CENTRE FOR DISTANCE EDUCATION
ANNA UNIVERSITY

DMC8018

MASTER OF COMPUTER APPLICATIONS

DATA WAREHOUSING AND DATA MINING

CENTREFORDISTANCEEDUCATION ANNA UNIVERSITY
CHENNAI–600025

SYLLABUS

UNIT I DATA WAREHOUSE: Data Warehousing - Operational Database Systems vs Data Warehouses - Multidimensional Data Model - Schemas for Multidimensional Databases – OLAP operations – Data Warehouse Architecture – Indexing – OLAP queries & Tools.

UNIT II DATA MINING & DATA PREPROCESSING: Introduction to KDD process – Knowledge Discovery from Databases - Need for Data Preprocessing – Data Cleaning – Data Integration and Transformation – Data Reduction – Data Discretization and Concept Hierarchy Generation.

UNIT III ASSOCIATION RULE MINING: Introduction - Data Mining Functionalities - Association Rule Mining - Mining Frequent Itemsets with and without Candidate Generation - Mining Various Kinds of Association Rules - Constraint-Based Association Mining.

UNIT IV CLASSIFICATION & PREDICTION: Classification vs Prediction – Data preparation for Classification and Prediction – Classification by Decision Tree Introduction – Bayesian Classification – Rule Based Classification – Classification by Back propagation – Support Vector Machines – Associative Classification – Lazy Learners – Other Classification Methods – Prediction – Accuracy and Error Measures – Evaluating the Accuracy of a Classifier or Predictor – Ensemble Methods – Model Section.

UNIT V CLUSTERING: Cluster Analysis: - Types of Data in Cluster Analysis – A Categorization of Major Clustering Methods – Partitioning Methods – Hierarchical methods – Density-Based Methods – Grid-Based Methods – Model-Based Clustering Methods – Clustering High Dimensional Data – Constraint-Based Cluster Analysis – Outlier Analysis.

Unit 01 - Introduction to Data warehousing

In today's data-driven world, businesses generate vast amounts of data from various sources such as transactions, customer interactions, social media, sensors, and more. This data is often scattered across different systems and formats, making it challenging to access, analyze, and derive meaningful insights. To address this challenge, organizations use **data warehousing**.

The Need for Data Warehousing

1. **Data Integration**: Data is often stored in disparate systems, such as relational databases, spreadsheets, and cloud storage. These systems are typically optimized for specific functions, making it difficult to access and analyze data holistically. A data warehouse integrates data from these heterogeneous sources into a single, cohesive repository, ensuring that data is consistent and easily accessible.
2. **Efficient Query Processing**: Operational systems are designed for transaction processing (Online Transaction Processing), where the primary focus is on quick inserts, updates, and deletions. These systems are not optimized for complex queries and analytics. A data warehouse is designed for analytical processing (Online Analytical Processing), enabling faster and more efficient querying, reporting, and analysis.
3. **Historical Data Storage**: In operational systems, data is often purged after a certain period to maintain performance and storage efficiency. However, businesses need to analyze historical trends to make informed decisions. A data warehouse stores historical data, allowing organizations to perform trend analysis, forecasting, and other time-based analyses.
4. **Business Decision-Making**: Modern businesses require timely and accurate information to make strategic decisions. A data warehouse provides a central repository of integrated and cleansed data, making it easier for decision-makers to access the insights they need without relying on IT for every query.
5. **Consistency and Accuracy**: With data coming from multiple sources, there is a risk of inconsistencies and inaccuracies. A data warehouse standardizes and cleanses data, ensuring that the information used for analysis is consistent and accurate.

Real-Life Applications of Data Warehousing

1. **Retail and E-commerce**: Companies like Amazon and Walmart use data warehouses to analyze customer behavior, manage inventory, and optimize supply chains. By integrating data from various sources (e.g., sales transactions, customer feedback, website analytics), they can identify trends, personalize marketing efforts, and improve operational efficiency.
2. **Finance and Banking**: Financial institutions use data warehouses to monitor transactions, detect fraud, and comply with regulations. By aggregating data from different branches and systems, they can analyze customer portfolios, assess risks, and ensure regulatory compliance.

3. **Healthcare**: Hospitals and healthcare providers use data warehousing to store and analyze patient records, treatment outcomes, and research data. This helps in improving patient care, managing resources, and conducting medical research.
4. **Telecommunications**: Telecom companies use data warehouses to analyze network performance, customer usage patterns, and billing information. This helps them optimize network resources, reduce churn, and develop new services.
5. **Government**: Government agencies use data warehousing to analyze census data, manage public resources, and monitor social programs. This enables them to make data-driven decisions and improve public services.

What exactly is a data warehouse?

A data warehouse is a database managed separately from an organization's operational databases. Data warehouse solutions enable the integration of many application systems. They aid information processing by providing a firm foundation of consolidated historical data for analysis.
William H. Inmon, a major architect in the creation of data warehouse systems, defines a data warehouse as "a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process". This concise definition summarizes the key aspects of a data warehouse.

A data warehouse is a semantically consistent data store that serves as a physical implementation of a decision support data model and stores the information on which an enterprise needs to make strategic decisions. A data warehouse is also often viewed as an architecture, constructed by integrating data from multiple heterogeneous sources to support structured and/or ad hoc queries, analytical reporting, and decision making.

Key features of Data warehouse

Subject-oriented: A data warehouse is organized around major subjects, such as customer, supplier, product, and sales. Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the modeling and analysis of data for decision makers. Hence, data warehouses typically provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process.

Integrated: A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and on-line transaction records. Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on.

Time-variant: Data are stored to provide information from a historical perspective (e.g., the past 5–10 years). Every key structure in the data warehouse contains, either implicitly or explicitly, an element of time.

Nonvolatile: A data warehouse is always a physically separate store of data transformed from the application data found in the operational environment. Due to this separation, a data warehouse does not require transaction processing, recovery, and concurrency control mechanisms. It usually requires only two operations in data accessing: initial loading of data and access of data.

Operational Database Systems vs Data Warehouses

We know what a data warehouse is, then what is an operational database system?

An Operational Database System (ODBS) is a type of database that is designed to manage and store data required for day-to-day operations of an organization. These databases are optimized for high-volume transaction processing and are typically used in environments where data is frequently updated, such as customer management systems, inventory control, and financial transactions.

Let us take a look at the key features of ODB as well to understand the differences between ODB and Data warehouse

Key characteristics of an operational database system include:

1. **Real-Time Processing**: ODBs handle real-time data processing, allowing businesses to manage their operations as they happen. This includes tasks like order processing, payments, and customer interactions.
2. **Transactional Support**: They are designed to handle numerous concurrent transactions, ensuring data consistency and integrity. This includes support for ACID (Atomicity, Consistency, Isolation, and Durability) properties.
3. **High Availability**: ODBs are built to be highly available and reliable, minimizing downtime to ensure continuous access to critical data.
4. **Scalability**: These databases can scale to accommodate large volumes of data and a high number of transactions, which is essential for growing businesses.
5. **Short-Term Data Storage**: Typically, operational databases store current, up-to-date data that is frequently accessed and modified. Historical data is often moved to data warehouses for long-term storage and analysis.

Examples of operational databases include MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

The primary function of online operational database systems is to execute online transaction and query processing. These systems are known as OnLine Transaction Processing (OLTP) systems. They oversee an organization's daily operations, including purchasing, inventories, manufacturing, banking, payroll, registration, and accounting.
Data warehouse solutions assist knowledge workers with data analysis and decision-making. These systems may arrange and show data in a variety of formats to meet the unique needs of their users. These systems are called OnLine Analytical Processing (OLAP) systems.

Now, what would be the difference between the ODB and Data warehouses?

Users and System Orientation:
OLTP systems are designed for customers and are used primarily for transaction processing and query handling by clerks, clients, and IT professionals. On the other hand, OLAP systems are oriented towards market analysis, catering to knowledge workers such as managers, executives, and analysts, who require data analysis tools.

Data Contents:
OLTP systems handle current data, which is usually highly detailed and not well-suited for decision-making processes. Conversely, OLAP systems manage vast amounts of historical data, offering tools for summarization and aggregation. They store and manage information at varying levels of detail, which aids in making informed decisions.

Database Design:
OLTP systems typically use an entity-relationship (ER) data model with an application-centric database design. In contrast, OLAP systems generally use a star or snowflake schema and are designed with a subject-oriented approach.

View:
OLTP systems focus on the current data within a specific enterprise or department, with little to no reference to historical data or data from different organizations. In contrast, OLAP systems often span multiple database schema versions due to the evolutionary nature of organizations. They integrate data from various sources and organizations, leading to massive data volumes that are stored across multiple storage media.

Access Patterns:
The access patterns in OLTP systems are characterized by short, atomic transactions, necessitating robust concurrency control and recovery mechanisms. Meanwhile, OLAP systems are primarily accessed through read-only operations, often involving complex queries, as they typically deal with historical data rather than real-time information.
Other distinguishing features between OLTP and OLAP systems include database size, the frequency of operations, and performance metrics as in Table 1.

Table 1: OLTP Vs OLAP

Features	**OLTP (Online Transaction Processing)**	**OLAP (Online Analytical Processing)**
Orientation	Customer-oriented, used by clerks, clients, and IT professionals.	Market-oriented, used by managers, executives, and analysts.
Data Contents	Manages current, highly detailed data.	Manages large amounts of historical data, supports summarization and aggregation.
Database Design	Typically uses ER data model, application-oriented design.	Typically uses star or snowflake schema, subject-oriented design.

View	Focuses on current data within an enterprise or department.	Spans multiple database schema versions, integrates data from various sources.
Access Patterns	Characterized by short, atomic transactions, requiring concurrency control.	Primarily read-only operations, often involving complex queries.

What is a Multidimensional data model?

In the world of data warehousing and OLAP tools, the multidimensional data model is a fundamental concept that allows data to be organized and analyzed from multiple perspectives. This model is visualized as a data cube, which, despite its name, can represent data in more than three dimensions—hence the term *n-dimensional* data cube.

The multidimensional data model is designed to facilitate complex queries and analysis by organizing data into dimensions and facts. Dimensions represent the various perspectives or entities that an organization needs to track, such as time, products, locations, and more. Facts, on the other hand, are the numerical measures that quantify the relationships between these dimensions, such as sales amounts, units sold, or budget allocations.

By using this model, organizations can perform interactive data analysis at multiple levels of abstraction, making it a powerful tool for decision-making and strategic planning. Concept hierarchies, which define levels of granularity within dimensions, further enhance this capability by allowing users to drill down or roll up data to gain insights at different levels of detail.

How exactly is the modeling done?

The key terms needed to understand the modeling are:

- Data cube
- Dimensions
- Dimension table
- Facts
- Fact table

A **data cube** enables data to be modeled and viewed across multiple dimensions. It is defined by two key elements: dimensions and facts.

- **Dimensions**: These are the perspectives or entities for which an organization wants to keep records. For example, an electronics retail chain like AllElectronics might create a sales data warehouse to track sales data with respect to various dimensions like time, item, branch, and location. These dimensions help the organization monitor metrics such as monthly sales per item and the specific branches and locations where these items were sold. Each dimension is typically associated with a **dimension table** that provides additional attributes describing the dimension. For example, a dimension table for the *item* dimension may include attributes such as item name, brand, and type. These tables can be manually defined or automatically generated based on the data.

- **Facts**: These are the numerical measures that quantify the relationships between the dimensions. In the context of a sales data warehouse, examples of facts might include *dollars sold* (the sales amount in dollars), *units sold* (the number of items sold), and *amount budgeted*. These facts are stored in a **fact table**, which contains the names of the facts along with keys linking to each of the associated dimension tables.

Though the term "cube" might suggest a 3-dimensional structure, in data warehousing, a data cube can have any number of dimensions. To illustrate this concept, consider a simple 2-dimensional data cube, which is essentially a table or spreadsheet. For example, we could look at sales data from AllElectronics for items sold per quarter in a specific city, such as Vancouver. In this 2-dimensional view, sales data for Vancouver could be represented with respect to the time dimension (organized by quarters) and the item dimension (organized by item type), with the fact or measure being the sales amount in thousands of dollars.

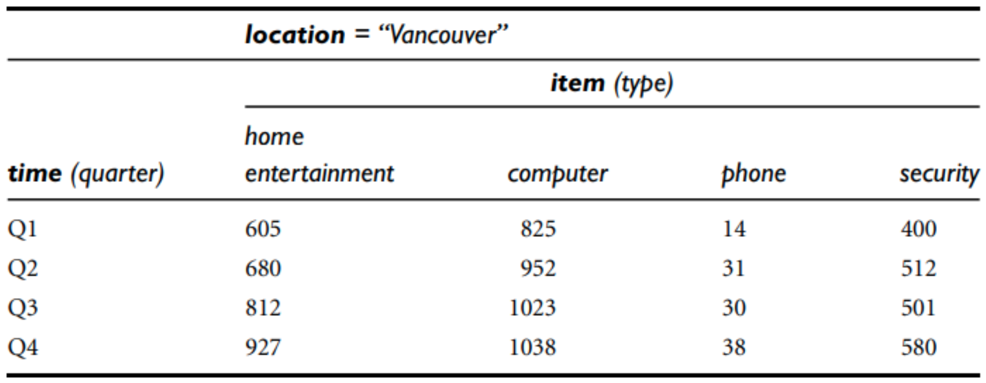
By transitioning from traditional tables and spreadsheets to multidimensional data cubes, organizations gain a powerful tool for conducting complex analyses, enabling them to extract meaningful insights from their data and support better decision-making processes.

Let's take an example for the same

Let us take Electronics sale data and try to model it in 2 dimensions, 3 dimensions and 4 dimensions

A 2-D view of sales data for AllElectronics according to the dimensions, time and item, where the sales are from branches located in the city of Vancouver. The measure displayed is dollars sold (in thousands) as in table 2.

Table 2: Dataset Example



	location = *"Vancouver"*			
	item *(type)*			
time *(quarter)*	*home entertainment*	*computer*	*phone*	*security*
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

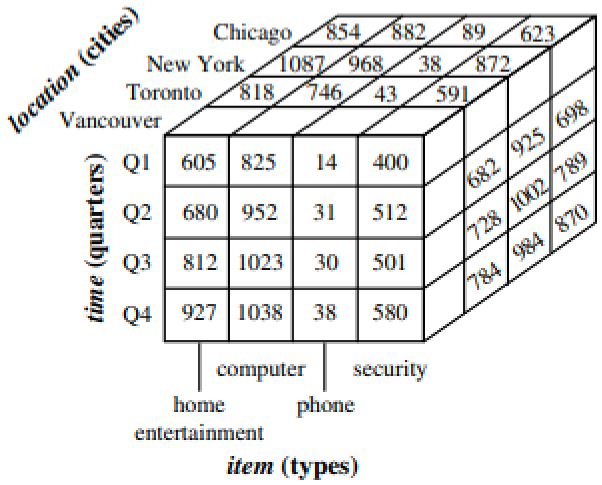


Fig 1.1: 3-D View

A 3-D view of sales data for AllElectronics, according to the dimensions time, item, and location. The measure displayed is dollars sold (in thousands) as in figure 1.1.

Table 3: Dataset Example 2

	location **= "Chicago"**				***location*** **= "New York"**				***location*** **= "Toronto"**				***location*** **= "Vancouver"**			
	item				***item***				***item***				***item***			
time	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580

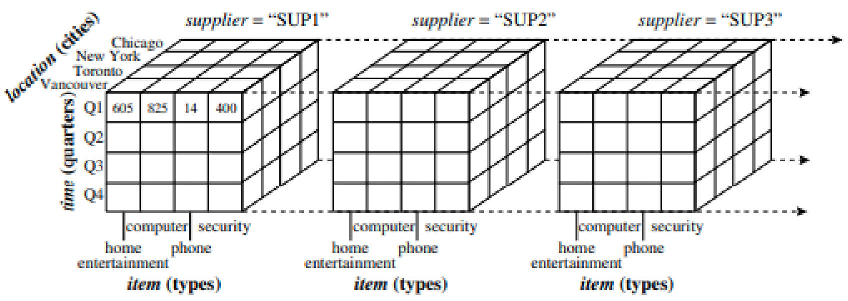


Fig 1.2: 3-D View

2 A 4-D data cube representation of sales data, according to the dimensions time, item, location, and supplier. The measure displayed is dollars sold (in thousands). For improved readability, only some of the cube values are shown in figure 1.3.

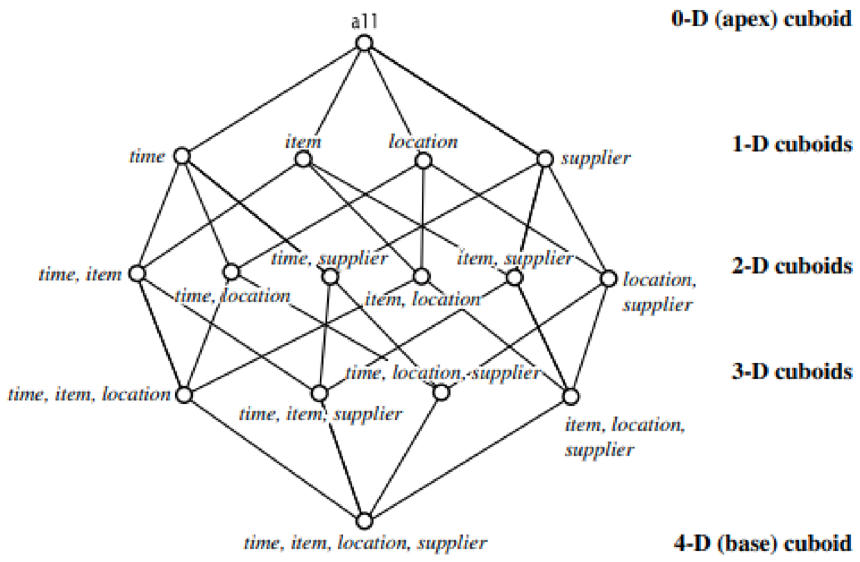


Fig 1.3: 0-D to 4-D View

Schemas for Multidimensional Databases: Stars, Snowflakes, and Fact Constellations

The entity-relationship (ER) model is widely used in designing relational databases, where a database schema consists of a set of entities and the relationships among them. This model is well-suited for online transaction processing (OLTP) systems. However, data warehouses require a concise, subject-oriented schema that supports online analytical processing (OLAP). The most popular data model for a data warehouse is the multidimensional model, which can be represented using a star schema, snowflake schema, or fact constellation schema. Let's explore each of these schema types.

Star Schema:

The star schema is the most common design for data warehouses . It consists of a large central table, known as the fact table, which contains the bulk of the data without redundancy, and a set of smaller dimension tables, one for each dimension as in Figure 1.4 and Figure 1.5. The schema resembles a star, with the dimension tables arranged around the central fact table.

Example:

Consider a star schema for sales in the AllElectronics database. Sales data is analyzed across four dimensions: time, item, branch, and location. The central fact table records sales data and includes keys that link to each of the four dimensions, along with measures like dollars sold and units sold. In this schema, each dimension is represented by a single table. For example, the location dimension table includes attributes like location key, street, city, province or state, and country. However, this design can introduce redundancy. For instance, cities like Vancouver and Victoria, both in British Columbia, Canada, will have repeated province and country information in the location table. Dimension tables can form either a hierarchy (total order) or a lattice (partial order).

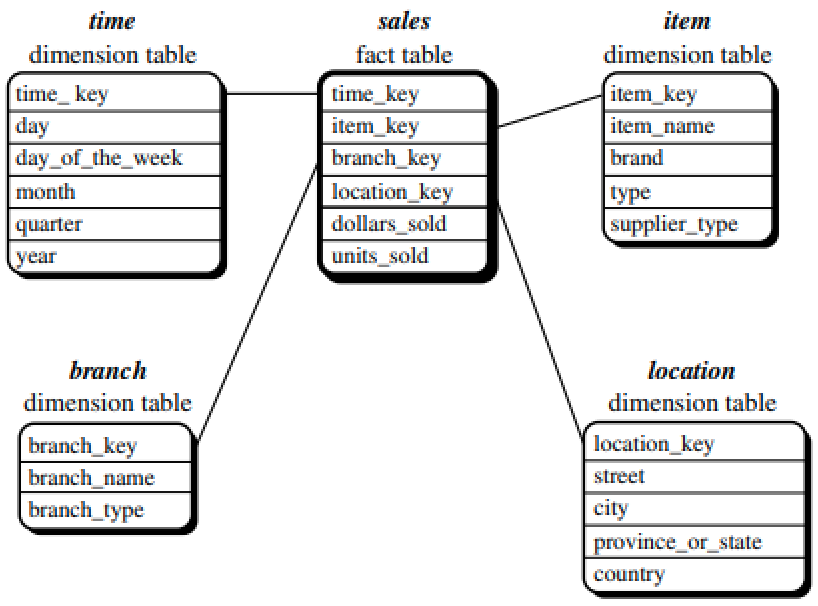


Fig 1.4: Star Schema - Example

Snowflake Schema:

The snowflake schema is a variant of the star schema, where some dimension tables are normalized, further splitting the data into additional tables as in figure 1.6 and figure 1.7. This normalization reduces redundancy, but it also increases the number of joins required to execute a query, which can impact system performance.

Examples:

In a snowflake schema for AllElectronics sales, the sales fact table remains the same as in the star schema. The difference lies in the dimension tables. For instance, the item dimension table in the star schema might be normalized into two separate tables: item and supplier. The item table includes attributes such as item key, item name, brand, type, and supplier key, with the supplier key linked to a supplier table. Similarly, the location dimension table can be normalized into location and city tables, where the city key links to the city table. Further normalization can be applied to attributes like province or state and country if needed.

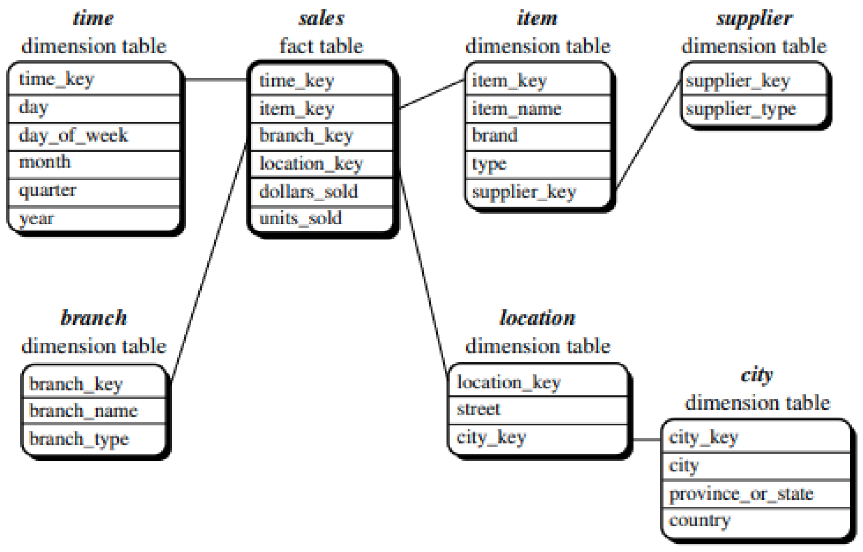


Fig 1.6: Snowflake Schema - Example

Fact Constellation Schema:

For more complex applications, multiple fact tables may share dimension tables. This type of schema, known as a galaxy schema or fact constellation, is essentially a collection of star schemas.

Examples:
A fact constellation schema might include both sales and shipping fact tables as in figure 1.8 and figure 1.9. The sales table is similar to the one in the star schema, while the shipping table includes dimensions such as item key, time key, shipper key, from location, and to location, along with measures like dollars cost and units shipped. The time, item, and location dimension tables are shared between the sales and shipping fact tables.

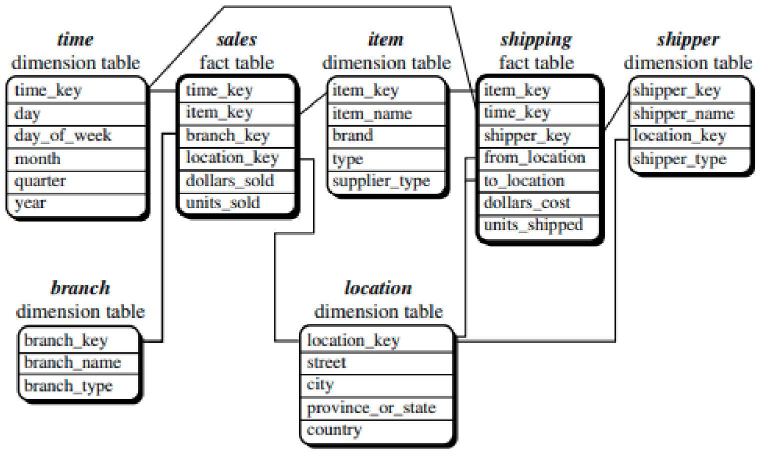


Fig 1.8: Fast Constellation Schema - Example

Table 4: Comparison of Different Schemas

Features	**Star Schema**	**Snowflake Schema**	**Fact constellation Schema**
Elements	Single Fact Table connected to multiple dimension tables with no sub-dimension tables	Single Fact Table connects to multiple dimension tables that connect to multiple sub-dimension tables	Multiple Fact Tables connect to multiple dimension tables that connect to multiple sub-dimension tables
Normalization	Denormalized	Normalized	Normalized
Number of Dimensions	Multiple dimension tables map to a single Fact Table	Multiple dimension tables map to multiple dimension tables	Multiple dimension tables map to multiple Fact Tables
Data Redundancy	High	Low	Low
Performance	Fewer foreign keys resulting in increased performance	Decreased performance compared to Star Schema due to more foreign keys	Decreased performance compared to Star and Snowflake. Used for complex data aggregation
Complexity	Simple, designed to be easy to understand	More complicated compared to Star Schema—can be more challenging to understand	Most complicated to understand. Reserved for highly complex data structures
Storage Usage	Higher disk space due to data redundancy	Lower disk space due to limited data redundancy	Low disk space usage compared to the level of sophistication due to the limited data redundancy
Design Limitations	One Fact Table only, no sub-dimensions	One Fact Table only, multiple sub-dimensions are permitted	Multiple Fact Tables permitted, only first-level dimensions are permitted

Data Warehouses vs. Data Marts:

A data warehouse is an enterprise-wide repository that gathers information on various subjects such as customers, items, sales, assets, and personnel. Fact constellation schemas are typically used for data warehouses because they can model multiple, interrelated subjects. In contrast, a data mart is a departmental subset of a data warehouse that focuses on specific subjects, making it department-wide in scope. Star and snowflake schemas are commonly used for data marts, with the star schema being more popular due to its efficiency and simplicity.

OLAP Operations in Multidimensional Data Models

In a multidimensional model, records are organized across various dimensions, each encompassing multiple levels of abstraction described by concept hierarchies. This organization allows users to view data from different perspectives, making it possible to perform interactive queries and explore the data. Online Analytical Processing supports a user-friendly environment for interactive data analysis through several operations on data cubes, which represent the multidimensional data.

Roll-Up

The roll-up operation, also known as drill-up or aggregation, performs data aggregation by climbing up concept hierarchies, reducing the number of dimensions in a data cube. This operation is like zooming out on the data, showing higher-level summaries.

For example, consider a sales data cube with dimensions for location and time. By performing a roll-up on the location dimension, the data might be aggregated from the city level to the country level. When dimensions are reduced, one or more are removed from the cube. For instance, rolling up by removing the time dimension would result in a cube that only aggregates sales by location.

Example:

Consider temperature data recorded weekly:

Temperature	**64**	**65**	**68**	**69**	**70**	**71**	**72**	**75**	**80**	**81**	**83**	**85**
Week 1	1	0	1	0	1	0	0	0	0	0	1	0
Week 2	0	0	0	1	0	0	1	2	0	1	0	0

To aggregate this data into levels (hot: 80-85, mild: 70-75, cool: 64-69), we can perform a roll-up:

Temperature	**Cool**	**Mild**	**Hot**
Week 1	2	1	1

Week 2	2	1	1

This roll-up operation groups the data by temperature levels.

Drill-Down

Drill-down, also known as roll-down, is the reverse of roll-up. It navigates from summarized data to more detailed data, similar to zooming in on a data cube. Drill-down can be performed by descending a concept hierarchy or adding additional dimensions to the cube.

For example, a drill-down on the time dimension might go from the quarter level to the month level. Alternatively, adding a new dimension, such as customer group, to the cube can also achieve drill-down.

Example:

Drill-down adds more detail to the following temperature data:

Temperature	**Cool**	**Mild**	**Hot**
Day 1	0	0	0
Day 2	0	0	0
Day 3	0	0	1
Day 4	0	1	0
Day 5	1	0	0
Day 6	0	0	0
Day 7	1	0	0
Day 8	0	0	0
Day 9	1	0	0
Day 10	0	1	0
Day 11	0	1	0
Day 12	0	1	0

Day 13	0	0	1
Day 14	0	0	0

Slice

The slice operation creates a subcube by selecting a single value for one or more dimensions, effectively reducing the cube's dimensionality.

Example:

If we select Temperature = Cool, the resulting subcube will be:

Temperature	**Cool**
Day 1	0
Day 2	0
Day 3	0
Day 4	0
Day 5	1
Day 6	1
Day 7	1
Day 8	1
Day 9	1
Day 10	0
Day 11	0
Day 12	0
Day 13	0
Day 14	0

Dice

Dice is an operation that creates a subcube by selecting values for two or more dimensions.

Example:

Applying the selection (Day = 3 OR Day = 4) AND (Temperature = Cool OR Temperature = Hot) to the original data results in the following subcube:

Temperature	**Cool**	**Hot**
Day 3	0	1
Day 4	0	0

Pivot

The pivot operation, also known as rotation, changes the view of the data by rotating the axes. This operation can involve swapping rows and columns or moving a row dimension into a column dimension.

Other OLAP Operations

- **Drill-Through:** Executes queries across multiple fact tables, often utilizing relational SQL to access detailed data in backend relational tables.
- **Ranking:** Allows for ranking top-N or bottom-N elements in lists.
- **Calculations:** Includes operations like moving averages, growth rates, statistical analysis, and other mathematical computations.

OLAP engines offer robust data analysis capabilities, including summarization, aggregation, and hierarchical views at different levels of granularity. They also support advanced analytical models for forecasting, trend analysis, and statistical tasks.

Data Warehouse Architecture

Steps for the Design and Construction of Data Warehouses

This subsection outlines a business analysis framework essential for designing a data warehouse. It also details the fundamental steps involved in the design process.

Designing a Data Warehouse: A Business Analysis Framework

One of the key questions addressed here is, "What benefits do business analysts gain from a data warehouse?" A data warehouse offers several advantages:

1. It can provide a competitive edge by offering critical information that helps measure performance and make necessary adjustments to outperform competitors.
2. It enhances business productivity by efficiently gathering and presenting accurate organizational information.

3. It supports customer relationship management by providing a consistent view of customers and products across various business units and markets.
4. It can lead to cost savings by consistently tracking trends, patterns, and exceptions over extended periods.

To create an effective data warehouse, it is crucial to understand and analyze business needs, which can be achieved by constructing a business analysis framework. Designing a large and complex information system is akin to constructing a large and intricate building, where the owner, architect, and builder each have different perspectives. These perspectives are integrated into a comprehensive framework that reflects both a top-down, business-driven approach and a bottom-up, implementation-driven approach.

Four key perspectives must be considered when designing a data warehouse:

1. **The Top-Down View:** This view involves selecting the relevant information required for the data warehouse, aligned with current and future business needs.
2. **The Data Source View:** This view reveals the information captured, stored, and managed by operational systems. This information can vary in detail and accuracy, from individual data source tables to integrated data source tables, and is often modeled using traditional data modeling techniques like the entity-relationship model or CASE (computer-aided software engineering) tools.
3. **The Data Warehouse View:** This view includes fact tables and dimension tables, representing the information stored within the data warehouse. It also includes precalculated totals and counts, along with details about the data source, date, and time of origin, providing historical context.
4. **The Business Query View:** This perspective represents how end users view and interact with the data in the data warehouse.

Building and utilizing a data warehouse is a complex task that requires a combination of business, technology, and program management skills. Business skills are necessary for understanding how data is stored and managed, building extractors to transfer data from operational systems to the data warehouse, and developing software to refresh the warehouse with updated data. Using a data warehouse also requires an understanding of the significance of the data it contains and the ability to translate business requirements into queries that the warehouse can satisfy. Technology skills are essential for analyzing quantitative information, discovering patterns and trends, predicting future trends, identifying anomalies, and making managerial recommendations. Program management skills are crucial for coordinating the involvement of various technologies, vendors, and end users to deliver results efficiently and cost-effectively.

The Process of Data Warehouse Design

A data warehouse can be developed using a top-down approach, a bottom-up approach, or a combination of both. The top-down approach begins with an overall design and planning, making it suitable when the technology is mature and the business problems are well understood. The bottom-up approach starts with experiments and prototypes, making it ideal during the early stages of business modeling and technology development. It allows organizations to progress with minimal costs and evaluate the benefits before committing to significant investments. The combined approach leverages the strategic planning of the top-down approach while incorporating the rapid implementation and flexibility of the bottom-up approach.

From a software engineering perspective, designing and constructing a data warehouse typically involves the following steps: planning, requirements analysis, problem analysis, warehouse design, data integration and testing, and finally, deployment. Large software systems can be developed using either the waterfall method or the spiral method. The waterfall method involves structured and systematic analysis at each step before moving on to the next, similar to a waterfall flowing from one step to the next. The spiral method, on the other hand, focuses on the rapid development of increasingly functional systems with short intervals between successive releases. This method is particularly suited for data warehouse development, especially for data marts, due to its short turnaround time, quick modifications, and adaptability to new designs and technologies.

The general steps in the warehouse design process are as follows:

1. **Choose a Business Process to Model:** Select a business process such as orders, invoices, shipments, inventory, account administration, sales, or the general ledger. If the process is organizational and involves multiple complex object collections, a data warehouse model should be followed. If the process is departmental and focuses on analyzing a specific type of business process, a data mart model is more appropriate.

2. **Determine the Grain of the Business Process:** The grain refers to the fundamental, atomic level of data to be represented in the fact table for this process, such as individual transactions or daily snapshots.

3. **Select the Dimensions for Each Fact Table Record:** Typical dimensions include time, item, customer, supplier, warehouse, transaction type, and status.

4. **Choose the Measures for Each Fact Table Record:** Common measures include numeric additive quantities like dollars sold and units sold.

Given the complexity and long-term nature of data warehouse construction, it is crucial to clearly define the implementation scope. The goals for the initial implementation should be specific, achievable, and measurable. This includes determining time and budget allocations, selecting a subset of the organization to model, choosing the data sources, and identifying the departments to be served.

After the data warehouse is designed and built, the initial deployment includes installation, roll-out planning, training, and orientation. Platform upgrades and maintenance should also be considered. Data warehouse administration tasks include data refreshment, data source synchronization, disaster recovery planning, access control and security management, data growth management, database performance management, and warehouse enhancement and extension. Scope management involves controlling the number and range of queries, dimensions, and reports, as well as managing the size of the warehouse and the schedule, budget, or resources.

Various tools are available to assist in data warehouse design. These tools offer functionalities such as defining and editing metadata repository contents (including schemas, scripts, and rules), answering queries, generating reports, and exchanging metadata with relational database system catalogs. Planning and analysis tools help assess the impact of schema changes and refresh performance when adjusting refresh rates or time windows.

A Three-Tier Data Warehouse Architecture

Data warehouses typically use a three-tier architecture, as shown in figure 1.10 and outlined below:

1. **Bottom Tier**: This layer is a warehouse database server, typically a relational database system. Data is fed into this tier from operational databases or external sources, such as customer profiles, using back-end tools and utilities. These tools handle data extraction, cleaning, transformation (e.g., merging data from various sources into a uniform format), loading, and refreshing to keep the data warehouse updated. Data extraction relies on application program interfaces (APIs) called gateways, supported by the underlying DBMS. These gateways, such as ODBC (Open Database Connection), OLEDB (Open Linking and Embedding for Databases) by Microsoft, and JDBC (Java Database Connection), allow client programs to generate SQL code to be executed at a server. This tier also includes a metadata repository, which stores information about the data warehouse and its contents.
2. **Middle Tier**: The middle tier is an OLAP (Online Analytical Processing) server. It can be implemented using either a relational OLAP (ROLAP) model, which extends relational DBMS to handle multidimensional data, or a multidimensional OLAP (MOLAP) model, which directly implements multidimensional data and operations. The OLAP server supports complex analytical queries and provides fast response times, crucial for decision-making processes.
3. **Top Tier**: The top tier is a front-end client layer, which includes tools for querying, reporting, analysis, and data mining (e.g., trend analysis and prediction). This layer is the interface through which end-users interact with the data warehouse, allowing them to perform various analytical tasks.

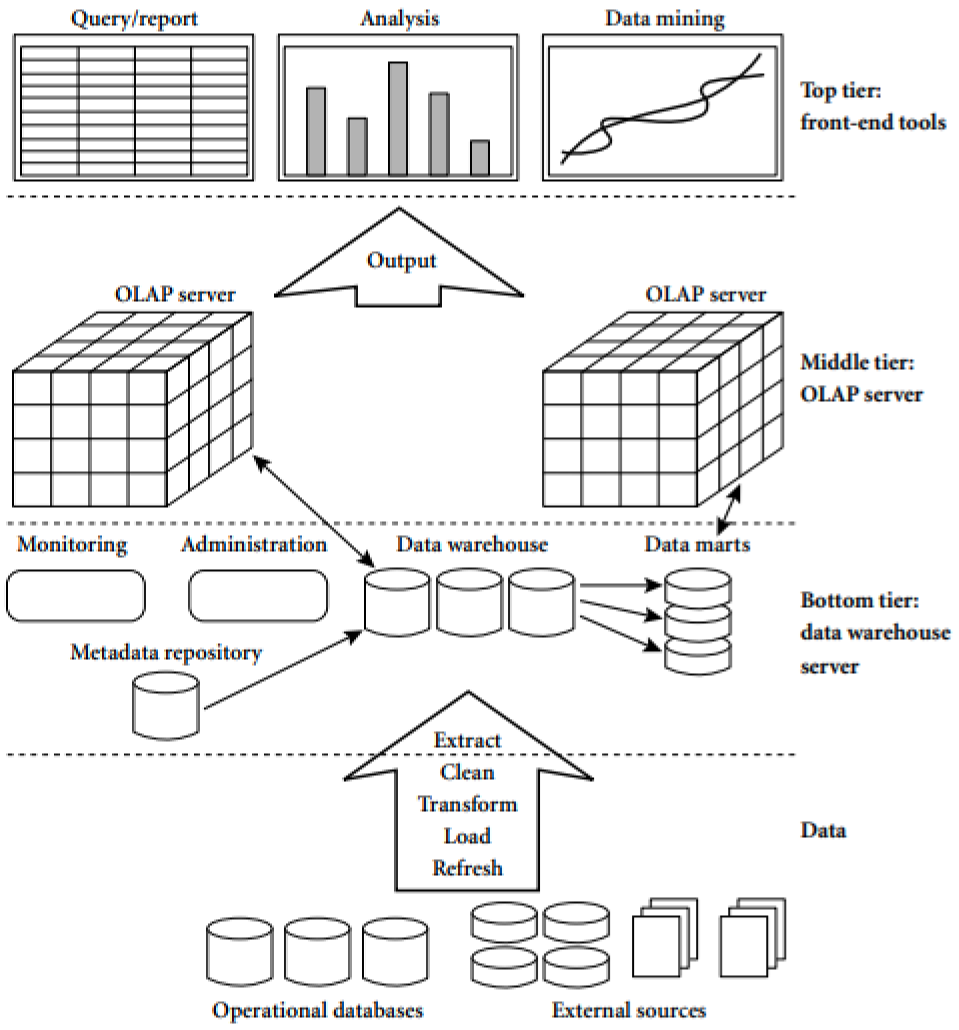


Fig 1.10: Data Warehouse Architecture

Data Warehouse Models

Data warehouses can be modeled in three main ways: enterprise warehouses, data marts, and virtual warehouses.

- **Enterprise Warehouse**: This model gathers all relevant information across the entire organization, providing corporate-wide data integration from multiple operational systems or external sources. It is cross-functional and may contain detailed and summarized data, ranging from a few gigabytes to terabytes or more. Enterprise warehouses require extensive business modeling and can take years to design and build.
- **Data Mart**: A data mart is a subset of a corporate-wide data warehouse, focusing on specific subjects for a particular group of users, such as a marketing data mart that focuses on customers, items, and sales. Data marts are typically implemented on lower-cost departmental servers and can be developed much faster than enterprise warehouses, often within weeks. However, integrating multiple data marts into a consistent enterprise

warehouse can be challenging if not planned properly. Data marts can be independent, sourced from operational systems or external providers, or dependent, sourced directly from enterprise data warehouses.

- **Virtual Warehouse**: This model consists of views over operational databases, where only some summary views may be materialized for efficient query processing. Virtual warehouses are easier to build but require additional capacity on operational database servers.

Data Warehouse Back-End Tools and Utilities

Data warehouse systems utilize back-end tools and utilities to populate and refresh their data. These include:

- **Data Extraction**: Collects data from multiple, heterogeneous, and external sources.
- **Data Cleaning**: Identifies and corrects errors in the data.
- **Data Transformation**: Converts data from its original format to the warehouse format.
- **Loading**: Involves sorting, summarizing, consolidating, computing views, checking integrity, and building indices and partitions.
- **Refreshing**: Propagates updates from data sources to the warehouse.

These tools play a vital role in maintaining the quality of the data within the warehouse, which in turn affects the accuracy and reliability of data analysis and mining.

Metadata Repository

Metadata is essentially data about data. In the context of a data warehouse, metadata defines the objects within the warehouse. They include descriptions of the warehouse's structure (e.g., schema, views, dimensions, hierarchies), operational metadata (e.g., data lineage, currency, usage statistics), algorithms for summarization, and mappings from operational systems to the data warehouse. Metadata is critical for guiding various processes within the data warehouse, such as locating data, mapping data during transformation, and summarizing data at different levels. It is essential that metadata are stored and managed persistently, typically on disk, to ensure they are accessible and reliable.

By utilizing this architecture and the accompanying tools and repositories, a data warehouse can effectively support complex queries and analytical tasks, providing valuable insights for decision-making across an organization.

Indexing

To enhance data retrieval efficiency, many data warehouse systems utilize indexing structures and materialized views (via cuboids). While previous sections discussed general approaches for selecting cuboids for materialization, this section explores indexing OLAP data through bitmap indexing and join indexing.

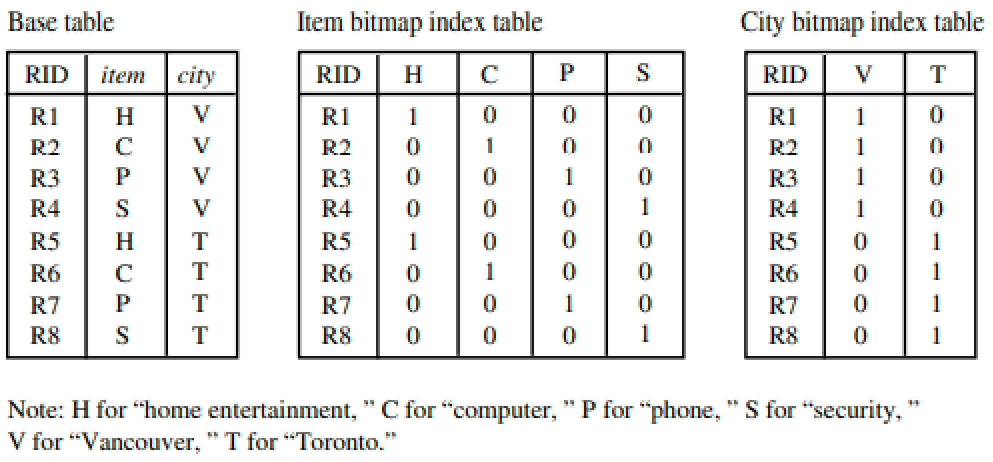
Bitmap Indexing

Bitmap indexing is a favored technique in OLAP systems due to its ability to quickly search through data cubes. Instead of listing record IDs (RIDs), bitmap indexing uses a bit vector for each attribute value. For an attribute with ‘n’ possible values, the bitmap index comprises n bit vectors. Each bit vector corresponds to one value in the attribute’s domain. In each bit vector, a bit is set to 1 if the attribute has that value for a specific row, while all other bits are set to 0.

Example 1: Bitmap Indexing in Action

Consider the AllElectronics data warehouse, where the top-level dimension item has four categories: “home entertainment,” “computer,” “phone,” and “security.” For a data cube with 100,000 rows, each of these four values is represented by a bit vector. Thus, the bitmap index table for the item dimension contains four bit vectors, each with 100,000 bits. Bitmap indexing is particularly effective for domains with low cardinality because it allows for fast comparisons, joins, and aggregations through bitwise operations, significantly reducing processing time. For higher-cardinality domains, bitmap indexing can be optimized using compression techniques.

Table 5: BitMap Indexing in action



Base table

RID	*item*	*city*
R1	H	V
R2	C	V
R3	P	V
R4	S	V
R5	H	T
R6	C	T
R7	P	T
R8	S	T

Item bitmap index table

RID	H	C	P	S
R1	1	0	0	0
R2	0	1	0	0
R3	0	0	1	0
R4	0	0	0	1
R5	1	0	0	0
R6	0	1	0	0
R7	0	0	1	0
R8	0	0	0	1

City bitmap index table

RID	V	T
R1	1	0
R2	1	0
R3	1	0
R4	1	0
R5	0	1
R6	0	1
R7	0	1
R8	0	1

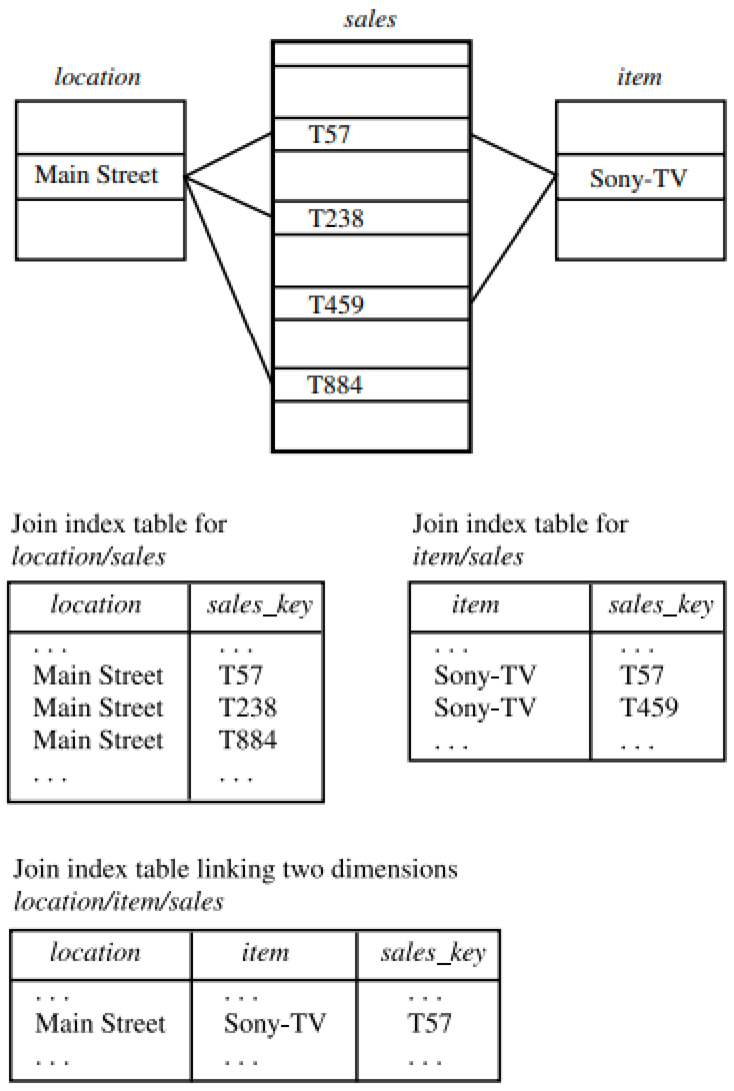
Note: H for “home entertainment, ” C for “computer, ” P for “phone, ” S for “security, ”
V for “Vancouver, ” T for “Toronto.”

Join Indexing

Join indexing, widely used in relational database systems, differs from traditional indexing by focusing on joinable rows between two relations. Instead of mapping column values to row lists, join indexing creates records that indicate which rows from two relations can be joined. For instance, if two relations R and S are joined on attributes A and B, the join index maintains records of joinable tuples, such as pairs (RID,SID, where RID and SID are record identifiers from relations R and S, respectively. This approach eliminates the need for costly join operations and is particularly useful for managing foreign key relationships and their corresponding primary keys.

Example 2: Join Indexing

In the AllElectronics star schema, which includes dimensions like time, item, branch, and location, join indexing simplifies cross-table searches. For example, the fact table in the schema is joined with dimension tables for location and item. If the location dimension includes "Main Street" and the item dimension includes "Sony-TV," the join index helps efficiently link these values to relevant tuples in the sales fact table. With a large number of sales tuples and only a subset of items participating in the joins, join indices help minimize additional I/O operations required for bringing together fact and dimension tables as in figure 1.11.



Join index table for
location/sales

location	*sales_key*
. . .	. . .
Main Street	T57
Main Street	T238
Main Street	T884
. . .	. . .

Join index table for
item/sales

item	*sales_key*
. . .	. . .
Sony-TV	T57
Sony-TV	T459
. . .	. . .

Join index table linking two dimensions
location/item/sales

location	*item*	*sales_key*
. . .	. . .	. . .
Main Street	Sony-TV	T57
. . .	. . .	. . .

Fig 1.11: Join Indexing

Bitmapped Join Indices

To further optimize query performance, bitmap indexing and join indexing can be combined into bitmapped join indices. This integration leverages the benefits of both methods, providing an efficient way to access and analyze OLAP data.

This approach to indexing helps streamline data retrieval and query processing in data warehouses, making it easier to manage and analyze large volumes of data.

Efficient OLAP Query Processing

The goal of materializing cuboids and building OLAP index structures is to enhance query processing speed within data cubes. When working with materialized views, the query processing should follow these steps:

1. **Identify Required Operations:** Determine the operations that need to be performed on the available cuboids by translating any selection, projection, roll-up (group-by), or drill-down operations specified in the query into corresponding SQL or OLAP operations. For instance, slicing and dicing a data cube might correspond to selection and/or projection operations on a materialized cuboid.
2. **Select Relevant Cuboids:** Identify which materialized cuboids could potentially be used to answer the query. Prune this set based on "dominance" relationships among the cuboids, estimate the costs of using the remaining cuboids, and choose the cuboid with the lowest cost.

Example: Consider a data cube for AllElectronics with the structure sales cube[time, item, location]: sum(sales in dollars], and dimension hierarchies for time (day < month < quarter < year), item (item name < brand < type), and location (street < city < province or state < country).

For a query involving {brand, province or state} with the condition year = 2004, and given four materialized cuboids:

- Cuboid 1: {year, item name, city}
- Cuboid 2: {year, brand, country}
- Cuboid 3: {year, brand, province or state}
- Cuboid 4: {item name, province or state} (with year = 2004)

Cuboid 2 cannot be used because it includes country, which is a more general concept than province or state. Cuboids 1, 3, and 4 can be used since they include the necessary dimensions and meet the selection criteria.

To compare costs, using Cuboid 1 may be the most expensive due to its finer granularity compared to the query's requirements. Cuboid 3 is likely more efficient than Cuboid 4 if there are many items per brand but a few years, but if Cuboid 4 has efficient indexing, it might be a better choice. Thus, cost-based estimation is needed to select the optimal cuboid for the query.

MOLAP Storage and Processing: MOLAP servers use an n-dimensional array for storage, offering direct addressing capabilities. However, sparse data can lead to poor storage utilization. To improve efficiency, sparse matrix and data compression techniques should be employed. A two-level approach to MOLAP query processing can be used: dense arrays are indexed using B-trees, while sparse matrices are managed separately. This approach improves storage efficiency while maintaining direct addressing capabilities.

Additional Strategies for Quick Query Responses:

- **On-line Aggregation:** Provides intermediate results during query processing, giving users partial answers and feedback as the computation progresses. This increases interactivity and helps users adjust their queries without waiting for complete results.

- **Top N Queries:** Focuses on retrieving only the top N items (e.g., best-selling products) rather than the full sorted list, optimizing query performance and resource usage while enhancing user interaction.

UNIT EXERCISE

2 Marks

1. Define a data warehouse.
2. What is a Multidimensional data model?
3. Mention the key characteristics of the Operational database system.
4. How do you perform efficient OLAP Query processing?
5. Give a few additional strategies for Quick Query Responses.
6. Define On-line aggregation.
7. What is a virtual warehouse?
8. Which components are significant in the top tier of a data warehouse architecture?
9. What are the four key perspectives that must be considered when designing a data warehouse?

8 Marks

1. Elaborate on the differences between OLAP and OLTP.
2. Explain the data warehouse architecture in detail.
3. What is Join Indexing? Explain with an example.

Unit_02 - Knowledge Discovery in Databases

Knowledge Discovery in Databases (KDD) is a comprehensive process of discovering useful knowledge from large volumes of data.

KDD is essential for leveraging the vast amounts of data available today to gain valuable insights, make informed decisions, and create strategic advantages. Its applications span various industries and domains, helping organizations to improve operations, understand their customers, and stay competitive in a data-driven world.

Why Do We Need KDD?

1. **Data Explosion**: In today's digital age, organizations and individuals generate vast amounts of data daily. This data can be overwhelming and often contains valuable insights that are not immediately apparent. KDD helps in managing and extracting meaningful information from this large volume of data.
2. **Decision Making**: Businesses and organizations rely on data to make informed decisions. KDD transforms raw data into actionable insights that can guide strategic decisions, improve operations, and drive competitive advantage.
3. **Competitive Advantage**: In a competitive market, having access to actionable insights can provide a significant edge over competitors. KDD enables organizations to identify trends, customer preferences, and market opportunities that may not be visible through traditional analysis.
4. **Improving Efficiency**: By discovering patterns and anomalies, KDD can help in streamlining processes, optimizing resource allocation, and improving overall efficiency in various domains.
5. **Predictive Analysis**: KDD techniques allow organizations to anticipate future trends and behaviors, making it possible to proactively address potential issues or capitalize on emerging opportunities.

Real-Life Applications of KDD

1. **Healthcare**:
   - **Disease Prediction and Diagnosis**: KDD helps in analyzing patient data to predict diseases, identify risk factors, and improve diagnostic accuracy.
   - **Personalized Medicine**: By analyzing patient history and genetic data, KDD can contribute to personalized treatment plans.
2. **Finance**:
   - **Fraud Detection**: Financial institutions use KDD to identify unusual patterns and behaviors that may indicate fraudulent activities.
   - **Credit Scoring**: KDD helps in assessing the creditworthiness of individuals or businesses by analyzing financial history and other relevant data.
3. **Retail**:

    - **Customer Segmentation**: Retailers use KDD to segment customers based on purchasing behavior, preferences, and demographics to tailor marketing strategies.
    - **Recommendation Systems**: E-commerce platforms use KDD to recommend products based on past purchases and browsing behavior.
4. **Manufacturing**:
    - **Predictive Maintenance**: KDD can predict equipment failures or maintenance needs by analyzing sensor data from machinery.
    - **Quality Control**: Analyzing production data helps in identifying defects and improving product quality.
5. **Telecommunications**:
    - **Churn Analysis**: KDD helps telecom companies understand why customers leave and develop strategies to improve customer retention.
    - **Network Optimization**: Analyzing network traffic data aids in optimizing performance and managing resources.
6. **Transportation and Logistics**:
    - **Route Optimization**: KDD can optimize delivery routes based on traffic patterns and historical data, reducing costs and improving efficiency.
    - **Demand Forecasting**: Analyzing historical data helps in predicting demand and adjusting logistics accordingly.

Ultimate Goal of KDD

The ultimate goal of KDD is to **transform raw data into actionable knowledge** that can drive decision-making and strategy. This involves several key objectives:

1. **Insight Generation**: Discovering patterns, correlations, and trends in the data that provide new insights or validate existing hypotheses.
2. **Decision Support**: Providing decision-makers with relevant, accurate, and timely information to support strategic and operational decisions.
3. **Knowledge Creation**: Building new knowledge that can be used to solve problems, innovate, or improve processes and outcomes.
4. **Problem Solving**: Using discovered knowledge to address specific business or research problems, leading to solutions that enhance efficiency, effectiveness, and overall performance.

Introduction to KDD Process

1. KDD Process Steps

1. **Data Selection**
    - **Description**: Identify and retrieve relevant data from different sources. This step involves determining which data sources are pertinent to the analysis.
    - **Activities**: Data extraction, data collection, and data integration from multiple databases or data warehouses.
2. **Data Cleaning**

   - **Description**: Address issues like missing values, noise, and inconsistencies in the data. This step ensures that the data is accurate and reliable.
   - **Activities**: Data imputation, outlier detection, and correction of errors.
3. **Data Transformation**
   - **Description**: Convert data into a suitable format or structure for analysis. This often involves normalization, aggregation, and generalization.
   - **Activities**: Data normalization, feature extraction, and dimensionality reduction.
4. **Data Mining**
   - **Description**: Apply algorithms and techniques to uncover patterns or relationships in the data. This is where the actual extraction of patterns, trends, or knowledge occurs.
   - **Activities**: Classification, clustering, association rule mining, regression analysis, and anomaly detection.
5. **Pattern Evaluation**
   - **Description**: Assess the discovered patterns to determine their usefulness and validity. This involves evaluating the quality and relevance of the patterns.
   - **Activities**: Validating patterns against predefined criteria, measuring accuracy, and interpreting the results.
6. **Knowledge Presentation**
   - **Description**: Present the discovered knowledge in an understandable and actionable format. This step aims to communicate the results effectively to stakeholders.
   - **Activities**: Visualization of results, generation of reports, and creating summaries.
7. **Knowledge Utilization**
   - **Description**: Apply the discovered knowledge to decision-making processes, business strategies, or further research.
   - **Activities**: Implementing insights into business operations, developing new strategies, or refining models.

3. Tools and Techniques

- **Data Mining Tools**: Software like RapidMiner, Weka, and KNIME.
- **Algorithms**: Decision trees, neural networks, clustering algorithms, etc.
- **Visualization Tools**: Tableau, Power BI, or custom visualization libraries.

4. Challenges in KDD

- **Data Quality**: Ensuring data is accurate and reliable.
- **Scalability**: Managing and analyzing large datasets efficiently.
- **Interpretability**: Making the results understandable and actionable for non-technical stakeholders.
- **Privacy**: Ensuring that data analysis respects privacy and security concerns.

By following the KDD process, organizations can extract valuable insights from their data, driving informed decision-making and strategic planning.

Why is Data Preprocessing Important?

As the manager at AllElectronics tasked with analyzing branch sales data, you start by examining the company's database and data warehouse. You select key attributes like item, price, and units sold for your analysis. However, you find several issues: some attributes are missing, such as whether items were on sale, and you encounter errors, unusual values, and inconsistencies in the data.

In the real world, databases and data warehouses often contain incomplete, noisy, and inconsistent data. Missing data can result from various issues, like unrecorded attributes or errors during data entry. Noisy data might be due to faulty collection instruments, human or computer errors, or data transmission issues. Inconsistent data may arise from different naming conventions or formats. Additionally, duplicate records need to be addressed.

Data cleaning is crucial for preparing the data for analysis. It involves filling in missing values, smoothing out noisy data, identifying and removing outliers, and resolving inconsistencies. Without clean data, the results of data mining might be unreliable. Although mining algorithms have some mechanisms to handle dirty data, they often focus more on avoiding overfitting rather than thorough data cleaning. Thus, preprocessing steps like data cleaning are essential.

When integrating data from multiple sources, you might face challenges such as inconsistent attribute names or redundant data. For example, customer IDs might be labeled differently across databases. Data cleaning and integration help resolve these issues and avoid redundancies.

If you plan to use distance-based algorithms like neural networks or clustering, normalizing the data is important. Normalization scales attributes to a standard range, which ensures that no single attribute dominates the distance measurements. Additionally, data transformation techniques, like normalization and aggregation, help improve the mining process.

Given the large volume of your data, data reduction techniques can help manage it more effectively without compromising the analysis results. Strategies for data reduction include data aggregation, attribute subset selection, dimensionality reduction, and numerosity reduction. Generalization through concept hierarchies, where specific concepts are replaced with broader ones, is another way to reduce data complexity.

Data Cleaning

Real-world data is often incomplete, noisy, and inconsistent. Data cleaning, also known as data cleansing, involves routines that attempt to fill in missing values, smooth out noise, identify outliers, and correct inconsistencies in the data. This section explores basic data cleaning methods.

Handling Missing Values

Imagine analyzing sales and customer data from AllElectronics, where many records lack values for attributes like customer income. How can missing values be handled? Here are some common methods:

1. **Ignore the Tuple**: This approach is often used when the class label is missing, particularly in classification tasks. However, it's not effective if many attributes have missing values, and it performs poorly when the missing data percentage varies widely between attributes.

2. **Fill in Manually**: This method is labor-intensive and not feasible for large datasets with numerous missing values.

3. **Use a Global Constant**: Replace missing values with a constant, such as "Unknown" or $-\infty$. However, this can create biases, as the replacement value might be seen as meaningful.

4. **Use the Attribute Mean**: Replace missing values with the mean value of the attribute, such as using an average customer income of $56,000 for missing values.

5. **Use Class-Specific Means**: For classification tasks, use the mean value of the attribute for the same class as the missing value, such as using the average income for customers with similar credit risk.

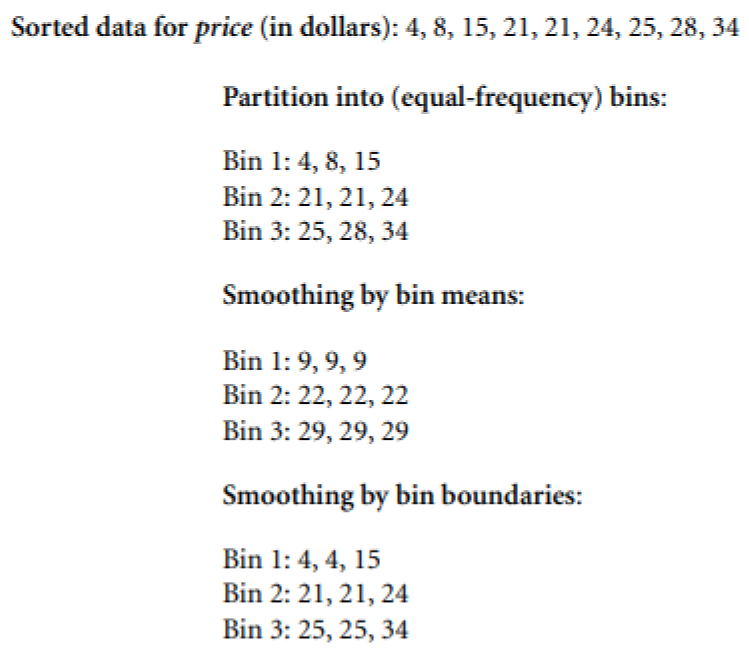
6. **Use the Most Probable Value**: Employ statistical models like regression, Bayesian inference, or decision tree induction to predict missing values based on other attributes. This approach utilizes more data from the dataset, preserving relationships between attributes better than the other methods.

While methods 3 to 6 can introduce biases, using the most probable value (method 6) is often preferred because it leverages the data to make more informed predictions.

Smoothing Noisy Data

Noise refers to random errors or variances in data. Here are some methods to smooth out noisy data:

1. **Binning**: This technique smooths data by grouping sorted values into bins and replacing values with bin means, medians, or boundaries. For example, a sorted list of prices is divided into bins of equal frequency, and each value within a bin is replaced by the bin's mean. It is shown in figure 2.1.



Sorted data for ***price*** **(in dollars)**: 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equal-frequency) bins:

Bin 1: 4, 8, 15
Bin 2: 21, 21, 24
Bin 3: 25, 28, 34

Smoothing by bin means:

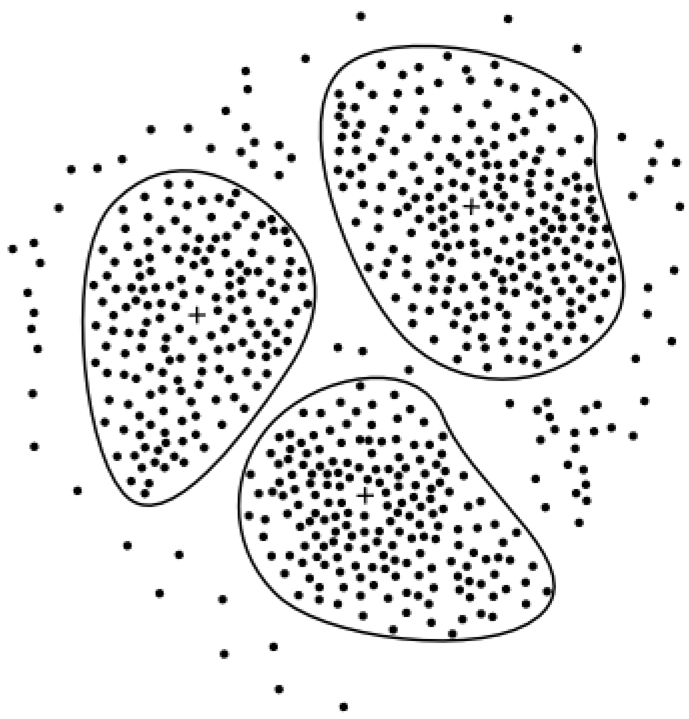
Bin 1: 9, 9, 9
Bin 2: 22, 22, 22
Bin 3: 29, 29, 29

Smoothing by bin boundaries:

Bin 1: 4, 4, 15
Bin 2: 21, 21, 24
Bin 3: 25, 25, 34

Fig 2.1: Illustration of Binning

2. **Regression**: Data can be smoothed by fitting it to a function, such as a regression line. Linear regression finds the best fit line between two variables, while multiple regression extends this to multiple variables, fitting data to a multidimensional surface.



A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster centroid is marked with a "+", representing the average point in space for that cluster. Outliers may be detected as values that fall outside of the sets of clusters.

Fig 2.2: Clustering

3. **Clustering**: Similar values are grouped into clusters, and values that fall outside these clusters can be identified as outliers. Clustering is further explored in Unit 4 and shown in figure 2.2.

Some smoothing techniques also reduce data by discretizing it, which is useful in data mining methods like decision tree induction.

Data Cleaning as a Process

Data cleaning involves detecting discrepancies, which can be due to errors in data entry, inconsistent data representations, system errors, or data integration issues. Here's how to approach the data cleaning process:

1. **Discrepancy Detection**: Use existing knowledge or metadata to identify data properties, such as acceptable value ranges and known attribute dependencies. Descriptive data summaries, like mean and standard deviation, help identify anomalies.
2. **Error and Inconsistency Management**: Look for inconsistent codes, date formats, and rules violations. Errors may include typos, missing values, or null conditions like "don't

know" or "N/A." Software tools like data scrubbing and data auditing can help detect and correct these issues.

3. **Data Transformation**: Once discrepancies are found, transformations must be defined and applied to correct them. Tools like data migration and ETL (extraction, transformation, loading) software allow users to specify transformations via graphical interfaces, though custom scripts are often needed.

This iterative process of discrepancy detection and transformation is time-consuming and prone to errors. Modern data cleaning tools like Potter's Wheel emphasize interactivity, allowing users to build transformations step-by-step, with immediate feedback. Declarative languages for data transformation and interactive tools help streamline the data cleaning process, making it more efficient and user-friendly.

Continuous updates to metadata during data cleaning can enhance future data cleaning efforts, making the process more effective.

Data Integration

It is likely that your data analysis task will involve data integration, which combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

There are a number of issues to consider during data integration. Schema integration and object matching can be tricky. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the entity identification problem. For example, how can the data analyst or the computer be sure that customer id in one database and cust number in another refer to the same attribute? Examples of metadata for each attribute include the name, meaning, data type, and range of values permitted for the attribute, and null rules for handling blank, zero, or null values. Such metadata can be used to help avoid errors in schema integration. The metadata may also be used to help transform the data (e.g., where data codes for pay type in one database may be "H" and "S", and 1 and 2 in another). Hence, this step also relates to data cleaning, as described earlier.

Redundancy is another important issue. An attribute (such as annual revenue, for instance) may be redundant if it can be "derived" from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Some redundancies can be detected by correlation analysis. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For numerical attributes, we can evaluate the correlation between two attributes, A and B, by computing the correlation coefficient (also known as Pearson's product moment coefficient, named after its inventor, Karl Pearson) as given in *Equation 2.1*.

$$r_{A,B} = \frac{\sum_{i=1}^{N}(a_i - \bar{A})(b_i - \bar{B})}{N\sigma_A\sigma_B} = \frac{\sum_{i=1}^{N}(a_ib_i) - N\bar{A}\bar{B}}{N\sigma_A\sigma_B}, \qquad \textbf{(2.1)}$$

where N is the number of tuples, ai and bi are the respective values of A and B in tuple i, A’ and B’ are the respective mean values of A and B, σA and σB are the respective standard deviations of A and B, and Σ(ai*bi) is the sum of the AB cross-product (that is, for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \leq r(A,B) \leq 1$. If r(A,B) is greater than 0, then A and B are positively correlated, meaning that the values of A increase as the values of B increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that A (or B) may be removed as a redundancy. If the resulting value is equal to 0, then A and B are independent and there is no correlation between them. If the resulting value is less than 0, then A and B are negatively correlated, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes.

Note that correlation does not imply causality. That is, if A and B are correlated, this does not necessarily imply that A causes B or that B causes A. For example, in analyzing a demographic database, we may find that attributes representing the number of hospitals and the number of car thefts in a region are correlated. This does not mean that one causes the other. Both are actually causally linked to a third attribute, namely, population.

For categorical (discrete) data, a correlation relationship between two attributes, A and B, can be discovered by a χ2 (chi-square) test. Suppose A has c distinct values, namely a1,a2,...,ac. B has r distinct values, namely b1,b2,...,br. The data tuples described by A and B can be shown as a contingency table, with the c values of A making up the columns and the r values of B making up the rows. Let (Ai,Bj) denote the event that attribute A takes on value ai and attribute B takes on value bj, that is, where (A=ai,B=bj) given in *Equation 2.2*. Each and every possible (Ai,Bj) joint event has its own cell (or slot) in the table.

$$\chi^2 = \sum_{i=1}^{c}\sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{2.2}$$

where $o_{i\,j}$ is the observed frequency (i.e., actual count) of the joint event (A_i, B_j) and e_{ij} is the expected frequency of (A_i, B_j), which can be computed as given in *Equation 2.3*

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{N}, \tag{2.3}$$

where N is the number of data tuples, $count(A = a_i)$is the number of tuples having value a_i for A, and $count(B = b_j)$ is the number of tuples having value bj for B. The sum in Equation (2.9) is computed over all of the r ×c cells. Note that the cells that contribute the most to the χ 2 value are those whose actual count is very different from that expected.

Data Transformation

Data transformation involves modifying or consolidating data into suitable forms for mining. This process can include several techniques:

1. **Smoothing:** This technique removes noise from the data using methods such as binning, regression, and clustering.
2. **Aggregation:** In this step, data is summarized or aggregated, like combining daily sales data to calculate monthly or yearly totals. Aggregation is often used to build data cubes, allowing analysis at various levels of detail.
3. **Generalization:** Here, raw data is replaced by higher-level concepts using concept hierarchies. For instance, a categorical attribute like "street" can be generalized to "city" or "country," and numerical values like "age" can be grouped into broader categories such as "youth," "middle-aged," and "senior."
4. **Normalization:** This process scales attribute data to fit within a specified range, such as 0.0 to 1.0 or -1.0 to 1.0. Normalization is especially beneficial for classification algorithms involving neural networks and distance-based methods like nearest-neighbor classification and clustering. For example, in neural network classification, normalizing input values speeds up the learning phase. In distance-based methods, normalization ensures that attributes with larger initial ranges, such as income, do not dominate those with smaller ranges, like binary attributes. Common normalization techniques include min-max normalization, z-score normalization, and decimal scaling.
5. **Attribute Construction (or Feature Construction):** New attributes are created and added from the existing ones to improve the mining process. The max-min normalization equation is given in *Equation 2.4*

Min-max normalization performs a linear transformation on the original data. Sup- pose that min_A and max_A are the minimum and maximum values of an attribute, A. Min-max normalization maps a value, v, of A to v' in the range [new_min$_A$, new max$_A$ by computing

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A. \qquad \textbf{(2.4)}$$

Min-max normalization preserves the relationships among the original data values. It will encounter an “out-of-bounds” error if a future input case for normalization falls outside of the original data range for A.

Data Reduction

When analyzing data from the AllElectronics data warehouse, you often deal with extremely large datasets. Performing complex data analysis and mining on such massive amounts of data can be time-consuming and impractical.

To address this, data reduction techniques can be used to create a smaller, more manageable representation of the dataset that retains the essential characteristics of the original data. The goal is to make the analysis more efficient while still achieving comparable results.

The main strategies for data reduction include:

1. **Data Cube Aggregation**: Aggregation operations are applied to construct a data cube, summarizing the data.
2. **Attribute Subset Selection**: Irrelevant, weakly relevant, or redundant attributes are identified and removed to reduce the dataset.
3. **Dimensionality Reduction**: Techniques like encoding mechanisms are used to reduce the size of the dataset by transforming it into a lower-dimensional space.
4. **Discretization and Concept Hierarchy Generation**: Raw data values are replaced with ranges or higher conceptual levels, allowing data to be analyzed at multiple levels of abstraction. This approach is particularly useful for generating concept hierarchies automatically.

While these techniques reduce the data volume, it's crucial that the computational time spent on data reduction does not negate the benefits of having a smaller dataset for analysis.

1. Data cube aggregation

This method is used to condense data into a more manageable format. Data Cube Aggregation is a multidimensional aggregation that represents the original data set by aggregating at multiple layers of a data cube, resulting in data reduction as shown in figure 2.3.

Consider the following scenario: you have quarterly sales data for All Electronics from 2018 to 2022. To calculate the yearly sale for each year, just add up the quarterly sales for each year. Aggregation gives you the needed data, which is considerably smaller in size, and we achieve data reduction without losing any data in this method.

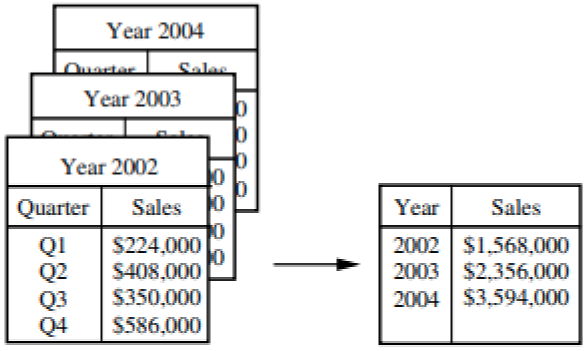


Fig 2.3: Illustration of Data Cube

This has been aggregated to as shown in figure 4.

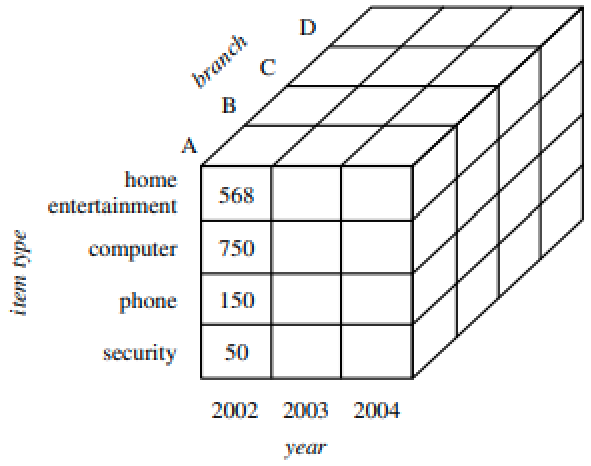


Fig 2.4: Illustration of Aggregation Cube

2. Attribute Subset Selection

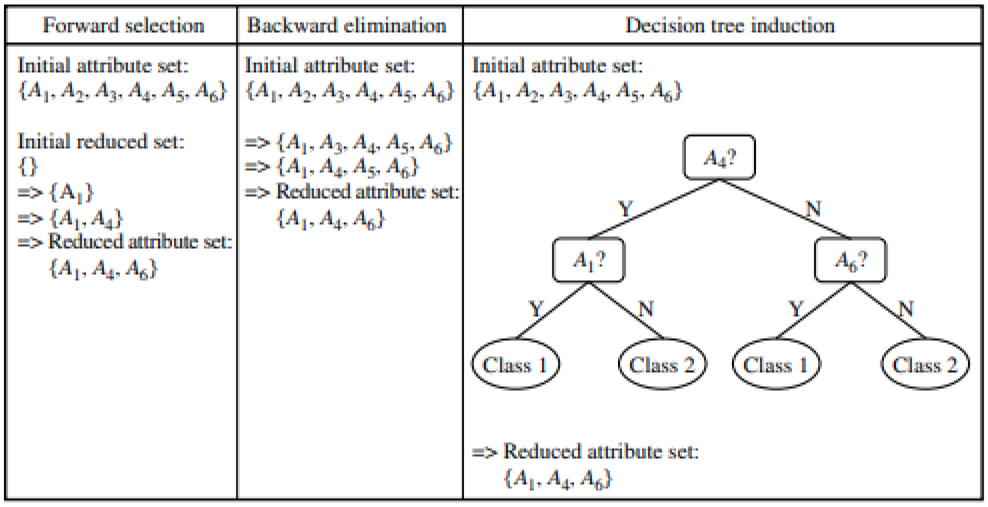
In data analysis, datasets often contain numerous attributes, some of which may not contribute meaningfully to the task at hand, or might be redundant. For instance, in a scenario where we want to predict whether a customer will purchase a new product based on their past behavior, attributes like the customer's postal code might be less relevant compared to attributes like purchase history or customer preferences.

Identifying the most relevant attributes manually can be challenging and time-consuming, especially when the nature of the data is not well-understood. Including irrelevant attributes or omitting important ones can negatively impact the performance of data mining algorithms, leading to poorer quality patterns and slowing down the mining process due to increased computational load.

Attribute subset selection aims to simplify the dataset by removing unnecessary or redundant attributes, thus focusing only on those that are crucial for the task. The objective is to identify a minimal set of attributes that best preserves the original data's distribution and class probabilities. By working with a reduced set of attributes, the data mining process becomes more efficient and the resulting patterns are generally easier to interpret.

Given that there are pow(2, N^2) possible subsets of attributes for n attributes, a complete search for the best subset is often impractical, especially as n increases. Therefore, heuristic methods are commonly employed to navigate the search space more efficiently. These methods typically use a greedy approach, making locally optimal choices with the aim of finding a globally optimal solution. While these methods may not always guarantee the best possible subset, they are practical and can produce very effective results.

To evaluate which attributes are most significant, statistical tests are often used. These tests assume that attributes are independent, and help in determining which attributes contribute the most to the predictive power of the model as shown in figure 5.



Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: {} => $\{A_1\}$ => $\{A_1, A_4\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$

Fig 2.5: Attribute Subset Selection

Basic heuristic methods of attribute subset selection include the following techniques

1. **Stepwise Forward Selection**: This method begins with an empty set of attributes and identifies the best attribute from the original set to include in the reduced set. In each subsequent step, it adds the next best attribute from the remaining original attributes.
2. **Stepwise Backward Elimination**: Starting with the full set of attributes, this technique iteratively removes the least useful attribute at each step until the optimal subset is found.
3. **Combination of Forward Selection and Backward Elimination**: This approach merges forward selection and backward elimination methods. At each step, it selects the best attribute and simultaneously removes the least useful attribute from the remaining set.
4. **Decision Tree Induction**: Originally designed for classification, decision tree algorithms like ID3, C4.5, and CART build a flowchart-like structure. Each internal node represents a test on an attribute, each branch denotes the outcome of the test, and each leaf node indicates a class prediction. When used for attribute subset selection, attributes that do not appear in the decision tree are considered irrelevant. The attributes that are included in the tree make up the reduced subset.

The stopping criteria for these methods can vary, often involving a threshold or measure to determine when to cease the attribute selection process.

3. Dimensionality Reduction

We employ the characteristic necessary for our study if we come across data that is just marginally essential. Dimensionality reduction removes characteristics from the data set in question, resulting in a reduction in the size of the original data. It shrinks data by removing obsolete or superfluous characteristics. Here are three approaches for reducing dimensionality.

Wavelet Transform: Assume that a data vector A is transformed into a numerically different data vector A' such that both A and A' vectors are of the same length using the wavelet transform. Then, because the data received from the wavelet transform may be abbreviated, how is it beneficial in data reduction? By keeping the smallest piece of the strongest wavelet coefficients, compressed data can be produced. Data cubes, sparse data, and skewed data can all benefit from the Wavelet transform.

1. Wavelet transforms are mathematical tools for analyzing data where features vary over different scales. For signals, features can be frequencies varying over time, transients, or slowly varying trends. For images, features include edges and textures. Wavelet transforms were primarily created to address limitations of the Fourier transform.
2. While Fourier analysis consists of decomposing a signal into sine waves of specific frequencies, wavelet analysis is based on decomposing signals into shifted and scaled versions of a *wavelet*. A wavelet, unlike a sine wave, is a rapidly decaying, wave-like oscillation. This enables wavelets to represent data across multiple scales. Different wavelets can be used depending on the application.
3. Audio signals, time-series financial data, and biomedical signals typically exhibit piecewise smooth behavior punctuated by transients. Similarly, images typically include homogenous, piecewise smooth regions separated by transients, which appear as edges. For both signals and images, the smooth regions and transients can be sparsely represented with wavelet transforms.
4. Wavelet transforms can be classified into two broad classes:
   4.1 the continuous wavelet transform (CWT) and
   4.2 the discrete wavelet transform (DWT).
      i. The continuous wavelet transform is a time-frequency transform, which is ideal for analysis of non-stationary signals. A signal being nonstationary means that its frequency-domain representation changes over time. CWT is similar to the short-time Fourier transform (STFT)(Refer Fig 6). The STFT uses a fixed window to create a local frequency analysis, while CWT tiles the time-frequency plane with variable-sized windows. The window widens in time, making it suitable for low-frequency phenomena, and narrows for high-frequency phenomena. The continuous wavelet transform can be used to analyze transient behavior, rapidly changing frequencies, and slowly varying behavior.

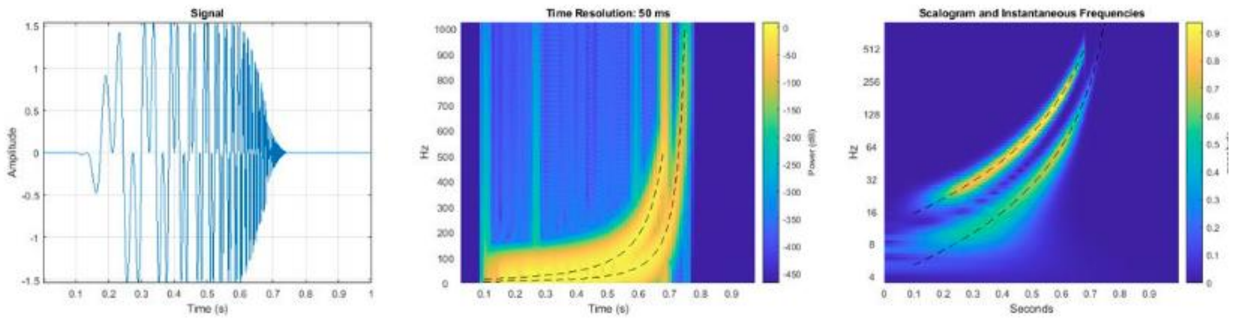


Fig 2.6: Illustration of Wavelet Transform

      ii. With the discrete wavelet transform scales are discretized more coarsely than with CWT. This makes DWT useful for compressing and denoising signals and images while preserving important features as in Fig 2.7. You can use discrete wavelet transforms to perform multiresolution analysis and split signals into physically meaningful and interpretable components.

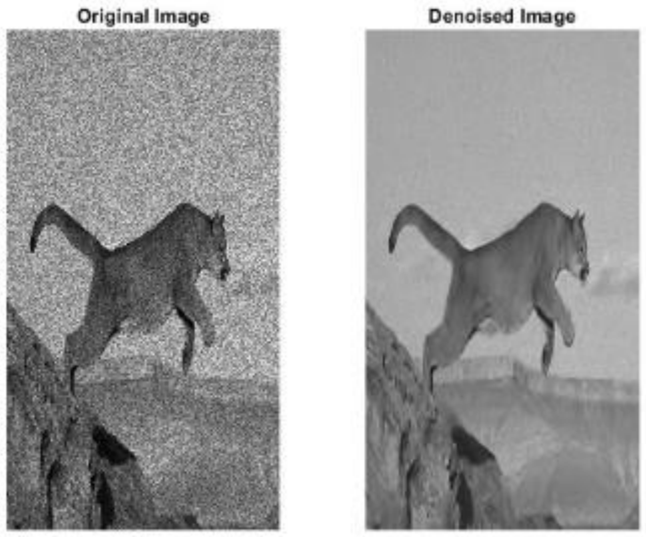


Fig 2.7: Illustration of DWT

5. **Principal Component Analysis:** Assume we have a data set with n properties that needs to be evaluated. The main component analysis finds k distinct tuples with n properties that may be used to describe the data collection. The original data may be cast on a considerably smaller space in this fashion, resulting in dimensionality reduction. Principal component analysis can be used on data that is sparse or skewed.

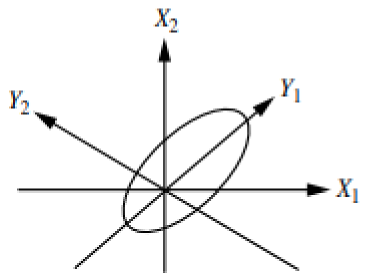
This section provides a basic overview of Principal Component Analysis (PCA) as a method for reducing dimensionality. A more in-depth theoretical discussion is beyond the scope of this text.

Imagine we have data described by nnn attributes or dimensions. PCA, also known as the Karhunen-Loeve (K-L) method, seeks to find k orthogonal vectors in this n-dimensional space, where k≤n, that best represent the data. This process effectively projects the original data onto a smaller dimensional space, thereby reducing its complexity. Unlike attribute subset selection, which reduces the number of attributes by selecting a subset from the original set, PCA combines the essence of the attributes into a new, smaller set of variables. This transformation often uncovers relationships that were previously unnoticed, leading to new interpretations.

The PCA procedure involves the following steps:

1. **Normalization**: The data are standardized so that each attribute has a similar range. This step prevents attributes with larger ranges from overshadowing those with smaller ranges.
2. **Computation of Principal Components**: PCA identifies kkk orthonormal vectors that form a basis for the normalized data. These unit vectors, known as principal components, are perpendicular to each other. The data can then be expressed as a linear combination of these principal components.

3. **Sorting Principal Components**: The principal components are ranked according to their "significance" or strength, where the first component accounts for the most variance, the second for the next most, and so on.This sorting helps reveal patterns or clusters within the data.
4. **Dimensionality Reduction**: By retaining only the most significant principal components and discarding the less significant ones (those with lower variance), the data's dimensionality is reduced. This approach allows for a good approximation of the original data using the most important components.



Principal components analysis. $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ are the first two principal components for the given data.

Fig 2.8: Illustration of PCA

PCA is efficient in terms of computation, can handle both ordered and unordered attributes, and is effective with sparse and skewed data. It can manage multidimensional data by reducing it to two dimensions if necessary. Principal components can also serve as inputs for regression and cluster analysis. Compared to wavelet transforms, PCA generally performs better with sparse data. Refer figure 2.8 for PCA Illustration.

Data discretization

Data discretization techniques can be used to reduce the number of values for a given continuous attribute by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values. Replacing numerous values of a continuous attribute by a small number of interval labels thereby reduces and simplifies the original data. This Leads Concise, easy-to-use, knowledge-level representation of mining results.

Discretization techniques can be classified based on how they are applied, either using class information or based on their method of execution (i.e., top-down vs. bottom-up). If class information is utilized in the discretization process, it is referred to as supervised discretization. Conversely, if class information is not used, it is termed unsupervised discretization.

Top-down discretization, also known as splitting, begins by identifying one or more points (known as split points or cut points) to divide the entire range of an attribute. This process is then applied recursively to the resulting intervals. On the other hand, bottom-up discretization, or merging, starts by considering all possible split points and then merges neighboring values to form intervals, recursively applying this method to the resulting intervals.

Discretization can be performed recursively to create a hierarchical or multi-resolution partitioning of attribute values, known as a concept hierarchy. Concept hierarchies are beneficial for analyzing data at various levels of abstraction. For instance, a concept hierarchy for a numerical attribute might categorize values into broader groups, such as replacing specific ages with broader categories like youth, middle-aged, or senior. Although this generalization results in a loss of detail, it often makes the data more interpretable and consistent across different mining tasks. Additionally, mining on a reduced, generalized dataset is more efficient and requires fewer input/output operations compared to working with a larger, more detailed dataset. Therefore, discretization techniques and concept hierarchies are generally applied as a preprocessing step before data mining, rather than during the mining process itself. Multiple concept hierarchies can be defined for the same attribute to meet different user needs.

Concept Hierarchy Generation for Numerical Data

Binning

Binning is a technique that involves dividing data into a specified number of bins through a top-down approach. This method is used for data smoothing and can also aid in reducing numerosity and creating concept hierarchies. For example, you can discretize attribute values by using equal-width or equal-frequency binning, and then replace each bin's value with the bin mean or median—known as smoothing by bin means or smoothing by bin medians. These techniques can be applied recursively to further partition the data, generating hierarchical concepts. Binning is an unsupervised discretization method as it does not rely on class information, and its effectiveness can be influenced by the number of bins specified and the presence of outliers.

Histogram Analysis

Similar to binning, histogram analysis is an unsupervised discretization technique that does not use class information. It involves dividing attribute values into distinct ranges called buckets.Histograms partition data into ranges. In an equal-width histogram, the ranges are evenly sized (e.g., $10 increments for price). In an equal-frequency histogram, the data is divided so that each range contains an approximately equal number of data points. Histogram analysis can be applied recursively to generate a multilevel concept hierarchy, with the process stopping once a predefined number of levels is reached. A minimum interval size can be set to control the recursion, specifying the smallest range width or the minimum number of data points per partition at each level. Additionally, histograms can be further partitioned based on cluster analysis of the data distribution.

Entropy-Based Discretization

Entropy-based discretization is a method for converting continuous data into discrete intervals, using concepts from information theory. Introduced by Claude Shannon, entropy measures the uncertainty or impurity in data. In discretization, this method uses entropy to find the best way to split data into intervals that are useful for classification tasks.

Here’s how entropy-based discretization works:

1. **Identify Potential Split Points**: For a given numerical attribute A, consider each distinct value of A as a potential split point. Each split point divides the data into two subsets: one where A is less than or equal to the split point, and one where A is greater than the split point.
2. **Calculate Entropy**: To determine the best split point, calculate the entropy for each potential split. Entropy measures how mixed the class labels are within each subset created by the split. The goal is to find the split that minimizes the entropy, meaning that the split results in the clearest classification of the data. This is done by computing the expected information requirement (or entropy) for each split point and choosing the one with the lowest value.
3. **Recursive Partitioning**: After selecting the optimal split point, apply the same process recursively to each resulting interval. This continues until a stopping criterion is met, such as reaching a predefined number of intervals or when further splits no longer significantly improve classification.

Entropy-based discretization helps in improving classification accuracy by effectively using class information to determine split points. It creates a hierarchy of intervals that can represent the attribute in a more manageable and interpretable way.

Interval Merging by χ^2 Analysis

ChiMerge is another method used for discretizing numerical data, but it operates in a bottom-up manner. Instead of starting with potential splits and refining them, ChiMerge begins by treating each distinct value of an attribute as its own interval and then merges adjacent intervals based on their class distribution.

Here's a summary of the ChiMerge process:

1. **Initial Intervals**: Start by treating each distinct value of a numerical attribute as a separate interval.
2. **Merge Intervals**: Perform χ^2 tests on adjacent intervals to assess how similar their class distributions are. If two adjacent intervals have similar distributions (indicated by a low χ^2 value), they are merged into a single interval. This process continues recursively.
3. **Stopping Criteria**: The merging process stops based on several criteria. These can include a threshold for the χ^2 values, a maximum number of intervals, or a predefined level of class frequency consistency within intervals.

ChiMerge helps simplify data by combining intervals with similar class distributions, making the data more manageable while retaining meaningful distinctions.

Cluster Analysis

Cluster analysis involves grouping data into clusters based on similarity. This method can be used for discretizing numerical attributes by partitioning them into clusters or groups.

Key points about cluster-based discretization:

1. **Clustering**: Apply a clustering algorithm to the data, which groups similar values together. These clusters represent discrete intervals of the attribute.
2. **Hierarchy Creation**: Depending on the clustering strategy, the resulting clusters can either be further divided (top-down) or combined (bottom-up) to form a concept hierarchy.

Clustering methods produce a concept hierarchy by either creating finer clusters from existing ones or by merging neighboring clusters, providing a natural way to discretize numerical data.

Discretization by Intuitive Partitioning

For those who prefer simpler, more intuitive intervals, the 3-4-5 rule can be applied. This method divides numerical ranges into intervals that are easy to understand and interpret.

Here's how the 3-4-5 rule works:

1. **Determine Interval Size**: Based on the number of distinct values in the most significant digit of the data, partition the range into 3, 4, or 5 equal-width intervals. This rule helps in creating intervals that are more natural and easy to understand.
2. **Recursive Application**: Apply this rule recursively to create a hierarchy of intervals. For example, salaries might be divided into ranges like \$50,000-\$60,000, making the data more readable.
3. **Handling Outliers**: To avoid distortion from extreme values, base the top-level discretization on the majority of the data (e.g., between the 5th and 95th percentiles), and handle extreme values separately.

The 3-4-5 rule provides a straightforward way to create understandable intervals, making the data easier to analyze and interpret.

Concept Hierarchy Generation for Categorical Data

Categorical data consists of discrete values with no inherent ordering. Examples include geographic locations, job categories, and item types. Generating concept hierarchies for categorical data helps in organizing these attributes into meaningful levels of abstraction. Here's how you can create concept hierarchies for categorical data:

1. Specification of Partial Ordering at the Schema Level

One straightforward method for generating concept hierarchies is to define them explicitly. This approach involves specifying a partial or total ordering of attributes directly at the schema level. For example, in a geographical database, you might have attributes like street, city, province or state, and country. By specifying the total ordering as:

- Street < City < Province/State < Country

You create a hierarchy that reflects this ordering. This method is intuitive and allows users or experts to directly define how different attributes relate to each other.

2. Specification of a Portion of a Hierarchy by Explicit Data Grouping

In practice, defining an entire concept hierarchy manually can be cumbersome, especially for large databases. Instead, users can define only part of the hierarchy manually. For example, after establishing that "province" and "country" are part of a hierarchy, you might manually group specific provinces into regions:

- {Alberta, Saskatchewan, Manitoba} ⊂ Prairies Canada
- {British Columbia, Prairies Canada} ⊂ Western Canada

This approach allows for incremental development of the hierarchy, focusing on relevant subsets of data.

3. Automatic Hierarchy Generation from a Set of Attributes

When explicit ordering is not provided, an automatic approach can be used to generate a concept hierarchy. The idea is based on the observation that higher-level concepts typically have fewer distinct values compared to lower-level ones. For instance, "country" generally has fewer distinct values than "street".

Here's how you can generate a hierarchy automatically:

1. **Count Distinct Values**: For each attribute, count the number of distinct values it has.
2. **Order Attributes**: Arrange attributes in descending order based on the number of distinct values. Attributes with more distinct values are placed at lower levels of the hierarchy, while those with fewer values are placed at higher levels.
3. **Generate Hierarchy**: Construct the hierarchy based on this ordering. For example, if you have attributes like "city" with many values and "continent" with fewer values, "city" would be at a lower level, and "continent" would be at a higher level.

This heuristic approach works well in many cases, although it may need adjustments based on specific data characteristics. Users or experts might need to tweak the automatically generated hierarchy to better fit the actual data semantics.

UNIT EXERCISE

2 Marks

1. What is KDD and explain its uses in real life.
2. Why is Data Preprocessing important?

1. How do you handle missing values?
2. Mention the different data normalization techniques

3. Differentiate CWT and DWT.
4. What are the steps in the 3-4-5 rule?
5. Explain the steps in the Chi-Merge process.
6. Differentiate entropy based discretization and Cluster based discretization.
7. Draw a diagram which illustrates the essence of PCA by assuming the necessaries.
8. What is Max-Min Normalization and write the equation.

8 Marks

1. What is PCA and explain the steps involved in it?
2. Explain Data cube aggregation with an example.
3. Explain the Binning method with an example.

Unit_03 - Association Rule Mining

Frequent patterns are patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (frequent) sequential pattern. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (frequent) structured pattern. Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks as well. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

Market Basket Analysis: A Motivating Example

Introduction

Frequent itemset mining is a crucial technique in data mining that uncovers associations and correlations among items in extensive transactional or relational databases. With the exponential growth of data collected and stored by businesses, the ability to analyze this data and extract meaningful patterns is becoming increasingly valuable. One of the most prominent applications of frequent itemset mining is **Market Basket Analysis**.

Market Basket Analysis

Market Basket Analysis involves examining customer purchasing behavior by analyzing the items that customers buy together during a shopping trip. The goal is to identify patterns and relationships between products, which can then be used to make informed business decisions.

Objectives

1. **Catalog Design**: By understanding which items are frequently bought together, retailers can design more effective product catalogs and organize their offerings to maximize sales.
2. **Cross-Marketing**: Retailers can create targeted marketing campaigns based on item associations, such as offering promotions on complementary products.
3. **Shelf Space Planning**: Insights from market basket analysis help in optimizing store layout by placing frequently co-purchased items in close proximity, improving the shopping experience and potentially increasing sales.

Process

1. **Data Collection**: Collect transactional data from point-of-sale systems, which includes details about items purchased together in each transaction.

2. **Frequent Itemset Mining**: Use algorithms like Apriori or FP-Growth to discover frequent itemsets — groups of items that appear together in transactions more frequently than a specified threshold.
3. **Association Rule Mining**: Generate association rules from the frequent itemsets. For example, a rule might indicate that customers who buy milk are also likely to buy bread.
4. **Analysis and Application**: Analyze the discovered rules to develop marketing strategies, adjust store layouts, or create promotions.

Frequent Itemsets, Closed Itemsets, and Association Rules

Let I= {11, 12,..., Im} be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that T⊆I. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if A⊆T. An association rule is an implication of the form A⇒B, where A⊂ I, B⊂ I. The rule A⇒B holds in the transaction set D with support s, where s is the percentage of transactions in D that contain AUB (i.e., the union of sets A and B, or say, both A and B). This is taken to be the probability, P(AUB).[1] 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). That is,

$$support(A\ with\ B) = P(A \cup B)$$

$$confidence(A) = P(B/A) \qquad \textbf{(3.1, 3.2)}$$

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.2 An itemset that contains k items is a k-itemset. The set {computer, antivirus_software) is an 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 defined in Equation (5.2) 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 L_k.

$$confidence(A \rightarrow B) = support(A \cap B) / support(A) \qquad \textbf{(3.3)}$$

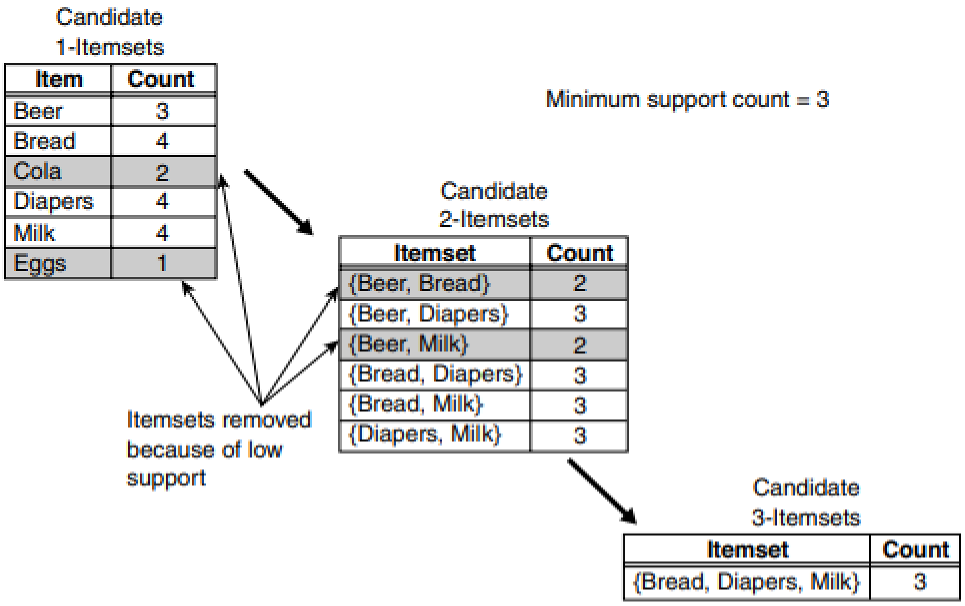
In general, association rule mining can be viewed as a two-step process: 1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min sup. 2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation

Apriori Principle: If an itemset is frequent, then all of its subsets must also be frequent.

Monotonicity Property: Let I be a set of items, and J = 2I be the power set of I. A measure f is monotone (or upward closed) if $\forall X, Y \in J : (X \subseteq Y) \longrightarrow f(X) \leq f(Y)$

Frequent Itemset Generation in the Apriori Algorithm Apriori is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets. The below figure provides a high-level illustration of the frequent itemset generation part of the Apriori algorithm for the transactions shown in figure 3.1. We assume that the support threshold is 60%, which is equivalent to a minimum support count equal to 3.


Candidate 1-Itemsets
Minimum support count = 3
Candidate 2-Itemsets
Itemsets removed because of low support
Candidate 3-Itemsets

Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Itemset	Count
{Beer, Bread}	2
{Beer, Diapers}	3
{Beer, Milk}	2
{Bread, Diapers}	3
{Bread, Milk}	3
{Diapers, Milk}	3

Itemset	Count
{Bread, Diapers, Milk}	3

Fig 3.1: Transaction table

Initially, every item is considered as a candidate 1-itemset. After counting their supports, the candidate itemsets {Cola} and {Eggs} are discarded because they appear in fewer than three transactions. In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the Apriori principle ensures that all supersets of the infrequent 1-itemsets must be infrequent. Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is (4 2) = 6. Two of these six candidates, {Beer, Bread} and {Beer, Milk}, are subsequently found to be infrequent after computing their support values. The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets. Without support-based pruning, there are (6 3) = 20 candidate 3-itemsets that can be formed using the six items given in this example. With the Apriori principle, we only need to keep candidate 3-itemsets whose subsets are frequent. The only candidate that has this property is {Bread, Diapers, Milk}.

Algorithm for Frequent Itemset generation using Apriori

Input: Transaction database *T*, Minimum support threshold *minsup*
Output: Frequent itemsets

```
1    $k \leftarrow 1$;
2    $F_k \leftarrow \{i \mid i \in I$ and $\sigma(\{i\}) \geq N \times minsup\}$ ; // Find all frequen 1-itemsets
3    repeat
4      $k \leftarrow k + 1$;
5      $C_k \leftarrow apriori - gen(F_{k-1})$ ;       // Generate candidate itemsets
6      for each transaction $t \in T$ do
7          $C_t \leftarrow subset(C_k, t)$ ;              // Identify all candidates that belong to t
8          for each candidate itemset $c \in C_t$ do
9              $\sigma(c) \leftarrow \sigma(c) + 1$ ;         // Increment support count
10         end for
11     end for
12     $F_k = \{c \mid c \in C_k$ and $\sigma(c) \geq N \times minsup\}$ ; // Extract the frequent k-itemsets
13   until $F_k = \emptyset$ ;
14   Result $\leftarrow \cup F_k$
```

• To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in Ck that are contained in each transaction t.

• After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than minsup (step 12).

• The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_K = \emptyset$ (step 13).

Example with explanation

Use of Apriori Algorithm for Association Rule Mining

Implementing the Apriori algorithm performs the task of association rule mining, carefully examining the data to discover patterns. Let's continue the above example.

There can be thousands of products that a supermarket might be selling at any given point in time. As they record the transactions, i.e., the products sold to customers, association mining is possible on such transaction data. A general association rule mining algorithm finds all possible associations, i.e., combinations of all the products in transaction data. Then, it loops these combinations through every transaction to find the association rules.

To put what I said in context, let’s say a supermarket sells four products – A, B, C, and D. This would mean that there can be 11 ways one can buy a combination of these products.

1. A, B
2. A, C
3. A, D
4. B, C
5. B,D
6. C, D
7. A, B, C
8. B, C, D
9. A, C, D
10. A,B,D
11. A,B,C,D

In association rule mining, the Apriori algorithm efficiently discovers item combinations by iteratively looping through transactions, initially counting occurrences to identify large 1-itemsets. This approach avoids the inefficiency of calculating all possible combinations in a single iteration, especially with numerous items.

The 1-itemsets in our example (where you were dealing with four items – A, B, C, and D) would be {A}, {B}, {C}, {D}. A certain threshold is then identified, and if it is found that the count of occurrences of these products is less than the predetermined threshold, then that particular product is discarded from all the plausible combinations. This dramatically reduces the total number of combinations, making the process faster.

To properly answer what the Apriori algorithm is, you first need to explore the important metrics calculated during its working.

Key Metrics for Apriori Algorithm

As discussed above, the number of possible associations can be in the thousands, especially if the number of items is large. The question is now how to identify the associations that are better than the others. This is where key metrics of the Apriori algorithm come into play.

When implementing the Apriori algorithm, we calculate three key metrics—Support, Confidence, and Lift—to identify the best associations. Each of these parameters can be regarded as an Apriori property. They are computed for various associations and then compared against a predetermined threshold. Let's discuss these parameters in detail.

Support: Support indicates an item's popularity, calculated by counting the transactions where that particular item was present. For item 'Z,' its Support would be the number of times the item was purchased, as the transaction data indicates.

Sometimes, this count is divided by the total number of transactions to make the number easily representable. Let's understand Support with an example. Suppose there is transaction data for a day having 1,000 transactions.

The items you are interested in are apples, oranges, and apples+oranges (a combination item). Now, you count the transactions where these items were bought and find that the count for apples, oranges, and apples+oranges is 200, 150, and 100.

The formula for Support is:

Support (Z) = Transactions containing item Z / Total transactions

The Support, therefore, for the above-mentioned association would be

Support(Apple) = Count of Transaction where Apples bought / Total number of transactions

Support(Apple) = 200 / 1000

Support(Apple) = 0.20

Support(Orange) = Count of Transaction where Oranges bought / Total number of transactions

Support(Orange) = 150 / 1000

Support(Orange) = 0.15

Support(Apple+Orange) = Count of Transaction where Apples+Oranges bought / Total number of transactions

Support(Apple+Orange) = 100 / 1000 = 0.10

Support(Apple+Orange) = 0.10

In the Apriori algorithm, such a metric is used to calculate the "support" for different items and itemsets to establish that the frequency of the itemsets is enough to be considered for generating candidate itemsets for the next iteration. Here, the support threshold plays a crucial role as it's used to define items/itemsets that are not frequent enough.

Confidence

Another Apriori property is Confidence. This key metric is used in the Apriori algorithm to indicate the probability of an item 'Y' being purchased if a customer has bought an item 'Z'. If you notice, here, conditional probability is getting calculated, i.e., in this case, it's the conditional probability that item Z appears in a transaction, given that another item Y appears in the same transaction. Therefore, the formula for calculating Confidence is

P(Z|Y) = P(Y and Z) / P(Y)

It can also be written as

Support(Y ∪ Z) / Support(Y)

Confidence is typically denoted by

(Y → Z)

In the above example, apples and oranges were bought together in 100 transactions, while apples were bought in 200 transactions. In such a case, the Confidence (Apples → Oranges) would be

Confidence (Apples → Oranges) = Transaction containing both Apples and Oranges / Transaction only containing Apples

Confidence (Apples → Oranges) = 100 / 200

Confidence (Apples → Oranges) = 0.5

The value of Confidence can be interpreted by using a predetermined threshold value. If the value of Confidence is beyond the predetermined threshold value, then it indicates Z is more likely to be purchased with item Y.

In the above example, a value of Confidence = 0.5 means that the association between “apple” and “orange” is 0.5, which means that when a customer buys “apple”, there is a 50% chance that they will also buy “orange”.

This information can be useful in recommending products to customers or product placement optimization in a store. If, for example, your predetermined threshold value is 0.3 then this association can be considered.

Lift

After calculating metrics such as Support and Confidence, you can reduce the number of associations by selecting a threshold value for these metrics, considering associations beyond the set threshold.

However, even after applying thresholds for such metrics, there still can be a huge number of associations that require further filtering. Here, another metric known as Lift can be helpful. Lift denotes the strength of an association rule. Suppose you need to calculate the Lift(Y → Z); then you can do so by dividing Confidence(Y → Z) by Support(Z), i.e.,

Lift(Y → Z) = Confidence(Y → Z) / Support(Z)

Another way of calculating Lift is by considering Support of (Y, Z) and dividing by Support(Y)*Support(Z), i.e., it’s the ratio of Support of two items occurring together to the Support of the individual items multiplied together.

In the above example, the Lift for Apples → Oranges would be the following-

Lift(Apple → Orange) = Confidence(Apple → Orange) / Support(Orange)

Lift(Apple → Orange) = 0.5 / 0.15

Lift(Apple → Orange) = 33.33

Support(Y) and Support(Z) in the denominator indicate the independent occurrence of Y and Z in transactions. A high value in the denominator of Lift indicates that there is no such association rule as the purchase happens because of randomness. Therefore, Lift helps you to identify the association rule to consider.

A Lift value of 1 generally indicates randomness, suggesting independent items, and the association rule can be disregarded. A value above 1 signifies a positive association, indicating that two items will likely be purchased together. Conversely, a value below 1 indicates a negative association, suggesting that the two items are more likely to be purchased separately.

Now, with a grasp of the Apriori algorithm's fundamental concepts and key metrics, let's delve into understanding this algorithm's role in data mining.

What is the Apriori Algorithm in Data Mining?

As you would have noticed by now, the Apriori algorithm is perfect for performing tasks such as market basket analysis. The Apriori algorithm in data mining can help data analysts understand the underlying patterns in their data and help businesses handle their customers better. This section discusses how the Apriori algorithm works when mining data.

Apriori algorithm Steps in Data Mining

Typically, Apriori algorithm steps in data mining are the following-

- **Define minimum threshold**
  - The first step is to decide on the threshold value for the support metric. The metric determining the minimum number of times an item should appear in the data to be considered significant is "Support." The support value is based on data size, domain knowledge, or other considerations.
- **Create a list of frequent items**
  - After determining the support threshold, the subsequent step involves scanning the entire dataset to identify items that meet the support threshold. The selected itemsets, meeting the support threshold, are called frequent itemsets.
- **Create candidate item sets**

   - ➢ The next step is to use the previously identified frequent k-item sets and generate a list of candidate item sets. The length of these candidate itemsets is k+1.
- ❖ **Calculate the Support of each candidate**
   - ➢ The algorithm needs to scan the dataset again and count the frequency of every candidate item, i.e., the number of times each item appeared in the data.
- ❖ **Prune the candidate item sets**
   - ➢ The minimum threshold is again used to remove itemsets that fail to meet the minimum support threshold.
- ❖ **Iterate**
   - ➢ This can be considered the most crucial stage of the Apriori algorithm. Steps 3 – 5 are repeated until no frequent itemsets can be generated.
- ❖ **Generate Association Rules**
   - ➢ The algorithm now generates the association rules using the final frequent item sets identified at the end of the previous step.
- ❖ **Evaluate Association Rules**
   - ➢ Metrics such as Confidence and Lift can be employed to filter the relevant association rules. At the end of this step, you get association rules that indicate the probability of a customer purchasing an item Z if they have already purchased an item Y (here, Y and Z are itemsets).

To better understand all the Apriori algorithm steps in data mining mentioned above, I will explain the Apriori algorithm with example transaction data and create association rules.

Understanding Apriori Algorithm with an example

Let me now explain the apriori algorithm with an example. Suppose you are dealing with the following transaction data as in Table 3.1.

Table 3.1: Transaction Table Example

Transaction ID	Items
Tranx001	{egg,bread}
Tranx002	{juice,egg.butter}
Tranx003	{juice,egg,bread}

Tranx004	{juice,bread}
Tranx005	{juice,egg}
Tranx006	{juice,bread,butter}
Tranx007	{juice,egg,butter}
Tranx008	{bread,butter}
Tranx009	{juice,bread}
Tranx0010	{egg,butter}
Tranx0011	{juice,egg,butter}

Step 1: Deciding Threshold

In this example, the threshold value of Support is considered as 3. Therefore, an item must appear in at least three transactions to be considered frequent.

Step 2: Computing Support

The frequency for each unique item in the data is calculated as in Table 3.2., i.e., the Support for 1-itemsets.

Table 3.2: Example - Support for K = 1

Item	Support(Frequency)
Bread	5
Butter	7
Egg	7
Juice	8

As all the 1-itemsets support is more than the support threshold of 3, they can be classified as frequent 1-itemsets and used to generate the candidates for 2-itemsets.

Step 3: Forming Candidate Itemsets

The above frequent 1-itemsets are considered to generate the candidate 2-itemset. Here, the combinations for all the above-identified frequent products are calculated such that each combination consists of two items. For the items bread, butter, egg, and juice, the 2-itemsets are

1. {bread, butter}
2. {egg, bread}
3. {egg, butter}
4. {juice, bread}
5. {juice, butter}
6. {juice, egg}

Step 4: Finding Frequent Combinations

The transaction data is scanned for the above combinations, and their support value is calculated, as seen below.

Table 3.3: Transaction table Example with Support for K = 2

Candidate Item Sets	**Support (Frequency)**
{bread, butter}	2
{egg, bread}	2
{egg, butter}	3
{juice, bread}	3
{juice, butter}	5
{juice, egg}	5

According to the above data, juice and egg were purchased together in five transactions as in Table 3.3, whereas egg and bread were bought together in only two transactions. Now, if you go by the support threshold value of 3, then the frequent 2-itemsets would be the following-

1. {egg, butter}
2. {juice, bread}
3. {juice, butter}
4. {juice, egg}

These frequent 2-itemsets can now be used to generate candidates for 3-itemset, and they would be

1. {juice, bread, butter}
2. {juice, egg, bread}
3. {juice, egg, butter}

The support value for the above itemsets would be as follows:

Table 3.4: Transaction table Example with Support for K = 3

Candidate Item Sets	**Support (Frequency)**
{juice, bread, butter}	1
{juice, egg, bread}	1
{juice, egg, butter}	3

As you would have noticed, of the 3-itemsets, only one can be considered frequent as only {juice, egg, butter} itemset has the support value of 3 or above as in Table 3.4. Now, as only one itemset remains, a candidate for 4-itemsets cannot be generated, leading us to terminate the process of finding the frequent itemsets. Therefore, the association rules can now be generated.

Step 5: Forming Association Rules

The main purpose of explaining the Apriori algorithm with examples in data mining was to show you how one can come up with association rules. In our example, for the items egg, butter, and juice, the following association rules can be generated

1. {egg, butter} → {juice}
2. {juice, butter} → {egg}
3. {juice, egg} → {butter}

Step 6: Calculating other metrics

For this example, I am using the confidence metric to define the utility of my association rules as in Table 3.5.

Final Table

Table 3.5: Transaction table Example with Support and Confidence for K = 2

Candidate Item Sets	**Support(Frequency)**	**Confidence**
{juice, butter}	{egg}	60
{juice, egg}	{butter}	100
{egg, butter}	{juice}	60

If you consider a confidence threshold of 0.7, i.e., 70%, then of all the associations mentioned above rules, the one that you can use to recommend products to the customer or optimize product placement would be {egg, butter} □ {juice} as here you find a high value of Confidence that if a customer buys egg and butter, they are highly likely to purchase juice also.

So far, the discussion around the Apriori algorithm has been in terms of data mining and particularly market basket analysis. Machine learning and big data analytics fields also use this algorithm.

Mining Frequent Itemsets without Candidate Generation

As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs:

It may need to generate a huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets. Moreover, to discover a frequent pattern of size 100, such as {a1,..., a 100}, it has to generate at least $2^{100} - 1 \approx 10^{30}$ candidates in total. It may need to repeatedly scan the database and check a large set of candidates by pattern matching. It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

“Can we design a method that mines the complete set of frequent itemsets without candidate generation?” An interesting method in this attempt is called frequent-pattern growth, or simply FP-growth, which adopts a divide-and-conquer strategy as follows. First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then 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.

You’ll see how it works with the following example in Table 3.6.

Table 3.6: Transaction data example

TID	***List of Items_IDs***
T100	I1,I2,I5
T200	I2,I5
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

FP-growth (finding frequent itemsets without candidate generation). We re-examine the mining of transaction database, D, of Table 3.1 using the frequent- pattern growth approach.

The first scan of the database is the same as Apriori, which derives the set of fre- quent items (1-itemsets) and their support counts (frequencies). Let the minimum sup- port count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or list is denoted L. Thus, we have L = {{12: 7}, {11: 6}, {13: 6}, {14: 2}, {15: 2}}. 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. For example, the scan of the first transaction, "T100: 11, 12, 15," which contains three items (12, 11, 15 in L order), leads to the construction of the first branch of the tree with three nodes, (12: 1), (11:1), and (15: 1), where 12 is linked as a child of the root, 11 is linked to 12, and 15 is linked to 11. The second transaction, T200, contains the items 12 and 14 in L order, which would result in a branch where 12 is linked to the root and 14 is linked to 12. However, this branch would share a common prefix, 12, with the existing path for T100. Therefore, we instead increment the count of the 12 node by 1, and create a new node, (14: 1), which is linked as a child of (12:2). In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links. The tree obtained after scanning all of the transactions is shown in Fig 3.2 with the associated node-links. In this way, the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree.

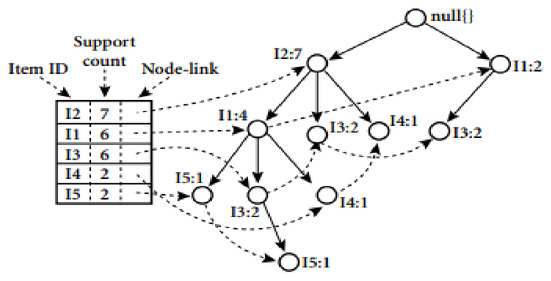


Fig 3.2: FP Tree example

Table 3.7: FP-tree example

Item	***Conditional Pattern Base***	***Conditional FP-tree***	***Frequent Pattern Generated***
I5	{{I2,I1 : 1},{I2,I1,I3: 1}}	{I2 : 2, I1 : 2}	{I2,I5 : 2},{I1,I5 : 2},{I2,I1,I5 : 2}
I4	{{I2,I1 : 1},{I2: 1}}	{I2: 2}	{I2,I4 : 2}
I3	{{I2,I1 : 2},{I2 : 2}, {I1 : 2}}	{I2 : 4, I1 : 2}{I1 : 2}	{I2,I3 : 4},{I1,I3 : 4},{I2,I1,I3 : 2}
I1	{{I2: 4}}	{I2: 4}	{I2,I1 : 4}

The FP-tree is mined as follows as shown in Table 3.7. 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 such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. Mining of the FP-tree is summarized in Table 3.7 and detailed as follows. We first consider I5, which is the last item in L, rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two branches of the FP-tree of figure 3.3. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are hI2, I1, I5: 1i and hI2, I1, I3, I5: 1i. Therefore, considering I5 as a suffix, its corresponding two prefix paths are hI2, I1: 1i and hI2, I1, I3: 1i, which form its conditional pattern base. Its conditional FP-tree contains only a single path, hI2: 2, I1: 2i; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}. For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, hI2: 2i, and derives one frequent.pattern, {I2, I1: 2}. Notice that although I5 follows I4 in the first branch, there is no need to include I5 in the analysis here because any frequent pattern involving I5 is analyzed in the examination of I5. Similar to the above analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}. Its conditional FP-tree has two branches, hI2: 4, I1: 2i and hI1: 2i, as shown in Figure 5.8, which generates the set of patterns, {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}. Finally, I1's conditional pattern base is {{I2: 4}}, whose FP-tree contains only one node, hI2: 4i, which generates one frequent pattern, {I2, I1: 4}. This mining process is summarized in Figure 5.9. The FP-growth method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs. When the database is large, it is sometimes unrealistic to construct a main memory based FP-tree. An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database. Such a process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory. A study on the performance of the FP-growth method 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. It is also faster than a Tree-Projection algorithm, which recursively projects a database into a tree of projected databases.

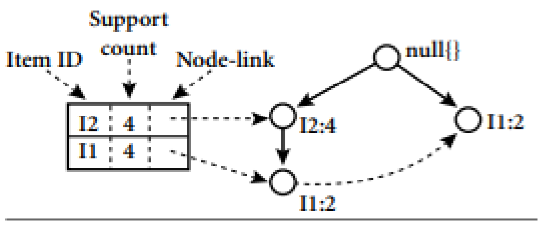


Fig 3.3: FP Tree

Mining Various Kinds of Association Rules

We have studied efficient methods for mining frequent itemsets and association rules. In this section, we consider additional application requirements by extending our scope to include mining multilevel association rules, multidimensional association rules, and quantitative association rules in transactional and/or relational databases and data warehouses. Multilevel association rules involve concepts at different levels of abstraction. Multidimensional association rules involve more than one dimension or predicate (e.g., rules relating what a customer buys as well as the customer's age.) Quantitative association rules involve numeric attributes that have an implicit ordering among values (e.g., age).

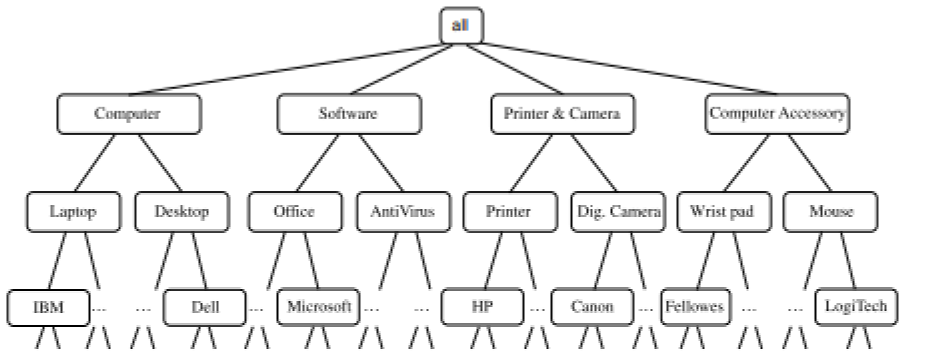
Mining Multilevel Association Rules:

For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data at those levels. Strong associations discovered at high levels of abstraction may represent commonsense knowledge. Moreover, what may represent common sense to one user may seem novel to another. Therefore, data mining systems should provide capabilities for mining association rules at multiple levels of abstraction, with sufficient flexibility for easy traversal among different abstraction spaces. Let's examine the following example.

Example: Mining multilevel association rules. Suppose we are given the task-relevant set of transactional data in Table 3.8 for sales in an AllElectronics store, showing the items purchased for each transaction. The concept hierarchy for the items is shown in Table 3.8. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher level, more general concepts. Data can be generalized by replacing low-level concepts within the data by their higher-level concepts, or ancestors, from a concept hierarchy as in figure 3.4.

Table 3.8: Task-relevant data, D

TID	***Items Purchased***
T100	IBM-ThinkPad-T40/2373,HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003,Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003
………..	………..

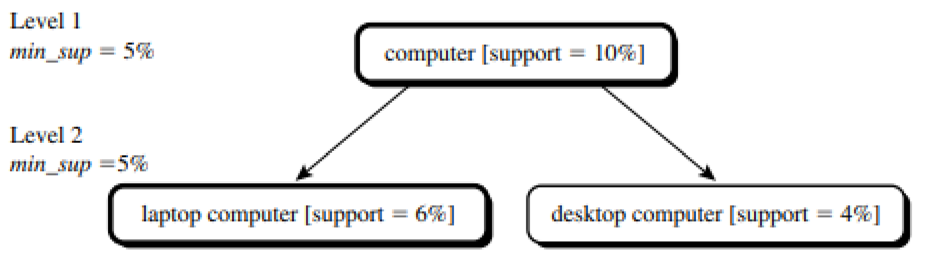


A concept hierarchy for *AllElectronics* computer items.

Fig 3.4: Concept Hierarchy Example - Tree

Association rules generated from mining data at multiple levels of abstraction are called multiple-level or multilevel association rules. Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework.

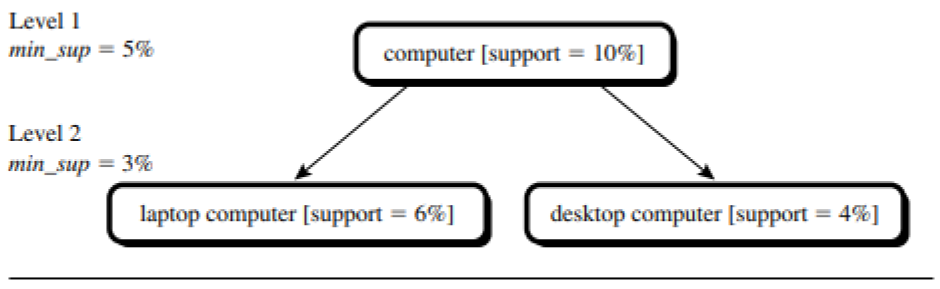
- Using uniform minimum support for all levels: The same minimum support threshold is used when mining at each level of abstraction. For example, in figure 3.5, a minimum support threshold of 5% is used throughout (e.g., for mining from "computer" down to "laptop computer"). Both "computer" and "laptop computer" are found to be frequent, while "desktop computer" is not. When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. An Apriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendants: The search avoids examining itemsets containing any item whose ancestors do not have minimum support.



Multilevel mining with uniform support.

Fig 3.5: Multi Level Mining with uniform support

- Using reduced minimum support at lower levels: Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. For example, in figure 3.6, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, "computer," "laptop computer," and "desktop computer" are all considered frequent.



Multilevel mining with reduced support.

Fig 3.6: Multi Level Mining with reduced support

- Using item or group-based minimum support (referred to as group-based support): Because users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price, or on items of interest, such as by setting particularly low support thresholds for laptop computers and flash drives in order to pay particular attention to the association patterns containing items in these categories.

Mining Multidimensional Association Rules from Relational Databases and Data Warehouses

Mining multidimensional association rules from relational databases and data warehouses involves discovering interesting relationships between different attributes across multiple dimensions. Here are some key points about this process:

What are Multidimensional Association Rules?

Multidimensional association rules involve more than one dimension or predicate. For example, a rule might look like this:

Age(X,"20...29")∧ Occupation(X,"Student")⇒Buys(X,"Laptop")

This rule indicates that students aged between 20 and 29 are likely to buy laptops.

Approaches to Mining Multidimensional Association Rules

1. **Static Discretization of Quantitative Attributes**:
    - Quantitative attributes are discretized before mining.
    - Discretized attributes are treated as categorical.
    - The Apriori algorithm is used to find frequent predicate sets.
2. **Dynamic Discretization of Quantitative Attributes**:
    - Known as mining quantitative association rules.
    - Numeric attributes are dynamically discretized during the mining process.
    - Example:
      Age(X,"20..25")∧ Income(X,"30K..41K")⇒Buys(X,"Laptop")
3. **Distance-Based Discretization with Clustering**:
    - This involves clustering to find time periods or groups.
    - Association rules are derived by searching for groups of clusters that occur together.

Applications

- **Market Basket Analysis**: Analyzing customer buying habits by finding associations between items in their shopping baskets.
- **Web Usage Mining**: Discovering patterns in web data to understand user behavior.
- **Continuous Production**: Identifying relationships in production data to improve processes.

Example

In a retail scenario, a multidimensional association rule might reveal that customers aged 30-40 who buy smartphones are also likely to buy accessories like headphones and cases.

Benefits

- **Improved Decision Making**: Helps businesses make informed decisions based on discovered patterns.
- **Targeted Marketing**: Enables more effective marketing strategies by understanding customer behavior.
- **Enhanced Data Insights**: Provides deeper insights into data by considering multiple dimensions.

Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes

Mining multidimensional association rules using static discretization of quantitative attributes is a technique in data mining that helps uncover relationships among large datasets.

Multidimensional Association Rules

Multidimensional association rules involve more than one dimension or attribute. These attributes can be either categorical or quantitative. Quantitative attributes are numeric and need to be discretized before mining.

Static Discretization

Static discretization is a process where quantitative attributes are converted into categorical attributes before the mining process begins. This conversion is done by dividing the range of a numeric attribute into intervals, which are then treated as distinct categories.

Process

1. Discretization: Quantitative attributes are discretized into intervals. For example, an age attribute might be divided into intervals like 20-30, 31-40, etc.
2. Apriori Algorithm: The discretized attributes are then used in the Apriori algorithm to find frequent predicate sets. This algorithm scans the dataset multiple times to identify frequent itemsets.
3. Frequent Predicate Sets: Each subset of a frequent predicate set must also be frequent. For instance, if (age, income, purchases) is frequent, then (age, income), (age, purchases), and (income, purchases) must also be frequent.

Example

Consider a dataset with attributes like age, income, and purchases. After discretization, the age attribute might have intervals like 20-30, 31-40, etc. The Apriori algorithm would then be used to find frequent combinations of these intervals with other attributes.

Benefits

- Efficiency: Static discretization simplifies the mining process by converting numeric attributes into categorical ones, making it easier to apply traditional association rule mining algorithms.
- Interpretability: The resulting rules are easier to interpret since they involve categorical intervals rather than raw numeric values.

Constraint-Based Association Mining

In data mining, a process may uncover thousands of rules from a given dataset, many of which might be unrelated or uninteresting to users. Often, users have a good sense of which direction of mining may lead to interesting patterns and the form of the patterns or rules they would like to find. Therefore, a useful approach is to have users specify their intuition or expectations as constraints to narrow the search space. This strategy is known as constraint-based mining. The constraints can include:

- Knowledge type constraints: Specify the type of knowledge to be mined, such as association or correlation.
- Data constraints: Specify the set of task-relevant data.
- Dimension/level constraints: Specify the desired dimensions (or attributes) of the data, or levels of the concept hierarchies, to be used in mining.
- Interestingness constraints: Specify thresholds on statistical measures of rule interestingness, such as support, confidence, and correlation.
- Rule constraints: Specify the form of rules to be mined. These constraints may be expressed as metarules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates.

These constraints can be specified using a high-level declarative data mining query language and user interface.

Using rule constraints to focus the mining task allows users to describe the rules they would like to uncover, making the data mining process more effective. Additionally, a sophisticated mining query optimizer can exploit the constraints specified by the user, making the mining process more efficient. Constraint-based mining encourages interactive exploratory mining and analysis. For ease of discussion, we assume that the user is searching for association rules. The procedures presented can easily be extended to the mining of correlation rules by adding a correlation measure of interestingness to the support-confidence framework.

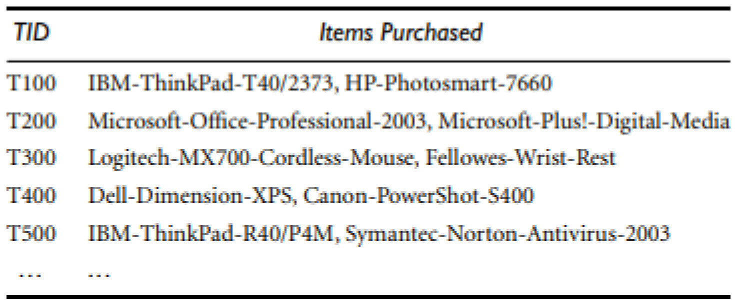
UNIT EXERCISE

2 Marks

1. What is Market Basket Analysis and why is it called so?
2. Write the formula for support and confidence.
1. What is the difference between lift and Confidence

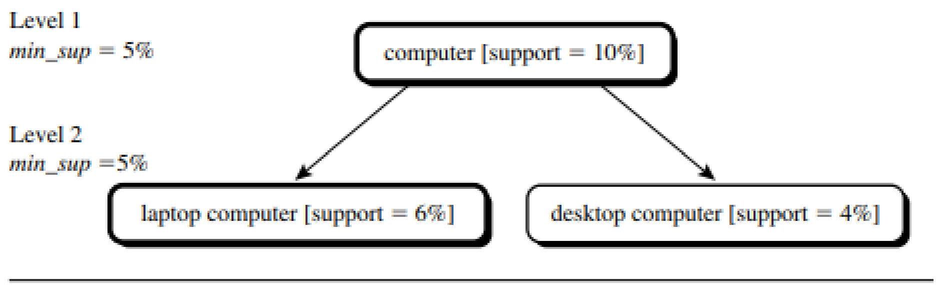
2. What is the concept behind Reduced support and Uniform support?
3. What are Multidimensional Association Rules?
4. What is static discretization?
5. Consider the table below and draw the first level of the Concept hierarchy tree.

Task-relevant data, *D*.



TID	*Items Purchased*
T100	IBM-ThinkPad-T40/2373, HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003, Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003
...	...

6. What are the different kinds of association?
7. Define Relative support in Association rule mining
8. Consider the diagram below. Is it a multi level mining with uniform support or not? If yes, explain how or else justify your answer.



Multilevel mining with uniform support.

8 Marks

1. Write all the steps in Apriori algorithm and apply it for the example:

   T1: A,B,C

   T2: B,C,D

   T3: A,B,D

   T4: A,B

T5: A,C,D

2. Explain the difference between mining frequent itemsets with and without candidate set generation.
3. What is constraint based association mining? Explain all the constraints possible.

Unit 04 - Classification vs. Prediction

Classification,

- Predicts categorical class labels (discrete or nominal).
- Classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data.

Prediction Models continuous-valued functions, i.e., predicts unknown or missing values.

Typical Applications

- Document categorization
- Credit approval
- Target marketing
- Medical diagnosis
- Treatment effectiveness analysis

Supervised vs. Unsupervised Learning

Supervised learning (classification):

- Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations.
- New data is classified based on the training set.

Unsupervised learning (clustering):

- The class labels of training data are unknown.
- Given a set of measurements, observations, etc., with the aim of establishing the existence of classes or clusters in the data.

Classification and Prediction Related Issues

- Data Preparation:
  - Data cleaning: Preprocess data to reduce noise and handle missing values.
  - Relevance analysis (feature selection): Remove irrelevant or redundant attributes.
- Data transformation: Generalize and/or normalize data.
- Performance Analysis:
  - Predictive accuracy: Ability to classify new or previously unseen data.
  - Speed and scalability: Time to construct the model and time to use the model.
  - Robustness: Model makes correct predictions, handling noise and missing values.
  - Scalability: Efficiency in disk-resident databases.
  - Interpretability: Understanding and insight provided by the model.
- Goodness of rules: Decision tree size and compactness of classification rules.

Common Test Corpora

- Reuters: Collection of newswire stories from 1987 to 1991, labeled with categories.
- TREC-AP: Newswire stories from 1988 to 1990, labeled with categories.

- OHSUMED: Medline articles from 1987 to 1991, MeSH categories assigned.
- UseNet newsgroups.
- WebKB: Web pages gathered from university CS departments.

Classification

A classical problem extensively studied by statisticians and machine learning researchers.

Predicts categorical class labels.

Example Applications:

{credit history, salary} → credit approval (Yes/No)

{Temp, Humidity} → Rain (Yes/No)

A set of documents → sports, technology, etc.

Another Example:

If $x >= 90$ then grade = A.

If $80 <= x < 90$ then grade = B.

If $70 <= x < 80$ then grade = C.

If $60 <= x < 70$ then grade = D.

If $x < 50$ then grade = F.

Classification types are shown in figure 4.1. They are:

o Distance based

o Partitioning based

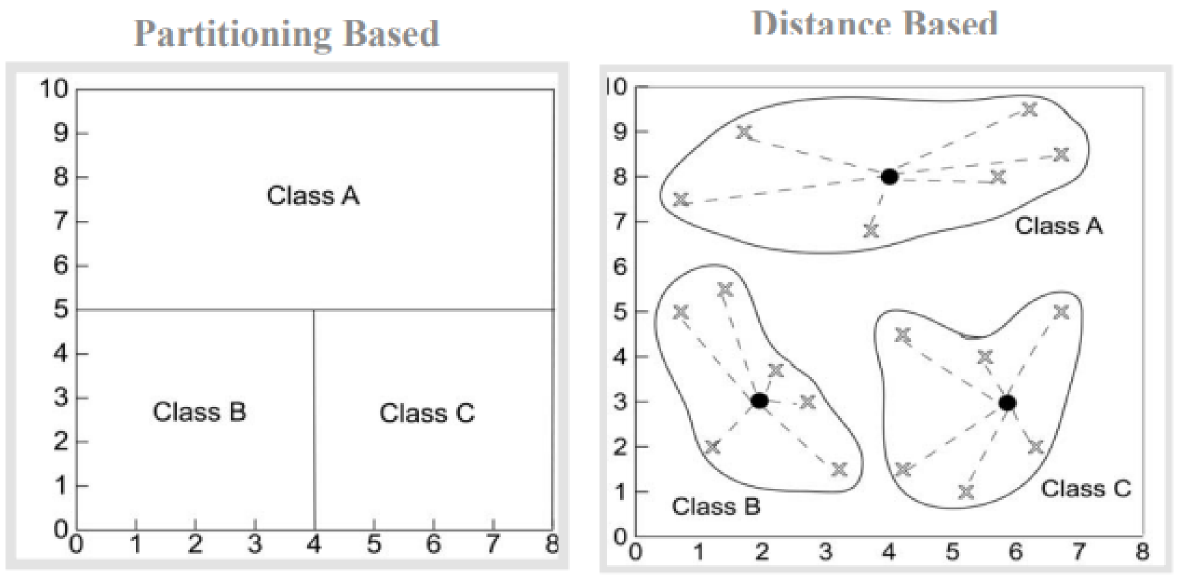


Fig 4.1: Types of Classification

Classification as a Two-Step Process

1. Model Construction

This is the initial phase where the model is built using a training dataset.

- Objective: Create a model that can accurately classify data into predetermined classes.
- Assumption: Each data sample belongs to a predefined class, as indicated by the class label attribute.
- Steps:
    1. Data Collection: Gather a training dataset where each sample is labelled with the correct class.
    2. Feature Selection: Identify and select the most relevant features (attributes) that will be used to build the model.
    3. Model Building: Use algorithms to construct the model. Common representations include:
        - Classification Rules: Simple if-then rules that classify data based on attribute values.
        - Decision Trees: Tree-like structures where each node represents a decision based on an attribute, and each branch represents the outcome of that decision.
        - Mathematical Formulas: Equations that define the boundaries between different classes.

4. Training: Apply the chosen algorithm to the training data to build the model. This involves finding patterns and relationships between the features and the class labels.

2. Model Usage

Once the model is constructed, it can be used to classify new, unseen data.

- Objective: Use the model to classify unknown data samples.
- Steps:
    1. Data Input: Provide the model with new data samples that need to be classified.
    2. Classification: The model applies the learned patterns to classify the new data into one of the predetermined classes.
    3. Evaluation: Assess the model's accuracy by comparing its predictions with the actual class labels (if available). This can be done using metrics like accuracy, precision, recall, and F1-score.
    4. Adjustment: If the model's accuracy is not satisfactory, it may be necessary to refine the model. This could involve collecting more training data, selecting different features, or using a different algorithm.

Example

Let's consider an example to illustrate this process:

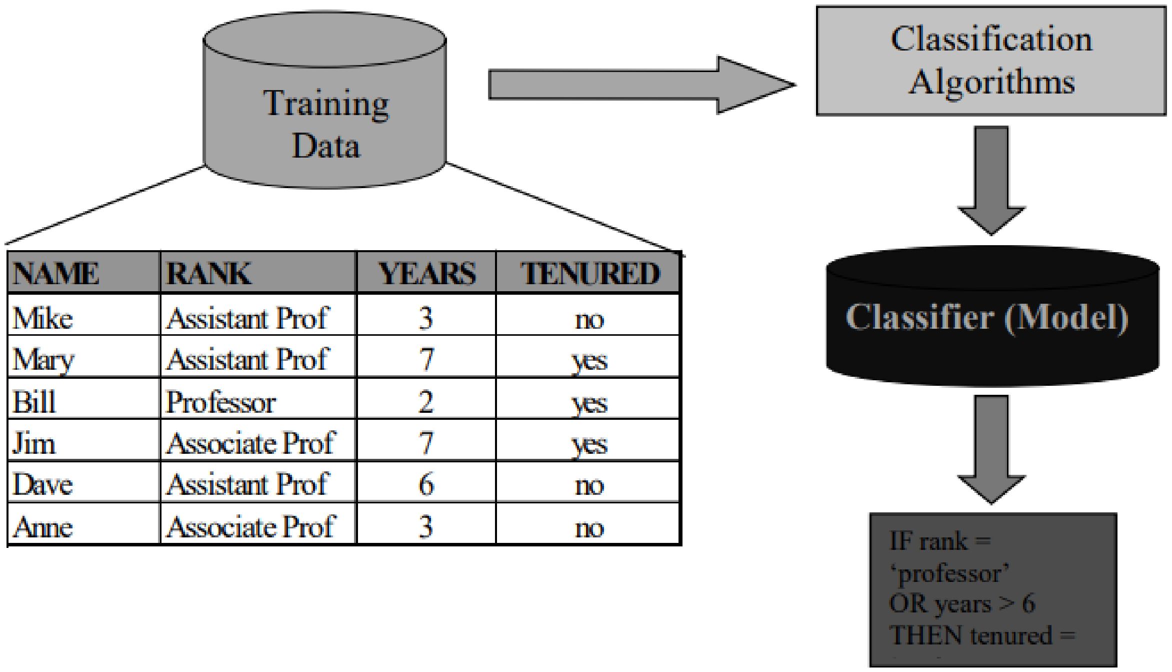
- Model Construction:
    - Training Data: A dataset of emails labelled as "spam" or "not spam".
    - Feature Selection: Identify features such as the presence of certain keywords, email length, and sender's address.
    - Model Building: Use a decision tree algorithm to create a model that classifies emails based on these features as shown in Figure 4.2.
- Model Usage:
    - New Data: An incoming email that needs to be classified .
    - Classification: The model analyzes the email's features and classifies it as either "spam" or "not spam" as shown in Figure 4.3.
    - Evaluation: If the email is later confirmed to be spam or not, this information can be used to further refine the model.

Partitioning and Distance-Based Methods

In the context of clustering (unsupervised learning), two common approaches are:

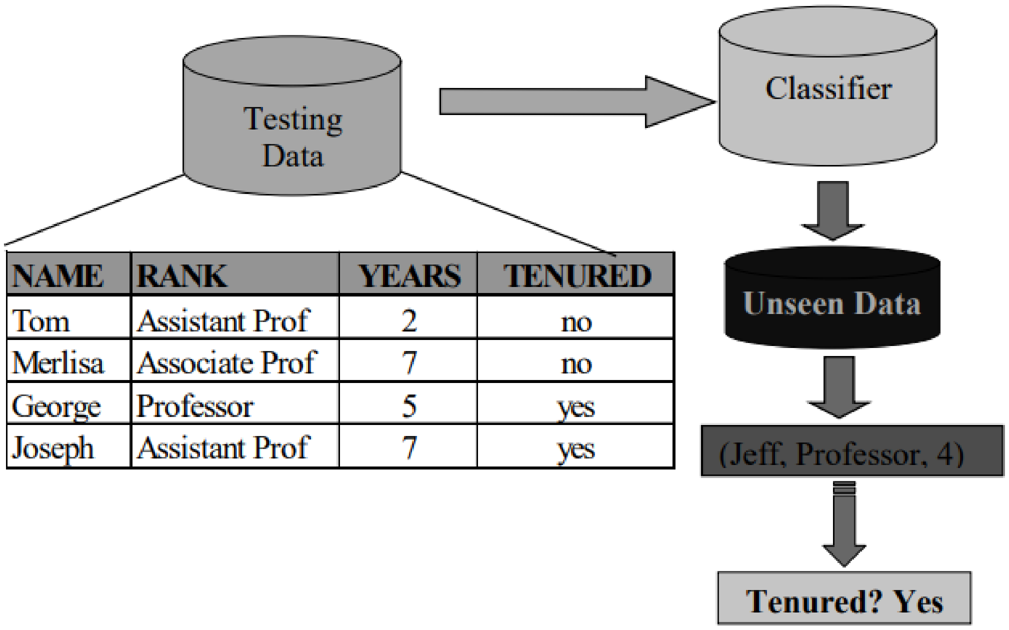
- Partitioning Methods: These methods divide the data into distinct clusters. An example is the k-means algorithm, which partitions the data into k clusters based on the distance between data points and the cluster centroids.

- Distance-Based Methods: These methods use distance metrics (e.g., Euclidean distance) to group similar data points together. Hierarchical clustering is an example, where data points are grouped into a hierarchy of clusters based on their distances.


Training Data
Classification Algorithms
Classifier (Model)
IF rank = 'professor' OR years > 6 THEN tenured =

NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Fig 4.2: Classifier(Model)



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Fig 4.3: Testing the Classifier(Model)

Decision Tree Induction

Example: Training Dataset [Quinlan's ID3]

Table 4.1: Dataset for Decision Tree

age	**income**	**student**	**credit_rating**	**buys_computer**
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

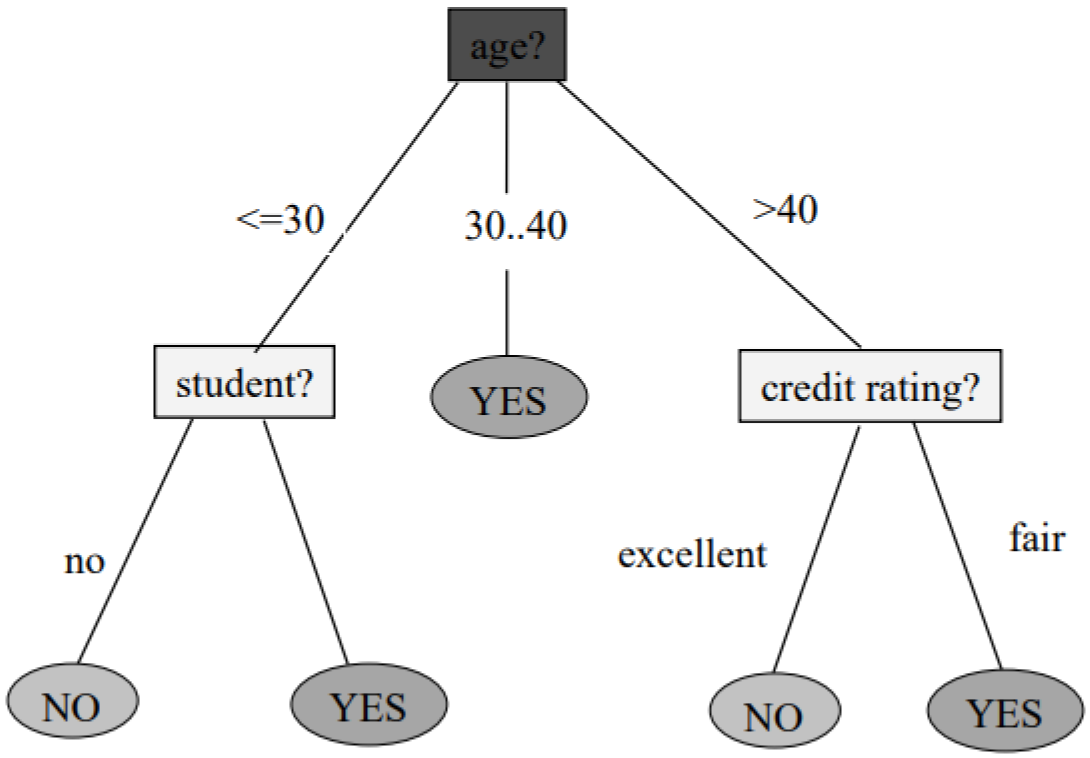


Fig. 4.4: Decision Tree Induction Algorithm with an Example Solved

Basic algorithm (a greedy algorithm)

- Tree is constructed in a top-down recursive divide-and- conquer manner as shown in Figure 4.4
- At start, all the training examples are at the root
- Attributes are categorical (if continuous-valued, they are discretized in advance)
- Samples are partitioned recursively based on selected attributes
- Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

Conditions for stopping partitioning

- All samples for a given node belong to the same class
- There are no remaining attributes for further partitioning - majority voting is employed for classifying the leaf
- There are no samples left

Formulas

1. Information Content Formula:

$$I(s_1, s_2, \ldots, s_m) = -\sum_{i=1}^{m} \frac{s_i}{s} \log_2 \frac{s_i}{s}$$

2. Expected Information after choosing Attribute *A*:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \cdots + s_{mj}}{s} I(s_{1j}, \ldots, s_{mj})$$

3. Information Gain:

$$\text{Gain}(A) = I(s_1, s_2, \ldots, s_m) - E(A)$$

• Information Gain Computation

o Class P:

buys_computer = "yes" ▪ p: number of samples

o Class N:

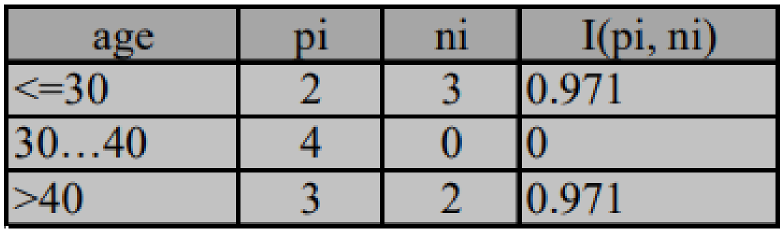
buys_computer = "no"

n: number of samples

The expected information:

I(p, n) = 1(9, 5)=0.940

Compute the entropy for age:



age	pi	ni	I(pi, ni)
<=30	2	3	0.971
30…40	4	0	0
>40	3	2	0.971

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

Hence

Gain(age) = I(p,n)-E(age) = 0.94- 0.694 = 0.246

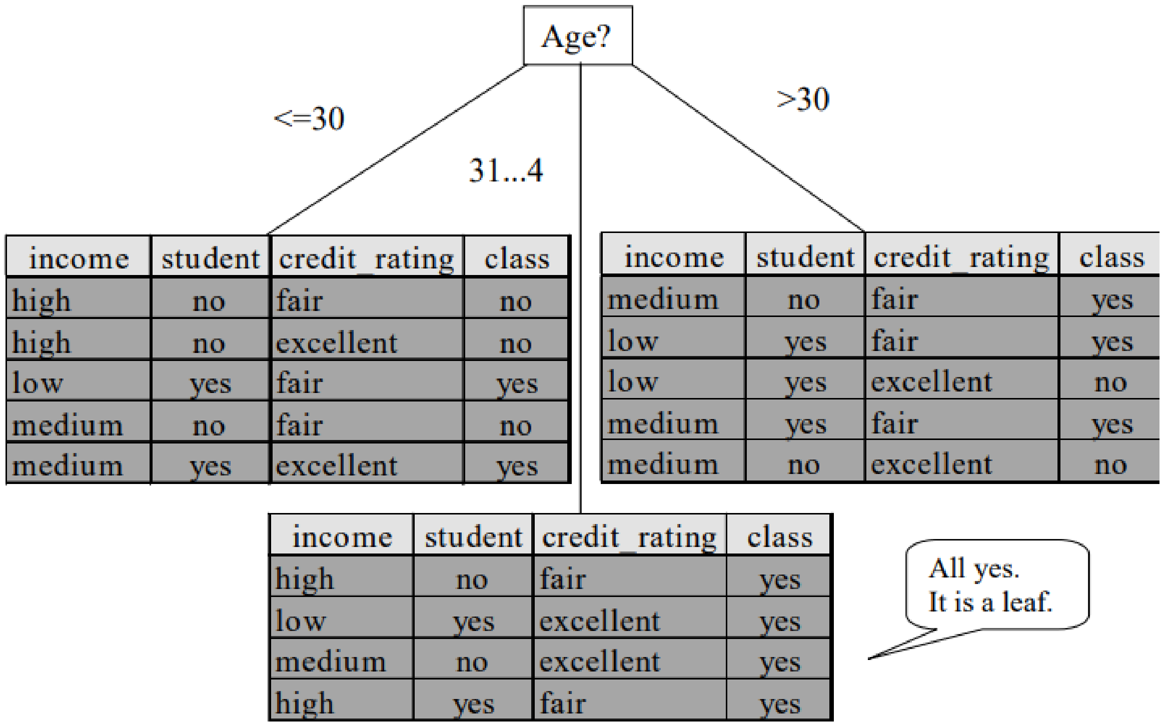
Similarly,

Gain(income) = 0.029 Gain(student) = 0.151

Gain(credit_rating) = 0.048

We say that age is more informative than income, student, and credit_rating.

So age would be chosen as the root of the tree as shown in Figure 4.5



income	student	credit_rating	class
high	no	fair	no
high	no	excellent	no
low	yes	fair	yes
medium	no	fair	no
medium	yes	excellent	yes

income	student	credit_rating	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

income	student	credit_rating	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Fig. 4.5: Decision Tree

ID3 Algorithm

```
function ID3 (R: a set of non - categorical attributes ,
C: the categorical attribute ,
S: a training set)
    returns a decision tree ;
begin
        If S is empty , return a single node with value Failure ;
        If S consists of records all with the same value for the categorical attribute ,
            return a single node with that value ;
        If R is empty , then return a single node with as value
            the most frequent of the values of the categorical attribute
            that are found in records of S; [note that then there
            will be errors , that is , records that will be improperly
            classified];
        Let D be the attribute with largest Gain (D,S)
            among attributes in R;
        Let {dj | j=1 ,2 , .., m} be the values of attribute D;
        Let {Sj | j=1 ,2 , .., m} be the subsets of S consisting
            respectively of records with value dj for attribute D;
        Return a tree with root labeled D and arcs labeled
            d1 , d2 , .., dm going respectively to the trees
            ID3 (R -{D}, C, S1), ID3 (R -{D}, C, S2), .., ID3 (R -{D},
        C, Sm);
end ID3;
```

Other Attribute Selection Measures

Gini Index (CART, IBM IntelligentMiner)

- **Assumptions**:
  - All attributes are assumed to be continuous-valued.
  - There exist several possible split values for each attribute.
  - Other tools, such as clustering, may be needed to determine the possible split values.
  - Can be modified for categorical attributes.
- **Formal Definition**:

If a dataset (T) contains examples from (n) classes, the Gini index, Gini(T), is defined as:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2 \qquad \textbf{(4.4)}$$

where (p_j) is the relative frequency of class (j) in (T).

If a dataset (T) is split into two subsets (T_1) and (T_2) with sizes (N_1) and (N_2) respectively, the Gini index of the split data is defined as:

$$Gini(T) = \sum_{j=1}^{k} \frac{|T_j|}{|T|} \cdot Gini(T_j) \qquad \textbf{(4.5)}$$

where (N) is the total number of samples in (T).

- ○ The attribute that provides the smallest Gini(T) is chosen to split the node. This requires enumerating all possible splitting points for each attribute.

Extracting Classification Rules from Trees

- **Representation**:
    - ○ Represent the knowledge in the form of IF-THEN rules.
    - ○ One rule is created for each path from the root to a leaf.
    - ○ Each attribute-value pair along a path forms a conjunction.
    - ○ The leaf node holds the class prediction.
    - ○ Rules are easier for humans to understand.
- **Example**:
    - ○ IF age = “<=30” AND student = “no” THEN buys_computer = “no”
    - ○ IF age = “<=30” AND student = “yes” THEN buys_computer = “yes”
    - ○ IF age = “31…40” THEN buys_computer = “yes”
    - ○ IF age = “>40” AND credit_rating = “excellent” THEN buys_computer = “yes”
    - ○ IF age = “<=30” AND credit_rating = “fair” THEN buys_computer = “no”

Avoid Overfitting in Classification

- **Overfitting**:

  An induced tree may overfit the training data, leading to:

    - ○ Too many branches, some of which may reflect anomalies due to noise or outliers.
    - ○ Poor accuracy for unseen samples.

- **Approaches to Avoid Overfitting**:
  - **Pre Pruning**: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold.
    - Difficult to choose an appropriate threshold.
  - **Post Pruning**: Remove branches from a “fully grown” tree—get a sequence of progressively pruned trees.
    - Use a set of data different from the training data to decide which is the “best pruned tree”.

Classification in Large Databases

- **Challenges**:
  - Classifying datasets with millions of examples and hundreds of attributes with reasonable speed.
- **Advantages of Decision Tree Induction in Data Mining**:
  - Relatively faster learning speed compared to other classification methods.
  - Convertible to simple and easy-to-understand classification rules.
  - Can use SQL queries for accessing databases.
  - Comparable classification accuracy with other methods.

Bayesian Classification

Why Bayesian?

- Probabilistic learning: Calculate explicit probabilities for hypothesis, among the most practical approaches to certain types of learning problems
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct. Prior knowledge can be combined with observed data.
- Probabilistic prediction: Predict multiple hypotheses, weighted by their probabilities
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Basics

Let X be a data sample whose class label is unknown Let H be a hypothesis that X belongs to class C Posterior Probability:

o For classification problems, determine P(H/X): the probability that the hypothesis H holds given the observed data sample X

Prior Probability:

- P(H): prior probability of hypothesis H

- It is the initial probability before we observe any data
- It reflects the background knowledge

P(X): probability that sample data is observed

P(X/H) probability of observing the sample X, given that the hypothesis H holds

Bayesian Theorem:

o Given training data X, posteriori probability of a hypothesis H, P(H/X) follows the Bayes theorem

$$P(H|X) = \frac{P(X|H) \cdot P(H)}{P(X)} \qquad \textbf{(4.6)}$$

• Informally, this can be written as

POSTERIOR = PRIOR X LIKELIHOOD / OBSERVED **(4.7)**

Maximum Posteriori (MAP) Hypothesis:

Since P(X) is constant of all hypotheses, we have the following:

$$h_{\text{MAP}} = \text{argmax}P(h|D) = \text{argmax}P(D|h) \cdot P(h)\ h \quad h$$

Where D is the data sample, H is the set of all available hypothesis (e.g., all available classes).

Practical difficulty:

- Require initial knowledge of many probabilities
- Significant computational cost

Naive Bayes Classifier

A simplified assumption would be considering all the attributes are independent:

$$P(\text{age} \leq 30|\text{buys computer} = \text{"yes"}) = 0.222$$

The product of occurrence of say 2 elements x1 and x2, given the current class is C, is the product of the probabilities of each element taken separately, given the same class P([y1,y2],C) = P(y1,C) * P(y2,C)

- No dependence relation between attributes

- Greatly reduces the computation cost, only count the class distribution.
- Once the probability P(X|Ci) is known, assign X to the class with maximum P(X|C;)*P(Ci)

Example:

$$P(H_1|D) = \frac{P(D|H_1) \cdot}{P(D)}$$
$$P(H_2|D) = \frac{P(D|H_2) \cdot}{P(D)}$$

1. **Prior Probabilities:**

$$P(H_1) = 0.6$$
$$P(H_2) = 0.4$$

2. 2. **Likelihood:**

$$P(D|H_1) = 0.5$$
$$P(D|H_2) = 0.7$$

3. **Posterior Probabilities:** Using Bayes' Theorem:
4. **Evidence:**

$$P(D) = P(D|H_1) \cdot P(H_1) + P(D|H_2) \cdot P(H_2)$$

5. **Compute Posterior Probabilities:**

$$P(H_1|D) = \frac{0.5 \cdot 0.6}{0.5 \cdot 0.6 + 0.7 \cdot 0.4} = \frac{0.3}{0.58} = 0.5172$$
$$P(H_2|D) = \frac{0.7 \cdot 0.4}{0.5 \cdot 0.6 + 0.7 \cdot 0.4} = \frac{0.28}{0.58} = 0.4828$$

6. **MAP Hypothesis:**

$$h_{\text{MAP}} = \text{argmax} P(H_i|D)$$

$$H_i$$

In this example:

$$P(H_1|D) = 0.5172$$
$$P(H_2|D) = 0.4828$$

Thus, the MAP hypothesis is:

$$h_{\text{MAP}} = H_1$$

Advantages:

- Easy to implement

- Good results obtained in most of the cases

<u>Disadvantages</u>

- Assumption: class conditional independence, therefore loss of accuracy
- Practically, dependencies exist among variables
- E.g., hospitals: patients: Profile: age, family history etc o Symptoms: fever, cough etc., Disease: lung cancer, diabetes etc
- Dependencies among these cannot be modeled by Naïve Bayesian Classifier

<u>How to deal with these dependencies?</u>

Bayesian Belief Networks

• Objectives:

o The naïve Bayesian classifier assume that attributes are conditionally independents

o Belief Nets are PROVEN TECHNOLOGY

- Medical Diagnosis
- DSS for complex machines
- Forecasting, Modeling, Information Retrieval, etc.

Definition

A bayesian network is a causal directed acyclic graph (DAG), associated with an underlying distribution of probability.

It has a DAG structure

Each node is represented by a variable v

v depends (only) on its parents conditional probability:

P(vi/parenti <0,1,...>)

v is INDEPENDENT of non-descendants, given assignments to its parents

- We must specify the conditional probability distribution for each node.
- If the variables are discrete, each node is described by a table (Conditional Probability Table (CPT)) which lists the probability that the child node takes on each of its different values for each combination of values of its parents.

Example

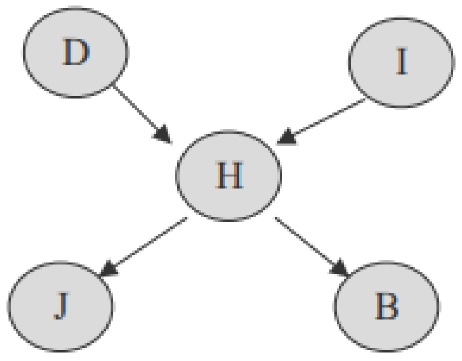


Fig 4.6: Bayesian Network

Consider

P(H=1)	**P(H=0)**
0.05	0.95

h	**P(B=1 | H = h)**	**P(B=0 | H=h)**
1	0.95	0.05
0	0.03	0.97

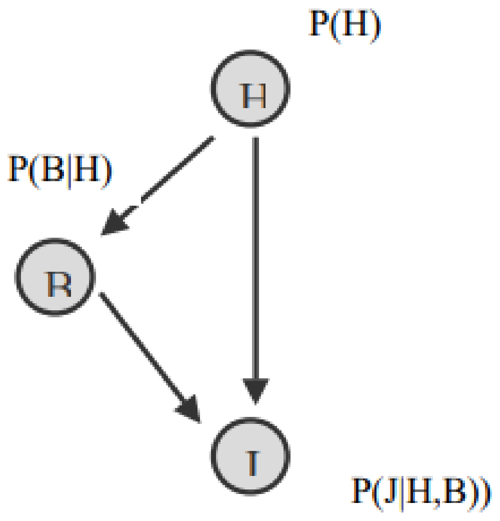


Fig 4.7: DAG

Neural Networks: Classification by Backpropagation

- A neural network is a set of connected input/output units where each connection has a weight associated with it as shown in Figure 4.8.
- During the learning process, the network learns by adjusting the weights to predict the correct class label of the input samples.

Typical NN structure for classification:

One output node per class

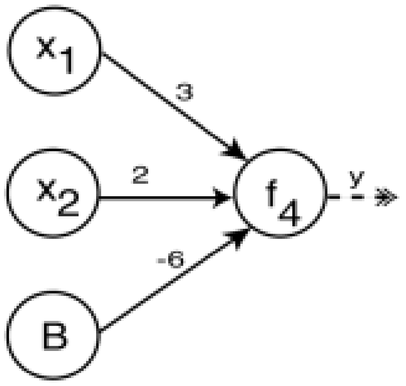
Output value is class membership function value

Perceptron

It is one of the simplest NN

No hidden layers.

• **Algorithms**: Propagation, Backpropagation, Gradient Descent



- Supervised learning
- For each tuple in training set, propagate it through NN, Adjust weights on edges to improve further classification.

Algorithms:

- Propagation
- Backpropagation
- Gradient Descent

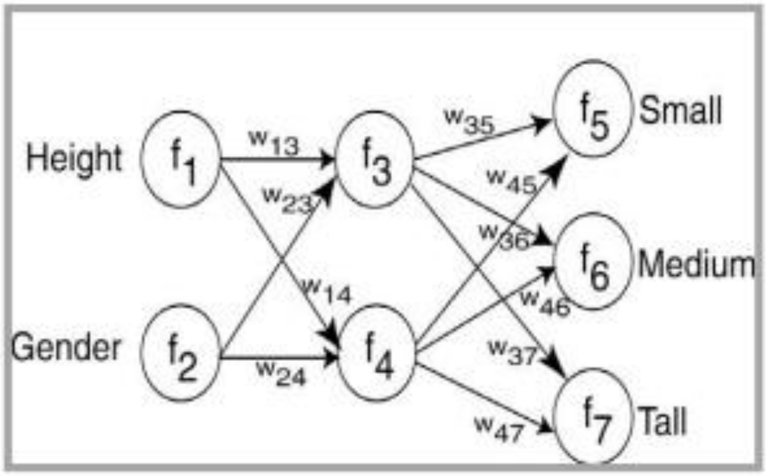


Fig 4.8: A simple Neural Network

Neural network Issues

- Number of source nodes
- Number of hidden layers
- Training data
- Number of sinks
- Interconnections
- Weights
- Activation Functions
- Learning Technique
- When to stop learning

Backpropagation Algorithm

Backpropagation is a powerful algorithm in deep learning, primarily used to train artificial neural networks, particularly feed-forward networks. It works iteratively, minimizing the cost function by adjusting weights and biases.

In each epoch, the model adapts these parameters, reducing loss by following the error gradient. Backpropagation often utilizes optimization algorithms like gradient descent or stochastic gradient descent. The algorithm computes the gradient using the chain rule from calculus, allowing it to effectively navigate complex layers in the neural network to minimize the cost function.

Why is Backpropagation Important?

Backpropagation plays a critical role in how neural networks improve over time.

- Efficient Weight Update: It computes the gradient of the loss function with respect to each weight using the chain rule, making it possible to update weights efficiently.
- Scalability: The backpropagation algorithm scales well to networks with multiple layers and complex architectures, making deep learning feasible.
- Automated Learning: With backpropagation, the learning process becomes automated, and the model can adjust itself to optimize its performance.

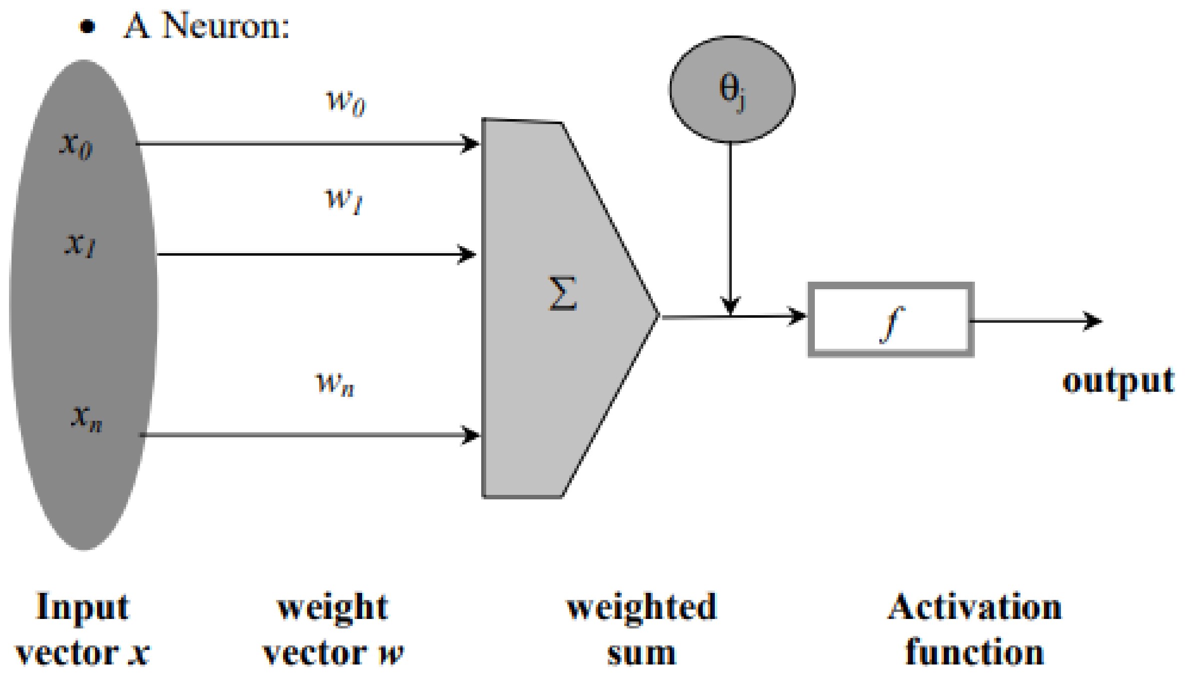


Fig: 4.9: A simple Neuron

It works in two main phases: **Forward Pass** and **Backward Pass**.

1. Forward Pass **(General Explanation)**

The forward pass computes the output of the neural network by propagating inputs through the layers.

- **Steps**:
  1. Input data is passed to the input layer.
  2. Weighted sums are computed at each neuron:

     $$z = W \cdot x + b$$

     Where, W is the weight, x is the input, and b is the bias.

  3. Apply the activation function

     $$y = f(z)$$

     The cumulative sum is given as input to the activation function to produce outputs.

  4. The output from the final layer is compared to the actual output to calculate the **loss** or error.

$$L = 1/2\left(y - y_{\{\{target\}\}}\right)^2$$

2. Backward Pass **(General Explanation)**

The backward pass propagates the error back through the network to update the weights and biases using **gradient descent**.

- **Steps**:
  1. Compute the gradient of the loss with respect to the output layer's weights, biases, and inputs.
  2. Use the chain rule to calculate gradients for hidden layers.
  3. Update weights and biases using **gradient descent**:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W}$$

Where η is the learning rate and $\frac{\partial L}{\partial W}$ is the gradient of the loss

$$\frac{\partial L}{\partial y} = y - y$$

Example (Forward and Backward Pass)

Summation Formulas for Forward and Backward Pass

1. Forward Pass - Summation Example:

$$z_i = \sum_{j=1}^{3} W_{ij} x_j + b_i$$
$$z_i = (W_{i1} \cdot x_1) + (W_{i2} \cdot x_2) + (W_{i3} \cdot x_3) + b_i$$

For example, let:

$$W_{i1} = 2,\ W_{i2} = 3,\ W_{i3} = 1,\ x_1 = 1,\ x_2 = 2,\ x_3 = 1,\ b_i = 1$$

Then:

$$z_i = (2 \cdot 1) + (3 \cdot 2) + (1 \cdot 1) + 1 = 2 + 6 + 1 + 1 = 10$$

2. Loss Function (Summation Form):

$$L \ = \ 1/2\left(y \ - \ y_{\{\{target\}\}}\right)^2$$

For example, let: $y_1 = 3$, $y_2 = 4$, $y_3 = 5$, $y_{\text{target1}} = 2$, $y_{\text{target2}} = 4$, $y_{\text{target3}} = 6$

Then:

$$L = \frac{1}{2}\left[(3-2)^2 + (4-4)^2 + (5-6)^2\right] = \frac{1}{2}\left[1^2 + 0^2 + (-1)^2\right] = \frac{1}{2}(1+0+1) = 1$$

3. Weight Update Rule:

$$W_{ij}^{\text{new}} = W_{ij}^{\text{old}} - \eta \frac{\partial L}{\partial W_{ij}}$$

For example, let:

$$W_{ij}^{\text{old}} = 0.5,\ \eta = 0.01,\ \frac{\partial L}{\partial W_{ij}} = 2$$

Then:

$$W_{ij}^{\text{new}} = 0.5 - (0.01 \cdot 2) = 0.5 - 0.02 = 0.48$$

Support Vector Machines

This set of notes presents the Support Vector Machine (SVM) learning algorithm. SVMs are among the best (and many believe is indeed the best) “off-the-shelf” supervised learning algorithm. To tell the SVM story, we’ll need to first talk about margins and the idea of separating data with a large “gap.” Next, we’ll talk about the optimal margin classifier, which will lead us into a digression on Lagrange duality. We’ll also see kernels, which give a way to apply SVMs efficiently in very high dimensional (such as infinite dimensional) feature spaces, and finally, we’ll close off the story with the SMO algorithm, which gives an efficient implementation of SVMs.

Notation:

To make our discussion of SVMs easier, we'll first need to introduce a new notation for talking about classification. We will be considering a linear classifier for a binary classification problem with labels y and features z. From now, we'll use y {-1,1} (instead of {0, 1}) to denote the class labels. Also, rather than parameterizing our linear classifier with the vector 0, we will use parameters w, b, and write our classifier as

$$h_{\{w,b\}(x)} = g(w^T x + b) \qquad \textbf{(4.17)}$$

Here, g(z) = 1 if z ≥ 0, and g(2) = -1 otherwise. This "w,b" notation allows us to explicitly treat the intercept term b separately from the other parameters. (We also drop the convention we had previously of letting to = 1 be an extra coordinate in the input feature vector.) Thus, b takes the role of what was previously b_0, and w takes the role of set of parameters.

Note also that, from our definition of g above, our classifier will directly predict either 1 or -1 (cf. the perceptron algorithm), without first going through the intermediate step of estimating the probability of y being 1 (which was what logistic regression did).

Functional and Geometric margins

Let’s formalise the notions of the functional and geometric margins. Given a training example (x, y), we define the functional margin of (w, b) with respect to the training example

$$\hat{y} = y^{(i)}(w^T x + b) \qquad \textbf{(4.18)}$$

Note that if y = 1, then for the functional margin to be large (ie., for our prediction to be confident and correct), then we need $w^T x+b$ to be a large positive number. Conversely, if y=-1, then for the functional margin to be large, then we need $w^T x+b$ to be a large negative number. Moreover, if $y(w^T x+b) > 0$, then our prediction on this example is correct. (Check this yourself.) Hence, a large functional margin represents a confident and a correct prediction.

For a linear classifier with the choice of g given above (taking values in {-1,1}), there's one property of the functional margin that makes it not a very good measure of confidence, however.

Given our choice of g, we note that if we replace w with 2w and b with 2b, then since $g(w^Tx+b) = g(2w^Tx+2b)$,

this would not change hw,b(x) at all. I.e., g, and hence also h(x), depends only on the sign, but not on the magnitude, of wr+b. However, replacing (w, b) with (2w, 2b) also results in multiplying our functional margin by a factor of 2. Thus, it seems that by exploiting our freedom to scale w and b, we can make the functional margin arbitrarily large without really changing anything meaningful. Intuitively, it might therefore make sense to impose some sort of normalisation condition such as that $||w||^2 = 1$; i.e., we might replace (w, b) with $(w/||wl|^2, b/||w||^2)$, and instead consider the functional margin of $(w/|lw||^2, b/|lw||^2)$. We'll come back to this later.

Given a training set S = {(x,y); i = 1,..., m}, we also define the function margin of (w, b) with respect to S as the smallest of the functional margins of the individual training examples. Denoted by, this can therefore be written:

$$\hat{y} = min\, y^{(i)} \quad \textbf{(4.19)}$$

Let's talk about Geometric margins

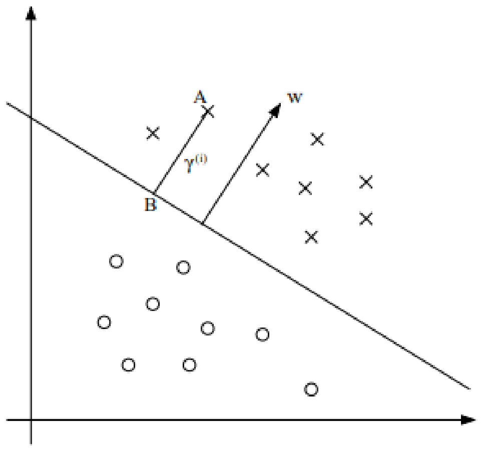


Fig: 4.10: Hyperplane separating two classes

The decision boundary corresponding to (w, b) is shown, along with the vector w. Note that w is orthogonal (at 90°) to the separating hyperplane. (You should convince yourself that this must be the case.) Consider the point at A, which represents the input r) of some training example with label y) = 1. Its distance to the decision boundary, y), is given by the line segment AB.

We find that the point B is given by(xi - ϒi) * w/||wl|. But this point lies on the decision boundary, and all points on the decision boundary satisfy the equation $w^Tx+b=0$. Hence,

$$w^T(x^{(i)} - \gamma^{(i)}(\frac{w}{||w||}) + b = 0$$

Solving for ϒi,

$$\hat{\gamma}_i = \frac{y_i(w^T x_i + b)}{||w||} \qquad \textbf{(4.21)}$$

This was worked out for the case of a positive training example at A in the figure, where being on the "positive" side of the decision boundary is good. More generally, we define the geometric margin of (w, b) with respect to a training example (xi, yi) to be

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{\mathbf{w}}{||\mathbf{w}||} \right)^T \mathbf{x}^{(i)} + \frac{b}{||\mathbf{w}||} \right) \qquad \textbf{(4.22)}$$

Note that if ||w|| = 1, then the functional margin equals the geometric margin-this thus gives us a way of relating these two different notions of margin. Also, the geometric margin is invariant to rescaling of the parame- ters; i.e., if we replace w with 2w and b with 2b, then the geometric margin does not change. This will in fact come in handy later. Specifically, because of this invariance to the scaling of the parameters, when trying to fit w and b to training data, we can impose an arbitrary scaling constraint on w without changing anything important; for instance, we can demand that ||w|| = 1, or w1 = 5, or |w1+b|+|w2|= 2, and any of these can be satisfied simply by rescaling w and b.

Finally, given a training set S = {(x, y); i = 1,...,m}, we also define the geometric margin of (w, b) with respect to S to be the smallest of the geometric margins on the individual training examples:

$$\gamma = min\, \gamma i \qquad \textbf{(4.33)}$$

The optimal margin classifier

Given a training set, it seems from our previous discussion that a natural desideratum is to try to find a decision boundary that maximises the (geometric) margin, since this would reflect a very confident set of predictions on the training set and a good “fit” to the training data. Specifically, this will result in a classifier that separates the positive and the negative training examples with a “gap” (geometric margin). For now, we will assume that we are given a training set that is linearly separable; i.e., that it is possible to separate the positive and negative examples using some separating hyperplane. How do we find the one that achieves the maximum geometric margin? We can pose the following optimization problem:

$$\max_{\boldsymbol{w},b} \gamma$$

subject to

$$y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \ ||w|| = 1$$

I.e., we want to maximise ϒ, subject to each training example having functional margin at least ϒ. The ||w|| = 1 constraint moreover ensures that the functional margin equals to the geometric margin, so we are also guaranteed that all the geometric margins are at least ϒ. Thus, solving this problem will result in (w,b) with the largest possible geometric margin with respect to the training set.

If we could solve the optimization problem above, we'd be done. But the "||w|||= 1" constraint is a nasty (non-convex) one, and this problem certainly isn't in any format that we can plug into standard optimization software to solve. So, lets try transforming the problem into a nicer one.

Since the geometric and functional margins are related by y= ϒ/w, this will give us the answer we want. Moreover, we've gotten rid of the constraint ||w||= 1 that we didn't like. The downside is that we now have a nasty (again, non-convex) objective function; and, we still don't have any off-the-shelf software that can solve this form of an optimization problem.

Lets keep going. Recall our earlier discussion that we can add an arbitrary scaling constraint on w and b without changing anything. This is the key idea we'll use now. We will introduce the scaling constraint that the functional margin of w,b with respect to the training set must be 1

$$\Upsilon = 1$$

Since multiplying w and b by some constant results in the functional margin being multiplied by that same constant, this is indeed a scaling constraint, and can be satisfied by rescaling w, b. Plugging this into our problem above, and noting that maximising ϒ/||w|| = 1/||w|| is the same thing as minimising ||wl|2, we now have the following optimization problem:

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2$$

Subject to the condition, $y^{(i)}(w^T x^{(i)} + b) \geq 1$

We've now transformed the problem into a form that can be efficiently solved. The above is an optimization problem with a convex quadratic objective and only linear constraints. Its solution gives us the optimal mar- gin classifier. This optimization problem can be solved using commercial quadratic programming (QP) code.

While we could call the problem solved here, what we will instead do is make a digression to talk about Lagrange duality. This will lead us to our optimization problem's dual form, which will play a key role in allowing us to use kernels to get optimal margin classifiers to work efficiently in very high dimensional spaces. The dual form will also allow us to derive an efficient algorithm for solving the above optimization problem that will typically do much better than generic QP software.

Kernels

Back in our discussion of linear regression, we had a problem in which the input x was the living area of a house, and we considered performing regres-sion using the features x, x^2 and x^3 (say) to obtain a cubic function. To distinguish between these two sets of variables, we'll call the "original" input value the input attributes of a problem (in this case, x, the living area). When that is mapped to some new set of quantities that are then passed to the learning algorithm, we'll call those new quantities the input features. (Unfortunately, different authors use different terms to describe these two things, but we'll try to use this terminology consistently in these notes.) We will also let φ denote the feature mapping, which maps from the attributes to the features. For instance, in our example, we had

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

Rather than applying SVMs using the original input attributes, we may instead want to learn using some features Φ(x) . To do so, we simply need to go over our previous algorithm, and replace x everywhere in it with Φ(x).

Since the algorithm can be written entirely in terms of the inner products (x, z), this means that we would replace all those inner products with (Φ(x), Φ(z)). Specifically, given a feature mapping Φ, we define the corresponding Kernel to be

$$K(x, z) = \phi(x)^T\phi(z). \tag{4.20}$$

Then, everywhere we previously had (x, z) in our algorithm, we could simply replace it with K(r, 2), and our algorithm would now be learning using the features Φ.

Now, given Φ, we could easily compute K(x, z) by finding Φ(z) and Φ(z) and taking their inner product. But what's more interesting is that often, K(x,z) may be very inexpensive to calculate, even though Φ(x) itself may be very expensive to calculate (perhaps because it is an extremely high dimensional vector). In such settings, by using in our algorithm an efficient way to calculate

K(x, z), we can get SVMs to learn in the high dimensional feature space given by Φ, but without ever having to explicitly find or represent vectors Φ(x).

Lets see an example. Suppose z, z ЄR", and consider

$$K(x,z) = (x^T z)^2. \tag{4.21}$$

We can also write this as:

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

Thus, we see that K(x, z) =Φ(x)Φ(z), where the feature mapping Φ is given

(shown here for the case of n = 3) by

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}.$$

Note that whereas calculating the high-dimensional Φ(x) requires O(n^2) time, finding K(x, z) takes only O(n) time linear in the dimension of the input attributes.

For a related kernel, also consider

$$\begin{aligned} K(x,z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2. \end{aligned}$$

Lazy Learning

Lazy learning, also known as instance-based learning, is a type of machine learning where the system delays the generalisation of the training data until it receives a query. This contrasts with eager learning, where the system generalises the training data before receiving any queries.

How Lazy Learning Works

In lazy learning, the algorithm stores the training data and waits until it needs to make a prediction. When a new query is received, the algorithm retrieves similar instances from the training set and uses them to generate a prediction. The similarity between instances is usually calculated using distance metrics, such as Euclidean distance or cosine similarity.

Key Characteristics

1. Instance-Based: Lazy learning algorithms memorise the training data rather than constructing a general model.
2. Local Approximation: The target function is approximated locally for each query, which allows the system to adapt to changes in the problem domain.
3. No Training Phase: There is no explicit training phase; the system processes data only when a prediction is needed.

Advantages

1. Flexibility: Since the model is built at query time, lazy learning can adapt to new data without retraining.
2. Simplicity: These algorithms are often simpler to implement compared to eager learning algorithms.
3. Handling Complex Data: Lazy learning methods excel in situations where the underlying data distribution is complex or where the training data is noisy.

Disadvantages

1. Computational Cost: Lazy learning can be computationally expensive during prediction since it requires searching through the training data to find nearest neighbours.
2. Storage Requirements: Storing the entire training dataset can require significant memory, especially for large datasets.
3. Noisy Data: Particularly noisy training data can increase the case base unnecessarily, as no abstraction is made during the training phase.

Popular Algorithms

1. k-Nearest Neighbors (k-NN): One of the most popular lazy learning algorithms. It considers the k closest training instances to the query point and uses their class labels to determine the class of the query.

2. Locally Weighted Learning: This method assigns weights to the training instances based on their distance to the query point and uses these weights to make predictions.

Real-World Applications

1. Recommendation Systems: Used in platforms like Amazon and Netflix to provide personalised recommendations by comparing user preferences to similar users in the training set.
2. Medical Diagnosis: Helps in diagnosing diseases by comparing patient symptoms and medical histories to similar cases in the training data.
3. Anomaly Detection: Useful for detecting anomalies or outliers in datasets, such as credit card fraud detection.

Lazy Learning vs. Eager Learning

- Training Phase: Eager learning algorithms construct a general model during the training phase, while lazy learning algorithms defer model construction until prediction time.
- Computational Cost: Eager learning algorithms typically have faster prediction times once the model is trained, whereas lazy learning algorithms can be computationally expensive during prediction.

Prediction

• Prediction (often called regression) is similar to classification

- First, construct a model
- Second, use model to predict unknown value

Major method for prediction is regression

- Linear and multiple regression
- Non-linear regression

Prediction is different from classification

- Classification refers to predict categorical class label o Prediction models continuous-valued functions • Many classification methods can also be used for regressions
- Decision trees can also produce probabilities o Bayesian networks: probability
- Neural networks: continuous output

Regress Analysis and Log-Linear Models in Prediction

Linear regression: Y = a+bX

o Two parameters, a and b specify the line and are to be estimated by using the data at hand.

o Using the least squares criterion to the known values of Y1, Y2, ..., X1, X2,....

• Multiple regression: Y=b0+b1 X1 + b2 X2.

o Many nonlinear functions can be transformed into the above.

Non-linear models:

o Can always be converted to a linear model

Classification Accuracy: Estimating Error Rates

Partition: Training-and-testing

o Use two independent data sets, e.g., training set (2/3), test set(1/3)

o Used for data set with large number of samples

Cross-validation

o Divide the data set into k subsamples

o Use k-1 subsamples as training data and one sub-sample

as test data-k-fold cross-validation

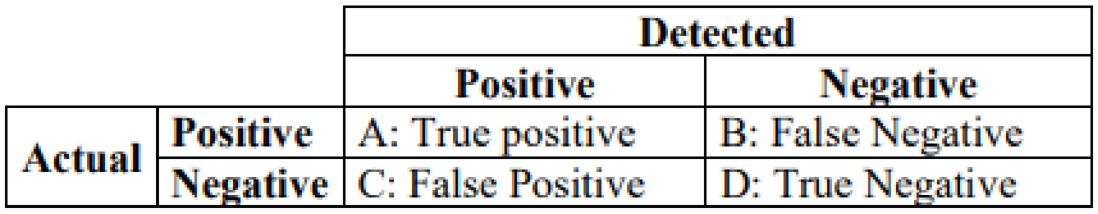
o For data set with moderate size

Bootstrapping (leave-one-out) o For small size data

Confusion Matrix:

o This matrix shows not only how well the classifier predicts different classes

o It describes information about actual and detected classes:



		Detected	
		Positive	**Negative**
Actual	**Positive**	A: True positive	B: False Negative
	Negative	C: False Positive	D: True Negative

The recall (or the true positive rate) and the precision (or the positive predictive rate) can be derived from the confusion matrix as follows:

Recall = A/(A+B)

Precision = A/(A+C)

Ensemble methods

Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning.

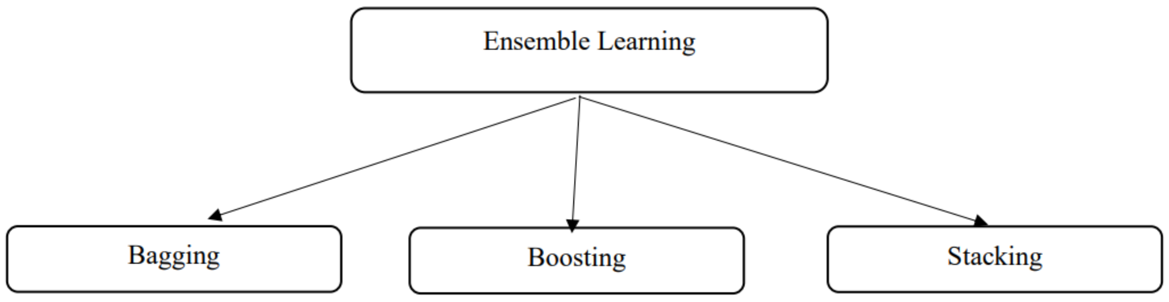


Fig 4.11: Types of Ensemble Methods

Categories of Ensemble Methods

Ensemble methods fall into two broad categories, i.e., sequential ensemble techniques and parallel ensemble techniques. Sequential ensemble techniques generate base learners in a sequence, e.g., Adaptive Boosting (AdaBoost). The sequential generation of base learners promotes the dependence between the base learners. The performance of the model is then improved by assigning higher weights to previously misrepresented learners.

In parallel ensemble techniques, base learners are generated in a parallel format, e.g. Random forest. Parallel methods utilise the parallel generation of base learners to encourage independence between the base learners. The independence of base learners significantly reduces the error due to the application of averages.

The majority of ensemble techniques apply a single algorithm in base learning, which results in homogeneity in all base learners. Homogenous base learners refer to base learners of the same type, with similar qualities. Other methods apply to heterogeneous base learners, giving rise to heterogeneous ensembles. Heterogeneous base learners are learners of distinct types.

Main Types of Ensemble Methods

1. Bagging

Bagging, the short form for bootstrap aggregating, is mainly applied in classification and regression. It increases the accuracy of models through decision trees, which reduces variance to a large extent. The reduction of variance increases accuracy, eliminating overfitting, which is a challenge to many predictive models.

Bagging is classified into two types, i.e., bootstrapping and aggregation. Bootstrapping is a sampling technique where samples are derived from the whole population (set) using the replacement procedure. The sampling with replacement method helps make the selection procedure randomised. The base learning algorithm is run on the samples to complete the procedure.

Aggregation in bagging is done to incorporate all possible outcomes of the prediction and randomise the outcome. Without aggregation, predictions will not be accurate because all outcomes are not put into consideration. Therefore, the aggregation is based on the probability bootstrapping procedures or on the basis of all outcomes of the predictive models.

Bagging is advantageous since weak base learners are combined to form a single strong learner that is more stable than single learners. It also eliminates any variance, thereby reducing the overfitting of models. One limitation of bagging is that it is computationally expensive. Thus, it can lead to more bias in models when the proper procedure of bagging is ignored.

2. Boosting

Boosting is an ensemble technique that learns from previous predictor mistakes to make better predictions in the future. The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.

Boosting takes many forms, including gradient boosting, Adaptive Boosting (AdaBoost), and XGBoost (Extreme Gradient Boosting). AdaBoost uses weak learners in the form of decision trees, which mostly include one split that is popularly known as decision stumps. AdaBoost's main decision stump comprises observations carrying similar weights.

Gradient Boosting adds predictors sequentially to the ensemble, where preceding predictors correct their successors, thereby increasing the model's accuracy. New predictors are fit to counter the effects of errors in the previous predictors. The gradient of descent helps the gradient booster identify problems in learners' predictions and counter them accordingly.

XGBoost makes use of decision trees with boosted gradients, providing improved speed and performance. It relies heavily on the computational speed and the performance of the target model. Model training should follow a sequence, thus making the implementation of gradient boosted machines slow.

3. Stacking

Stacking, another ensemble method, is often referred to as stacked generalisation. This technique works by allowing a training algorithm to ensemble several other similar learning algorithm predictions. Stacking has been successfully implemented in regression, density estimations, distance learning, and classifications. It can also be used to measure the error rate involved during bagging.

Variance Reduction

Ensemble methods are ideal for reducing the variance in models, thereby increasing the accuracy of predictions. The variance is eliminated when multiple models are combined to form a single prediction that is chosen from all other possible predictions from the combined models. An ensemble of models combines various models to ensure that the resulting prediction is the best possible, based on the consideration of all predictions.

Why do ensembles work?

Dietterich(2002) showed that ensembles overcome three problems –

- Statistical Problem –

The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data!

- Computational Problem –

The Computational Problem arises when the learning algorithm cannot guarantee finding the best hypothesis.

- Representational Problem –

The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es).

Main Challenge for Developing Ensemble Models?

The main challenge is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. For example, if ensembles are used for classification, high accuracies can be accomplished if different base models misclassify different training examples, even if the base classifier accuracy is low.

UNIT EXERCISE

2 Marks

1. Name the types of classification.
2. What are the conditions for stopping partitioning in the ID3 Decision Tree algorithm?
3. Define Gini Index and write its formula.
4. What is the difference between Bayes and Naive Bayes classifiers?
5. Illustrate how you do backpropagation for a single hidden layer network with two hidden neurons and 1 input neuron and 1 output neuron
6. What is the main challenge of developing the Ensemble methods?
7. Differentiate Baaging, Boosting and Stacking.
8. Give the equation of a linear model and explain.
9. Define the kernel and write the equation of any one of the kernels.
10. What is the difference between Functional and Geometric margin?

8 Marks

11. Explain how do you construct the model and test the model for the following application: "Sentiment Analysis"
12. What do you understand from the optimal margin classifier in SVM?
13. Explain the working of Lazy learning and explain any one of the algorithms that uses this type of learning

Unit 5 - Cluster Analysis

Cluster Analysis:

- The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering.
- 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 and so may be considered as a form of data compression.
- Cluster analysis tools based on k-means, k-medoids, and several methods have also been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS.

Applications:

- Cluster analysis has been widely used in numerous applications, including market research, pattern recognition, data analysis, and image processing.
- In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns.
- In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations.
- Clustering may also help in the identification of areas of similar land use in an earth observation database and in the identification of groups of houses in a city according to house type, value,and geographic location, as well as the identification of groups of automobile insurance policy holders with a high average claim cost.
- Clustering is also called data segmentation in some applications because clustering partitions large data sets into groups according to their *similarity*.
- Clustering can also be used for outlier detection,Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

Typical Requirements Of Clustering In Data Mining:

- **Scalability:**

  Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a sample of a given large data set may lead to biased results.

  Highly scalable clustering algorithms are needed.

- **Ability to deal with different types of attributes:**

  Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical

  (nominal), and ordinal data, or mixtures of these data types.

- **Discovery of clusters with arbitrary shape:**

  Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density.

  However, a cluster could be of any shape. It is important to develop algorithms thatcan detect clusters of arbitrary shape.

- **Minimal requirements for domain knowledge to determine input parameters:**

  Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but it also makes the quality of clustering difficult to control.

- **Ability to deal with noisy data:**

  Most real-world databases contain outliers or missing, unknown, or erroneous data.

  Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

- **Incremental clustering and insensitivity to the order of input records:**

  Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data.

  That is, given a set of data objects, such an algorithm may return dramatically different clusterings depending on the order of presentation of the input objects.

  It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.

- **High dimensionality:**

A database or a data warehouse can contain several dimensions or attributes.Many clustering algorithms are good at handling low-dimensional data,involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

- **Constraint-based clustering:**

  Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automated banking machines (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks, and the type and number of customers per cluster. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

- **Interpretability and usability:**

  Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied to specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and methods.

Major Clustering Methods:

- Partitioning Methods
- Hierarchical Methods
- Density-Based Methods
- Grid-Based Methods
- Model-Based Methods

Partitioning Methods:

A partitioning method constructs *k* partitions of the data, where each partition represents a cluster and *k* <= *n*. That is, it classifies the data into *k* groups, which together satisfy the following requirements:

- Each group must contain at least one object, and
- Each object must belong to exactly one group.

A partitioning method creates an initial partitioning. It then uses an iterative relocation technique that 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.

Hierarchical Methods:

A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed.

- The agglomerative approach, also called the bottom-up approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one or until a termination condition holds.
- The divisive approach, also called the top-down approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.

Hierarchical methods suffer from the fact that once a step (merge or split) is done,it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices.

There are two approaches to improving the quality of hierarchical clustering:

- Perform careful analysis of object —linkages at each hierarchical partitioning, such as in Chameleon, or
- Integrate hierarchical agglomeration and other approaches by first using a hierarchical agglomerative algorithm to group objects into microclusters, and then performing macro clustering on the microclusters using another clustering method such as iterative relocation.

Density-based methods:

- Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes.
- Other clustering methods have been developed based on the notion of density. Their general idea is to continue growing the given cluster as long as the density in the neighborhood

exceeds some threshold; that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise (outliers)and discover clusters of arbitrary shape.

- ❖ DBSCAN and its extension, OPTICS, are typical density-based methods that grow clusters according to a density-based connectivity analysis. DENCLUE is a method that clusters objects based on the analysis of the value distributions of density functions.

Grid-Based Methods:

- ❖ Grid-based methods quantize the object space into a finite number of cells that form a grid structure.
- ❖ All of the clustering operations are performed on the grid structure i.e., on the quantized space. The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.
- ❖ STING is a typical example of a grid-based method. Wave Cluster applies wavelet transformation for clustering analysis and is both grid-based and density-based.

Model-Based Methods:

- ❖ Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model.
- ❖ A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points.
- ❖ It also leads to a way of automatically determining the number of clusters based on standard statistics, taking —noise‖ or outliers into account and thus yielding robust clustering methods.

Tasks in Data Mining:

- ➢ Clustering High-Dimensional Data
- ➢ Constraint-Based Clustering

Clustering High-Dimensional Data:

- It is a particularly important task in cluster analysis because many applications require the analysis of objects containing a large number of features or dimensions. For example, text documents may contain thousands of terms or keywords as features, and DNA micro array data may provide information on the expression levels of thousands of genes under hundreds of conditions.

- Clustering high-dimensional data is challenging due to the curse of dimensionality.
- Many dimensions may not be relevant. As the number of dimensions increases, the data become increasingly sparse so that the distance measurement between pairs of points become meaningless and the average density of points anywhere in the data is likely to be low. Therefore, a different clustering methodology needs to be developed for high-dimensional data.
- CLIQUE and PROCLUS are two influential subspace clustering methods, which search for clusters in subspaces of the data, rather than over the entire data space.
- Frequent pattern–based clustering,another clustering methodology, extract distinct frequent patterns among subsets of dimensions that occur frequently. It uses such patterns to group objects and generate meaningful clusters.

Constraint-Based Clustering:

- It is a clustering approach that performs clustering by incorporation of user-specified or application-oriented constraints.
- A constraint expresses a user's expectation or describes properties of the desired clustering results, and provides an effective means for communicating with the clustering process.
- Various kinds of constraints can be specified, either by a user or as per application requirements.
- Spatial clustering employs the existence of obstacles and clustering under user specified constraints. In addition, semi-supervised clustering employs pairwise constraints in order to improve the quality of the resulting clustering.

Classical Partitioning Methods:

The most well-known and commonly used partitioning methods are

❖ The *k*-Means Method

❖ k-Medoids Method

Centroid-Based Technique: The *K*-Means Method:

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the inter cluster similarity is low.

Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's centroid or center of gravity.

The *k*-means algorithm proceeds as follows.

- First, it randomly selects *k* of the objects, each of which initially represents a cluster mean or center.
- 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.
- It then computes the new mean for each cluster.
- This process iterates until the criterion function converges.

Typically, the square-error criterion is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2, \quad \textbf{(5.1)}$$

whereE is the sum of the square error for all objects in the data set

pis the point in space representing a given object $m_i$is the mean of

cluster C_i

The k-means partitioning algorithm:

The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster as in figure 5.1.

Input:

- k: the number of clusters,
- D: a data set containing n objects.

Output: A set of k clusters.

Method:

```
(1) arbitrarily choose $k$ objects from $D$ as the initial cluster centers;
(2) repeat
(3)       (re)assign each object to the cluster to which the object is the most similar,
                based on the mean value of the objects in the cluster;
(4)       update the cluster means, i.e., calculate the mean value of the objects for
                each cluster;
(5) until no change;
```

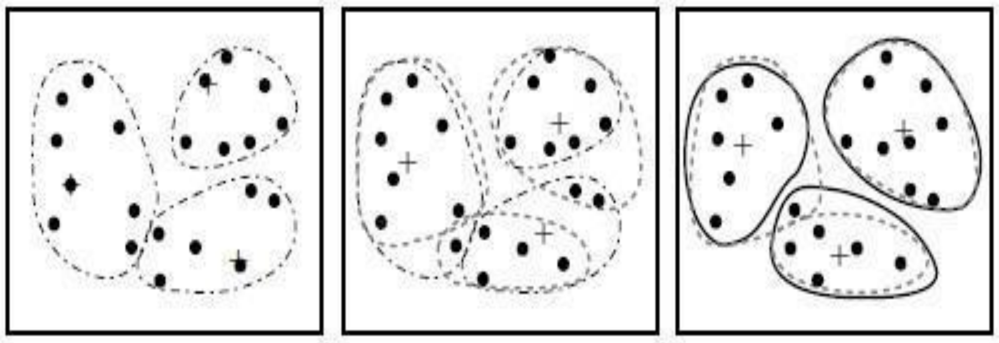


Fig 5.1: Clustering of a set of objects based on the k-means method

The *k*-Medoids Method:

- The k-means algorithm is sensitive to outliers because an object with an extremely large value may substantially distort the distribution of data. This effect is particularly exacerbated due to the use of the square-error function.
- Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is clustered with the representative object to which it is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point. That is, an absolute-error criterion is used, defined as

$$E = \sum_{j=1}^{k} \sum_{p \in C_j} |p - o_j|,$$

(5.2)

where*E* is the sum of the absolute error for all objects in the data set ***p***

is the point in space representing a given object in clusterC_j $\mathbf{o_j}$is the

representative object of C_j

- The initial representative objects are chosen arbitrarily. The iterative process of replacing representative objects by non representative objects continues as long as the quality of the resulting clustering is improved.
- This quality is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster.
- To determine whether a non representative object, oj random, is a good replacement for a current representative object, oj, the following four cases are examined for each of the nonrepresentative objects as in Fig 2.

Case 1:

p currently belongs to representative object, o_j. If $o_j$is replaced by o_{random}asa representative object and p is closest to one of the other representative objects, o_i,$i \neq j$, then p is reassigned to o_i.

Case 2:

p currently belongs to a representative object, oj. If o_j is replaced by o_{random} as a representative object and p is closest to o_{random}, then p is reassigned to o_{random}.

Case 3:

p currently belongs to a representative object, o_i, $i \neq j$. If o_j is replaced by o_{random} as a representative object and p is still closest to o_i, then the assignment does not change.

Case 4:

p currently belongs to a representative object, oi, $i \neq j$. If o_j is replaced by o_{random} as a representative object and p is closest to o_{random}, then p is reassigned to o_{random}.

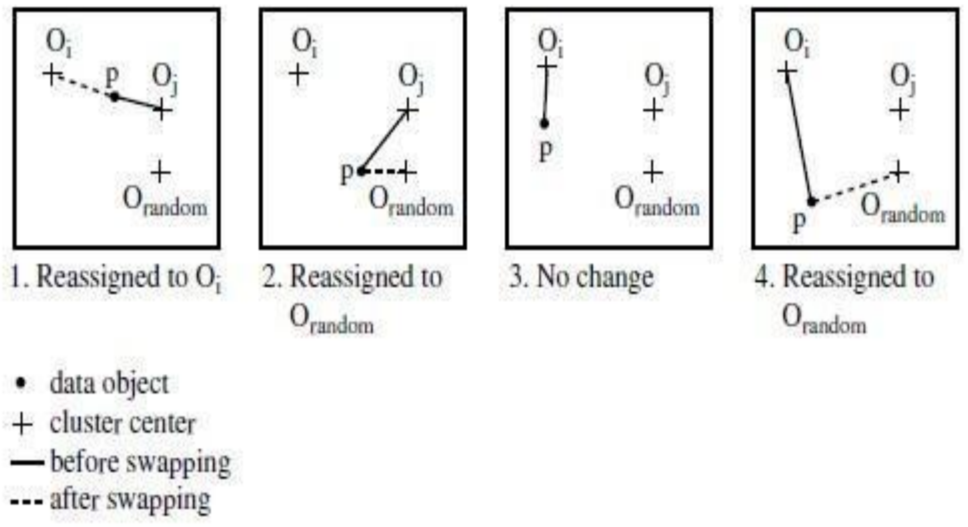


Fig.5.2: Four cases of the cost function for k-medoids clustering

The *k*-MedoidsAlgorithm:

The k-medoids algorithm for partitioning based on medoid or central objects.

Input:

- k: the number of clusters,
- D: a data set containing n objects.

Output: A set of k clusters.

Method:

```
(1) arbitrarily choose $k$ objects in $D$ as the initial representative objects or seeds;
(2) repeat
(3)     assign each remaining object to the cluster with the nearest representative object;
(4)     randomly select a nonrepresentative object, $o_{random}$;
(5)     compute the total cost, $S$, of swapping representative object, $o_j$, with $o_{random}$;
(6)     if $S < 0$ then swap $o_j$ with $o_{random}$ to form the new set of $k$ representative objects;
(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 influenced by outliers or other extreme values than a mean. However, its processing is more costly than the *k*-means method.

Hierarchical Clustering Methods:

- A hierarchical clustering method works by grouping data objects into a tree of clusters.
- The quality of a pure hierarchical clustering method suffers from its inability to perform adjustment once a merge or split decision has been executed. That is, if a particular merge or split decision later turns out to have been a poor choice, the method cannot backtrack and correct it.

Hierarchical clustering methods can be further classified as either agglomerative or divisive, depending on whether the hierarchical decomposition is formed in a bottom-up or top-down fashion.

Agglomerative hierarchical clustering:

- This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied.
- Most hierarchical clustering methods belong to this category. They differ only in their definition of inter cluster similarity.

Divisive hierarchical clustering:

- This top-down strategy does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster.
- It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the diameter of each cluster is within a certain threshold.

Constraint-Based Cluster Analysis:

Constraint-based clustering finds clusters that satisfy user-specified preferences or constraints. Depending on the nature of the constraints, constraint-based clustering may adopt rather different approaches.

There are a few categories of constraints.

- **Constraints on individual objects:**

We can specify constraints on the objects to be clustered. In a real estate application, for example, one may like to spatially cluster only those luxury mansions worth over a million dollars. This constraint confines the set of objects to be clustered. It can easily be handled by preprocessing after which the problem reduces to an instance of constrained clustering.

- **Constraints on the selection of clustering parameters:**

A user may like to set a desired range for each clustering parameter. Clustering parameters are usually quite specific to the given clustering algorithm. Examples of parameters include k, the desired number of clusters in a k-means algorithm; or e the radius and the minimum number of points in the DBSCAN algorithm. Although such user-specified parameters may strongly influence the clustering results, they are usually confined to the algorithm itself. Thus, their fine tuning and processing are usually not considered a form of constraint-based clustering.

- **Constraints on distance or similarity functions:**

We can specify different distance or similarity functions for specific attributes of the objects to be clustered, or different distance measures for specific pairs of objects.When clustering sportsmen, for example,we may use different weighting schemes for height, body weight, age, and skill level. Although this will likely change the mining results, it may not alter the clustering process per se. However, in some cases, such changes may make the evaluation of the distance function nontrivial, especially when it is tightly intertwined with the clustering process.

- **User-specified constraints on the properties of individual clusters:**

A user may like to specify desired characteristics of the resulting clusters, which may strongly influence the clustering process.

- **Semi-supervised clustering based on partial supervision:**

The quality of unsupervised clustering can be significantly improved using some weak form of supervision.This may be in the form of pairwise constraints (i.e., pairs of objects labeled as belonging to the same or different cluster). Such a constrained clustering process is called semi-supervised clustering.

Outlier Analysis:

- There exist data objects that do not comply with the general behavior or model of the data. Such data objects, which are grossly different from or inconsistent with the remaining set of data, are called outliers.

- Many data mining algorithms try to minimize the influence of outliers or eliminate them all together. This, however, could result in the loss of important hidden information because one person's noise could be another person's signal. In other words, the outliers may be of particular interest, such as in the case of fraud detection, where outliers may indicate fraudulent activity. Thus, outlier detection and analysis is an interesting data mining task, referred to as outlier mining.
- It can be used in fraud detection, for example, by detecting unusual usage of credit cards or telecommunication services. In addition, it is useful in customized marketing for identifying the spending behavior of customers with extremely low or extremely high incomes, or in medical analysis for finding unusual responses to various medical treatments.

Outlier mining can be described as follows: Given a set of *n* data points or objects and *k*, the expected number of outliers, find the top *k* objects that are considerably dissimilar, exceptional, or inconsistent with respect to the remaining data. The outlier mining problem can be viewed as two subproblems:

- Define what data can be considered as inconsistent in a given data set, and
- Find an efficient method to mine the outliers so defined.

Types of outlier detection:

- Statistical Distribution-Based Outlier Detection
- Distance-Based Outlier Detection
- Density-Based Local Outlier Detection
- Deviation-Based Outlier Detection

Statistical Distribution-Based Outlier Detection:

The statistical distribution-based approach to outlier detection assumes a distribution or probability model for the given data set (e.g., a normal or Poisson distribution) and then identifies outliers with respect to the model using a discordancy test. Application of the test requires knowledge of the data set parameters, knowledge of distribution parameters such as the mean and variance and the expected number of outliers.

A statistical discordancy test examines two hypotheses:

- A working hypothesis
- An alternative hypothesis

A working hypothesis, H, is a statement that the entire data set of n objects comes from an initial distribution model, F, that is,

$$H : o_i \in F, \quad \text{where } i = 1, 2, \ldots, n.$$

The hypothesis is retained if there is no statistically significant evidence supporting its rejection. A discordancy test verifies whether an object, oi, is significantly large (or small) in relation to the distribution F. Different test statistics have been proposed for use as a discordancy test, depending on the available knowledge of the data. Assuming that some statistic, T, has been chosen for discordancy testing, and the value of the statistic for object oi is vi, then the distribution of T is constructed. Significance probability, SP(vi)=Prob(T > vi), is evaluated. If SP(vi) is sufficiently small, then oi is discordant and the working hypothesis is rejected.

An alternative hypothesis, H, which states that o_i comes from another distribution model, G, is adopted. The result is very much dependent on which model F is chosen because o_i may be an outlier under one model and a perfectly valid value under another. The alternative distribution is very important in determining the power of the test, that is, the probability that the working hypothesis is rejected when oi is really an outlier.

There are different kinds of alternative distributions.

- **Inherent alternative distribution:**

  In this case, the working hypothesis that all of the objects come from distribution F is rejected in favor of the alternative hypothesis that all of the objects arise from another distribution, G:

  H :oi € G, where i = 1, 2,…, n

  F and G may be different distributions or differ only in parameters of the same distribution. There are constraints on the form of the *G* distribution in that it must have potential to produce outliers. For example, it may have a different mean or dispersion, or a longer tail.

- **Mixture alternative distribution:**

  The mixture alternative states that discordant values are not outliers in the F population, but contaminants from some other population, G. In this case, the alternative hypothesis is

  This alternative states that all of the objects (apart from some prescribed small number) arise independently from the initial model, F, with its given parameters, whereas the remaining objects are independent observations from a modified version of F in which the parameters have been shifted.

There are two basic types of procedures for detecting outliers: **Block procedures:**

In this case, either all of the suspect objects are treated as outliers or all of them are accepted as consistent.

Consecutive procedures:

An example of such a procedure is the *inside out procedure*. Its main idea is that the object

that is least likely to be an outlier is tested first. If it is found to be an outlier, then all of the more extreme values are also considered outliers; otherwise, the next most extreme object is tested, and so on. Thisprocedure tends to be more effective than block procedures.

Distance-Based Outlier Detection:

The notion of distance-based outliers was introduced to counter the main limitations imposed by statistical methods. An object, o, in a data set, D, is a distance-based (DB)outlier with parameters pct and dmin,that is, a DB(pct;dmin)-outlier, if at least a fraction,pct, of the objects in D lie at a distance greater than dmin from o. In other words, rather than relying on statistical tests, we can think of distance-based outliers as those objects that do not have enough neighbors, where neighbors are defined based on distance from the given object. In comparison with statistical-based methods, distance based outlier detection generalizes the ideas behind discordancy testing for various standard distributions. Distance-based outlier detection avoids the excessive computation that can be associated with fitting the observed distribution into some standard distribution and in selecting discordancy tests.

For many discordancy tests, it can be shown that if an object, o, is an outlier according to the given test, then o is also a DB(pct, dmin)-outlier for some suitably defined pct and dmin. For example, if objects that lie three or more standard deviations from the mean are considered to be outliers, assuming a normal distribution, then this definition can be generalized by a DB(0.9988, 0.13s) outlier.

Several efficient algorithms for mining distance-based outliers have been developed. **Index-based algorithm:**

Given a data set, the index-based algorithm uses multidimensional indexing structures, such as R-trees or k-d trees, to search for neighbors of each object ***o*** within radius *dmin* around that object. Let *M Be* the maximum number of objects within the *dmin*-neighborhood of an outlier. Therefore, once*M*+1 neighbors of object ***o*** are found, it is clear that ***o*** is not an outlier. This algorithm has a worst-case complexity of *O*(*n*2*k*), where *n* is the number of objects in the data set and *k* is the dimensionality. The index-based algorithm scales well as *k* increases. However, this complexity evaluation takes only the search time into account, even though the task of building an index in itself can be computationally intensive.

Nested-loop algorithm:

The nested-loop algorithm has the same computational complexity as the index-based algorithm but avoids index structure construction and tries to minimize the number of I/Os. It divides the memory buffer space into two halves and the data set into several logical blocks. By carefully choosing the order in which blocks are loaded into each half, I/O efficiency can be achieved.

Cell-based algorithm:

To avoidO(n^2) computational complexity, a cell-based algorithm was developed for memory resident data sets. Its complexity is O(c^k+n), where c is a constant depending on the number of cells and k is the dimensionality.

In this method, the data space is partitioned into cells with a side length equal to $\frac{dmin}{2\sqrt{k}}$.

Eachcell has two layers surrounding it. The first layer is one cell thick, while the second is $\lceil 2\sqrt{k} - 1 \rceil$ cells thick, rounded up to the closest integer. The algorithm count outliers on a cell-by-cell rather than an object-by-object basis. For a given cell, it accumulates three counts—the number of objects in the cell, in the cell and the first layer together, and in the cell and both layers together. Let's refer to these counts as cell count, cell + 1 layer count, and cell + 2 layers count, respectively.

Let M be the maximum number of outliers that can exist in the dmin-neighborhood of an outlier.

- An object, **o**, in the current cell is considered an outlier only if cell + 1 layer count is less than or equal to M. If this condition does not hold, then all of the objects in the cell can be removed from further investigation as they cannot be outliers.
- If cell_+ 2_layers_count is less than or equal to M, then all of the objects in the cell are considered outliers. Otherwise, if this number is more than M, then itis possible that some of the objects in the cell may be outliers. To detect the outliers, object-by-object processing is used where, for each object, **o**, in the cell,objects in the second layer of **o** are examined. For objects in the cell, only those objects having no more than M points in their dmin-neighborhoods are outliers.The dmin-neighborhood of an object consists of the object's cell, all of its first layer, and some of its second layer.

A variation to the algorithm is linear with respect to n and guarantees that no more than three passes over the data set are required. It can be used for large disk-resident data sets, yet does not scale well for high dimensions.

Density-Based Local Outlier Detection:

Statistical and distance-based outlier detection both depend on the overall or global distribution of the given set of data points, D. However, data are usually not uniformly distributed. These methods encounter difficulties when analyzing data with rather different density distributions.

To define the local outlier factor of an object, we need to introduce the concepts of distance, k-distance neighborhood, reachability distance,13 and local reachability density.

These are defined as follows:

The k-distance of an object p is the maximal distance that p gets from its k nearest neighbors. This distance is denoted as k-distance(p). It is defined as the distance, d(p, o), between p and an object o 2 D, such that for at least k objects, o_0 2 D, it holds that d(p, o')_d(p, o). That is, there are at least k objects in D that are as close as or closer to p than o, and for at most k-1 objects, 2 D, it holds that d(p;o'') <d(p, o).

That is, there are at most k-1 objects that are closer to p than o. You may be wondering at this point how k is determined. The LOF method links to density-based clustering in that it sets k to the parameter rMinPts,which specifies the minimum number of points for use in identifying clusters based on density.

Here, MinPts (as k) is used to define the local neighborhood of an object, p.

The k-distance neighborhood of an object p is denoted $N_{kdistance(p)(}$p), or N_k(p)for short. By setting k to MinPts, we get N_{MinPts}(p). It contains the MinPts-nearestneighbors of p. That is, it contains every object whose distance is not greater than theMinPts-distance of p. The reachability distance of an object p with respect to object o (where o is within theMinPts-nearest neighbors of p), is defined as reach distMinPts(p, o) = max{MinPtsdistance(o), d(p, o)}.

Intuitively, if an object p is far away , then the reachability distance between the two is simply their actual distance. However, if they are sufficiently close (i.e., where p is within the MinPts-distance neighborhood of o), then the actual distance is replaced by the MinPtsdistance of o. This helps to significantly reduce the statistical fluctuations of d(p, o) for all of the p close to o.

The higher the value of MinPts is, the more similar is the reachability distance for objects within the same neighborhood.

Intuitively, the local reachability density of p is the inverse of the average reachability density based on the MinPts-nearest neighbors of p. It is defined as

$$lrd_{MinPts}(\boldsymbol{p}) = \frac{|N_{MinPts}(\boldsymbol{p})|}{\Sigma_{\boldsymbol{o} \in N_{MinPts}(\boldsymbol{p})} reach\_dist_{MinPts}(\boldsymbol{p}, \boldsymbol{o})}. \quad \textbf{(5.3)}$$

The local outlier factor (LOF) of ***p*** captures the degree to which we call ***p*** an outlier. It is defined as

$$LOF_{MinPts}(\boldsymbol{p}) = \frac{\sum_{\boldsymbol{o}\in N_{MinPts}(\boldsymbol{p})} \frac{lrd_{MinPts}(\boldsymbol{o})}{lrd_{MinPts}(\boldsymbol{p})}}{|N_{MinPts}(\boldsymbol{p})|}. \quad \textbf{(5.4)}$$

It is the average of the ratio of the local reachability density of ***p*** and those of ***p***'s *MinPts*-nearest neighbors. It is easy to see that the lower ***p***'s local reachability density is, and the higher the local reachability density of ***p***'s *MinPts*-nearest neighbors are, the higher *LOF*(***p***) is.

Deviation-Based Outlier Detection:

Deviation-based outlier detection does not use statistical tests or distance-based measures to identify exceptional objects. Instead, it identifies outliers by examining the main characteristics of objects in a group.Objects that —deviate‖ from this description are considered outliers. Hence, in this approach the term deviations is typically used to refer to outliers. In this section, we study two techniques for deviation-based outlier detection.The first sequentially compares objects in a set, while the second employs an OLAPdata cube approach.

Sequential Exception Technique:

The sequential exception technique simulates the way in which humans can distinguish unusual objects from among a series of supposedly like objects. It uses implicit redundancy of the data. Given a data set, D, of n objects, it builds a sequence of subsets,{D1, D2, …,Dm}, of these objects with 2<=m <= n such that

$$D_{j-1} \subset D_j, \quad \text{where } D_j \subseteq D.$$

Dissimilarities are assessed between subsets in the sequence. The technique introduces the following key terms.

Exception set:

This is the set of deviations or outliers. It is defined as the smallest subset of objects whose removal results in the greatest reduction of dissimilarity in the residual set.

Dissimilarity function:

This function does not require a metric distance between the objects. It is any function that, if given a set of objects, returns a low value if the objects are similar to one another. The greater the dissimilarity among the objects, the higher the value returned by the function. The dissimilarity of a subset is incrementally computed based on the subset prior to it in the sequence. Given a subset of *n* numbers, $\{x_1, \ldots, x_n\}$, a possible dissimilarity function is the variance of the numbers in the set, that is,

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2, \quad \textbf{(5.5)}$$

where x is the mean of the n numbers in the set. For character strings, the dissimilarity function may be in the form of a pattern string (e.g., containing wildcard character that is used to cover all of the patterns seen so far. The dissimilarity increases when the pattern covering all of the strings in D_{j-1} does not cover any string in D_j that is not in D_{j-1}.

Cardinality function:

This is typically the count of the number of objects in a given set.

Smoothing factor:

This function is computed for each subset in the sequence. Itassesses how much the dissimilarity can be reduced by removing the subset from the original set of objects.

UNIT EXERCISE

2 Marks

1. Compare the time complexity of Cell based and Nested loop algorithms for distance based outlier detection.
2. What is clustering and mention a few real time applications of clustering
3. What are the types of outlier detection?
4. Write the K-Medoids algorithm
5. Define smoothing factor
6. What is dissimilarity function and write the equation
7. What is the difference between consecutive and block procedures?
8. Explain the four cases of the cost function for *k*-medoids clustering
9. What is a sequential Exception technique?
10. Mention the Applications of clustering in real life.

8 Marks

1. Explain Typical Requirements Of Clustering In Data Mining
2. Explain all the Density based outlier detection methods with the necessary equations
3. What are the differences between Agglomerative and Divisive hierarchical clustering and explain with a numerical example.