N-1} a_k = 2 & stability \\ \sum_{k=0}^{N-1} (-1)^k k^m a_k = 0 & convergence \\ \sum_{k=0}^{N-1} a_k a_{k+2m} = 0 & orthogonalith\ of\ wavelets \\ \sum_{k=0}^{N-1} {a_k}^2 = 2 & orthogonalith\ of\ scaling\ functions \end{cases}$$

Wavelet transform is widely used in signal processing, image compression. The reason it is called ‘wavelet transform’ is that the information here is present in the form of waves, like how a frequency is depicted graphically as signals. The wavelet transform also has efficiency for data cubes, sparse or skewed data. It is mostly applicable for image compression and for signal processing.

B. Principal Component Analysis (PCA):

Also known as the Karhunen-Loève technique. It is a linear algebra-based technique that transforms the original features into a new set of uncorrelated variables called principal components, ordered by the amount of variance they capture from the data. PCA is useful when features are correlated, and we want to reduce redundancy. The process of PCA can be broken down into four key steps.

1. Standardize the data: This involves mean centering $X_{centred}$(subtracting the mean) and scaling (typically dividing by the standard deviation).
2. Compute the covariance matrix: The covariance matrix captures how variables are linearly related to each other. For a dataset with p features, the covariance matrix is an $p \times p$ matrix. The diagonal elements are the variances of each feature and the off-diagonal elements are the covariances between pairs of features.

$$C = \frac{1}{n-1} {X_{centred}}^T X_{centred}$$

3. Calculate eigenvectors (define the direction of principal components) and eigenvalues (indicate the amount of variance captured by each eigenvector.) eigenvalues (λ), solve: $\det(\mathrm{C} - \lambda \mathrm{I}) = 0$ with I is the identity matrix. The eigenvectors $\vec{v}$ is obtained by solving this: $(\mathrm{C} - \lambda \mathrm{I})\vec{v} = 0$
4. Select top k components based on explained variance.

C. Attribute Subset Selection:

The large data set has many attributes, some of which are irrelevant to data mining or some are redundant. The core attribute subset selection reduces the data volume and dimensionality. The attribute subset selection reduces the volume of data by eliminating redundant and irrelevant attributes. Common techniques:

Filter methods: Evaluate each feature independently of any machine learning model. They rely on statistical measures to assign a score, rank, and select the most relevant features before training. Common techniques include the correlation coefficient (to measure linear relationships with the target variable), the chi-square test (to assess the dependence between categorical variables), information gain (to quantify the reduction in uncertainty provided by a feature), and the variance threshold (to remove features with low variance or that are not informative). Fast, simple, and model-independent, these methods are particularly useful in the early stages of dimensionality reduction.

Example: Using the Chi-square test χ^2 for feature selection. Suppose we want to predict whether a customer will buy a product (Yes/No). We have a categorical variable: "Has a discount coupon" (Yes/No).

We collect the following data:

Has Coupon	Bought = Yes	Bought = No	Total
Yes	40	10	50
No	20	30	50
Total	60	40	100

1. Compute all expected counts: Expected value for each cell:

$$E_{ij} = \frac{(Raw\ total)(Column\ total)}{Grand\ total}$$

Has Coupon	Bought = Yes	Bought = No
Yes	30	20
No	30	20

2. Compute the Chi-Square Statistic

$$\chi^2 = \sum \frac{(E-O)^2}{E}$$

Where O = observed, E = expected.
Calculate each cell:

- (40 − 30)² / 30 = 3.33
- (10 − 20)² / 20 = 5
- (20 − 30)² / 30 = 3.33
- (30 − 20)² / 20 = 5

Total: χ^2=3.33+5+3.33+5=16.66

3. Compare with Chi-Square Table
Degrees of freedom: $df = (2-1)(2-1) = 1$
Critical value at significance 0.05: $\chi^2_{0.05,1}$=3.84
Since: 16.66>3.84

This means: The feature "Has Coupon" is strongly associated with the target "Bought". Therefore, it is an important feature and should be selected.

Wrapper Methods: Use a predictive model to evaluate different subsets of features.
Step-wise Backward Selection: This selection starts with a set of complete attributes in the original data and at each point, it eliminates the worst remaining attribute in the set. Suppose there are the following attributes in the data set in which few attributes are redundant.

Initial attribute Set: {X1, X2, X3, X4, X5, X6}
Initial reduced attribute set: {X1, X2, X3, X4, X5, X6}
Step-1: {X1, X2, X3, X4, X5}
Step-2: {X1, X2, X3, X5}
Step-3: {X1, X2, X5}
Final reduced attribute set: {X1, X2, X5}
Step-wise Forward Selection: The selection begins with an empty set of attributes later on we decide best of the original attributes on the set based on their relevance to other attributes. We know it as a p-value in statistics. We take the same previous example.
Initial reduced attribute set: { }
Step-1: {X1}
Step-2: {X1, X2}
Step-3: {X1, X2, X5}
Final reduced attribute set: {X1, X2, X5}

Embedded Methods: Perform feature selection during model training. Built into algorithms like decision trees or Lasso regression.

- **Autoencoders**: Autoencoders are a type of artificial neural network used for non-linear dimensionality reduction and they are often used in image processing, anomaly detection, and feature learning. They consist of two main parts:
  - Encoder: Compresses the input into a lower-dimensional representation (called the bottleneck).
  - Decoder: Reconstructs the input from that compressed form.

2.3.4.2. Numerosity Reduction

Numerosity reduction is a data reduction technique that decreases data volume by replacing the original dataset with a more compact and relevant representation. It is particularly useful when the dataset is too large to process efficiently or when it contains a lot of redundant data and outliers. Numerosity reduction methods can be parametric, in which case only the parameters of a model are stored instead of the data itself, or nonparametric, using reduced representations such as histograms, clustering results, or sampled subsets of the data.

1. Parametric Methods: For parametric methods, data is represented using regression or log-linear model to estimate the data.

Regression: Regression is a predictive modeling technique used to describe the relationship between a dependent variable and one or more independent variables. Regression can be a simple linear regression or multiple linear regression. When there is only single independent attribute, such regression model is called simple linear regression and if there are multiple independent attributes, then such regression models are called multiple linear regression.

Simple Linear Regression: In simple linear regression, the goal is to model the relationship between a dependent variable $\boldsymbol{y}$ and a single independent variable $\boldsymbol{x}$. The model assumes a linear relationship:

$$\mathbf{y} = \mathbf{ax} + \mathbf{b} + \boldsymbol{\varepsilon}$$

where, a is the slope of the regression line, b is the intercept, two regression coefficients, and ε is the error term accounting for deviations of the data from the exact line. The optimal values of a and b are typically estimated using the least squares method, which minimizes the sum of squared errors:

$$min_{a,b} \sum_{i=1}^{n} \left(y_i - (ax_i + b)\right)^2$$

Multiple Linear Regression: When the dependent variable $\boldsymbol{y}$ depends on multiple independent variables, the model becomes:

$$\mathbf{y} = \mathbf{a_1x_1} + \mathbf{a_2x_2} + \cdots + \mathbf{a_px_p} + \mathbf{b} + \boldsymbol{\varepsilon}$$

Log-Linear Model: A log-linear model is a statistical model used to describe the joint probability distribution of a set of categorical (discrete) variables. It expresses the logarithm of cell probabilities in a contingency table as a linear combination of parameters corresponding to different interactions among variables.

1. Data Representation

Suppose we have d discrete attributes: $x_1, x_2, \ldots, x_d$

A specific data point corresponds to a cell in a d-dimensional contingency table:$(x_1, x, \ldots, x_d)$

Let the joint probability of this cell be $P(x_1, x, \dots, x_d)$.

2. Core Idea of the Log-Linear Model: The model expresses probabilities as:

$$\log P(x_1, x, \dots, x_d) = \lambda_0 + \sum_i \lambda_i(xi) + \sum_{i<j} \lambda_{ij}(x_i, x_j) + \sum_{i<j<k} \lambda_{ijk}(x_i, x_j, x_k) + \dots$$

With λ_0 is a constant term, $\lambda_i(xi)$ is a main effect of attribute X_i, $\lambda_{ij}(x_i, x_j)$ is a pairwise interaction between attributes x_i, x_j. In practice, we often truncate the model and keep only main effects, or main effects with pairwise interactions.

Log-linear models are therefore also useful for:

- Dimensionality reduction, since lower-dimensional points collectively occupy less space than the original data points.
- Data smoothing, because aggregated estimates in the lower-dimensional space are less affected by sampling variations than those in the higher-dimensional space.

Both regression models and log-linear models can be applied to sparse data, although their applicability may be limited. While both methods can handle skewed data, regression generally performs exceptionally well. However, regression can become computationally expensive when applied to high-dimensional data, whereas log-linear models offer good scalability up to roughly 10 dimensions.

2. Non-Parametric Methods: These methods are used for storing reduced representations of the data include histograms, clustering, sampling and data cube aggregation.

Histograms: Histogram is the data representation in terms of frequency. It uses binning to approximate data distribution and is a popular form of data reduction.

Step 1: Divide the data into bins (buckets)

The dataset is partitioned into smaller intervals called bins.

Step 2: Replace each bin with a representative value

Common representatives: Average (mean) of the bin, Sum of values in the bin, Median or boundary values (depending on the application).

This reduces noise and simplifies the dataset.

Partitioning Rules

There are two common methods for dividing data into bins: equal-width and Equal-Frequency (Equal-Depth) partitioning.

1. Equal-Width Partitioning

- The range of the data is divided into intervals of equal size.
- Each bin covers the same numeric width.

If the data range is: $range = Max - Min$ and we want k bins, the bin width is: $width = \frac{range}{k}$

2.Equal-Frequency (Equal-Depth) Partitioning

- Each bin contains approximately the same number of data points.
- Bin widths may vary, but the number of observations is equal.

Example: Data: 10 values and k = 5 bins then each bin gets 2 values, regardless of numeric range. This is useful when data is skewed, because it avoids having empty or sparse bins.

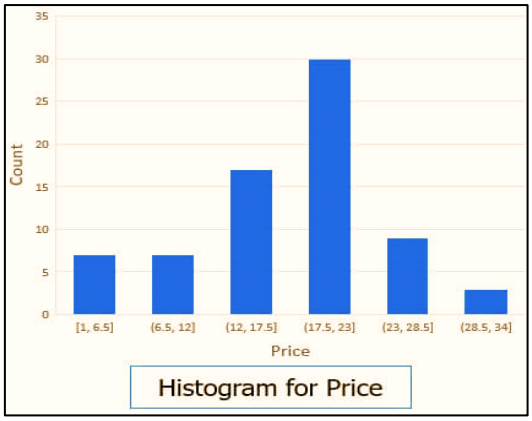


Figure 2.6. Example of Histogram of price[5].

Clustering: Clustering techniques groups the similar objects from the data in such a way that the objects in a cluster are similar to each other but they are dissimilar to objects in another cluster. How much similar are the objects inside a cluster can be calculated by using a distance function. More is the similarity between the objects in a cluster closer they appear in the cluster. The quality of cluster depends on the diameter of the cluster i.e., the at max distance between any two objects in the cluster. The original data is replaced by the cluster representation. This technique is more effective if the present data can be classified into a distinct clustered.

Sampling: One of the methods used for data reduction is sampling as it is capable to reduce the large data set into a much smaller data sample. Below we will discuss the different method in which we can sample a large data set $\boldsymbol{D}$ containing $\boldsymbol{N}$ tuples:

- *Simple random sample without replacement (SRSWOR) of size s:* In this 's number' of tuples are drawn from N tuples such that in the data set D $(s < N)$. The probability of drawing any tuple from the data set D is $1/N$ this means all tuples have an equal probability of getting sampled.
- *Simple random sample with replacement (SRSWR) of size s:* It is similar to the SRSWOR but the tuple is drawn from data set D, is recorded and then replaced back into the data set D so that it can be drawn again.
- *Cluster sample:* The tuples in data set D are clustered into M mutually disjoint subsets. From these clusters, a simple random sample of size s could be generated where $s < M$. The data reduction can be applied by implementing SRSWOR on these clusters.
- *Stratified sample:* The large data set D is partitioned into mutually disjoint sets called 'strata'. Now a simple random sample is taken from each stratum to get stratified data. This method is effective for skewed data.

Data Cube Aggregation: Data cube aggregation involves moving the data from detailed level to a fewer number of dimensions. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

[5] https://binaryterms.com/data-reduction.html

The numerosity reduction can have both advantages and disadvantages. It can improve the efficiency and performance of machine learning algorithms by reducing the number of data points in a dataset. However, it can also result in a loss of information and make it harder to interpret the results. It's important to use numerosity reduction methods and carefully assess the risks and benefits before implementing it. There are many other ways of organizing methods of data reduction. The computational time spent on data reduction should not outweigh or "erase" the time saved by mining on a reduced data set size.

In data compression, transformations are applied so as to obtain a reduced or "compressed" representation of the original data. If the original data can be reconstructed from the compressed data without any information loss, the data reduction is called lossless. If, instead, we can reconstruct only an approximation of the original data, then the data reduction is called lossy. There are several lossless algorithms for string compression; however, they typically allow only limited data manipulation. Dimensionality reduction and numerosity reduction techniques can also be considered forms of data compression.

Conclusion

In this chapter, we highlighted the crucial role of exploratory data analysis (EDA) and data preprocessing in any data mining project. Before applying mining or machine learning techniques, the data must be thoroughly understood, cleaned, transformed, and organized. EDA provides an initial understanding of the dataset through descriptive statistics and visualizations, helping to identify variable types, detect trends, and reveal issues such as missing values, noise, and outliers.

Data preprocessing techniques improve both data quality and the performance of learning algorithms.

Most of the figures presented in this chapter are from my own practical work and illustrate concrete examples of EDA and preprocessing tasks applied to real-world datasets.

Practical work 02

EDA & Data Preprocessing

1. Aim of the practice

- Load and inspect the IRIS dataset.
- Perform univariate and bivariate EDA.
- Preprocessing the IRIS dataset.

2. Load and Inspect the IRIS dataset

Exercice 01 : Import the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
```

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

Exercice 02 : Data understanding:

- Size of dataset
- Explore the top 10 rows (head) and the last 10 rows (tail).
- Getting each attribute's Data Type by df.dtypes and df.info()
- What return this command: `print(df['species'].value_counts())`
- Display the statistical summary of the data by df.describe()
- Check missing values by df.isnull().
- Reviewing Correlation between variables by: corr_matrix=df.corr()

Exercice 03 : Univariate Analysis:

- Plot histograms for each numerical feature.
- Plot boxplots to check for outliers.
- Comment on variable distributions.

Exercice 04 : Bivariate Analysis:

- Compute the correlation matrix.
- Plot the correlation heatmap.
- Create scatterplots for:
    - sepal_length vs sepal_width
    - petal_length vs petal_width
- Plot pairplot colored by species.

Exercice 05 : Data preprocessing:

- Impute missing values using:
    - Mean for sepal_length

  - Median for sepal_width
  - 0 for Petal_length
  - Forward fill for Petal_width
  - Mode for target
- Use the IQR method to detect outliers for sepal_length.
- Remove detected outliers.

Exercice 06 : Encoding & Scaling

- Encode the target variable using LabelEncoder.
- Scale numerical features using:
  - StandardScaler
  - MinMaxScaler

Solution of practical work 02

Chapter 3.

Association rules mining

Introduction

We recall that data mining tasks fall into two main categories:

- **Prediction Methods:** Aim to use known variables to predict unknown or future values of other variables. The common techniques include: classification, regression, outlier detection.
- **Descriptive Methods:** Focus on discovering patterns that are understandable and meaningful to humans. The common techniques include: Clustering, Association Rule Mining, Sequential Pattern Discovery.

In the past few years, extensive research was done in the database community on learning rules using exhaustive search under the name of association rule mining. The objective there is to find all rules in data that satisfy the user-specified minimum support and minimum confidence.

For example, if a customer buys milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers to do selective marketing and plan their shelf space. For instance, placing milk and bread within close proximity may further encourage the sale of these items together within single visits to the store.

How can we find association rules from large amounts of data, where the data are either transactional or relational? Which association rules are the most interesting? How can we help or guide the mining procedure to discover interesting associations?

3.1. Basic Concepts

Association rule mining aims to find frequent patterns called associations, among sets of items or objects in transaction databases, relational databases, and other information repositories.

Rule general form:

Body$\Rightarrow$ Head [support, confidence]

With: Body is the antecedent (the "if" part of the rule), Head is the consequent (the "then" part), and support, confidence are two key metrics.

Example:

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. In the Market basket analysis, each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently associated or purchased together. These patterns can be represented in

the form of association rules. For example, the information that customers who purchase milk also tend to buy bread at the same time is represented in the following association rule:

milk ⇒bread [0.5%, 60%]

where:

- Support = 0.5% of all transactions in the dataset include both milk and bread.
- Confidence = 60% Among the customers who bought milk, 60% also bought bread. This shows how strong the rule is (i.e., how likely it is that bread is bought when milk is bought).

Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. These thresholds can be a set by users or domain experts. Additional analysis can be performed to discover interesting statistical correlations between associated items.

Let I = {I1, I2,..., Im} be an itemset (a set of items). Let D, the task-relevant data, be a set of database transactions where each transaction T is a nonempty itemset (or subset of items) such that $T \subseteq I$. Each transaction is associated with an identifier, called a TID. The database of transactions can be represented by the following table:

TID	**Transaction**
1	I1, I2, I11, I20
2	I1, I2, I3, I20, I11, I9
3	I1, I2
4	I4, I5, I9

Let A be a set of items. A transaction T is said to contain A if $A \subseteq T$. An association rule is an implication of the form A⇒B, where A⊂I, B⊂I, and A∩B = ∅.

The rule A⇒B holds in the transaction set D with support s, where s is the percentage of transactions in D that contain A∪B (i.e., the union of sets A and B say, or, both A and B). This is taken to be the probability, P(A∪B).

Support(A⇒B)= P(A∪B)

The rule A⇒B has confidence c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B/A).

Confidence(A⇒B)= P(B/A) with P(B/A)=$\frac{P(A \cap B)}{P(A)}$.

Rules that satisfy both a minimum support threshold (min sup) and a minimum confidence threshold (min conf) are called strong. By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an itemset. An itemset that contains k items is a k-itemset. The set {milk, bread} is a 2-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known, simply, as the frequency, support count, or count of the itemset. Note that the itemset support Support(A⇒B) is

sometimes referred to as relative support, whereas the occurrence frequency is called the absolute support. If the relative support of an itemset I satisfies a prespecified minimum support threshold (i.e., the absolute support of I satisfies the corresponding minimum support count threshold), then I is a frequent itemset. The set of frequent k-itemsets is commonly denoted by Lk.

confidence(A⇒B) = P(B/A)=$\frac{support(A \cup B)}{Support(A)}$=$\frac{support\_count(A \cup B)}{Support\_count(A)}$ …… (3)

The above equation shows that the confidence of rule A⇒B can be easily derived from the support counts of A and A∪B. That is, once the support counts of A, B, and A∪B are found, it is straightforward to derive the corresponding association rules A⇒B and B⇒A and check whether they are strong. Thus, the problem of mining association rules can be reduced to that of mining frequent itemsets.

In general, association rule mining can be viewed as a two-step process:

1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min sup.
2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence. Additional interestingness measures can be applied for the discovery of correlation relationships between associated items,

Because the second step is much less costly than the first, the overall performance of mining association rules is determined by the first step.

Association rules can be classified in various ways, based on the following criteria:

1. The types of values handled in the rule: If a rule concerns associations between the presence or absence of items, it is a Boolean association rule. For example, milk ⇒bread [0.5%, 60%]. If a rule describes associations between quantitative items or attributes, then it is a quantitative association rule. In these rules, quantitative values for items or attributes are partitioned into intervals. Example: age(X,"30-34")^ income(X,"42K - 48K") ⇒ buys(X,"high resolution TV"). Note that the quantitative attributes, age and income, have been discretized.
2. The dimensions of data involved in the rule: If the items or attributes in an association rule each reference only one dimension, then it is a single- dimensional association rule. Example of Boolean rule could be rewritten as buys(X, "milk") ⇒ buys(X, "bread") is therefore a single-dimensional association rule since it refers to only one dimension, i.e., buys. If a rule references two or more dimensions, such as the dimensions buys, time of transaction, and customer category, then it is a multidimensional association rule. The previous example of quantitative association is considered a multidimensional association rule since it involves three dimensions, age, income, and buys.

3.2. Methods for finding frequent itemsets

We present two common methods for frequent pattern mining: Apriori and FP-Growth.

3.2.1. Apriori algorithm

The Apriori algorithm introduced in 1994 by Rakesh Agrawal and Ramakrishnan Srikant. It is an influential algorithm for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties.
Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k+1)-itemsets. First, the set of frequent 1-itemsets is found. This set is denoted L1, which is used to find L2, the frequent 2-itemsets, which is used to find L3, and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database. To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property, presented below, is used to reduce the search space.

Apriori property. All non-empty subsets of a frequent itemset must also be frequent. This property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, s, then I is not frequent, i.e., P(I) < s. If an item A is added to the itemset I, then I∪A cannot occur more frequently than I. This property belongs to a special category of properties called anti-monotone in the sense that if a set cannot pass a test, all of its supersets will fail the same test as well. It is called anti-monotone because the property is monotonic in the context of failing a test.

Apriori algorithm works as follows:
Each of the key steps in the Apriori algorithm looks to identify itemsets and all their possible supersets looking for the most frequent to create the association rules.

Step 1: Frequent itemsets generation: The algorithm first identifies the unique items, sometimes referred to as 1-itemsets, in the dataset along with their frequencies. Then, it combines the items that appear together with a probability above a specified threshold into candidate itemsets and filters out the infrequent itemsets to reduce the compute cost in further steps. This process, known as frequent itemset mining, looks just for itemsets with meaningful frequencies.

Step 2: Expand and then prune itemsets: Using the Apriori property, the algorithm combines frequent itemsets further to form larger itemsets. The larger itemset combinations with a lower probability are pruned. This further reduces the search space and makes the computation more efficient.

Step 3: Repeat steps 1 and 2: The algorithm repeats steps 1 and 2 until all frequent itemsets meeting the defined threshold probability are generated exhaustively. Each iteration generates more complex and comprehensive associations in the itemsets. Frequent itemset generation is still computationally expensive.

Once Apriori has created the itemsets the strength of the generated associations and relationships can be investigated. This can be done using confidence (see equation 3). The conditional probability is expressed in terms of itemset support count, where support count(A∪B) is the number of transactions containing the itemsets A∪B, and support

count(A) is the number of transactions containing the itemset A. Based on this equation, association rules can be generated as follows:

- For each frequent itemset l, generate all nonempty subsets of l.
- For every nonempty subset s of l, output the rule:

  s⇒(l−s) if $\frac{support\_count(l)}{Support\_count(s)}$≥ min_conf, where min_conf is the minimum confidence threshold.

Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

To make applying the Apriori algorithm more efficient it is often combined with other association rule mining techniques. Two of the most common are the **FP-growth** algorithm and its variant **FP-Max** to reduce memory and computation constraints. The Apriori algorithm can also be combined with decision trees, where the Apriori algorithm identifies the frequent itemset, and the decision tree technique helps identify the association rules.

Another popular variant of the Apriori algorithm is **Dynamic Itemset Counting** (DIC) which starts counting potential itemsets early, without waiting for all the transactions to be recorded. DIC divides the dataset into smaller segments and processes each segment separately. This segmentation enables early stopping when the algorithm is not able to identify any frequent itemsets, but the partitioning of the data also helps reduce the compute cost significantly.

Apriori algorithms can also be useful in unsupervised learning-based artificial intelligence applications like clustering algorithms when the data supports it. It helps identify relationships and associations between seemingly independent entities, grouping them into possible clusters.

3.2.2. Frequent-Pattern-Growth Approach

Most of the previous studies adopt an Apriori like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there are a large number of patterns. The Pattern-Growth approach is a method used for frequent pattern mining without generating candidate sets like the Apriori algorithm does. Instead of generate and test, it adopts a divide-and-conquer strategy.

First, FP-growth compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, Efficient Frequent Itemset Mining Methods, which retains the itemset association information.

Then, it divides the compressed database into a set of conditional databases (a special kind of projected database); each associated with one frequent item or “pattern fragment,” and mines each such database separately.

The approach consists of two steps:

Step 1. Build an FP-Tree: The first scan of the database is the same as Apriori, which derives the set of frequent items (1- itemsets) and their support counts (frequencies). An FP-tree is then constructed as follows. First, create the root of the tree, labeled with “null.” Scan database D a second time. The items in each transaction are processed in L order (i.e., sorted according to descending support count), and a branch is created for each transaction.

Step 2. Mine the FP-Tree Recursively: The FP-tree is mined as follows. Start from each frequent length-1 pattern (as an initial suffix pattern), construct its conditional pattern base (a “sub-database,” which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then construct its (conditional) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

A study of the FP-growth method performance shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm.

3.3. Rule evaluation metrics

Beyond Support and Confidence, in association rule mining, after finding rules, we often need additional metrics to better judge their interestingness. Three popular ones are Lift, Conviction, and Leverage.

3.3.1. Lift

Lift measures how much more often the antecedent and consequent occur together than if they were statistically independent.

$$Lift(A \Rightarrow B) = \frac{support(A \cup B)}{Support(A) \times Support(B)}$$

With:

- Lift > 1: Positive association between A and B (they occur together more often than random chance).
- Lift = 1: A and B are independent (no association).
- Lift < 1: Negative association (A and B occur together less often than expected).

Example: If lift = 1.5, it means "A and B happen together 1.5 times more often than if they were independent."

3.3.2. Conviction

Conviction measures the degree of implication of the rule, considering how often A occurs without B.

$$Conviction\ (A \Rightarrow B) = \frac{1 - support(B)}{1 - Confidence(\mathrm{A} \Rightarrow \mathrm{B})}$$

Where:

- Conviction is high when the rule has high confidence and B rarely occurs without A.
- Conviction = 1 means the antecedent and consequent are independent.
- Higher conviction means stronger evidence that A implies B.

Example: A conviction of 3 means that "the rule would be wrong one-third as often as if A and B were independent."

3.3.3. Leverage

Leverage measures the difference between the observed frequency of A and B appearing together and what would be expected if A and B were independent.

$$Leverage\ (A \Rightarrow B) = Support(A \cup B) - (Support(A) \times Support(B))$$

With:

- Leverage = 0 means no correlation (independent).
- Positive leverage indicates positive correlation (occur more often together).
- Negative leverage indicates negative correlation (occur less often together).

Example: A leverage of 0.08 means "8% more transactions contain both A and B than what we would expect if they were independent."

3.4. Applications

Market basket analysis: One of the most common applications of the rule association mining techniques is performing market basket analysis. Retailers analyze customer purchase history and optimize the way stores are laid out by placing frequently purchased items near each other or on the same shelf. E-commerce platforms use Apriori algorithms to study product-based relationships based on user preferences and purchase pattern mining analysis to create efficient customer recommendation systems. The same kind of analysis can be used to optimize the purchase of services, for example, choosing training courses from a catalog, or recommending other types of coverage when selecting insurance.

Healthcare: The ARM techniques can be used to find strong association rules between symptoms and diseases to improve the efficiency of diagnosis and devise targeted treatment plans. For example, which patients are likely to develop diabetes or the role that diet or lifestyles play in disease. They can also help identify factors associated with adverse drug reactions.

Web analytics: The ARM techniques are also applicable in nontransactional databases. Data analysts will often use Apriori for web usage mining, to analyze clickstream data, and to interpret user behavior.

Finance: Another common application of the ARM techniques is to identify fraudulent patterns in financial transactions. Identifying specific purchase patterns as being possibly fraudulent allows a financial institution to act quickly to suspend transactions or contact an account holder.

Conclusion

Association rule mining is a powerful descriptive data mining technique used to uncover meaningful relationships among items in large datasets. This chapter introduced the fundamental concepts of association rules, including support, confidence, and additional interestingness measures such as lift, conviction, and leverage. We presented the two main steps of the mining process: discovering frequent itemsets and generating strong rules.

Two major algorithms were discussed: Apriori, which relies on candidate generation and the Apriori property, and FP-Growth, which avoids candidate generation by using an efficient FP-tree structure. FP-Growth is generally faster and more scalable, especially for dense datasets.

Finally, we highlighted several practical applications of association rule mining in domains such as retail, healthcare, web analytics, and finance. Overall, association rule mining provides valuable insights for decision-making by revealing hidden patterns and correlations within data.

PRACTICAL WORK 03

ASSOCIATION RULES MINING

1. Aim of the practice

Understand and implement the main algorithms for finding frequent itemsets in a transactional dataset:

- Apriori algorithm.
- Pattern-Growth approach.

Exercise 01. Creation of transactional dataset

Transaction ID	Items
T1	Milk, Bread, Butter
T2	Juice, Bread
T3	Milk, Bread, Juice
T4	Bread, Butter
T5	Milk, Bread, Butter, Juice

Exercise 02. One-Hot Encoding

```
from mlxtend.frequent_patterns import apriori, fpgrowth
from mlxtend.frequent_patterns import association_rules
import pandas as pd
```

Convert the transactional dataset into a format suitable for frequent itemset mining.

- List all unique items in the dataset.
- Create a one-hot encoded DataFrame where each column represents an item and each row a transaction.
- Print the resulting DataFrame.

Exercise 03. Apriori Algorithm

Find frequent itemsets using the Apriori algorithm.:

- Set a minimum support threshold of 60% (0.6).
- Apply the Apriori algorithm to find frequent itemsets.
- Print all frequent itemsets found.
- (Optional) Generate association rules from the frequent itemsets and show support, confidence, and lift.

Exercise 04: FP-Growth Algorithm

Find frequent itemsets using the FP-Growth algorithm.

- Apply the FP-Growth algorithm on the same dataset with minimum support 60%.
- Print all frequent itemsets found.
- (Optional) Generate association rules from FP-Growth frequent itemsets.
- Compare the results with the Apriori algorithm.

Exercise 05: Analysis

Analyze and interpret the results:

- Compare the number of frequent itemsets generated by Apriori and FP-Growth.
- Discuss which algorithm is more efficient for this dataset and why.

Solution of practical work 03.

Chapter 4.

Classification

Introduction

Classification plays a vital role in data mining and involves assigning predefined labels to data instances based on their characteristics or attributes. It consists of predicting certain outcome based on a given input. In order to predict the outcome, the algorithm processes a training set containing a set of attributes and the respective outcome, usually called **target** or **prediction attribute**. The algorithm tries to discover relationships between the attributes that would make it possible to predict the outcome. Next the algorithm is given a data set not seen before, called prediction set, which contains the same set of attributes, except for the prediction attribute, not yet known. The algorithm analyses the input and produces a prediction. The prediction accuracy defines how "good" the algorithm is. For example, in a medical database the training set would have relevant patient information recorded previously, where the prediction attribute is whether or not the patient had a heart problem.
Classification is a form of data analysis that extracts models describing important data classes.

4.1. Basic concepts

Classification is a machine learning method that is applied to foretell the clusters that data instances belong.

4.1.1. Labeled dataset

While unlabeled data consists of raw inputs with no designated outcome, labeled data is precisely the opposite. Labeled data is carefully annotated with meaningful tags, or labels, that classify the data's elements or outcomes. For example, in a dataset of emails, each email might be labeled as "spam" or "not spam." These labels then provide a clear guide for a machine learning algorithm to learn from.
A tuple, X, is represented by n-dimentional attribute vector, X= (x_1, x_2, …, x_n), depicting n measurements made on the tuple from n database attributes, respectively, A_1, A_2, …, A_n. Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute, which is discrete-value and unordered. It is categorical or nominal in that each value serves as a category or class.

A labeled data set refers to a collection of data where each data point is assigned a specific label or category. Labeling data can be a lengthy, resource-intensive and costly process, particularly for complex data such as images. For example, manual annotation of a single radiology image can take a significant amount of time, especially if it requires a specialist's knowledge. There are three approaches to Data Labeling:

1. **Manual Labeling:** Involves humans labeling data directly. It is highly accurate but slow and costly for large datasets.

2. **Semi-Automated Labeling:** Uses machine learning to label data first, with humans refining the results. It is faster than manual labeling but may inherit algorithmic errors. Universal Data Tool can be used on various platforms to create and label datasets consisting of images, audio, text, videos, and documents. Sloth is a tool for labeling image and video data for computer vision research. Doccano offers easy-to-use annotation tools for text classification, sequence labeling, and sequence-to-sequence tasks.
3. **Crowdsourcing:** Distributes labeling tasks to many people via platforms like Mechanical Turk. It is inexpensive, but quality can be inconsistent due to varying expertise levels.

4.1.2. Classifier

A classifier is a type of machine learning model or algorithm that assigns labels or categories to data based on input features. It is a core component in supervised learning, where the model is trained on labeled data to learn how to categorize new, unseen data. Common types of Classifiers: Logistic Regression (Good for binary classification), Decision Trees (Simple, interpretable tree-like models), Random Forests (Ensemble of decision trees for improved accuracy), Support Vector Machines (SVM) to find the optimal boundary between classes, Naive Bayes (efficient for text classification), K-Nearest Neighbors (KNN) Classifies based on the closest training examples, Neural Networks (used in deep learning), and XGBoost/LightGBM (Gradient boosting models) that are high-performing for structured data. Classifiers are central to supervised learning tasks such as spam detection, fraud detection, and image recognition.

4.1.3. Model selection

Model Selection is the process of choosing the most appropriate machine learning model from a set of candidates based on their performance on a given task. It involves comparing different models using evaluation metrics (e.g., accuracy, precision, recall, F1-score) on validation data. The goal is to select the model that generalizes best to unseen data, not just the one that performs best on the training set. Techniques like cross-validation, grid search, or random search are commonly used during model selection to ensure robust evaluation.

4.1.4. Overfitting

In order to discuss this phenomenon more formally, we need to differentiate between training error and generalization error. The training error is the error of our model as calculated on the training dataset, while generalization error is the expectation of our model's error were, we apply it to an infinite stream of additional data examples drawn from the same underlying data distribution as the original sample. Problematically, we can never calculate the generalization error exactly. That is because the stream of infinite data is an imaginary object.

Overfitting is a modeling error in machine learning where a model learns the training data too well including its noise and outliers resulting in poor generalization to new, unseen data. An overfitted model performs very well on the training set but poorly on the validation or test set because it has essentially memorized the training data rather than learning the underlying patterns. Overfitting is characterized by large gap between training and

validation performance. Common causes may be related to very complex a model, insufficient training data or too many training epochs (in iterative models).
It performs very well on training data but poorly on test data. Overfitting happens due to Model too complex, Too many features, Very little data, No regularization, High variance. The techniques used to combat overfitting are called regularization, such as:

- **L1 regularization (Lasso):** adds a penalty proportional to the absolute value of the weights, encouraging sparsity.
- **L2 regularization (Ridge):** adds a penalty proportional to the square of the weights, preventing large coefficients.
- **Dropout:** randomly disactivates neurons during training to prevent co-adaptation in neural networks.
- **Early stopping:** halts training when validation performance stops improving.
- **Data augmentation:** increases the variety of the training data to improve generalization.
- **Ensemble methods:** combine multiple models to reduce variance.
- **Pruning** (in decision trees or neural networks): removes unnecessary branches or nodes to reduce model complexity.

4.1.5. Underfitting

The opposite situation, underfitting happens when the model fails to learn important patterns. It performs poorly on both training and testing data. Underfitting happens due to: model is too simple, very high regularization, features are weak or missing, not enough training, and high bias. The techniques used to address underfitting include:

- Increasing model complexity: using a more expressive model (e.g., deeper neural networks, more tree depth).
- Adding more relevant features: incorporating informative attributes that help the model learn richer patterns.
- Reducing regularization strength: lowering L1/L2 penalties to allow the model to fit the data more flexibly.
- Training for more epochs: allowing the model to continue learning instead of stopping too early.
- Feature engineering: transforming or combining existing features to reveal hidden relationships.

4.2. Process of classification

Classification in machine learning typically involves two main steps:

Training Phase (Model construction): In this step, the classifier learns from a labeled dataset. Each input is paired with a known output (features + labels), and the model identifies patterns or decision boundaries that best separate the classes. This first step of the classification process can also be seen as learning a mapping or a function $y = f(x)$ that can predict the class label y associated with a given tuple x. The model is represented as classification rules, decision Trees, or mathematical formula.

Prediction Phase (Model usage): After training, the model is used to classify new, unseen data by predicting the most likely class based on what it has learned. The input is unlabeled data (features only) and the output is a predicted label(s). First, the predictive accuracy of the classifier is estimated. If the training set were used to measure the classifier's accuracy, the estimate would likely be overly optimistic because the classifier tends to overfit the data. That is, during training, it may incorporate certain errors or anomalies specific to the training data that are not present in the overall dataset.
Therefore, a test set is used, consisting of test tuples and their associated class labels. These tuples are independent of the training data, meaning they were not used to build the classifier. The accuracy of a classifier on a given test set is the percentage of test tuples that are correctly classified by the classifier. The actual class label of each test tuple is compared to the class predicted by the learned classifier for that tuple.
If the classifier's accuracy is deemed acceptable, it can then be used to classify future data tuples for which the class label is unknown.

4.3. Classification techniques

Classification techniques are fundamental tools in machine learning and data mining, used to categorize data into predefined classes based on input features. This section outlines several widely used classification algorithms, each with unique strengths and assumptions.

4.3.1. Traditional methods

4.3.1.1. Decision trees

A decision tree is a set of conditions organized in a hierarchical structure starts with one main question at the top called a node which further branches out into different possible outcomes where:

- **Root Node** is the starting point that represents the entire dataset.
- **Branches:** These are the lines that connect nodes. It shows the flow from one decision to another.
- **Internal Nodes** are points where decisions are made based on the input features.
- **Leaf Nodes:** These are the terminal nodes at the end of branches that represent final outcomes or predictions

A decision tree is a predictive model in which an instance is classified by following the path of satisfied conditions from the root of the tree until reaching a leaf, which will correspond to a class label. It can easily be converted to a set of classification rules. A decision tree for the concept student buys computer can be represented by the following decision tree.

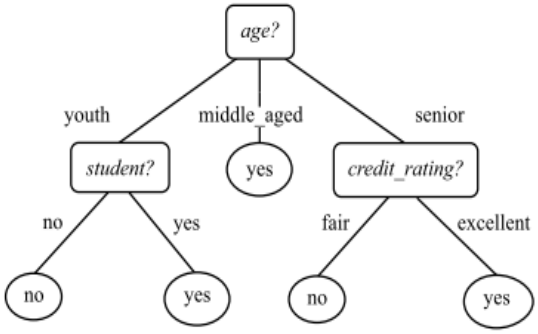


Figure 4.1. Example of decision tree model[6]

The decision tree occurs in two main stages: **Tree growth**, at a training set, and **tree pruning**, which occurs when the size of the tree is decreased for easy understanding. Some types of decision trees are ID3, C45, and CART. The decision tree works through the classification of instances by arranging them from the root to some leaf node. Each node in the tree indicates a test of some characteristic of the instance, and every branch that inclines from the node matches a probable equivalent of the feature.

Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data partition D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- attribute list, the set of candidate attributes;
- Attribute selection method, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a splitting attribute and, possibly, either a split point or splitting subset.

Output: A decision tree.

Classification starts by starting a root node of the tree, then checking the characteristics for the specified feature of the tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple.

ID3 is also known as Iterative Dichotomiser 3, and it is a common algorithm used in machine learning. It was originally developed by J. Ross Quinlan at the University of Sydney, and he first presented it in the 1975 book "Machine Learning". One of its advantages is that it is easy to understand. This is admitted by Quinlan (1986), who posited that its popularity is due to its efficiency and simple use. On the contrary, it cannot be used to handle missing values and does not have a backtracking search and global optimization.

Similarly, C4.5, as an extended version of the ID3 algorithm, reduces the weaknesses found in ID3. While ID3 uses information gain as the splitting criterion and is limited to categorical features, C4.5 improves upon it by handling both continuous and categorical

[6] Barros, R. C., de Carvalho, A. C., & Freitas, A. A. (2020). Decision-tree induction. In Automatic Design of Decision-Tree Induction Algorithms (pp. 7-45). Cham: Springer International Publishing.

attributes, managing missing values, and using gain ratio to overcome the bias of information gain toward attributes with many values. Additionally, C4.5 incorporates pruning mechanisms to reduce overfitting by removing branches that have little predictive power. These enhancements make C4.5 more robust and suitable for practical applications involving complex and noisy datasets.

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. If the splitting attribute is continuous-valued or if we are restricted to binary trees then, respectively, either a split point or a splitting subset must also be determined as part of the splitting criterion. The tree node created for partition D is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly. This section briefly describes three popular attribute selection measures: information gain, gain ratio, and gini index.

Information gain: This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node N represents or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. The expected information needed to classify a tuple in D is given by:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

Where p_i is the non-zero probability that an arbitrary tuple in D belongs to class C_i and is estimated by $\frac{|C_i, \mathrm{D}|}{|D|}$. A log function to the base 2 is used, because the information is encoded in bits. Info(D) is just the average amount of information needed to identify the class label of a tuple in D.

Suppose we were to partition the tuples in D on some attribute A having v distinct values, $\{a_1, a_2, \ldots, a_v\}$, as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, $\{D_1, D_2, \ldots, D_v\}$, where D_j contains those tuples in D that have outcome a_j of A. How much more information would we still need (after the partitioning) in order to arrive at an exact classification? This amount is measured by:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

The term $\frac{|D_j|}{|D|}$ acts as the weight of the j^{th} partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions. Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D)$$

Gain ratio: The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. C4.5, uses an extension to information gain known as gain ratio, which attempts to overcome this bias. It applies a kind of normalization to information gain using a "split information" value defined analogously with Info(D) as:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times log_2\left(\frac{|D_j|}{|D|}\right)$$

The gain ratio is defined as:

$$GainRation(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

The Gini index is used in CART method. Using the notation described above, the Gini index measures the impurity of D, a data partition or set of training tuples, as:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

Where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $\frac{|C_{i,}\,\mathrm{D}|}{|D|}$. The sum is computed over m classes.

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D_1 and D_2, the gini index of D given that partitioning is:

$$Gini_A(D) = \frac{|D_1|}{D} Gini(D_1) + \frac{|D_2|}{D} Gini(D_2)$$

For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described above for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point. The point giving the minimum Gini index for a given (continuous-valued) attribute is taken as the split-point of that attribute. Recall that for a possible split-point of A, D_1 is the set of tuples in D satisfying A ≤ split point, and D_2 is the set of tuples in D satisfying A>split point. The reduction in impurity that would be incurred by a binary split on a discrete or continuous-valued attribute A is:

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. There are two common approaches to tree pruning: prepruning and postpruning.

In the prepruning approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node).

The second and more common approach is postpruning, which removes sub-trees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and

replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.

4.3.1.2. K-Nearest Neighbors

K-Nearest Neighbors (K-NN) is a simple, instance-based learning algorithm that classifies a data point based on the majority label among its k closest neighbors in the feature space. The closeness is typically measured using distance metrics such as Euclidean or Manhattan distance. K-NN is non-parametric and makes no assumptions about the underlying data distribution, which makes it versatile. However, it can be computationally expensive, especially on large datasets, and its performance depends heavily on the choice of k and the distance metric.

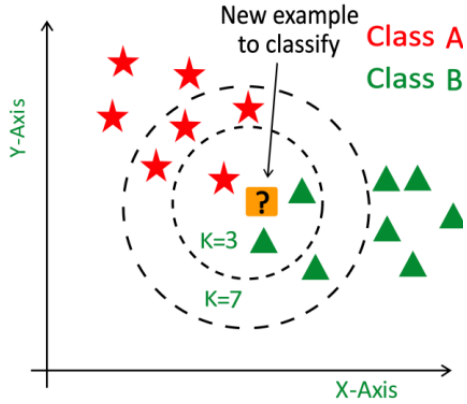


Figure 4.2. Example of K-NN model[7]

In the diagram depicted in Figure 4.2, we need to determine which class the new example should be grouped with. K=3 means the new example will be identified as a Class B member. However, if K=7, the new example will belong to Class A.

Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

While the KNN algorithm can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

For classification problems, a class label is assigned on the basis of a majority vote i.e. the label that is most frequently represented around a given data point is used. While this is technically considered “plurality voting”, the term, “majority vote” is more commonly used in literature.

Regression problems use a similar concept as classification problem, but in this case, the average the k nearest neighbors is taken to make a prediction about a classification. The main distinction here is that classification is used for discrete values, whereas regression is

[7] https://medium.com/@kenzaharifi/bien-comprendre-lalgorithme-des-k-plus-proches-voisins-fonctionnement-et-impl%C3%A9mentation-sur-r-et-a66d2d372679

used with continuous ones. However, before a classification can be made, the distance must be defined. Euclidean distance is most commonly used, which we'll delve into more below.

To classify a new point, the algorithm calculates the "distance" between the new point and all other points in the dataset. The most common way to measure this distance is by using the Euclidean distance and Manhattan Distance. Once the distances are calculated, the algorithm takes a majority vote from the K-nearest points and classifies the new point accordingly.

$$Euclidian = \sqrt{\sum_{i=1}^{n} \left(x_1^i - x_2^i\right)^2}$$

$$Manhattan = \left|\sum_{i=1}^{n} x_1^i - x_2^i\right|$$

One important step in using KNN is feature scaling. If one feature is much larger in value than another, it can dominate the distance calculation. For example, if we're using both age and income as features to classify a person's group, the income in thousands could skew the results. To avoid this, we normalize the data so that each feature has equal weight.

Algorithm Steps

Data: A set of *m* labeled training instances: $(x_1,c(x_1)),(x_2,c(x_2)),...,(x_m,c(x_m))$

Input: A new instance *y* to classify, *k: integer* (number of neighbors)

Compute Distances: Calculate the distance between y and all training instances (e.g., Euclidean distance).

Find Neighbors: Identify the k closest instances to y (i.e., those with the smallest distances).

Vote for Class: Aggregate the classes of the k neighbors (e.g., by majority vote).

Optionally, weight votes by distance (closer neighbors have more influence).

Assign Class: Assign y the class most common among its k nearest neighbors:

$c(y)=majority\_vote(c(x_1),...,c(x_k))$

Output: Predicted class label c(y) for the instance y

K-Nearest Neighbors (KNN) is a simple and intuitive algorithm, but it faces several key challenges. It can be computationally expensive and slow at prediction time, especially with large datasets, since it requires comparing a new instance to all training samples. KNN also suffers from the curse of dimensionality, where distance measures become less meaningful in high-dimensional spaces, reducing classification accuracy. Additionally, the algorithm is highly sensitive to the choice of distance metric and feature scaling, and it performs poorly when irrelevant or highly correlated features are present.

Choosing the right value for *k* is crucial, as too small a value can be led to overfitting while too large a value may cause underfitting. KNN also struggles with imbalanced data, often favoring the majority class, and it requires substantial memory as it stores the entire training

set. These limitations make KNN less suitable for large-scale or high-dimensional applications without careful preprocessing and parameter tuning.

4.3.1.3. Naïve Bayes

Naïve Bayes is a probabilistic classifier based on **Bayes' Theorem**, with the "naïve" assumption that all features are conditionally independent given the class label. Despite this strong and often unrealistic assumption, Naïve Bayes performs surprisingly well in many real-world scenarios, especially with text classification problems such as spam detection and sentiment analysis. It is highly scalable, requires a small amount of training data, and is robust to irrelevant features.

Bayes' Theorem, named after 18th-century British mathematician Thomas Bayes, is a mathematical formula for determining conditional probability.
Conditional probability is the likelihood of an outcome occurring based on a previous outcome in similar circumstances. Bayes' Theorem provides a way to revise existing predictions or theories (update probabilities) given new or additional evidence.
Bayes' theorem relies on incorporating prior probability distributions in order to generate posterior probabilities.

In Bayesian terms, prior probability is the probability of an event occurring before new data is collected. In other words, it represents the best rational assessment of the probability of a particular outcome based on current knowledge before an experiment is performed. Let X, be an "evidence.", which is described by measurements made on a set of n attributes. Let H be some hypothesis, such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine $P(H/X)$, the posterior probability or a posteriori probability, of H conditioned on X.
$P(H)$ and $P(X)$ represent the prior, or a priori, probabilities of H and X, respectively.

Posterior probability is the revised probability of an event occurring after considering the new information. Posterior probability is calculated by updating the prior probability using Bayes' theorem. P(X|H) is the posterior probability of X conditioned on H. Bayes' theorem is useful in that it provides a way of calculating the posterior probability, P(H|X), from P(H), P(X|H), and P(X). Bayes' theorem is:

$$P(H/X) = \frac{P(X/H)P(H)}{P(X)}$$

Naïve Bayes Classification begins with data preprocessing, followed by computing the prior probabilities P(C) of the class label C, the likelihoods $P(x_i/C)$, and the posterior probabilities $P(C/X)$ then selects the class with the highest posterior
Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by n-dimensional attribute vector, X =(x_1, x_2… x_n), depicting n measurements made on the tuple from n attributes, respectively, A_1, A_2… A_n.

Calculate prior probabilities: For each class C, compute the prior probability.

$$P(C) = \frac{\text{Number of samples in class C}}{\text{Total number of samples}}$$

Calculate likelihood probabilities for each feature given the class

$$P(x_i/C) = \frac{\text{Number of times feature xi appears in class C}}{\text{Total samples in class C}}$$

Apply Bayes' Theorem to compute the posterior probability for each class:

$$P(X/C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

Ignore P(X) because it's constant across classes for classification.

Select the class with the highest posterior probability: Assign the input instance X to the class with the maximum $P(C/X)$

Various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains. In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case, owing to inaccuracies in the assumptions made for its use, such as class conditional independence, and the lack of available probability data.

4.3.1.4. Support Vector Machines (SVM)

Support Vector Machines are powerful, supervised learning models used for both classification and regression tasks. The core idea of SVM is to find the hyperplane that best separates the data into different classes with the maximum margin. For non-linearly separable data, SVMs use kernel functions (e.g., polynomial, radial basis function) to transform the input space into a higher-dimensional space where a separating hyperplane can be found. SVMs are effective in high-dimensional spaces and are particularly useful when the number of dimensions exceeds the number of samples. However, they can be sensitive to the choice of kernel and hyperparameters, and may not scale well to very large datasets.

In two-dimensional space, a hyperplane is simply a "line" that separates the data points into two classes. In three-dimensional space, a hyperplane is a "plane" that separates the data points into two classes. Similarly, in N-dimensional space, a "hyperplane" has (N-1)-dimensions.

It can be used to make predictions on new data points by evaluating which side of the hyperplane they fall on. Data points on one side of the hyperplane are classified as belonging to one class, while data points on the other side of the hyperplane are classified as belonging to another class.

Figure 4.3. illustrates the SVM technique.

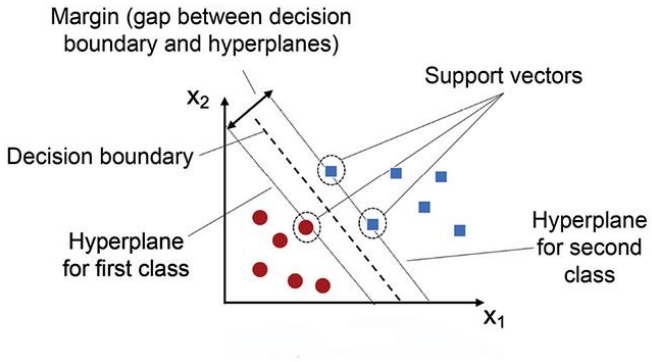


Figure 4.3. SVM Classifier[8]

A larger margin indicates a greater degree of confidence in the classification, as it means that there is a larger gap between the decision boundary and the closest data points from each class. The margin is a measure of how well-separated the classes are in feature space. SVMs are designed to find the hyperplane that maximizes this margin, which is why they are sometimes referred to as maximum-margin classifiers.

Support Vectors

They are the data points that lie closest to the decision boundary (hyperplane) in a Support Vector Machine (SVM). These data points are important because they determine the position and orientation of the hyperplane, and thus have a significant impact on the classification accuracy of the SVM. In fact, SVMs are named after these support vectors because they "*support*" or define the decision boundary. The support vectors are used to calculate the margin, which is the distance between the hyperplane and the closest data points from each class. The goal of SVMs is to maximize this margin while minimizing classification errors.

For a binary classification problem with:

Input features: $x_i \in \mathbb{R}^n$

Labels: $y_i \in \{-1, +1\}$

Classifier: $f(x) = w^T x + b$

Distance from a Data Point to the Hyperplane:

$$\hat{y} = \begin{cases} 1: f(x) \geq 0 \\ 0: f(x) < 0 \end{cases}$$

When data is not linearly separable (i.e., it can't be divided by a straight line), SVM uses a technique called **kernels** to map the data into a higher-dimensional space where it becomes separable. This transformation helps SVM find a decision boundary even for non-linear data.

Support Vector Machines (SVM) offer several advantages in classification tasks. One of their main strengths is their effectiveness in high-dimensional spaces, which makes them particularly useful when the number of features exceeds the number of samples. SVMs are also memory-efficient, as they use only a subset of the training data (the support vectors) to construct the decision boundary. Additionally, they are highly versatile thanks to the use of

[8] https://medium.com/low-code-for-advanced-data-science/

kernel functions, allowing them to handle both linear and non-linear classification problems. When properly regularized, SVMs are also less prone to overfitting and tend to generalize well, especially when the classes are clearly separable by a margin.
However, SVMs come with certain limitations. They are not well-suited for very large datasets, as the training time can be computationally intensive. SVMs can also perform poorly on noisy datasets where classes are not clearly separable or contain significant overlap.

4.3.1.5. Artificial Neural Network

Artificial Neural network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information. ANN is capable of predicting new observations from existing observations. A neural network consists of interconnected processing elements also called units, nodes, or neurons. The neurons within the network work together, in parallel, to produce an output function. Since the computation is performed by the collective neurons, a neural network can still produce the output function even if some of the individual neurons are malfunctioning (the network is robust and fault tolerant). The neuron receives input from some other neurons or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The neuron applies a function f to the weighted sum of its inputs as shown in Figure 4.4.

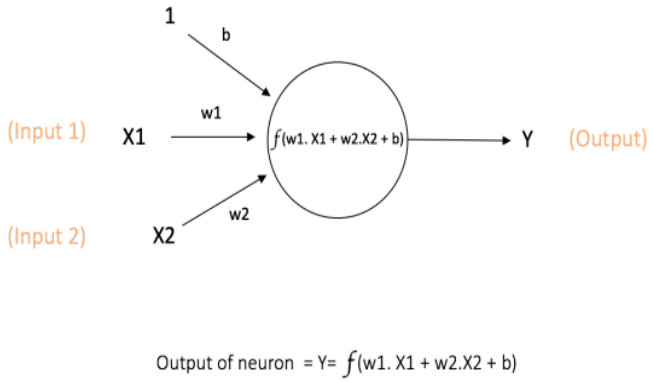


Figure 4.4. An example of neuron[9]

A neuron takes numerical inputs X1 and X2 and has weights w1 and w2 associated with those inputs. Additionally, there is another input 1 with weight b (called the Bias) associated with it, which works like an intercept in linear equations, enabling the neuron to produce an output even when all inputs are zero. This makes the model more flexible and powerful.
The function f is non-linear and is called the Activation Function. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real-world data are non-linear and we want neurons to learn these non-linear representations. Every activation function takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions, such as Sigmoid, tanh, and ReLU.

[9] https://medium.com/@ariesiitr/an-artificial-neural-network-ann-is-a-computational-model-that-is-inspired-by-the-way-biological-c17b07166d4c

Sigmoid: takes a real-valued input and squashes it to range between 0 and 1. It is often used in binary classification.

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

tanh: takes a real-valued input and squashes it to the range [-1, 1]. It preferred over sigmoid in hidden layers because it's zero-centered.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero).

$$ReLU(x) = max(0, x)$$

The neurons within a neural network are usually arranged in layers. The number of layers within the neural network, and the number of neurons within each layer normally matches the nature of the investigated phenomenon.

After the size has been determined, the network is usually then subjected to training. During the training phase, the network learns by iteratively adjusting the weights to accurately predict the correct class label for the input data. It also referred to as connectionist learning due to the connections between units.

During the training of deep learning models, the concepts of **epoch** and **batch** are essential for organizing the learning process. An **epoch** refers to one complete pass through the entire training dataset, where the model sees and learns from all samples once. Since using all the data at once can be computationally expensive, the dataset is divided into smaller groups called **batches**. After processing each batch, the model's weights are updated — this is called an **iteration**. For example, if the dataset contains 1,000 samples and the batch size is set to 100, then one epoch will consist of 10 batches. Repeating this process over multiple epochs helps the model gradually improve its prediction accuracy. After the training phase, the network is ready to perform predictions in new sets of data.

The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons (nodes) arranged in layers. Nodes from adjacent layers have connections or edges between them. All these connections have weights associated with them. The inputs to the network correspond to the attributes measured for each training tuple. Inputs are fed simultaneously into the units making up the input layer. They are then weighted and fed simultaneously to a hidden layer. The number of hidden layers is arbitrary, although usually only one. The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction. The network is feed-forward in that none of the weight’s cycles back to an input unit or to an output unit of a previous layer.

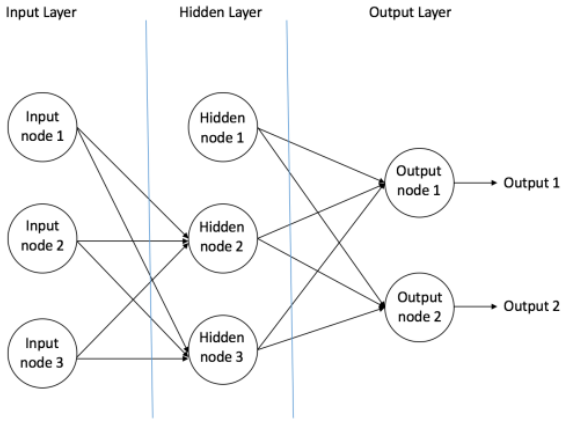


Figure 4.5. An example of feedforward neural network neuron[10]

For each training tuple, the weights are modified to minimize the mean squared error between the network's prediction and the actual target value. Modifications are made in the "backwards" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "backpropagation". Efficiency of backpropagation: Each epoch (one interation through the training set) takes O(|D| * w), with |D| tuples and w weights, but # of epochs can be exponential to n, the number of inputs, in the worst case.

Neural networks are powerful but are often criticized for being "black boxes" due to their lack of interpretability. This limits their usefulness in domains where understanding why a decision is made is as important as the decision itself.

4.3.2. Deep learning-based methods

Deep learning–based methods are a subset of machine learning techniques that rely on artificial neural networks with multiple hidden layers to automatically learn hierarchical representations of data. Unlike traditional methods that often depend on handcrafted features, deep learning models extract complex patterns directly from raw inputs through layered transformations. These methods, such as DNN, RNN, and CNN excel at modeling high-dimensional, nonlinear data and are widely used in domains like natural language processing, computer vision, speech recognition, and time-series analysis. Transformers are a modern deep learning architecture designed to handle sequential data without relying on recurrence or convolution. We briefly describe the main deep learning architectures.

4.3.2.1. DNN

A Deep Neural Network (DNN) is an artificial neural network characterized by multiple hidden layers between the input and output layers. Each layer is composed of interconnected neurons that apply linear transformations followed by nonlinear activation functions. By stacking many such layers, DNNs are capable of learning complex hierarchical representations of data, where lower layers capture simple features and deeper layers extract high-level abstractions. This depth enables DNNs to model highly nonlinear relationships and makes them effective for a wide range of tasks, including image recognition, natural language processing, and predictive analytics.

[10] https://medium.com/@ariesiitr/an-artificial-neural-network-ann-is-a-computational-model-that-is-inspired-by-the-way-biological-c17b07166d4c

Deep neural networks can be found in numerous healthcare sectors due to their effectiveness. They are currently applied from medical imaging, diagnosis, drug development, prognosis, and risk assessment, to remote monitoring and sports medicine. The largest number of recent studies report the use of DNNs in the analysis of radiological images, among which include: models detecting apparent and non-apparent scaphoid fractures using only plain wrist radiographs, algorithms for COVID-19 detection from CXR images, models for mammography screening, and tools for the segmentation of intracerebral haemorrhage on CT scans. DNNs are also involved in assessing endomyocardial biopsy data of patients with myocardial injury. Others can identify hypertension on the basis of ballistocardiogram signals or aortic stenosis using audio files.
Figure 4.6. shows a graphical representation of an artificial neural network (ANN) and a deep neural network (DNN).

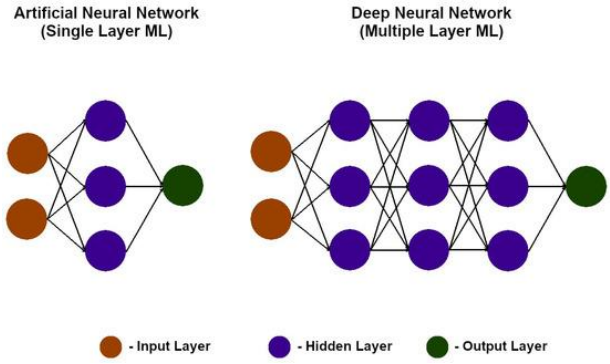


Figure 4.6. Difference between ANN and DNN[11]

4.3.2.2. RNN

Recurrent Neural Networks (RNNs) are a class of neural networks designed to model sequential data, where the current output depends not only on the present input but also on previous inputs. Unlike traditional feed-forward networks, RNNs contain recurrent connections that allow information to persist over time. At each time step, the network updates a hidden state that acts as a memory, capturing contextual information from earlier elements in the sequence. This makes RNNs suitable for tasks, for example, to understand the context of a word in a sentence, one needs to know the words that came before it. To handle this, the inputs are arranged in order of their sequence, and each time an input is passed to the model, it also receives information from the previous hidden layer. In a biological sense, the same concept can be applied to DNA and RNA sequences where we are interested in an ordered sequence of nucleotides or amino acids.
Figure 4.7 shows an example of a char RNN.

[11] McBee, M.P.; Awan, O.A.; Colucci, A.T.; Ghobadi, C.W.; Kadom, N.; Kansagra, A.P.; Tridandapani, S.; Auffermann, W.F. Deep Learning in Radiology. *Acad. Radiol.* 2018, *25*, 1472–1480.

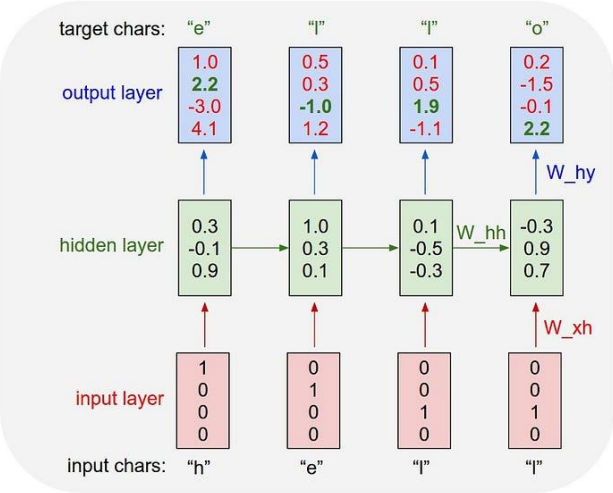


Figure 4.7. An example of a char RNN[12]

However, standard RNNs suffer from issues like vanishing and exploding gradients, which limit their ability to learn long-term dependencies, challenges addressed by advanced variants such as LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit).

4.3.2.3. CNN

Convolutional Neural Networks (CNNs) are a class of deep learning models specialized for processing data with a grid-like structure, such as images. CNN is a type of a feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the visual cortex. This neural network consists of multiple layers of receptive fields. It used convolutional layers that apply filters (kernels) to automatically extract local patterns, edges, textures, shapes, and gradually build more abstract representations in deeper layers. CNNs also include pooling layers to reduce spatial dimensions and fully connected layers for final predictions. Their ability to learn spatial hierarchies of features makes them highly effective for tasks like image classification, object detection, and computer vision applications. During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform. For example, an image classification CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.

An example of CNN architecture is depicted in Figure 4.8.

[12] https://camrongodbout.medium.com/recurrent-neural-networks-for-beginners-7aca4e933b82

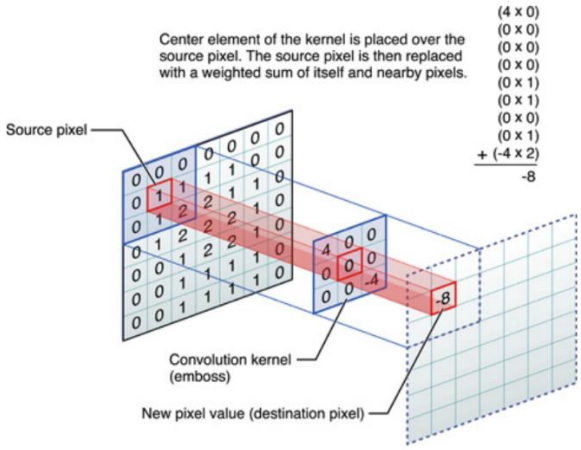


Figure 4.8. An example of CNN architecture[13]

4.3.2.4. Transformer

Transformers are an advanced deep learning architecture introduced to overcome the limitations of recurrent and convolutional models when processing sequential data. Unlike RNNs and LSTMs, which process sequences step by step, the Transformer relies entirely on a mechanism called self-attention to model relationships between all elements of a sequence in parallel.

The core idea of the Transformer is the self-attention mechanism, which computes how much each token in the sequence should contribute to the representation of every other token.

- Each input token is transformed into three vectors: Query (Q), Key (K), and Value (V).
- The attention score between two tokens is computed as a scaled dot-product between their Q and K vectors.
- These scores determine how much information each token receives from the others via the V vectors.

This allows the model to capture short- and long-range dependencies efficiently.

Multi-Head Attention: Instead of computing self-attention once, the Transformer uses multiple attention heads. Each head learns different types of relationships (e.g., syntactic roles, semantic similarity), giving the model richer contextual understanding.

Encoder–Decoder Structure: The original Transformer consists of two parts:

Encoder: A stack of identical layers, each containing

- Multi-head self-attention
- A feed-forward neural network
- Residual connections and layer normalization

Decoder: Similar structure but includes

- Self-attention
- Attention over encoder outputs (encoder–decoder attention)
- Feed-forward networks

[13] https://sites.google.com/view/cs502project/deep-learning-for-biology-a-tutorial/deep-learning-models-dnns-cnns-rnns-autoencoders

This encoder–decoder design is ideal for tasks such as translation, summarization, and question answering. Transformers allow parallel processing of entire sequences, unlike RNNs which process tokens sequentially. This leads to faster training, better scalability, and more effective learning from large datasets.

4.4. Evaluation and comparison of classifiers

4.4.1. Evaluation measures

Once a classification model is built, it is common for many questions to arise regarding its performance and reliability. The classifier evaluation measures are: Accuracy (also known as recognition rate), precision, recall, F1, sensitivity, and specificity. There are four additional terms we need to know. These terms are the "building blocks" used in computing many evaluation measures. Understanding these terms will make it easy to grasp the meaning of the various measures.

- **True positives (TP):** These refer to the positive tuples that were correctly labeled by the classifier. Let TP be the number of true positives.
- **True negatives (TN):** These are the negative tuples that were correctly labeled by the classifier. Let TN be the number of true negatives.
- **False positives (FP):** These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class buy computer = no for which the classifier predicted buys computer = yes). Let FP be the number of false positives.
- **False negatives (FN):** These are the positive tuples that were mislabeled as negative (e.g., tuples of class buy computer = yes for which the classifier predicted buys computer = no). Let FN be the number of false negatives.

These terms are summarized in the **confusion matrix** of Figure 4.9. The confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes. TP and TN tell us when the classifier is getting things right, while FP and FN tell us when the classifier is getting things wrong (i.e., mislabeling).

		Predicted class		
		Yes	No	Total
Actual class	Yes	**TP**	**FN**	P
	No	**FP**	**TN**	N
	Total	P'	N'	P + N

Figure 4.9. A confusion matrix, shown with totals for positive and negative tuples[14].

Given m classes (where $m \geq 2$), a confusion matrix is a table of at least size m by m. **The accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. That is,

$$accuracy = \frac{TP + TN}{P + N}$$

[14] Banerjee, S., & Bhowmik, P. S. (2025). Machine learning based classifiers for dynamic and transient disturbance classification in smart microgrid system. Measurement, 240, 115576.

The error rate or misclassification rate of a classifier, M, which is simply 1−Accuracy(M).

Precision can be thought of as a measure of exactness (that is, what percentage of tuples labeled as positive are actually such), whereas **recall** is a measure of completeness (what percentage of positive tuples are labeled as such). If recall seems familiar, that's because it is the same as sensitivity (or the true positive rate). These measures can be computed as:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P}$$

An alternative way to use precision and recall is to combine them into a single measure. This is the approach of the **F measure** (also known as the F1 score or F-score) and the F_β measure. They are defined as:

$$F = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

$$F_\beta = \frac{(1 + \beta^2) \times Precision \times Recall}{\beta^2 \times Precision + Recall}$$

Where β is a non-negative real number. The F measure is the harmonic mean of precision and recall. It gives equal weight to precision and recall. The F_β measure is a weighted measure of precision and recall. It assigns β times as much weight to recall as to precision. Commonly used F_β measures are F_2 (which weights recall twice as much as precision) and F0.5 (which weights precision twice as much as recall).

We now consider the class imbalance problem, where the main class of interest is rare. That is, the data set distribution reflects a significant majority of the negative class and a minority positive class. For example, in fraud detection applications, the class of interest (or positive class) is "fraud", which occurs much less frequency than the negative "non fraudulant" class. In medical data, there may be a rare class, such as "cancer". Suppose that you have trained a classifier to classify medical data tuples, where the class label attribute is "cancer" and the possible class values are "yes" and "no". Sensitivity is also referred to as the true positive (recognition) rate (that is, the proportion of positive tuples that are correctly identified), while specificity is the true negative rate (that is, the proportion of negative tuples that are correctly identified). These measures are defined as:

$$sensitivity = \frac{TP}{P}$$

$$specificity = \frac{TN}{N}$$

It can be shown that accuracy is a function of sensitivity and specificity:

$$accuracy = sensitivity \frac{P}{(P + N)} + specificity \frac{N}{(P + N)}$$

The accuracy measure works best when the data classes are fairly evenly distributed. Other measures, such as sensitivity (or recall), specificity, precision, F and F_β are better suited for the class imbalance problem, where the main class of interest is rare.

In the Hold-out method, the given data are randomly partitioned into two independent sets, a training set and a test set. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the model, whose accuracy is estimated with the test set. The estimate is pessimistic because only a portion of the initial data is used to derive the model. **Random subsampling** is a variation of the holdout method in which the hold-out method is repeated k times. The overall accuracy estimate is taken as the average of the accuracies obtained from iteration.

In k-fold cross-validation, the initial data are randomly partitioned into k-mutually exclusive subsets or "folds," D_1, D_2, …, D_k, each of approximately equal size. Training and testing is performed k times. In iteration i, partition D_i is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets D_2, …, D_k, collectively serve as the training set in order to obtain a first model, which is tested on D_1 ; the second iteration is trained on subsets D_1, D_3,…, D_k, and tested on D_2 ; and so on. Each sample is used the same number of times for training and once for testing. For classification, the accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of tuples in the initial data.

Leave-one-out is a special case of k-fold cross-validation where k is set to the number of initial tuples. That is, only one sample is "left out" at a time for the test set. In **stratified cross-validation**, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data. In general, stratified 10-fold cross-validation is recommended for estimating accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

4.4.2. Comparing classifiers based on ROC curve

The cost associated with a false negative (such as, incorrectly predicting that a cancerous patient is not cancerous) is far greater than that of a false positive (incorrectly yet conservatively labeling a noncancerous patient as cancerous). In such cases, we can outweigh one type of error over another by assigning a different cost to each. Similarly, the benefits associated with a true positive decision may be different than that of a true negative. Up to now, to compute classifier accuracy, we have assumed equal costs and essentially divided the sum of true positives and true negatives by the total number of test tuples.

ROC curves are a useful visual tool for comparing two classification models. The name ROC stands for Receiver Operating Characteristic. ROC curves come from signal detection theory that was developed during World War II for the analysis of radar images. An ROC curve for a given model shows the trade-off between the true positive rate (TPR) and the false positive rate (FPR). Given a test set and a model, TPR is the proportion of positive (or 'yes') tuples that are correctly labeled by the model; FP R is the proportion of negative (or 'no') tuples that are mislabeled as positive.

Given that TP, FP, P, and N are the number of true positive, false positive, positive and negative tuples, respectively. We know that: $TPR = \frac{TP}{P}$, which is sensitivity. Furthermore, $FPR = \frac{FP}{N}$ which is $1 - Specificity$. For a two-class problem, a ROC curve allows us to visualize the trade-off between the rate at which the model can accurately recognize positive cases versus the rate at which it mistakenly identifies negative cases as positive for different portions of the test set.

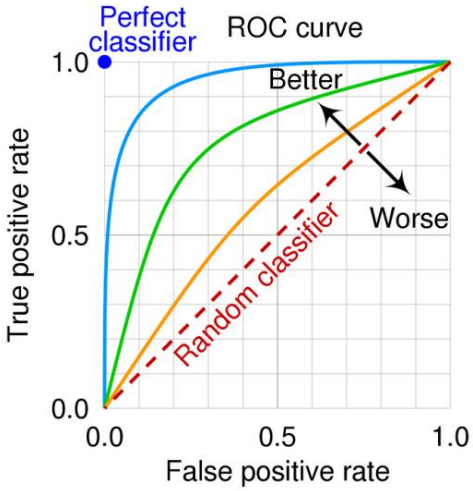


Figure 4.10. A ROC curve[15]

The vertical axis of an ROC curve represents TPR. The horizontal axis represents FPR. A ROC curve for M is plotted as follows. Starting at the bottom left-hand corner (where TPR = FPR = 0), we check the actual class label of the tuple at the top of the list. If we have a true positive, then TP and thus TPR increase. On the ROC curve, we move up and plot a point. If, instead, the model classifies a negative tuple as positive, we have a false positive, and so both FP and FPR increase. On the ROC curve, we move right and plot a point. This process is repeated for each of the test tuples in ranked order, each time moving up on the curve for a true positive or toward the right for a false positive

The **area under the ROC curve** (AUC) is a measure of the accuracy of the model. It represents the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance. In simpler terms, it measures the model's ability to distinguish between positive and negative classes. AUC ranges from 0 to 1, where 0.5 indicates a random classification, and 1 signifies a perfect classifier.

Conclusion

This chapter summarized the core concepts and methods of classification, a key task in machine learning used to assign predefined labels to data. We reviewed traditional techniques such as decision trees, K-NN, Naïve Bayes, and SVM, as well as deep learning models capable of handling complex and high-dimensional data. Challenges like overfitting, underfitting, and model selection were highlighted, along with essential evaluation metrics including accuracy, precision, recall, F1-score, and ROC curves. Overall, the chapter

[15] https://medium.com/@ilyurek/roc-curve-and-auc-evaluating-model-performance-c2178008b02

provided a concise understanding of how classification models are built, evaluated, and applied in real-world scenarios.

Practical work 04

Classification with KNN and ANN

1. Aim of the practice

- Learn to perform classification tasks using a classical ML model (KNN) and a neural network (ANN), and compare their performance.

Exercise 01. Classification using KNN

Implement K-Nearest Neighbors classifier.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

1. Load the Iris dataset.
2. Preprocess the data:
3. Split the dataset into training and test sets (e.g., 80/20).
4. Implement KNN classifier:
    - Select an appropriate k (e.g., 3, 5, or 7)
    - Train the model and predict on test set
5. Evaluate performance: Accuracy, confusion matrix, classification report

Exercise 2: Classification using ANN

Implement a feedforward Artificial Neural Network for the same dataset.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
```

1. Use Keras to build an ANN:
    - Input layer matches number of features
    - One or two hidden layers with activation relu (input=16)
    - Output layer with softmax activation (multiclass=3)
2. Compile the model with categorical_crossentropy loss and adam optimizer.
3. Train the model on the training set.
4. Evaluate the model on the test set.
5. Compare ANN performance with KNN.

Solution of Practical work 04

Chapter 5.

Clustering

Introduction

The process of grouping (or partitioning) a set of physical or abstract objects into classes of similar objects is called clustering (also called segmentation or partitioning). A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications.

Clustering is an important human activity. Early in childhood, one learns how to distinguish between cats and dogs, or between animals and plants, by continuously improving subconscious classification schemes. Clustering or cluster analysis has been widely used in numerous applications, including pattern recognition, data analysis, image processing, and market research. By clustering, one can identify crowded and sparse regions, and therefore, discover overall distribution patterns and interesting correlations among data attributes.

In machine learning, clustering is an unsupervised learning method and used either as a stand-alone tool to get insight into data distribution or as a preprocessing step for other algorithms. Unlike classification, clustering and unsupervised learning do not rely on predefined classes and labeled dataset training examples. For this reason, clustering is a form of learning by observation, rather than learning by examples. In conceptual clustering, a group of objects forms a class only if it is describable by a concept. It consists of two components:
(1) it discovers the appropriate classes, and
(2) it forms descriptions for each class, as in classification. The guideline of striving for high intraclass similarity and low interclass similarity still applies.

In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in large databases.

5.1. Clustering Methods

We describe the most well-known clustering algorithms. The main reason for having many clustering methods is the fact that the notion of “cluster” is not precisely defined. Consequently, many clustering methods have been developed, each of which uses a different induction principle. Han and Kamber (2001) suggest categorizing the methods into five main categories:

- **Partitioning methods**: Construct various partitions and then evaluate them by some criterion.
- **Hierarchical methods**: Create a hierarchical decomposition of the set of data (or objects) using some criterion.

- **Density-based methods**: based on connectivity and density functions.
- **Grid-based methods**: based on a multiple-level granularity structure.
- **Model-based methods**: A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each other.

5.1.1. Partitioning methods

Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster, and $k \leq n$. It classifies the data into k groups, which together satisfy the following requirements: (1) each group must contain at least one object, and (2) each object must belong to exactly one group. Notice that the second requirement can be relaxed in some fuzzy partitioning techniques.

Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an iterative relocation technique which attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are “close" or related to each other, whereas objects of different clusters are “far apart" or very different.

There are various kinds of other criteria for judging the quality of partitions. To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of two popular heuristic methods: k-means algorithm, where each cluster is represented by the mean value of the objects in the cluster; and (2) k-medoids algorithm, where each cluster is represented by one of the objects located near the center of the cluster.

5.1.1.1. K-means

K-means defines a prototype in terms of centroid, which is usually the mean of a group of points, and is typically applied to objects in a continuous n-dimensional space. It proceeds as follows:

- First, it randomly selects k of the objects, which initially each represent a cluster mean or center (also called center of gravity).
- For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean (e.g., using Euclidean distance).
- It then computes the new mean for each cluster. This process iterates until the criterion function converges. The center of each cluster is calculated as the mean of all the instances belonging to that cluster:

$$\mu_k = \frac{1}{N_k} \sum_{q=1}^{N_k} x_q$$

Where N_k is the number of instances belonging to cluster k and μ_k is the mean of the cluster k. The k-means procedure is summarized below.

Algorithm Basic K-means algorithm.

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.

Example: let us consider 8 points: A1=(2,10), A2=(2,5), A3=(8,4), A4=(5,8), A5=(7,5), A6=(6,4), A7=(1,2), A8=(4,9). Use the k-means algorithm and Euclidean distance to cluster the points into 3 clusters: A1, A4 and A7.

1st epoch:

Points	Distance with centroid			Cluster
	C1=(2,10)	C2=(5,8)	C3=(1,2)	
A1=(2,10)	0	3.60	8.06	1
A2=(2,5)	5	4.24	3.16	3
A3=(8,4)	6	5	7.28	2
A4=(5,8)	3.60	0	7.21	2
A5=(7,5)	7.07	3.60	6.70	2
A6=(6,4)	7.21	4.12	5.38	2
A7=(1,2)	8.06	7.21	0	3
A8=(4,9)	2.23	1.41	7.61	2

2nd epoch: C2= ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6), C3= ((2+1)/2, (5+2)/2) = (1.5, 3.5) (5,8)

Points	Distance with centroid			Cluster	New cluster
	C1=(2,10)	C2= (6, 6)	C3= (5,8)		
A1=(2,10)	0	5.66	3.61	1	1
A2=(2,5)	5	4.12	4.24	3	2
A3=(8,4)	8.49	2.83	5	2	2
A4=(5,8)	3.61	2.24	0	2	3
A5=(7,5)	7.07	1.41	3.61	2	2
A6=(6,4)	7.21	2	4.12	2	2
A7=(1,2)	8.06	6.40	7.21	3	2
A8=(4,9)	2.23	3.61	1.41	2	3

3th epoch: C2= (4.8, 4), C3= (4.5, 8.5)

Points	Distance with centroid			Cluster	New cluster
	C1=(2,10)	C2= (4.8, 4)	C3= (4.5,8.5)		
A1=(2,10)	0	6.62	2.92	1	1
A2=(2,5)	5	2.97	4.30	2	2
A3=(8,4)	8.49	3.2	5.70	2	2
A4=(5,8)	3.61	4	0.71	3	3

A5=(7,5)	7.07	2.42	4.30	2	2
A6=(6,4)	7.21	1.2	4.74	2	2
A7=(1,2)	8.06	4.29	7.38	2	2
A8=(4,9)	2.24	5.06	0.71	3	3

The data points no longer change clusters between iterations, so the algorithm stop with three clusters: C1{A1}, C2{A2, A3, A5, A6, A7}, and C3{A4, A8}

The method works well when the clusters are compact clouds that are rather well separated from one another. It is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is O(nkt), where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \leq n$ and $t \leq n$. The method often terminates at a local optimum. The k-means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical attributes are involved. The necessity for users to specify k, the number of clusters, in advance can be seen as a disadvantage. The k-means method is not suitable for discovering clusters with non-convex shapes, or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

Fuzzy C-Means (FCM) is a soft clustering algorithm where each data point can belong to multiple clusters with varying degrees of membership. Unlike hard clustering methods like K-Means, where each point is assigned to exactly one cluster, FCM assigns a membership value (between 0 and 1) to each data point for each cluster. The design of membership functions is the most important problem in fuzzy clustering; different choices include those based on similarity decomposition and centroids of clusters. A generalization of the FCM algorithm has been proposed through a family of objective functions.

5.1.1.2. K-medoids

The k-means algorithm is sensitive to outliers since an object with an extremely large value may substantially distort the distribution of data. To deal with this problem, instead of taking the mean value of the objects in a cluster as a reference point, the medoid can be used, which is the most centrally located object in a cluster. Thus, the partitioning method can still be performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point.

The basic strategy of k-medoids clustering algorithms is to find k clusters in n objects by first arbitrarily finding a representative object (the medoid) for each cluster. Each remaining object is clustered with the medoid to which it is the most similar. The strategy then iteratively replaces one of the medoids by one of the non-medoids as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function which measures the average dissimilarity between an object and the medoid of its cluster. The set of best objects for each cluster in one iteration form the medoids for the next iteration. For large values of n and k, such computation becomes very costly.

A general k-medoids algorithm for partitioning based on medoid or central objects.
Input: The number of clusters k, and a database containing n objects.
Output: A set of k clusters which minimizes the sum of the dissimilarities of all the objects to their nearest medoid.
Method:

1) Arbitrarily choose k objects as the initial medoids;
2) Repeat
3) Assign each remaining object to the cluster with the nearest medoid;
4) Randomly select a non-medoid object, o_{random};
5) Compute the total cost, S, of swapping o_j with o_{random};
6) If S < 0 then swap o_j with o_{random} to form the new set of k medoids;
7) Until no change;

The k-medoids method is more robust than k-means in the presence of noise and outliers because a medoid is less nuanced by outliers or other extreme values than a mean. However, its processing is more costly than the k-means method. Both methods require the user to specify k, the number of clusters.

5.1.1.3. Determining the Number of Clusters

Determining the right number of clusters in a dataset is important, not only because some clustering algorithms like k-means requires such a parameter, but also because the appropriate number of clusters controls the proper granularity of cluster analysis.
Determining the proper number of clusters on a data set can be regarded as finding a good balance between the compressibility and the accuracy in cluster analysis. However, determining the number of clusters is far from easy, often because the right number is ambiguous. The interpretation of the number of clusters often depends on the shape and scale of the distribution in a data set, as well as the clustering resolution required by a user. There are many possible ways to estimate the number of clusters.

One simple heuristic for selecting the number of clusters is to set it around $\sqrt{\frac{n}{2}}$, where n is the number of data points, assuming each cluster will have roughly $\sqrt{2n}$ members.
A more systematic approach is the *elbow method*, which relies on plotting the within-cluster variance as a function of the number of clusters, k. While increasing k typically reduces this variance (as more clusters better capture small, cohesive groups), the rate of improvement eventually slows down. The point where this reduction sharply changes, the "elbow" is taken as an indication of the optimal number of clusters.
In the average silhouette method, a silhouette value for every datapoint is calculated, the mean of which is used to find the optimal number of clusters. The silhouette value represents how similar a datapoint is to its own cluster relative to all other clusters or their centroids. The value ranges from -1 to +1 (more details in section 5.3.1.3).

Another approach is cross-validation, adapted from classification tasks. The dataset is split into m parts. Each time, $m-1$ parts are used to form clusters, and the remaining part is used for testing. The quality is assessed based on the distance from test points to their nearest cluster centers. This process is repeated for different k values, and the one yielding the best average clustering performance across all folds is chosen.

The Gap Statistic is a method used to determine the optimal number of clusters (k) in a dataset by comparing the clustering performance on the real data with that on random reference data (data with no clustering structure).

The process of Gap method is as follows:

1. Run the clustering algorithm (e.g., K-Means) on actual dataset for each value of k (e.g., from 1 to 10). Compute the within-cluster dispersion (for cluster C_i) :

$$W_k = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \overline{x_i}\|^2$$

2. Generate B reference datasets with the same size and range as the data, but sampled uniformly at random (no structure). For each reference dataset:
   - Apply the clustering algorithm for each k.
   - Compute the average log of W_k for the reference datasets:

$$E^*[\log(W_k)]$$

3. Calculate the Gap value: $Gap(k) = E^*[\log(W_k)] - \log(W_k)$
4. Select the optimal k: Choose the smallest k such that:

$$Gap(k) \geq Gap(k+1) - S_{k+1}$$

where $S_{k+1}$is the standard deviation of the reference W values.

5.1.2. Hierarchical methods

Hierarchical clustering techniques are a second important category of clustering methods. As with k-means; these approaches are relatively olds compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating hierarchical clustering:

- **Agglomerative:** also called the bottom-up. It starts with the points as individual clusters and, at each step merges the closest pair of clusters. This requires defining the notion of cluster proximity.
- **Divisive:** also called the top-down. All objects initially belong to one cluster. Then the cluster is divided into sub-clusters, which are successively divided into their own sub-clusters. This process continues until the desired cluster structure is obtained.

The result of the hierarchical methods is a dendrogram, representing the nested grouping of objects and similarity levels at which groupings change. A clustering of the data objects is obtained by cutting the dendrogram at the desired similarity level.

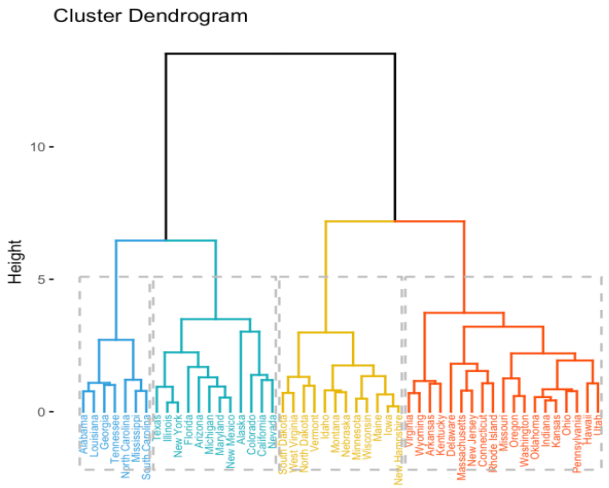


Figure 5.1. Example of dendrogram for hierarchical clustering[16]

The merging or division of clusters is performed according to some similarity measure, chosen so as to optimize some criterion. Agglomerative Hierarchical Clustering (AHC) techniques are by far the most common.

The hierarchical clustering methods could be further divided according to the manner that the similarity measure is calculated:

Single-link clustering (also called the connectedness, the minimum method or the nearest neighbor method): methods that consider the distance between two clusters to be equal to the shortest distance from any member of one cluster to any member of the other cluster. If the data consist of similarities, the similarity between a pair of clusters is considered to be equal to the greatest similarity from any member of one cluster to any member of the other cluster.

Complete-link clustering (also called the diameter, the maximum method or the furthest neighbor method): methods that consider the distance between two clusters to be equal to the longest distance from any member of one cluster to any member of the other cluster.

Average-link clustering (also called minimum variance method): methods that consider the distance between two clusters to be equal to the average distance from any member of one cluster to any member of the other cluster.

Example: a dataset consists of five elements: A(1, 2), B(2, 1), C(4, 5), D(5, 4), and E(8, 8).

Step 1: Start with each point as its own cluster:

Clusters: {A}, {B}, {C}, {D}, {E}

Step 2: Compute all pairwise distances:

Using Euclidean distance:

d(A, B) ≈ 1.41

d(C, D) ≈ 1.41

d(A, C) ≈ 4.24

[16] https://bookdown.org/content/f097ddae-23f5-4b2d-b360-ad412a6ca36a/chapter-2.-hierarchical-clustering.html

d(B, C) ≈ 4.24
d(D, E) ≈ 4.24
d(C, E) ≈ 5.00
etc.

Step 3: Merge the closest pair:
Merge {A} and {B} → new cluster {A, B}
Clusters: {A, B}, {C}, {D}, {E}

Step 4: Update distances (single linkage = min distance):
d({A, B}, C) = min[d(A, C), d(B, C)] = 4.24
d(C, D) = 1.41
d(D, E) = 4.24
d(C, E) = 5.00

Step 5: Merge the next closest pair:
Merge {C} and {D} → {C, D}
Clusters: {A, B}, {C, D}, {E}

Step 6: Merge next closest:
d({C, D}, E) = min[d(C, E), d(D, E)] = 4.24
d({A, B}, {C, D}) = 4.24
→ Merge {A, B} and {C, D} → {A, B, C, D}
Clusters: {A, B, C, D}, {E}

Step 7: Final merge:
d({A, B, C, D}, E) ≈ 4.24
→ Merge all into one cluster.

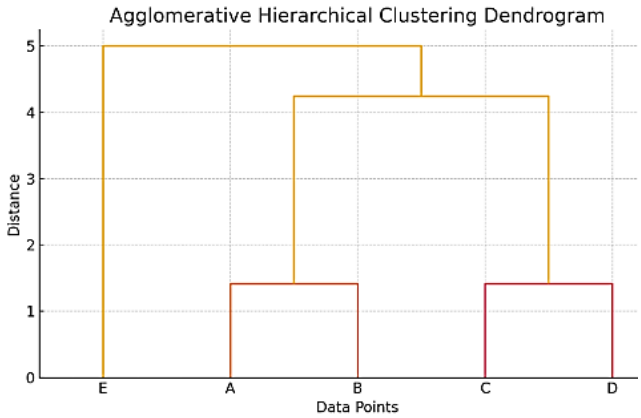


Figure 5.2. Dendrogram of the example of dataset

Generally, hierarchical methods are characterized with the following strengths:

- Versatility: The single-link methods, for example, maintain good performance on data sets containing non-isotropic clusters, including well-separated, chain-like and concentric clusters.
- Multiple partitions: hierarchical methods produce not one partition, but multiple nested partitions, which allow different users to choose different partitions, according to the desired similarity level.

The hierarchical partition is presented using the dendrogram. The main disadvantages of the hierarchical methods are:

- Inability to scale well: The time complexity of hierarchical algorithms is at least $O(m^2)$ (where m is the total number of instances), which is non-linear with the number of objects.
- Clustering a large number of objects using a hierarchical algorithm is also characterized by huge I/O costs.

Hierarchical methods can never undo what was done previously. Namely there is no back-tracking capability. As a result, poor decisions at any step can negatively impact the final clustering quality. Additionally, hierarchical methods do not scale efficiently, since each step requires examining many objects or clusters.
To improve clustering quality, multiphase approaches combine hierarchical clustering with other techniques. Two notable examples are BIRCH and Chameleon:

- **BIRCH** builds a hierarchical structure using tree-based microclusters, then applies another clustering algorithm to group these into larger clusters.
- **Chameleon** uses dynamic models to enhance hierarchical clustering adaptability.

Moreover, hierarchical clustering methods can also be categorized from different perspectives, including algorithmic, probabilistic, and Bayesian approaches.

5.1.2.1. BIRCH: Multiphase Hierarchical Clustering

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is designed for clustering a large amount of numeric data by integrating hierarchical clustering (at the initial microclustering stage) and other clustering methods. It overcomes the two difficulties in agglomerative clustering methods: (1) scalability, and (2) the inability to undo what was done in the previous step. BIRCH uses the notion of clustering feature to summarize a cluster, and clustering feature tree (CF tree) to represent a cluster hierarchy.
Consider a cluster of n d-dimensional data objects or points. The clustering feature (CF) of the cluster is a 3-dimensional vector summarizing information about clusters of objects. It is defined as follows, $CF = \langle n, LS, SS \rangle$, where $LS = \sum_{i=1}^{n} x_i$ is the linear sum of the n point, and $SS = \sum_{i=1}^{n} {x_i}^2$ is the square sum of the data points.
Clustering features are additive. That is, for two disjoint clusters C_1 and C_2 with the clustering features $CF_1 = \langle n_1, LS_1, SS_1 \rangle$ and $CF_2 = \langle n_2, LS_2, SS_2 \rangle$, respectively, the clustering feature for the cluster that formed by merging C_1 and C_2 is simply:

$$CF_1 + CF_2 = \langle n_1 + n_2,\ LS_1 + LS_2, SS_1 + SS_2 \rangle$$

Multiphase Strategy:

Phase 1 – Tree Building (Micro-Clustering):
Constructs a CF Tree, a height-balanced tree that incrementally builds compact summaries of data. Each node (especially leaves) represents a micro-cluster, storing summary statistics (number of points, linear sum, square sum).
Phase 2 – Global Clustering (Macro-Clustering):

Applies a standard clustering algorithm (e.g., K-Means or Agglomerative) to the leaf entries (micro-clusters) of the CF tree. This second phase refines and groups micro-clusters into final macro-clusters.

For Phase 1, the CF tree is built dynamically as objects are inserted. Thus, the method is incremental. An object is inserted into the closest leaf entry (subcluster). If the diameter of the subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split. After the insertion of the new object, information about the object is passed toward the root of the tree. The size of the CF tree can be changed by modifying the threshold. If the size of the memory that is needed for storing the CF tree is larger than the size of the main memory, then a larger threshold value can be specified and the CF tree is rebuilt. The rebuild process is performed by building a new tree from the leaf nodes of the old tree. Thus, the process of rebuilding the tree is done without the necessity of rereading all of the objects or points. This is similar to the insertion and node split in the construction of B+-trees. Therefore, for building the tree, data has to be read just once. Some heuristics and methods have been introduced to deal with outliers and improve the quality of CF trees by additional scans of the data. Once the CF tree is built, any clustering algorithm, such as a typical partitioning algorithm, can be used with the CF tree in Phase 2.
These structures help the clustering method achieve good speed and scalability in large or even streaming databases, and also make it effective for incremental and dynamic clustering of incoming objects. The time complexity of the algorithm is O(n), where n is the number of objects to be clustered. Experiments have shown the linear scalability of the algorithm with respect to the number of objects, and good quality of clustering of the data.

5.1.2.2. Chameleon: Dynamic Multiphase Hierarchical Clustering

Chameleon is a multiphase hierarchical clustering algorithm designed to handle datasets with clusters of arbitrary shapes, sizes, and densities. It improves on traditional hierarchical methods by using dynamic modeling to adapt to the internal characteristics of the data, rather than relying on fixed rules for merging clusters.

Chameleon uses a k-nearest neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object is among the k-most similar objects of the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partition the k-nearest neighbor graph into a large number of relatively small subclusters such that it minimizes the edge cut.
Specifically, Chameleon determines the similarity between each pair of clusters C_i and C_j according to their relative interconnectivity, $\mathrm{RI}(\mathrm{C_i}, \mathrm{C_j})$, and their relative closeness, $\mathrm{RC}(\mathrm{C_i}, \mathrm{C_j})$:
Interconnectivity $\mathrm{RI}(\mathrm{C_i}, \mathrm{C_j})$ measures how strongly two clusters are connected (based on edge weights between them.

$$\mathrm{RI}(\mathrm{C_i}, \mathrm{C_j}) = \frac{\left|EC_{\{C_i,C_j\}}\right|}{\frac{1}{2}(\left|EC_{C_i}\right| + \left|EC_{C_j}\right|)}$$

where $EC_{\{C_i,C_j\}}$ is the edge cut as defined above for a cluster containing both Ci and Cj. Similarly, EC_{C_i} (or EC_{C_j}) is the minimum sum of the cut edges that partition Ci (or Cj) into two roughly equal parts.

Relative closeness $\mathrm{RC}(\mathrm{C_i}, \mathrm{C_j})$ measures how close the clusters are relative to their internal distances.

$$\mathrm{RC}(\mathrm{C_i}, \mathrm{C_j}) = \frac{\bar{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i| + |C_j|}\bar{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i| + |C_j|}\bar{S}_{EC_{C_j}}}$$

Where $\bar{S}_{EC_{\{C_i,C_j\}}}$ is the average weight of the edges that connect vertices in Ci to vertices in Cj, and $\bar{S}_{EC_{C_i}}$ (or $\bar{S}_{EC_{C_j}}$) is the average weight of the edges that belong to the min-cut bisector of cluster Ci (or Cj).

Chameleon has been shown to have greater power at discovering arbitrarily shaped clusters of high quality than several well-known algorithms such as BIRCH and density-based DBSCAN (Section 2.3.1). However, the processing cost for high-dimensional data may require O(n^2) time for n objects in the worst case.

5.1.2.3. Probabilistic Hierarchical Clustering

In probabilistic hierarchical clustering, each cluster is assumed to be generated by a probability distribution (e.g., Gaussian). The clustering process then aims to find a hierarchy that best explains the data under these probabilistic assumptions.

A Gaussian Mixture Model (GMM) is a probabilistic clustering model that assumes the data is generated from a mixture of several Gaussian (normal) distributions. Each component in the mixture represents a cluster, and the overall distribution of the data is modeled as a weighted sum of Gaussians. Instead of assigning each data point to a single cluster (as in K-Means), GMM assigns probabilities for a point to belong to each cluster. This makes it a soft clustering method.

For a dataset $X = \{x1, x2, \ldots, xn\}$ GMM models the probability density function as:

$$P(x) = \sum_{k=1}^{K} \pi_k . N(x|\overline{x_k}, cov(x_k))$$

Where:

- K: Number of clusters (Gaussian components)
- π_k: Mixing coefficient for component k (sums to 1)
- x_k: Mean of the k^{th} Gaussian
- $cov(x_k)$: Covariance matrix of the k^{th} Gaussian
- $N(x|\overline{x_k}, cov(x_k))$: Multivariate Gaussian density function

Training GMM – Expectation-Maximization (EM) Algorithm:

1. Initialization: Start with initial estimates of parameters ($\overline{x_k}$, $cov(x_k)$, π_k)
2. E-step (Expectation): Compute the responsibilities; probabilities that each point belongs to each cluster
3. M-step (Maximization): Update the parameters to maximize the expected likelihood
4. Repeat until convergence

A Gaussian Mixture Model is a powerful clustering tool that provides a probabilistic framework for modeling complex, real-world data. By using soft assignments and covariance structures, it can uncover more nuanced patterns than traditional clustering algorithms like K-Means.

5.1.3. Density-based methods

Density-based methods assume that the points that belong to each cluster are drawn from a specific probability distribution. The overall distribution of the data is assumed to be a mixture of several distributions. The aim of these methods is to identify the clusters and their distribution parameters. These methods are designed for discovering clusters of arbitrary shape which are not necessarily convex, namely: $x_i , x_j \in C_k$. This does not necessarily imply that: $\alpha . x_i + (1 - \alpha) . x_j \in C_k$.

The idea is to continue growing the given cluster as long as the density (number of objects or data points) in the neighborhood exceeds some threshold. Namely, the neighborhood of a given radius has to contain at least a minimum number of objects. When each cluster is characterized by local mode or maxima of the density function, these methods are called mode-seeking.

5.1.3.1. DBSCAN

The DBSCAN algorithm (density-based spatial clustering of applications with noise) is an example of density-based clustering, which discovers clusters of arbitrary shapes and is efficient for large spatial databases. The algorithm searches for clusters by searching the neighborhood of each object in the database and checks if it contains more than the minimum number of objects. Unlike K-Means or hierarchical clustering which assumes clusters are compact and spherical, DBSCAN perform well in handling real-world data irregularities such as:

- Arbitrary-Shaped Clusters: Clusters can take any shape not just circular or convex.
- Noise and Outliers: It effectively identifies and handles noise points without assigning them to any cluster.

Key Parameters in DBSCAN

1. Eps (eplison or ε): This defines the radius of the neighborhood around a data point. If the distance between two points is less than or equal to eps they are considered neighbors. A common method to determine eps is by analyzing the k-distance graph. Choosing the right eps is important:

- If eps is too small most points will be classified as noise.

- If eps is too large clusters may merge and the algorithm may fail to distinguish between them.

2. MinPts: This is the minimum number of points required within the eps radius to form a dense region. A general rule of thumb is to set MinPts >= D+1 where D is the number of dimensions in the dataset.
For most cases a minimum value of MinPts = 3 is recommended.
DBSCAN works by categorizing data points into three types:

1. Core points which have a sufficient number of neighbors within eps.
2. Border points which are near core points but lack enough neighbors to be core points themselves.
3. Noise points which do not belong to any cluster.

By iteratively expanding clusters from core points and connecting density-reachable points, DBSCAN forms clusters without relying on rigid assumptions about their shape or size.

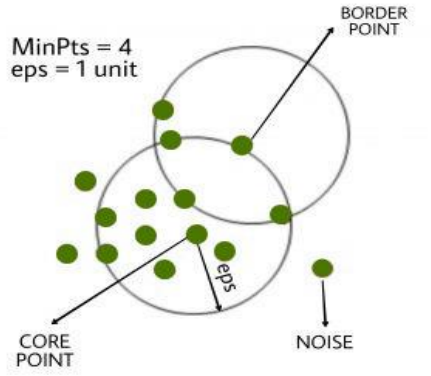


Figure 5.3. key parameters of DBSCAN algorithm[17]

Steps in the DBSCAN Algorithm

1. Identify Core Points: For each point in the dataset count the number of points within its eps neighborhood. If the count meets or exceeds MinPts mark the point as a core point.
2. Form Clusters: For each core point that is not already assigned to a cluster, create a new cluster. Recursively find all density-connected points, i.e points within the eps radius of the core point and add them to the cluster.
3. Density Connectivity: Two points a and b are density-connected if there exists a chain of points where each point is within the eps radius of the next and at least one point in the chain is a core point. This chaining process ensures that all points in a cluster are connected through a series of dense regions.
4. Label Noise Points: After processing all points any point that does not belong to a cluster is labeled as noise.

[17] https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/

Among density-based clustering methods, DBSCAN is the most widely used. It is favored for its simplicity, ability to discover clusters of arbitrary shapes, and built-in handling of noise and outliers. Unlike traditional methods such as K-Means, DBSCAN does not require the user to specify the number of clusters beforehand. Instead, it groups together points that are closely packed (based on two parameters: eps for distance and minPts for density) and marks points in low-density regions as noise. However, it can struggle when clusters have significantly different densities. In such cases, more advanced methods like HDBSCAN or OPTICS are preferred. Despite these limitations, DBSCAN remains the most commonly used density-based algorithm due to its balance of effectiveness and ease of use.

Example: let us consider 8 points: A (1, 2), B (2, 2), C (2, 3), D (8, 7), E (8, 8), F (25, 80). We start with eps=1.5 and MinPts=2.

1. Identify which points are core points.
2. Determine the clusters formed.
3. Identify any noise points.

Step 1: Calculate pairwise distances using Euclidean distance with the distance are within eps. D(A, B)=$\sqrt{(\mathbf{1}-\mathbf{2})^{\mathbf{2}}+(\mathbf{2}-\mathbf{2})^{\mathbf{2}}}=\mathbf{1}$, D(A, C)=$\sqrt{\mathbf{2}}=\mathbf{1.41}$, D(B, C)=1, D(D, E)=1

Step 2: Identify core points

Core points must have at least 2 points **(**MinPts=2**).**

- **A**: neighbors = {B, C} → 3 points → core point
- **B**: neighbors = {A, C} → 3 points → core point
- **C**: neighbors = {A, B} → 3 points → core point
- **D**: neighbors = {E} → only 2 points → core point
- **E**: neighbors = {D} → only 2 points → core point
- **F**: no neighbors → □ not a core point

So, Core Points are: A, B, C, D, E.

Step 3: Form clusters

Cluster 1: {A, B, C} all connected via core points (mutual neighbors).

Cluster 2: {D, E} both D and E count as core points (since they have 2 points including themselves). Both are core points, connected.

Step 4: Noise points

F has no neighbors within ε = 1.5 → not part of any cluster → **Noise**

5.1.3.2. HDBSCAN

HDBSCAN extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters. HDBSCAN is especially helpful for datasets with complex structures or varying densities because it creates a hierarchical tree of clusters that enable users to examine the data at different levels of granularity.

HDBSAN examines the density of the data points in the dataset. It starts by calculating a density-based clustering hierarchy, which creates clusters from densely

connected data points. This hierarchical structure enables the recognition of clusters of various shapes and sizes.
The algorithm then extracts clusters from the hierarchy, taking into the account the stability of cluster assignments across different levels of the hierarchy. It identifies stable clusters as those with consistent memberships at multiple levels, ensuring cluster formation robustness.
In addition, HDBSCAN differentiates between noise and meaningful clusters by taking into account points with the low densities or the ones not belonging to any cluster. HDBSCAN captures and eliminates the noise by constantly adjusting the minimum cluster size parameter and adding a minimum spanning tree.

5.1.3.3. OPTICS

OPTICS (Ordering Points To Identify Clustering Structure) is an augmented ordering algorithm which means that instead of assigning cluster memberships, it stores the order in which the points are processed. OPTICS requires the same eps and MinPts hyperparameters as DBSCAN, but the eps parameter is theoretically unnecessary. Explicitly setting the value of this parameter is only used for the practical purpose of reducing the algorithm's runtime complexity.
Instead of directly assigning clusters, OPTICS generates an ordering of the data points based on their density-reachability. This ordering can later be used to extract clusters using different density thresholds. Key Concepts of OPTICS are:

Core Distance: The core distance of a point is the smallest distance eps such that at least MinPts points (including the point itself) are within that distance. If the point does not have enough neighbors to satisfy MinPts, its core distance is undefined.

Reachability Distance: The reachability distance of a point p with respect to another point o is defined as the maximum of the core distance of o and the actual distance between p and o. This value describes how far p is from a dense region (centered at o).

Ordering of Points: OPTICS starts with an arbitrary point and expands the cluster by visiting neighboring points in the order of their reachability distances. This creates a linear ordering of the points that reflects the structure of the data.

Reachability Plot: OPTICS produces a reachability plot, which is a 1D plot of the reachability distances of points in the order they were processed. Valleys in this plot represent clusters: deep valleys correspond to dense clusters, while peaks or high values suggest sparse regions or noise.

OPTICS gives a more flexible and informative view of the clustering structure, especially useful when the dataset contains clusters with different densities. Instead of fixed labels, it provides a reachability plot from which meaningful clusters can be extracted at various density levels.

5.1.4. Grid-based methods

These methods partition the space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed. The main

advantage of the approach is its fast-processing time. STING is a typical example of a grid-based method. CLIQUE and Wave-Cluster are two clustering algorithms which are both grid-based and density-based.

5.1.4.1. STING

A STING (Statistical Information Grid) is a grid-based clustering technique, which uses a multidimensional grid data structure that quantifies space into a finite number of cells. The idea is to capture statistical information associated with spatial cells in such a manner that whole classes of queries and clustering problems can be answered without recourse to the individual objects. Instead of focusing on data points, it focuses on the value space surrounding the data points.
In STING, the spatial area is divided into rectangular cells and several levels of cells at different resolution levels. High-level cells are divided into several low-level cells.

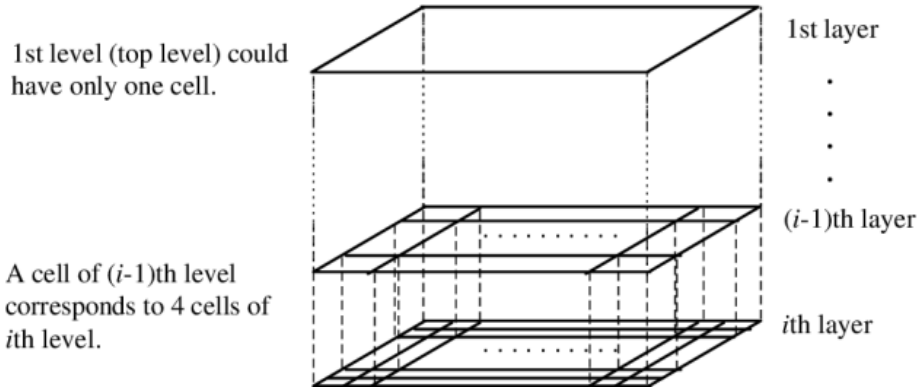


Figure 5.4. Hierarchical Structure by STING method[18].

A cell in level i corresponds to the union of the areas of its children at level $i+1$. For each cell, there two classes of parameters: attribute-dependent and attribute-independent parameters.

The attribute-independent parameter is: n a number of objects (points) in this cell.

The attribute-dependent parameters (for numerical values of attributes): m mean of all values in this cell, s standard deviation of all values of the attribute in this cell, min the minimum value of the attribute in this cell, max the maximum value of the attribute in this cell, and $distribution$ the type of distribution that the attribute value in this cell follows.
Potential distribution types are: normal, uniform, exponential, and so on. The value NONE is assigned if the distribution type is unknown. These statistical parameters are useful for query processing and other data analysis tasks. The statistical parameter of higher-level cells can easily be computed from the parameters of the lower-level cells.

Statistical Information Grid-based Algorithm:

Step 1: Determine a layer, to begin with.

[18] https://www.semanticscholar.org/paper/STING%3A-A-Statistical-Information-Grid-Approach-to-Wang-Yang/cf304776b89bbfeb109fdc2bccc6ab8d1da144cf

Step 2: For each cell of this layer, it calculates the confidence interval or estimated range of probability that this is cell is relevant to the query.

Step 3: From the interval calculate above, it labels the cell as relevant or not relevant.

Step 4: If this layer is the bottom layer, go to point 6, otherwise, go to point 5.

Step 5: It goes down the hierarchy structure by one level. Go to point 2 for those cells that form the relevant cell of the high-level layer.

Step 6: If the specification of the query is met, go to point 8, otherwise go to point 7.

Step 7: Retrieve those data that fall into the relevant cells and do further processing. Return the result that meets the requirement of the query. Go to point 9.

Step 8: Find the regions of relevant cells. Return those regions that meet the requirement of the query. Go to point 9.

Step 9: Stop.

Example: Consider a dataset containing 8 objects represented in 2D space.

(1,2), (2,3), (3,3), (11,17), (15,18), (35,80), (36,82), (37,81).

Step 1: Divide Space into Grid Cells: Assume the 2D space is divided into a 10x10 grid: Each cell = 10 units x 10 units. Grid size = (0–100, 0–100) divided into 100 cells:

(1,2), (2,3), (3,3) → cell A (0–10, 0–10)

(11,17), (15,18) → cell B (10–20, 10–20)

(35,80), (36,82), (37,81) → cell C (30–40, 80–90)

Step 2: Compute Statistics for Each Cell

Cell A: count = 3, mean_x = 2, mean_y = 2.67

Cell B: count = 2, mean_x = 8.67, mean_y = 11.67

Cell C: count = 3, mean_x = 36, mean_y = 81

Step 3: Clustering

STING groups neighboring, statistically similar cells into clusters.

Cell A and Cell B: Even though they're in different y-ranges, they are closer in distance compared to Cell C. Together; they can be treated as Cluster 1.

Step 4: Output Clusters

STING identifies:

Cluster 1: (1,2), (2,3), (3,3), (11,17), (15,18)

Cluster 2: (35,80), (36,82), (37,81)

Two main clusters are identified across three cells using the STING method. This structure helps STING efficiently approximate clustering without distance calculations between all pairs of points.

5.1.4.2. CLIQUE

CLIQUE (Clustering In QUEst) is a density-based and grid-based subspace clustering algorithm. It discretizes the data space through a grid and estimates the density by counting the number of points in a grid cell. A cluster is a maximal set of connected dense units in a subspace (A unit is dense if the fraction of total data points contained in the unit exceeds the input model parameter). A subspace cluster

is a set of neighboring dense cells in an arbitrary subspace. If a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any (k-1)-dimensional projections of this space.

The CLIQUE algorithm consists of the following steps:

1. **Identification of subspaces that contain clusters:** The CLIQUE first divides the data space into grids. It is done by dividing each dimension into equal intervals called units. After that, it identifies dense units. A unit is dense if the data points in this are exceeding the threshold value. The algorithm proceeds level-by-level. It first determines 1-dimensional dense units by making a pass over the data. Having determined (k-1)-dimension al dense units, the candidate k-dimensional units are determined using the candidate generation procedure given below. A pass over the data is made to and those candidate units that are dense. The algorithm terminates when no more candidates are generated. Once the algorithm finds dense cells along one dimension, the algorithm tries to find dense cells along two dimensions, and it works until all dense cells along the entire dimension are found.
2. **Identification of clusters:** After finding all dense cells in all dimensions, the algorithm proceeds to find the largest set ("cluster") of connected dense cells. The input to the next step of CLIQUE is a set of dense units D, all in the same k-dimensional space S. The output will be a partition of D into $D^1, \dots, D^q$, such that all units in D^i are connected and no two units $u^i \in D^i$, $u^j \in D^j$ with $i \neq j$ are connected. Each such partition is a cluster according to our definition. The problem is equivalent to finding connected components in a graph, where the vertices correspond to dense units, and there is an edge between two vertices if and only if the corresponding dense units have a common face. Units corresponding to vertices in the same connected component of the graph are connected because there is a path of units that have a common face between them; therefore, they are in the same cluster. On the other hand, units corresponding to vertices in different components cannot be connected, and therefore cannot be in the same cluster. The depth-first search algorithm can be used to find the connected components of the graph. Starting with some unit u in D, assign it the first cluster number, and find all the units it is connected to. Then, if there still are units in D that have not yet been visited, we find one and repeat the procedure.
3. **Generation of minimal description for the clusters**: The input to this step consists of disjoint sets of connected k-dimensional units in the same subspace. Each such set is a cluster and the goal is to generate a concise description for it. Clusters are then generated from all dense subspaces using the Apriori approach.

Example: Consider a dataset containing 12 objects represented in 2D space.
(1, 1), (1, 2), (2, 1), (2, 2), (8, 8), (8, 9), (9, 8), (9, 9), (5, 0), (5, 1), (6, 0), (6, 1).

We will use a 2D space, split into a 3x3 grid (3 intervals per dimension), with a density threshold of 3 points per unit.

Step 1: 1D Dense Units: Divide each dimension into 3 equal bins:
X-axis ([0–10]) → bins: X0: [0–3.33[, X1: [3.33–6.66[, X2: [6.66–10]
Y-axis ([0–10]) → bins: Y0: [0–3.33[, Y1: [3.33–6.66[, Y2: [6.66–10]
Count how many points fall into each 1D bin:
X-axis:
X0: (1,1), (1,2), (2,1), (2,2) → 4 points
X1: (5,0), (5,1), (6,0), (6,1) → 4 points
X2: (8,8), (8,9), (9,8), (9,9) → 4 points
Y-axis:
Y0: (1,1), (1,2), (2,1), (2,2), (5,0), (5,1), (6,0), (6,1)→ 8 points
Y1: → 0
Y2: (8,8), (8,9), (9,8), (9,9) → 4 points
Dense 1D units:
X0, X1, X2
Y0, Y2
Step 2: Generate 2D Candidate Units from 1D Dense Units: we find the different combinations between X and Y. (X0, Y0), (X0, Y2), (X1, Y0), (X1, Y2), (X2, Y0), and (X2, Y2).

D1=(X0, Y0): (1,1), (1,2), (2,1), (2,2) → 4 points

D2= (X1, Y0): (5,0), (5,1), (6,0), (6,1) → 4 points

D3= (X2, Y2): (8,8), (8,9), (9,8), (9,9) → 4 points

(X0, Y2), (X1, Y2), (X2, Y0) are empty.

2D Dense Units:
(X0, Y0), (X1, Y0), (X2, Y2).

Step 3: Identification of clusters

Three clusters:

C1: (1,1), (1,2), (2,1), (2,2)

C2 : (5,0), (5,1), (6,0), (6,1)

C3 : (8,8), (8,9), (9,8), (9,9)

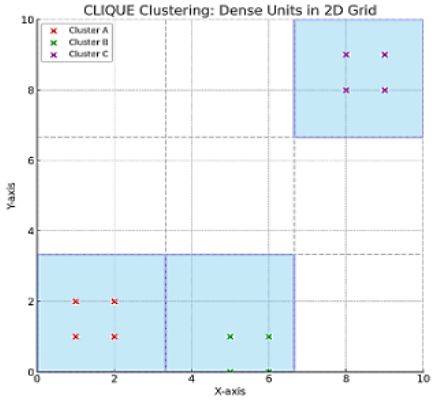


Figure 5.5. Graphical representation of clusters via CLIQUE method[19]

[19] https://ionreporter.thermofisher.com/ionreporter/help/GUID-0B9F2E37-34C5-4825-86BD-2FFB1C4A7421.html

CLIQUE finds high-quality clusters only in subspaces with the highest dimensionality, making it an efficient method. The threshold density and grid size must be properly adjusted in order to produce meaningful clustering results.

5.1.5. Model-based methods

These methods attempt to optimize the fit between the given data and some mathematical models. Unlike conventional clustering, which identifies groups of objects, model-based clustering methods also find characteristic descriptions for each group, where each group represents a class. The most frequently used induction methods are SVC and neural networks.

5.1.5.1. SVC

Support Vector Clustering (SVC) is an unsupervised machine learning algorithm inspired by the principles of Support Vector Machines (SVM). In SVC data points are mapped from data space to a high dimensional feature space using a kernel function. In the kernel's feature space the algorithm searches for the smallest sphere that encloses the image of the data using the Support Vector Domain Description (SVDD) algorithm. This sphere, when mapped back to data space, forms a set of contours which enclose the data points. Those contours are then interpreted as cluster boundaries, and points enclosed by each contour are associated by SVC to the same cluster.

A kernel function is a mathematical tool used to compute the similarity between two data points in a (possibly high-dimensional) space without explicitly computing their coordinates in that space. There are different types of kernel function, such as linear (the dot product), Sigmoid, polynomial. In SVC, the Gaussian kernel function is used between two points x_i and x_j as follows:

$$K(x_i, x_j) = e^{-q\|x_i - x_j\|^2}$$

With width parameter q.

The cluster description algorithm does not differentiate between points that belong to different clusters. To do so, SOM used a geometric approach (sphere form) involving radius $R(x)$, based on the following observation: given a pair of data points that belong to different components (clusters), any path that connects them must exit from the sphere in feature space. Therefore, such a path contains a segment of points y such that $R(y) > R$. This leads to the definition of the adjacency matrix A_{ij} between pairs of points x_i and x_j whose images lie in or on the sphere in feature space:

$$A_{ij} = \begin{cases} 1 & if, for\ all\ y\ on\ the\ line\ segment\ connecting\ x_i\ and\ x_j, R(y) \leq R \\ 0 & otherwise \end{cases}$$

Clusters are then defined as the connected components of the graph induced by A. This labeling procedure is justified by the observation that nearest neighbors in data space can be connected by a line segment that is contained in the high dimensional sphere.

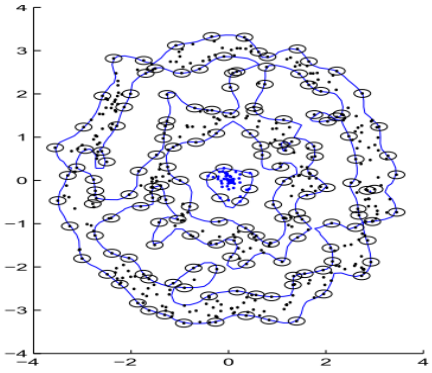


Figure 5.6. Example of SVC clustering results[20]

SVC algorithm consists of three steps:

1. Kernel Mapping: Data is transformed into a high-dimensional feature space using a Gaussian kernel function. This step helps separate clusters that are not linearly separable in the original space.
2. Enclosing Sphere: In the high-dimensional space, the algorithm finds the smallest enclosing sphere that contains all data points. This is done using the SVM technique, identifying support vectors that lie on the boundary.
3. Cluster Formation: The sphere is projected back to the original space. The pre-image of the sphere boundaries forms complex, nonlinear cluster boundaries. Data points inside each boundary form a cluster.

SVC offers several advantages over traditional clustering methods. One of its key strengths is its ability to detect arbitrarily shaped clusters, unlike algorithms such as K-Means that assume clusters are convex or spherical. SVC also does not require the number of clusters to be specified in advance; instead, clusters naturally emerge from the structure of the data. This makes it especially useful in exploratory data analysis where the number of groups is unknown.

Another major advantage is SVC's capability to handle high-dimensional data effectively, thanks to its use of kernel functions. These kernels map the input data into a higher-dimensional space, allowing the algorithm to separate clusters that would otherwise overlap in lower dimensions. The flexibility of choosing different kernel types (e.g., RBF, polynomial) makes SVC adaptable to various types of datasets and structures.

Moreover, SVC is built upon the solid theoretical foundation of Support Vector Machines (SVM), providing a reliable and mathematically sound approach to clustering. It is particularly useful in applications requiring precise boundary detection, such as image segmentation, anomaly detection, and bioinformatics.

However, it's worth noting that SVC can be computationally expensive, especially with large datasets, and it requires careful tuning of parameters such as the kernel and its associated hyperparameters. Despite these challenges, its flexibility and accuracy make it a powerful tool in complex clustering tasks.

[20] https://proceedings.neurips.cc/paper_files/paper/2000/file/14cfdb59b5bda1fc245aadae15b1984a-Paper.pdf

5.1.5.2. SOM Neural network

Self-Organizing Map (or Kohonen Map or SOM) is a type of Artificial Neural Network which follows an unsupervised learning approach and trained its network through a competitive learning algorithm. SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation. The SOM consists of two layers: an input layer and an output layer. During the learning process, a **"winner-takes-all"** strategy is used, where neurons in the output layer compete to respond to each input. The neuron whose weight vector is most similar to the current input is selected as the winner. Both this neuron and its neighboring neurons update their weights to better match the input pattern.

Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example. First, the model initializes the weights of size (n, C) where C is the number of clusters. Then iterating over the input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g., Euclidean distance) from training example). Weight updating rule is given by:

$w_{ij} = w_{ij}(old) + \alpha(t) * (x_i^k - w_{ij}(old))$

where α is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data. After training the SOM network, trained weights are used for clustering new examples. A new example falls in the cluster of winning vectors.

SOM converts the complex statistical relationships between data into simple geometric relationships of their image points on a low-dimensional display, usually a regular two-dimensional grid of neurons. As the SOM thereby compresses information, while preserving the most important topological and statistical relationships of the primary data elements on the display, it may also be thought to produce some kind of abstractions. These characteristics, abstraction, dimensionality reduction, and visualization in synergy with clustering, have been utilized in a widespread and extensive set of data analysis tasks.

SOM Algorithm

Training:

Step 1: Initialize the weights w_{ij} random value may be assumed. Initialize the learning rate α.

Step 2: For each neuron j, compute the squared Euclidean distance between the input vector x and the weight vector w_j:

$$D(j) = \sum \left(w_{ij} - x_i\right)^2$$

where i=1 to n and j=1 to m

Step 3: Winning Neuron Selection. Identify the winning neuron J such that D(J) is minimal. $J = \arg\min_j D(j)$

Step 4: For each j within a specific neighborhood of j and for all i, calculate the new weight.

$w_{ij}(new) = w_{ij}(old) + \alpha(t) * (x_i^k - w_{ij}(old))$

Step 5: Update the learning rule by using: $\alpha(t+1) = 0.5 * t$

Step 6: Test the Stopping Condition.

As training progresses, similar input vectors are grouped together on nearby neurons, effectively forming clusters on the map. This makes SOM a powerful tool for visualizing patterns, clustering, and detecting anomalies in complex datasets.

5.2. Clustering evaluation metrics

Various unsupervised cluster evaluation measure can be used to approximately the correct or natural number of clusters. The greater the similarity within a group and the greater difference between groups, the better the clustering. Clustering evaluation metrics can be classified based on whether they require ground truth labels (external evaluation metrics) or not (internal evaluation metrics).

5.2.1. Internal evaluation metrics

These evaluate the clustering based only on the data itself and the resulting cluster structure (no ground truth labels required).

Sum of Squared Error (SSE)

SSE is the sum of the squared differences between each observation and its group's mean. It can be used as a measure of variation within a cluster. If all cases within a cluster are identical the SSE would then be equal to 0. SSE is the simplest and most widely used in K-means. It is calculated as:

$$SSE = \sum_{k=1}^{K} \sum_{\forall x_i \in C_k} \|x_i - \overline{x_k}\|^2$$

Where C_k is the set of instances in cluster k; $\overline{x_k}$ is the mean of cluster k. Clustering methods that minimize the SSE, criterion is often called minimum variance partitions.

Calinski-Harabasz Index

The Calinski-Harabasz Index (CHI) measures how good the clusters are in a dataset. It looks at:

- How close the points are inside each cluster?
- How far apart the clusters are?

A higher score is better, as it means the clusters are tight and well-separated. It helps determine the ideal number of clusters. CHI reflecting the within-cluster scatter and the between-cluster scatter. CHI is represented as follows:

$$CHI = \frac{B}{W} \times \frac{N-K}{K-1}$$

where,

- B is the sum of squares between clusters.
- W is the sum of squares within clusters.
- N is the total number of data points.
- K is the number of clusters.

Calculating between group sum of squares (B)

$$B = \sum_{k=1}^{K} n_k \times \|C_k - C\|^2$$

where,

- $n_k$is the number of observation in cluster 'k'
- C_k is the centroid of cluster 'k'
- C is the centroid of the dataset
- K is number of clusters

Calculating within the group sum of squares (W)

$$W = \sum_{i=1}^{n_k} \|X_{ik} - C_k\|^2$$

where,

- X_{ik} is the i[th] observation of cluster 'k'
- C_k is the centroid of cluster 'k'

Silhouette score

The Silhouette Score is a way to measure how good the clusters are in a dataset. It helps us understand how well the data points have been grouped. The score ranges from -1 to 1.

- A score close to 1 means a point fits really well in its group (cluster) and is far from other groups.
- A score close to 0 means the point is on the border between two clusters.
- A score close to -1 means the point might be in the wrong cluster.

For a data point i, the silhouette coefficient $S(i)$ is defined as:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

With $\mathrm{a(i)}$ is the average distance from i to other data points in the same cluster and $\mathrm{b(i)}$ is the smallest average distance from i to data points in a different cluster.

Davies-Bouldin Index (DBI)

The Davies-Bouldin Index (DBI) aims to assess the quality of clustering by examining intra-cluster similarity and inter-cluster differences. Lower DBI values (close to 0) indicate better clustering (low intra-cluster distances, high inter-cluster distances) and higher DBI values suggest poorer clustering (clusters are overlapping or not well separated). For a clustering result with k clusters, the DBI is defined as:

$$DBI = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{S_i + S_j}{M_{ij}} \right)$$

Where:

- $\mathrm{S_i}$ is the average distance between each point in cluster i and the centroid of cluster i (measures dispersion within the cluster).
- $\mathrm{M_{ij}}$is the distance between the centroids of clusters i and j (measures separation between clusters).

- The term $\frac{S_i+S_j}{M_{ij}}$ reflects the similarity between cluster i and cluster j.

Dunn index

The Dunn Index evaluates the quality of clustering by comparing the smallest distance between clusters (inter-cluster separation) to the largest cluster diameter (intra-cluster compactness).

$$Dunn = \frac{\min_{i \neq j} d(C_i,\ C_j)}{\max_{1 \leq i \leq K} \delta(C_k)}$$

Where:

- $d(C_i,\ C_j)$: Distance between clusters C_i and C_j (e.g., distance between centroids or minimum distance between points).
- $\delta(C_k)$: Diameter of cluster C_k, typically the maximum distance between any two points in cluster C_k.

Higher values of the Dunn Index indicate better clustering performance. This is because a higher value suggests that the clusters are well separated, meaning the distance between clusters (the numerator of the formula) is large, and the clusters themselves are compact, meaning the internal spread or diameter of each cluster (the denominator) is small. On the other hand, lower Dunn Index values typically indicate poor clustering structure, where clusters are either overlapping or not clearly separated. Dunn Index is sensitive to noise and outliers, which can distort distance and diameter measurements. It is most effective when the expected clusters are compact and well-separated, as it emphasizes clear boundaries between clusters.

5.2.2. External evaluation metrics

External measures can be useful for examining whether the structure of the clusters match to some predefined clustering of the instances (ground truth).

Rand index

The Rand index (Rand, 1971) is a simple criterion used to compare an induced clustering structure (C_1) with a given clustering structure (C_2). Let a be the number of pairs of instances that are assigned to the same cluster in C_1 and in the same cluster in C_2; b be the number of pairs of instances that are in the same cluster in C_1, but not in the same cluster in C_2; c be the number of pairs of instances that are in the same cluster in C_2, but not in the same cluster in C_1; and d be the number of pairs of instances that are assigned to different clusters in C_1 and C_2. The quantities a and d can be interpreted as agreements, and b and c as disagreements. The Rand index (RI) is defined as:

$$RI = \frac{a+d}{a+b+c+d}$$

The Rand index lies between 0 and 1. When the two partitions agree perfectly, the Rand index is 1. A problem with the Rand index is that its expected value of two

random clustering does not take a constant value (such as zero). Hubert and Arabie (1985) suggest an adjusted Rand index that overcomes this disadvantage.

Adjust Rand index

Adjust rand index (ARI) measures the similarity between two clusterings (predicted vs. ground truth), adjusted for chance. It improves upon the Rand Index (RI) by correcting for the possibility that random labeling could result in high RI scores. ARI checks how well the pairs of points are grouped:

- Are the same pairs together in both the real and predicted clusters?
- Are different pairs also kept apart correctly?

The score ranges from -1 to 1:

- 1 means perfect match - the clustering is exactly right.
- 0 means random guess - no better than chance.
- Below 0 means worse than random - very poor clustering.

Adjusted Rand Index (ARI) is calculated as:

$$\text{ARI} = \frac{RI - Expected_{RI}}{(\max(RI) - Expected_{RI})}$$

With $Expected_{RI}$ is the expected value of the Rand Index.

Mutual Information

Mutual information (MI) compares how much the true cluster labels match with the predicted labels. It shows how much knowing about one variable helps us predict the other. The more agreement there is, the higher the score.

- Higher values mean better agreement between the clusters.
- Zero means no agreement at all.

MI between true labels Y and predicted labels Z is calculated as:

$$MI(y,z) = \sum_i \sum_j p(y_i, z_j) . \log\left(\frac{p(y_i, z_j)}{p(y_i).p(z_j)}\right)$$

where,

- y_i is a true label.
- z_j is a predicted label.
- $p(y_i, z_j)$ is the joint probability of y_i and z_j .
- $p(y_i)$ and $p(z_j)$ are the marginal probabilities.

These clustering metrics help in evaluating the quality and performance of clustering algorithms, allowing for informed decisions when selecting the most suitable clustering solution for a given dataset.

Conclusion

This chapter presented the fundamental concepts and techniques of clustering, an unsupervised learning approach used to group similar data instances without predefined labels. We outlined the main clustering families, including partitioning

methods such as K-means, hierarchical approaches, and density-based techniques like DBSCAN, highlighting their strengths, limitations, and suitable application contexts. Key challenges such as determining the optimal number of clusters, sensitivity to initialization, and handling noise or irregular cluster shapes were also discussed. Finally, we reviewed internal and external evaluation measures used to assess clustering quality. Overall, the chapter provided a comprehensive foundation for understanding how clustering algorithms operate and how to evaluate their effectiveness in practical scenarios.

PRACTICAL WORK 05

CLUSTERING

2. Aim of the practice

Learn to perform clustering using:

- K-Means
- Agglomerative Hierarchical Clustering (AHC) with dendrogram
- Density-Based Clustering (DBSCAN)

Part One: Clustering using K-means

Exercise 01.

Import libraries and generate dataset of moons.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from scipy.cluster.hierarchy import dendrogram, linkage
```

Generate synthetic dataset:

```
# Generate synthetic dataset
X, _ = make_moons(n_samples=600, noise=0.06, random_state=42)
```

1. Scalar the dataset using StandardScaler.
2. Use a scatter plot to display the graphical representation of the dataset, with *Feature 1* on the x-axis and *Feature 2* on the y-axis.

Exercise 02. Kmeans with k=2.

```
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)
```

1. Give me the result of kmeans.
2. What is the value of silhouette score?

Exercise 03. AHC

1. Draw the Dendrogram using:

```
Z = linkage(X_scaled, method='ward')

plt.figure(figsize=(10,5))
dendrogram(Z, truncate_mode='level', p=5)
```

2. Show the AHC using scatter

```
ahc = AgglomerativeClustering(n_clusters=2, linkage='ward')
ahc_labels = ahc.fit_predict(X_scaled)
```

3. Calculate the silhouette score.

Exercise 04. DBSCAN

```
dbscan = DBSCAN(eps=0.25, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_scaled)

plt.figure(figsize=(6,6))
```

1. Show the DBSCAN using scatter.
2. Compare the result.

Solution of practical work 05.

Solution of

Practical works

SOLUTION OF PRACTICAL WORK 01

Exercise 01:

```
import numpy as np
a= np.array([[1, 2, 3], [4, 5, 6]])
print(a)
```

```
type(a)
```
```
numpy.ndarray
```

```
type([0, 1, 2])
```
```
list
```

```
np.shape(a)
```
```
(2, 3)
```

```
a[0]
```
```
array([1, 2, 3])
```

```
a[0,0]
```
```
1
```

```
a[:, 0]
```
```
array([1, 4])
```

```
np.amax(a)
```
```
6
```

```
len(a)
```
```
2
```

```
b=a[:, (1,2)]
b
```
```
array([[2, 3],
       [5, 6]])
```

```
c=np.sin(a)
c
```
```
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ]])
```

```
len(a[0])
```
```
3
```

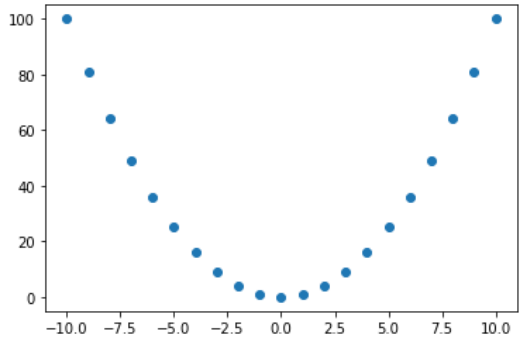
Exercise 02:

```
array = []

for i in range(1, 6):   # rows: 1 to 5
    row = []
    for j in range(0, 3):  # columns: 3 values per row
        row.append(i + 5 * j)
    array.append(row)

for row in array:
    print(row)
```

Exercise 03:

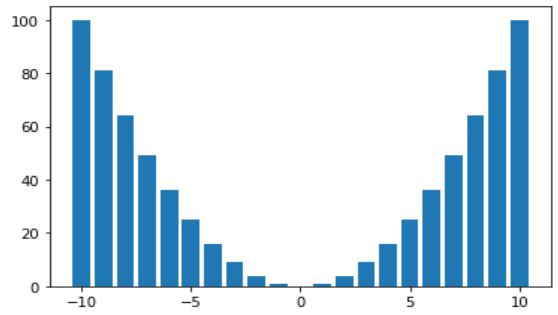
```
import matplotlib.pyplot as plt
x = np.arange(-10, 11)
y = x**2
plt.scatter(x, y)
```

```
<matplotlib.collections.PathCollection at 0x27de2feb448>
```

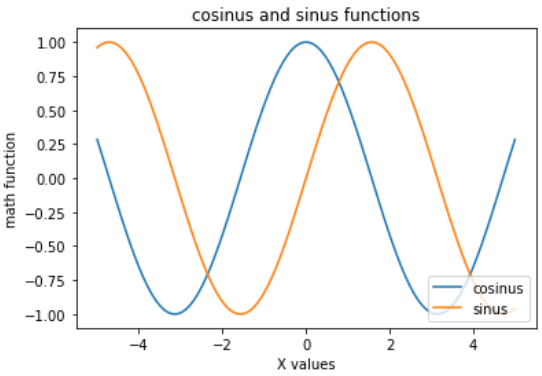



```
import matplotlib.pyplot as plt
x = np.arange(-10, 11)
y = x**2
plt.bar(x, y)
```

```
<BarContainer object of 21 artists>
```



```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5, 5, 100)
y = np.cos(x)
z=np.sin(x)
plt.plot(x, y, label = 'cosinus')
plt.plot(x, z, label = 'sinus')
plt.title('cosinus and sinus functions')
plt.xlabel('X values')
plt.ylabel('math function')
plt.legend(loc='lower right')
```



Exercise 04:

```
import pandas as pd
df=pd.DataFrame([student, ST])
df
```

	Name	Age	Level
0	Nacer	19	2RTW
1	Bader	20	3RTW

Read CSV: Answer of the question

```
s=pd.read_csv("stds.csv")
s
```

	Unnamed: 0	Name	Age	Level
0	0	Ali	18	1Master
1	1	Omar	19	3Licence
2	2	Sami	20	2RTW
3	3	Sarah	18	1RTW

Exercise 04:

```
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
df.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

```
df.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
df.shape
```

```
(303, 14)
```

```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.3
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.6
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0

```
df.corr()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.068001	-0.225439
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210041	-0.280937
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161736	0.433798
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062210	-0.144931
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098803	-0.085239
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032019	-0.028046
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011981	0.137230
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096439	0.421741
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206754	-0.436757
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210244	-0.430696
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104764	0.345877
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151832	-0.391724
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000000	-0.344029
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.344029	1.000000

```
df1 = df.rename(columns={'cp': 'chest pain'})
df1
```

	age	sex	chest pain	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
df1.dtypes
```

```
age           int64
sex           int64
chest pain    int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

SOLUTION OF PRACTICAL WORK 02

Exercice 01 : Import the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
```

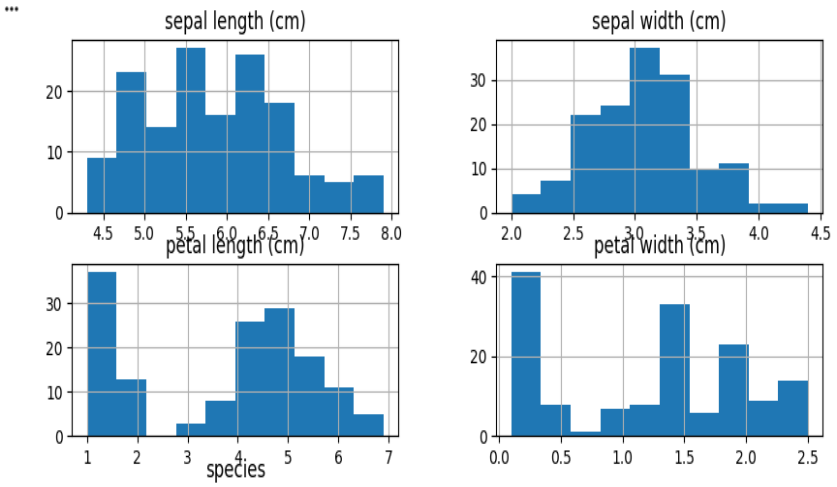
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

Exercice 02 : Data understanding:

```
df['species'] = iris.target

print(df.shape)
print(df.head(10))
print(df.tail(10))
print(df.dtypes)
print(df.info())
print(df['species'].value_counts())
print(df.describe())
print(df.isnull().sum())
corr_matrix=df.corr()
print(corr_matrix)
```

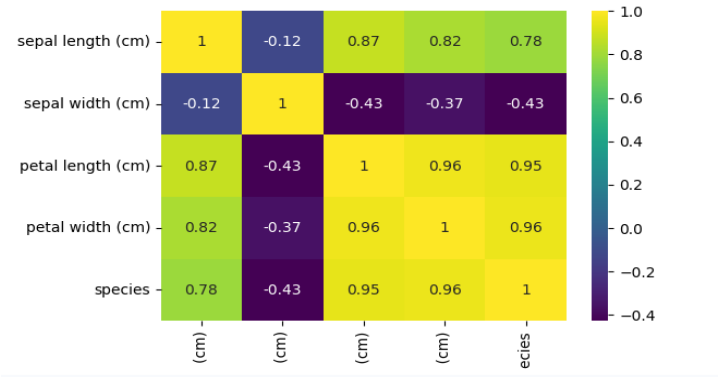
Exercice 03 : Univariate Analysis:

```
df.hist(figsize=(10, 6))
plt.show()

plt.figure(figsize=(8,5))
df.boxplot()
plt.show()
```



Exercice 04 : Bivariate Analysis:

```
corr = df.corr(numeric_only=True)
plt.figure(figsize=(6,4))
sns.heatmap(corr, annot=True, cmap='viridis')
plt.show()

sns.scatterplot(data=df, x='sepal length (cm)', y='sepal width (cm)', hue='species')
plt.show()

sns.scatterplot(data=df, x='petal length (cm)', y='petal width (cm)', hue='species')
plt.show()

sns.pairplot(df, hue='species')
plt.show()
```



Exercice 05 : Data preprocessing:

```
df_missing = df.copy()

# randomly introduce 5% missing values
for col in df_missing.columns:
    df_missing.loc[df_missing.sample(frac=0.05).index, col] = np.nan

print(df_missing.isnull().sum())
```

```
sepal length (cm)    8
sepal width (cm)     8
petal length (cm)    8
petal width (cm)     8
species              8
dtype: int64
```

```
df_missing['sepal length (cm)'].fillna(df_missing['sepal length (cm)'].mean(), inplace=True)
df_missing['sepal width (cm)'].fillna(df_missing['sepal width (cm)'].median(), inplace=True)
df_missing['petal length (cm)'].fillna(0, inplace=True)
df_missing['petal width (cm)'].fillna(method='ffill', inplace=True)
df_missing['species'] = df_missing['species'].fillna(df_missing['species'].mode()[0])

print(df_missing.isnull().sum())
```

```
Q1 = df['sepal length (cm)'].quantile(0.25)
Q3 = df['sepal length (cm)'].quantile(0.75)
IQR = Q3 - Q1

lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

outliers = df[(df['sepal length (cm)'] < lower) | (df['sepal length (cm)'] > upper)]
print("Number of outliers:", len(outliers))

df_no_outliers = df[
    (df['sepal length (cm)'] >= lower) &
    (df['sepal length (cm)'] <= upper)
]
print("Rows after removing outliers:", df_no_outliers.shape)
```

```
Number of outliers: 0
Rows after removing outliers: (150, 5)
```

Exercice 06 : Encoding & Scaling

```
scaler = MinMaxScaler()
numerical_cols = ['sepal length (cm)', 'sepal width (cm)',
                  'petal length (cm)', 'petal width (cm)']

df_missing[numerical_cols] = scaler.fit_transform(df_missing[numerical_cols])
print(df_missing.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0           0.222222          0.625000           0.202899          0.041667
1           0.166667          0.416667           0.202899          0.041667
2           0.111111          0.416667           0.188406          0.041667
3           0.083333          0.458333           0.217391          0.041667
4           0.194444          0.666667           0.202899          0.041667

   species
0      0.0
1      0.0
2      0.0
```

SOLUTION OF PRACTICAL WORK 03

ASSOCIATION RULES MINING

Exercise 01. Creation of transactional dataset

```
from mlxtend.frequent_patterns import apriori, fpgrowth
from mlxtend.frequent_patterns import association_rules
import pandas as pd

# -------------------------
# Step 1: Define dataset
# -------------------------
transactions = [
    ['Milk', 'Bread', 'Butter'],
    ['Juice', 'Bread'],
    ['Milk', 'Bread', 'Juice'],
    ['Bread', 'Butter'],
    ['Milk', 'Bread', 'Butter', 'Juice']
]
print(transactions)
```

Exercise 02. One-Hot Encoding

```
all_items = sorted(set(item for sublist in transactions for item in sublist))
encoded_vals = []
for t in transactions:
    encoded_vals.append({item: (item in t) for item in all_items})

df = pd.DataFrame(encoded_vals)
print("One-hot encoded dataset:\n", df)
```

```
One-hot encoded dataset:
    Bread  Butter  Juice   Milk
0   True    True  False   True
1   True   False   True  False
2   True   False   True   True
3   True    True  False  False
4   True    True   True   True
```

Exercise 03. Apriori Algorithm

```
min_support = 0.6  # 60%
frequent_itemsets_apriori = apriori(df, min_support=min_support, use_colnames=True)
print("\nFrequent itemsets (Apriori):")
print(frequent_itemsets_apriori)

# Optional: Generate association rules
rules_apriori = association_rules(frequent_itemsets_apriori, metric="lift", min_threshold=1.0)
print("\nAssociation rules (Apriori):")
print(rules_apriori[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

```
Frequent itemsets (Apriori):
   support         itemsets
0      1.0          (Bread)
1      0.6         (Butter)
2      0.6          (Juice)
3      0.6           (Milk)
4      0.6  (Butter, Bread)
5      0.6   (Juice, Bread)
6      0.6    (Milk, Bread)

Association rules (Apriori):
  antecedents consequents  support  confidence  lift
0    (Butter)     (Bread)      0.6         1.0   1.0
1     (Bread)    (Butter)      0.6         0.6   1.0
2     (Juice)     (Bread)      0.6         1.0   1.0
3     (Bread)     (Juice)      0.6         0.6   1.0
4      (Milk)     (Bread)      0.6         1.0   1.0
5     (Bread)      (Milk)      0.6         0.6   1.0
```

Exercise 04: FP-Growth Algorithm

```
frequent_itemsets_fpgrowth = fpgrowth(df, min_support=min_support, use_colnames=True)
print("\nFrequent itemsets (FP-Growth):")
print(frequent_itemsets_fpgrowth)

# Optional: Generate association rules
rules_fpgrowth = association_rules(frequent_itemsets_fpgrowth, metric="lift", min_threshold=1.0)
print("\nAssociation rules (FP-Growth):")
print(rules_fpgrowth[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

```
Frequent itemsets (FP-Growth):
   support         itemsets
0      1.0          (Bread)
1      0.6           (Milk)
2      0.6         (Butter)
3      0.6          (Juice)
4      0.6    (Milk, Bread)
5      0.6  (Butter, Bread)
6      0.6   (Juice, Bread)

Association rules (FP-Growth):
  antecedents consequents  support  confidence  lift
0      (Milk)     (Bread)      0.6         1.0   1.0
1     (Bread)      (Milk)      0.6         0.6   1.0
2    (Butter)     (Bread)      0.6         1.0   1.0
3     (Bread)    (Butter)      0.6         0.6   1.0
4     (Juice)     (Bread)      0.6         1.0   1.0
5     (Bread)     (Juice)      0.6         0.6   1.0
```

Exercise 05: Analysis

1. Both algorithms generate the same frequent itemsets because they are deterministic with the same support threshold. Number of itemsets is identical: 4 (1-itemsets) + 4 (2-itemsets) + 1 (3-itemsets) = 9 frequent itemsets.

2. Discuss which algorithm is more efficient for this dataset and why

   For small datasets like this one: Apriori and FP-Growth perform similarly. Efficiency difference is negligible.

   For large datasets: Apriori is less efficient because it generates many candidate itemsets at each iteration and repeatedly scans the dataset.

   FP-Growth is more efficient because it builds an FP-tree and mines frequent itemsets without generating all candidates.

SOLUTION OF PRACTICAL WORK 04

CLASSIFICATION WITH KNN AND ANN

Exercise 01. Classification using KNN

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# KNN classifier
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluation
print("KNN Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Exercise 2: Classification using ANN

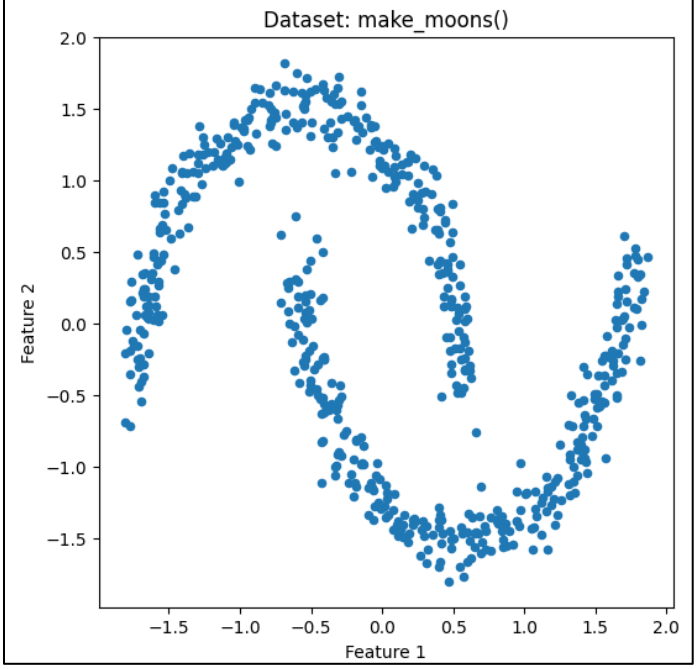
```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix

# -------------------------
# Step 1: Load and preprocess dataset
# -------------------------
iris = load_iris()
X = iris.data      # features
y = iris.target    # labels (0,1,2)

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# One-hot encode labels
y_categorical = to_categorical(y, num_classes=3)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_categorical, test_size=0.2, random_state=42)
```

```
# -------------------------
# Step 2: Build feedforward ANN
# -------------------------
model = Sequential([
    Dense(16, input_dim=4, activation='relu'),   # hidden layer
    Dense(16, activation='relu'),                # optional second hidden layer
    Dense(3, activation='softmax')               # output layer for 3 classes
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# -------------------------
# Step 3: Train the model
# -------------------------
history = model.fit(X_train, y_train, epochs=50, batch_size=8, validation_split=0.2)

# -------------------------
# Step 4: Evaluate the model
# -------------------------
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

# Predict classes
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)
y_true = np.argmax(y_test, axis=1)

# Print classification report and confusion matrix
print("\nClassification Report:")
print(classification_report(y_true, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred))
```

SOLUTION OF PRACTICAL WORK 05

Exercise 01.

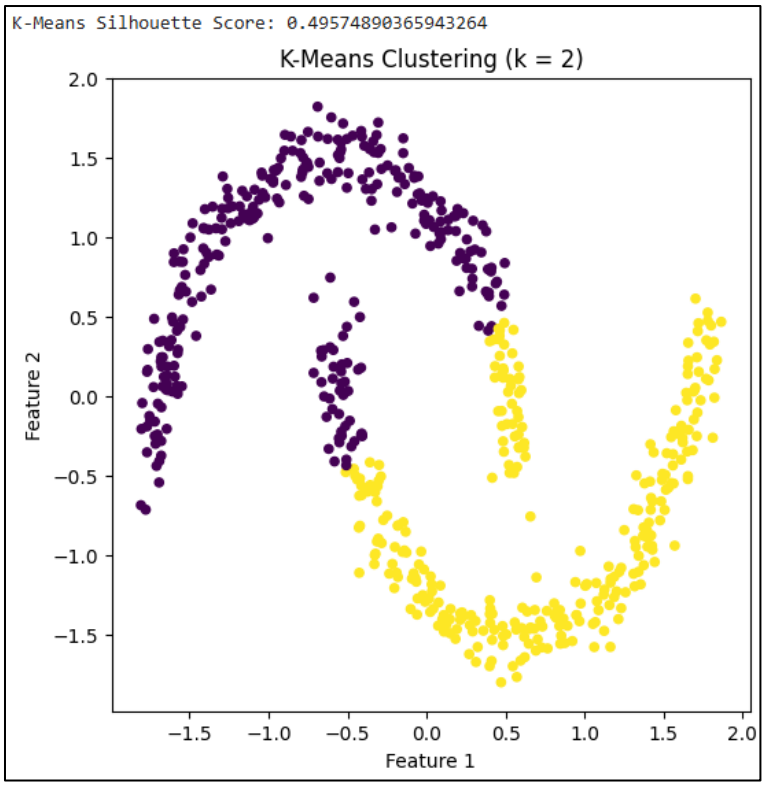
```
# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

plt.figure(figsize=(6,6))
plt.scatter(X_scaled[:,0], X_scaled[:,1], s=20)
plt.title("Dataset: make_moons()")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



Exercise 02. Kmeans with k=2.

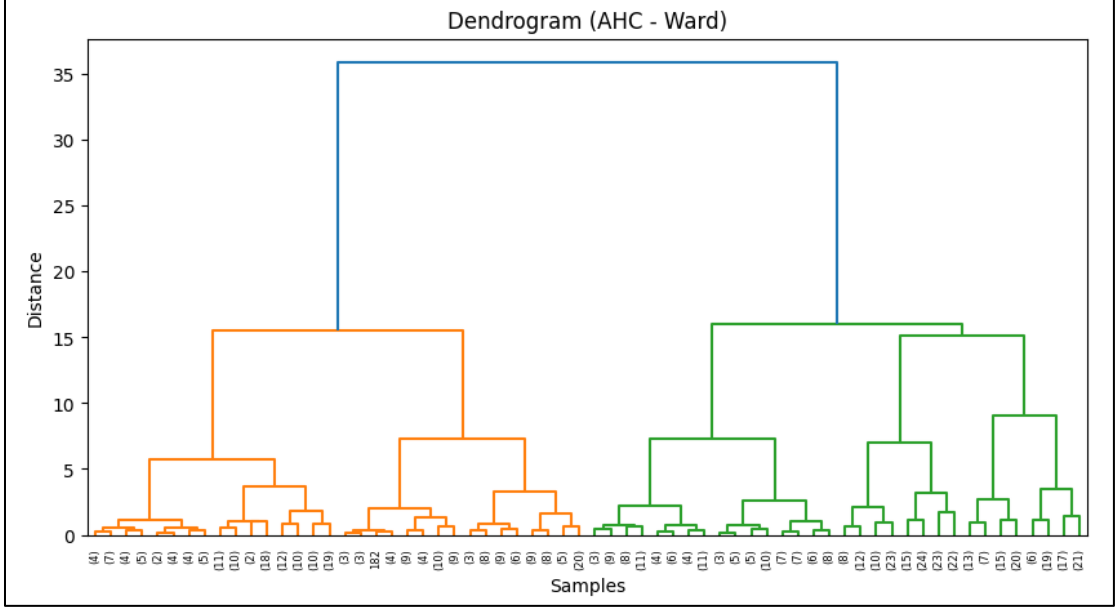
```
from sklearn.metrics import silhouette_score

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# Compute silhouette score
kmeans_sil = silhouette_score(X_scaled, kmeans_labels)
print("K-Means Silhouette Score:", kmeans_sil)

plt.figure(figsize=(6,6))
plt.scatter(X_scaled[:,0], X_scaled[:,1], c=kmeans_labels, cmap='viridis', s=20)
plt.title("K-Means Clustering (k = 2)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
K-Means Silhouette Score: 0.49574890365943264
```



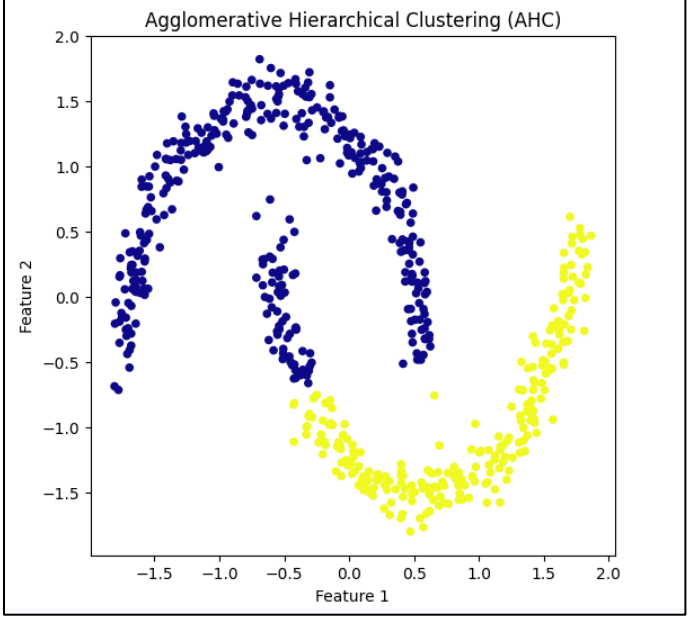
Exercise 03. AHC

```
plt.title("Dendrogram (AHC - Ward)")
plt.xlabel("Samples")
plt.ylabel("Distance")
plt.show()
```



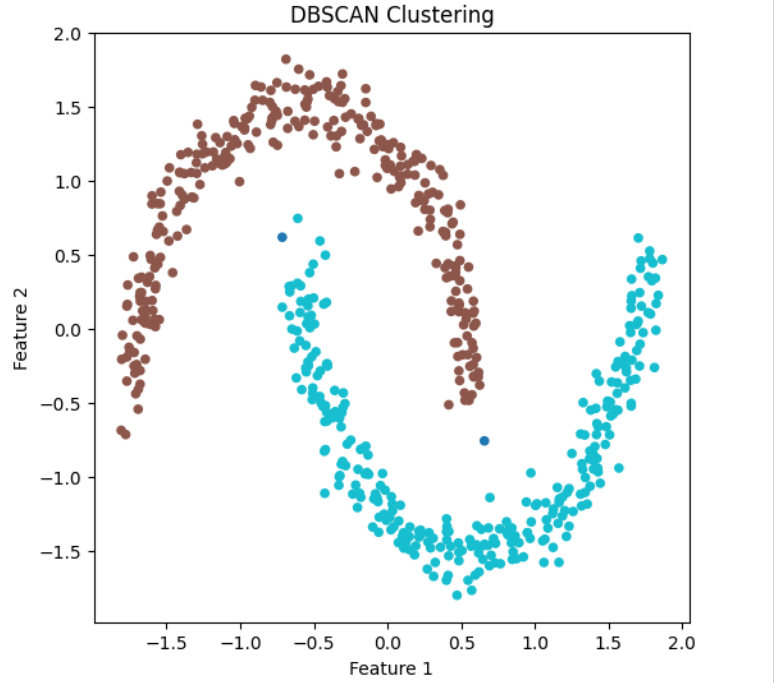

```
ahc_sil = silhouette_score(X_scaled, ahc_labels)
print("AHC Silhouette Score:", ahc_sil)

AHC Silhouette Score: 0.4600513057915405
```

```
plt.figure(figsize=(6,6))
plt.scatter(X_scaled[:,0], X_scaled[:,1], c=ahc_labels, cmap='plasma', s=20)
plt.title("Agglomerative Hierarchical Clustering (AHC)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



Exercise 04. DBSCAN

```
plt.figure(figsize=(6,6))
plt.scatter(X_scaled[:,0], X_scaled[:,1], c=dbscan_labels, cmap='tab10', s=20)
plt.title("DBSCAN Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



For the *make_moons()* dataset, DBSCAN clearly provides the most accurate and meaningful clustering**,** as it captures the intrinsic curved structure of the data and

handles noise robustly. AHC offers moderate performance depending on the linkage criterion, while K-Means consistently underperforms due to its assumption of convex cluster shapes.

References

- Aggarwal, C. C. (2015). Data mining: the textbook (Vol. 1, No. 3). New York: springer.
- Berry, M., and Linoff, G. (2012). Data Mining Techniques: For Marketing, Sales and Customer Relationship Management.
- Nasser, F. K., & Behadili, S. F. (2022). A review of Data Mining and knowledge discovery approaches for bioinformatics. Iraqi Journal of Science, 3169-3188.
- Zong, R. Xia, and J. Zhang. Text Data Mining. Springer Singapore, 2021.
- Lu, Y. J., Lee, W. C., & Wang, C. H. (2023). Using data mining technology to explore causes of inaccurate reliability data and suggestions for maintenance management. Journal of Loss Prevention in the Process Industries, 83, 105063.
- Durugkar, S. R., Raja, R., Nagwanshi, K. K., & Kumar, S. (2022). Introduction to data mining. Data Mining and Machine Learning Applications, 1-19.
- Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. SN computer science, 2(3), 160.
- Shimaoka, A. M., Ferreira, R. C., & Goldman, A. (2024). The evolution of CRISP-DM for data science: Methods, processes and frameworks. SBC Reviews on Computer Science, 4(1), 28-43.
- James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). Statistical learning. In An introduction to statistical learning: With applications in Python (pp. 15-67). Cham: Springer International Publishing.
- Carpenter, J. R., & Smuk, M. (2021). Missing data: A statistical framework for practice. Biometrical Journal, 63(5), 915-947.
- Bardab, S. N., Ahmed, T. M., & Mohammed, T. A. A. (2021). Data mining classification algorithms: An overview. Int. J. Adv. Appl. Sci, 8(2), 1-5.
- Molina Menéndez, E., & Parraga Alava, J. (2024). Artificial Neural Networks for Classification Tasks: A Systematic Literature Review. Enfoque UTE, 1-10.
- Xie, Y., Shekhar, S., & Li, Y. (2022). Statistically-robust clustering techniques for mapping spatial hotspots: A survey. ACM Computing Surveys (CSUR), 55(2), 1-38.
- Sarker, I. H. (2021). Data science and analytics: an overview from data-driven smart computing, decision-making and applications perspective. SN Computer Science, 2(5), 377.
- Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. SN computer science, 2(3), 160.
- Hu, L., Liu, H., Zhang, J., & Liu, A. (2021). KR-DBSCAN: A density-based clustering algorithm based on reverse nearest neighbor and influence space. Expert Systems with Applications, 186, 115763.
- Oyewole, G. J., & Thopil, G. A. (2023). Data clustering: application and trends. Artificial intelligence review, 56(7), 6439-6475.
- Bhattacharjee, P., & Mitra, P. (2021). A survey of density-based clustering algorithms. Frontiers of Computer Science, 15, 1-27.
- Dafir, Z., Lamari, Y., & Slaoui, S. C. (2021). A survey on parallel clustering algorithms for big data. Artificial Intelligence Review, 54(4), 2411-2443.
- Gan, G., Ma, C., & Wu, J. (2020). Data clustering: theory, algorithms, and applications. Society for Industrial and Applied Mathematics.
- Birjandi, S. M., & Khasteh, S. H. (2021). A survey on data mining techniques used in medicine. Journal of diabetes & metabolic disorders, 20(2), 2055-2071.

- Sousa, M. S., Mattoso, M. L. Q., & Ebecken, N. F. F. (2025). Data mining: a database perspective. WIT Transactions on Information and Communication Technologies, 22.
- Saraswat, P. (2021). Supervised machine learning algorithm: a review of classification techniques. Integrated Emerging Methods of Artificial Intelligence & Cloud Computing, 477-482.
- Cunningham, P., & Delany, S. J. (2021). K-nearest neighbour classifiers-a tutorial. ACM computing surveys (CSUR), 54(6), 1-25.
- Alazaidah, R. (2023, December). A comparative analysis of discretization techniques in machine learning. In 2023 24th International Arab Conference on Information Technology (ACIT) (pp. 1-6). IEEE.
- Zaki. M.J. and Meira.W. (2020) Data Mining and Machine Learning: Fundamental Concepts and Algorithms. Cambridge University Press.
- Farzanehdehkordi, M., Ghaffaripour, S., Tirdad, K., Cruz, A. D., & Sadeghian, A. (2022). A wavelet feature-based neural network approach to estimate electrical arc characteristics. Electric Power Systems Research, 208, 107893.