

➤ **Data warehousing and OLAP:**

The construction of data warehouses involves data cleaning, data integration and transformation and can be viewed as an important preprocessing step for data mining. Data warehouses provide online analytical processing (OLAP) tools for the interactive analysis of multidimensional data of varied granularities, which facilitates effective data generalization and data mining.

Many other data mining functions, such as association, classification, prediction, and clustering, can be integrated with OLAP operations to enhance interactive mining of knowledge at multiple levels of abstraction. The data warehouse has become an increasingly important platform for data analysis and online analytical processing and will provide an effective platform for data mining.

Data warehousing and OLAP form an essential step in the knowledge discovery process. We study a well-accepted definition of the data warehouse and see why more and more organizations are building data warehouses for the analysis of their data. In that data cube, a multidimensional data model for data warehouses and OLAP, as well as OLAP operations such as roll-up, drill-down, slicing, and dicing.

➤ **What is Data Warehouse?**

Data warehousing provides architectures and tools for business executives to systematically organize, understand, and use their data to make strategic decisions.

According to William H. Inmon, a leading architect in the construction of data warehouse systems, "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support for management's decision making process". The four keywords, subject-oriented, integrate, time-variant, and nonvolatile, distinguish data warehouses from other data repository systems, such as relational database systems, transaction processing systems and file system.

Data warehousing is the process of constructing and using a data warehouse. A data warehouse is constructed by integrating data from multiple heterogeneous sources that support analytical reporting, structured and/or ad hoc queries, and decision making. Data warehousing involves data cleaning, data integration, and data consolidations.

A Data Warehouse provides integrated, enterprise-wide, historical data and focuses on providing support for decision-makers for data modeling and analysis. Data Warehouse is a group of data specific to the entire organization, not only to a particular group of users. It is not used for daily operations and transaction processing but used for making decisions.

A Data Warehouse can be viewed as a data system with the following attributes:

- It is a database designed for investigative tasks, using data from various applications.
- It supports a relatively small number of clients with relatively long interactions.
- It includes current and historical data to provide a historical perspective of information.
- Its usage is read-intensive.
- It contains a few large tables.

"Data Warehouse is a subject-oriented, integrated, and time-variant store of information in support of management's decisions."

Characteristics of Data Warehouse

Subject-Oriented

A data warehouse targets the modeling and analysis of data for decision-makers. Therefore, data warehouses typically provide a concise and straightforward view around a particular subject, such as customer, product, or sales, instead of the global organization's ongoing operations. This is done by excluding data that are not useful concerning the subject and including all data needed by the users to understand the subject.

Integrated

A data warehouse integrates various heterogeneous data sources like RDBMS, flat files, and online transaction records. It requires performing data cleaning and integration during data warehousing to ensure consistency in naming conventions, attributes types, etc., among different data sources.

Time-Variant

Historical information is kept in a data warehouse. For example, one can retrieve files from 3 months, 6 months, 12 months, or even previous data from a data warehouse. These variations with a transactions system, where often only the most current file is kept.

Non-Volatile

The data warehouse is a physically separate data storage, which is transformed from the source operational RDBMS. The operational updates of data do not occur in the data warehouse, i.e., update, insert, and delete operations are not performed. It usually requires only two procedures in data accessing: Initial loading of data and access to data. Therefore, the DW does not require transaction processing, recovery, and concurrency capabilities, which allows for substantial speedup of data retrieval. Non-Volatile defines that once entered into the warehouse, and data should not change.

Data Warehousing is an approach that can collect and manage data from multiple sources to provide the business a significant business insight. A data warehouse is specifically designed to provide management decisions.

In simple terms, a data warehouse defines a database that is maintained independently from an organization's operational databases. Data warehouse systems enable the integration of multiple application systems. They support data processing by providing a solid platform of consolidated, historical data for analysis.

A data warehouse is a semantically consistent data save that handle as a physical execution of a decision support data model. It saves the data an enterprise required to create strategic decisions. A data warehouse is viewed as architecture, constructed by integrating records from several heterogeneous sources to provide structured and ad hoc queries, analytical documenting, and decision making.

The construction of a data warehouse needed data cleaning, data integration, and data consolidation. The use of a data warehouse necessitates a set of decision support technologies. This enables "knowledge workers" such as managers, analysts, and executives to need the warehouse to fastly and conveniently obtain an overview of the data, and to create sound decisions depends on data in the warehouse.

Some organizations use this data to provide business decision-making activities, including

- It can be raising user focus, which involves the analysis of user buying patterns including buying preference, buying time, budget cycles, and demand for spending.
- It can be used to repositioning products and handling product portfolios by comparing the implementation of sales by quarter, by year, and by geographic regions to fine-tune production methods.
- It can be analyzing operations and looking for sources of profit.
- It can be managing customer relationships, making environmental corrections, and managing the cost of corporate assets.

Data warehousing is beneficial from the point of view of heterogeneous database integration. Organizations generally collect several kinds of data and supports large databases from multiple, heterogeneous, autonomous, and distributed data sources.

It is hugely desirable, yet challenging, to combine such data and supports easy and effective access to it. Much effort has been used in the database market and research community toward achieving this aim.

The traditional database method to heterogeneous database integration is to construct wrappers and integrators (or mediators) on top of multiple, heterogeneous databases. When a query is posed to a

user site, a metadata dictionary can be used to interpret the query into queries appropriate for the single heterogeneous sites included.

These queries are mapped and shared to local query processors. The results restored from the multiple sites are unified into a global answer set. This query-driven method needed complex data filtering and integration procedure, and competes with local sites for processing resources. It is ineffective and potentially costly for frequent queries, particularly queries needed aggregations.

Need for Data Warehouse

Data Warehouse is needed for the following reasons:

1. **1) Business User:** Business users require a data warehouse to view summarized data from



t

. Since these people are non-technical, the data may be presented to them in an elementary form.

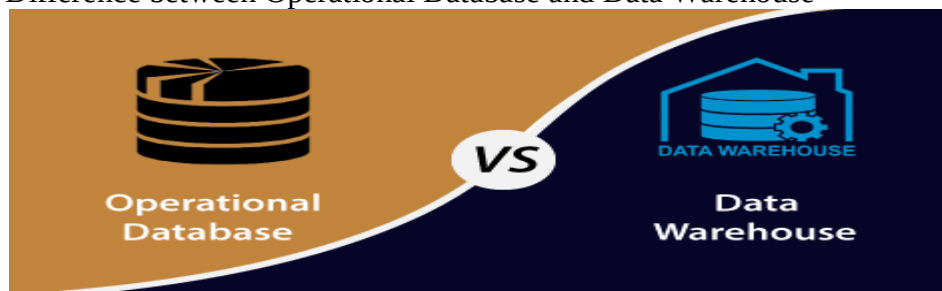
2. **2) Store historical data:** Data Warehouse is required to store the time variable data from the past. This input is made to be used for various purposes.
3. **3) Make strategic decisions:** Some strategies may be depending upon the data in the data warehouse. So, data warehouse contributes to making strategic decisions.
4. **4) For data consistency and quality:** Bringing the data from different sources at a commonplace, the user can effectively undertake to bring the uniformity and consistency in data.
5. **5) High response time:** Data warehouse has to be ready for somewhat unexpected loads and types of queries, which demands a significant degree of flexibility and quick response time.

Benefits of Data Warehouse

1. Understand business trends and make better forecasting decisions.
2. Data Warehouses are designed to perform well enormous amounts of data.
3. The structure of data warehouses is more accessible for end-users to navigate, understand, and query.
4. Queries that would be complex in many normalized databases could be easier to build and maintain in data warehouses.
5. Data warehousing is an efficient method to manage demand for lots of information from lots of users.
6. Data warehousing provide the capabilities to analyze a large amount of historical data.

Differences between Operational Database Systems and Data Warehouses

Difference between Operational Database and Data Warehouse



The Operational Database is the source of information for the data warehouse. It includes detailed information used to run the day to day operations of the business. The data frequently changes as updates are made and reflect the current value of the last transactions.

Operational Database Management Systems also called as OLTP (Online Transactions Processing Databases), are used to manage dynamic data in real-time.

Data Warehouse Systems serve users or knowledge workers in the purpose of data analysis and decision-making. Such systems can organize and present information in specific formats to accommodate the diverse needs of various users. These systems are called as Online-Analytical Processing (OLAP) Systems.

Data Warehouse and the OLTP database are both relational databases. However, the goals of both these databases are different.

The major distinguishing features between OLTP and OLAP are summarized as follows:

Users and system orientation: An OLTP system is customer-oriented and is used for transaction and query processing by clerks, clients, and information technology professionals. An OLAP system is market-oriented and is used for data analysis by knowledge workers, including managers, executives, and analytics.

Data contents: An OLTP system manages current data that typically are too detailed to be easily used for decision making. An OLAP system manages large amounts of historical data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity. These features make the data easier to use in informed decision making.

Database design: An OLTP system usually adopts an entity-relationship(ER) data model and an application-oriented database design. An OLAP system typically adopts either a star or snowflake model or a subject oriented database design.

View: An OLTP system focuses mainly on the current data within an enterprise or department, without referring to historical data or data in different organizations. OLAP systems also deal with information that originates from different organizations, integrating information from many data stores.

Access patterns: The access patterns of an OLTP system consist mainly of short; atomic transactions. Such a system requires concurrency control and recovery mechanisms. Accesses to OLAP systems are mostly read-only operations (because most data warehouses store historical rather than up-to-date information), although many could be complex queries.

<i>Feature</i>	<i>OLTP</i>	<i>OLAP</i>
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long-term informational requirements, decision support
DB design	ER based, application-oriented	star/snowflake, subject-oriented
Data	current; guaranteed up-to-date	historical; accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
Number of records accessed	tens	millions

Number of users	thousands	hundreds
DB size	100 MB to GB	100 GB to TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Why Have a Separate Data Warehouse?

An operational database is designed and tuned from known tasks and workloads, such as indexing and hashing using primary keys, searching for particular records, and optimizing “canned” queries. They involve the computation of large groups of data at summarized levels, and may require the use of special data organization, access, and implementation methods based on multidimensional views. Processing OLAP queries in operational databases would substantially degrade the performance of operational tasks. Moreover, an operational database supports the concurrent processing of multiple transactions. Concurrency control and recovery mechanisms, such as locking and logging, are required to ensure the consistency and robustness of transactions. An OLAP query often needs read-only access of data records for summarization and aggregation. Concurrency control and recovery mechanisms, if applied for such OLAP operations, may jeopardize the execution of concurrent transactions and thus substantially reduce the throughput of an OLTP system.

Finally, the separation of operational databases from data warehouses is based on the different structures, contents, and uses of the data in these two systems. Decision support requires historical data, whereas operational databases do not typically maintain historical data. In this context, the data in operational databases, though abundant, is usually far from complete for decision making. Decision support requires consolidation (such as aggregation and summarization) of data from heterogeneous sources, resulting in high-quality, clean, and integrated data. In contrast, operational databases contain only detailed raw data, such as transactions, which need to be consolidated before analysis. Because the two systems provide quite different functionalities and require different kinds of data, it is presently necessary to maintain separate databases. However, many vendors of operational relational database management systems are beginning to optimize such systems to support OLAP queries. As this trend continues, the separation between OLTP and OLAP systems is expected to decrease.

➤ Data Warehouse Modeling

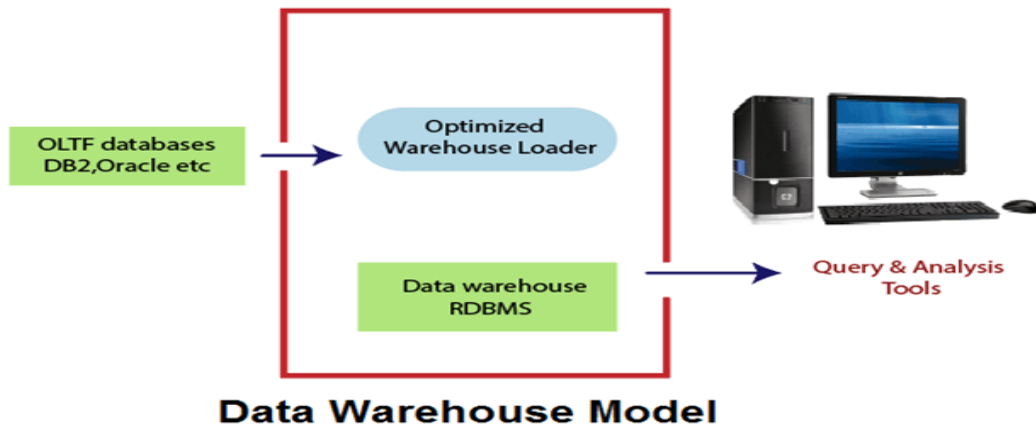
Data warehouse modeling is the process of designing the schemas of the detailed and summarized information of the data warehouse. The goal of data warehouse modeling is to develop a schema describing the reality, or at least a part of the fact, which the data warehouse is needed to support.

Data warehouse modeling is an essential stage of building a data warehouse for two main reasons. Firstly, through the schema, data warehouse clients can visualize the relationships among the warehouse data, to use them with greater ease. Secondly, a well-designed schema allows an effective data warehouse structure to emerge, to help decrease the cost of implementing the warehouse and improve the efficiency of using it.

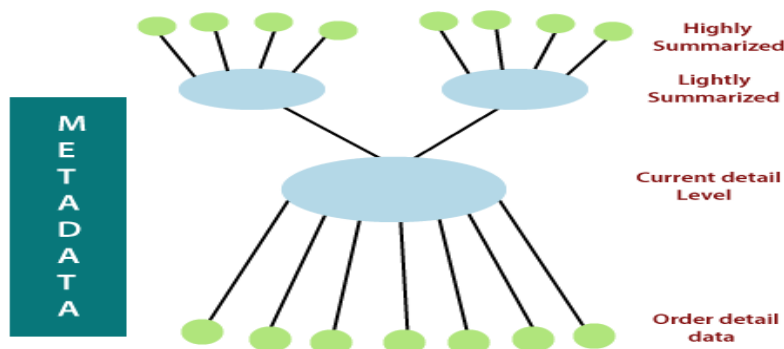
Data modeling in data warehouses is different from data modeling in operational database systems. The primary function of data warehouses is to support DSS processes. Thus, the objective of data warehouse modeling is to make the data warehouse efficiently support complex queries on long term information.

In contrast, data modeling in operational database systems targets efficiently supporting simple transactions in the database such as retrieving, inserting, deleting, and changing data. Moreover, data warehouses are designed for the customer with general information knowledge about the enterprise, whereas operational database systems are more oriented toward use by software specialists for creating distinct applications.

Data Warehouse model is illustrated in the given diagram.



The data within the specific warehouse itself has a particular architecture with the emphasis on various levels of summarization, as shown in figure:



The Structure of data inside the data warehouse

The current detail record is central in importance as it:

- Reflects the most current happenings, which are commonly the most stimulating.
- It is numerous as it is saved at the lowest method of the Granularity.
- It is always (almost) saved on disk storage, which is fast to access but expensive and difficult to manage.

Older detail data is stored in some form of mass storage, and it is infrequently accessed and kept at a level detail consistent with current detailed data.

Lightly summarized data is data extract from the low level of detail found at the current, detailed level and usually is stored on disk storage. When building the data warehouse have to remember what unit of time is summarization done over and also the components or what attributes the summarized data will contain.

Highly summarized data is compact and directly available and can even be found outside the warehouse.

Metadata is the final element of the data warehouses and is really of various dimensions in which it is not the same as file drawn from the operational data, but it is used as:-

- A directory to help the DSS investigator locate the items of the data warehouse.
- A guide to the mapping of record as the data is changed from the operational data to the data warehouse environment.
- A guide to the method used for summarization between the current, accurate data and the lightly summarized information and the highly summarized data, etc.

Data Modeling Life Cycle

In this section, we define a data modeling life cycle. It is a straight forward process of transforming the business requirements to fulfill the goals for storing, maintaining, and accessing the data within IT systems. The result is a logical and physical data model for an enterprise data warehouse.

The objective of the data modeling life cycle is primarily the creation of a storage area for business information. That area comes from the logical and physical data modeling stages, as shown in

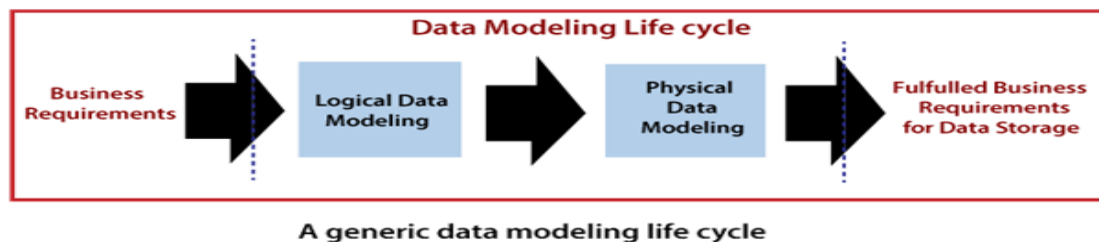


Figure:

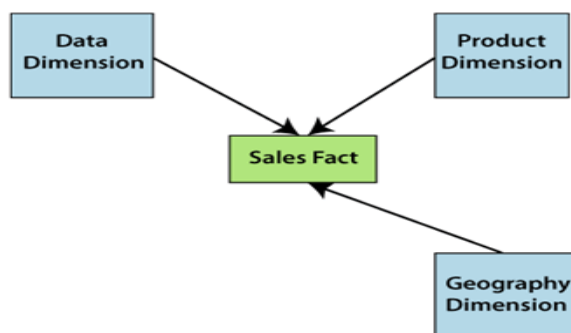
Conceptual Data Model

A conceptual data model recognizes the highest -level relationships between the different entities.

Characteristics of the conceptual data model

- It contains the essential entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.

We can see that the only data shown via the conceptual data model is the entities that define the data and the relationships between those entities. No other data, as shown through the conceptual data model.



Example of Conceptual Data Model

Logical Data Model

A logical data model defines the information in as much structure as possible, without observing how they will be physically achieved in the database. The primary objective of logical data modeling is to document the business data structures, processes, rules, and relationships by a single view - the logical data model.

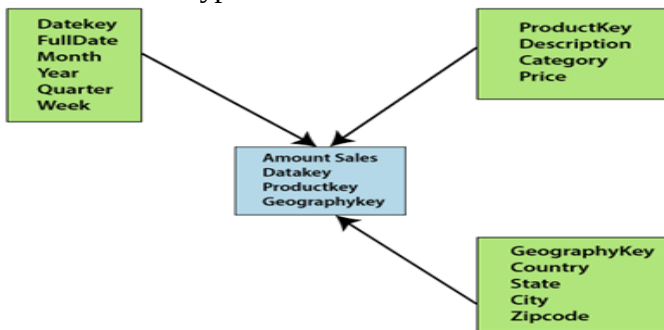
Features of a logical data model

- It involves all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is stated.
- Referential Integrity is specified (FK Relation).

The phase for designing the logical data model which are as follows:

- Specify primary keys for all entities.
- List the relationships between different entities.
- List all attributes for each entity.

- Normalization.
- No data types are listed



Example of Logical Data Model

Physical Data Model

Physical data model describes how the model will be presented in the database. A physical database model demonstrates all table structures, column names, data types, constraints, primary key, foreign key, and relationships between tables. The purpose of physical data modeling is the mapping of the logical data model to the physical structures of the RDBMS system hosting the data warehouse. This contains defining physical RDBMS structures, such as tables and data types to use when storing the information. It may also include the definition of new data structures for enhancing query performance.

Characteristics of a physical data model

- Specification all tables and columns.
- Foreign keys are used to recognize relationships between tables.

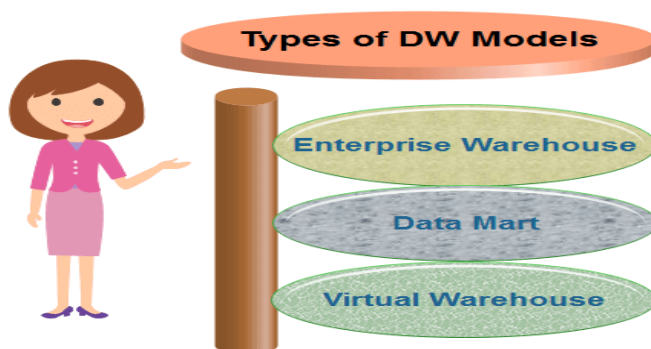
The steps for physical data model design which are as follows:

- Convert entities to tables.
- Convert relationships to foreign keys.
- Convert attributes to columns.



Example of Physical Data Model

Types of Data Warehouse Models



Enterprise Warehouse

An Enterprise warehouse collects all of the records about subjects spanning the entire organization. It supports corporate-wide data integration, usually from one or more operational systems or external data providers, and it's cross-functional in scope. It generally contains detailed information as well as summarized information and can range in estimate from a few gigabyte to hundreds of gigabytes, terabytes, or beyond.

An enterprise data warehouse may be accomplished on traditional mainframes, UNIX super servers, or parallel architecture platforms. It required extensive business modeling and may take years to develop and build.

Data Mart

A data mart includes a subset of corporate-wide data that is of value to a specific collection of users. The scope is confined to particular selected subjects. For example, a marketing data mart may restrict its subjects to the customer, items, and sales. The data contained in the data marts tend to be summarized.

Data Marts is divided into two parts:

Independent Data Mart: Independent data mart is sourced from data captured from one or more operational systems or external data providers, or data generally locally within a different department or geographic area.

Dependent Data Mart: Dependent data marts are sourced exactly from enterprise data-warehouses.

Virtual Warehouses

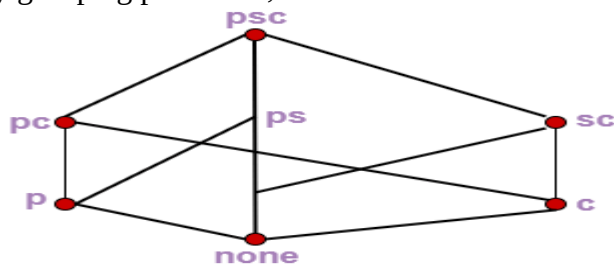
Virtual Data Warehouses is a set of perception over the operational database. For effective query processing, only some of the possible summary vision may be materialized. A virtual warehouse is simple to build but required excess capacity on operational database servers.

➤ What is Data Cube?

When data is grouped or combined in multidimensional matrices called Data Cubes. The data cube method has a few alternative names or a few variants, such as "Multidimensional databases," "materialized views," and "OLAP (On-Line Analytical Processing)."

The general idea of this approach is to materialize certain expensive computations that are frequently inquired.

For example, a relation with the schema sales (part, supplier, customer, and sale-price) can be materialized into a set of eight views as shown in fig, where **psc** indicates a view consisting of aggregate function value (such as total-sales) computed by grouping three attributes part, supplier, and customer, **p** indicates a view composed of the corresponding aggregate function values calculated by grouping part alone, etc.



Eight views of data cubes for sales information.

A data cube is created from a subset of attributes in the database. Specific attributes are chosen to be measure attributes, i.e., the attributes whose values are of interest. Another attributes are selected as dimensions or functional attributes. The measure attributes are aggregated according to the dimensions.

For example, XYZ may create a sales data warehouse to keep records of the store's sales for the dimensions time, item, branch, and location. These dimensions enable the store to keep track of

things like monthly sales of items, and the branches and locations at which the items were sold. Each dimension may have a table identify with it, known as a dimensional table, which describes the dimensions. For example, a dimension table for items may contain the attributes item_name, brand, and type.

Data cube method is an interesting technique with many applications. Data cubes could be sparse in many cases because not every cell in each dimension may have corresponding data in the database.

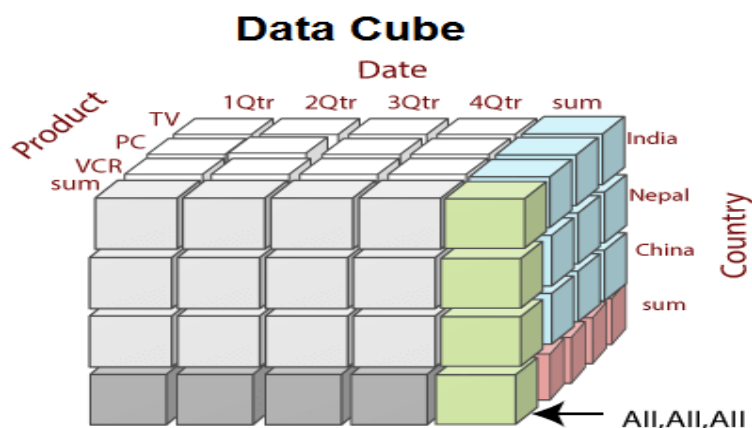
Techniques should be developed to handle sparse cubes efficiently.

If a query contains constants at even lower levels than those provided in a data cube, it is not clear how to make the best use of the precomputed results stored in the data cube.

The model view data in the form of a data cube. OLAP tools are based on the multidimensional data model. Data cubes usually model n-dimensional data.

A data cube enables data to be modeled and viewed in multiple dimensions. A multidimensional data model is organized around a central theme, like sales and transactions. A fact table represents this theme. Facts are numerical measures. Thus, the fact table contains measure (such as Rs_sold) and keys to each of the related dimensional tables.

Dimensions are a fact that defines a data cube. Facts are generally quantities, which are used for analyzing the relationship between dimensions.



Example: In the 2-D representation, we will look at the All Electronics sales data for **items sold per quarter** in the city of Vancouver. The measured display in dollars sold (in thousands).

2-D view of Sales Data

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

3-Dimensional Cuboids

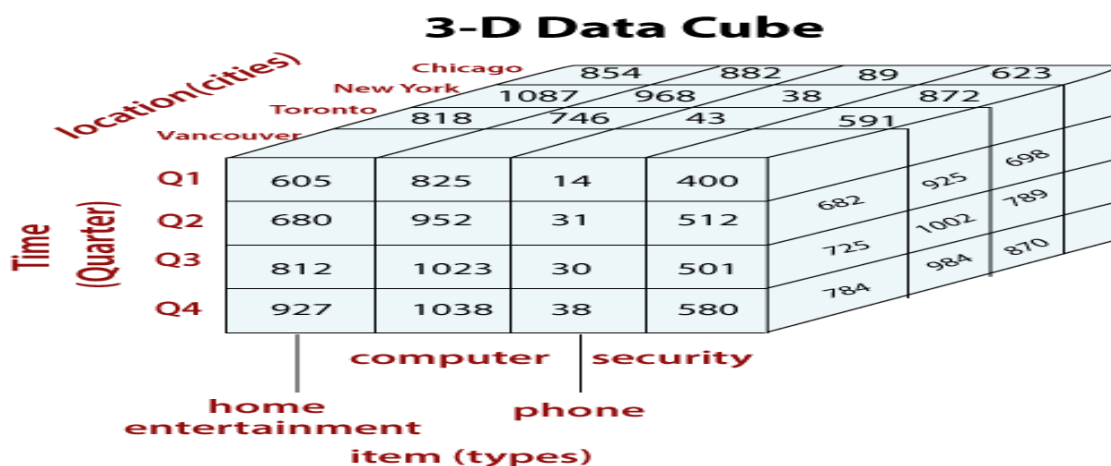
Let suppose we would like to view the sales data with a third dimension. For example, suppose we would like to view the data according to time, item as well as the location for the cities Chicago, New

York, Toronto, and Vancouver. The measured display in dollars sold (in thousands). These 3-D data are shown in the table. The 3-D data of the table are represented as a series of 2-D tables.

3-D view of Sales Data

location ="Chicago"					location ="New York"					location ="Toronto"				
item					item					item				
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	
Q2	943	890	64	698	1130	1024	41	925		894	769	52	682	
Q3	1032	924	59	789	1034	1048	45	1002		940	795	58	728	
Q4	1129	992	63	870	1142	1091	54	984		978	864	59	784	

Conceptually, we may represent the same data in the form of 3-D data cubes, as shown in fig:



Let us suppose that we would like to view our sales data with an additional fourth dimension, such as a supplier.

In data warehousing, the data cubes are n-dimensional. The cuboid which holds the lowest level of summarization is called a **base cuboid**.

For example, the **4-D cuboid** in the figure is the base cuboid for the given time, item, location, and supplier dimensions.

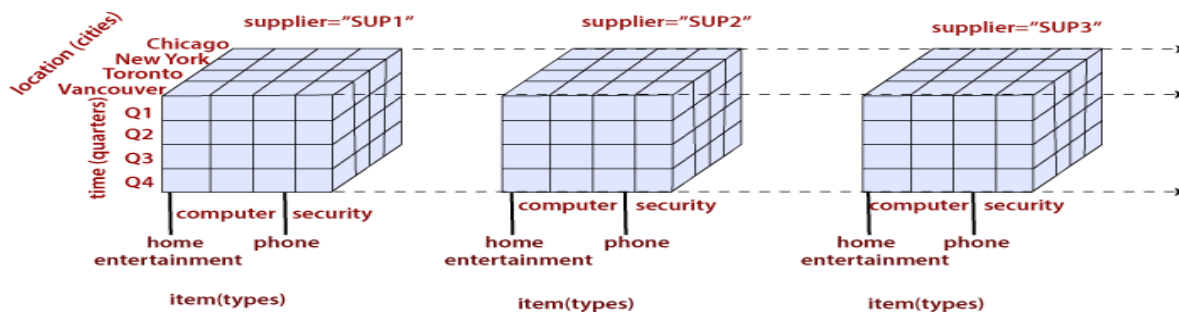
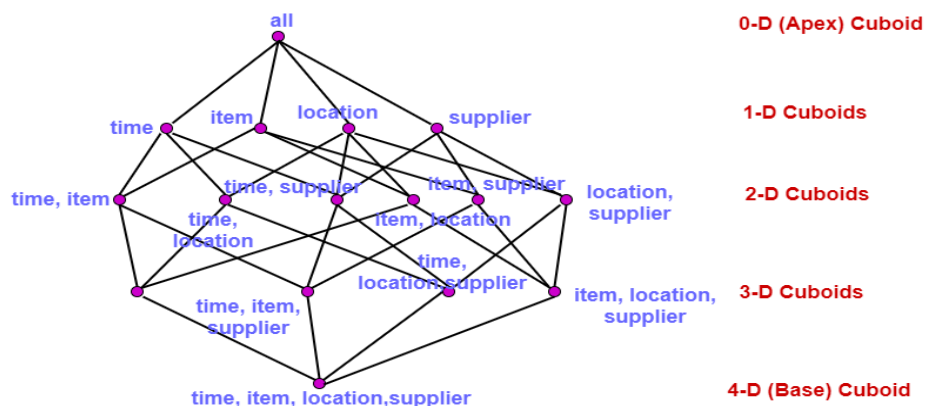


Figure is shown a **4-D data cube** representation of sales data, according to the dimensions time, item, location, and supplier. The measure displayed is dollars sold (in thousands).

The topmost **0-D cuboid**, which holds the highest level of summarization, is known as the apex cuboid. In this example, this is the total sales, or dollars sold, summarized over all four dimensions.

The lattice of cuboid forms a data cube. The figure shows the lattice of cuboids creating 4-D data cubes for the dimension time, item, location, and supplier. Each cuboid represents a different degree of summarization.



➤ What is OLAP (Online Analytical Processing)?

OLAP stands for **On-Line Analytical Processing**. OLAP is a classification of software technology which authorizes analysts, managers, and executives to gain insight into information through fast, consistent, interactive access in a wide variety of possible views of data that has been transformed from raw information to reflect the real dimensionality of the enterprise as understood by the clients.

OLAP implement the multidimensional analysis of business information and support the capability for complex estimations, trend analysis, and sophisticated data modeling. It is rapidly enhancing the essential foundation for Intelligent Solutions containing Business Performance Management, Planning, Budgeting, Forecasting, Financial Documenting, Analysis, Simulation-Models, Knowledge Discovery, and Data Warehouses Reporting. OLAP enables end-clients to perform ad hoc analysis of record in multiple dimensions, providing the insight and understanding they require for better decision making.

Who uses OLAP and Why?

OLAP applications are used by a variety of the functions of an organization.

Finance and accounting:

- Budgeting
- Activity-based costing
- Financial performance analysis
- And financial modeling

Sales and Marketing

- Sales analysis and forecasting
- Market research analysis
- Promotion analysis
- Customer analysis
- Market and customer segmentation

Production

- Production planning
- Defect analysis

OLAP cubes have two main purposes. The first is to provide business users with a data model more intuitive to them than a tabular model. This model is called a Dimensional Model.

The second purpose is to enable fast query response that is usually difficult to achieve using tabular models.

How OLAP Works?

Fundamentally, OLAP has a very simple concept. It pre-calculates most of the queries that are typically very hard to execute over tabular databases, namely aggregation, joining, and grouping. These queries are calculated during a process that is usually called 'building' or 'processing' of the OLAP cube. This process happens overnight, and by the time end users get to work - data will have been updated.

OLAP Guidelines (Dr.E.F.Codd Rule)

Dr E.F. Codd, the "father" of the relational model, has formulated a list of 12 guidelines and requirements as the basis for selecting OLAP systems:



1) Multidimensional Conceptual View: This is the central features of an OLAP system. By needing a multidimensional view, it is possible to carry out methods like slice and dice.

2) Transparency: Make the technology, underlying information repository, computing operations, and the dissimilar nature of source data totally transparent to users. Such transparency helps to improve the efficiency and productivity of the users.

3) Accessibility: It provides access only to the data that is actually required to perform the particular analysis, present a single, coherent, and consistent view to the clients. The OLAP system must map its own logical schema to the heterogeneous physical data stores and perform any necessary transformations. The OLAP operations should be sitting between data sources (e.g., data warehouses) and an OLAP front-end.

4) Consistent Reporting Performance: To make sure that the users do not feel any significant degradation in documenting performance as the number of dimensions or the size of the database increases. That is, the performance of OLAP should not suffer as the number of dimensions is increased. Users must observe consistent run time, response time, or machine utilization every time a given query is run.

5) Client/Server Architecture: Make the server component of OLAP tools sufficiently intelligent that the various clients to be attached with a minimum of effort and integration programming. The server should be capable of mapping and consolidating data between dissimilar databases.

6) Generic Dimensionality: An OLAP method should treat each dimension as equivalent in both its structure and operational capabilities. Additional operational capabilities may be allowed to selected dimensions, but such additional tasks should be grantable to any dimension.

7) Dynamic Sparse Matrix Handling: To adapt the physical schema to the specific analytical model being created and loaded that optimizes sparse matrix handling. When encountering the sparse matrix, the system must be easy to dynamically assume the distribution of the information and adjust the storage and access to obtain and maintain a consistent level of performance.

8) Multiuser Support: OLAP tools must provide concurrent data access, data integrity, and access security.

9) Unrestricted cross-dimensional Operations: It provides the ability for the methods to identify dimensional order and necessarily functions roll-up and drill-down methods within a dimension or across the dimension.

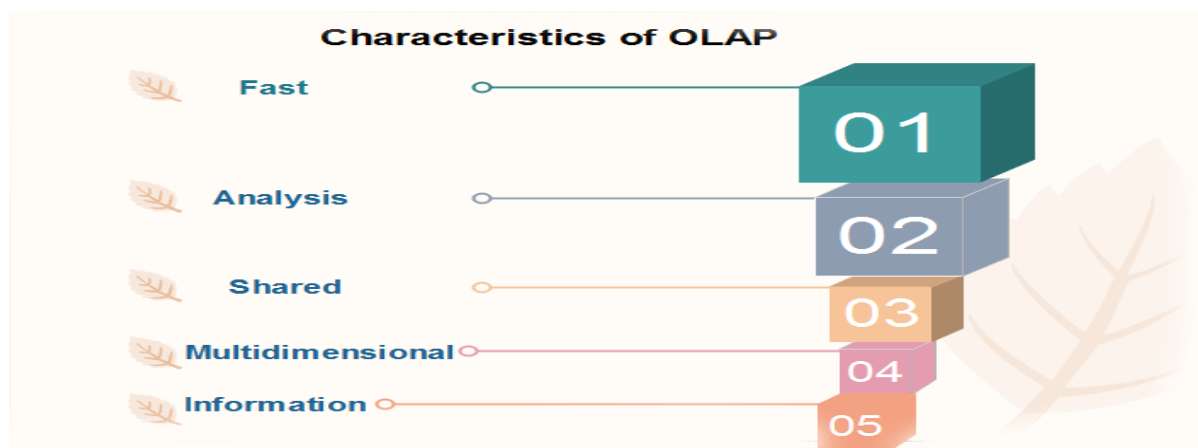
10) Intuitive Data Manipulation: Data Manipulation fundamental the consolidation direction like as reorientation (pivoting), drill-down and roll-up, and another manipulation to be accomplished naturally and precisely via point-and-click and drag and drop methods on the cells of the scientific model. It avoids the use of a menu or multiple trips to a user interface.

11) Flexible Reporting: It implements efficiency to the business clients to organize columns, rows, and cells in a manner that facilitates simple manipulation, analysis, and synthesis of data.

12) Unlimited Dimensions and Aggregation Levels: The number of data dimensions should be unlimited. Each of these common dimensions must allow a practically unlimited number of customer-defined aggregation levels within any given consolidation path.

➤ **Characteristics of OLAP**

In the **FASMI characteristics of OLAP methods**, the term derived from the first letters of the characteristics are:



Fast

It defines which the system targeted to deliver the most feedback to the client within about five seconds, with the elementary analysis taking no more than one second and very few taking more than 20 seconds.

Analysis

It defines which the method can cope with any business logic and statistical analysis that is relevant for the function and the user, keep it easy enough for the target client. Although some preprogramming may be needed we do not think it acceptable if all application definitions have to be allow the user to define new Adhoc calculations as part of the analysis and to document on the data in any desired method, without having to program so we excludes products (like Oracle Discoverer) that do not allow the user to define new Adhoc calculation as part of the analysis and to document on the data in any desired product that do not allow adequate end user-oriented calculation flexibility.

Share

It defines which the system tools all the security requirements for understanding and, if multiple write connection is needed, concurrent update location at an appropriated level, not all functions need customer to write data back, but for the increasing number which does, the system should be able to manage multiple updates in a timely, secure manner.

Multidimensional

This is the basic requirement. OLAP system must provide a multidimensional conceptual view of the data, including full support for hierarchies, as this is certainly the most logical method to analyze business and organizations.

Information

The system should be able to hold all the data needed by the applications. Data sparsity should be handled in an efficient manner.

The main characteristics of OLAP are as follows:

1. **Multidimensional conceptual view:** OLAP systems let business users have a dimensional and logical view of the data in the data warehouse. It helps in carrying slice and dice operations.
2. **Multi-User Support:** Since the OLAP techniques are shared, the OLAP operation should provide normal database operations, containing retrieval, update, adequacy control, integrity, and security.
3. **Accessibility:** OLAP acts as a mediator between data warehouses and front-end. The OLAP operations should be sitting between data sources (e.g., data warehouses) and an OLAP front-end.
4. **Storing OLAP results:** OLAP results are kept separate from data sources.
5. **Uniform documenting performance:** Increasing the number of dimensions or database size should not significantly degrade the reporting performance of the OLAP system.
6. OLAP provides for distinguishing between zero values and missing values so that aggregates are computed correctly.
7. OLAP system should ignore all missing values and compute correct aggregate values.
8. OLAP facilitate interactive query and complex analysis for the users.
9. OLAP allows users to drill down for greater details or roll up for aggregations of metrics along a single business dimension or across multiple dimension.
10. OLAP provides the ability to perform intricate calculations and comparisons.
11. OLAP presents results in a number of meaningful ways, including charts and graphs.

Benefits of OLAP

OLAP holds several benefits for businesses: -

1. OLAP helps managers in decision-making through the multidimensional record views that it is efficient in providing, thus increasing their productivity.
2. OLAP functions are self-sufficient owing to the inherent flexibility support to the organized databases.
3. It facilitates simulation of business models and problems, through extensive management of analysis-capabilities.
4. In conjunction with data warehouse, OLAP can be used to support a reduction in the application backlog, faster data retrieval, and reduction in query drag.

Motivations for using OLAP

1) Understanding and improving sales: For enterprises that have much products and benefit a number of channels for selling the product, OLAP can help in finding the most suitable products and the most famous channels. In some methods, it may be feasible to find the most profitable users. **For example**, considering the telecommunication industry and considering only one product, communication minutes, there is a high amount of record if a company want to analyze the sales of products for every hour of the day (24 hours), difference between weekdays and weekends (2 values) and split regions to which calls are made into 50 region.

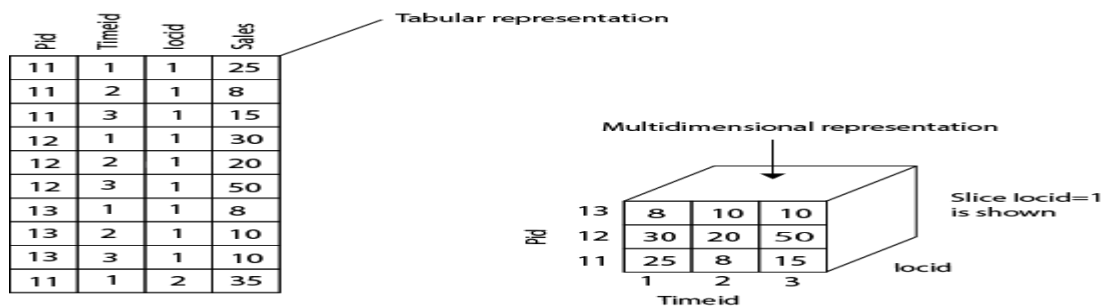
2) Understanding and decreasing costs of doing business: Improving sales is one method of improving a business, the other method is to analyze cost and to control them as much as suitable without affecting sales. OLAP can assist in analyzing the costs related to sales. In some methods, it may also be feasible to identify expenditures which produce a high return on investments (ROI). **For example**, recruiting a top salesperson may contain high costs, but the revenue generated by the salesperson may justify the investment.

➤ What is Multi-Dimensional Data Model?

A multidimensional model views data in the form of a data-cube. A data cube enables data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

The dimensions are the perspectives or entities concerning which an organization keeps records. For example, a shop may create a sales data warehouse to keep records of the store's sales for the dimension time, item, and location. These dimensions allow the save to keep track of things, for example, monthly sales of items and the locations at which the items were sold. Each dimension has a table related to it, called a dimensional table, which describes the dimension further. For example, a dimensional table for an item may contain the attributes item_name, brand, and type.

A multidimensional data model is organized around a central theme, for example, sales. This theme is represented by a fact table. Facts are numerical measures. The fact table contains the names of the facts or measures of the related dimensional tables.



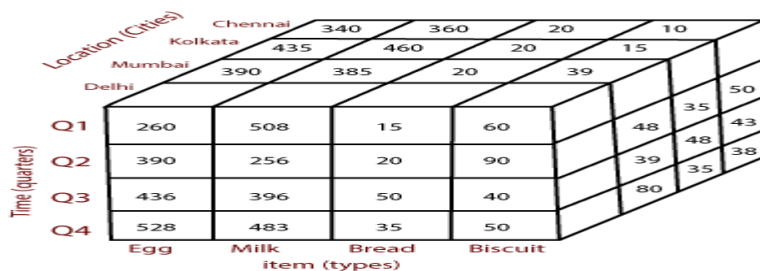
Consider the data of a shop for items sold per quarter in the city of Delhi. The data is shown in the table. In this 2D representation, the sales for Delhi are shown for the time dimension (organized in quarters) and the item dimension (classified according to the types of an item sold). The fact or measure displayed in rupee sold (in thousands).

Location="Delhi"				
Time (quarter)	item (type)			
	Egg	Milk	Bread	Biscuit
Q1	260	508	15	60
Q2	390	256	20	90
Q3	436	396	50	40
Q4	528	483	35	50

Now, if we want to view the sales data with a third dimension, For example, suppose the data according to time and item, as well as the location is considered for the cities Chennai, Kolkata, Mumbai, and Delhi. These 3D data are shown in the table. The 3D data of the table are represented as a series of 2D tables.

	Location="Chennai"				Location="Kolkata"				Location="Mumbai"				Location="Delhi"			
	item				item				item				item			
Time	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit
Q1	340	360	20	10	435	460	20	15	390	385	20	39	260	508	15	60
Q2	490	490	16	50	389	385	45	35	463	366	25	48	390	256	20	90
Q3	680	583	46	43	684	490	39	48	568	594	36	39	436	396	50	40
Q4	535	694	39	38	335	365	83	35	338	484	48	80	528	483	35	50

Conceptually, it may also be represented by the same data in the form of a 3D data cube, as shown in fig:



➤ Data Cube Implementations

In computer programming contexts, a **data cube** (or **datacube**) is a multi-dimensional ("n-D") array of values. Typically, the term datacube is applied in contexts where these arrays are massively larger than the hosting computer's main memory; examples include multi-terabyte/petabyte data warehouses and time series of image data.

The data cube is used to represent data (sometimes called facts) along some dimensions of interest. For example, in OLAP such dimensions could be the subsidiaries a company has, the products the company offers, and time; in this setup, a fact would be a sales event where a particular product has been sold in a particular subsidiary at a particular time. In satellite image timeseries dimensions would be Latitude and Longitude coordinates and time; a fact (sometimes called measure) would be a pixel at a given space and time as taken by the satellite (following some processing that is not of concern here). Even though it is called a *cube* (and the examples provided above happen to be 3-dimensional for brevity), a data cube generally is a multi-dimensional concept which can be 1-dimensional, 2-dimensional, 3-dimensional, or higher-dimensional. In any case, every dimension divides data into groups of cells whereas each cell in the cube represents a single measure of interest. Sometimes cubes hold only few values with the rest being empty, i.e. undefined, sometimes most or all cube coordinates hold a cell value. In the first case such data are called sparse, in the second case they are called dense, although there is no hard delineation between both.

Many high-level computer languages treat data cubes and other large arrays as single entities distinct from their contents. These languages, of which Fortran, APL, IDL, NumPy, PDL, and S-Lang are examples, allow the programmer to manipulate complete film clips and other data en masse with simple expressions derived from linear algebra and vector mathematics. Some languages (such as PDL) distinguish between a list of images and a data cube, while many (such as IDL) do not. Array DBMSs (Database Management Systems) offer a data model which generically supports definition, management, retrieval, and manipulation of n-dimensional datacubes. This database category has been pioneered by the rasdaman system since 1994.

➤ OLAP Operations in the Multidimensional Data Model/Data cube Operations

In the multidimensional model, the records are organized into various dimensions, and each dimension includes multiple levels of abstraction described by concept hierarchies. This organization support users with the flexibility to view data from various perspectives. A number of OLAP data cube operation exist to demonstrate these different views, allowing interactive queries and search of the record at hand. Hence, OLAP supports a user-friendly environment for interactive data analysis.

Consider the OLAP operations which are to be performed on multidimensional data. The figure shows data cubes for sales of a shop. The cube contains the dimensions, location, and time and

item, where the **location** is aggregated with regard to city values, **time** is aggregated with respect to quarters, and an **item** is aggregated with respect to item types.

Roll-Up

The roll-up operation (**also known as drill-up or aggregation operation**) performs aggregation on a data cube, by climbing down concept hierarchies, i.e., dimension reduction. Roll-up is like **zooming-out** on the data cubes. Figure shows the result of roll-up operations performed on the dimension location. The hierarchy for the location is defined as the Order Street, city, province, or state, country. The roll-up operation aggregates the data by ascending the location hierarchy from the level of the city to the level of the country.

When a roll-up is performed by dimensions reduction, one or more dimensions are removed from the cube. For example, consider a sales data cube having two dimensions, location and time. Roll-up may be performed by removing, the time dimensions, appearing in an aggregation of the total sales by location, relatively than by location and by time.

Example

Consider the following cubes illustrating temperature of certain days recorded weekly:

Temperature	64	65	68	69	70	71	72	75	80	81	83	85
Week1	1	0	1	0	1	0	0	0	0	0	1	0
Week2	0	0	0	1	0	0	1	2	0	1	0	0

Consider that we want to set up levels (hot (80-85), mild (70-75), cool (64-69)) in temperature from the above cubes.

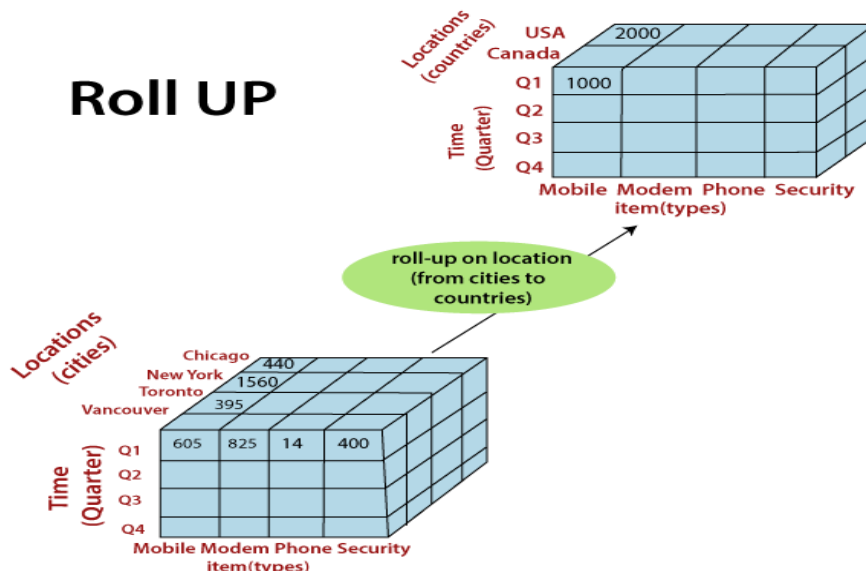
To do this, we have to group column and add up the value according to the concept hierarchies. This operation is known as a roll-up.

By doing this, we contain the following cube:

Temperature	cool	mild	hot
Week1	2	1	1
Week2	2	1	1

The roll-up operation groups the information by levels of temperature.

The following diagram illustrates how roll-up works.



Drill-Down

The drill-down operation (**also called roll-down**) is the reverse operation of **roll-up**. Drill-down is like **zooming-in** on the data cube. It navigates from less detailed record to more detailed data. Drill-down can be performed by either **stepping down** a concept hierarchy for a dimension or adding additional dimensions.

Figure shows a drill-down operation performed on the dimension time by stepping down a concept hierarchy which is defined as day, month, quarter, and year. Drill-down appears by descending the time hierarchy from the level of the quarter to a more detailed level of the month.

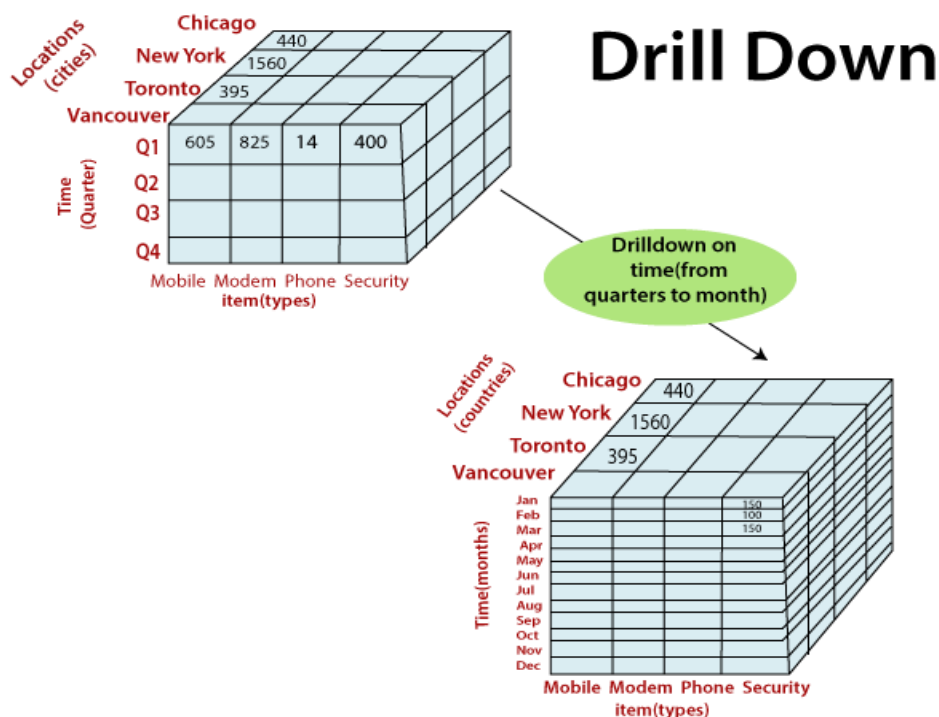
Because a drill-down adds more details to the given data, it can also be performed by adding a new dimension to a cube. For example, a drill-down on the central cubes of the figure can occur by introducing an additional dimension, such as a customer group.

Example

Drill-down adds more details to the given 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

The following diagram illustrates how Drill-down works.



Slice

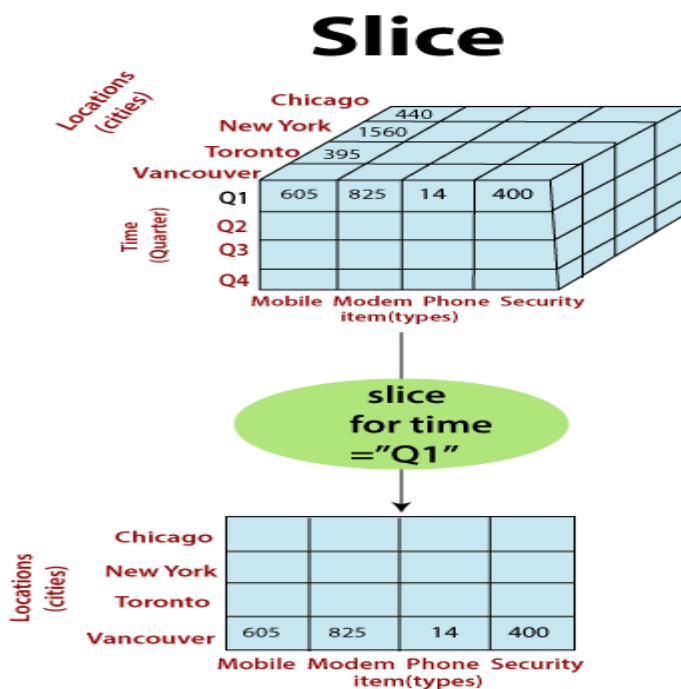
A **slice** is a subset of the cubes corresponding to a single value for one or more members of the dimension. For example, a slice operation is executed when the customer wants a selection on one

dimension of a three-dimensional cube resulting in a two-dimensional site. So, the Slice operations perform a selection on one dimension of the given cube, thus resulting in a subcube.

For example, if we make the selection, temperature=cool we will obtain the following cube:

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 11	0
Day 12	0
Day 13	0
Day 14	0

The following diagram illustrates how Slice works.



Here Slice is functioning for the dimensions "time" using the criterion time = "Q1".

It will form a new sub-cube by selecting one or more dimensions.

Dice

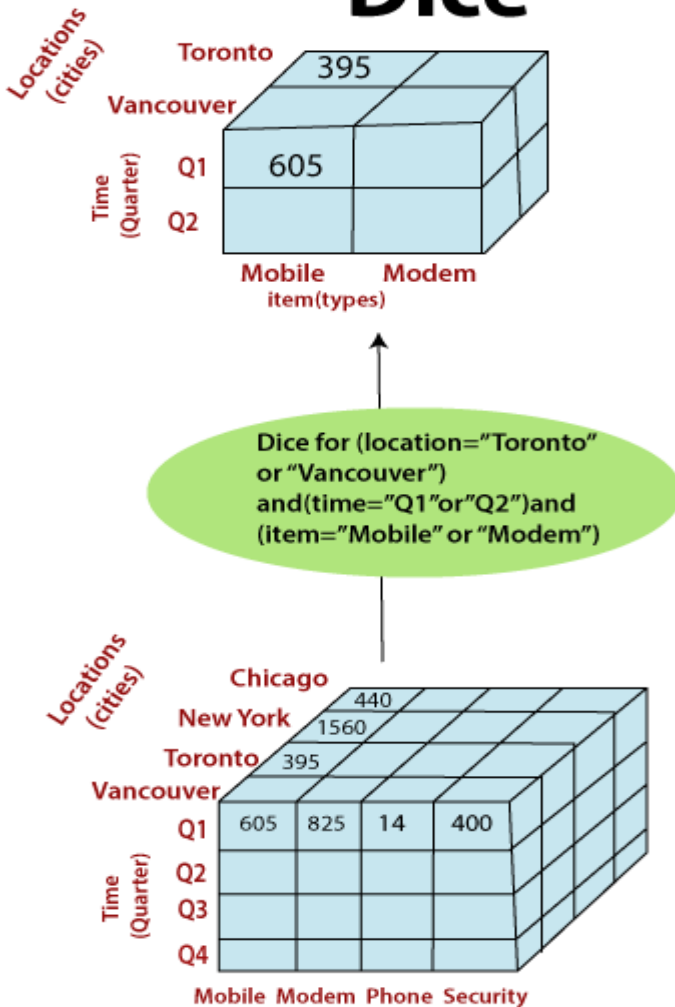
The dice operation describes a sub cube by operating a selection on two or more dimension.

For example, Implement the selection (time = day 3 OR time = day 4) AND (temperature = cool OR temperature = hot) to the original cubes we get the following subcube (still two-dimensional)

Temperature	cool	hot
Day 3	0	1
Day 4	0	0

Consider the following diagram, which shows the dice operations.

Dice

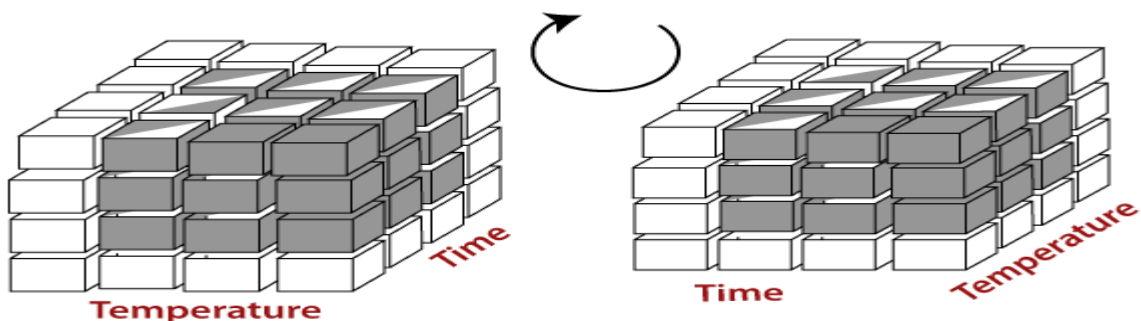


The dice operation on the cubes based on the following selection criteria involves three dimensions.

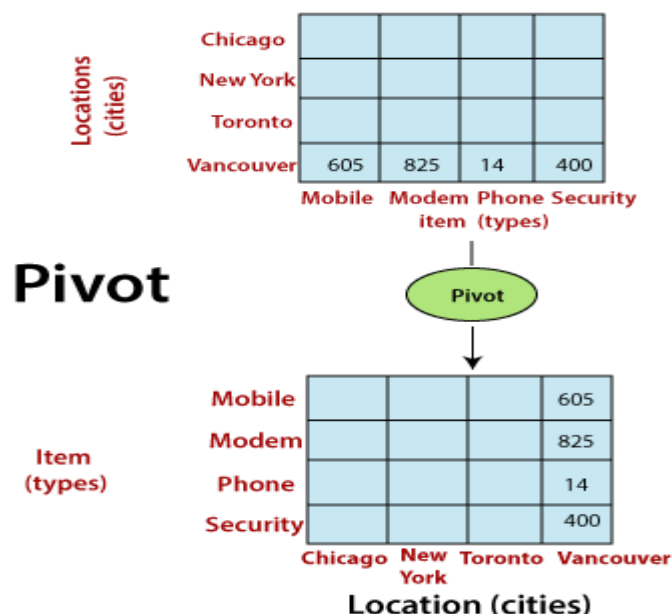
- (location = "Toronto" or "Vancouver")
- (time = "Q1" or "Q2")
- (item = "Mobile" or "Modem")

Pivot

The pivot operation is also called a rotation. Pivot is a visualization operations which rotates the data axes in view to provide an alternative presentation of the data. It may contain swapping the rows and columns or moving one of the row-dimensions into the column dimensions.



Consider the following diagram, which shows the pivot operation.



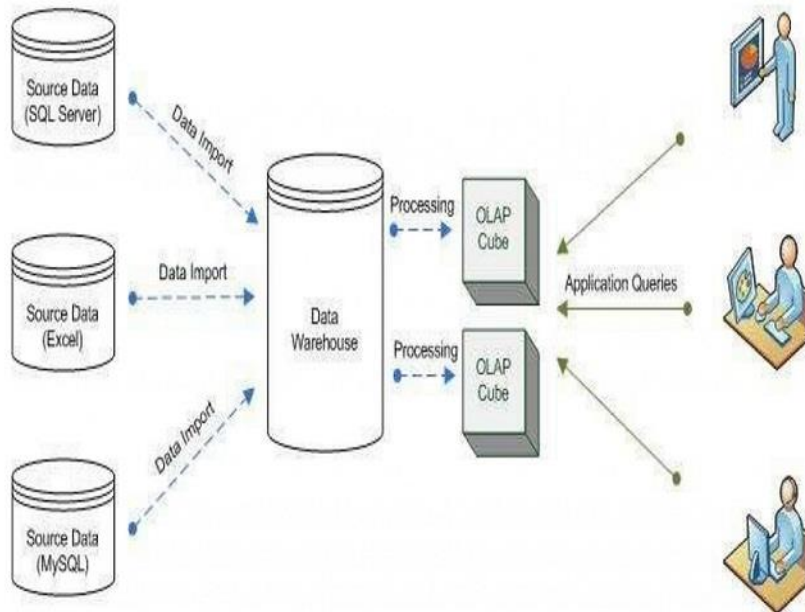
Other OLAP Operations

Executes queries containing more than one fact table. The drill-through operations make use of relational SQL facilitates to drill through the bottom level of a data cubes down to its back-end relational tables.

Other OLAP operations may contain ranking the top-N or bottom-N elements in lists, as well as calculate moving average, growth rates, and interests, internal rates of returns, depreciation, currency conversions, and statistical tasks.

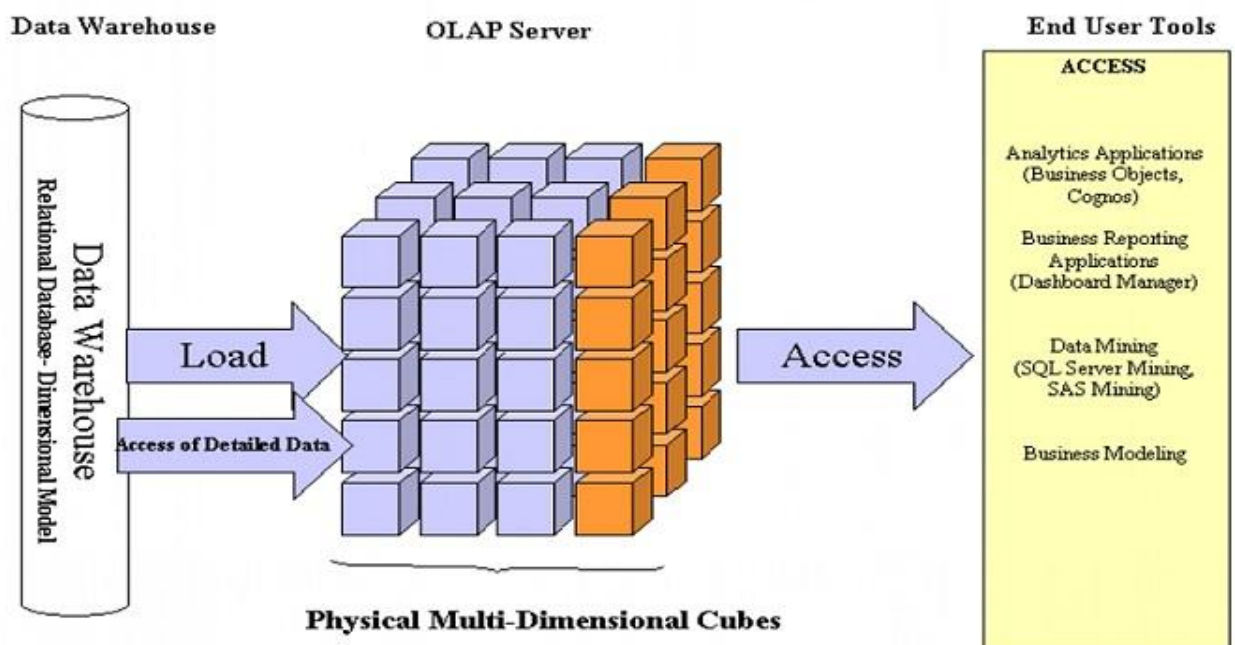
➤ Overview on OLAP Software

- Some popular OLAP server software programs include:
 - Oracle Express Server
 - Hyperion Solutions Essbase
- OLAP processing is often used for **data mining**.
- OLAP products are **typically designed for multiple-user environments**, with the cost of the software based on the number of users.

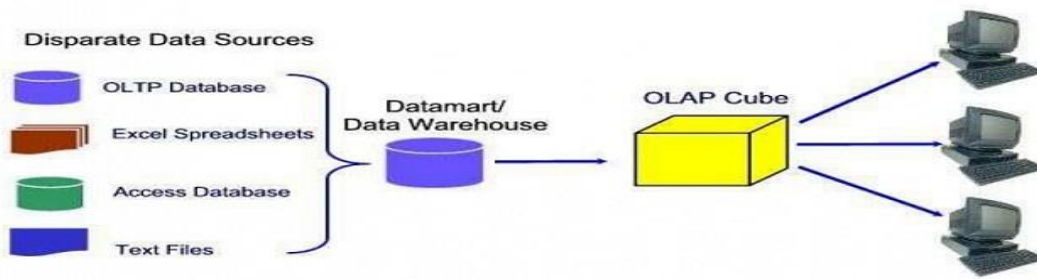


OLAP offers analytical modeling capabilities, containing a calculation engine for determining ratios, variance, etc. and for computing measures across various dimensions. It can generate summarization, aggregation, and hierarchies at each granularity level and at every dimensions intersection. OLAP also provide functional models for forecasting, trend analysis, and statistical analysis. In this context, the OLAP engine is a powerful data analysis tool.

OLAP CUBE



For example, a company might wish to summarize financial data by product, by time-period, and by city to compare actual and budget expenses. Product, time, city and scenario (actual and budget) are the data's dimensions



Advantages of OLAP Tools

- It helps us to analyze and modify reports at a much faster pace since the data is used from in-memory from data cubes rather than the data warehouse.
- OLAP tools like MicroStrategy have intelligent secure Cube data share which allows data to be shared in a secure way.
- Another benefit is the consistency of information and calculations. It means the speed with which data is shared in OLAP servers does not interfere with reporting and the reporting always remains consistent.
- The multidimensional presentation using OLAP tools helps in better understanding of relationships which was not present previously.
- Another popular scenario is the “What if” scenarios of OLAP software. They are made eminently more possible by multidimensional processing of OLAP tools.
- We can apply security restrictions on users and objects using OLAP tools.
- It creates a single platform for planning, forecasting, reporting, and analysis.

Drawbacks of Traditional OLAP

The traditional OLAP had its drawbacks. Couple of them is as follows. Disadvantages of traditional OLAP:

- Pre-modelling is a must in traditional OLAP tools which is a time-consuming process.
- Great dependence on IT: In this case, a user is a business person but he should have good knowledge of IT. The traditional OLAP tools require heavy involvement of IT technicians and people who have good business expertise along with IT.

➤ Data Mining and its Applications:

➤ Introduction

Data mining is one of the most useful techniques that help entrepreneurs, researchers, and individuals to extract valuable information from huge sets of data. Data mining is also called **Knowledge Discovery in Database (KDD)**. The knowledge discovery process includes Data cleaning, Data integration, Data selection, Data transformation, Data mining, Pattern evaluation, and Knowledge presentation.

Our Data mining tutorial includes all topics of Data mining such as applications, Data mining vs Machine learning, Data mining tools, Social Media Data mining, Data mining techniques, Clustering in data mining, Challenges in Data mining, etc.

➤ What is Data Mining?

The process of extracting information to identify patterns, trends, and useful data that would allow the business to take the data-driven decision from huge sets of data is called Data Mining.

In other words, we can say that Data Mining is the process of investigating hidden patterns of information to various perspectives for categorization into useful data, which is collected and assembled in particular areas such as data warehouses, efficient analysis, data mining algorithm, helping decision making and other data requirement to eventually cost-cutting and generating revenue.

Data mining is the act of automatically searching for large stores of information to find trends and patterns that go beyond simple analysis procedures. Data mining utilizes complex mathematical algorithms for data segments and evaluates the probability of future events. Data Mining is also called Knowledge Discovery of Data (KDD).

Data Mining is a process used by organizations to extract specific data from huge databases to solve business problems. It primarily turns raw data into useful information.

Data Mining is similar to Data Science carried out by a person, in a specific situation, on a particular data set, with an objective. This process includes various types of services such as text mining, web mining, audio and video mining, pictorial data mining, and social media mining. It is done through software that is simple or highly specific. By outsourcing data mining, all the work can be done faster with low operation costs. Specialized firms can also use new technologies to collect data that is impossible to locate manually. There are tonnes of information available on various platforms, but very little knowledge is accessible. The biggest challenge is to analyze the data to extract important information that can be used to solve a problem or for company development. There are many powerful instruments and techniques available to mine data and find better insight from it.



Types of Data Mining

Data mining can be performed on the following types of data:

Relational Database:

A relational database is a collection of multiple data sets formally organized by tables, records, and columns from which data can be accessed in various ways without having to recognize the database tables. Tables convey and share information, which facilitates data searchability, reporting, and organization.

Data warehouses:

A Data Warehouse is the technology that collects the data from various sources within the organization to provide meaningful business insights. The huge amount of data comes from multiple places such as Marketing and Finance. The extracted data is utilized for analytical purposes and helps in decision-making for a business organization. The data warehouse is designed for the analysis of data rather than transaction processing.

Data Repositories:

The Data Repository generally refers to a destination for data storage. However, many IT professionals utilize the term more clearly to refer to a specific kind of setup within an IT structure. For example, a group of databases, where an organization has kept various kinds of information.

Object-Relational Database:

A combination of an object-oriented database model and relational database model is called an object-relational model. It supports Classes, Objects, Inheritance, etc.

One of the primary objectives of the Object-relational data model is to close the gap between the Relational database and the object-oriented model practices frequently utilized in many programming languages, for example, C++, Java, C#, and so on.

Transactional Database:

A transactional database refers to a database management system (DBMS) that has the potential to undo a database transaction if it is not performed appropriately. Even though this was a

unique capability a very long while back, today, most of the relational database systems support transactional database activities.

Advantages of Data Mining

- The Data Mining technique enables organizations to obtain knowledge-based data.
- Data mining enables organizations to make lucrative modifications in operation and production.
- Compared with other statistical data applications, data mining is a cost-efficient.
- Data Mining helps the decision-making process of an organization.
- It Facilitates the automated discovery of hidden patterns as well as the prediction of trends and behaviors.
- It can be induced in the new system as well as the existing platforms.
- It is a quick process that makes it easy for new users to analyze enormous amounts of data in a short time.

Disadvantages of Data Mining

- There is a probability that the organizations may sell useful data of customers to other organizations for money. As per the report, American Express has sold credit card purchases of their customers to other organizations.
- Many data mining analytics software is difficult to operate and needs advance training to work on.
- Different data mining instruments operate in distinct ways due to the different algorithms used in their design. Therefore, the selection of the right data mining tools is a very challenging task.
- The data mining techniques are not precise, so that it may lead to severe consequences in certain conditions.

Data Mining Applications

Data Mining is primarily used by organizations with intense consumer demands- Retail, Communication, Financial, marketing company, determine price, consumer preferences, product positioning, and impact on sales, customer satisfaction,



and corporate profits. Data mining enables a retailer to use point-of-sale records of customer purchases to develop products and promotions that help the organization to attract the customer.

These are the following areas where data mining is widely used:

Data Mining in Healthcare:

Data mining in healthcare has excellent potential to improve the health system. It uses data and analytics for better insights and to identify best practices that will enhance health care services and reduce costs. Analysts use data mining approaches such as Machine learning, Multi-dimensional database, Data visualization, Soft computing, and statistics. Data Mining can be used to forecast patients in each category. The procedures ensure that the patients get intensive care at the right place and at the right time. Data mining also enables healthcare insurers to recognize fraud and abuse.

Data Mining in Market Basket Analysis:

Market basket analysis is a modeling method based on a hypothesis. If you buy a specific group of products, then you are more likely to buy another group of products. This technique may enable the retailer to understand the purchase behavior of a buyer. This data may assist the retailer in understanding the requirements of the buyer and altering the store's layout accordingly. Using a different analytical comparison of results between various stores, between customers in different demographic groups can be done.

Data mining in Education:

Education data mining is a newly emerging field, concerned with developing techniques that explore knowledge from the data generated from educational Environments. EDM objectives are recognized as affirming student's future learning behavior, studying the impact of educational support, and promoting learning science. An organization can use data mining to make precise decisions and also to predict the results of the student. With the results, the institution can concentrate on what to teach and how to teach.

Data Mining in Manufacturing Engineering:

Knowledge is the best asset possessed by a manufacturing company. Data mining tools can be beneficial to find patterns in a complex manufacturing process. Data mining can be used in system-level designing to obtain the relationships between product architecture, product portfolio, and data needs of the customers. It can also be used to forecast the product development period, cost, and expectations among the other tasks.

Data Mining in CRM (Customer Relationship Management):

Customer Relationship Management (CRM) is all about obtaining and holding Customers, also enhancing customer loyalty and implementing customer-oriented strategies. To get a decent relationship with the customer, a business organization needs to collect data and analyze the data. With data mining technologies, the collected data can be used for analytics.

Data Mining in Fraud detection:

Billions of dollars are lost to the action of frauds. Traditional methods of fraud detection are a little bit time consuming and sophisticated. Data mining provides meaningful patterns and turning data into information. An ideal fraud detection system should protect the data of all the users. Supervised methods consist of a collection of sample records, and these records are classified as fraudulent or non-fraudulent. A model is constructed using this data, and the technique is made to identify whether the document is fraudulent or not.

Data Mining in Lie Detection:

Apprehending a criminal is not a big deal, but bringing out the truth from him is a very challenging task. Law enforcement may use data mining techniques to investigate offenses, monitor suspected terrorist communications, etc. This technique includes text mining also, and it seeks meaningful patterns in data, which is usually unstructured text. The information collected from the previous investigations is compared, and a model for lie detection is constructed.

Data Mining Financial Banking:

The Digitalization of the banking system is supposed to generate an enormous amount of data with every new transaction. The data mining technique can help bankers by solving business-related problems in banking and finance by identifying trends, casualties, and correlations in business information and market costs that are not instantly evident to managers or executives because the data volume is too large or are produced too rapidly on the screen by experts. The manager may find these data for better targeting, acquiring, retaining, segmenting, and maintain a profitable customer.

Challenges of Implementation in Data mining

Although data mining is very powerful, it faces many challenges during its execution. Various challenges could be related to performance, data, methods, and techniques, etc. The process of data mining becomes effective when the challenges or problems are correctly recognized and adequately resolved.



Incomplete and noisy data:

The process of extracting useful data from large volumes of data is data mining. The data in the real-world is heterogeneous, incomplete, and noisy. Data in huge quantities will usually be inaccurate or unreliable. These problems may occur due to data measuring instrument or because of human errors. Suppose a retail chain collects phone numbers of customers who spend more than \$ 500, and the accounting employees put the information into their system. The person may make a digit mistake when entering the phone number, which results in incorrect data. Even some customers may not be willing to disclose their phone numbers, which results in incomplete data. The data could get changed due to human or system error. All these consequences (noisy and incomplete data) makes data mining challenging.

Data Distribution:

Real-worlds data is usually stored on various platforms in a distributed computing environment. It might be in a database, individual systems, or even on the internet. Practically, It is a quite tough task to make all the data to a centralized data repository mainly due to organizational and technical concerns. For example, various regional offices may have their servers to store their data. It is not feasible to store, all the data from all the offices on a central server. Therefore, data mining requires the development of tools and algorithms that allow the mining of distributed data.

Complex Data:

Real-world data is heterogeneous, and it could be multimedia data, including audio and video, images, complex data, spatial data, time series, and so on. Managing these various types of data and extracting useful information is a tough task. Most of the time, new technologies, new tools, and methodologies would have to be refined to obtain specific information.

Performance:

The data mining system's performance relies primarily on the efficiency of algorithms and techniques used. If the designed algorithm and techniques are not up to the mark, then the efficiency of the data mining process will be affected adversely.

Data Privacy and Security:

Data mining usually leads to serious issues in terms of data security, governance, and privacy. For example, if a retailer analyzes the details of the purchased items, then it reveals data about buying habits and preferences of the customers without their permission.

Data Visualization:

In data mining, data visualization is a very important process because it is the primary method that shows the output to the user in a presentable way. The extracted data should convey the exact meaning of what it intends to express. But many times, representing the information to the end-user in a precise and easy way is difficult. The input data and the output information being complicated, very efficient, and successful data visualization processes need to be implemented to make it successful.

There are many more challenges in data mining in addition to the problems above-mentioned. More problems are disclosed as the actual data mining process begins, and the success of data mining relies on getting rid of all these difficulties.

Prerequisites

Before learning the concepts of Data Mining, you should have a basic understanding of Statistics, Database Knowledge, and Basic programming language.

Audience

Our Data Mining Tutorial is prepared for all beginners or computer science graduates to help them learn the basics to advanced techniques related to data mining.

Problems

We assure you that you will not find any difficulty while learning our Data Mining tutorial. But if there is any mistake in this tutorial, kindly post the problem or error in the contact form so that we can improve it.

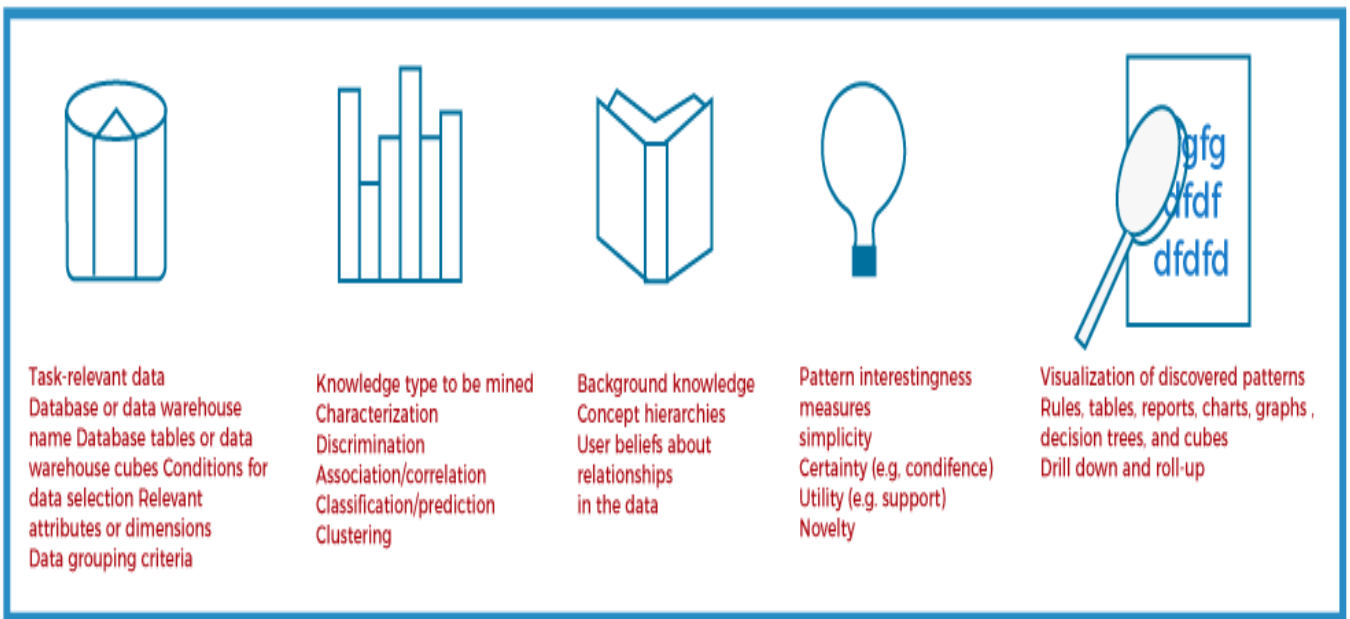
➤ Data Mining Tasks

A data mining task can be specified in the form of a data mining query, which is input to the data mining system. A data mining query is defined in terms of data mining task primitives. These primitives allow the user to interactively communicate with the data mining system during discovery to direct the mining process or examine the findings from different angles or depths. The data mining primitives specify the following,

1. Set of task-relevant data to be mined.
2. Kind of knowledge to be mined.
3. Background knowledge to be used in the discovery process.
4. Interestingness measures and thresholds for pattern evaluation.
5. Representation for visualizing the discovered patterns.

A data mining query language can be designed to incorporate these primitives, allowing users to interact with data mining systems flexibly. Having a data mining query language provides a foundation on which user-friendly graphical interfaces can be built.

Designing a comprehensive data mining language is challenging because data mining covers a wide spectrum of tasks, from data characterization to evolution analysis. Each task has different requirements. The design of an effective data mining query language requires a deep understanding of the power, limitation, and underlying mechanisms of the various kinds of data mining tasks. This facilitates a data mining system's communication with other information systems and integrates with the overall information processing environment.



List of Data Mining Task Primitives

A data mining query is defined in terms of the following primitives, such as:

1. The set of task-relevant data to be mined

This specifies the portions of the database or the set of data in which the user is interested. This includes the database attributes or data warehouse dimensions of interest (the relevant attributes or dimensions).

In a relational database, the set of task-relevant data can be collected via a relational query involving operations like selection, projection, join, and aggregation.

The data collection process results in a new data relational called the initial data relation. The initial data relation can be ordered or grouped according to the conditions specified in the query. This data retrieval can be thought of as a subtask of the data mining task.

This initial relation may or may not correspond to physical relation in the database. Since virtual relations are called Views in the field of databases, the set of task-relevant data for data mining is called a minable view.

2. The kind of knowledge to be mined

This specifies the data mining functions to be performed, such as characterization, discrimination, association or correlation analysis, classification, prediction, clustering, outlier analysis, or evolution analysis.

3. The background knowledge to be used in the discovery process

This knowledge about the domain to be mined is useful for guiding the knowledge discovery process and evaluating the patterns found. Concept hierarchies are a popular form of background knowledge, which allows data to be mined at multiple levels of abstraction.

Concept hierarchy defines a sequence of mappings from low-level concepts to higher-level, more general concepts.

- Rolling Up - Generalization of data: Allow to view data at more meaningful and explicit abstractions and makes it easier to understand. It compresses the data, and it would require fewer input/output operations.
- Drilling Down - Specialization of data: Concept values replaced by lower-level concepts. Based on different user viewpoints, there may be more than one concept hierarchy for a given attribute or dimension.

An example of a concept hierarchy for the attribute (or dimension) age is shown below. User beliefs regarding relationships in the data are another form of background knowledge.

4. The interestingness measures and thresholds for pattern evaluation

Different kinds of knowledge may have different interesting measures. They may be used to guide the mining process or, after discovery, to evaluate the discovered patterns. For example, interesting measures for association rules include support and confidence. Rules whose support and confidence values are below user-specified thresholds are considered uninteresting.

- Simplicity: A factor contributing to the interestingness of a pattern is the pattern's overall simplicity for human comprehension. For example, the more complex the structure of a rule is, the more difficult it is to interpret, and hence, the less interesting it is likely to be. Objective measures of pattern simplicity can be viewed as functions of the pattern structure, defined in terms of the pattern size in bits or the number of attributes or operators appearing in the pattern.
- Certainty (Confidence): Each discovered pattern should have a measure of certainty associated with it that assesses the validity or "trustworthiness" of the pattern. A certainty measure for association rules of the form "A =>B" where A and B are sets of items is confidence. Confidence is a certainty measure. Given a set of task-relevant data tuples, the confidence of "A => B" is defined as
- Confidence (A=>B) = # tuples containing both A and B / # tuples containing A
- Utility (Support): The potential usefulness of a pattern is a factor defining its interestingness. It can be estimated by a utility function, such as support. The support of an association pattern refers to the percentage of task-relevant data tuples (or

transactions) for which the pattern is true.

Utility (support): usefulness of a pattern

Support ($A \Rightarrow B$) = # tuples containing both A and B / total #of tuples

- Novelty: Novel patterns are those that contribute new information or increased performance to the given pattern set. For example $\rightarrow$ A data exception. Another strategy for detecting novelty is to remove redundant patterns.

5. The expected representation for visualizing the discovered patterns

This refers to the form in which discovered patterns are to be displayed, which may include rules, tables, cross tabs, charts, graphs, decision trees, cubes, or other visual representations.

Users must be able to specify the forms of presentation to be used for displaying the discovered patterns. Some representation forms may be better suited than others for particular kinds of knowledge.

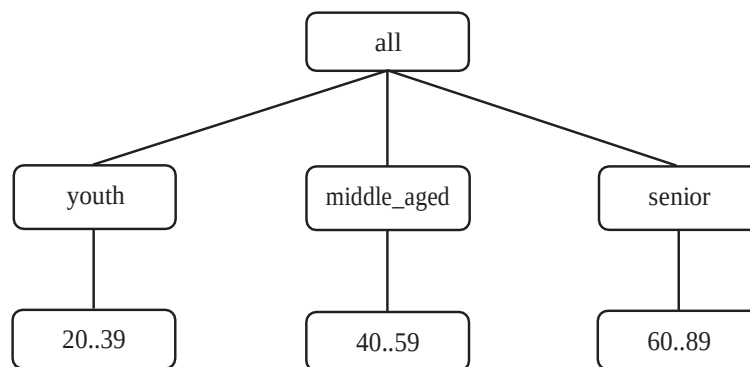
For example, generalized relations and their corresponding cross tabs or pie/bar charts are good for presenting characteristic descriptions, whereas decision trees are common for classification.

This facilitates a data mining system's communication with other information systems and its integration with the overall information processing environment.

Designing a comprehensive data mining language is challenging because data mining covers a wide spectrum of tasks, from data characterization to evolution analysis. Each task has different requirements. The design of an effective data mining query language requires a deep understanding of the power, limitation, and underlying mechanisms of the various kinds of data mining tasks.

Example of Data Mining Task Primitives

Suppose, as a marketing manager of AllElectronics, you would like to classify customers based on their buying patterns. You are especially interested in those customers whose salary is no less than \$40,000 and who have bought more than \$1,000 worth of items, each of which is



priced at no less than \$100.

There are several proposals on data mining languages and standards. In this book, we use a data mining query language known as **DMQL** (Data Mining Query Language), which

was designed as a teaching tool, based on the above primitives. Examples of its use to specify data mining queries appear throughout this book. The language adopts an SQL-like syntax, so that it can easily be integrated with the relational query language, SQL. Let's look at how it can be used to specify a data mining task.

In particular, you are interested in the customer's age, income, the types of items purchased, the purchase location, and where the items were made. You would like to view the resulting classification in the form of rules. This data mining query is expressed in DMQL3 as follows, where each line of the query has been enumerated to aid in our discussion.

1. use database AllElectronics_db
2. use hierarchy location_hierarchy for T.branch, age_hierarchy for C.age
3. mine classification as promising_customers
4. in relevance to C.age, C.income, I.type, I.place_made, T.branch
5. from customer C, an item I, transaction T
6. where I.item_ID = T.item_ID and C.cust_ID = T.cust_ID and C.income $\geq$ 40,000 and I.price $\geq$ 100
7. group by T.cust_ID

➤ **Challenges of Data Mining**

Nowadays Data Mining and knowledge discovery are evolving a crucial technology for business and researchers in many domains. Data Mining is developing into established and trusted discipline, many still pending challenges have to be solved.

Some of these challenges are given below.

1. **Security and Social Challenges**

Decision-Making strategies are done through data collection-sharing, so it requires considerable security. Private information about individuals and sensitive information are collected for customers profiles, user behaviour pattern understanding. Illegal access to information and the confidential nature of information becoming an important issue.

2. **User Interface:**

The knowledge discovered is discovered using data mining tools is useful only if it is interesting and above all understandable by the user. From good visualization interpretation of data, mining results can be eased and helps better understand their requirements. To obtain good visualization many research is carried out for big data sets that display and manipulate mined knowledge.

(i) Mining based on Level of Abstraction: Data Mining process needs to be collaborative because it allows users to concentrate on pattern finding, presenting and optimizing requests for data mining based on returned results

(ii) Integration of Background Knowledge: Previous information may be used to express discovered patterns to direct the exploration processes and to express discovered patterns.

3. Mining Methodology Challenges:

These challenges are related to data mining approaches and their limitations. Mining approaches that cause the problem are:

- (i) Versatility of the mining approaches,
- (ii) Diversity of data available,
- (iii) Dimensionality of the domain,
- (iv) Control and handling of noise in data, etc.

Different approaches may implement differently based upon data consideration. Some algorithms require noise-free data. Most data sets contain exceptions, invalid or incomplete information lead to complication in the analysis process and some cases compromise the precision of the results.

4. Complex Data:

Real-world data is heterogeneous and it could be multimedia data containing images, audio and video, complex data, temporal data, spatial data, time series, natural language text etc. It is difficult to handle these various kinds of data and extract the required information. New tools and methodologies are developing to extract relevant information.

(i) **Complex data types:** The database can include complex data elements, objects with graphical data, spatial data, and temporal data. Mining all these kinds of data is not practical to be done one device.

(ii) **Mining from Varied Sources:** The data is gathered from different sources on Network. The data source may be of different kinds depending on how they are stored such as structured, semi-structured or unstructured.

5. Performance:

The performance of the data mining system depends on the efficiency of algorithms and techniques are using. The algorithms and techniques designed are not up to the mark lead to affect the performance of the data mining process.

(i) **Efficiency and Scalability of the Algorithms:** The data mining algorithm must be efficient and scalable to extract information from huge amounts of data in the database.

(ii) **Improvement of Mining Algorithms:** Factors such as the enormous size of the database, the entire data flow and the difficulty of data mining approaches inspire the creation of parallel & distributed data mining algorithms.

➤ Which Technologies Are Used For data Mining

As a highly application-driven domain, data mining has incorporated many techniques from other domains such as statistics, machine learning, pattern recognition, database and data warehouse systems, information retrieval, visualization, algorithms, high-performance computing, and many application domains. The interdisciplinary nature of data mining research and development contributes significantly to the success of data mining and its

extensive applications. In this section, we give examples of several disciplines that strongly influence the development of data mining methods.

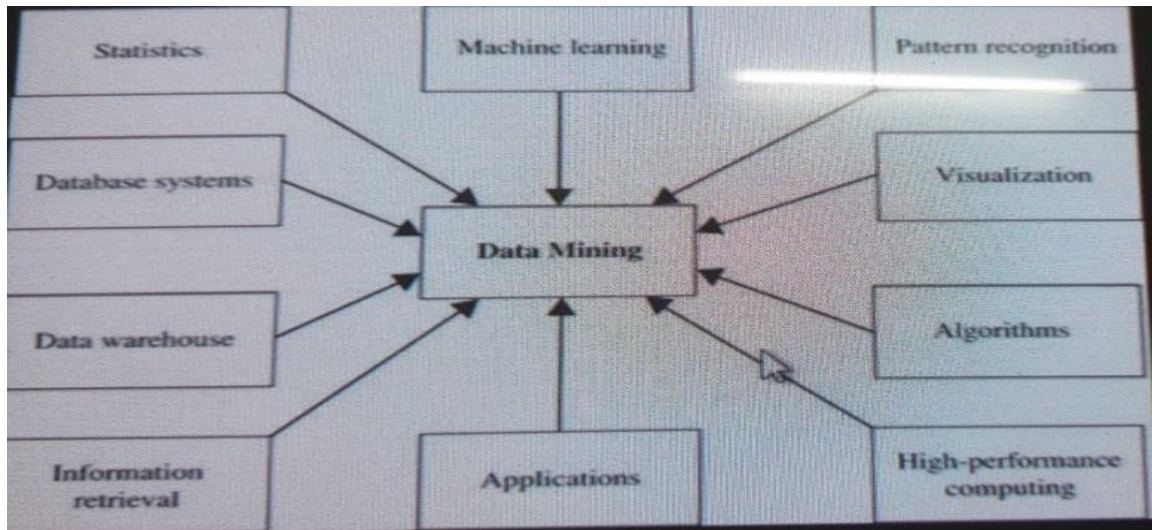


FIGURE 1.11 Data mining adopts techniques from many domains.

Statistics

Statistics studies the collection, analysis, interpretation or explanation, and presentation of data. Data mining has an inherent connection with statistics.

A **statistical model** is a set of mathematical functions that describe the behavior of the objects in a target class in terms of random variables and their associated probability distributions. Statistical models are widely used to model data and data classes. For example, in data mining tasks like data characterization and classification, statistical models of target classes can be built. In other words, such statistical models can be the outcome of a data mining task. Alternatively, data mining tasks can be built on top of statistical models. For example, we can use statistics to model noise and missing data values. Then, when mining patterns in a large data set, the data mining process can use the model to help identify and handle noisy or missing values in the data.

Statistics research develops tools for prediction and forecasting using data and statistical models. Statistical methods can be used to summarize or describe a collection of data. Basic **statistical descriptions** of data are introduced in [Chapter 2](#). Statistics is useful for mining various patterns from data as well as for understanding the underlying mechanisms generating and affecting the patterns. **Inferential statistics** (or **predictive statistics**) models data in a way that accounts for randomness and uncertainty in the observations and is used to draw inferences about the process or population under investigation.

Statistical methods can also be used to verify data mining results. For example, after a classification or prediction model is mined, the model should be verified by statistical

hypothesis testing. A statistical hypothesis test (sometimes called confirmatory data analysis) makes statistical decisions using experimental data. A result is called statistically significant if it is unlikely to have occurred by chance. If the classification or prediction model holds true, then the descriptive statistics of the model increases the soundness of the model.

Applying statistical methods in data mining is far from trivial. Often, a serious challenge is how to scale up a statistical method over a large data set. Many statistical methods have high complexity in computation. When such methods are applied on large data sets that are also distributed on multiple logical or physical sites, algorithms should be carefully designed and tuned to reduce the computational cost. This challenge becomes even tougher for online applications, such as online query suggestions in search engines, where data mining is required to continuously handle fast, real-time data streams.

- **Machine learning** investigates how computers can learn (or improve their performance) based on data. A main research area is for computer programs to *automatically* learn to recognize complex patterns and make intelligent decisions based on data. For example, a typical machine learning problem is to program a computer so that it can automatically recognize handwritten postal codes on mail after learning from a set of examples.

Machine learning is a fast-growing discipline. Here, we illustrate classic problems in machine learning that are highly related to data mining.

- **Supervised learning** is basically a synonym for classification. The supervision in the learning comes from the labeled examples in the training data set. For example, in the postal code recognition problem, a set of handwritten postal code images and their corresponding machine-readable translations are used as the training examples, which supervise the learning of the classification model.
- **Unsupervised learning** is essentially a synonym for clustering. The learning process is unsupervised since the input examples are not class labeled. Typically, we may use clustering to discover classes within the data. For example, an unsupervised learning method can take, as input, a set of images of handwritten digits. Suppose that it finds 10 clusters of data. These clusters may correspond to the 10 distinct digits of 0 to 9, respectively. However, since the training data are not labeled, the learned model cannot tell us the semantic meaning of the clusters found.
- **Semi-supervised learning** is a class of machine learning techniques that make use of both labeled and unlabeled examples when learning a model. In one approach, labeled examples are used to learn class models and unlabeled examples are used to refine the boundaries between classes. For a two-class problem, we can think of the set of examples belonging to one class as the positive examples and those belonging to the other class as the negative examples. ■ **Active learning** is a machine learning approach that lets users play an active role in the learning process. An active learning approach can ask a user (e.g., a domain expert) to label an example, which may be from a set of unlabeled examples or synthesized by the learning

program. The goal is to optimize the model quality by actively acquiring knowledge from human users, given a constraint on how many examples they can be asked to label.

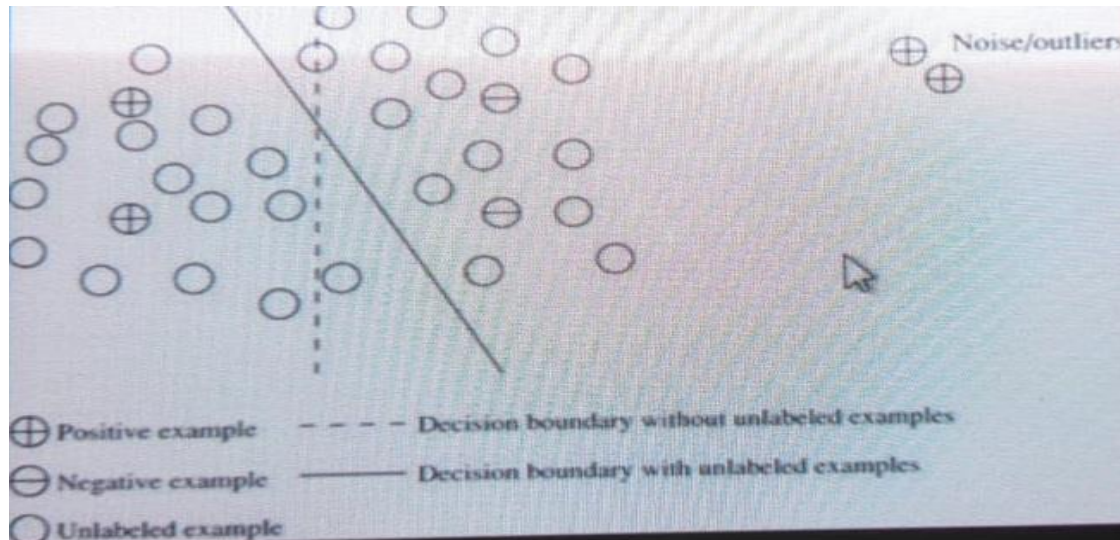


FIGURE 1.12 Semi-supervised learning.

You can see there are many similarities between data mining and machine learning. For classification and clustering tasks, machine learning research often focuses on the accuracy of the model. In addition to accuracy, data mining research places strong emphasis on the efficiency and scalability of mining methods on large data sets, as well as on ways to handle complex types of data and explore new, alternative methods.

Database Systems and Data Warehouses

Database systems research focuses on the creation, maintenance, and use of databases for organizations and end-users. Particularly, database systems researchers have established highly recognized principles in data models, query languages, query processing and optimization methods, data storage, and indexing and accessing methods. Database systems are often well known for their high scalability in processing very large, relatively structured data sets.

Many data mining tasks need to handle large data sets or even real-time, fast streaming data. Therefore, data mining can make good use of scalable database technologies to achieve high efficiency and scalability on large data sets. Moreover, data mining tasks can be used to extend the capability of existing database systems to satisfy advanced users' sophisticated data analysis requirements.

Recent database systems have built systematic data analysis capabilities on database data using data warehousing and data mining facilities. A **data warehouse** integrates data originating from multiple sources and various timeframes. It consolidates data in multidimensional space to form partially materialized data cubes. The data cube model not

only facilitates OLAP in multidimensional databases but also promotes multidimensional data mining

Information Retrieval

Information retrieval (IR) is the science of searching for documents or information in documents. Documents can be text or multimedia, and may reside on the Web. The differences between traditional information retrieval and database systems are twofold: Information retrieval assumes that (1) the data under search are unstructured; and (2) the queries are formed mainly by keywords, which do not have complex structures (unlike SQL queries in database systems).

The typical approaches in information retrieval adopt probabilistic models. For example, a text document can be regarded as a bag of words, that is, a multiset of words appearing in the document. The document's **language model** is the probability density function that generates the bag of words in the document. The similarity between two documents can be measured by the similarity between their corresponding language models.

Furthermore, a topic in a set of text documents can be modeled as a probability distribution over the vocabulary, which is called a **topic model**. A text document, which may involve one or multiple topics, can be regarded as a mixture of multiple topic models. By integrating information retrieval models and data mining techniques, we can find the major topics in a collection of documents and, for each document in the collection, the major topics involved.

Increasingly large amounts of text and multimedia data have been accumulated and made available online due to the fast growth of the Web and applications such as digital libraries, digital governments, and health care information systems. Their effective search and analysis have raised many challenging issues in data mining. Therefore, text mining and multimedia data mining, integrated with information retrieval methods, have become increasingly important.

➤ **What Kinds of Patterns Can Be Mined?**

We have observed various types of data and information repositories on which data mining can be performed. Let us now examine the kinds of patterns that can be mined.

There are a number of data mining functionalities. These include characterization and discrimination; the mining of frequent patterns, associations, and correlations; classification and regression; clustering analysis; and outlier analysis. Data mining functionalities are used to specify the kinds of patterns to be found in data mining tasks. In general, such tasks can be classified into two categories: descriptive and predictive. Descriptive mining tasks characterize properties of the data in a target data set. Predictive mining tasks perform induction on the current data in order to make predictions.

Data mining functionalities, and the kinds of patterns they can discover, are described below. In addition, Section 1.4.6 looks at what makes a pattern interesting. Interesting patterns represent knowledge.

Class/Concept Description: Characterization and Discrimination

Data entries can be associated with classes or concepts. For example, in the AllElectronics store, classes of items for sale include computers and printers, and concepts of customers include big Spenders and budget Spenders. It can be useful to describe individual classes and concepts in summarized, concise, and yet precise terms. Such descriptions of a class or a concept are called class/concept descriptions. These descriptions can be derived using (1) data characterization, by summarizing the data of the class under study (often called the target class) in general terms, or (2) data discrimination, by comparison of the target class with one or a set of comparative classes (often called the contrasting classes), or (3) both data characterization and discrimination.

Data characterization is a summarization of the general characteristics or features of a target class of data. The data corresponding to the user-specified class are typically collected by a query. For example, to study the characteristics of software products with sales that increased by 10% in the previous year, the data related to such products can be collected by executing an SQL query on the sales database.

The output of data characterization can be presented in various forms. Examples include pie charts, bar charts, curves, multidimensional data cubes, and multidimensional tables, including crosstabs. The resulting descriptions can also be presented as generalized relations or in rule form (called characteristic rules).

Example

Data characterization

A customer relationship manager at All Electronics may order the following data mining task: Summarize the characteristics of customers who spend more than \$5000 a year at AllElectronics. The result is a general profile of these customers, such as that they are 40 to 50 years old, employed, and have excellent credit ratings. The data mining system should allow the customer relationship manager to drill down on any dimension, such as on occupation to view these customers according to their type of employment.

A customer relationship manager at AllElectronics may order the following data mining task: Summarize the characteristics of customers who spend more than \$5000 a year at AllElectronics. The result is a general profile of these customers, such as that they are 40 to 50 years old, employed, and have excellent credit ratings. The data mining system should allow the customer relationship manager to drill down on any dimension, such as on occupation to view these customers according to their type of employment.

Data discrimination is a comparison of the general features of the target class data objects against the general features of objects from one or multiple contrasting classes. The target and contrasting classes can be specified by a user, and the corresponding data objects can be retrieved through database queries. For example, a user may want to compare the general features of software products with sales that increased by 10% last year against those

with sales that decreased by at least 30% during the same period. The methods used for data discrimination are similar to those used for data characterization.

“How are discrimination descriptions output?” The forms of output presentation are similar to those for characteristic descriptions, although discrimination descriptions should include comparative measures that help to distinguish between the target and contrasting classes. Discrimination descriptions expressed in the form of rules are referred to as discriminant rules.

Example

Data discrimination

A customer relationship manager at *AllElectronics* may want to compare two groups of customers—those who shop for computer products regularly (e.g., more than twice a month) and those who rarely shop for such products (e.g., less than three times a year). The resulting description provides a general comparative profile of these customers, such as that 80% of the customers who frequently purchase computer products are between 20 and 40 years old and have a university education, whereas 60% of the customers who infrequently buy such products are either seniors or youths, and have no university degree. Drilling down on a dimension like *occupation*, or adding a new dimension like *income_level*, may help to find even more discriminative features between the two classes.

A customer relationship manager at AllElectronics may want to compare two groups of customers—those who shop for computer products regularly (e.g., more than twice a month) and those who rarely shop for such products (e.g., less than three times a year). The resulting description provides a general comparative profile of these customers, such as that 80% of the customers who frequently purchase computer products are between 20 and 40 years old and have a university education, whereas 60% of the customers who infrequently buy such products are either seniors or youths, and have no university degree. Drilling down on a dimension like *occupation*, or adding a new dimension like *income_level*, may help to find even more discriminative features between the two classes.

Concept description, including characterization and discrimination, is described in Chapter 4.

Mining Frequent Patterns, Associations, and Correlations

Frequent patterns, as the name suggests, are patterns that occur frequently in data. There are many kinds of frequent patterns, including frequent itemsets, frequent subsequences (also known as sequential patterns), and frequent substructures. A frequent itemset typically refers to a set of items that often appear together in a transactional data set—for example, milk and bread, which are frequently bought together in grocery stores by many customers. A frequently occurring subsequence, such as the pattern that customers tend to purchase first a laptop, followed by a digital camera, and then a memory card, is a (frequent) sequential pattern. A substructure can refer to different structural forms (e.g., graphs, trees, or lattices) that may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (frequent) structured pattern. Mining frequent patterns leads to the discovery of interesting associations and correlations within data.

Example

Association analysis

Suppose that, as a marketing manager at *AllElectronics*, you want to know which items are frequently purchased together (i.e., within the same transaction). An example of such a rule, mined from the *AllElectronics* transactional database, is

where X is a variable representing a customer. A **confidence**, or certainty, of 50% means that if a customer buys a computer, there is a 50% chance that she will buy software as well. A 1% **support** means that 1% of all the transactions under analysis show that computer and software are purchased together. This association rule involves a single attribute or predicate (i.e., *buys*) that repeats. Association rules that contain a single predicate are referred to as **single-dimensional association rules**. Dropping the predicate notation, the rule can be written simply as “computer $\Rightarrow$ software [1%, 50%].”

Suppose, instead, that we are given the *AllElectronics* relational database related to purchases. A data mining system may find association rules like

The rule indicates that of the *AllElectronics* customers under study, 2% are 20 to 29 years old with an income of \$40,000 to \$49,000 and have purchased a laptop (computer) at *AllElectronics*. There is a 60% probability that a customer in this age and income group will purchase a laptop. Note that this is an association involving more than one attribute or predicate (i.e., *age*, *income*, and *buys*). Adopting the terminology used in multidimensional databases, where each attribute is referred to as a dimension, the above rule can be referred to as a **multidimensional association rule**.

Suppose that, as a marketing manager at *AllElectronics*, you want to know which items are frequently purchased together (i.e., within the same transaction). An example of such a rule, mined from the *AllElectronics* transactional database, is where X is a variable representing a customer. A confidence, or certainty, of 50% means that if a customer buys a computer, there is a 50% chance that she will buy software as well. A 1% support means that 1% of all the transactions under analysis show that computer and software are purchased together. This association rule involves a single attribute or predicate (i.e., *buys*) that repeats. Association rules that contain a single predicate are referred to as single-dimensional association rules. Dropping the predicate notation, the rule can be written simply as “computer $\Rightarrow$ software [1%, 50%].”

Suppose, instead, that we are given the *AllElectronics* relational database related to purchases. A data mining system may find association rules like

The rule indicates that of the *AllElectronics* customers under study, 2% are 20 to 29 years old with an income of \$40,000 to \$49,000 and have purchased a laptop (computer) at *AllElectronics*. There is a 60% probability that a customer in this age and income group will purchase a laptop. Note that this is an association involving more than one attribute or predicate (i.e., *age*, *income*, and *buys*). Adopting the terminology used in multidimensional databases, where each attribute is referred to as a dimension, the above rule can be referred to as a multidimensional association rule.

Typically, association rules are discarded as uninteresting if they do not satisfy both a minimum support threshold and a minimum confidence threshold. Additional analysis can be

performed to uncover interesting statistical correlations between associated attribute–value pairs.

Frequent itemset mining is a fundamental form of frequent pattern mining. The mining of frequent patterns, associations, and correlations is discussed in Chapters 6 and 7, where particular emphasis is placed on efficient algorithms for frequent itemset mining. Sequential pattern mining and structured pattern mining are considered advanced topics.

Classification and Regression for Predictive Analysis

Classification is the process of finding a model (or function) that describes and distinguishes data classes or concepts. The model are derived based on the analysis of a set of training data (i.e., data objects for which the class labels are known). The model is used to predict the class label of objects for which the class label is unknown.

“How is the derived model presented?” The derived model may be represented in various forms, such as classification rules (i.e., IF-THEN rules), decision trees, mathematical formulae, or neural networks (Figure 1.9). A decision tree is a flowchart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can easily be converted to classification rules. A neural network, when used for classification, is typically a collection of neuron-like processing units with weighted connections between the units. There are many other methods for constructing classification models, such as naïve Bayesian classification, support vector machines, and k-nearest-neighbor classification.

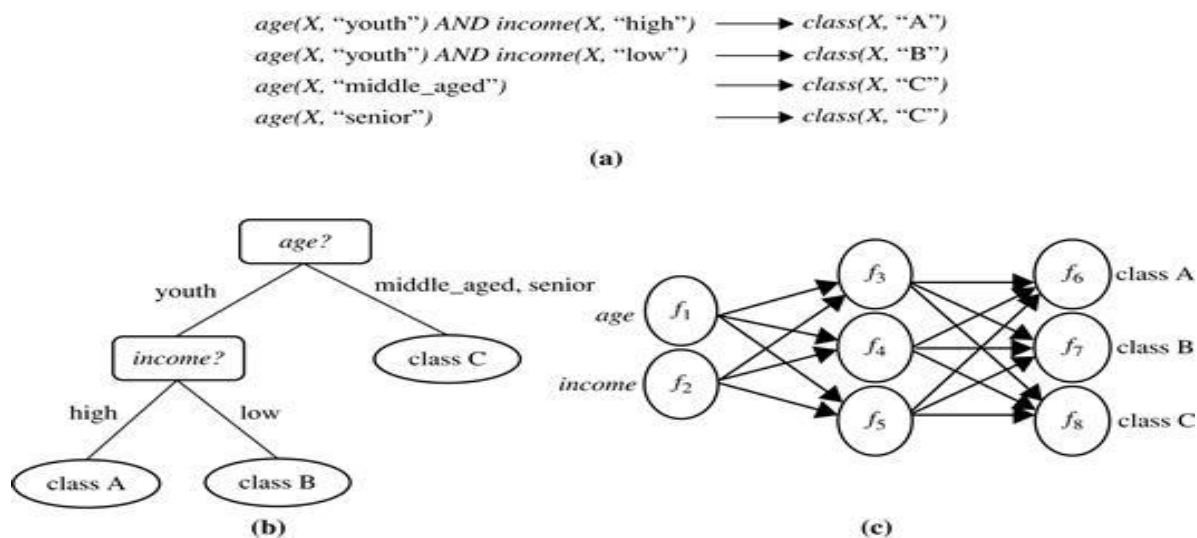


FIGURE 1.9 A classification model can be represented in various forms: (a) IF-THEN rules, (b) a decision tree, or (c) a neural network.

Whereas classification predicts categorical (discrete, unordered) labels, regression models continuous-valued functions. That is, regression is used to predict missing or

unavailable numerical data values rather than (discrete) class labels. The term prediction refers to both numeric prediction and class label prediction. Regression analysis is a statistical methodology that is most often used for numeric prediction, although other methods exist as well. Regression also encompasses the identification of distribution trends based on the available data.

Classification and regression may need to be preceded by relevance analysis, which attempts to identify attributes that are significantly relevant to the classification and regression process. Such attributes will be selected for the classification and regression process. Other attributes, which are irrelevant, can then be excluded from consideration.

Example 1.8

Classification and regression

Suppose as a sales manager of AllElectronics you want to classify a large set of items in the store, based on three kinds of responses to a sales campaign: good response, mild response and no response. You want to derive a model for each of these three classes based on the descriptive features of the items, such as *price*, *brand*, *place_made*, *type*, and *category*. The resulting classification should maximally distinguish each class from the others, presenting an organized picture of the data set.

Suppose that the resulting classification is expressed as a decision tree. The decision tree, for instance, may identify *price* as being the single factor that best distinguishes the three classes. The tree may reveal that, in addition to *price*, other features that help to further distinguish objects of each class from one another include *brand* and *place_made*. Such a decision tree may help you understand the impact of the given sales campaign and design a more effective campaign in the future.

Suppose instead, that rather than predicting categorical response labels for each store item, you would like to predict the amount of revenue that each item will generate during an upcoming sale at AllElectronics, based on the previous sales data. This is an example of regression analysis because the regression model constructed will predict a continuous function (or ordered value.)

Suppose as a sales manager of AllElectronics you want to classify a large set of items in the store, based on three kinds of responses to a sales campaign: good response, mild response and no response. You want to derive a model for each of these three classes based on the descriptive features of the items, such as *price*, *brand*, *place_made*, *type*, and *category*. The resulting classification should maximally distinguish each class from the others, presenting an organized picture of the data set.

Suppose that the resulting classification is expressed as a decision tree. The decision tree, for instance, may identify *price* as being the single factor that best distinguishes the three classes. The tree may reveal that, in addition to *price*, other features that help to further distinguish objects of each class from one another include *brand* and *place_made*. Such a decision tree may help you understand the impact of the given sales campaign and design a more effective campaign in the future.

Suppose instead, that rather than predicting categorical response labels for each store item, you would like to predict the amount of revenue that each item will generate during an upcoming sale at AllElectronics, based on the previous sales data. This is an example of

regression analysis because the regression model constructed will predict a continuous function (or ordered value.)

Cluster Analysis

Unlike classification and regression, which analyze class-labeled (training) data sets, clustering analyzes data objects without consulting class labels. In many cases, class-labeled data may simply not exist at the beginning. Clustering can be used to generate class labels for a group of data. The objects are clustered or grouped based on the principle of maximizing the intraclass similarity and minimizing the interclass similarity. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are rather dissimilar to objects in other clusters. Each cluster so formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate taxonomy formation, that is, the organization of observations into a hierarchy of classes that group similar events together.

Example 1.9

Cluster analysis

Cluster analysis can be performed on AllElectronics customer data to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing. [Figure 1.10](#) shows a 2-D plot of customers with respect to customer locations in a city. Three clusters of data points are evident.

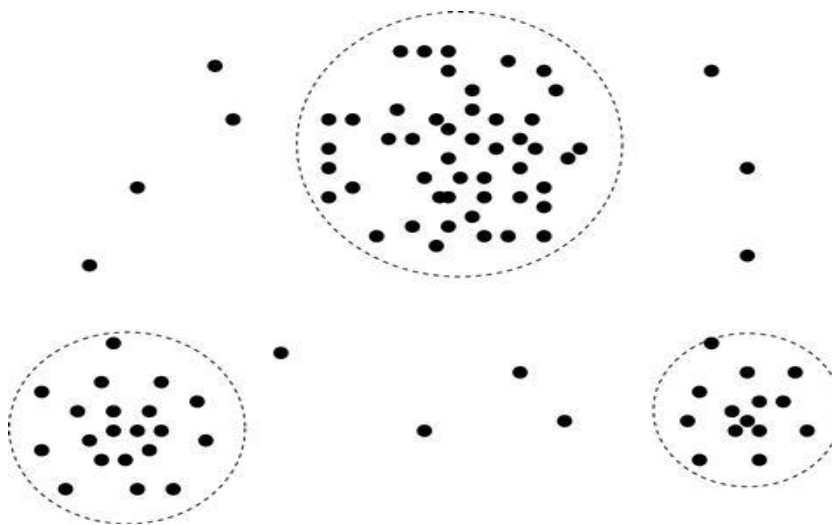


FIGURE 1.10 A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters.

Cluster analysis can be performed on AllElectronics customer data to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing. Figure 1.10 shows a 2-D plot of customers with respect to customer locations in a city. Three clusters of data points are evident.

Outlier Analysis

A data set may contain objects that do not comply with the general behavior or model of the data. These data objects are outliers. Many data mining methods discard outliers as noise or

exceptions. However, in some applications (e.g., fraud detection) the rare events can be more interesting than the more regularly occurring ones. The analysis of outlier data is referred to as outlier analysis or anomaly mining.

Outliers may be detected using statistical tests that assume a distribution or probability model for the data, or using distance measures where objects that are remote from any other cluster are considered outliers. Rather than using statistical or distance measures, density-based methods may identify outliers in a local region, although they look normal from a global statistical distribution view.

Example 1.10

Outlier analysis

Outlier analysis may uncover fraudulent usage of credit cards by detecting purchases of unusually large amounts for a given account number in comparison to regular charges incurred by the same account. Outlier values may also be detected with respect to the locations and types of purchase, or the purchase frequency

Outlier analysis may uncover fraudulent usage of credit cards by detecting purchases of unusually large amounts for a given account number in comparison to regular charges incurred by the same account. Outlier values may also be detected with respect to the locations and types of purchase, or the purchase frequency.

Are All Patterns Interesting?

A data mining system has the potential to generate thousands or even millions of patterns, or rules.

You may ask, “Are all of the patterns interesting?” Typically, the answer is no—only a small fraction of the patterns potentially generated would actually be of interest to a given user.

This raises some serious questions for data mining. You may wonder, “What makes a pattern interesting? Can a data mining system generate all of the interesting patterns? Or, Can the system generate only the interesting ones?”

To answer the first question, a pattern is interesting if it is (1) easily understood by humans, (2) valid on new or test data with some degree of certainty, (3) potentially useful, and (4) novel. A pattern is also interesting if it validates a hypothesis that the user sought to confirm. An interesting pattern represents knowledge.

Several objective measures of pattern interestingness exist. These are based on the structure of discovered patterns and the statistics underlying them. An objective measure for association rules of the form is rule support, representing the percentage of transactions from a transaction database that the given rule satisfies. This is taken to be the probability , where indicates that a transaction contains both X and Y, that is, the union of itemsets X and Y. Another objective measure for association rules is confidence, which assesses the degree of certainty of the detected association. This is taken to be the conditional probability), that is, the probability that a transaction containing X also contains Y. More formally, support and confidence are defined as

$$\text{support}(X \Rightarrow Y) = P(X \cup Y),$$

$$\text{confidence}(X \Rightarrow Y) = P(Y|X).$$

In general, each interestingness measure is associated with a threshold, which may be controlled by the user. For example, rules that do not satisfy a confidence threshold of, say, 50% can be considered uninteresting. Rules below the threshold likely reflect noise, exceptions, or minority cases and are probably of less value.

Other objective interestingness measures include accuracy and coverage for classification (IF-THEN) rules. In general terms, accuracy tells us the percentage of data that are correctly classified by a rule. Coverage is similar to support, in that it tells us the percentage of data to which a rule applies. Regarding understandability, we may use simple objective measures that assess the complexity or length in bits of the patterns mined.

Although objective measures help identify interesting patterns, they are often insufficient unless combined with subjective measures that reflect a particular user's needs and interests. For example, patterns describing the characteristics of customers who shop frequently at AllElectronics should be interesting to the marketing manager, but may be of little interest to other analysts studying the same database for patterns on employee performance. Furthermore, many patterns that are interesting by objective standards may represent common sense and, therefore, are actually uninteresting.

Subjective interestingness measures are based on user beliefs in the data. These measures find patterns interesting if the patterns are unexpected (contradicting a user's belief) or offer strategic information on which the user can act. In the latter case, such patterns are referred to as actionable. For example, patterns like "a large earthquake often follows a cluster of small quakes" may be highly actionable if users can act on the information to save lives. Patterns that are expected can be interesting if they confirm a hypothesis that the user wishes to validate or they resemble a user's hunch.

The second question—"Can a data mining system generate all of the interesting patterns?"—refers to the completeness of a data mining algorithm. It is often unrealistic and inefficient for data mining systems to generate all possible patterns. Instead, user-provided constraints and interestingness measures should be used to focus the search. For some mining tasks, such as association, this is often sufficient to ensure the completeness of the algorithm. Association rule mining is an example where the use of constraints and interestingness measures can ensure the completeness of mining.

Finally, the third question—"Can a data mining system generate only interesting patterns?"—is an optimization problem in data mining. It is highly desirable for data mining systems to generate only interesting patterns. This would be efficient for users and data mining systems because neither would have to search through the patterns generated to identify the truly interesting ones. Progress has been made in this direction; however, such optimization remains a challenging issue in data mining.

Measures of pattern interestingness are essential for the efficient discovery of patterns by target users. Such measures can be used after the data mining step to rank the discovered patterns according to their interestingness, filtering out the uninteresting ones. More important, such measures can be used to guide and constrain the discovery process, improving the search efficiency by pruning away subsets of the pattern space that do not satisfy prespecified interestingness constraints. Methods to assess pattern interestingness, and their use to improve data mining efficiency, are discussed throughout the book with respect to each kind of pattern that can be mined.

➤ Data Preprocessing

Why do We Need to Preprocess the Data?

Much of the raw data contained in databases is unpreprocessed, incomplete, and noisy. For example, the databases may contain

- fields that are obsolete or redundant;
- missing values;
- outliers;
- data in a form not suitable for the data mining models;
- values not consistent with policy or common sense.

In order to be useful for data mining purposes, the databases need to undergo preprocessing, in the form of *data cleaning* and *data transformation*. Data mining often deals with data that has not been looked at for years, so that much of the data contains field values that have expired, are no longer relevant, or are simply missing. The overriding objective is to minimize garbage in, garbage out (GIGO), to minimize the Garbage that gets Into our model, so that we can minimize the amount of Garbage that our models give Out.

Depending on the data set, data preprocessing alone can account for 10–60% of all the time and effort for the entire data mining process. In this chapter, we shall examine several ways to preprocess the data for further analysis downstream.

➤ Data Preprocessing

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Why is Data preprocessing important?

Preprocessing of data is mainly to check the data quality. The quality can be checked by the following

- **Accuracy:** To check whether the data entered is correct or not.
- **Completeness:** To check whether the data is available or not recorded.
- **Consistency:** To check whether the same data is kept in all the places that do or do not match.
- **Timeliness:** The data should be updated correctly.
- **Believability:** The data should be trustable.

- **Interpretability:** The understandability of the data.

Major Tasks in Data Preprocessing:

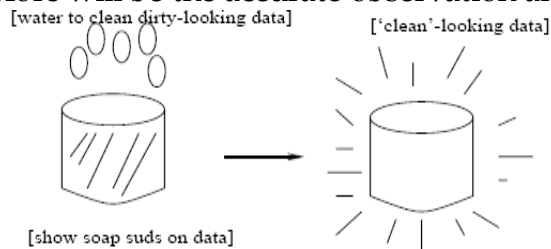
1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

Need of Data Preprocessing

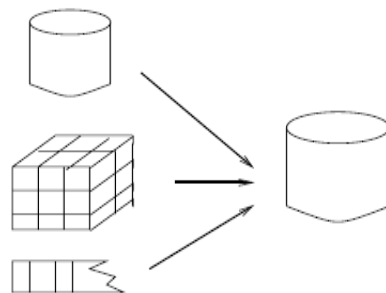
Data preprocessing refers to the set of techniques implemented on the databases to remove noisy, missing, and inconsistent data. Different Data preprocessing techniques involved in data mining are data cleaning, data integration, data reduction, and data transformation.

The need for data preprocessing arises from the fact that the real-time data and many times the data of the database is often incomplete and inconsistent which may result in improper and inaccurate data mining results. Thus to improve the quality of data on which the observation and analysis are to be done, it is treated with these four steps of data preprocessing. More the improved data, More will be the accurate observation and prediction.

Data Cleaning



Data Integration



Data Transformation

-2, 32, 100, 59, 48 → -0.02, 0.32, 1.00, 0.59, 0.48

Data Reduction



➤ Data cleaning

Data in the real world is usually incomplete, incomplete and noisy. The data cleaning process includes the procedure which aims at filling the missing values, smoothing out the noise which determines the outliers and rectifies the inconsistencies in data. Let us discuss the basic methods of data cleaning,

Missing Values

Assume that you are dealing with any data like sales and customer data and you observe that there are several attributes from which the data is missing. One cannot compute data with missing values. In this case, there are some methods which sort out this problem. Let us go through them one by one

1. Ignore the data row

This is usually done when the *class label* is missing (assuming your data mining goal is classification), or many attributes are missing from the row (not just one). However, you'll obviously get poor performance if the percentage of such rows is high.

For example, let's say we have a database of students enrolment data (age, SAT score, state of residence, etc.) and a column classifying their success in college to "Low", "Medium" and "High". Let's say our goal is to build a model predicting a student's success in college. Data rows who are missing the success column are not useful in predicting success so they could very well be ignored and removed before running the algorithm.

2. Use a global constant to fill in for missing values

Decide on a new global constant value, like "*unknown*", "*N/A*" or minus infinity, that will be used to fill all the missing values. This technique is used because sometimes it just doesn't make sense to try and predict the missing value.

For example, let's look at the students enrollment database again. Assuming the *state of residence* attribute data is missing for some students. Filling it up with some state doesn't really makes sense as opposed to using something like "*N/A*".

3. Use attribute mean

Replace missing values of an attribute with the mean (or median if its discrete) value for that attribute in the database.

For example, in a database of US family incomes, if the average *income* of a US family is X you can use that value to replace missing income values.

4. Use attribute mean for all samples belonging to the same class

Instead of using the mean (or median) of a certain attribute calculated by looking at all the rows in a database, we can limit the calculations to the relevant class to make the value more relevant to the row we're looking at.

Let's say you have a cars pricing database that, among other things, classifies cars to "Luxury" and "Low budget" and you're dealing with missing values in the cost field. Replacing missing cost of a luxury car with the average cost of all luxury cars is probably more accurate than the value you'd get if you factor in the low budget cars.

5. Use a data mining algorithm to predict the most probable value

The value can be determined using regression, inference based tools using Bayesian formalism, decision trees, clustering algorithms (K-Mean\Median etc.).

For example, we could use clustering algorithms to create clusters of rows which will then be used for calculating an attribute mean or median as specified in technique #3. Another example could be using a decision tree to try and predict the probable value in the missing attribute, according to other attributes in the data.

I'd suggest looking into regression and decision trees first (ID3 tree generation) as they're relatively easy and there are plenty of examples on the net...

Noisy Data

Noise is a random error or variance in a measured variable. Noisy Data may be due to faulty data collection instruments, data entry problems and technology limitation.

How to Handle Noisy Data?

Binning:

Binning methods sorted data value by consulting its "neighbor- hood," that is, the values around it. The sorted values are distributed into a number of "buckets," or bins.

For example

Price = 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equal-frequency) bins:

Bin a: 4, 8, 15

Bin b: 21, 21, 24

Bin c: 25, 28, 34

In this example, the data for price are first sorted and then partitioned into equal-frequency bins of size 3.

Smoothing by bin means:

Bin a: 9, 9, 9

Bin b: 22, 22, 22

Bin c: 29, 29, 29

In smoothing by bin means, each value in a bin is replaced by the mean value of the bin.

Smoothing by bin boundaries:

Bin a: 4, 4, 15

Bin b: 21, 21, 24

Bin c: 25, 25, 34

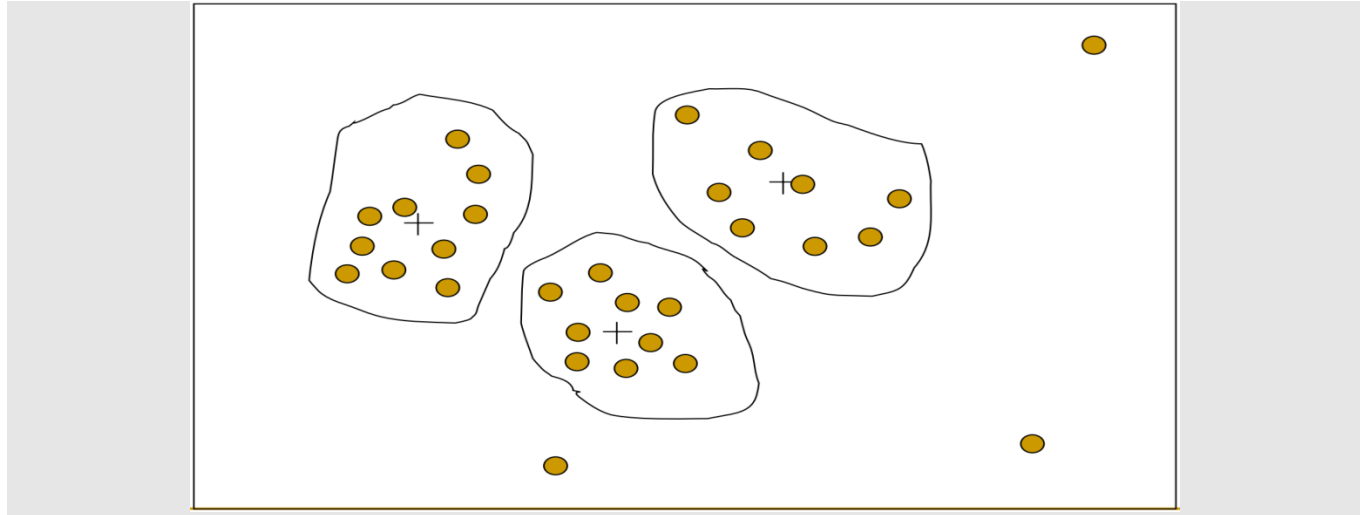
In smoothing by bin boundaries, each bin value is replaced by the closest boundary value.

Regression

Data can be smoothed by fitting the data into a regression functions.

Clustering:

Outliers may be detected by clustering, where similar values are organized into groups, or “clusters. Values that fall outside of the set of clusters may be considered outliers.



Data Cleaning as a Process

The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and the inconsistent use of codes. Errors in instrumentation devices that record data, and system errors, are another source of discrepancies. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).

As a data analyst, you should be on the lookout for the inconsistent use of codes and any inconsistent data representations (such as “2010/12/25” and “25/12/2010” for *date*). Field overloading is another source of errors that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., using an unused bit of an attribute whose value range uses only, say, 31 out of 32 bits).

The data should also be examined regarding unique rules, consecutive rules, and null rules. A unique rule says that each value of the given attribute must be different from all other values for that attribute. A consecutive rule says that there can be no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers). A null rule specifies the use of blanks, question marks, special characters, or other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled. As mentioned in Section 3.2.1, reasons for missing values may

include (1) the person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a *license-number* attribute left blank by non drivers); (2) the data entry person does not know the correct value; or (3) the value is to be provided by a later step of the process. The null rule should specify how to record the null condition, for example, such as to store zero for numeric attributes, a blank for character attributes, or any other conventions that may be in use (such as that entries like “don’t know” or “?” should be transformed to blank).

There are a number of different commercial tools that can aid in the step of discrepancy detection. Data scrubbing tools use simple domain knowledge (e.g., knowledge of postal addresses, and spell-checking) to detect errors and make corrections in the data. These tools rely on parsing and fuzzy matching techniques when cleaning data from multiple sources. Data auditing tools find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions. They are variants of data mining tools. For example, they may employ statistical analysis to find correlations, or clustering to identify outliers. Commercial tools can assist in the data transformation step. Data migration tools allow simple transformations to be specified, such as to replace the string “gender” by “sex”. ETL (extraction/transformation/loading) tools allow users to specify transforms through a graphical user interface (GUI). These tools typically support only a restricted set of transforms so that, often, we may also choose to write custom scripts for this step of the data cleaning process.

The two-step process of discrepancy detection and data transformation (to correct discrepancies) iterates. This process, however, is error-prone and time-consuming. Some transformations may introduce more discrepancies. Some *nested discrepancies* may only be detected after others have been fixed. For example, a typo such as “20010” in a year field may only surface once all date values have been converted to a uniform format. Transformations are often done as a batch process while the user waits without feedback. Only after the transformation is complete can the user go back and check that no new anomalies have been created by mistake. Typically, numerous iterations are required before the user is satisfied. Any tuples that cannot be automatically handled by a given transformation are typically written to a file without any explanation regarding the reasoning behind their failure. As a result, the entire data cleaning process also suffers from a lack of interactivity.

Users gradually build a series of transformations by composing and debugging individual transformations, one step at a time, on a spreadsheet-like interface. The transformations can be specified graphically or by providing examples. Results are shown immediately on the records that are visible on the screen. The user can choose to undo the transformations, so that transformations that introduced

additional errors can be “erased.” The tool performs discrepancy checking automatically in the background on the latest transformed view of the data. Users can gradually develop and refine transformations as discrepancies are found, leading to more effective and efficient data cleaning.

Another approach to increased interactivity in data cleaning is the development of declarative languages for the specification of data transformation operators. Such work focuses on defining powerful extensions to SQL and algorithms that enable users to express data cleaning specifications efficiently. As we discover more about the data, it is important to keep updating the metadata to reflect this knowledge. This will help speed up data cleaning on future versions of the same data store.

➤ Data Integration

Data mining often requires data integration the merging of data from multiple data stores. Integration can help reduce and avoid redundancies and inconsistencies in the resulting data set. It help improve the accuracy and speed of the subsequent mining process.

It has been an integral part of data operations because data can be obtained from several sources. It is a strategy that integrates data from several sources to make it available to users in a single uniform view that shows their status. There are communication sources between systems that can include multiple databases, data cubes, or flat files. Data fusion merges data from various diverse sources to produce meaningful results. The consolidated findings must exclude inconsistencies, contradictions, redundancies, and inequities.

Data integration is important because it gives a uniform view of scattered data while also maintaining data accuracy. It assists the data-mining program in meaningful mining information, which in turn assists the executive and managers make strategic decisions for the enterprise's benefit.

The data integration methods are formally characterized as a triple (G, S, M) , where;

G represents the **global schema**,

S represents the **heterogeneous** source of schema,

M represents the **mapping** between **source** and **global** schema queries.

Why is the Data Integration Important?

Companies that want to stay competitive and relevant welcome big data and all of its benefits and drawbacks. One of the most common applications for data integration services and technologies is market and consumer data collection. Data integration supports queries in these vast datasets, benefiting from corporate intelligence and consumer data analytics to stimulate real-time information delivery. Enterprise data integration feeds integrated data into data centers to enable enterprise reporting, predictive analytics, and business intelligence.

Data integration is particularly important in the healthcare industry. Integrated data from various patient records and clinics assist clinicians in identifying medical disorders and diseases by integrating data from many systems into a single perspective of beneficial information from which useful insights can be derived. Effective data collection and integration also improve medical insurance claims processing accuracy and ensure that patient names and contact information are recorded consistently and accurately. Interoperability refers to the sharing of information across different systems.

Data Integration Approaches

There are mainly two types of approaches for data integration. These are as follows:

Tight Coupling

It is the process of using ETL (**Extraction, Transformation, and Loading**) to combine data from various sources into a single physical location.

Loose Coupling

Facts with loose coupling are most effectively kept in the actual source databases. This approach provides an interface that gets a query from the user, changes it into a format that the supply database may understand, and then sends the query to the source databases without delay to obtain the result.

Issues in Data Integration

When you integrate the data in Data Mining, you may face many issues. There are some of those issues:

Entity Identification Problem

As you understand, the records are obtained from heterogeneous sources, and how can you 'match the real-world entities from the data'. For example, you were given client data from specialized statistics sites. Customer identity is assigned to an entity from one statistics supply, while a customer range is assigned to an entity from another statistics supply. Analyzing such metadata statistics will prevent you from making errors during schema integration.

Structural integration is completed by guaranteeing that the functional dependency and referential constraints of a character in the source machine match the functional dependency and referential constraints of the identical character in the target machine. For example, assume that the discount is applied to the entire order in one machine, but in every other machine, the discount is applied to each item in the order. This distinction should be noted before the information from those assets is included in the goal system.

Redundancy and Correlation Analysis

One of the major issues in the course of data integration is redundancy. Unimportant data that are no longer required are referred to as redundant data. It may also appear due to attributes created from the use of another property inside the information set. For example, if one truth set contains the patronage and distinct data set as the purchaser's date of the beginning, then

age may be a redundant attribute because it can be deduced from the use of the beginning date.

Inconsistencies further increase the level of redundancy within the characteristic. The use of correlation analysis can be used to determine redundancy. The traits are examined to determine their interdependence on each difference, consequently discovering the link between them.

Tuple Duplication

Information integration has also handled duplicate tuples in addition to redundancy. Duplicate tuples may also appear in the generated information if the denormalized table was utilized as a deliverable for data integration.

Integration tools

There are various integration tools in data mining. Some of them are as follows:

On-premise data integration tool

An on-premise data integration tool integrates data from local sources and connects legacy databases using middleware software.

Open-source data integration tool

If you want to avoid pricey enterprise solutions, an open-source data integration tool is the ideal alternative. Although, you will be responsible for the security and privacy of the data if you're using the tool.

Cloud-based data integration tool

A cloud-based data integration tool may provide an '**integration platform as a service**'.

➤ Data Transformation

In data transformation, the data are transformed or consolidated into forms appropriate for mining. Data transformation increases the efficiency of business and analytic processes, and it enables businesses to make better data-driven decisions. During the data transformation process, an analyst will determine the structure of the data. Data transformation can involve the following:

1. Data Smoothing

Data smoothing is a process that is used to remove noise from the dataset using some algorithms. It allows for highlighting important features present in the dataset. It helps in predicting the patterns. When collecting data, it can be manipulated to eliminate or reduce any variance or any other noise form.

The concept behind data smoothing is that it will be able to identify simple changes to help predict different trends and patterns. This serves as a help to analysts or traders who need to look at a lot of data which can often be difficult to digest for finding patterns that they wouldn't see otherwise.

We have seen how the noise is removed from the data using the techniques such as binning, regression, clustering.

- **Binning:** This method splits the sorted data into the number of bins and smoothens the data values in each bin considering the neighborhood values around it.
- **Regression:** This method identifies the relation among two dependent attributes so that if we have one attribute, it can be used to predict the other attribute.
- **Clustering:** This method groups similar data values and form a cluster. The values that lie outside a cluster are known as outliers.

2. Attribute Construction

In the attribute construction method, the new attributes consult the existing attributes to construct a new data set that eases data mining. New attributes are created and applied to assist the mining process from the given attributes. This simplifies the original data and makes the mining more efficient.

For example, suppose we have a data set referring to measurements of different plots, i.e., we may have the height and width of each plot. So here, we can construct a new attribute 'area' from attributes 'height' and 'weight'. This also helps understand the relations among the attributes in a data set.

3.Data Aggregation

Data collection or aggregation is the method of storing and presenting data in a summary format. The data may be obtained from multiple data sources to integrate these data sources into a data analysis description. This is a crucial step since the accuracy of data analysis insights is highly dependent on the quantity and quality of the data used.

4. Data Normalization

Normalizing the data refers to scaling the data values to a much smaller range such as [-1, 1] or [0.0, 1.0]. There are different methods to normalize the data, as discussed below.

Consider that we have a numeric attribute A and we have n number of observed values for attribute A that are $V_1, V_2, V_3, \dots, V_n$.

- **Min-max normalization:** This method implements a linear transformation on the original data. Let us consider that we have $\min_A$ and $\max_A$ as the minimum and maximum value observed for attribute A and V_i is the value for attribute A that has to be normalized
- The min-max normalization would map V_i to the V'_i in a new smaller range [$\text{new_min}_A, \text{new_max}_A$]. The formula for min-max normalization is given below:

$$v'_i = \frac{v_i - \min_A}{\max_A - \min_A} (\text{new_max}_A - \text{new_min}_A) + \text{new_min}_A$$

For example, we have \$1200 and \$9800 as the minimum, and maximum value for the attribute income, and [0.0, 1.0] is the range in which we have to map a value of \$73,600.

The value \$73,600 would be transformed using min-max normalization as follows:

$$\frac{73600-1200}{9800-1200}(1.0 - 0.0) + 0.0 = 0.716$$

- **Z-score normalization:** This method normalizes the value for attribute A using the *mean* and *standard deviation*. The following formula is used for Z-score normalization:

$$v'_i = \frac{v_i - \bar{A}}{\sigma_A}$$

Here $\bar{A}$ and σ_A are the mean and standard deviation for attribute A, respectively. For example, we have a mean and standard deviation for attribute A as \$54,000 and \$16,000. And we have to normalize the value \$73,600 using z-score normalization.

$$\frac{73600-54000}{16000} = 1.225$$

- **Decimal Scaling:** This method normalizes the value of attribute A by moving the decimal point in the value. This movement of a decimal point depends on the maximum absolute value of A. The formula for the decimal scaling is given below:

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

Here j is the smallest integer such that $\max(|v'_i|) < 1$. For example, the observed values for attribute A range from -986 to 917, and the maximum absolute value for attribute A is 986. Here, to normalize each value of attribute A using decimal scaling, we have to divide each value of attribute A by 1000, i.e., j=3

So, the value -986 would be normalized to -0.986, and 917 would be normalized to 0.917. The normalization parameters such as mean, standard deviation, the maximum absolute value must be preserved to normalize the future data uniformly.

5. Data Generalization

It converts low-level data attributes to high-level data attributes using concept hierarchy. This conversion from a lower level to a higher conceptual level is useful to get a clearer picture of the data. Data generalization can be divided into two approaches:

- Data cube process (OLAP) approach.
- Attribute-oriented induction (AOI) approach.

For example, age data can be in the form of (20, 30) in a dataset. It is transformed into a higher conceptual level into a categorical value (young, old).

➤ Data Reduction

Data reduction is a process that reduces the volume of original data and represents it in a much smaller volume. Data reduction techniques are used to obtain a reduced representation of the dataset that is much smaller in volume by maintaining the integrity of the original data.

Data reduction aims to define it more compactly. When the data size is smaller, it is simpler to apply sophisticated and computationally high-priced algorithms. The reduction of the data may be in terms of the number of rows (records) or terms of the number of columns (dimensions).

Strategies for data reduction include the following:

1. **Data cube aggregation:** where aggregation operations are applied to the data in the construction of a data cube.
2. **Attribute subset selection,** where irrelevant, weakly relevant, or redundant attributes or dimensions may be deleted and removed.
3. **Dimensionality reduction,** where encoding mechanisms are used to reduce the data set size.
4. **Numerosity reduction,** where the data are replaced or estimated by alternative, smaller data representations such as parametric models or nonparametric methods such as clustering, sampling, and the use of histograms,
5. **Discretization and concept hierarchy generation,** where raw data values for attributes are replaced by ranges or higher conceptual levels. Data discretization is a form of numerosity reduction that is very useful for the automatic generation of concept hierarchies. Discretization and concept hierarchy generation are powerful tools for data mining, in that they allow the mining of data at multiple levels of abstraction.

1. Data Cube Aggregation

Let's consider the situation of a company's data. This data consist of the AllElectronics sales per quarter, for the years 2002 to 2004. Now we are, generally, interested in the annual sales (total per year), rather than the total per quarter.

Thus the data can be reduced so that the resulting data summarizes the total sales per year instead of per quarter.

Year 2004	
Quarter	Sales
Q1	440
Q2	556
Q3	395
Q4	400

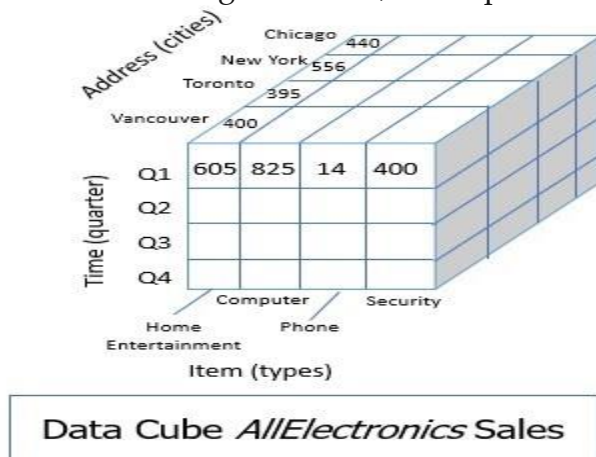
Year 2003	
Quarter	Sales
Q1	605
Q2	825
Q3	14
Q4	400

Year 2002	
Quarter	Sales
Q1	\$224,000
Q2	\$408,000
Q3	\$350,000
Q4	\$586,000

Year	Sales
2002	\$1,568,000
2003	\$2,356,000
2004	\$3,594,000

Figure 2.13 Sales data for a given branch of *AllElectronics* for the years 2002 to 2004. On the left, the sales are shown per quarter. On the right, the data are aggregated to provide the annual sales.

- Data cubes store multidimensional aggregated(summarized) information.
- Each cell holds an aggregate data value, corresponding to the data point in multidimensional space.
Data cubes provide fast access to precomputed, summarized data, thereby benefiting on-line analytical processing as well as data mining.
- The lowest level of a data cube (base cuboid)
- The cube created at the lowest level of abstraction is referred to as the base cuboid.
- The aggregated data for an individual entity of interest
- E.g., a customer in a phone calling data warehouse A cube at the highest level of abstraction is the apex cuboid.
- Multiple levels of aggregation in data cubes
- Further, reduce the size of data to deal with Queries regarding aggregated information should be answered using data cube, when possible



2. Attribute Subset Selection

Attribute Subset Selection

- Select a minimum set of features such that the probability distribution of different classes given the values for those features is as close as possible to the original distribution given the values of all features
- reduce # of attributes in the discovered patterns, easier to understand

To find Good Subset for Original Attributes

Let's consider for n attributes, there are 2^n possible subsets.

- An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as n and the number of data classes increase.
- Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection.
- These methods are typically greedy in that while searching through attribute space, they always make what looks to be the best choice at the time.
- Their strategy is to make a locally optimal choice in the hope that this will lead to a globally optimal solution.
- Such greedy methods are effective in practice and may come close to estimating an optimal solution.
- The “best” (and “worst”) attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another.

Heuristic Feature Selection Methods

1. Best step-wise forward selection:

- Here, the best single-feature is picked first.
- Then the next best feature condition to the first.

2. Step-wise backward elimination:

- In this method, it repeatedly eliminates the worst feature.

3. Best combined forward selection and backward elimination.

4. Optimal branch and bound:

- Use feature elimination and backtracking

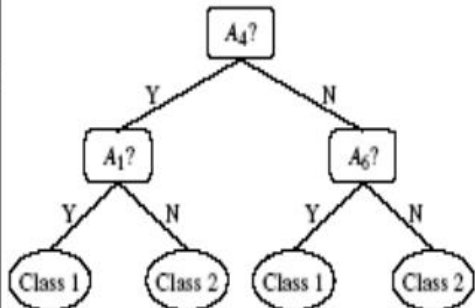
Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: $\{\}$ $\Rightarrow \{A_1\}$ $\Rightarrow \{A_1, A_4\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_4, A_5, A_6\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  <pre> graph TD A4["A4?"] -- Y --> A1["A1?"] A4 -- N --> A6["A6?"] A1 -- Y --> C1_1((Class 1)) A1 -- N --> C2_1((Class 2)) A6 -- Y --> C1_2((Class 1)) A6 -- N --> C2_2((Class 2)) </pre> $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$

Figure 2.15 Greedy (heuristic) methods for attribute subset selection.

Benefits of Data Reduction

The main benefit of data reduction is simple: the more data you can fit into a terabyte of disk space, the less capacity you will need to purchase. Here are some benefits of data reduction, such as:

- Data reduction can save energy.
- Data reduction can reduce your physical storage costs.
- And data reduction can decrease your data center track.

Data reduction greatly increases the efficiency of a storage system and directly impacts your total spending on capacity.

3. Dimensionality Reduction

Wavelet Transforms

The discrete wavelet transform (DWT) is a linear signal processing technique. It transforms a vector into a numerically different vector (D to D') of wavelet coefficients. The two vectors are of the same length. However it is useful for compression in the sense that wavelet-transformed data can be truncated. A small compressed approximation of the data can be retained by storing only a small fraction of the strongest wavelet coefficient e.g., retain all wavelet coefficients larger than some particular threshold and the remaining coefficients are set to zero. The resulting data representation is sparse. Computations that can take advantage of sparsity are very fast if performed in wavelet space.

Given a set of coefficients, an approximation of the original data can be got by applying the inverse DWT. The DWT is closely related to the discrete Fourier transform (DFT) a signal processing technique involving sine's and cosines. The general procedure for applying a discrete wavelet transform uses a hierarchical pyramid algorithm that halves the data in each iteration, resulting in fast computational speed. The method is as follows:

1. The length, L , of the input data vector must be an integer power of 2. This condition can be met by padding the data vector with zeros as necessary.
2. Each transform involves applying two functions. The first applies some data smoothing, such as sum or weighted average. The second performs a weighted difference, which acts to bring out the detailed features of the data.
3. The two functions are applied to pairs of input data, resulting in two sets of data of length $L/2$. In general these represent a smoothed or low frequency version of the input data and the high frequency content of it.
4. The two functions are recursively applied to sets of data obtained in the previous loop, until the resulting data sets obtained are of length 2.
5. A selection of values from the data sets obtained in the above iterations are designated the wavelet coefficients of the transformed data.

Wavelet transforms can be applied to multidimensional data such as data cubes. Wavelet transforms have many real world applications, including the compression of fingerprint images, computer vision, and analysis of time-series data and data cleaning.

Principal Components Analysis

An intuitive introduction is provided for principal components analysis in this section. Let the data to be compressed consist of data vectors, from k dimensions. Principal Component Analysis or PCA searches for c k -dimensional orthogonal vectors that can be best used to represent the data, where $c \leq k$. The original data are thus projected onto a much smaller space. PCA can be used to perform dimensionality reduction. The basic procedure is as follows:

1. The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.
2. PCA computes c orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that each point in a direction perpendicular to the others. These vectors are referred to as the principal components. The input data are a linear combination of principal components.
3. Strength. The principal components essentially serve as a new set of axes for the data, by providing important information about various.
4. Since the components are sorted according to decreasing order of "significance" elimination the weaker components can reduce the size of the data. Using the

strongest principal components it should be possible to reconstruct the original data to a good approximation.

PCA is computationally inexpensive and it can be ordered or unordered.

4. Numerosity Reduction

Techniques of luminosity reduction can be applied for the purpose of representing data in “smaller” forms. These techniques could be parametric or non-parametric. For parametric methods a model is used to estimate the data, so that typically only the data parameters need to be stored, instead of the actual data. Log-linear models, which estimate discrete multidimensional probability distributions, are an example. Non parametric methods for storing data include histograms, clustering and sampling.

Regression and Log-Linear models:

Regression and log-linear models can be used to approximate the given data. In linear regression, the data are modeled to fit a straight line. For example a random variable Y (called a response variable) can be modeled as a linear function of another variable, X (the predictor variable) with the equation.

$$Y = \alpha + \beta X$$

Where the variance of Y is assumed to be constant. The coefficients are known as regression coefficients; specify the Y intercept and the slope of the line respectively. The coefficients can be solved for by the method of least squares, which minimizes the error between the actual and the estimate.

Multiple regressions is an extension of the linear regression.

Log-Linear models approximate discrete multidimensional probability distributions. The method can be used to estimate the probability of each cell in base cuboids for a set of discretized attributes, based on smaller cuboids making up the data cube lattice.

Regression and log-linear models can both be used on sparse data although their application may be limited. These models will be further discussed later.

Histograms:

Histograms use binning to approximate data distributions and are a popular form of data reduction. A histogram for an attribute A partitions the data distribution of A into disjoint subsets, or buckets. The

buckets are displayed on a horizontal axis, while the height (and area) typically reflects the average frequency of the values represented by the bucket. Often buckets represent continuous ranges for the given attribute.

Clustering:

Clustering techniques consider data tuples as objects. They partition the objects into groups or clusters, so that objects within a cluster are “similar” to another and “dissimilar” to objects in other clusters. Similarity is often defined on the basis of how close the objects are in space, based on a distance function. The “quality” of a cluster may be represented by its diameter,

the maximum distance between any two objects in the cluster. In data reduction, cluster representations are used to replace the actual data. The effectiveness of this technique depends on the nature of the data.

Sampling::

Sampling can be used as a data reduction technique since it allows a larger data set to be represented by a much smaller random (or subset) of the data. Suppose a large data set D , contains N tuples some of the possible samples for D are:

- Simple random sample without replacement of size n : This created by drawing n of the N tuples from D ($n < N$), where the probability of drawing any tuple in D is $1/N$, that is all the tuples are equally likely.
- Simple random sample with replacement of size n : This is similar to the above except that each time a tuple is drawn from D , it is recorded and then replaced. That is after a tuple is drawn, it is placed back in D so that it could be drawn again.
- Cluster sample: If the tuples in D are grouped into M mutually disjoint "clusters" then a simple random sample of m clusters can be obtained, where $m < M$. A reduced data set representation can be obtained by applying say SRSWOR to the pages, resulting in a cluster sample of the tuples.
 - Stratified sample: If D is divided into mutually disjoint parts called strata, a stratified random sample is obtained by simple random sample at each stratum.

➤ Association Analysis

Association analysis is useful for discovering interesting relationships hidden in large data sets. The uncovered relationships can be represented in the form of association rules or sets of frequent items.

For example, given a table of market basket transactions

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

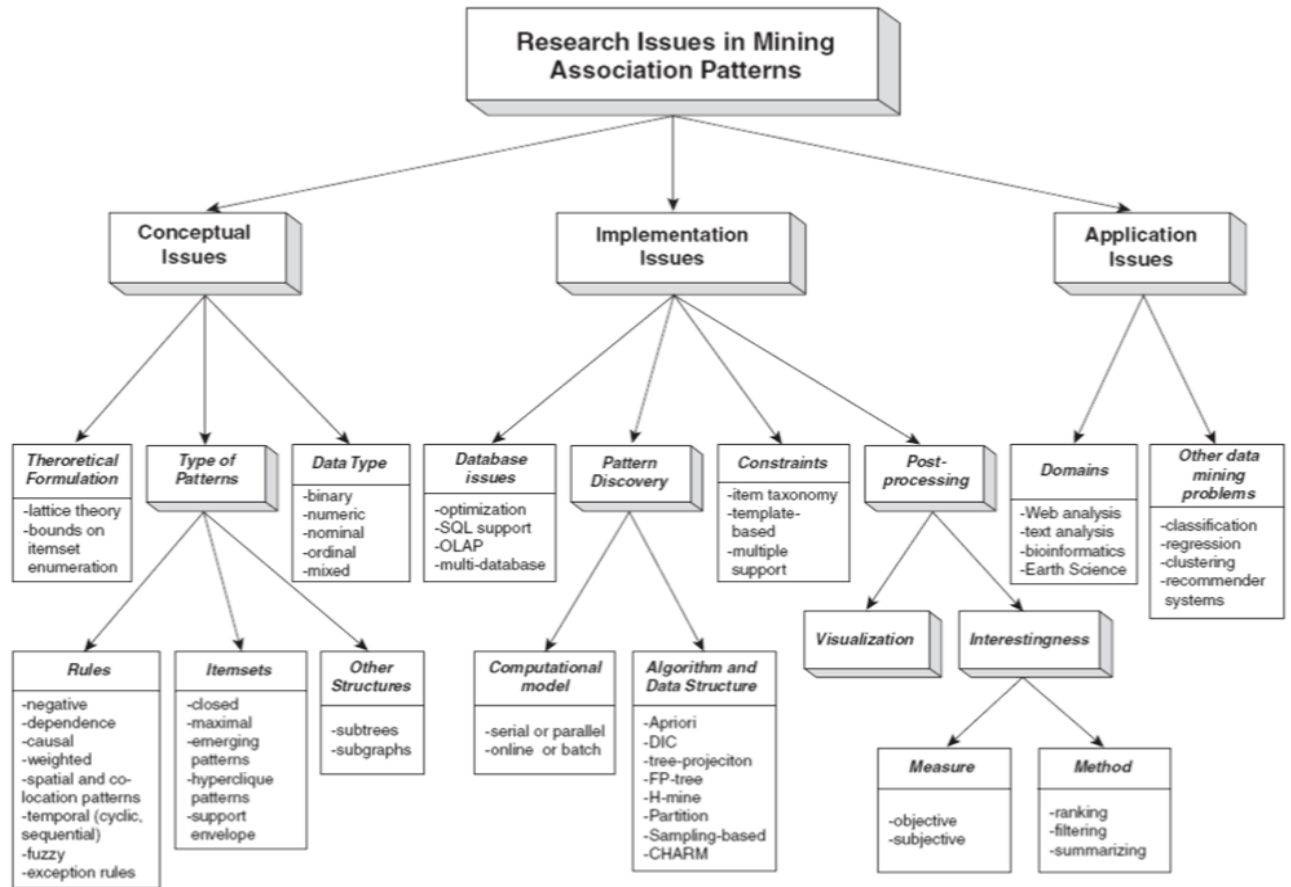
market basket data, association analysis is also applicable to other application domains such as bioinformatics, medical diagnosis, Web mining, and scientific data analysis. In the analysis of Earth science data, for example, the association patterns may reveal interesting connections among the ocean, land, and atmospheric processes. Such information may help Earth scientists develop a better understanding of how the different elements of the Earth system interact with each other. Even though the techniques presented here are generally applicable to a wider variety of data sets. There are two key issues that need to be addressed when applying association analysis to market basket data. First, discovering patterns from a large transaction data set can be computationally expensive. Second, some of the discovered patterns are potentially spurious because they may happen simply by chance.

Association Rule An association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence

a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. Frequent Itemset Generation, whose objective is to find all the item- sets that satisfy the minsup threshold. These itemsets are called frequent itemsets.
2. Rule Generation, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

The computational requirements for frequent itemset generation are generally more expensive than those of rule generation.



Definitions

Support Count

$$\sigma(X) = |\{t_i | X \subset t_i, t_i \in T\}| \quad \sigma(X) = |\{t_i | X \subset t_i, t_i \in T\}|$$

- $I = i_1, i_2, \dots, i_N$ is the set of all items
- $T = t_1, t_2, \dots, t_N$ is the set of all transactions
- Each t_i is a transaction which contains a subset of items chosen from I
- X is a subset of t_i

Association Rule

An association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets ($X \cap Y = \emptyset$).

The strength of an association rule can be measured in terms of its support and confidence. A rule that has very low support may occur simply by chance. Confidence measures the reliability of the inference made by a rule.

- **Support** of an association rule $X \rightarrow Y$
 - $\sigma(X)$ is the support count of X
 - N is the count of the transactions set T .

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \quad c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

- **Confidence** of an association rule $X \rightarrow Y$
 - $\sigma(X)$ is the support count of X
 - N is the count of the transactions set T .

$$\text{conf}(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \quad \text{conf}(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

- **Interest** of an association rule $X \rightarrow Y$
 - $P(Y) = s(Y)$ is the support of Y (fraction of baskets that contain Y)
 - If interest of a rule is close to 1, then it is uninteresting.
 - $I(X \rightarrow Y) = 1 \rightarrow X$ and Y are independent
 - $I(X \rightarrow Y) > 1 \rightarrow X$ and Y are positive correlated
 - $I(X \rightarrow Y) < 1 \rightarrow X$ and Y are negative correlated
- $$I(X \rightarrow Y) = \frac{P(X, Y)}{P(X) \times P(Y)}$$

For example, given a table of market basket transactions:

TID	Items
1	{Bread, Milk}
2	{Bread, Diaper, Beer, Eggs}
3	{Milk, Diaper, Beer, Coke}
4	{Bread, Milk, Diaper, Beer}
5	{Bread, Milk, Diaper, Coke}

We can conclude that

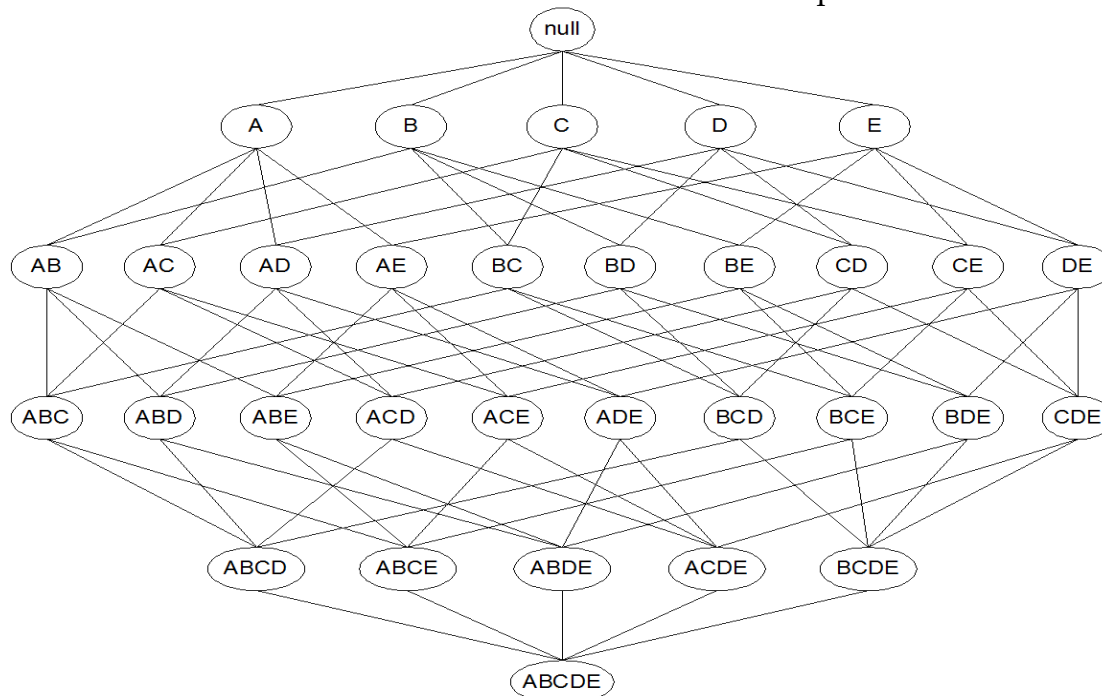
$$s(\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}) = 2/5 = 0.4$$

$$\text{conf}(\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}) = 2/3 = 0.67$$

$$I(\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}) = 2/5 \times 3/5 = 1.11$$

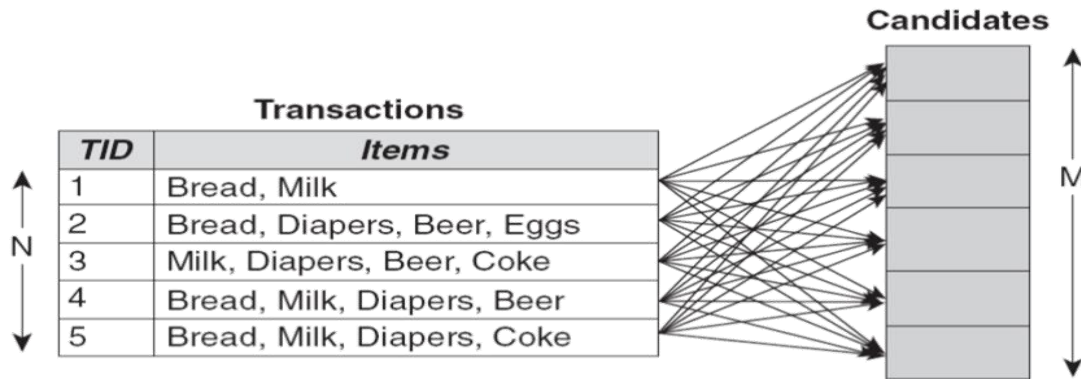
➤ Frequent Itemset Generation

A lattice structure can be used to enumerate the list of all possible itemsets:



However, the cost of frequent itemset generation is large. Given d items, there are 2^d possible candidate itemsets. There are several ways to reduce the computational complexity of frequent itemset generation:

1. Reduce the number of candidate itemsets (MM): The Apriori Principle
2. Reduce the number of comparison while counting supports: By using more advanced data structures, we can reduce the number of comparisons for matching each candidate itemset against every transaction.



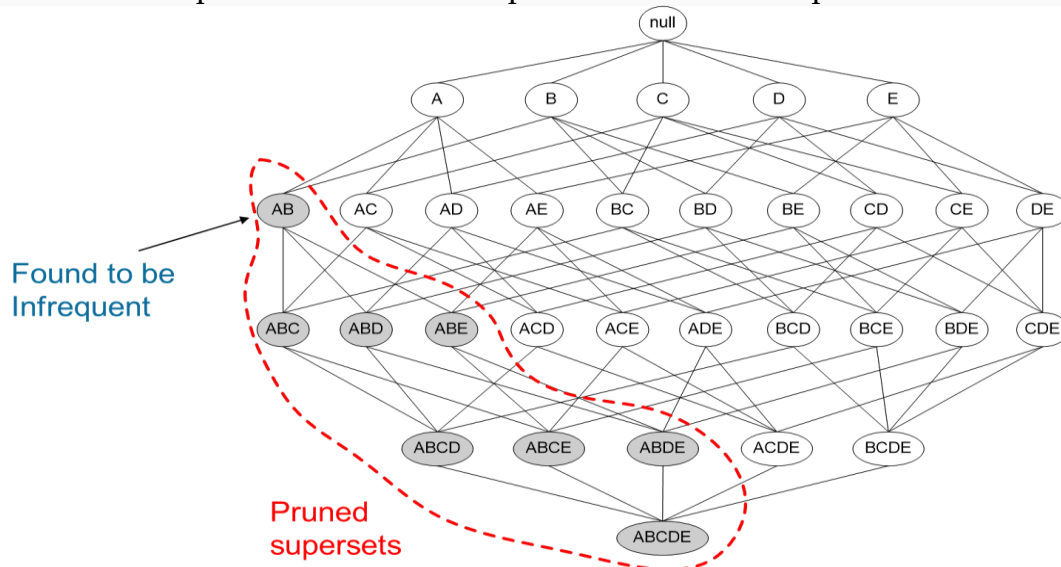
Factors Affecting Complexity:

- **Choice of minimum support threshold** lowering support threshold results in more frequent itemsets. This may increase number of candidates and max length of frequent itemsets
- **Dimensionality (number of items) of the data set** More space is needed to store support count of each item.
- **Number of transactions** Since Apriori makes multiple passes, run time of algorithm may increase with number of transactions.
- **Average transaction width** This may increase max length of frequent itemsets and traversals of hash tree.

Frequent Itemset Generation Using Apriori

The Apriori Principle:

If an itemset is frequent, then all of its subsets must also be frequent. Conversely, if a subset is infrequent, then all of its supersets must be infrequent, too.

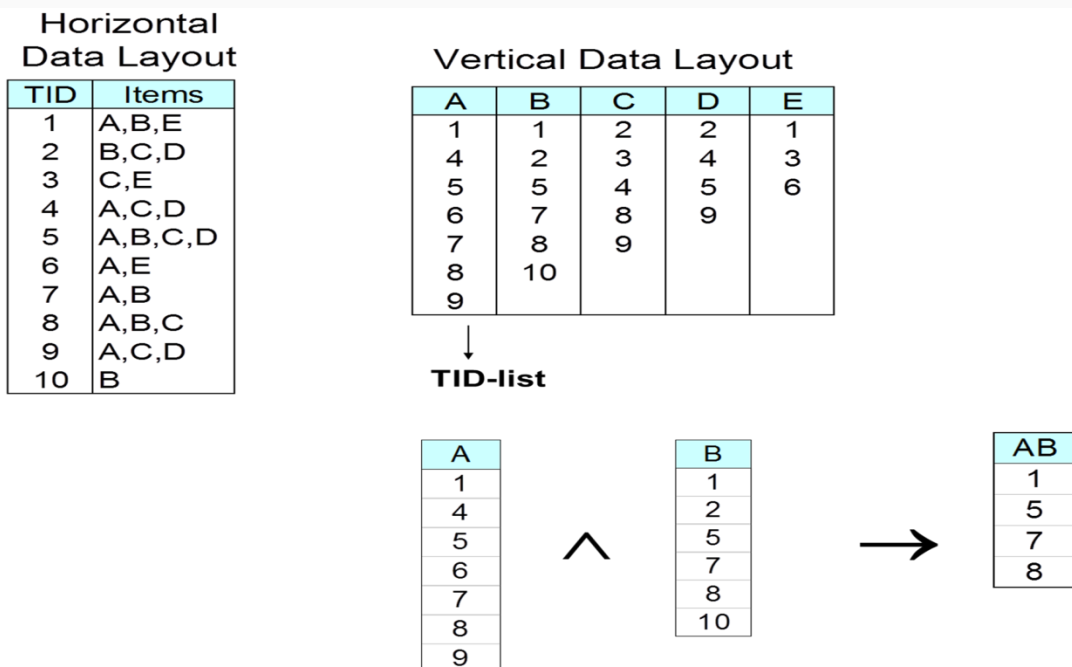


Apriori Algorithm:

- Generate frequent itemsets of length k (initially k=1)
- Repeat until no new frequent itemsets are identified
 - Generate length (k+1) candidate itemsets from length k frequent itemsets
 - Prune length (k+1) candidate itemsets that contain subsets of length k that are infrequent
 - Count the support of each candidate
 - Eliminate length (k+1) candidates that are infrequent
- k=1
- F[1] = all frequent 1-itemsets
- **while** F[k] != Empty Set:
 - k += 1
 - ## Candidate Itemsets Generation and Pruning
 - C[k] = candidate itemsets generated by F[k-1]
 - ## Support Counting
 - **for** transaction t **in** T:
 - C[t] = subset(C[k], t) # Identify all candidates that belong to t
 - **for** candidate itemset c **in** C[t]:
 - support_count[c] += 1 # Increment support count
 - F[k] = {c | c **in** C[k] **and** support_count[c] >= N*minsup}
 - **return** F[k] **for** all values of k

Frequent Itemset Generation Using ECLAT

Instead of horizontal data layout, ECLAT uses vertical data layout to store a list of transaction ids for each item.



- Advantage: very fast support counting
- Disadvantage: intermediate tid-lists may become too large for memory

Frequent Itemset Generation Using FP-Growth

FP-Growth uses FP-tree (Frequent Pattern Tree), a compressed representation of the database. Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets.

Build Tree

How to construct a FP-Tree?

1. Create the root node (null)
2. Scan the database, get the frequent itemsets of length 1, and sort these 1-itemsets in decreasing support count.
3. Read a transaction at a time. **Sort** items in the transaction according to the last step.
4. For each transaction, insert items to the FP-Tree from the root node and increment occurrence record at every inserted node.
5. Create a new child node if reaching the leaf node before the insertion completes.
6. If a new child node is created, **link** it from the last node consisting of the same item.

Mining Tree

How to mine the frequent itemsets using the FP-Tree?

FP-Growth finds all the frequent itemsets ending with a particular suffix by employing a divide-and-conquer strategy to split the problem into smaller subproblems.

1. Using the pointer in the header table, decompose the FP-Tree into multiple subtrees, each represent a subproblem (ex. finding frequent itemsets ending in e)
2. For each subproblem, traverse the corresponding subtree bottom-up to obtain **conditional pattern bases** for the subproblem recursively.

Benefits of using FP-Tree structure:

- No need to generate itemset candidates
- No need to scan the database frequently. (Only 2 pass is needed.)

➤ Rule Generation

Given a frequent itemset L , find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement.

If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \emptyset$ and $\emptyset \rightarrow L$)

So how to efficiently generate rules from frequent itemsets?

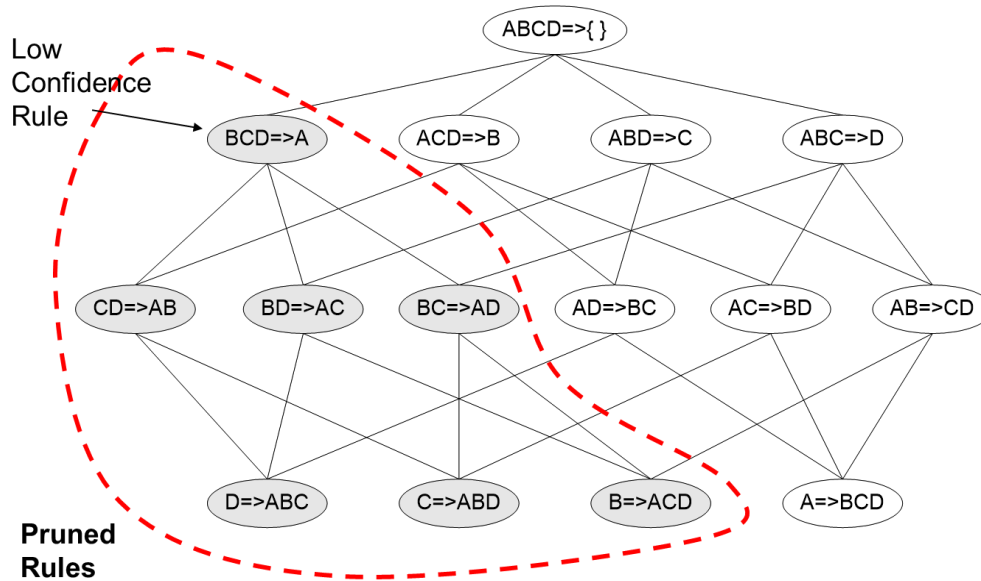
Confidence-Based Pruning

In general, confidence does not have an anti-monotone property. But confidence of rules generated from the same itemset has an anti-monotone property:

$L = \{A, B, C, D\}$
 $c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$

If we compare rules generated from the same frequent itemset Y , the following theorem holds for the confidence measure:

If a rule $X \rightarrow Y$ does not satisfy the confidence threshold, then any rule $X' \rightarrow Y$ (where $X' \subset X$), must not satisfy the confidence threshold as well.



Rule Generation in Apriori Algorithm

Candidate rule is generated by merging two rules that share the same prefix in the rule consequent.

F = frequent k -itemsets for $k \geq 2$

for itemset f in F :

$H[1]$ = 1-itemsets in f

 rules += ap_genrules(f , $H[1]$)

return rules

def ap_genrules(f , H):

k = size of itemset f

m = size of itemsets in H

 if $k > m+1$:

$H[m+1]$ = $(m+1)$ -itemsets in f

 for h in $H[m+1]$:

$\text{conf} = \text{support_count}[f] / \text{support_count}[f-h]$

 if $\text{conf} \geq \text{minconf}$:

 output the rule $(f-h) \rightarrow h$

 else:

$H[m+1] = H[m+1] - h$

 ap_genrules(f , $H[m+1]$)

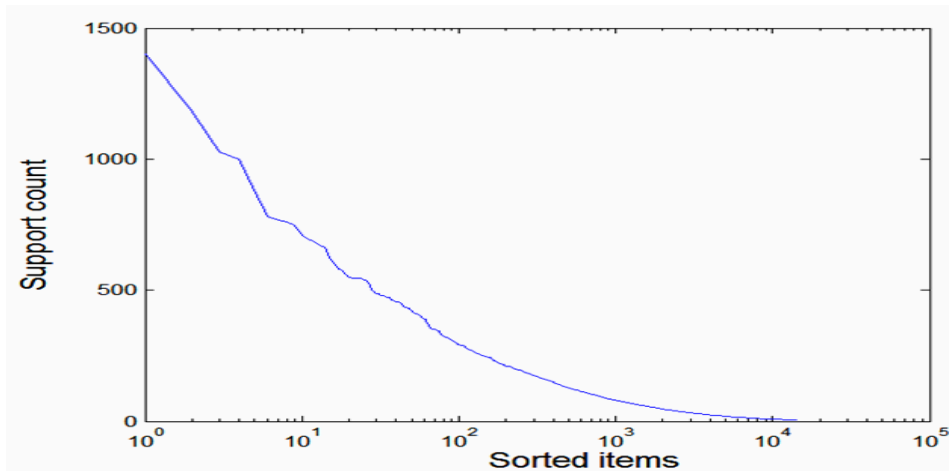
➤ Compact Representation of Frequent Itemsets

In practice, the number of frequent itemsets produced from a transaction data set can be very large. It is useful to identify a small representation set of itemsets from which all other frequent itemsets can be derived.

To reduce the number of rules we can post-process them and only output:

- Maximal frequent itemsets
 - No immediate superset is frequent
 - Gives more pruning
- Closed itemsets

- Using a single minimum support threshold may not be effective since many real data sets have skewed support distribution.
- If minsup is set too high, we could miss itemsets involving interesting rare items. (e.g., expensive products)
- If minsup is set too low, it is computationally expensive and the number of itemsets is very large.



But how to apply multiple minimum supports?

$MS(i)$ = minimum support for item i
 $MS(A,B) = \min(MS(A), MS(B))$
 $MS(i)$ = minimum support for item i
 $MS(A,B) = \min(MS(A), MS(B))$

For example, given

- $MS(\text{Milk}) = 5\%$
- $MS(\text{Coke}) = 3\%$
- $MS(\text{Broccoli}) = 0.1\%$
- $MS(\text{Salmon}) = 0.5\%$
- $MS(\{\text{Milk}, \text{Broccoli}\}) = \min(MS(\text{Milk}), MS(\text{Broccoli})) = 0.1\%$
- However, with multiple minimum support, support is no longer anti-monotone.

Need to modify Apriori such that C_{k+1} (Candidate itemsets of size $(k+1)$) is generated from L_k (set of items whose support is $\geq MS(k)$) instead of F_k (set of frequent k -itemset candidates.)

➤ Alternative methods for generating Frequent Item sets

The performance of the Apriori algorithm may degrade significantly for dense data sets because of the increasing width of transactions. Several alternative methods have been developed to overcome these limitations and improve upon the efficiency of the Apriori algorithm. The following is a high-level description of these methods.

Traversal of Itemset Lattice A search for frequent itemsets can be conceptually viewed as a traversal on the itemset lattice shown in Figure . The search strategy employed by an algorithm dictates how the lattice structure is traversed during the frequent itemset generation process. Some search strategies are better than others, depending on the configuration of frequent itemsets in the lattice. An overview of these strategies is presented next.

- **General-to-Specific** versus Specific-to-General: The Apriori algorithm uses a general-to-specific search strategy, where pairs of frequent $(k-1)$ -itemsets are merged to obtain candidate k -itemsets. This general to-specific search strategy is effective, provided the maximum length of a frequent itemset is not too long. The configuration of frequent itemsets that works best with this strategy is shown in Figure(a), where the darker nodes represent infrequent itemsets.

- **specific to-general search** strategy looks for more specific frequent itemsets first, before finding the more general frequent itemsets. This strategy is useful to discover maximal frequent itemsets in dense transactions, where the frequent itemset border is located near the bottom of the lattice, as shown in Figure 5.19(b). The Apriori principle can be applied to prune all subsets of maximal frequent itemsets. Specifically, if a candidate k -itemset is maximal frequent, we do not have to examine any of its subsets of size $k - 1$. However, if the candidate k -itemset is infrequent, we need to check all of its $k - 1$ subsets in the next iteration. Another approach is to combine both general-to-specific and specific-to-general search strategies. This bidirectional approach .

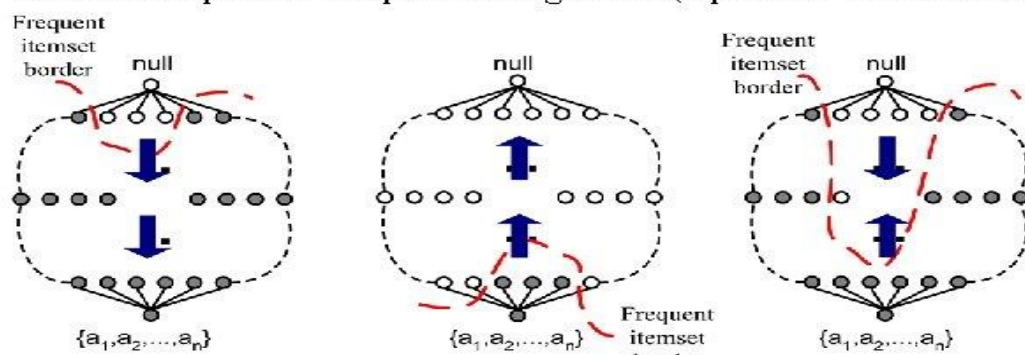
- **Equivalence Classes:** Another way to envision the traversal is to first partition the lattice into disjoint groups of nodes (or equivalence classes). A frequent itemset generation algorithm searches for frequent itemsets within a particular equivalence class first before moving to another equivalence class.

Apriori algorithm can be considered to be partitioning the lattice on the basis of itemset sizes; i.e., the algorithm discovers all frequent 1-itemsets first before proceeding to larger-sized itemsets. Equivalence classes can also be defined according to the prefix or suffix labels of an itemset

Alternative Methods for Frequent Itemset Generation

- Traversal of Itemset Lattice

- General-to-specific vs Specific-to-general (top-down vs. bottom-up)



(a) General-to-specific

(b) Specific-to-general

(c) Bidirectional

Apriori

From $(k-1)$ -itemsets, create k -itemsets (more "specific")

Used for finding Max-Frequent Itemsets

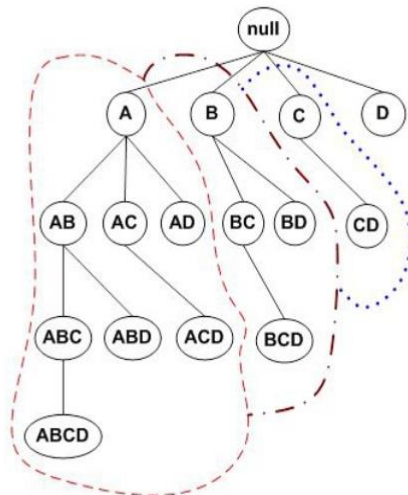
If a k -itemset in the lattice isn't Max-Freq, then we don't need to examine any of its subsets of size $(k-1)$.

Breadth-First versus Depth-First: The Apriori algorithm traverses the lattice in a breadth-first manner, as shown in Figure(a). It first discovers all the frequent 1-itemsets, followed by the frequent 2-itemsets, and so on, until no new frequent itemsets are generated. The itemset lattice can also be traversed in a depth-first manner, as shown in Figures (b). The algorithm can start from, say, node a in Figure , and count its support to determine whether it is frequent. If so, the algorithm progressively expands the next level of nodes, i.e., ab, abc, and so on, until an infrequent node is reached, say, abcd. It then backtracks to another branch, say, abce, and continues the search from there. The depth-first approach is often used by algorithms designed to find maximal frequent itemsets. This approach allows the frequent itemset border to be detected more quickly than using a breadth-first approach.

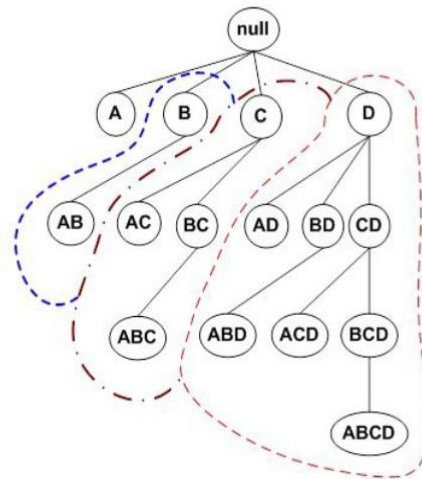
Alternative Methods for Frequent Itemset Generation

Traversal of Itemset Lattice

- Search within an equivalence class first before moving to another equivalence class.
 - APRIORI algorithm (implicitly) partitions itemsets into equivalence classes based on their length (same length – same class)
 - However, we can search by partitioning (implicitly) according to the prefix or suffix labels of an itemset.



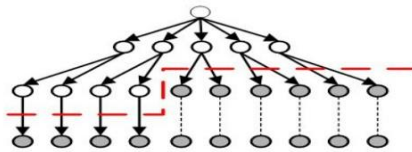
(a) Prefix tree



(b) Suffix tree

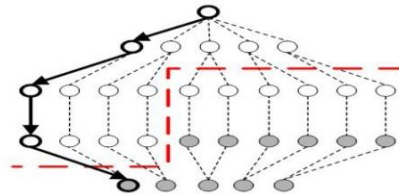
Alternative Methods for Frequent Itemset Generation

- Traversal of Itemset Lattice
 - Breadth-first vs. Depth-first



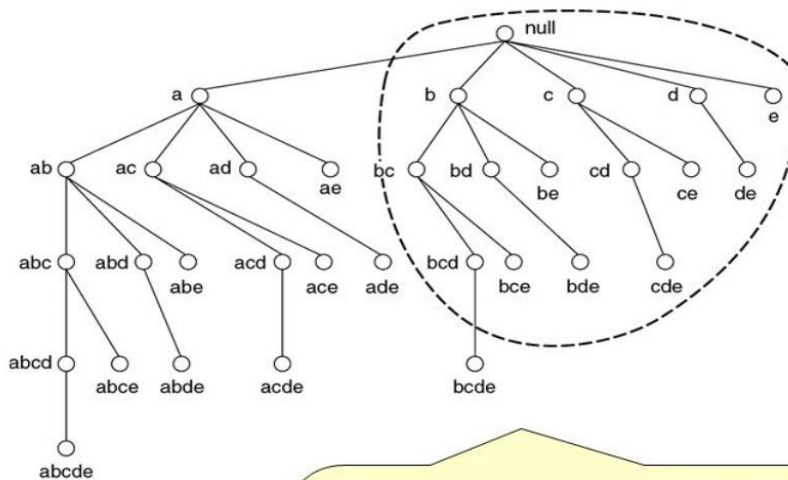
(a) Breadth first

APRIORI algorithm traverses the lattice in a **breadthfirst** manner. It first discovers all the frequent 1-itemsets, followed by the frequent 2-itemsets, and so on.



(b) Depth first

Lattice can also be traversed in a **depthfirst** way. An algorithm can start from, say, node *a* and count its support to determine whether it is frequent. If so, the algorithm progressively expands the next level of nodes, i.e., *ab*, *abc*, and so on, until an infrequent node is reached, say, *abcd*. It then backtracks to another branch, say, *abce*, ...



Depthfirst approach is often used by algorithms designed to find **max-freq itemsets**.

–Allows the frequent itemset border to be detected more quickly.

Once a max-freq itemset is found, substantial pruning can be performed on its subsets.

–E.g., if *bcde* is maximal frequent, then the algorithm does not have to visit the subtrees rooted at *bd*, *be*, *c*, *d*, and *e* because they will not contain any max-freq itemsets.

Representations of Transaction Data Set There are many ways to represent a transaction data set. The choice of representation can affect the I/O costs incurred when computing the support of candidate itemsets. Figure shows two different ways of representing market basket transactions. The representation on the left is called a horizontal data layout, which is adopted by many association rule mining algorithms, including Apriori. Another possibility is to store the list of transaction identifiers (TID-list) associated with each item. Such a representation is known as the vertical data layout. The support for each candidate itemset is obtained by intersecting the TID-lists of its

subset items. The length of the TID-lists shrinks as we progress to larger sized itemsets. However, one problem with this approach is that the initial set of TID-lists might be too large to fit into main memory, thus requiring more sophisticated techniques to compress the TID-lists. We describe another effective approach to represent the data in the next section.

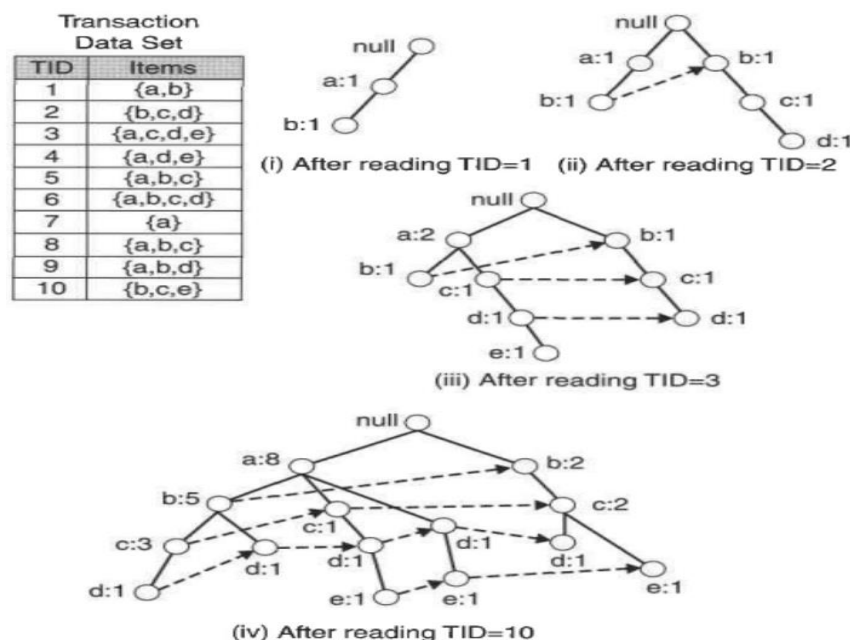
➤ **FP Growth Algorithm**

This algorithm is an improvement to the Apriori method. A frequent pattern is generated without the need for candidate generation. FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree.

This tree structure will maintain the association between the itemsets. The database is fragmented using one frequent item. This fragmented part is called “pattern fragment”. The itemsets of these fragmented patterns are analyzed. Thus with this method, the search for frequent itemsets is reduced comparatively.

FP-Tree Representation

An FP-tree is a compressed representation of the input data. It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree. As different transactions can have several items in common, their paths may overlap. The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure. If the size of the FP-tree is small enough to fit into main memory, this will allow us to extract frequent itemsets directly from the structure in memory instead of making repeated passes over the data stored on disk.



The structures of the FP-tree after reading the first three transactions are also depicted in the diagram. Each node in the tree contains the label of an item along

with a counter that shows the number of transactions mapped onto the given path. Initially, the FP-tree contains only the root node represented by the null symbol. The FP-tree is subsequently extended in the following way:

1. The data set is scanned once to determine the support count of each item. Infrequent items are discarded, while the frequent items are sorted in decreasing support counts. For the data set shown in Figure 6.24, a is the most frequent item, followed by b , c , d , and e .
2. The algorithm makes a second pass over the data to construct the FP-tree. After reading the first transaction, $\{a, b\}$, the nodes labeled as a and b are created. A path is then formed from $\text{null} \rightarrow a \rightarrow b$ to encode the transaction. Every node along the path has a frequency count of 1.
3. After reading the second transaction, $\{b, c, d\}$, a new set of nodes is created for items b , c , and d . A path is then formed to represent the transaction by connecting the nodes $\text{null} \rightarrow b \rightarrow c \rightarrow d$. Every node along this path also has a frequency count equal to one. Although the first two transactions have an item in common, which is b , their paths are disjoint because the transactions do not share a common prefix.
4. The third transaction, $\{a, c, d, e\}$, shares a common prefix item (which is a) with the first transaction. As a result, the path for the third transaction, $\text{null} \rightarrow a \rightarrow c \rightarrow d \rightarrow e$, overlaps with the path for the first transaction, $\text{null} \rightarrow a \rightarrow b$. Because of their overlapping path, the frequency count for node a is incremented to two, while the frequency counts for the newly created nodes, c , d , and e , are equal to one.
5. This process continues until every transaction has been mapped onto one of the paths given in the FP-tree. The resulting FP-tree after reading all the transactions is shown at the bottom of Figure 6.24.

The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions in market basket data often share a few items in common. In the best-case scenario, where all the transactions have the same set of items, the FP-tree contains only a single branch of nodes. The worst-case scenario happens when every transaction has a unique set of items. As none of the transactions have any items in common, the size of the FP-tree is effectively the same as the size of the original data. However, the physical storage requirement for the FP-tree is higher because it requires additional space to store pointers between nodes and counters for each item.

Frequent Itemset Generation in FP-Growth Algorithm

FP-growth is an algorithm that generates frequent itemsets from an FP-tree by exploring the tree in a bottom-up fashion. The algorithm looks for frequent itemsets ending in e first, followed by d , c , b , and finally, a . This bottom-up strategy for finding frequent item sets ending with a particular item is equivalent to the suffix based approach. Since every transaction is mapped onto a path in the FP-tree, we can derive the frequent itemsets ending with a particular item, say, e , by examining only the paths containing node e . These paths can be accessed rapidly using the pointers associated with node e .

1) The first step is to scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database is called support count or frequency of 1-itemset.

2) The second step is to construct the FP tree. For this, create the root of the tree. The root is represented by null.

3) The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count and so on. It means that the branch of the tree is constructed with transaction itemsets in descending order of count.

4) The next transaction in the database is examined. The itemsets are ordered in descending order of count. If any itemset of this transaction is already present in another branch (for example in the 1st transaction), then this transaction branch would share a common prefix to the root.

This means that the common itemset is linked to the new node of another itemset in this transaction.

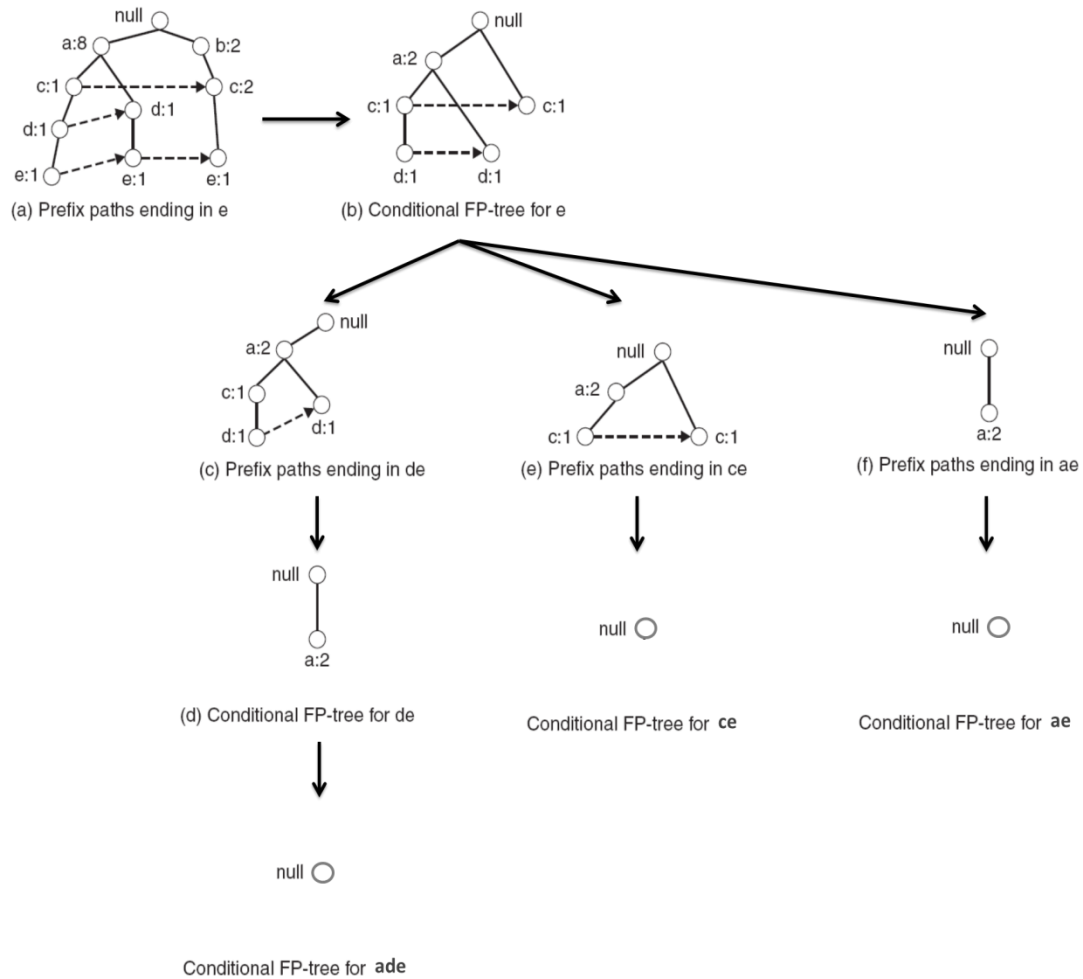
5) Also, the count of the itemset is incremented as it occurs in the transactions. Both the common node and new node count is increased by 1 as they are created and linked according to transactions.

6) The next step is to mine the created FP Tree. For this, the lowest node is examined first along with the links of the lowest nodes. The lowest node represents the frequency pattern length 1. From this, traverse the path in the FP Tree. This path or paths are called a conditional pattern base.

Conditional pattern base is a sub-database consisting of prefix paths in the FP tree occurring with the lowest node (suffix).

7) Construct a Conditional FP Tree, which is formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the Conditional FP Tree.

8) Frequent Patterns are generated from the Conditional FP Tree.



Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e}, {a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

Advantages of FP Growth Algorithm

1. This algorithm needs to scan the database only twice when compared to Apriori which scans the transactions for each iteration.
2. The pairing of items is not done in this algorithm and this makes it faster.
3. The database is stored in a compact version in memory.
4. It is efficient and scalable for mining both long and short frequent patterns.

Disadvantages of FP-Growth Algorithm

1. FP Tree is more cumbersome and difficult to build than Apriori.
2. It may be expensive.
3. When the database is large, the algorithm may not fit in the shared memory.

➤ Evaluation of Association Patterns

Association rule algorithms tend to produce too many rules. Many of them are uninteresting or redundant.

Objective Evaluation

An objective measure is a data-driven approach for evaluating the quality of association patterns. It is domain-independent and requires minimal input from the users. Patterns that involve a set of mutually independent items or cover very few transactions are considered uninteresting because they may capture spurious relationships in the data. Such patterns can be eliminated by applying an objective interestingness measure.

An objective measure is usually computed based on contingency table. For example, the table below is a 2-way contingency table for variable AA and BB.

	BB	B ⁻ B ⁻	
AA	f ₁₁ f ₁₁	f ₁₀ f ₁₀	f ₁₊ f ₁₊
A ⁻ A ⁻	f ₀₁ f ₀₁	f ₀₀ f ₀₀	f ₀₊ f ₀₊
	f ₊₁ f ₊₁	f ₊₁ f ₊₁	NN

- $f_{11}=N \times P(A,B)$ $f_{11}=N \times P(A,B)$ denotes the number of transaction that contains AA and BB.
- $f_{10}=N \times P(A,B^-)$ $f_{10}=N \times P(A,B^-)$ denotes the number of transaction that contains AA but not BB.
- $f_{1+}=N \times P(A)$ $f_{1+}=N \times P(A)$ denotes the support count for AA.
- $f_{+1}=N \times P(B)$ $f_{+1}=N \times P(B)$ denotes the support count for BB.

The pitfall of confidence can be traced to the fact that the measure ignores the support of the itemset in the rule consequent (e.g. $P(B)P(B)$ in the above case).

Subjective Evaluation

A pattern is considered subjectively uninteresting unless it reveals unexpected information about the data. Unlike Objective measures, which rank patterns based on statistics computed from data, subjective measures rank patterns according to user's interpretation.

Objective Measures of Interestingness

Interest of an association rule $X \rightarrow Y$ $X \rightarrow Y$

- $P(Y)=s(Y)$ $P(Y)=s(Y)$ is the support of Y (fraction of baskets that contain Y)
- If interest of a rule is close to 1, then it is uninteresting.
- $I(X \rightarrow Y)=1 \rightarrow X$ $I(X \rightarrow Y)=1 \rightarrow X$ and Y are independent
- $I(X \rightarrow Y)>1 \rightarrow X$ $I(X \rightarrow Y)>1 \rightarrow X$ and Y are positive correlated
- $I(X \rightarrow Y)<1 \rightarrow X$ $I(X \rightarrow Y)<1 \rightarrow X$ and Y are negative correlated

$$I(X \rightarrow Y) = \frac{P(X, Y)}{P(X) \times P(Y)} \quad I(X \rightarrow Y) = \frac{P(X, Y)}{P(X) \times P(Y)}$$

- **Lift** of an association rule $X \rightarrow Y$ $X \rightarrow Y$
- $P(Y|X)=\frac{P(X, Y)}{P(X)}=f_{11}/f_{1+}$ $P(Y|X)=\frac{P(X, Y)}{P(X)}=f_{11}/f_{1+}$
- $P(Y)=f_{+1}/N$ $P(Y)=f_{+1}/N$

$$\text{Lift}(X \rightarrow Y) = \frac{P(Y|X)}{P(Y)} \quad \text{Lift}(X \rightarrow Y) = \frac{P(Y|X)}{P(Y)}$$

There are lots of measures proposed in the literature. Some measures are good for certain applications, but not for others:

#	Measure	Formula
1	ϕ -coefficient	$\frac{P(A,B) - P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
2	Goodman-Kruskal's (λ)	$\frac{\sum_j \max_k P(A_j, B_k) + \sum_k \max_j P(A_j, B_k) - \max_j P(A_j) - \max_k P(B_k)}{2 - \max_j P(A_j) - \max_k P(B_k)}$
3	Odds ratio (α)	$\frac{P(A,B)P(\bar{A},\bar{B})}{P(A,\bar{B})P(\bar{A},B)}$
4	Yule's Q	$\frac{P(A,B)P(\bar{A}\bar{B}) - P(A,\bar{B})P(\bar{A},B)}{P(A,B)P(\bar{A}\bar{B}) + P(A,\bar{B})P(\bar{A},B)} = \frac{\alpha-1}{\alpha+1}$
5	Yule's Y	$\frac{\sqrt{P(A,B)P(\bar{A}\bar{B})} - \sqrt{P(A,\bar{B})P(\bar{A},B)}}{\sqrt{P(A,B)P(\bar{A}\bar{B})} + \sqrt{P(A,\bar{B})P(\bar{A},B)}} = \frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$
6	Kappa (κ)	$\frac{P(A,B) + P(\bar{A},\bar{B}) - P(A)P(B) - P(\bar{A})P(\bar{B})}{1 - P(A)P(B) - P(\bar{A})P(\bar{B})}$
7	Mutual Information (M)	$\frac{\sum_i \sum_j P(A_i, B_j) \log \frac{P(A_i, B_j)}{P(A_i)P(B_j)}}{\min(-\sum_i P(A_i) \log P(A_i), -\sum_j P(B_j) \log P(B_j))}$
8	J-Measure (J)	$\max \left(P(A, B) \log \left(\frac{P(B A)}{P(B)} \right) + P(\bar{A}\bar{B}) \log \left(\frac{P(\bar{B} \bar{A})}{P(\bar{B})} \right), \right.$ $\left. P(A, B) \log \left(\frac{P(A B)}{P(A)} \right) + P(\bar{A}\bar{B}) \log \left(\frac{P(\bar{A} \bar{B})}{P(\bar{A})} \right) \right)$
9	Gini index (G)	$\max \left(P(A)[P(B A)^2 + P(\bar{B} A)^2] + P(\bar{A})[P(B \bar{A})^2 + P(\bar{B} \bar{A})^2] \right.$ $\left. - P(B)^2 - P(\bar{B})^2, \right.$ $\left. P(B)[P(A B)^2 + P(\bar{A} B)^2] + P(\bar{B})[P(A \bar{B})^2 + P(\bar{A} \bar{B})^2] \right.$ $\left. - P(A)^2 - P(\bar{A})^2 \right)$
10	Support (s)	$P(A, B)$
11	Confidence (c)	$\max(P(B A), P(A B))$
12	Laplace (L)	$\max \left(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2} \right)$
13	Conviction (V)	$\max \left(\frac{P(A)P(\bar{B})}{P(\bar{A}\bar{B})}, \frac{P(B)P(\bar{A})}{P(\bar{B}\bar{A})} \right)$
14	Interest (I)	$\frac{P(A,B)}{P(A)P(B)}$
15	cosine (IS)	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
16	Piatetsky-Shapiro's (PS)	$P(A, B) - P(A)P(B)$
17	Certainty factor (F)	$\max \left(\frac{P(B A) - P(B)}{1 - P(B)}, \frac{P(A B) - P(A)}{1 - P(A)} \right)$
18	Added Value (AV)	$\max(P(B A) - P(B), P(A B) - P(A))$
19	Collective strength (S)	$\frac{P(A,B) + P(\bar{A}\bar{B})}{P(A)P(B) + P(\bar{A})P(\bar{B})} \times \frac{1 - P(A)P(B) - P(\bar{A})P(\bar{B})}{1 - P(A,B) - P(\bar{A}\bar{B})}$
20	Jaccard (ζ)	$\frac{P(A,B)}{P(A) + P(B) - P(A,B)}$
21	Klogsen (K)	$\sqrt{P(A, B) \max(P(B A) - P(B), P(A B) - P(A))}$

Properties of Good Objective Measure

Inversion Property

An objective measure MM is invariant under the inversion operation if its value remains the same when exchanging the frequent counts f11f11 with f10f10 and f10f10 with f01f01.

Null Addition Property

An objective measure MM is invariant under the null addition operation if it is not affected by increasing f00f00, while all other frequencies in the contingency table stay the same.

Scaling Invariance Property

An objective measure MM is invariant under the row/column scaling operation.

Basic Concepts

The general approach to classification as a two-step process. In the first step, we build a classification model based on previous data. In the second step, we determine if the model's accuracy is acceptable, and if so, we use the model to classify new data.

➤ What Is Classification?

A bank loans officer needs analysis of her data to learn which loan applicants are “safe” and which are “risky” for the bank. A marketing manager at AllElectronics needs data analysis to help guess whether a customer with a given profile will buy a new computer. A medical researcher wants to analyze breast cancer data to predict which one of three specific treatments a patient should receive. In each of these examples, the data analysis task is **classification**, where a model or **classifier** is constructed to predict class (categorical) labels, such as “safe” or “risky” for the loan application data; “yes” or “no” for the marketing data; or “treatment A,” “treatment B,” or “treatment C” for the medical data. These categories can be represented by discrete values, where the ordering among values has no meaning. For example, the values 1, 2, and 3 may be used to represent treatments A, B, and C, where there is no ordering implied among this group of treatment regimes.

Suppose that the marketing manager wants to predict how much a given customer will spend during a sale at AllElectronics. This data analysis task is an example of **numeric prediction**, where the model constructed predicts a continuous-valued function, or ordered value, as opposed to a class label. This model is a **predictor**. **Regression analysis** is a statistical methodology that is most often used for numeric prediction; hence the two terms tend to be used synonymously, although other methods for numeric prediction exist. Classification and numeric prediction are the two major types of **prediction problems**. This chapter focuses on classification.

➤ General Approach to Classification

“How does classification work?” **Data classification** is a two-step process, consisting of a learning step (where a classification model is constructed) and a classification step (where the model is used to predict class labels for given data). The process is shown for the loan application data of Figure (The data are simplified for illustrative purposes. In reality, we may expect many more attributes to be considered).

FIGURE The data classification process: (a) Learning: Training data are analyzed by a classification algorithm. Here, the class label attribute is `loan_decision`, and the learned model or classifier is represented in the form of classification rules. (b) Classification: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the **learning step** (or training phase), where a classification algorithm builds the classifier by analyzing or “learning from” a **training set** made up of database tuples and their

associated class labels. A tuple, X , is represented by an n -dimensional **attribute vector**, $X = (x_1, x_2, \dots, x_n)$,

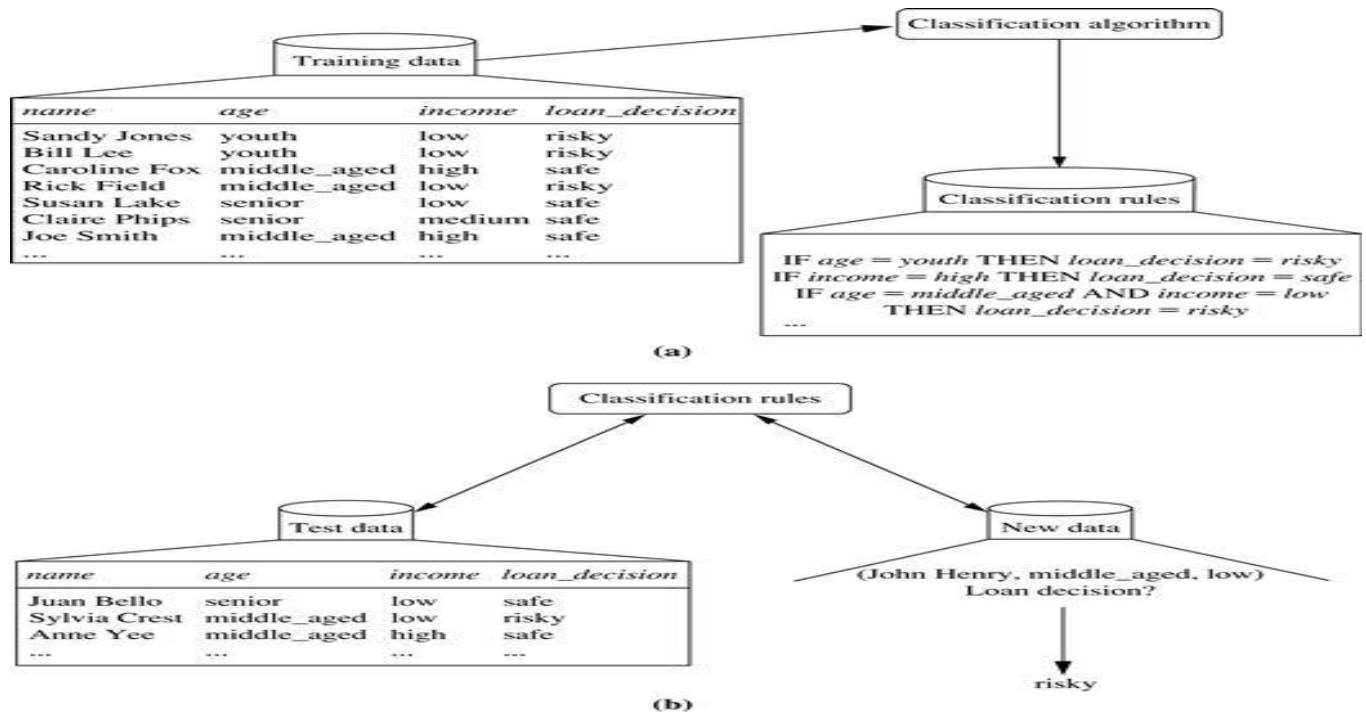
Depicting n measurements made on the tuple from n database attributes, $A_1, A_2, \dots, A_n$ respectively, Each tuple, X , is assumed to belong to a predefined class as determined by another database attribute called the **class label attribute**. The class label attribute is discrete-valued and unordered. It is categorical (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as **training tuples** and are randomly sampled from the database under analysis. In the context of classification, data tuples can be referred to as samples, examples, instances, data points, or objects.

Because the class label of each training tuple is provided, this step is also known as **supervised learning** (i.e., the learning of the classifier is “supervised” in that it is told to which class each training tuple belongs). It contrasts with **unsupervised learning** (or **clustering**), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance. For example, if we did not have the loan decision data available for the training set, we could use clustering to try to determine “groups of like tuples,” which may correspond to risk groups within the loan application data.

This first step of the classification process can also be viewed as the learning of a mapping or function, f , that can predict the associated class label y of a given tuple X . In this view, we wish to learn a mapping or function that separates the data classes. Typically, this mapping is represented in the form of classification rules, decision trees, or mathematical formulae. In our example, the mapping is represented as classification rules that identify loan applications as being either safe or risky. The rules can be used to categorize future data tuples, as well as provide deeper insight into the data contents. They also provide a compressed data representation.

“What about classification accuracy?” In the second step the model is used for classification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier’s accuracy, this estimate would likely be optimistic, because the classifier tends to **overfit** the data (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a **test set** is used, made up of **test tuples** and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier.

The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared



with the learned classifier's class prediction for that tuple.

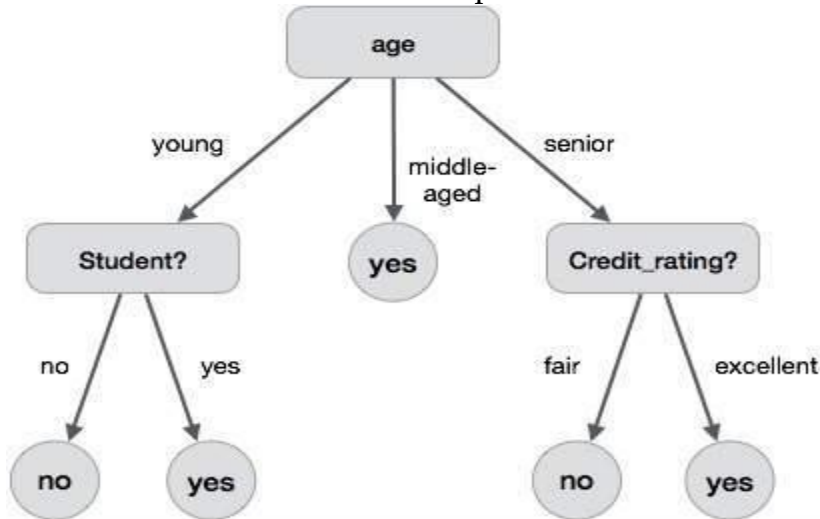
➤ Improving accuracy of classification:

- **Accuracy:** The accuracy of the classifier can be referred to as the ability of the classifier to predict the class label correctly, and the accuracy of the predictor can be referred to as how well a given predictor can estimate the unknown value.
- **Speed:** The speed of the method depends on the computational cost of generating and using the classifier or predictor.
- **Robustness:** Robustness is the ability to make correct predictions or classifications. In the context of data mining, robustness is the ability of the classifier or predictor to make correct predictions from incoming unknown data.
- **Scalability:** Scalability refers to an increase or decrease in the performance of the classifier or predictor based on the given data.
- **Interpretability:** Interpretability is how readily we can understand the reasoning behind predictions or classification made by the predictor or classifier.

➤ Decision Trees:

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept buy_computer that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.



The benefits of having a decision tree are as follows –

- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

1.Decision Tree Induction Algorithm

A machine researcher named J. Ross Quinlan in 1980 developed a decision tree algorithm known as ID3 (Iterative Dichotomiser). Later, he presented C4.5, which was the successor of ID3. ID3 and C4.5 adopt a greedy approach. In this algorithm, there is no backtracking; the trees are constructed in a top-down recursive divide-and-conquer manner.

Generating a decision tree from training tuples of data partition D

Algorithm : Generate_decision_tree

Input:

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

Output:

A Decision Tree

Method

create a node N;

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

Input:

- Data partition, D , which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that “best” partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting_subset*.

Output: A decision tree.

Method:

- (1) create a node N ;
- (2) **if** tuples in D are all of the same class, C , **then**
- (3) return N as a leaf node labeled with the class C ;
- (4) **if** *attribute_list* is empty **then**
- (5) return N as a leaf node labeled with the majority class in D ; // majority voting
- (6) apply **Attribute_selection_method**(D , *attribute_list*) to **find** the “best” *splitting_criterion*;
- (7) label node N with *splitting_criterion*;
- (8) **if** *splitting_attribute* is discrete-valued **and**
 multiway splits allowed **then** // not restricted to binary trees
- (9) *attribute_list* $\leftarrow$ *attribute_list* – *splitting_attribute*; // remove *splitting_attribute*
- (10) **for each** outcome j of *splitting_criterion*
 // partition the tuples and grow subtrees for each partition
- (11) let D_j be the set of data tuples in D satisfying outcome j ; // a partition
- (12) **if** D_j is empty **then**
- (13) attach a leaf labeled with the majority class in D to node N ;
- (14) **else** attach the node returned by **Generate_decision_tree**(D_j , *attribute_list*) to node N ;
- endfor**
- (15) return N ;

The algorithm is called with three parameters: D , *attribute_list*, and *Attribute_selection_method*. We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter *attribute_list* is a list of attributes describing the tuples. *Attribute_selection_method* specifies a heuristic procedure for selecting the attribute that “best” discriminates the given tuples according to class. This procedure employs an attribute selection measure such as information gain or the Gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini index, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits

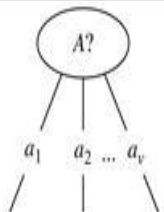
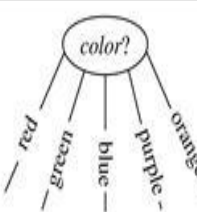
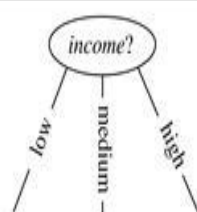
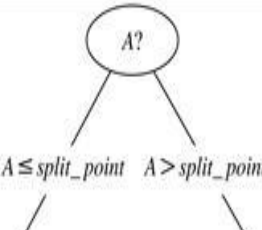
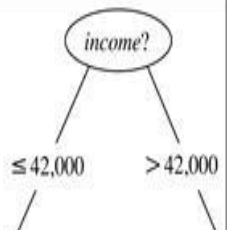
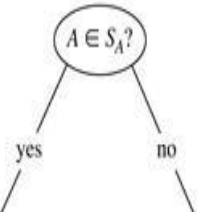
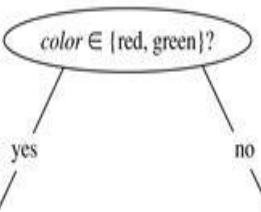
- The tree starts as a single node, N , representing the training tuples in D (step 1).
- If the tuples in D are all of the same class, then node N becomes a leaf and is labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All terminating conditions are explained at the end of the algorithm.
- Otherwise, the algorithm calls *Attribute_selection_method* to determine the **splitting criterion**. The splitting criterion tells us which attribute to test at node N by determining the “best” way to separate or partition the tuples in D into individual classes (step 6). The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the **splitting attribute** and may also

indicate either a **split-point** or a **splitting subset**. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as “pure” as possible. A partition is **pure** if all the tuples in it belong to the same class. In other words, if we split up the tuples in D according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

■ The node N is labeled with the splitting criterion, which serves as a test at the node. A branch is grown from node N for each of the outcomes of the splitting criterion. The tuples in D are partitioned accordingly

1. A is discrete-valued: In this case, the outcomes of the test at node N correspond directly to the known values of A . A branch is created for each known value, a_j , of A and labeled with that value. Partition D_j is the subset of class-labeled tuples in D having value a_j of A . Because all the tuples in a given partition have the same value for A , A need not be considered in any future partitioning of the tuples. Therefore, it is removed from `attribute_list`.

2. A is continuous-valued: In this case, the test at node N has two possible outcomes, corresponding to the conditions `split_point` and `split_point`, respectively, where `split_point` is the `split-point` returned by `Attribute_selection_method` as part of the splitting criterion. (In practice, the `split-point`, a , is often taken as the midpoint of two known adjacent values of A from the training data.) Two branches are grown from N and labeled according to the previous outcomes. The tuples are partitioned such that D_1 holds the subset of class-labeled tuples in D for which `split_point`, while D_2 holds the rest.

	Partitioning scenarios	Examples
(a)		 
(b)		
(c)		

3. A is discrete-valued and a binary tree must be produced (as dictated by the attribute selection

measure or algorithm being used): The test at node N is of the form “ $?$,” where S_A is the splitting subset for A , returned by `Attribute_selection_method` as part of the splitting criterion. It is a subset of the known values of A . If a given tuple has value a_j of A and if $a_j \in S_A$, then the test at node N is satisfied. Two branches are grown from N . By convention, the left branch out of N is labeled *yes* so that D_1 corresponds to the subset of class-labeled tuples in D that satisfy the test.

The right branch out of N is labeled *no* so that D_2 corresponds to the subset of class-labeled tuples from D that do not satisfy the test.

FIGURE 8.4 This figure shows three possibilities for partitioning tuples based on the splitting criterion, each with examples. Let A be the splitting attribute. (a) If A is discrete-valued, then one branch is grown for each known value of A . (b) If A is continuous-valued, then two branches are grown, corresponding to $A \leq \textit{split_point}$ and $A > \textit{split_point}$. (c) If A is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for A .

- The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_i , of D (step 14).
- The recursive partitioning stops only when any one of the following terminating conditions is true:
 1. All the tuples in partition D (represented at node N) belong to the same class (steps 2 and 3).
 2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting node N into a leaf and labeling it with the most common class in D . Alternatively, the class distribution of the node tuples may be stored.
 3. There are no tuples for a given branch, that is, a partition D_i is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).
- The resulting decision tree is returned (step 15).

1. Attribute Selection Measures

An attribute selection measure is a heuristic for selecting the splitting criterion that “best” separates a given data partition, D , of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class). Conceptually, the “best” splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.

The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure⁴ is chosen as the splitting attribute for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees, then, respectively, either a split point or a splitting subset must also be determined as part of the splitting criterion. The tree node created for partition D is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly. This section describes three popular attribute selection measures — information gain, gain ratio, and Gini index.

Information Gain

ID3 uses **information gain** as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or “information content” of messages. Let node N represent or hold the tuples of partition D . The

attribute with the highest information gain is chosen as the splitting attribute for node N . This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or “impurity” in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.

The expected information needed to classify a tuple in D is given by $Info(D) = -\sum_{i=1}^m p_i \log_2(p_i)$,

Gain Ratio

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. For example, consider an attribute that acts as a unique identifier such as `product_ID`. A split on `product_ID` would result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify data set D based on this partitioning would be 0. Therefore, the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

C4.5, a successor of ID3, uses an extension to information gain known as *gain ratio*, which attempts to overcome this bias. It applies a kind of normalization to information gain using a “split information” value defined analogously with

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

This value represents the potential information generated by splitting the training data set, D , into v partitions, corresponding to the v outcomes of a test on attribute A . Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in D . It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as

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

Gini Index

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

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

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|C_i.D|/|D|$. The sum is computed over m classes.

The Gini index considers a binary split for each attribute. Let's first consider the case where A is a discrete-valued attribute having v distinct values, $\{v\}$, occurring in D . To determine the best binary split on A , we examine all the possible subsets that can be formed using known values of A . Each subset, S_A , can be considered as a binary test for attribute A of the form “?” Given a tuple, this test is satisfied if the value of A for the tuple is among the values listed in S_A . If A has v possible values, then there are $2^v - 2$ possible subsets. For example, if income has three possible values, namely {low, medium, high}, then the possible subsets are {low, medium, high}, {low, medium}, {low, high}, {medium, high}, {low}, {medium}, {high}, and {}. We exclude the power set, {low, medium, high}, and the empty set from consideration since, conceptually, they do not represent a split. Therefore, there are $(2^v - 2)/2$ possible ways to form two partitions of the data, D , based on a binary split on A .

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

3 Tree Pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. Such methods typically use statistical measures to remove the least-reliable branches. An unpruned tree and a pruned version of it are shown in Figure 8.6. Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

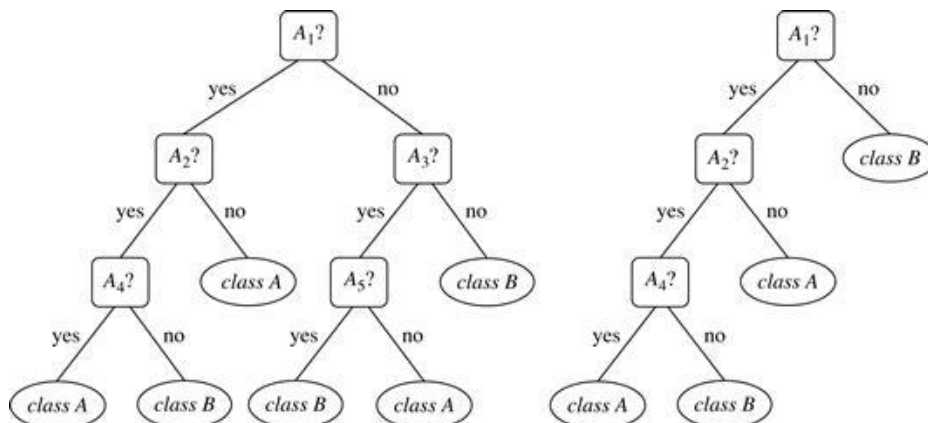


FIGURE 8.6 An unpruned decision tree and a pruned version of it.

“How does tree pruning work?” There are two common approaches to tree pruning: prepruning and postpruning.

In the prepruning approach, a tree is “pruned” by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

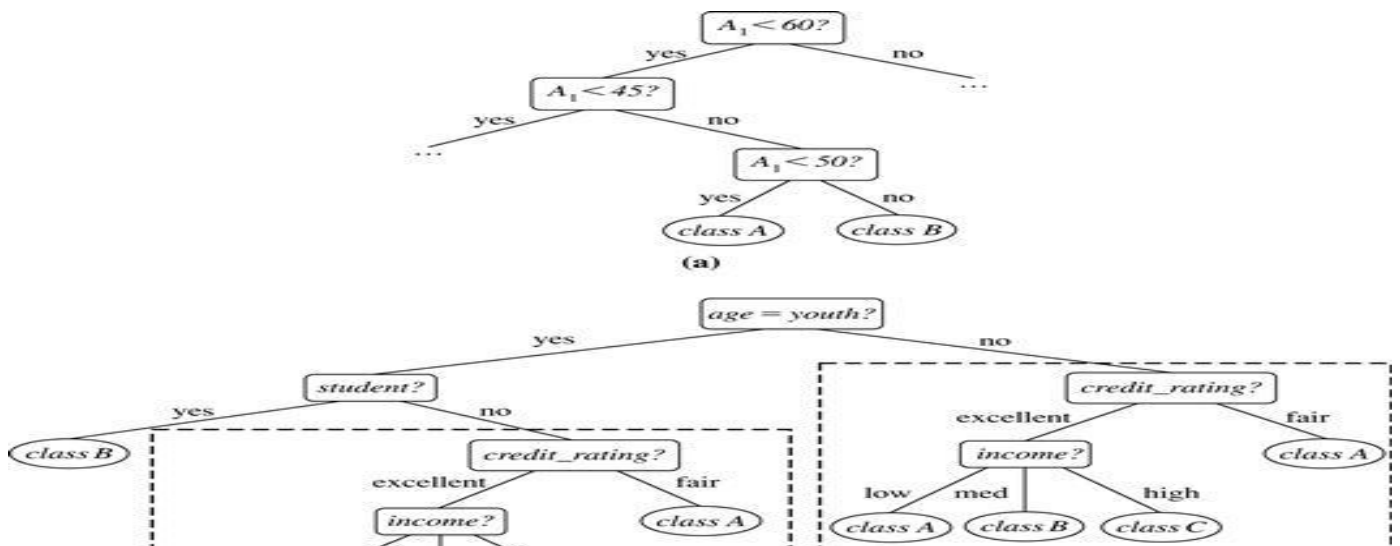
When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on, can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

The second and more common approach is postpruning, which removes subtrees from a “fully grown” tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.

The cost complexity pruning algorithm used in CART is an example of the postpruning approach. This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree (where the error rate is the percentage of tuples misclassified by the tree). It starts from the bottom of the tree. For each internal node, N , it computes the cost complexity of the subtree at N , and the cost complexity of the subtree at N if it were to be pruned (i.e., replaced by a leaf node). The two values are compared. If pruning the subtree at node N would result in a smaller cost complexity, then the subtree is pruned. Otherwise, it is kept.

Alternatively, prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree. No single pruning method has been found to be superior over all others. Although some pruning methods do depend on the availability of additional data for pruning, this is usually not a concern when dealing with large databases.

Although pruned trees tend to be more compact than their unpruned counterparts, they may still be rather large and complex. Decision trees can suffer from repetition and replication making them overwhelming to interpret. Repetition occurs when an attribute is repeatedly tested along a given branch of the tree (e.g., “age < 60?” followed by “age < 45?” and so on). In replication, duplicate subtrees exist within the tree. These situations can impede the accuracy



and comprehensibility of a decision tree. The use of multivariate splits (splits based on a combination of attributes) can prevent these problems. Another approach is to use a different form of knowledge representation, such as rules, instead of decision trees.

FIGURE An example of: (a) subtree repetition, where an attribute is repeatedly tested along a given branch of the tree (e.g., age) and (b) subtree replication, where duplicate subtrees exist within a tree (e.g., the subtree headed by the node “credit rating?”).

4 Scalability and Decision Tree Induction

“What if D, the disk-resident training set of class-labeled tuples, does not fit in memory? In other words, how scalable is decision tree induction?” The efficiency of existing decision tree algorithms, such as ID3, C4.5, and CART, has been well established for relatively small data sets. Efficiency becomes an issue of concern when these algorithms are applied to the mining of very large real-world databases. The pioneering decision tree algorithms that we have discussed so far have the restriction that the training tuples should reside in memory.

Several scalable decision tree induction methods have been introduced in recent studies. RainForest, for example, adapts to the amount of main memory available and applies to any decision tree induction algorithm. The method maintains an AVC-set (where “AVC” stands for “Attribute-Value, Classlabel”) for each attribute, at each tree node, describing the training tuples at the node. The AVC-set of an attribute A at node N gives the class label counts for each value of A for the tuples at N. Figure 8.8 shows AVC-sets for the tuple data of Table 8.1. The set of all AVC-sets at a node N is the AVC-group of N. The size of an AVC-set for attribute A at node N depends only on the number of distinct values of A and the number of classes in the set of tuples at N. Typically, this size should fit in memory, even for real-world data. RainForest also has techniques, however, for handling the case where the AVC-group does not fit in memory. Therefore, the method has high scalability for decision tree induction in very large data sets.

age	buys_computer	
	yes	no
youth	2	3
middle_aged	4	0
senior	3	2

income	buys_computer	
	yes	no
low	3	1
medium	4	2
high	2	2

student	buys_computer	
	yes	no
yes	6	1
no	3	4

credit_rating	buys_computer	
	yes	no
fair	6	2
excellent	3	3

FIGURE The use of data structures to hold aggregate information regarding the training data (e.g., these AVC-sets describing Table 8.1’s data) are one approach to improving the scalability of decision tree induction.

BOAT (Bootstrapped Optimistic Algorithm for Tree construction) is a decision tree algorithm that takes a completely different approach to scalability—it is not based on the use of any special data structures. Instead, it uses a statistical technique known as “bootstrapping” (Section 8.5.4) to create several smaller samples (or subsets) of the given training data, each of which fits in memory. Each subset is used to construct a tree, resulting in several trees. The trees are examined and used to construct a new tree, T , that turns out to be “very close” to the tree that would have been generated if all the original training data had fit in memory.

BOAT can use any attribute selection measure that selects binary splits and that is based on the notion of purity of partitions such as the Gini index. BOAT uses a lower bound on the attribute selection measure to detect if this “very good” tree, T , is different from the “real” tree, T , that would have been generated using all of the data. It refines to arrive at T .

➤ Rule-Based Classification

IF-THEN Rules

Rule-based classifier makes use of a set of IF-THEN rules for classification. We can express a rule in the following form – IF condition THEN conclusion

Let us consider a rule R_1 ,

R_1 : IF age = youth AND student = yes
THEN buy_computer = yes

Points to remember –

- The IF part of the rule is called **rule antecedent** or **precondition**.
- The THEN part of the rule is called **rule consequent**.
- The antecedent part the condition consist of one or more attribute tests and these tests are logically ANDed.
- The consequent part consists of class prediction.

Note – We can also write rule R_1 as follows –

R_1 : (age = youth) $\wedge$ (student = yes) (buys computer = yes)

If the condition holds true for a given tuple, then the antecedent is satisfied.

Rule Extraction

Here we will learn how to build a rule-based classifier by extracting IF-THEN rules from a decision tree.

Points to remember –

To extract a rule from a decision tree –

- One rule is created for each path from the root to the leaf node.
- To form a rule antecedent, each splitting criterion is logically ANDed.
- The leaf node holds the class prediction, forming the rule consequent.

Rule Induction Using Sequential Covering Algorithm

Sequential Covering Algorithm can be used to extract IF-THEN rules from the training data. We do not require to generate a decision tree first. In this algorithm, each rule for a given class covers many of the tuples of that class.

Some of the sequential Covering Algorithms are AQ, CN2, and RIPPER. As per the general strategy the rules are learned one at a time. For each time rules are learned, a tuple covered by the rule is removed and the process continues for the rest of the tuples. This is because the path to each leaf in a decision tree corresponds to a rule.

Note – The Decision tree induction can be considered as learning a set of rules simultaneously. The Following is the sequential learning Algorithm where rules are learned for one class at a time. When learning a rule from a class C_i , we want the rule to cover all the tuples from class C only and no tuple from any other class.

Algorithm: Sequential Covering

Input:

D , a data set class-labeled tuples,

Att_vals , the set of all attributes and their possible values.

Output: A Set of IF-THEN rules.

Method:

Rule_set={ }; // initial set of rules learned is empty

for each class c do

 repeat

 Rule = Learn_One_Rule (D , Att_vals , c);

 remove tuples covered by Rule from D ;

 until termination condition;

 Rule_set=Rule_set+Rule; // add a new rule to rule-set

end for

return Rule_Set;

Rule Pruning

The rule is pruned is due to the following reason –

- The Assessment of quality is made on the original set of training data. The rule may perform well on training data but less well on subsequent data. That's why the rule pruning is required.
- The rule is pruned by removing conjunct. The rule R is pruned, if pruned version of R has greater quality than what was assessed on an independent set of tuples.

FOIL is one of the simple and effective method for rule pruning. For a given rule R ,

$$FOIL_Prune = pos - neg / pos + neg$$

where pos and neg is the number of positive tuples covered by R , respectively.

Note – This value will increase with the accuracy of R on the pruning set. Hence, if the $FOIL_Prune$ value is higher for the pruned version of R , then we prune R

Rule Extraction from a Decision Tree

Decision tree classifiers are a popular method of classification—it is easy to understand how decision trees work and they are known for their accuracy. Decision trees can become large and difficult to interpret. In this subsection, we look at how to build a rule-based classifier by extracting IF-THEN rules from a decision tree. In comparison with a decision tree, the IF-

THEN rules may be easier for humans to understand, particularly if the decision tree is very large.

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent (“IF” part). The leaf node holds the class prediction, forming the rule consequent (“THEN” part).

Extracting classification rules from a decision tree

<i>R1</i>	:IF <i>Age</i>	= <i>youth</i>	AND <i>student</i> = <i>no</i>	THEN <i>buys_computer</i>	= <i>no</i>
<i>R2</i>	:IF <i>Age</i>	= <i>youth</i>	AND <i>student</i> = <i>yes</i>	THEN <i>buys_computer</i>	= <i>yes</i>
<i>R3</i>	:IF <i>Age</i>	= <i>middle_aged</i>		THEN <i>buys_computer</i>	= <i>yes</i>
<i>R4</i>	:IF <i>Age</i> = <i>senior</i>	=	AND <i>credit_rating</i> = <i>excellent</i>	THEN <i>buys_computer</i>	= <i>yes</i>
<i>R5</i>	:IF <i>Age</i> = <i>senior</i>	=	AND <i>credit_rating</i> = <i>fair</i>	THEN <i>buys_computer</i>	= <i>no</i>

A disjunction (logical OR) is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are **mutually exclusive** and **exhaustive**. Mutually exclusive means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple. (We have one rule per leaf, and any tuple can map to only one leaf.) Exhaustive means there is one rule for each possible attribute–value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter—they are unordered.

Rule Quality Measures

Learn_One_Rule needs a measure of rule quality. Every time it considers an attribute test, it must check to see if appending such a test to the current rule’s condition will result in an improved rule. Accuracy may seem like an obvious choice at first, but consider Example

Example

Choosing between two rules based on accuracy

Consider the two rules as illustrated in [Figure 8.12](#). Both are for the class *loan_decision* = *accept*. We use “*a*” to represent the tuples of class “accept” and “*r*” for the tuples of class “reject.” Rule *r* correctly classifies 38 of the 40 tuples it covers. Rule *s* covers only two tuples, which it correctly classifies. Their respective accuracies are 95% and 100%. Thus, *s* has greater accuracy than *r*, but it is not the better rule because of its small coverage.

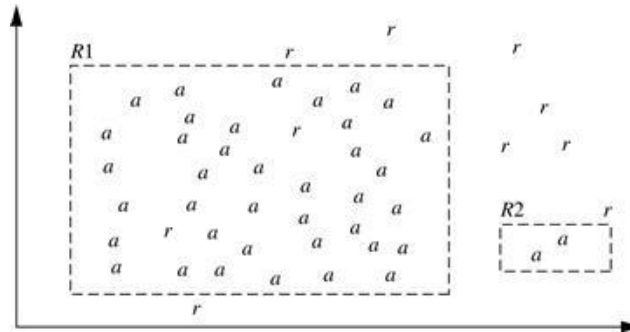


FIGURE Rules for the class `loan_decision = accept`, showing accept (a) and reject (r) tuples.

➤ Bayesian Classification

In numerous applications, the connection between the attribute set and the class variable is non-deterministic. In other words, we can say the class label of a test record can't be assumed with certainty even though its attribute set is the same as some of the training examples. These circumstances may emerge due to the noisy data or the presence of certain confusing factors that influence classification, but it is not included in the analysis. For example, consider the task of predicting the occurrence of whether an individual is at risk for liver illness based on individuals eating habits and working efficiency. Although most people who eat healthily and exercise consistently having less probability of occurrence of liver disease, they may still do so due to other factors. For example, due to consumption of the high-calorie street foods and alcohol abuse. Determining whether an individual's eating routine is healthy or the workout efficiency is sufficient is also subject to analysis, which in turn may introduce vulnerabilities into the learning issue.

Bayesian classification uses Bayes theorem to predict the occurrence of any event. Bayesian classifiers are the statistical classifiers with the Bayesian probability understandings. The theory expresses how a level of belief, expressed as a probability. Bayes theorem came into existence after Thomas Bayes, who first utilized conditional probability to provide an algorithm that uses evidence to calculate limits on an unknown parameter.

Bayes's theorem is expressed mathematically by the following equation that is given below.

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

Where X and Y are the events and $P(Y) \neq 0$

$P(X/Y)$ is a **conditional probability** that describes the occurrence of event X is given that Y is true.

$P(Y/X)$ is a **conditional probability** that describes the occurrence of event Y is given that X is true.

$P(X)$ and $P(Y)$ are the probabilities of observing X and Y independently of each other. This is known as the **marginal probability**.

Bayesian interpretation:

In the Bayesian interpretation, probability determines a "**degree of belief**." Bayes theorem connects the degree of belief in a hypothesis before and after accounting for evidence. For example, Lets us consider an example of the coin. If we toss a coin, then we get either heads or tails, and the percent of occurrence of either heads and tails is 50%. If the coin is flipped numbers of times, and the outcomes are observed, the degree of belief may rise, fall, or remain the same depending on the outcomes.

For proposition X and evidence Y ,

- $P(X)$, the prior, is the primary degree of belief in X
- $P(X/Y)$, the posterior is the degree of belief having accounted for Y .
- The quotient $\frac{P(Y/X)}{P(Y)}$ represents the supports Y provides for X .

Bayes theorem can be derived from the conditional probability:

$$P(X/Y) = \frac{P(X \cap Y)}{P(Y)}, \text{ if } P(Y) \neq 0$$

$$P(Y/X) = \frac{P(Y \cap X)}{P(X)}, \text{ if } P(X) \neq 0$$

Where $P(X \cap Y)$ is the **joint probability** of both X and Y being true, because

$$P(Y \cap X) = P(X \cap Y)$$

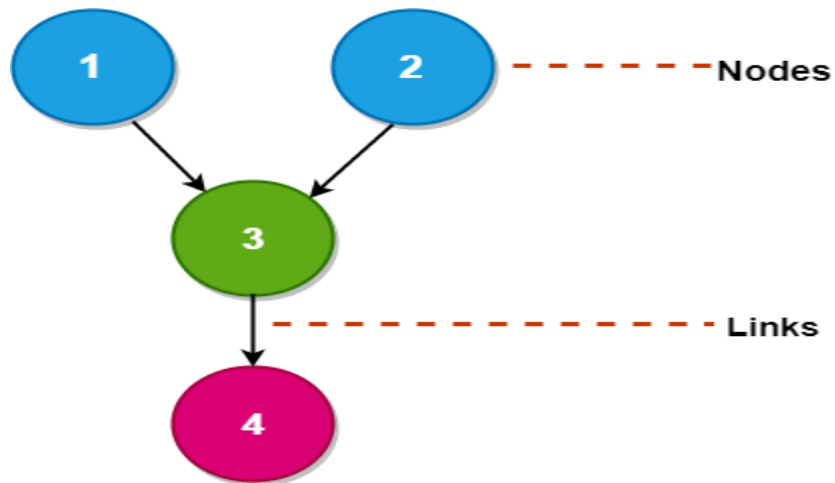
$$\text{or, } P(X \cap Y) = P(X/Y)P(Y) = P(Y/X)P(X)$$

$$\text{or, } P(X/Y) = \frac{P(Y/X)P(X)}{P(Y)}, \text{ if } P(Y) \neq 0$$

Bayesian network:

A Bayesian Network falls under the classification of Probabilistic Graphical Modelling (PGM) procedure that is utilized to compute uncertainties by utilizing the probability concept. Generally known as **Belief Networks**, **Bayesian Networks** are used to show uncertainties using **Directed Acyclic Graphs** (DAG)

A **Directed Acyclic Graph** is used to show a Bayesian Network, and like some other statistical graph, a DAG consists of a set of nodes and links, where the links signify the connection between the nodes.



The nodes here represent random variables, and the edges define the relationship between these variables.

A DAG models the uncertainty of an event taking place based on the Conditional Probability Distribution (CPD) of each random variable. A **Conditional Probability Table (CPT)** is used to represent the CPD of each variable in a network.

➤ Nearest Neighbour 'classifiers

- In k -NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.
- In k -NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k -NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k -NN classification) or the object property value (for k -NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k -NN algorithm is that it is sensitive to the local structure of the data.

What's KNN?

KNN (K — Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based "how similar" is a data (a vector) from other .

How it's working?

The KNN is pretty simple, imagine that you have a data about colored balls:

- Purple balls;
- Yellow balls;
- And a ball that you don't know if it's purple or yellow, but you has all the data about this color(except the color label).

So, how are you going to know the ball's color? imagine you like a machine that you only have the ball's characteristics(data), but doesn't the final label. How do you will to know the ball's color(final label/your class)?

	A	B	C	D	E	F	G	H	I
1	1	1	1	1	1	1	1	1	1 R
2	2	2	2	2	2	2	2	2	2 A
3	2	2	2	2	2	2	2	2	2 A
4	1	1	1	1	1	1	1	1	1 R
5	1	1	1	1	1	1	1	1	1 R

Obs: Let's suppose that data with number 1(and label R) are referring to the purple balls and the data with number 2 (and label A) are referring to the yellow balls, this's just to make the explanation easier, in future articles we will work with real data.

Each line refers to a ball and each column refers to a ball's characteristic, in the last column we have the class (color) of each of the balls:

- **R** -> purple;
- **A** -> yellow

We have 5 balls there (5 lines), each one with yours classification, you can try to discover the new ball's color (in the case the class) of N ways, one of these N ways is to comparing this new ball's characteristics with all the others, and see what it looks like most, if the data(characteristics) of this new ball (that you doesn't know the correct class) is similar to the data of the yellow balls, then the color of the new ball is yellow, if the data in the new ball is more similar to the data of the purple then yellow, then the color of the new ball is purple, it looks so simple, and that is almost what the knn does, but in a most sophisticated way.

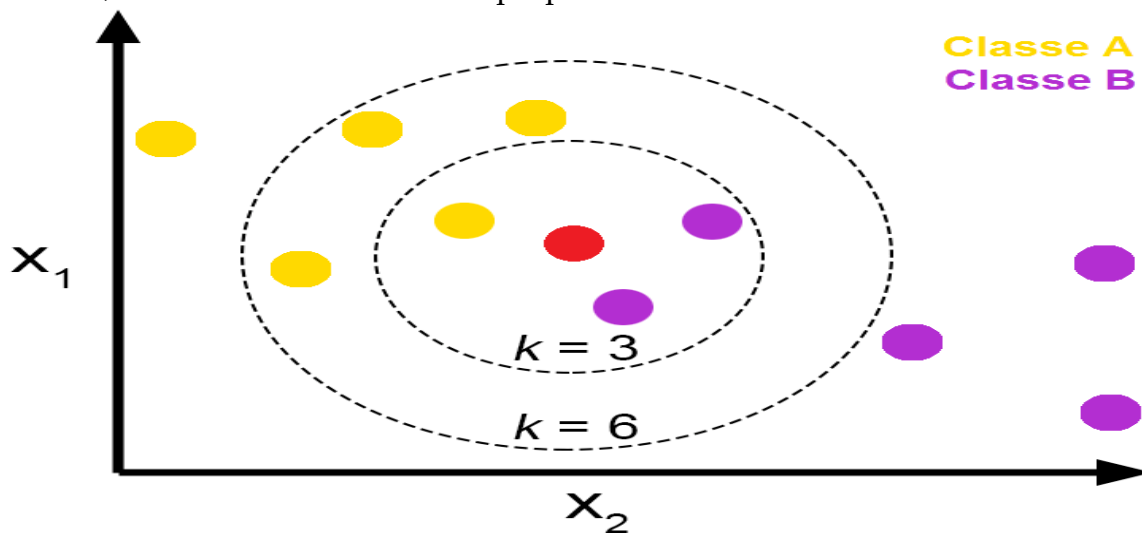
In the case of KNN, indeed it doesn't "compare" the new (unclassified) data with all other, actually he performs a mathematical calculation to measure the distance between the data to makes the classification, (that's almost the same thing). This calculation that I am talking about, can be anyone other calculation that measures the distance between two points, for example: Euclidian, Manhattan, Minkowski, Weighted ...

The KNN's steps are:

- 1 — Receive an unclassified data;
- 2 — Measure the distance (Euclidian, Manhattan, Minkowski or Weighted) from the new data to all others data that is already classified;
- 3 — Gets the K(K is a parameter that you define) smaller distances;
- 4 — Check the list of classes had the shortest distance and count the amount of each class that appears;
- 5 — Takes as correct class the class that appeared the most times;

6 —Classifies the new data with the class that you took in step 5;

Below we have an image with all the process that we talk about in this article, you have an unclassified data (in red) and all your other data already classified (yellow and purple), each one with your class (A or B). So you calculate the distances of your new data with all the others to know which ones have the smallest distance, so you get 3 (or 6) of the closest data and check which class that appears most , in the case of the image below, the closest data to the new data are those that are inside the first circle (inside circle), and inside this circle there are 3 other data (already classified with yellow), we will check which is the predominant class in there, look, it's purple, because there are 2 purple balls and only 1 yellow, so this new data that before was not classified, now it will be classified as purple.



Calculating distance:

To calculate the distance between two points (your new sample and all the data you have in your dataset) is very simple, as said before, there are several ways to get this value, in this article we will use the Euclidean distance.

The Euclidean distance's formula is like the image below:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$
$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.$$

Wait, don't worry, it isn't so much complicated as it sounds, after reading this article and continuing to find it complex, when we get into the code you will see just how simple it is.

Using this formula that you will check the distance between 1 point (your unclassified sample) and 1 other point in your dataset (1 other already classified data), one by one in all your dataset, the smaller the result of this calculation is the most similar between these two data. And yes, you will run this calculation so much times, until you have runned it in all the data, when you do

this, you will have an array (let's use an array as an example) with the result) of your unclassified data to another data already classified.

Ok, let's do it! It's simple but read it slowly, basically you'll:

- Get each characteristic from your dataset;
- Subtract each one, example, (line 1, column 5) — (line1,column5) = X ...(line 1, column 13) — (line1,column13) = Z;
- After get the subtract of all columns, you will get all the results and sum it $X+Y+Z...$;
- So you will get the sum's square root ;

Let's use the example from the previous worksheet but now with 1 unclassified data, that's the information we want to discover.

We have 5 data (lines) in this example, each sample (data/line) have your attributes (characteristics), let's **IMAGINE** that all these are images, each line would be an image and each column would be a image's pixel.

So let's start by that confuse explain above,.

	A	B	C	D	E	F	G	H	I
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2 A
3	2	2	2	2	2	2	2	2	2 A
4	1	1	1	1	1	1	1	1	1 R
5	1	1	1	1	1	1	1	1	1 R

Let's take the first line, which is the data that we want to classify and let's measure the Euclidean distance to line 2

1 — Subtraction

Let's subtract each attribute (column) from row 1 with the attributes from row 2, example:

$$(1-2) = -1$$

2 — Exponentiation:

After you had subtract column 1 from row 1, with column 1 from row 2, we will get the squared root, so the result numbers are always positive, example:

$$(1-2)^2 = (-1)^2(-1)^2 = 1$$

3 — Sum

After you have done step 2, for all the row 1's columns and row 2's columns, we will sum all these results, let's make an example using the spreadsheet's columns' image and we will have the following result:

	A	B	C	D	E	F	G	H	I
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2 A
3	2	2	2	2	2	2	2	2	2 A
4	1	1	1	1	1	1	1	1	1 R
5	1	1	1	1	1	1	1	1	1 R

$$(1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 = 8$$

Note that we have 8 attribute columns, both in row 1 and row 2, and we performed step 2 for each dataset's attribute, so our final result was 8, but why 8? Because each time we run step 2, the result gave 1, by a "coincidence" we have the same data in all columns and the result of $(1-2)$

² is equal to 1, I used those values to facilitate the math here, but no, this attribute doesn't need always be the same number, later we will see this better in implementing a code with this algorithm.

4 — Square root:

After performed step 3, we will get our subtractions's sum's square root . In step 3, the result was 8, so let's take the square root of number 8:

$$\sqrt{8} = 2,83 \text{ out } 8^{(1/2)} = 2,83$$

➤ Estimating Predictive accuracy of Classification methods

➤ Improving accuracy of classification methods

We will learn some tricks for increasing classification accuracy. We focus on ensemble methods. An ensemble for classification is a composite model, made up of a combination of classifiers. Ensembles tend to be more accurate than their component classifiers. The data are class-imbalanced, where the main class of interest is represented by only a few tuples. This is known as the *class* imbalance problem. We also study techniques for improving the classification accuracy of class-imbalanced data.

Introducing Ensemble Methods

Bagging, boosting, and random forests are examples of **ensemble methods**. An ensemble combines a series of k learned models $M_1, M_2, M_3, \dots, M_k$ (or base classifiers), with the aim of creating an improved composite classification model M^* . A given data set, D , is used to create k training sets, $D_1, D_2, \dots, D_k$ where D_i is used to generate classifier M_i . Given a new data tuple to classify, the base classifiers each vote by returning a class prediction. The ensemble returns a class prediction based on the votes of the base classifiers.

Increasing classifier accuracy: Ensemble methods generate a set of classification models, $M_1, M_2, \dots, M_k$. Given a new data tuple to classify, each classifier “votes” for the class label of that tuple. The ensemble combines the votes to return a class prediction.

2 Bagging

We now take an intuitive look at how bagging works as a method of increasing accuracy. Suppose that you are a patient and would like to have a diagnosis made based on your symptoms. Instead of asking one doctor, you may choose to ask several. If a certain diagnosis occurs more than any other, you may choose this as the final or best diagnosis. That is, the final diagnosis is made based on a majority vote, where each doctor gets an equal vote. Now replace each doctor by a classifier, and you have the basic idea behind bagging. Intuitively, a majority vote made by a large group of doctors may be more reliable than a majority vote made by a small group.

Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D , a set of d training tuples;
- k , the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, M^* .

Method:

- (1) **for** $i = 1$ to k **do** // create k models:
- (2) create bootstrap sample, D_i , by sampling D with replacement;
- (3) use D_i and the learning scheme to derive a model, M_i ;
- (4) **endfor**

To use the ensemble to classify a tuple, X :

let each of the k models classify X and return the majority vote;

The bagged classifier often has significantly greater accuracy than a single classifier derived from D , the original training data. It will not be considerably worse and is more robust to the effects of noisy data and overfitting. The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.

Boosting and AdaBoost

We now look at the ensemble method of boosting. As in the previous section, suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses. This is the essence behind boosting.

In **boosting**, weights are also assigned to each training tuple. A series of k classifiers is iteratively learned. After a classifier, M_i , is learned, the weights are updated to allow the subsequent classifier, M_{i+1} , to “pay more attention” to the training tuples that were misclassified by M_i . The final boosted classifier, M^* , combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.

AdaBoost (short for Adaptive Boosting) is a popular boosting algorithm. Suppose we want to boost the accuracy of a learning method. We are given D , a data set of d class-labeled tuples, $(X_1, y_1), (X_2, y_2), \dots, (X_d, y_d)$, where y_i is the class label of tuple X_i . Initially, AdaBoost assigns each training tuple an equal weight of $1/d$. Generating k classifiers for the ensemble requires k rounds through the rest of the algorithm. We can sample to form any sized training set, not necessarily of size d . Sampling with replacement is used—the same tuple may be selected more than once. Each tuple's chance of being selected is based on its weight. A classifier model, M_i , is derived from the training tuples of D_i . Its error is then calculated using D_i as a test set. The weights of the training tuples are then adjusted according to how they were classified.

Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D , a set of d class-labeled training tuples;
- k , the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

- (1) initialize the weight of each tuple in D to $1/d$;
- (2) **for** $i = 1$ to k **do** // for each round:
 - (3) sample D with replacement according to the tuple weights to obtain D_i ;
 - (4) use training set D_i to derive a model, M_i ;
 - (5) compute $error(M_i)$, the error rate of M_i (Eq. 8.34)
 - (6) **if** $error(M_i) > 0.5$ **then**
 - (7) go back to step 3 and try again;
 - (8) **endif**
 - (9) **for** each tuple in D_i that was correctly classified **do**
 - (10) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
 - (11) normalize the weight of each tuple;
- (12) **endfor**

To use the ensemble to classify tuple, X :

- (1) initialize weight of each class to 0;
- (2) **for** $i = 1$ to k **do** // for each classifier:
 - (3) $w_i = \log \frac{1 - error(M_i)}{error(M_i)}$; // weight of the classifier's vote
 - (4) $c = M_i(X)$; // get class prediction for X from M_i
 - (5) add w_i to weight for class c
- (6) **endfor**
- (7) return the class with the largest weight;

4 Random Forests

We now present another ensemble method called **random forests**. Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of classifiers is a “forest.” The individual decision trees are generated using a random selection of attributes at each node to determine the split. More formally, each tree depends on the values of a random

vector sampled independently and with the same distribution for all trees in the forest. During classification, each tree votes and the most popular class is returned.

Random forests can be built using bagging in tandem with random attribute selection. A training set, D , of D tuples is given. The general procedure to generate k decision trees for the ensemble is as follows. For each iteration $i(i=1,2\dots k)$, a training set, D_i , of D tuples is sampled with replacement from D . To construct a decision tree classifier, M_i , randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees. The trees are grown to maximum size and are not pruned. Random forests formed this way, with random input selection, are called Forest-RI.

Another form of random forest, called Forest-RC, uses random linear combinations of the input attributes. Instead of randomly selecting a subset of the attributes, it creates new attributes (or features) that are a linear combination of the existing attributes. That is, an attribute is generated by specifying L , the number of original attributes to be combined. At a given node, L attributes are randomly selected and added together with coefficients that are uniform random numbers on $(-1,1)$. F linear combinations are generated, and a search is made over these for the best split.

Improving Classification Accuracy of Class-Imbalanced Data

In this section, we revisit the class imbalance problem. In particular, we study approaches to improving the classification accuracy of class-imbalanced data.

Given two-class data, the data are class-imbalanced if the main class of interest (the positive class) is represented by only a few tuples, while the majority of tuples represent the negative class. For multiclass-imbalanced data, the data distribution of each class differs substantially where, again, the main class or classes of interest are rare. The class imbalance problem is closely related to cost-sensitive learning, wherein the costs of errors, per class, are not equal. In medical diagnosis, for example, it is much more costly to falsely diagnose a cancerous patient as healthy (a false negative) than to misdiagnose a healthy patient as having cancer (a false positive). A false negative error could lead to the loss of life and therefore is much more expensive than a false positive error. Other applications involving class-imbalanced data include fraud detection, the detection of oil spills from satellite radar images, and fault monitoring.

we look at general approaches for improving the classification accuracy of class-imbalanced data. These approaches include (1) oversampling, (2) undersampling, (3) threshold moving, and (4) ensemble techniques. The first three do not involve any changes to the construction of the classification model. That is, oversampling and undersampling change the distribution of tuples in the training set; threshold moving affects how the model makes decisions when classifying new data. Ensemble methods follow the techniques described in . For ease of explanation, we describe these general approaches with respect to the two-class imbalance data problem, where the higher-cost classes are rarer than the lower-cost classes.

Both oversampling and undersampling change the training data distribution so that the rare (positive) class is well represented. **Oversampling** works by resampling the positive

tuples so that the resulting training set contains an equal number of positive and negative tuples. **Undersampling** works by decreasing the number of negative tuples. It randomly eliminates tuples from the majority (negative) class until there are an equal number of positive and negative tuples.

Evaluation criteria for classification

Various evaluation metrics for the predictive accuracy of a classifier. Holdout and random subsampling cross-validation, and bootstrap methods are common techniques for assessing accuracy, based on randomly sampled partitions of the given data. What if we have more than one classifier and want to choose the “best” one? This is referred to as **model selection** (i.e., choosing one classifier over another).

1 Metrics for Evaluating Classifier Performance

This section presents measures for assessing how good or how “accurate” your classifier is at predicting the class label of tuples. We will consider the case of where the class tuples are more or less evenly distributed, as well as the case where classes are unbalanced (e.g., where an important class of interest is rare such as in medical tests). The classifier evaluation measures presented in this section are summarized in . They include accuracy (also known as recognition rate), sensitivity (or recall), specificity, precision, F_1 , and F_β . Note that although accuracy is a specific measure, the word “accuracy” is also used as a general term to refer to a classifier’s predictive abilities.

Measure	Formula
accuracy, recognition rate	$\frac{TP + TN}{P + N}$
error rate, misclassification rate	$\frac{FP + FN}{P + N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP + FP}$
F , F_1 , F -score, harmonic mean of precision and recall	$\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
F_β , where β is a non-negative real number	$\frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}$

There are four additional terms we need to know that are the “building blocks” used in computing many evaluation measures. Understanding them will make it easy to grasp the meaning of the various measures.

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

- **False negatives (FN):** These are the positive tuples that were mislabeled as negative (e.g., tuples of class `buys_computer = yes` for which the classifier predicted `buys_computer = no`). Let *FN* be the number of false negatives.

These terms are summarized in the **confusion matrix**

		Predicted class		
		yes	no	Total
Actual class	yes	TP	FN	P
	no	FP	TN	N
Total		P'	N'	P + N

The confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes. *TP* and *TN* tell us when the classifier is getting things right, while *FP* and *FN* tell us when the classifier is getting things wrong (i.e., mislabeling). Given *m* classes (where *m* ≥ 2), a **confusion matrix** is a table of at least size *m* by *m*.

The *precision* and *recall* measures are also widely used in classification. **Precision** can be thought of as a measure of *exactness* (i.e., what percentage of tuples labeled as positive are actually such), whereas **recall** is a measure of *completeness* (what percentage of positive tuples are labeled as such). If recall seems familiar, that's because it is the same as sensitivity (or the *true positive rate*). These measures can be computed as

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

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

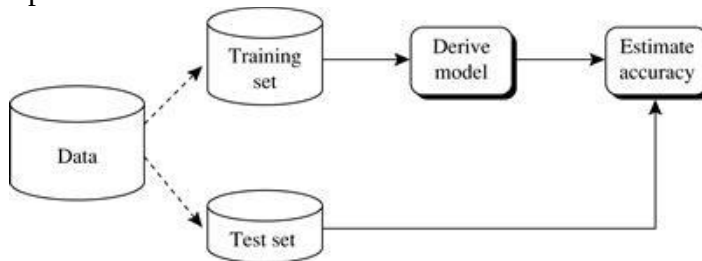
In addition to accuracy-based measures, classifiers can also be compared with respect to the following additional aspects:

- **Speed:** This refers to the computational costs involved in generating and using the given classifier.
- **Robustness:** This is the ability of the classifier to make correct predictions given noisy data or data with missing values. Robustness is typically assessed with a series of synthetic data sets representing increasing degrees of noise and missing values.
- **Scalability:** This refers to the ability to construct the classifier efficiently given large amounts of data. Scalability is typically assessed with a series of data sets of increasing size.
- **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess. Decision trees and classification rules can be easy to interpret, yet their interpretability may diminish the more they become complex. We discuss some work in this area, such as the extraction of classification rules from a “black box” neural network classifier called backpropagation.

2 Holdout Method and Random Subsampling

The **holdout** method is what we have alluded to so far in our discussions about accuracy. In this method, the given data are randomly partitioned into two independent sets, a training set and a test set. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the

model. The model's accuracy is then estimated with the test set. The estimate is pessimistic because only a portion of the initial data is used to derive the model.



Estimating accuracy with the holdout method.

Random subsampling is a variation of the holdout method in which the holdout method is repeated k times. The overall accuracy estimate is taken as the average of the accuracies obtained from each iteration.

3 Cross-Validation

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

Leave-one-out is a special case of k -fold cross-validation where k is set to the number of initial tuples. That is, only one sample is “left out” at a time for the test set. In **stratified cross-validation**, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data.

In general, stratified 10-fold cross-validation is recommended for estimating accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

4 Bootstrap

Unlike the accuracy estimation methods just mentioned, the **bootstrap method** samples the given training tuples uniformly with replacement. That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set. For instance, imagine a machine that randomly selects tuples for our training set. In sampling with replacement, the machine is allowed to select the same tuple more than once.

There are several bootstrap methods. A commonly used one is the **.632 bootstrap**, which works as follows. Suppose we are given a data set of D tuples. The data set is sampled D times, with replacement, resulting in a bootstrap sample or training set of D samples. It is very likely that some of the original data tuples will occur more than once in this sample. The data tuples that did not make it into the training set end up forming the test set. Suppose we were to try

this out several times. As it turns out, on average, 63.2% of the original data tuples will end up in the bootstrap sample, and the remaining 36.8% will form the test set (hence, the name, .632 bootstrap).

5 Comparing Classifiers Based on Cost–Benefit and ROC Curves

The true positives, true negatives, false positives, and false negatives are also useful in assessing the **costs and benefits** (or risks and gains) associated with a classification model. The cost associated with a false negative (such as incorrectly predicting that a cancerous patient is not cancerous) is far greater than those of a false positive (incorrectly yet conservatively labeling a noncancerous patient as cancerous). In such cases, we can outweigh one type of error over another by assigning a different cost to each. These costs may consider the danger to the patient, financial costs of resulting therapies, and other hospital costs. Similarly, the benefits associated with a true positive decision may be different than those of a true negative. Up to now, to compute classifier accuracy, we have assumed equal costs and essentially divided the sum of true positives and true negatives by the total number of test tuples.

Alternatively, we can incorporate costs and benefits by instead computing the average cost (or benefit) per decision. Other applications involving cost–benefit analysis include loan application decisions and target marketing mailouts. For example, the cost of loaning to a defaulter greatly exceeds that of the lost business incurred by denying a loan to a non defaulter. Similarly, in an application that tries to identify households that are likely to respond to mailouts of certain promotional material, the cost of mailouts to numerous households that do not respond may outweigh the cost of lost business from not mailing to households that would have responded. Other costs to consider in the overall analysis include the costs to collect the data and to develop the classification tool.

➤ Multiclass Problem

What is multiclass classification?

Multi-class or multiclass, the science is the same. Multiclass classification is a machine learning classification task that consists of more than two classes, or outputs. For example, using a model to identify animal types in images from an encyclopedia is a multiclass classification example because there are many different animal classifications that each image can be classified as. Multiclass classification also requires that a sample only have one class (ie. an elephant is only an elephant; it is not also a lemur).

In multiclass or multinomial classification is the problem of classifying instances into one of three or more classes (classifying instances into one of two classes is called binary classification).

While many classification algorithms (notably multinomial logistic regression) naturally permit the use of more than two classes, some are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

Multiclass classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance.

The existing multi-class classification techniques can be categorized into (i) transformation to binary (ii) extension from binary and (iii) hierarchical classification.

Transformation to binary

This section discusses strategies for reducing the problem of multiclass classification to multiple binary classification problems. It can be categorized into one vs rest and one vs one. The techniques developed based on reducing the multi-class problem into multiple binary problems can also be called problem transformation techniques.

One-vs.-rest

One-vs.-rest (OvR or one-vs.-all, OvA or one-against-all, OAA) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

In pseudocode, the training algorithm for an OvR learner constructed from a binary classification learner L is as follows:

Inputs:

- L , a learner (training algorithm for binary classifiers)
- samples X
- labels y where $y_i \in \{1, \dots, K\}$ is the label for the sample X_i

Output:

- a list of classifiers f_k for $k \in \{1, \dots, K\}$

Procedure:

- For each k in $\{1, \dots, K\}$
 - Construct a new label vector z where $z_i = y_i$ if $y_i = k$ and $z_i = 0$ otherwise
 - Apply L to X, z to obtain f_k
 - Making decisions means applying all classifiers to an unseen sample x and predicting the label k for which the corresponding classifier reports the highest confidence score:

Although this strategy is popular, it is a heuristic that suffers from several problems. Firstly, the scale of the confidence values may differ between the binary classifiers. Second, even if the class distribution is balanced in the training set, the binary classification learners see unbalanced distributions because typically the set of negatives they see is much larger than the set of positives.

One-vs.-one

In the one-vs.-one (OvO) reduction, one trains $K(K-1)/2$ binary classifiers for a K -way multiclass problem; each receives the samples of a pair of classes from the original training set, and must learn to distinguish these two classes. At prediction time, a voting scheme is applied: all $K(K-1)/2$ classifiers are applied to an unseen sample and the class that got the highest number of "+1" predictions gets predicted by the combined classifier.[2]:339

For instance, images of cats may all follow a pattern of pointed ears and whiskers, helping the model to identify future images of cats as compared to other animals without whiskers or pointed ears.

Multiclass classification use cases

For example, a cybersecurity company might want to be able to monitor a user's email inbox and classify incoming emails as either potential phishers or not. To do so, it might train a classification model on the email texts and inbound email addresses of previous phishing scams to teach the model to predict which URLs tend to accompany threatening emails.

As another example, a marketing company might serve an online ad and want to predict whether a given customer will click on it.

Multiclass classification algorithm models are just one of the many examples of the importance of machine learning.

Multi-Class Classification

Multi-class classification refers to those classification tasks that have more than two class labels.

Examples include:

- Face classification.
- Plant species classification.
- Optical character recognition.

Unlike binary classification, multi-class classification does not have the notion of normal and abnormal outcomes. Instead, examples are classified as belonging to one among a range of known classes.

The number of class labels may be very large on some problems. For example, a model may predict a photo as belonging to one among thousands or tens of thousands of faces in a face recognition system.

Problems that involve predicting a sequence of words, such as text translation models, may also be considered a special type of multi-class classification. Each word in the sequence of words to be predicted involves a multi-class classification where the size of the vocabulary defines the number of possible classes that may be predicted and could be tens or hundreds of thousands of words in size.

It is common to model a multi-class classification task with a model that predicts a Multinoulli probability distribution for each example.

The Multinoulli distribution is a discrete probability distribution that covers a case where an event will have a categorical outcome, e.g. K in $\{1, 2, 3, \dots, K\}$. For classification, this means that the model predicts the probability of an example belonging to each class label.

Many algorithms used for binary classification can be used for multi-class classification.

Popular algorithms that can be used for multi-class classification include:

- k-Nearest Neighbors.
- Decision Trees.
- Naive Bayes.
- Random Forest.
- Gradient Boosting.

Algorithms that are designed for binary classification can be adapted for use for multi-class problems.

➤ Clustering Techniques

Clustering is the process of grouping the data into classes or clusters, so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters. Dissimilarities are assessed based on the attribute values describing the objects. Clustering has its roots in many areas, including data mining, statistics, biology, and machine learning.

We examine several clustering techniques, organized into the following categories: partitioning methods, hierarchical methods, density-based methods, grid-based methods, model-based methods, methods for high-dimensional data (such as frequent pattern-based methods), and constraint-based clustering. Clustering can also be used for outlier detection.

Applications of cluster analysis

Cluster analysis has been widely used in many applications, such as business intelligence, image pattern recognition, Web search, biology, and security. In business intelligence, clustering can be used to organize a large number of customers into groups, where customers within a group share strong similar characteristics. This facilitates the development of business strategies for enhanced customer relationship management. Moreover, consider a consultant company with a large number of projects. To improve project management, clustering can be applied to partition projects into categories based on similarity so that project auditing and diagnosis (to improve project delivery and outcomes) can be conducted effectively.

In image recognition, clustering can be used to discover clusters or “sub-classes” in handwritten character recognition systems. Suppose we have a data set of handwritten digits, where each digit is labeled as either 1, 2, or 3, and so on. Note that there can be a large variance in the way in which people write the same digit. Take the number “2”, for example. Some people may write it with a small circle at the left bottom part, while some others may not. We can use clustering to determine subclasses for “2”, each of which represents a variation on the way in which “2” can be written. Using multiple models based on the subclasses can improve the overall recognition accuracy.

Clustering has also found many applications in Web search. For example, a keyword search may often return a very large number of hits (that is, pages relevant to the search) due to the extremely large number of Web pages. Clustering can be used to organize the search results into groups and present the results in a concise and easily accessible way.

As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on the detected clusters and the

selected attributes or features.

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, where outliers (values that are “far away” from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine learning, spatial database technology, information retrieval, Web search, biology, marketing, and many other application areas.

As a branch of statistics, cluster analysis has been extensively studied, with the main focus on distance-based cluster analysis. Cluster analysis tools based on k -means, k -medoids, and several other methods have also been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS. In machine learning, recall that classification is known as supervised learning because the class label information is given, that is, the learning algorithm is supervised in that it is told the class membership of each training tuple. Clustering is known as unsupervised learning because the class label information is not present. For this reason, clustering is a form of learning by observation, rather than learning by examples. In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in large databases. Active themes of research focus on the scalability of clustering methods, the effectiveness of methods for clustering complex shapes (e.g., non-convex) and types of data (e.g., text, graphs, and images), high-dimensional clustering techniques (e.g., clustering objects with thousands of features), and methods for clustering mixed numerical and nominal data in large databases.

➤ Requirements/Features for Cluster Analysis

Clustering is a challenging field of research. In this section, you will learn about the requirements for clustering as a data mining tool, as well as aspects that can be used for comparing clustering methods.

The following are 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 or even billions of objects, particularly in the Web search scenarios. Clustering on a sample of a given large data set may lead to biased results. Therefore, highly scalable clustering algorithms are needed.

Ability to deal with different types of attributes: Many algorithms are designed to cluster numeric (interval-based) data. However, applications may require clustering other types of data, such as binary, nominal (categorical), and ordinal data, or mixtures of these data types. Recently, more and more applications need clustering

techniques for complex data types such as graphs, sequences, images, and documents.

Discovery of clusters with arbitrary shape: Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures (Chapter 2). Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. Consider sensors, for example, which are often deployed for environment surveillance. Cluster analysis on sensor readings can detect interesting phenomena. We may want to use clustering to find the frontier of a running forest fire, which is often not spherical. It is important to develop algorithms that can detect clusters of arbitrary shape.

Requirements for domain knowledge to determine input parameters: Many clustering algorithms require users to provide domain knowledge in the form of input parameters, such as the desired number of clusters. Consequently, the clustering results may be sensitive to such parameters. Parameters are often hard to determine, especially for data sets of high dimensionality and where users have yet to grasp a deep understanding of their data. Requiring the specification of domain knowledge not only burdens users, but also makes the quality of clustering difficult to control.

Ability to deal with noisy data: Most realworld data sets contain outliers and/or missing, unknown, or erroneous data. Sensor readings, for example, are often noisy some readings may be inaccurate due to the sensing mechanisms, and some readings may be erroneous due to interferences from surrounding transient objects. Clustering algorithms can be sensitive to such noise and may produce clusters of poor quality. Clustering methods that are robust to noise are highly desirable.

Incremental clustering and insensitivity to input order: In many applications, incremental updates (representing newer data) may arrive at any time. Some clustering algorithms cannot incorporate incremental updates into existing clustering structures and, instead, have to recompute a new clustering from scratch. Clustering algorithms may also be sensitive to the order of input data. That is, given a set of data objects, such algorithms may return dramatically different clusterings depending on the order in which the objects are presented. Incremental clustering algorithms and algorithms that are insensitive to the order of input are needed.

Capability of clustering data of high dimensionality: A data set can contain numerous dimensions or attributes. When clustering documents, for example, each keyword can be regarded as a dimension, and there are often thousands of keywords. Most clustering algorithms are good at handling low-dimensional data, such as data sets involving only two or three dimensions. Finding clusters of data objects in a high-dimensional space is challenging, especially considering that such data can be very 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 automatic 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 types 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 want clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied in with specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and clustering methods.

➤ Types of Data and Computing Distance

The data set to be clustered contains n objects, which may represent persons, houses, documents, countries, and so on. Main memory-based clustering algorithms typically operate on either of the following two data structures.

Data matrix (two modes) object by variable Structure: this represents n objects, such as persons, with p variables (also called measurements or attributes), such as age, height, weight, gender, and so on. The structure is in the form of a relational table, or n -by- p matrix (n objects $\times p$ variables):

$$\begin{bmatrix} x_{11} & \dots & x_{1f} & \dots & x_{1p} \\ \dots & \dots & \dots & \dots & \dots \\ x_{i1} & \dots & x_{if} & \dots & x_{ip} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & \dots & x_{nf} & \dots & x_{np} \end{bmatrix}$$

Dissimilarity matrix (one mode) object –by-object structure: This stores a collection of proximities that are available for all pairs of n objects. It is often represented by an n -by- n table:

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ d(n,1) & d(n,2) & \dots & \dots & 0 \end{bmatrix}$$

Where $d(i,j)$ is the measured difference or dissimilarity between objects i and j . In general, $d(i,j)$ is a nonnegative number that is close to 0 when objects i and j are highly similar or “near” each other, and becomes larger the more they differ.

The rows and columns of the data matrix represent different entities, while those of the dissimilarity matrix represent the same entity. Thus, the data matrix is often called a two-

mode matrix, whereas the dissimilarity matrix is called a one-mode matrix. Many clustering algorithms operate on a dissimilarity matrix. If the data are represented in the form of a data matrix; it can first be transformed into a dissimilarity matrix before applying such clustering algorithms.

1. Interval-Scaled Variables

It describes distance measures that are commonly used for computing the dissimilarity of objects described by such variables. Interval-scaled variables are continuous measurements of a roughly linear scale. i.e. include examples weight and height, latitude and longitude coordinates (e.g., when clustering houses), and weather temperature.

The measurement unit used can affect the clustering analysis. For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure. In general, expressing a variable in smaller units will lead to a larger range for that variable, and thus a larger effect on the resulting clustering structure. To help avoid dependence on the choice of measurement units, the data should be standardized. However, in some applications, users may intentionally want to give more weight to a certain set of variables than to others. For example, when clustering basketball player candidates, we may prefer to give more weight to the variable height.

2. Binary Variables

A binary variable has only two states: 0 or 1, where 0 means that the variable is absent, and 1 means that it is present. Given the variable smoker describing a patient, for instance, 1 indicates that the patient smokes, while 0 indicates that the patient does not. Treating binary variables as if they are interval-scaled can lead to misleading clustering results. Therefore, methods specific to binary data are necessary for computing dissimilarities.

“So, how can we compute the dissimilarity between two binary variables?” One approach involves computing a dissimilarity matrix from the given binary data. If all binary variables are thought of as having the same weight, we have the 2-by-2 contingency table of Table 7.1, where q is the number of variables that equal 1 for both objects i and j , r is the number of variables that equal 1 for object i but that are 0 for object j , s is the number of variables that equal 0 for object i but equal 1 for object j , and t is the number of variables that equal 0 for both objects i and j . The total number of variables is p , where $p = q + r + s + t$.

“What is the difference between symmetric and asymmetric binary variables?” A binary variable is symmetric if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be the attribute gender having the states male and female. Dissimilarity that is based on symmetric binary variables is called symmetric binary

dissimilarity. Its dissimilarity (or distance) measure, defined in Equation (7.9), can be used to assess the dissimilarity between objects i and j .

$$d(i, j) = \frac{r+s}{q+r+s+t} \text{ . (7.9)}$$

		Object j		
Object i		1	0	sum
	1	q	r	$q + r$
	0	s	t	$s + t$
	Sum	$q + s$	$r + t$	m

A binary variable is asymmetric if the outcomes of the states are not equally important, such as the positive and negative outcomes of a disease test. By convention, we shall code the most important outcome, which is usually the rarest one, by 1 (e.g., HIV positive) and the other by 0 (e.g., HIV negative). Given two asymmetric binary variables, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary variables are often considered “monary” (as if having one state). The dissimilarity based on such variables is called **asymmetric binary dissimilarity**, where the number of negative matches, t , is considered unimportant and thus is ignored in the computation, as shown in Equation (7.10). $d(i, j) = \frac{r+s}{q+r+s}$. (7.10) complementarily, we can measure the distance between two binary variables based on the notion of similarity instead of dissimilarity. For example, the asymmetric binary similarity between the objects i and j , or $\text{sim}(i, j)$, can be computed as, $\text{sim}(i, j) = \frac{q}{q+r+s} = 1-d(i, j)$. (7.11) the coefficient $\text{sim}(i, j)$ is called the **Jaccard coefficient**, which is popularly referenced in the literature. When both symmetric and asymmetric binary variables occur in the same data set.

3. Categorical, Ordinal, and Ratio-Scaled Variables

How can we compute the dissimilarity between objects described by categorical, ordinal, and ratio-scaled variables?

Categorical Variables

A generalization of the binary variable in that it can take more than 2 states. For example, map-color is a categorical variable that may have, say five states: red, yellow, blue, green ,pink and blue.

Let the number of states of a categorical variable be M .The states can be denoted by letters,symbols,or a set of integers,such as $1,2,...M$.Notice that such integers are used just for data handling

$$d(i,j)=\frac{p-m}{p}$$

and do not represent any specific ordering.

The dissimilarity computed between objects i and j can be computed based on the ratio of mismatches

Where m is the number of matches and p is the total number of variables. weights can be assigned to increase the effect of m or to assign a greater weight to the matches in variables having a larger number of states

Ordinal Variables

An ordinal variable is similar to a categorical variable. The difference between the two is that there is a clear ordering of the categories. For example, suppose you have a variable, economic status, with three categories (low, medium and high). In addition to being able to classify people into these three categories, you can order the categories as low, medium and high. Now consider a variable like educational experience (with values such as elementary school graduate, high school graduate, some college and college graduate). These also can be ordered as elementary school, high school, some college, and college graduate.

Even though we can order these from lowest to highest, the spacing between the values may not be the same across the levels of the variables. Say we assign scores 1, 2, 3 and 4 to these four levels of educational experience and we compare the difference in education between categories one and two with the difference in educational experience between categories two and three, or the difference between categories three and four.

The difference between categories one and two (elementary and high school) is probably much bigger than the difference between categories two and three (high school and some college). In this example, we can order the people in level of educational experience but the size of the difference between categories is inconsistent (because the spacing between categories one and two is bigger than categories two and three). If these categories were equally spaced, then the variable would be an interval variable.

How ordinal variables are handles?

The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is a variable from a set of ordinal variables describing n objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the i th object is x_{if} , and f has M_f ordered states, representing the ranking $1, \dots, M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \dots, M_f\}$.
2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto $[0.0, 1.0]$ so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i th objects in the f th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

Ratio-Scaled Variables

A ratio-scaled variable makes a positive measurement on a nonlinear scale, such as an exponential scale, approximately following the formula Ae^{Bt} or Ae^{-Bt} .

Where A and B are positive constants, and t typically represents time. Common examples include the growth of a bacteria population or the decay of a radioactive element.

There are three methods to handle ratio-scaled variables for computing the dissimilarity between objects.

1. Treat ratio-scaled variables like interval-scaled variables. This however, is not usually a good choice since it is likely that scale may be distorted.
2. Apply logarithmic transformation to a ratio-scaled variable f having value x_{if} for object i by using the formula $y_{if} = \log(x_{if})$. The y_{if} values can be treated as interval valued.
3. Treat x_{if} as continuous ordinal data and treat their ranks as interval-valued.

4 Variables of Mixed Types:

Many real databases, objects are described by a mixture of variable types. "So, how can we compute the dissimilarity between objects of mixed variable types?" One approach is to group each kind of variable together, performing a separate cluster analysis for each variable type. This is feasible if these analysis derive compatible results. A more preferable approach is to process all variable types together, performing a single cluster analysis. One such technique combines the different variables into a single dissimilarity matrix, bringing all of the meaningful variables onto a common scale of the interval [0.0,1.0] Suppose that the data set contains p variables of mixed type. The dissimilarity $d(i, j)$ between objects i and j is defined as

$$d(i, j) = \frac{\sum_{f=1}^p \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^p \delta_{ij}^{(f)}}$$

5 Vector Objects

Vector objects: keywords in documents, gene features in micro-arrays, etc.

Broad applications: information retrieval, biologic taxonomy, etc.

Cosine measure

$$s(\vec{X}, \vec{Y}) = \frac{\vec{X}^t \cdot \vec{Y}}{|\vec{X}| |\vec{Y}|},$$

A variant: Tanimoto coefficient

$$s(\vec{X}, \vec{Y}) = \frac{\vec{X}^t \cdot \vec{Y}}{\vec{X}^t \cdot \vec{X} + \vec{Y}^t \cdot \vec{Y} - \vec{X}^t \cdot \vec{Y}},$$

➤ Types of Cluster Analysis Methods

Introduction

There exist a large number of clustering algorithms in the literature. The choice of clustering

algorithm depends both on the type of data available and on the particular purpose and application. If cluster analysis is used as a descriptive or exploratory tool, it is possible to try several algorithms on the same data to see what the data may disclose.

In general, major clustering methods can be classified into the following categories.

Partitioning methods:

Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $K \leq n$. That is, it classifies the data into k groups, which together satisfy the following requirements:

- (1) each group must contain at least one object, and
- (2) each object must belong to exactly one group-Notice that the second requirement can be relaxed in some fuzzy partitioning technique.

Given K , the number of partitions to construct, 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 clusters are "close" or related to each other, whereas objects of different clusters are "far apart" or very different. There are various kinds of other criteria for judging the quality of partitions.

To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of two popular heuristic methods;

1. the k -means algorithm, where each cluster is represented by the mean value of the objects in the cluster, and
2. The k -medoids algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium-sized databases. To find clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended. Partitioning-based clustering methods are studied in depth later.

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 close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The divisive approach, also called the top-down approach, starts with all the objects in the same cluster, 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 worrying about a combinatorial number of different choices. However, a major problem of such techniques is that they cannot correct erroneous decisions. There are two approaches to improving the quality of hierarchical partitioning, such as in CURE and Chameleon, or (2) integrate hierarchical agglomeration and iterative relocation by first using a hierarchical agglomerative algorithm and then refining the result using iterative relocation by first using a hierarchical agglomerative algorithm and then refining the result using iterative relocation, as in BIRCH.

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. Their general idea is to continue growing the given cluster as long as the density (number of objects or data points)in the "neighborhood" exceeds some threshold; 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 is a typical density-based method that grows clusters according to a density threshold, OPTICS is a density-based method that computes an augmented clustering ordering for automatic and interactive cluster analysis.

Grid -based method: 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. CLIQUE and wave-cluster are two clustering algorithms that are 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 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 .Model-based clustering methods are studied below.

Some clustering algorithms integrate the ideas of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one

clustering method category. Furthermore, some applications may have clustering criteria that require the integration of serial clustering techniques.

In the following sections, we examine each of the above five clustering methods in detail. We also introduce algorithms that integrate the ideas of several clustering methods. Outlier analysis, which typically involves clustering, is described at the end of this section.

Partitioning Methods

Given a database of objects and k , the number of clusters to form, a partitioning algorithm organizes the objects into k partitions ($k \leq n$), where each partition represents a cluster. The clusters are formed to optimize an objective-partitioning criterion, often called a similarity function, such as distance

so that the objects within a cluster are "similar," whereas the objects of different clusters are "dissimilar" in terms of the database attributes.

Classical Partitioning Methods: k-means and k-medoids

The most well-known and commonly used partitioning methods are k-means, k-medoids, and their variations.

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 intercluster 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 center of gravity.

"How does the k-means algorithm work?" 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 squared-error criterion is used, defined as

$$E = \sum_i \sum_{p \in C_i} \|p - m_i\|^2$$

where E is the sum of square-error for all objects in the database, p is the point in space representing a given object, and m_i is the mean of cluster C_i (both p and m_i are multidimensional). This criterion tries to make the resulting k clusters as compact and as separate as possible.

The algorithm attempts to determine K partitions that minimize the squared-error function. It works when the clusters are compact clouds that are rather well separated from one another. The method is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$. The method often terminates at a local optimum.

The k-means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical

attributes are involved-the necessity for users to specify k , the number of clusters, in advance can be seen as a disadvantage. the k -means method is not suitable for discovering clusters with nonconvex shapes or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

Hierarchical Methods

A hierarchical clustering method works by grouping data objects into a tree clusters. Hierarchical clustering methods can be further classified into agglomerative and divisive hierarchical clustering, depending on whether the hierarchy decomposition is formed in a bottom-up or top-down fashion. The quality of a pure hierarchical clustering method suffers from its inability to perform adjustment once a merge or split decision has been executed. Recent studies have emphasized the integration of hierarchical agglomeration with iterative relocation methods.

Agglomerative and Divisive Hierarchical Clustering

In general, there are two types of hierarchical clustering methods:

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 distance between the two closest clusters is above a certain threshold distance.

Four widely used measures for distance between clusters are as follows, where $|p-p'|$ is the distance between two objects or points p and p' , m , is the mean for cluster C , and n , is the number of objects of in C_i .

minimum distance :

maximum distance:

mean distance:

Average distance:

➤ Partitioning Methods

The simplest and most fundamental version of cluster analysis is partitioning, which organizes the objects of a set into several exclusive groups or clusters. To keep the problem specification concise, we can assume that the number of clusters is given as background knowledge. This parameter is the starting point for partitioning methods.

Formally, given a data set, D , of n objects, and k , the number of clusters to form, a **partitioning algorithm** organizes the objects into k partitions ($k \leq n$), where each partition

represents a cluster. The clusters are formed to optimize an objective partitioning criterion, such as a dissimilarity function based on distance, so that the objects within a cluster are “similar” to one another and “dissimilar” to objects in other clusters in terms of the data set attributes.

In this section you will learn the most well-known and commonly used partitioning methods— k -means and k -medoids. You will also learn several variations of these classic partitioning methods and how they can be scaled up to handle large data sets.

10.2.1 k -Means: A Centroid-Based Technique

Suppose a data set, D , contains n objects in Euclidean space. Partitioning methods distribute the objects in D into k clusters, $C_1, \dots, C_k$, that is, $C_i \subset D$ and $C_i \cap C_j = \emptyset$ for $(1 \leq i, j \leq k)$. An objective function is used to assess the partitioning quality so that objects within a cluster are similar to one another but dissimilar to objects in other clusters. This is, the objective function aims for high intracluster similarity and low intercluster similarity.

A centroid-based partitioning technique uses the *centroid* of a cluster, C_i , to represent that cluster. Conceptually, the centroid of a cluster is its center point. The centroid can be defined in various ways such as by the mean or medoid of the objects (or points) assigned to the cluster. The difference between an object $\mathbf{p} \in C_i$ and $\mathbf{c}_i$, the representative of the cluster, is measured by $\text{dist}(\mathbf{p}, \mathbf{c}_i)$, where $\text{dist}(x, y)$ is the Euclidean distance between two points $\mathbf{x}$ and $\mathbf{y}$. The quality of cluster C_i can be measured by the **within-cluster variation**, which is the sum of *squared error* between all objects in C_i and the centroid $\mathbf{c}_i$, defined as

where E is the sum of the squared error for all data set; $\mathbf{p}$ is the point in space representing a given object; and $\mathbf{c}_i$ is the centroid of cluster C_i (both $\mathbf{p}$ and $\mathbf{c}_i$ are multidimensional). In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This objective function tries to make the resulting k clusters as compact and as separate as possible.

Optimizing the within-cluster variation is computationally challenging. In the worst case, we would have to enumerate a number of possible partitioning that are exponential to the number of clusters, and check the within-cluster variation values. It has been shown that the problem is NP-hard in general Euclidean space even for two clusters (i.e., $k = 2$). Moreover, the problem is NP-hard for a general number of clusters k even in the 2-D Euclidean space. If the number of clusters k and the dimensionality of the space d are fixed, the problem can be solved in time $O(n^{dk+1} \log n)$, where n is the number of objects. To overcome the prohibitive computational cost for the exact solution, greedy approaches are often used in practice. A prime example is the k -means algorithm, which is simple and commonly used.

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

Input:

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

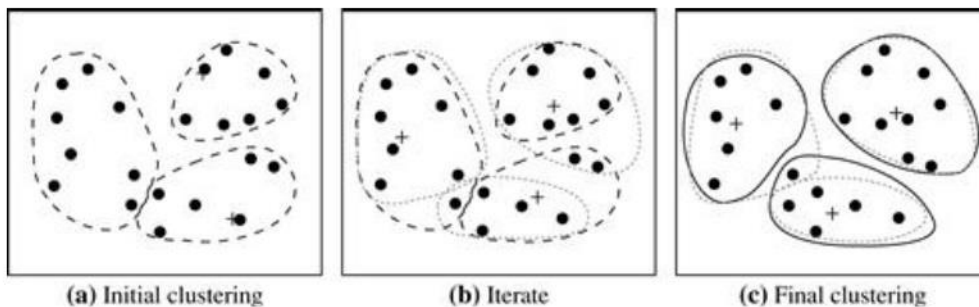
Output: A set of k clusters.

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, that is, calculate the mean value of the objects for each cluster;
- (5) **until** no change;

Clustering by k -means partitioning

Consider a set of objects located in 2-D space, as depicted in . Let $k = 3$, that is, the user would like the objects to be partitioned into three clusters.



K-Medoids: A Representative Object-Based Technique

The k -means algorithm is sensitive to outliers because such objects are far away from the majority of the data, and thus, when assigned to a cluster, they can dramatically distort the mean value of the cluster. This inadvertently affects the assignment of other objects to clusters. This effect is particularly exacerbated due to the use of the squared-error function of Eq. as observed in Example

“How can we modify the k -means algorithm to diminish such sensitivity to outliers?” 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 assigned to the cluster of which the representative object is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object p and its corresponding representative object. That is, an **absolute-error criterion** is used, defined as

$$E = \sum_{i=1}^k \sum_{p \in C_i} \text{dist}(p, o_i),$$

where E is the sum of the absolute error for all objects p in the data set, and o_i is the representative object of C_i . This is the basis for the ***k-medoids method***, which groups n objects into k clusters by minimizing the absolute error.

When $k = 1$, we can find the exact median in $O(n^2)$ time. However, when k is a general positive number, the k -medoid problem is NP-hard.

The **Partitioning Around Medoids (PAM)** algorithm is a popular realization of k -medoids clustering. It tackles the problem in an iterative, greedy way. Like the k -means algorithm, the initial representative objects (called seeds) are chosen arbitrarily. We consider whether replacing a representative object by a non representative object would improve the clustering quality. All the possible replacements are tried out. The iterative process of replacing representative objects by other objects continues until the quality of the resulting clustering cannot be improved by any replacement. This quality is measured by a cost function of the average dissimilarity between an object and the representative object of its cluster.

Specifically, let $o_1, \dots, o_k$ be the current set of representative objects (i.e., medoids). To determine whether a non representative object, denoted by o_{random} , is a good replacement for a current medoid o_j ($1 \leq j \leq k$), we calculate the distance from every object p to the closest object in the set $\{o_1, \dots, o_{j-1}, o_{\text{random}}, o_{j+1}, \dots, o_k\}$, and use the distance to update the cost function. The reassignments of objects to $\{o_1, \dots, o_{j-1}, o_{\text{random}}, o_{j+1}, \dots, o_k\}$ are simple. Suppose object p is currently assigned to a cluster represented by medoid o_j (Figure 10.4a or b). Do we need to reassign p to a different cluster if o_j is being replaced by o_{random} ? Object p needs to be reassigned to either o_{random} or some other cluster represented by o_i ($i \neq j$), whichever is the closest. For example, in Figure 10.4(a), p is closest to o_i and therefore is reassigned to o_i . In Figure 10.4(b), however, p is closest to o_{random} and so is reassigned to o_{random} . What if, instead, p is currently assigned to a cluster represented by some other object o_i , $i \neq j$? Object o remains assigned to the cluster represented by o_i as long as o is still closer to o_i than to o_{random} (Figure 10.4c). Otherwise, o is reassigned to o_{random} .

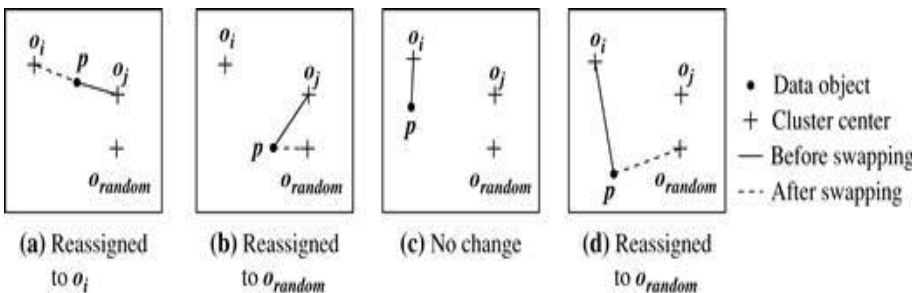


FIGURE Four cases of the cost function for k -medoids clustering.

Each time a reassignment occurs, a difference in absolute error, E , is contributed to the cost function. Therefore, the cost function calculates the *difference* in absolute-error value if a current representative object is replaced by a non representative object. The total cost of swapping is the sum of costs incurred by all non representative objects. If the total cost is negative, then o_j is replaced or swapped with o_{random} because the actual absolute-error E is reduced. If the total cost is positive, the current representative object, o_j , is considered acceptable, and nothing is changed in the iteration.

“Which method is more robust— k -means or k -medoids?” 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, the complexity of each iteration in the k -medoids algorithm is $O(k(n - k))$. For large values of n and k , such computation becomes very costly, and much more costly than the k -means method. Both methods require the user to specify k , the number of clusters.

“How can we scale up the k -medoids method?” A typical k -medoids partitioning algorithm like PAM effectively for small data sets, but does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method called **CLARA (Clustering LARGE Applications)** can be used. Instead of taking the whole data set into consideration, CLARA uses a random sample of the data set. The PAM algorithm is then applied to compute the best medoids from the sample. Ideally, the sample should closely represent the original data set. In many cases, a large sample works well if it is created so that each object has equal probability of being selected into the sample. The representative objects (medoids) chosen will likely be similar to those that would have been chosen from the whole data set. CLARA builds clusterings from multiple random samples and returns the best clustering as the output. The complexity of computing the medoids on a random sample is $O(ks^2 + k(n - k))$, where s is the size of the sample, k is the number of clusters, and n is the total number of objects. CLARA can deal with larger data sets than PAM.

Algorithm: k -medoids. PAM, a 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;

Hierarchical Methods

While partitioning methods meet the basic clustering requirement of organizing a set of objects into a number of exclusive groups, in some situations we may want to partition our data into groups at different levels such as in a hierarchy. A **hierarchical clustering method** works by grouping data objects into a hierarchy or “tree” of clusters.

An **agglomerative hierarchical clustering method** uses a bottom-up strategy. It typically starts by letting each object form its own cluster and iteratively merges clusters into larger and larger clusters, until all the objects are in a single cluster or certain termination conditions are satisfied. The single cluster becomes the hierarchy’s root. For the merging step, it finds the two clusters that are closest to each other (according to some similarity measure), and combines the two to form one cluster. Because two clusters are merged per iteration, where each cluster contains at least one object, an agglomerative method requires at most n iterations.

A **divisive hierarchical clustering method** employs a top-down strategy. It starts by placing all objects in one cluster, which is the hierarchy’s root. It then divides the root cluster into several smaller sub clusters, and recursively partitions those clusters into smaller ones. The partitioning process continues until each cluster at the lowest level is coherent enough—either containing only one object, or the objects within a cluster are sufficiently similar to each other.

In either agglomerative or divisive hierarchical clustering, a user can specify the desired number of clusters as a termination condition.

Example

Agglomerative versus divisive hierarchical clustering

Figure shows the application of **AGNES** (AGglomerative NESTing), an agglomerative hierarchical clustering method, and **DIANA** (DIVisive ANALysis), a divisive hierarchical clustering method, on a data set of five objects, {**a**, **b**, **c**, **d**, **e**}. Initially, AGNES, the agglomerative method, places each object into a cluster of its own. The clusters are then merged step-by-step according to some criterion. For example, clusters C_1 and C_2 may be merged if an object in C_1 and an object in C_2 form the minimum Euclidean distance between any two objects from different clusters. This is a **single-linkage** approach in that each cluster is represented by all the objects in the cluster, and the similarity between two clusters is measured by the similarity of the *closest* pair of data points belonging to different clusters. The cluster-merging process repeats until all the objects are eventually merged to form one cluster.

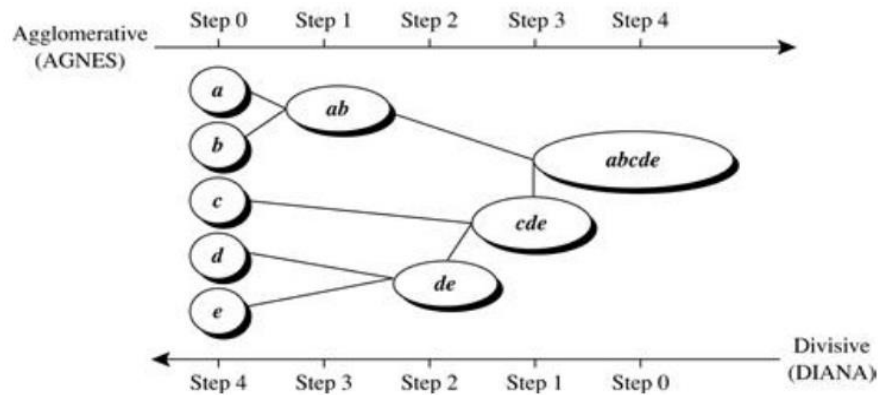
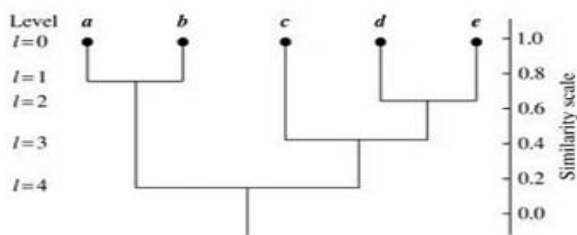


FIGURE Agglomerative and divisive hierarchical clustering on data objects $\{a, b, c, d, e\}$.

DIANA, the divisive method, proceeds in the contrasting way. All the objects are used to form one initial cluster. The cluster is split according to some principle such as the maximum Euclidean distance between the closest neighboring objects in the cluster. The cluster-splitting process repeats until, eventually, each new cluster contains only a single object.

A tree structure called a **dendrogram** is commonly used to represent the process of hierarchical clustering. It shows how objects are grouped together (in an agglomerative method) or partitioned (in a divisive method) step-by-step. Figure shows a dendrogram for the five objects presented in Figure where $l = 0$ shows the five objects as singleton clusters at level 0. At $l = 1$, objects **a** and **b** are grouped together to form the first cluster, and they stay together at all subsequent levels. We can also use a vertical axis to show the similarity scale between clusters. For example, when the similarity of two groups of objects, $\{a, b\}$ and $\{c, d, e\}$, is roughly 0.16, they

are merged together to form a single cluster.



Whether using an agglomerative method or a divisive method, a core need is to measure the distance between two clusters, where each cluster is generally a set of objects.

Four widely used measures for distance between clusters are as follows, where $|p - p'|$ is the distance between two objects or points, **p** and **p'**; m_i is the mean for cluster, C_i ; and n_i is the number of objects in C_i . They are also known as linkage measures.

Minimum distance: $dist_{min}(C_i, C_j) = \min_{p \in C_i, p' \in C_j} \{ |p - p'| \}$

Maximum distance: $dist_{max}(C_i, C_j) = \max_{p \in C_i, p' \in C_j} \{ |p - p'| \}$

Mean distance: $dist_{mean}(C_i, C_j) = |m_i - m_j|$

Average distance: $dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i, p' \in C_j} |p - p'|$

When an algorithm uses the minimum distance, $dmin(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a nearest-neighbor clustering algorithm. Moreover, if the clustering process is terminated when the distance between nearest clusters exceeds a user-defined threshold, it is called a single-linkage algorithm. If we view the data points as nodes of a graph, with edges forming a path between the nodes in a cluster, then the merging of two clusters, C_i and C_j , corresponds to adding an edge between the nearest pair of nodes in C_i and C_j . Because edges linking clusters always go between distinct clusters, the resulting graph will generate a tree. Thus, an agglomerative hierarchical clustering algorithm that uses the minimum distance measure is also called a minimal spanning tree algorithm, where a spanning tree of a graph is a tree that connects all vertices, and a minimal spanning tree is the one with the least sum of edge weights.

When an algorithm uses the maximum distance, $dmax(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a farthest-neighbor clustering algorithm. If the clustering process is terminated when the maximum distance between nearest clusters exceeds a user-defined threshold, it is called a complete-linkage algorithm. By viewing data points as nodes of a graph, with edges linking nodes, we can think of each cluster as a complete subgraph, that is, with edges connecting all the nodes in the clusters. The distance between two clusters is determined by the most distant nodes in the two clusters. Farthest-neighbor algorithms tend to minimize the increase in diameter of the clusters at each iteration. If the true clusters are rather compact and approximately equal size, the method will produce high-quality clusters. Otherwise, the clusters produced can be meaningless.

The previous minimum and maximum measures represent two extremes in measuring the distance between clusters. They tend to be overly sensitive to outliers or noisy data. The use of mean or average distance is a compromise between the minimum and maximum distances and overcomes the outlier sensitivity problem. Whereas the mean distance is the simplest to compute, the average distance is advantageous in that it can handle categorical as well as numeric data. The computation of the mean vector for categorical data can be difficult or impossible to define.

BIRCH: Multiphase Hierarchical Clustering Using Clustering Feature Trees

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is designed for clustering a large amount of numeric data by integrating hierarchical clustering (at the

initial microclustering stage) and other clustering methods such as iterative partitioning (at the later macroclustering stage). It overcomes the two difficulties in agglomerative clustering methods: (1) scalability and (2) the inability to undo what was done in the previous step.

BIRCH uses the notions of clustering feature to summarize a cluster, and clustering feature tree (CF-tree) to represent a cluster hierarchy. These structures help the clustering method achieve good speed and scalability in large or even streaming databases, and also make it effective for incremental and dynamic clustering of incoming objects.

Consider a cluster of n d -dimensional data objects or points. The clustering feature (CF) of the cluster is a 3-D vector summarizing information about clusters of objects. It is defined as

$$C \quad x_0 = \frac{\sum_{i=1}^n x_i}{n} = \frac{LS}{n},$$

$$R = \sqrt{\frac{\sum_{i=1}^n (x_i - x_0)^2}{n}} = \sqrt{\frac{nSS - 2LS^2 + nLS^2}{n^2}},$$

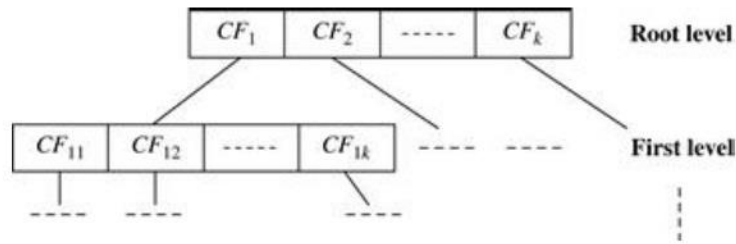
$$D = \sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^n (x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}.$$

where LS is the linear sum of the n points (i.e., $\sum x_i$), and SS is the square sum of the data points (i.e., $\sum x_i^2$).

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

Given a limited amount of main memory, an important consideration in BIRCH is to minimize the time required for input/output (I/O). BIRCH applies a multiphase clustering technique: A single scan of the data set yields a basic, good clustering, and one or more additional scans can optionally be used to further improve the quality. The primary phases are

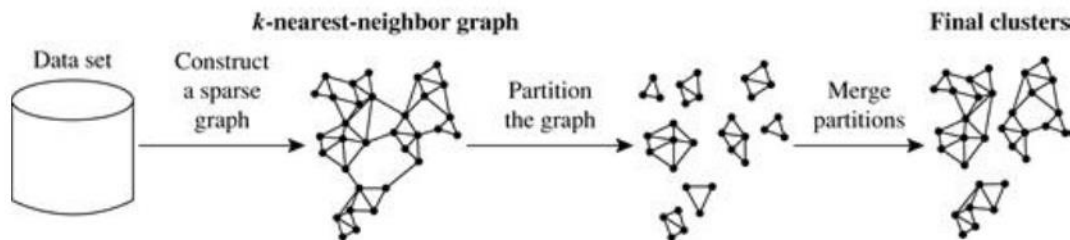
- Phase 1: BIRCH scans the database to build an initial in-memory CF-tree, which can be viewed as a multilevel compression of the data that tries to preserve the data's inherent clustering structure.
- Phase 2: BIRCH applies a (selected) clustering algorithm to cluster the leaf nodes of the CF-tree, which removes sparse clusters as outliers and groups dense clusters into larger ones.



Chameleon: Multiphase Hierarchical Clustering Using Dynamic Modeling

Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to determine the similarity between pairs of clusters. In Chameleon, cluster similarity is assessed based on (1) how well connected objects are within a cluster and (2) the proximity of clusters. That is, two clusters are merged if their interconnectivity is high and they are close together. Thus, Chameleon does not depend on a static, user-supplied model and can automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogeneous clusters and applies to all data types as long as a similarity function can be specified.

Chameleon uses a k -nearest-neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object is among the k -most similar objects to the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partition the k -nearest-neighbor graph into a large number of relatively small sub clusters such that it minimizes the **edge cut**. That is, a cluster C is partitioned into subclusters C_i and C_j so as to minimize the *weight of the edges* that would be cut should C be bisected into C_i and C_j . It assesses the *absolute* interconnectivity between clusters C_i and C_j .



Chameleon then uses an agglomerative hierarchical clustering algorithm that iteratively merges subclusters based on their similarity. To determine the pairs of most similar subclusters, it takes into account both the interconnectivity and the closeness of the clusters. Specifically, Chameleon determines the similarity between each pair of clusters C_i and C_j according to their *relative interconnectivity*, $RI(C_i, C_j)$, and their *relative closeness*, $RC(C_i, C_j)$.

ROCK: A Hierarchical Clustering Algorithm for Categorical Attributes

► **ROCK:** Robust Clustering using linKs

► **Major Ideas**

- Use links to measure similarity/proximity
- Not distance-based
- Computational complexity $O(n^2 + nm_m m_a + n^2 \log n)$
 - ↖ m_a : average number of neighbors
 - ↖ m_m : maximum number of neighbors
 - ↖ n : number of objects

► **Algorithm**

- Sampling-based clustering
- Draw random sample
- Cluster with links
- Label data in disk

➤ **Density Based Methods**

Density-Based Clustering refers to one of the most popular unsupervised learning methodologies used in model building and machine learning algorithms. The data points in the region separated by two clusters of low point density are considered as noise. The surroundings with a radius ϵ of a given object are known as the ϵ neighborhood of the object. If the ϵ neighborhood of the object comprises at least a minimum number, MinPts of objects, then it is called a core object.

1.DBSCAN: A Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density

DBSCAN: Density-Based Clustering Based on Connected Regions with High Density

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) finds core objects, that is, objects that have dense neighborhoods. It connects core objects and their neighborhoods to form dense regions as clusters.

How does DBSCAN quantify the neighborhood of an object?" A user-specified parameter > 0 is used to specify the radius of a neighborhood we consider for every object. The **-neighborhood** of an object o is the space within a radius centered at o .

Due to the fixed neighborhood size parameterized by ϵ , the **density of a neighborhood** can be measured simply by the number of objects in the neighborhood. To determine whether a neighborhood is dense or not, DBSCAN uses another user-specified parameter, MinPts, which specifies the density threshold of dense regions. An object is a **core object** if the ϵ -neighborhood of the object contains at least MinPts objects. Core objects are the pillars of dense regions.

Given a set, D , of objects, we can identify all core objects with respect to the given parameters, ϵ and MinPts. The clustering task is therein reduced to using core objects and

their neighborhoods to form dense regions, where the dense regions are clusters. For a core object q and an object p , we say that p is **directly density-reachable** from q (with respect to ϵ and $MinPts$) if p is within the ϵ -neighborhood of q . Clearly, an object p is directly density-reachable from another object q if and only if q is a core object and p is in the ϵ -neighborhood of q . Using the directly density-reachable relation, a core object can “bring” all objects from its ϵ -neighborhood into a dense region.

“How can we assemble a large dense region using small dense regions centered by core objects?” In DBSCAN, p is **density-reachable** from q (with respect to ϵ and $MinPts$ in D) if there is a chain of objects $p_1, \dots, p_n$, such that $p_1 = q$, $p_n = p$, and p_{i+1} is directly density-reachable from p_i with respect to ϵ and $MinPts$, for $1 \leq i \leq n$, $p_i \in D$. Note that density-reachability is not an equivalence relation because it is not symmetric. If both o_1 and o_2 are core objects and o_1 is density-reachable from o_2 , then o_2 is density-reachable from o_1 . However, if o_2 is a core object but o_1 is not, then o_1 may be density-reachable from o_2 , but not vice versa.

To connect core objects as well as their neighbors in a dense region, DBSCAN uses the notion of density-connectedness. Two objects $p_1, p_2 \in D$ are **density-connected** with respect to ϵ and $MinPts$ if there is an object $q \in D$ such that both p_1 and p_2 are density-reachable from q with respect to ϵ and $MinPts$. Unlike density-reachability, density-connectedness is an equivalence relation. It is easy to show that, for objects o_1, o_2 , and o_3 , if o_1 and o_2 are density-connected, and o_2 and o_3 are density-connected, then so are o_1 and o_3 .

Algorithm: DBSCAN: a density-based clustering algorithm.

Input:

- D : a data set containing n objects,
- ϵ : the radius parameter, and
- $MinPts$: the neighborhood density threshold.

Output: A set of density-based clusters.

Method:

```

(1) mark all objects as unvisited;
(2) do
(3)   randomly select an unvisited object  $p$ ;
(4)   mark  $p$  as visited;
(5)   if the  $\epsilon$ -neighborhood of  $p$  has at least  $MinPts$  objects
(6)     create a new cluster  $C$ , and add  $p$  to  $C$ ;
(7)     let  $N$  be the set of objects in the  $\epsilon$ -neighborhood of  $p$ ;
(8)     for each point  $p'$  in  $N$ 
(9)       if  $p'$  is unvisited
(10)        mark  $p'$  as visited;
(11)        if the  $\epsilon$ -neighborhood of  $p'$  has at least  $MinPts$  points,
(12)          add those points to  $N$ ;
(13)        if  $p'$  is not yet a member of any cluster, add  $p'$  to  $C$ ;
(14)     end for
(15)   output  $C$ ;
(16) else mark  $p$  as noise;
(17) until no object is unvisited;
```

2.OPTICS: Ordering Points to Identify the Clustering Structure

Although DBSCAN can cluster objects given input parameters such as ϵ (the maximum radius of a neighborhood) and $MinPts$ (the minimum number of points required in the neighborhood of a core object), it encumbers users with the responsibility of selecting parameter values that will lead to the discovery of acceptable clusters. This is a problem

associated with many other clustering algorithms. Such parameter settings are usually empirically set and difficult to determine, especially for real-world, high-dimensional data sets. Most algorithms are sensitive to these parameter values: Slightly different settings may lead to very different clusterings of the data. Moreover, real-world, high-dimensional data sets often have very skewed distributions such that their intrinsic clustering structure may not be well characterized by a single set of *global* density parameters.

Note that density-based clusters are monotonic with respect to the neighborhood threshold. That is, in DBSCAN, for a fixed *MinPts* value and two neighborhood thresholds, $\epsilon_1 < \epsilon_2$, a cluster C with respect to ϵ_1 and *MinPts* must be a subset of a cluster C' with respect to ϵ_2 and *MinPts*. This means that if two objects are in a density-based cluster, they must also be in a cluster with a lower density requirement.

To overcome the difficulty in using one set of global parameters in clustering analysis, a cluster analysis method called **OPTICS** was proposed. OPTICS does not explicitly produce a data set clustering. Instead, it outputs a **cluster ordering**. This is a linear list of all objects under analysis and represents the *density-based clustering structure* of the data. Objects in a denser cluster are listed closer to each other in the cluster ordering. This ordering is equivalent to density-based clustering obtained from a wide range of parameter settings. Thus, OPTICS does not require the user to provide a specific density threshold. The cluster ordering can be used to extract basic clustering information (e.g., cluster centers, or arbitrary-shaped clusters), derive the intrinsic clustering structure, as well as provide a visualization of the clustering.

To construct the different clusterings simultaneously, the objects are processed in a specific order. This order selects an object that is density-reachable with respect to the lowest ϵ value so that clusters with higher density (lower ϵ) will be finished first. Based on this idea, OPTICS needs two important pieces of information per object:

- The **core-distance** of an object p is the smallest value ϵ' such that the ϵ' -neighborhood of p has at least *MinPts* objects. That is, ϵ' is the minimum distance threshold that makes p a core object. If p is not a core object with respect to ϵ and *MinPts*, the core-distance of p is undefined.
- The **reachability-distance** to object p from q is the minimum radius value that makes p density-reachable from q . According to the definition of density-reachability, q has to be a core object and p must be in the neighborhood of q . Therefore, the reachability-distance from q to p is $\max\{\text{core-distance}(q), \text{dist}(p, q)\}$. If q is not a core object with respect to ϵ and *MinPts*, the reachability-distance to p from q is undefined.

An object p may be directly reachable from multiple core objects. Therefore, p may have multiple reachability-distances with respect to different core objects. The smallest reachability-distance of p is of particular interest because it gives the shortest path for which p is connected to a dense cluster.

Example 10.8

Core-distance and reachability-distance

Figure illustrates the concepts of core-distance and reachability-distance. Suppose that $\epsilon = 6 \text{ mm}$ and $\text{MinPts} = 5$. The core-distance of p is the distance, ϵ' , between p and the fourth closest data object from p . The reachability-distance of q_1 from p is the core-distance of p (i.e., $\epsilon' = 3 \text{ mm}$) because this is greater than the Euclidean distance from p to q_1 . The reachability-distance of q_2 with respect to p is the Euclidean distance from p to q_2 because this is greater than the core-distance of p .

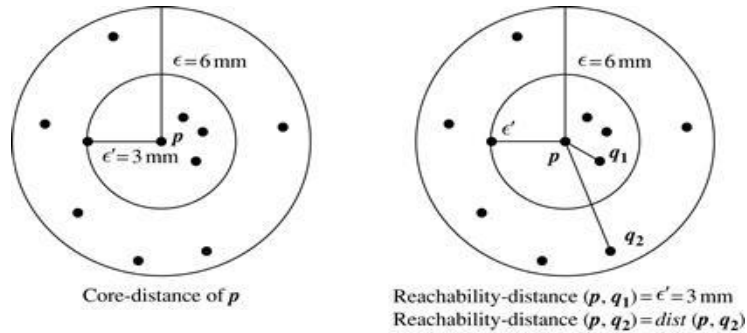


FIGURE OPTICS terminology. *Source:* Based on Ankerst, Breunig, Kriegel, and Sander

OPTICS computes an ordering of all objects in a given database and, for each object in the database, stores the core-distance and a suitable reachability-distance. OPTICS maintains a list called OrderSeeds to generate the output ordering. Objects in OrderSeeds are sorted by the reachability-distance from their respective closest core objects, that is, by the smallest reachability-distance of each object.

OPTICS begins with an arbitrary object from the input database as the current object, p . It retrieves the neighborhood of p , determines the core-distance, and sets the reachability-distance to undefined. The current object, p , is then written to output. If p is not a core object, OPTICS simply moves on to the next object in the OrderSeeds list (or the input database if OrderSeeds is empty). If p is a core object, then for each object, q , in the neighborhood of p , OPTICS updates its reachability-distance from p and inserts q into OrderSeeds if q has not yet been processed. The iteration continues until the input is fully consumed and OrderSeeds is empty.

3. DENCLUE: Clustering Based on Density Distribution Functions

Density estimation is a core issue in density-based clustering methods. DENCLUE (DENSity-based CLUstEring) is a clustering method based on a set of density distribution functions. We first give some background on density estimation, and then describe the DENCLUE algorithm.

In probability and statistics, **density estimation** is the estimation of an unobservable underlying probability density function based on a set of observed data. In the context of density-based clustering, the unobservable underlying probability density function is the

true distribution of the population of all possible objects to be analyzed. The observed data set is regarded as a random sample from that population.

In DBSCAN and OPTICS, density is calculated by counting the number of objects in a neighborhood defined by a radius parameter, ϵ . Such density estimates can be highly sensitive to the radius value used. For example, in Figure the density changes significantly as the radius increases by a small amount.

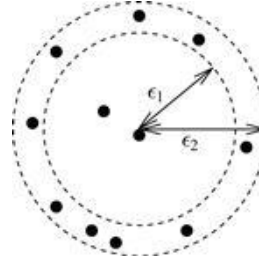


FIGURE The subtlety in density estimation in **DBSCAN** and **OPTICS**: Increasing the neighborhood radius slightly from ϵ_1 to ϵ_2 results in a much higher density.

To overcome this problem, **kernel density estimation** can be used, which is a nonparametric density estimation approach from statistics. The general idea behind kernel density estimation is simple. We treat an observed object as an indicator of high-probability density in the surrounding region. The probability density at a point depends on the distances from this point to the observed objects.

Formally, let $\mathbf{x}_1, \dots, \mathbf{x}_n$ be an independent and identically distributed sample of a random variable f . The kernel density approximation of the probability density function is

$$\hat{f}_h(\mathbf{x}) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \quad (10.21)$$

where $K()$ is a kernel and h is the bandwidth serving as a smoothing parameter. A **kernel** can be regarded as a function modeling the influence of a sample point within its neighborhood. Technically, a kernel $K()$ is a non-negative real-valued integrable function that should satisfy two requirements: and $K(-u) = K(u)$ for all values of u . A frequently used kernel is a standard Gaussian function with a mean of 0 and a variance of 1:

$$K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(\mathbf{x} - \mathbf{x}_i)^2}{2h^2}}.$$

DENCLUE uses a Gaussian kernel to estimate density based on the given set of objects to be clustered. A point $\mathbf{x}^*$ is called a **density attractor** if it is a local maximum of the estimated density function. To avoid trivial local maximum points, **DENCLUE** uses a noise threshold, ξ , and only considers those density attractors $\mathbf{x}^*$ such that. These nontrivial density attractors are the centers of clusters.

A cluster in **DENCLUE** is a set of density attractors X and a set of input objects C such that each object in C is assigned to a density attractor in X , and there exists a path between

every pair of density attractors where the density is above ξ . By using multiple density attractors connected by paths, DENCLUE can find clusters of arbitrary shape.

DENCLUE has several advantages. It can be regarded as a generalization of several well-known clustering methods such as single-linkage approaches and DBSCAN. Moreover, DENCLUE is invariant against noise. The kernel density estimation can effectively reduce the influence of noise by uniformly distributing noise into the input data.

➤ **Quality and Validity of Cluster Analysis**

Measure the Quality of Clustering

- Dissimilarity/Similarity metric: Similarity is expressed in terms of a distance function, which is typically metric: $d(i, j)$
- There is a separate “quality” function that measures the “goodness” of a cluster.
- The definitions of distance functions are usually very different for interval-scaled, boolean, categorical, and ordinal variables.
- Weights should be associated with different variables based on applications and data semantics.

The clustering-quality framework is richer and more flexible than that of clustering functions. In particular, it allows the postulation of axioms that capture the features that Kleinberg’s axioms aim to express, without leading to a contradiction.

A clustering-quality measure is a function that maps pairs of the form $(dataset, clustering)$ to some ordered set (say, the set of non-negative real numbers), so that these values reflect how ‘good’ or ‘cogent’ that clustering is.

Measures for the quality of a clustering are of interest not only as a vehicle for axiomatizing clustering. The need to measure the quality of a given data clustering arises naturally in many clustering issues. The aim of clustering is to uncover meaningful groups in data. However, not any arbitrary partitioning of a given data set reflects such a structure. Upon obtaining a clustering, usually via some algorithm, a user needs to determine whether this clustering is sufficiently meaningful to rely upon for further data mining analysis or practical applications. Clustering-quality measures (CQMs) aim to answer that need by quantifying how good is any specific clustering.

Clustering-quality measures may also be used to help in clustering model-selection by comparing different clusterings over the same data set (e.g., comparing the results of a given clustering paradigm over different choices of clustering parameters, such as the number of clusters).

When posed with the problem of finding a clustering-quality measure, a first attempt may be to invoke the loss (or objective) function used by a clustering algorithm, such as k -means or k -median, as a clustering-quality measure. However, such measures have some shortcomings for the purpose at hand. Namely, these measures are usually not scale-invariant, and they cannot be used to compare the quality of clusterings obtained by different algorithms aiming to minimize different clustering costs (e.g., k -means with different values of k). See Section 6 for more details.

Clustering quality has been previously discussed in the applied statistics literature, where a variety of techniques for evaluating ‘*cluster validity*’ were proposed. Many of these methods, such as the external criteria discussed in [2], are based on assuming some predetermined data generative model, and as such do not answer our quest for a general theory of clustering. In this work, we are concerned with quality measures regardless of any specific generative model, for examples, see the internal criteria surveyed in .

We formulate a theoretical basis for clustering-quality evaluations. We propose a set of requirements (‘axioms’) of clustering-quality measures. We demonstrate the relevance and consistency of these axioms by showing that several natural notions satisfy these requirements. In particular, we introduce quality-measures that reflect the underlying intuition of center-based and linkage-based clustering. These notions all satisfy our axioms, and, given a data clustering, their value on that clustering can be computed in polynomial time.

Paper outline: we begin by presenting Kleinberg’s axioms for clustering functions and discuss their failure. In Section 4.3 we show how these axioms can be translated into axioms pertaining clustering quality measures, and prove that the resulting set of axioms is consistent.

Cluster Validity

For supervised classification we have a variety of measures to evaluate how good our model is

Accuracy, precision, recall

For cluster analysis, the analogous question is how to evaluate the “goodness” of the resulting clusters?

But “clusters are in the eye of the beholder”!

Then why do we want to evaluate them?

- ⊙ To avoid finding patterns in noise
- ⊙ To compare clustering algorithms
- ⊙ To compare two sets of clusters
- ⊙ To compare two clusters

Different Aspects of Cluster Validation

1. Determining the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
 2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.
 3. Evaluating how well the results of a cluster analysis fit the data without reference to external information.
 - Use only the data
 4. Comparing the results of two different sets of cluster analyses to determine which is
-

better.

5. Determining the 'correct' number of clusters.

For 2, 3, and 4, we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

Measures of Cluster Validity

- ⊙ Numerical measures that are applied to judge various aspects of cluster validity, are classified into the following three types.
- ⊙ External Index: Used to measure the extent to which cluster labels match externally supplied class labels.
 - ▣ Entropy
- ⊙ Internal Index: Used to measure the goodness of a clustering structure *without* respect to external information.
 - ▣ Sum of Squared Error (SSE)
- ⊙ Relative Index: Used to compare two different clusterings or clusters.
 - ▣ Often an external or internal index is used for this function, e.g., SSE or entropy
- ⊙ Sometimes these are referred to as criteria instead of indices
- ⊙ However, sometimes criterion is the general strategy and index is the numerical measure that implements the criterion.

Framework for Cluster Validity

- ⊙ Need a framework to interpret any measure.
 - ⊙ For example, if our measure of evaluation has the value, 10, is that good, fair, or poor?
 - ⊙ Statistics provide a framework for cluster validity
 - ⊙ The more "atypical" a clustering result is, the more likely it represents valid structure in the data
 - ⊙ Can compare the values of an index that result from random data or clusterings to those of a clustering result.
 - ▣ If the value of the index is unlikely, then the cluster results are valid
 - ⊙ These approaches are more complicated and harder to understand.
 - ⊙ For comparing the results of two different sets of cluster analyses, a framework is less necessary.
 - ⊙ However, there is the question of whether the difference between two index values is significant
-