LECTURE NOTES

ON

DATA WAREHOUSING & MINING(15A05602)

III B.TECH II SEMESTER

(JNTUA-R15)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VEMU INSTITUTE OF TECHNOLOGY :: P.KOTHAKOTA
Chittoor-Tirupati National Highway, P.Kothakota, Near Pakala, Chittoor (Dt.), AP - 517112
(Approved by AICTE, New Delhi Affiliated to JNTUA Ananthapuramu. ISO 9001:2015 Certified Institute)
Course Objectives:
To know the basic concepts and principles of data warehousing and data mining
Learn pre-processing techniques and data mining functionalities
Learn and create multidimensional models for data warehousing
Study and evaluate performance of Frequent Item sets and Association Rules
Understand and Compare different types of classification and clustering algorithms

Course Outcomes:
Understand the basic concepts of data warehouse and data mining
Apply pre-processing techniques for data cleansing
Analyze and evaluate performance of algorithms for Association Rules
Analyze Classification and Clustering algorithms

UNIT I
Introduction: Fundamentals of data mining, Data Mining Functionalities, Classification of Data Mining systems, Data Mining Task Primitives, Integration of a Data Mining System with a Database or a Data Warehouse System, Major issues in Data Mining. Data Preprocessing: Need for Preprocessing the Data, Data Cleaning, Data Integration and Transformation, Data Reduction, Discretization and Concept Hierarchy Generation.

UNIT II
Data Warehouse and OLAP Technology for Data Mining: Data Warehouse, Multidimensional Data Model, Data Warehouse Architecture, Data Warehouse Implementation, Further Development of Data Cube Technology, From Data Warehousing to Data Mining. Data Cube Computation and Data Generalization: Efficient Methods for Data Cube Computation, Further Development of Data Cube and OLAP Technology, Attribute-Oriented Induction.

UNIT III
Mining Frequent Patterns, Associations and Correlations: Basic Concepts, Efficient and Scalable Frequent Itemset Mining Methods, Mining various kinds of Association Rules, From Association Mining to Correlation Analysis, Constraint-Based Association Mining, Classification and Prediction: Issues Regarding Classification and Prediction, Classification by Decision Tree Induction, Bayesian Classification, Rule-Based Classification, Classification by Back propagation, Support Vector Machines, Associative Classification, Lazy Learners, Other Classification Methods, Prediction, Accuracy and Error measures, Evaluating the accuracy of a Classifier or a Predictor, Ensemble Methods.

UNIT IV
Cluster Analysis Introduction: Types of Data in Cluster Analysis, A Categorization of Major Clustering Methods, Partitioning Methods, Hierarchical Methods, Density-Based Methods, Grid-Based Methods, Model-Based Clustering Methods, Clustering High-Dimensional Data, Constraint-Based Cluster Analysis, Outlier Analysis.

UNIT V
Mining Streams, Time Series and Sequence Data: Mining Data Streams, Mining Time-Series Data, Mining Sequence Patterns in Transactional Databases, Mining Sequence Patterns in Biological Data, Graph Mining, Social Network Analysis and Multi-relational Data Mining, Mining Object, Spatial, Multimedia, Text and Web Data: Multidimensional Analysis and Descriptive Mining of Complex Data Objects, Spatial Data Mining, Multimedia Data Mining, Text Mining, Mining the World Wide Web.

TEXT BOOKS:
2. Introduction to Data Mining – Pang-Ning Tan, Michael Steinbach and Vipin Kumar, Pearson Education.

REFERENCES:
What Is Data Mining?

Data mining refers to extracting or mining knowledge from large amounts of data. The term is actually a misnomer. Thus, data mining should have been more appropriately named as knowledge mining which emphasizes mining from large amounts of data.

It is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use.

The key properties of data mining are

Automatic discovery of patterns
Prediction of likely outcomes
Creation of actionable information
Focus on large datasets and databases

The Scope of Data Mining

Data mining derives its name from the similarities between searching for valuable business information in a large database — for example, finding linked products in gigabytes of store scanner data — and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find exactly where the value resides. Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing these capabilities:
Automated prediction of trends and behaviors. Data mining automates the process of finding predictive information in large databases. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data — quickly. A typical example of a predictive problem is targeted marketing. Data mining uses data on past promotional mailings to identify the targets most likely to maximize return on investment in future mailings. Other predictive problems include forecasting bankruptcy and other forms of default, and identifying segments of a population likely to respond similarly to given events.

Automated discovery of previously unknown patterns. Data mining tools sweep through databases and identify previously hidden patterns in one step. An example of pattern discovery is the analysis of retail sales data to identify seemingly unrelated products that are often purchased together. Other pattern discovery problems include detecting fraudulent credit card transactions and identifying anomalous data that could represent data entry keying errors.

Tasks primitives of Data Mining:

Data mining involves six common classes of tasks:

- **Anomaly detection (Outlier/change/deviation detection)** – The identification of unusual data records, that might be interesting or data errors that require further investigation.

- **Association rule learning (Dependency modelling)** – Searches for relationships between variables. For example a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.

- **Clustering** – is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data.

- **Classification** – is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam".

- **Regression** – attempts to find a function which models the data with the least error.
Summarization – providing a more compact representation of the data set, including visualization and report generation.

Architecture of Data Mining

A typical data mining system may have the following major components.

1. **Knowledge Base:**
   This is the domain knowledge that is used to guide the search or evaluate the interestingness of resulting patterns. Such knowledge can include concept hierarchies, used to organize attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may also be included. Other examples of domain knowledge are additional interestingness constraints or thresholds, and metadata (e.g., describing data from multiple heterogeneous sources).

2. **Data Mining Engine:**
   This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association and correlation analysis, classification, prediction, cluster analysis, outlier analysis, and evolution analysis.

3. **Pattern Evaluation Module:**
   This component typically employs interestingness measures interacts with the data mining modules so as to focus the search toward interesting patterns. It may use interestingness thresholds to filter out discovered patterns. Alternatively, the pattern evaluation module may be integrated with the mining module, depending on the implementation of the data mining method used. For efficient data mining, it is highly recommended to push the evaluation of pattern interestingness as deep as possible into the mining processes as to confine the search to only the interesting patterns.

4. **User Interface:**
   This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing exploratory data mining based on the intermediate data mining results. In addition, this component allows the user to browse database and data warehouse schemas or data structures, evaluate mined patterns, and visualize the patterns in different forms.
Data Mining Process:

Data Mining is a process of discovering various models, summaries, and derived values from a given collection of data.

The general experimental procedure adapted to data-mining problems involves the following steps:

5. State the problem and formulate the hypothesis

Most data-based modeling studies are performed in a particular application domain. Hence, domain-specific knowledge and experience are usually necessary in order to come up with a meaningful problem statement. Unfortunately, many application studies tend to focus on the data-mining technique at the expense of a clear problem statement. In this step, a modeler usually specifies a set of variables for the unknown dependency and, if possible, a general form of this dependency as an initial hypothesis. There may be several hypotheses formulated for a single problem at this stage. The first step requires the combined expertise of an application domain and a data-mining model. In practice, it usually means a close interaction between the data-mining expert and the application expert. In successful data-mining applications, this cooperation does not stop in the initial phase; it continues during the entire data-mining process.

6. Collect the data

This step is concerned with how the data are generated and collected. In general, there are two distinct possibilities. The first is when the data-generation process is under the control of an expert (modeler): this approach is known as a designed experiment. The second possibility is when the expert cannot influence the data-generation process: this is known as the observational approach. An observational setting, namely, random data generation, is assumed in most data-mining applications. Typically, the sampling
distribution is completely unknown after data are collected, or it is partially and implicitly given in the data-collection procedure. It is very important, however, to understand how data collection affects its theoretical distribution, since such a priori knowledge can be very useful for modeling and, later, for the final interpretation of results. Also, it is important to make sure that the data used for estimating a model and the data used later for testing and applying a model come from the same, unknown, sampling distribution. If this is not the case, the estimated model cannot be successfully used in a final application of the results.

7. Preprocessing the data

In the observational setting, data are usually "collected" from the existing databases, data warehouses, and data marts. Data preprocessing usually includes at least two common tasks:

1. Outlier detection (and removal) – Outliers are unusual data values that are not consistent with most observations. Commonly, outliers result from measurement errors, coding and recording errors, and, sometimes, are natural, abnormal values. Such nonrepresentative samples can seriously affect the model produced later. There are two strategies for dealing with outliers:

   a. Detect and eventually remove outliers as a part of the preprocessing phase, or
   b. Develop robust modeling methods that are insensitive to outliers.

2. Scaling, encoding, and selecting features – Data preprocessing includes several steps such as variable scaling and different types of encoding. For example, one feature with the range [0, 1] and the other with the range [−100, 1000] will not have the same weights in the applied technique; they will also influence the final data-mining results differently. Therefore, it is recommended to scale them and bring both features to the same weight for further analysis. Also, application-specific encoding methods usually achieve
dimensionality reduction by providing a smaller number of informative features for subsequent data modeling.

These two classes of preprocessing tasks are only illustrative examples of a large spectrum of preprocessing activities in a data-mining process. Data-preprocessing steps should not be considered completely independent from other data-mining phases. In every iteration of the data-mining process, all activities, together, could define new and improved data sets for subsequent iterations. Generally, a good preprocessing method provides an optimal representation for a data-mining technique by incorporating a priori knowledge in the form of application-specific scaling and encoding.

4. Estimate the model

The selection and implementation of the appropriate data-mining technique is the main task in this phase. This process is not straightforward; usually, in practice, the implementation is based on several models, and selecting the best one is an additional task. The basic principles of learning and discovery from data are given in Chapter 4 of this book. Later, Chapter 5 through 13 explain and analyze specific techniques that are applied to perform a successful learning process from data and to develop an appropriate model.

5. Interpret the model and draw conclusions

In most cases, data-mining models should help in decision making. Hence, such models need to be interpretable in order to be useful because humans are not likely to base their decisions on complex "black-box" models. Note that the goals of accuracy of the model and accuracy of its interpretation are somewhat contradictory. Usually, simple models are more interpretable, but they are also less accurate. Modern data-mining methods are expected to yield highly accurate results using high-dimensional models. The problem of interpreting these models, also very important, is considered a separate task, with specific
techniques to validate the results. A user does not want hundreds of pages of numeric results. He does not understand them; he cannot summarize, interpret, and use them for successful decision making.

The Data mining Process

Classification of Data mining Systems:

The data mining system can be classified according to the following criteria:

- Database Technology
- Statistics
- Machine Learning
- Information Science
- Visualization
- Other Disciplines
Some Other Classification Criteria:

Classification according to kind of data ases mined
Classification according to kind of knowledge mined
Classification according to kinds of tech niques utilized
Classification according to applications adapted

1. Classification according to kind of databases mined

We can classify the data mining system according to kind of databases mined. Database system can be classified according to different criteria such as data models, types of data etc. And the data mining system can be classified accordingly. For example if we classify the database according to data model then we may have a relational, transactional, object-relational, or data warehouse mining system.

2. Classification according to kind of knowledge mined

We can classify the data mining system according to kind of knowledge mined. It is means data mining system are classified on the basis of functionalities such as:

- Outlier Analysis
- Evolution Analysis

3. Classification according to kinds of techniques utilized

We can classify the data mining system according to kind of techniques used. We can describes these techniques according to degree of user interaction involved or the methods of analysis employed.

4. Classification according to applications adapted

We can classify the data mining system according to application adapted. These applications are as follows:

- Finance
- Telecommu nications
- DNA

Data mining query languages and ad hoc data mining. - Data Mining Query language that allows the user to describe ad hoc mining tasks, should be integrated with a
Data warehouse query language and optimized for efficient and flexible data mining.

Presentation and visualization of data mining results. - Once the patterns are discovered, it needs to be expressed in high-level languages, visual representations. These representations should be easily understandable by the users.

Handling noisy or incomplete data. - The data cleaning methods are required that can handle the noise, incomplete objects while mining the data regularities. If data cleaning methods are not there, then the accuracy of the discovered patterns will be poor.

Pattern evaluation. - It refers to interestingness of the problem. The patterns discovered should be interesting because either they represent common knowledge or lack novelty.

- Efficiency and scalability of mining algorithms. - In order to effectively extract information from huge amounts of data in databases, data mining algorithms must be efficient and scalable.

Parallel, distributed, and incremental mining algorithms. - The factors such as huge size of databases, wide distribution of data, and complexity of data mining methods motivate the development of parallel and distributed data mining algorithms. These algorithms divide the data into partitions which are further processed parallel. Then the results from the partitions are merged.
Data Warehouse:

A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process. Subject-Oriented: A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.

Integrated: A data warehouse integrates data from multiple data sources. For example, source A and source B may have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.

Time-Variant: Historical data is kept in a data warehouse. For example, one can retrieve data from 3 months, 6 months, 12 months, or even older data from a data warehouse. This contrasts with a transactions system, where often only the most recent data is kept. For example, a transaction system may hold the most recent address of a customer, where a data warehouse can hold all addresses associated with a customer.

Non-volatile: Once data is in the data warehouse, it will not change. So, historical data in a data warehouse should never be altered.

Data Warehouse Design Process:

A data warehouse can be built using a top-down approach, a bottom-up approach, or a combination of both.

- The top-down approach starts with the overall design and planning. It is useful in cases where the technology is mature and well known, and where the business problems that must be solved are clear and well understood.

- The bottom-up approach starts with experiments and prototypes. This is useful in the early stage of business modeling and technology development. It allows an organization to move forward at considerably less expense and to evaluate the benefits of the technology before making significant commitment.

A Three Tier Data Warehouse Architecture:
Tier-1:

The bottom tier is a warehouse database server that is almost always a relational database system. Backend tools and utilities are used to feed data into the bottom tier from operational databases or other external sources (such as customer profile information provided by external consultants). These tools and utilities perform data extraction, cleaning, and transformation (e.g., to merge similar data from different sources into a unified format), as well as load and refresh functions to update the data warehouse. The data are extracted using application program interfaces known as gateways. A gateway is an example of gateways include ODBC (Open Database Connection) and OLEDB (Open Linking and Embedding for Databases) by Microsoft and JDBC (Java Database Connection).

This tier also contains a metadata repository, which stores information about the data warehouse and its contents.

Tier-2:

The middle tier is an OLA server that is typically implemented using either a relational OLAP (ROLAP) model or a multidimensional OLAP.

- OLAP model is an extended relational DBMS that maps operations on
multidimensional data to standard relational operations.

- A multidimensional OLA P (MOLAP) model, that is, a special-purpose server that directly implements multidimensional data and operations.

Tier-3:

The top tier is a front-end client layer, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on). supported by the underlying DBMS and allows client programs to generate SQL code to be executed at a server.

Data Warehouse Models:

There are three data warehouse models.

1. Enterprise warehouse:
   - An enterprise warehouse collects all of the information about subjects spanning the entire organization.
   - It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope.
   - It typically contains detailed data as well as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond.
   - An enterprise data warehouse may be implemented on traditional mainframes, computer superservers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

2. Data mart:
   - A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales.
   - Depending on the source of data, data marts can be categorized as independent or dependent. Independent data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area. Dependent data marts are sourced directly from enterprise data warehouses.
3. Virtual warehouse:

- A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but requires excess capacity on operational database servers.

Meta Data Repository:

Metadata are data about data. When used in a data warehouse, metadata are the data that define warehouse objects. Metadata are created for the data names and definitions of the given warehouse. Additional metadata are created and captured for timestamping any extracted data, the source of the extracted data, and missing fields that have been added by data cleaning or integration processes.

A metadata repository should contain the following:

- A description of the structure of the data warehouse, which includes the warehouse schema, view, dimensions, hierarchies, and derived data definitions, as well as data mart locations and contents.
- Operational metadata, which include data lineage (history of migrated data and the sequence of transformations applied to it), currency of data (active, archived, or purged), and monitoring information (warehouse usage statistics, error reports, and audit trails).
- The algorithms used for summarization, which include measure and dimension.
Data Preprocessing:

Data Integration:

It combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

The data integration systems are formally defined as triple \( <G,S,M> \) Where G: The global schema  
S: Heterogeneous source of schemas  
M: Mapping between the queries of source and global schema

Issues in Data integration:

1. Schema integration and object matching:

   How can the data analyst or the computer be sure that customer id in one database and customer number in another reference to the same attribute.

2. Redundancy:

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

3. Detection and resolution of data value conflicts:

   For the same real-world entity, attribute values from different sources may differ.
Data Transformation:

In data transformation, the data are transformed or consolidated into forms appropriate for mining.

Data transformation can involve the following:

- **Smoothing**, which works to remove noise from the data. Such techniques include binning, regression, and clustering.
- **Aggregation**, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for analysis of the data at multiple granularities.
- **Generalization of the data**, where low-level or — primitively (raw) data are replaced by higher-level concepts through the use of concept hierarchies. For example, categorical attributes, like street, can be generalized to higher-level concepts, like city or country.
- **Normalization**, where the attribute data are scaled so as to fall within a small specified range, such as 0:0 to 1:0, or 0:0 to 1:0.
- **Attribute construction** (or feature construction), where new attributes are constructed and added from the given set of attributes to help the mining process.

Data Reduction:

Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results.

Strategies for data reduction include the following:

- **Data cube aggregation**, where aggregation operations are applied to the data in the construction of a data cube.
- **Attribute subset selection**, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.
- **Dimensionality reduction**, where encoding mechanisms are used to reduce the dataset size.
- **Numerosity reduction**, where the data are replaced or estimated by alternative,
smaller data representations such as parametric models (which need store only the model parameters instead of the actual data) or nonparametric methods such as clustering, sampling, and the use of histograms.

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 datamining, in that they allow the mining of data at multiple levels of abstraction.
Part A: (2 Marks)

1. Define Data mining.

   It refers to extracting or “mining” knowledge from large amount of data. Data mining is a process of discovering interesting knowledge from large amounts of data stored either, in database, data warehouse, or other information repositories.

2. Give some alternative terms for data mining.

   - Knowledge mining
   - Knowledge extraction
   - Data/pattern analysis.
   - Data Archaeology
   - Data dredging

3. What is KDD.

   KDD-Knowledge Discovery in Databases.

4. What are the steps involved in KDD process.

   - Data cleaning
   - Data Mining
   - Pattern Evaluation
   - Data Integration
   - Data Selection
   - Data Transformation

5. What is the use of the knowledge base?

   Knowledge base is domain knowledge that is used to guide search or evaluate the interestingness of resulting pattern. Such knowledge can include concept hierarchies used to organize attribute/attribute values in to different levels of abstraction.

6. Mention some of the data mining techniques.

   - Statistics
     - Machine learning
     - Decision Tree
     - Hidden markov models
   - Artificial Intelligence
     - Genetic Algorithm
   - Meta learning

7. Give few statistical techniques.

   - Point Estimation
     - Data Summarization
   - Bayesian Techniques
   - Testing Hypothesis
   - Correlation
   - Regression

8. What is meta learning.

   Concept of combining the predictions made from multiple models of data mining and analyzing those predictions to formulate a new and previously unknown prediction.

   GUI Pattern Evaluation
Search algorithm enables us to locate optimal binary string by processing an initial random population of binary strings by performing operations such as artificial mutation, crossover and selection.

10. What is the purpose of Data mining Technique?
It provides a way to use various data mining tasks.

11. Define Predictive model.
It is used to predict the values of data by making use of known results from a different set of sample data.

12. Data mining tasks that are belongs to predictive model
   - Classification
   - Regression
   - Time series analysis

13. Define descriptive model
It is used to determine the patterns and relationships in a sample data. Data mining tasks that belongs to descriptive model:
   - Clustering
   - Summarization
   - Association rules
   - Sequence discovery

14. Define the term summarization?
The summarization of a large chunk of data contained in a web page or a document.
   Summarization = characterization = generalization

15. List out the advanced database systems.
   - Extended-relational databases
   - Object-oriented databases
   - Deductive databases
   - Spatial databases
   - Temporal databases
   - Multimedia databases
   - Active databases
   - Scientific databases
   - Knowledge databases

16. Classifications of Data mining systems. Based on the kinds of databases mined: According to model
   - Relational mining system
   - Transactional mining system
   - Object-oriented mining system
   - Object-Relational mining system
   - Data warehouse mining system
   - Types of Data
   - Spatial data mining system
17. what are the functionalities of data mining?

- Characterization
- Discrimination
- Association
- Classification
- Clustering
- Outlier analysis
- Evolution analysis

18. According to levels of abstraction of the knowledge mined how many types are there?

- Generalized knowledge (High level of abstraction)
- Primitive-level knowledge (Raw data level)

19. Describe challenges to data mining regarding data mining methodology and user interaction issues.

- Mining different kinds of knowledge in databases
  - Interactive mining of knowledge at multiple levels of abstraction
  - Incorporation of background knowledge
- Data mining query languages and ad hoc data mining
- Presentation and visualization of data mining results
- Handling noisy or incomplete data
- Pattern evaluation

20. Describe challenges to data mining regarding performance issues.

- Efficiency and scalability of data mining algorithms
  - Parallel, distributed, and incremental mining algorithms
Part B: (10 Marks)

1. Define the datamining and their functionalities?
2. Define data mining. On what kind of data, data mining can be performed?
3. Compare and contrast operational database systems with data warehouse.
4. What is the importance of data marts in data warehouse?
5. Discuss about a three-tier data warehouse architecture?
6. What are Major issues in Data Mining?
7. What is Data Preprocessing and explain the Need for Preprocessing in the Data mining?
8. Explain briefly about Data Cleaning?
9. What is Data Integration and Transformation explain in detail?
10. Explain briefly about Concept Hierarchy Generation with neat diagram?
INTRODUCTION AND NEED FOR DATA WAREHOUSE

The construction of data warehouses, which involves data cleaning and data integration, can be viewed as an important preprocessing step for data mining. Moreover, data warehouses provide on-line analytical processing (OLAP) tools for the interactive analysis of multidimensional data of varied granularities, which facilitates effective data mining. Furthermore, many other data mining functions such as classification, prediction, association, and clustering, can be integrated with OLAP operations to enhance interactive mining of knowledge at multiple levels of abstraction.

Hence, data warehouse has become an increasingly important platform for data analysis and online analytical processing and will provide an effective platform for data mining. Such an overview is essential for understanding data mining technology.

WHAT IS DATA WAREHOUSE?

Defined in many different ways:

A decision support database that is maintained separately from the organization’s operational database.

Support information processing by providing a solid platform of consolidated, historical data for analysis.

Data warehousing provides architectures and tools for business executives to systematically organize, understand, and use their data to make strategic decisions. According to W. 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 of management’s decision making process.”

Subject Oriented: Data that gives information about a particular subject instead of about a company’s ongoing operations.

Integrated: Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.

Time-variant: All data in the data warehouse is identified with a particular time period.

Non-volatile: Data is stable in a data warehouse. More data is added but data is never removed. This enables management to gain a consistent picture of the business.

Data warehousing is defined as the process of constructing and using data warehouses. The construction of a data warehouse requires data integration, data cleaning, and data consolidation. The utilization of a data warehouse often necessitates a collection of decision support technologies. This allows ‘knowledge workers” (e.g., managers, analysts)

(3) Analyzing operations and looking for sources of profit, and
(4) managing the customer relationships, making environmental corrections, and managing the cost of corporate assets.

Data warehousing is also very useful from the point of view of heterogeneous database integration. Many organizations typically collect diverse kinds of data and maintain large databases from multiple, heterogeneous, autonomous, and distributed information sources. To integrate such data, and provide easy and efficient access to it is highly desirable, yet challenging. Much effort has been spent in the database industry and research community towards achieving this goal.

Data Warehousing Architecture

The design of a data warehouse: A business analysis framework

First, having a data warehouse may provide a competitive advantage by presenting relevant information from which to measure performance and make critical adjustments in order to help win over competitors.

Second, a data warehouse can enhance business productivity since it is able to quickly and efficiently gather information which accurately describes the organization. Third, a data warehouse facilitates customer relationship marketing since it provides a consistent view of customers and items across all lines of business, all departments, and all markets.

Finally, a data warehouse may bring about cost reduction by tracking trends, patterns, and exceptions over long periods of time in a consistent and reliable manner.

To design an effective data warehouse one needs to understand and analyze business needs, and construct a business analysis framework. The construction of a large and complex information system can be viewed as the construction of a large and complex building, for which the owner, architect, and builder have different views.
Four different views regarding the design of a data warehouse must be considered: the top-down view, the data source view, the data warehouse view, and the business query view.

The top-down view allows the selection of the relevant information necessary for the data warehouse. This information matches the current and coming business needs. The data source view exposes the information being captured, stored, and managed by operational systems. This information may be documented at various levels of detail and accuracy, from individual data source tables to integrated data source tables. Data sources are often modeled by traditional data modeling techniques, such as the entity-relationship model or CASE (Computer Aided Software Engineering) tools.

The data warehouse view includes fact tables and dimension tables. It represents the information that is stored inside the data warehouse, including pre-calculated totals and counts, as well as information regarding the source, date, and time of origin, added to provide historical context.

Finally, the business query view is the perspective of data in the data warehouse from the view point of the end-user.

The process of data warehouse design

A data warehouse can be built using a top-down approach, a bottom-up approach, or a combination of both.

- Top-down: Starts with overall design and planning (mature)
- Bottom-up: Starts with experiments and prototypes (rapid) From software engineering point of view
- Waterfall: structured and systematic analysis at each step before proceeding to the next
- Spiral: rapid generation of increasingly functional systems, short turn around time, quick turn around

Typical data warehouse design process

- Choose a business process to model, e.g., orders, invoices, etc.
- Choose the grain (atomic level of data) of the business process
Choose the dimensions that will apply to each fact table record
Choose the measure that will populate each fact table record

Various kinds of data warehouse design tools are available. Data warehouse development tools provide functions to define and edit metadata repository contents such as schemas, scripts or rules, answer queries, output reports, and ship metadata to and from relational database system catalogues. Planning and analysis tools study the impact of schema changes and of refresh performance when changing refresh rates or time windows.

Three-tier data warehouse architecture

Data warehouses often adopt three-tier architecture, as presented in Fig The bottom tier is a warehouse database server which is almost always a relational database system. The middle tier is an OLAP server which is typically implemented using either

1. A Relational OLAP (ROLAP) model, i.e., an extended relational DBMS that maps operations on multidimensional data to standard relational operations; or

2. A Multidimensional OLAP (MOLAP) model, i.e., a special purpose server that directly implements multidimensional data and operations. The top tier is a client, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).
From the architecture point of view, there are three data warehouse models: the enterprise warehouse, the data mart, and the virtual warehouse.

**Enterprise warehouse:**

An enterprise warehouse collects all of the information about subjects spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope. It typically contains detailed data as well as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond.

An enterprise data warehouse may be implemented on traditional mainframes, UNI X super servers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.
Data mart:

A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific, selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized. Data marts are usually implemented on low cost departmental servers that are UNIX-, Windows/NT-, or OS/2-based. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years.

However, it may involve complex integration in the long run if its design and planning were not enterprise-wide. Depending on the source of data, data marts can be categorized into the following two classes:

- Independent data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area.
- Dependent data marts are sourced directly from enterprise data warehouses.

Virtual warehouse:

A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but requires excess capacity on operational database servers.
Figure Multi-tiered Architecture

A recommended method for the development of data warehouse systems is to implement the warehouse in an incremental and evolutionary manner, as shown in fig. First, a high-level corporate data model is defined within a reasonably short period of time (such as one or two months) that provides a corporate-wide, consistent, integrated view of data among different subjects and potential usages. This high-level model, although it will need to be refined in the further development of enterprise data warehouses and departmental data marts, will greatly reduce future integration problems.

Figure: A recommended approach for data warehouse development.
Second, independent data marts can be implemented in parallel with the enterprise warehouse based on the same corporate data model set as above. Third, distributed data marts can be constructed to integrate different data marts via hub servers. Finally, a multi-tier data warehouse is constructed where the enterprise warehouse is the sole custodian of all warehouse data; this is then distributed to the various dependent data marts.

OLAP server architectures: ROLAP vs. MOLAP vs. HOLAP

Logically, OLAP engines present business users with multidimensional data from data warehouses or data marts, without concerns regarding how or where the data are stored.

However, the physical architecture and implementation of OLAP engines must consider data storage issues. Implementations of a warehouse server engine for OLAP processing include:

Relational OLAP (ROLAP) servers:
These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use a relational or extended-relational DBMS to store and manage warehouse data, and OLAP middleware to support missing pieces. ROLAP servers include optimization for each DBMS backend, implementation of aggregation navigation logic, and additional tools and services. ROLAP technology tends to have greater scalability than MOLAP technology. The DSS server of Microstrategy and Metacube of Informix, for example, adopt the ROLAP approach.

Multidimensional OLAP (MOLAP) servers:
These servers support multidimensional views of data through array-based multidimensional storage engines. They map multidimensional views directly to data cube array structures. For example, Essbase of Arbor is a MOLAP server. The advantage of using a data cube is that it allows fast indexing to precomputed summarized data. Notice that with multidimensional data stores, the storage utilization may be low if the data set is sparse. In such cases, sparse matrix compression techniques should be explored. Many OLAP servers adopt a two-level storage representation to handle sparse and dense data sets: the dense subcubes are identified and stored as array structures, while the sparse subcubes employ compression technology for efficient storage utilization.

Hybrid OLAP (HOLAP) servers:
The hybrid OLAP approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may allow large volumes of detail data to be stored in a relational database, while aggregations are kept in a separate MOLAP
store. The Microsoft SQL Server 7.0 O AP Services supports a hybrid OLAP server.

Specialized SQL servers:

To meet the growing demand of OLAP processing in relational databases, some relational and data warehousing firms (e.g., Redbrick) implement specialized SQL servers which provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

The OLAP functional architecture consists of three components: the data store, the OLAP server, and the user presentation module. The data store can be further classified as a relational data store or a multidimensional data store, depending on whether ROLAP or MOLAP architecture is adopted.

Data warehouse implementation

Data warehouses contain huge volumes of data. OLAP engines demand that decision support queries be answered in the order of seconds. Therefore, it is crucial for data warehouse systems to support highly efficient cube computation techniques, access methods, and query processing techniques. ‘How can this be done?’, you may wonder. In this section, we examine methods for the efficient implementation of data warehouse systems.

```
Figure: Lattice of cuboids, making up a 3-dimensional data cube.

Efficient computation of data cubes

At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions. In SQL terms, these aggregations are referred to as group-by's. The compute cube operator and its implementation One approach to cube computation extends SQL so as to include a compute cube operator. The compute cube
```
operator computes aggregates over all subsets of the dimensions specified in the operation. Example 10 Suppose that you would like to create a data cube for All-Electronics sales which contains the following: item, city, year, and sales in dollars. You would like to be able to analyze the data, with queries such as the following:

1. “Compute the sum of sales, grouping by item and city.”
2. “Compute the sum of sales, grouping by item.”
3. “Compute the sum of sales, grouping by city”.

What is the total number of cuboids, or group-by’s that can be computed for this data cube? Taking the three attributes, city, item, and year, as three dimensions and sales in dollars as the measure, the total number of cuboids, or group-by’s that can be computed for this data cube is 23 = 8. The possible group-by’s are the following: (city; item; year), (city; item), (city; year), (item; year), (city), (item), (year), ()g, where () means that the group-by is empty (i.e., the dimensions are not grouped).

This group-by’s form a lattice of cuboids for the data cube, as shown in fig. The base cuboid contains all three dimensions, city, item, and year. It can return the total sales for any combination of the three dimensions. The apex cuboid, or 0-D cuboid, refers to the case where the group-by is empty. It contains the total sum of all sales. Consequently, it is represented by the special value all.

An SQL query containing no group-by, such as ‘compute the sum of total sales” is a zero-dimensional operation. An SQL query containing one group-by, such as ‘compute the sum of sales, group by city” is a one-dimensional operation. A cube operator on n dimensions is equivalent to a collection of group by statements, one for each subset of the n dimensions. Therefore, the cube operator is the n-dimensional generalization of the group by operator. The data cube in Example 11, can be defined as

cube sales [item, city, year]: sum(sales in dollars)

For a cube with n dimensions, there are a total of 2n cuboids, including the base cuboid. The statement compute cube sales explicitly instructs the system to compute the sales aggregate cuboids for all of the eight subsets of the set of {item, city, year}, including the empty subset. A cube computation operator was first proposed and studied by Gray, et al. (1996).

On-line analytical processing may need to access different cuboids for different queries. Therefore, it does seem like a good idea to compute all or at least some of the cuboids in a data cube in advance. A pre-computation lead to fast response time and avoids some redundant computation. Actually, most, if not all, OLAP products resort to some degree of pre-computation of multidimensional aggregates.
A major challenge related to this pre-computation, however, is that the required storage space may explode if all of the cuboids in a data cube are pre-computed, especially when the cube has several dimensions associated with multiple level hierarchies. “How many cuboids are there in an n-dimensional data cube?” If there were no hierarchies associated with each dimension, then the total number of cuboids for an n-dimensional data cube, as we have seen above, is $2^n$.

However, in practice, many dimensions do have hierarchies. For example, the dimension time is usually not just one level, such as year, but rather a hierarchy or a lattice, such as day < week < month < quarter < year. For an n-dimensional data cube, the total number of cuboids that can be generated (including the cuboids generated by climbing up the hierarchies along each dimension) is:

$$T = \prod_{i=1}^{n} (L_i + 1),$$

where $L_i$ is the number of levels associated with dimension $i$ (excluding the virtual top level all since generalizing to all is equivalent to the removal of a dimension). This formula is based on the fact that at most one abstraction level in each dimension will appear in a cuboid. For example, if the cube has 10 dimensions and each dimension has 4 levels, the total number of cuboids that can be generated.

By now, you probably realize that it is unrealistic to pre-compute and materialize all of the cuboids that can possibly be generated for a data cube (or, from a base cuboid). If there are many cuboids, and these cuboids are large in size, a more reasonable option is partial materialization, that is, to materialize only some of the possible cuboids that can be generated.
Partial materialization: Selected computation of cuboids There are three choices for data cube materialization:

1. Pre-compute only the base cuboid and none of the remaining "non-base" cuboids (no materialization),
2. Pre-compute all of the cuboids (full materialization), and
3. Selectively compute a proper subset of the whole set of possible cuboids (partial materialization).

The first choice leads to computing expensive multidimensional aggregates on the y, which could be slow. The second choice may require huge amounts of memory space in order to store all of the pre-computed cuboids. The third choice presents an interesting trade-off between storage space and response time.

The partial materialization of cuboids should consider three factors:

1. Identify the subset of cuboids to materialize,
2. Exploit the materialized cuboids during query processing, and
3. Efficiently update the materialized cuboids during load and refresh.

The selection of the subset of cuboids to materialize should take into account the queries in the workload, their frequencies, and their accessing costs. In addition, it should consider workload characteristics, the cost for incremental updates, and the total storage requirements. The selection must also consider the broad context of physical database design, such as the generation and selection of indices. Several OLAP products have adopted heuristic approaches for cuboid selection.

A popular approach is to materialize the set of cuboids having relatively simple structure. Even with this restriction, there are often still a large number of possible choices. Under a simplified assumption, a greedy algorithm has been proposed and has shown good performance.

A multidimensional data model

Data warehouses and OLAP tools are based on a multidimensional data model. This model views data in the form of a data cube. In this section, you will learn how data cubes model n-dimensional data. You will also learn about concept hierarchies and how they can be used in basic OLAP operations to allow interactive mining at multiple levels of abstraction.

From tables and spreadsheets to data cubes

What is a data cube? A data cube allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.
In general terms, dimensions are the perspectives or entities with respect to which an organization wants to keep records. For example, AllElectronics may create a sales data warehouse in order to keep records of the store's sales with respect to the dimensions time, item, branch, and location. These dimensions allow 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 associated with it, called a dimension table, which further describes the dimension. For example, a dimension table for item may contain the attributes item name, brand, and type. Dimension tables can be specified by users or experts, or automatically generated and adjusted based on data distributions.

A multidimensional data model is typically organized around a central theme, like sales, for instance. This theme is represented by a fact table. Facts are numerical measures. Think of them as the quantities by which we want to analyze relationships between dimensions. Examples of facts for a sales data warehouse include dollars sold (sales amount in dollars), units sold (number of units sold), and amount budgeted. The fact table contains the names of the facts, or measures, as well as keys to each of the related dimension tables. You will soon get a clearer picture of how this works when we later look at multidimensional schemas.

Although we usually think of cubes as 3-D geometric structures, in data warehousing the data cube is n-dimensional. To gain a better understanding of data cubes and the multidimensional data model, let’s start by looking at a simple 2-D data cube which is, in fact, a table or spreadsheet for sales data from AllElectronics. In particular, we will look at the AllElectronics sales data for items sold per quarter in the city of Vancouver. These data are shown in Table 2.2. In this 2-D representation, the sales for Vancouver are shown with respect to the time dimension (organized in quarters) and the item dimension (organized according to the types of items sold). The fact, or measure displayed is dollars sold.

<table>
<thead>
<tr>
<th>time (quarter)</th>
<th>Sales for all locations in Vancouver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>item (type)</td>
</tr>
<tr>
<td></td>
<td>home</td>
</tr>
<tr>
<td></td>
<td>computer</td>
</tr>
<tr>
<td></td>
<td>phone</td>
</tr>
<tr>
<td></td>
<td>security</td>
</tr>
<tr>
<td></td>
<td>entertainment</td>
</tr>
<tr>
<td>Q1</td>
<td>605K</td>
</tr>
<tr>
<td>Q2</td>
<td>680K</td>
</tr>
<tr>
<td>Q3</td>
<td>812K</td>
</tr>
<tr>
<td>Q4</td>
<td>927K</td>
</tr>
<tr>
<td></td>
<td>825K</td>
</tr>
<tr>
<td></td>
<td>952K</td>
</tr>
<tr>
<td></td>
<td>1023K</td>
</tr>
<tr>
<td></td>
<td>14K</td>
</tr>
<tr>
<td></td>
<td>31K</td>
</tr>
<tr>
<td></td>
<td>30K</td>
</tr>
<tr>
<td></td>
<td>38K</td>
</tr>
<tr>
<td></td>
<td>400K</td>
</tr>
<tr>
<td></td>
<td>512K</td>
</tr>
<tr>
<td></td>
<td>501K</td>
</tr>
<tr>
<td></td>
<td>580K</td>
</tr>
</tbody>
</table>

Table: A 2-D view of sales data for AllElectronics according to the dimensions time and item, where the sales are from branches located in the city of Vancouver. The measure displayed is dollars sold.
Table 2.3: A 3-D view of sales data for AllElectronics, according to the dimensions time, item, and location. The measure displayed is dollars sold.

Now, suppose that we would like to view the sales data with a third dimension. For instance, suppose we would like to view the data according to time, item, as well as location. These 3-D data are shown in Table 2.3. The 3-D data of Table 2.3 are represented as a series of 2-D tables. Conceptually, we may also represent the same data in the form of a 3-D data cube, as in Figure 2.1.

Suppose that we would now like to view our sales data with an additional fourth dimension, such as supplier. Viewing things in 4-D becomes tricky. However, we can think of a 4-D cube as being a series of 3-D cubes, as shown in Figure 2.2. If we continue in this way, we may display any n-D data as a series of (n − 1)-D "cubes". The data cube is a metaphor for multidimensional data storage. The actual physical storage of such data may differ from its logical representation. The important thing to remember is that data cubes are n-dimensional, and do not confine data to 3-D.

The above tables show the data at different degrees of summarization. In the data warehousing research literature, a data cube such as each of the above is referred to as a cuboid. Given a set of dimensions, we can construct a lattice of cuboids, each showing the data at a different level of summarization, or group by (i.e., summarized by a
Figure 2.1: A 3-D data cube representation of the data in Table 2.3, according to the dimensions time, item, and location. The measure displayed is dollars sold.

Figure 2.2: A 4-D data cube representation of sales data, according to the dimensions time, item, location, and supplier. The measure displayed is dollars sold.

Different subset of the dimensions). The lattice of cuboids is then referred to as a data cube. Figure 2.3 shows a lattice of cuboids forming a data cube for the dimensions time, item, location, and supplier.

The cuboid which holds the lowest level of summarization is called the base cuboid. For example, the 4-D cuboid in Figure 2.2 is the base cuboid for the given time, item, location, and supplier dimensions. Figure 2.1 is a 3-D (non-base) cuboid for time,
item, and location, summarized for all suppliers. The 0-D cuboid which holds the highest level of summarization is called the apex cuboid. In our example, this is the total sales, or dollars sold, summarized for all four dimensions. The apex cuboid is typically denoted by all.

Stars, snow akes, and fact constellations: schemas for multidimensional databases

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities or objects, and the relationships between them. Such a data model is appropriate for on-line transaction processing. Data warehouses, however, require a concise, subject-oriented schema which facilitates on-line data analysis.
Figure: Lattice of cuboids, making up a 4-D data cube for the dimensions time, item, location, and supplier. Each cuboid represents a different degree of summarization.

form of a star schema, a snowflake schema, or a fact constellation schema. Let's have a look at each of these schema types.

Star schema: The most common modeling paradigm is the star schema in which the data warehouse contains
(1) a large central table (fact table) containing the bulk of the data, with no redundancy, and (2) a set of smaller attendant tables (dimension tables), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.
An example of a star schema for AllElectronics sales is shown in Figure 2.4. Sales are considered along four dimensions, namely time, item, branch, and location. The schema contains a central fact table for sales which contains keys to each of the four dimensions, along with two measures: dollars sold and units sold.

To minimize the size of the fact table, dimension identifiers (such as time key and item key) are system-generated identifiers.

Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the location dimension table contains the attribute set: location key, street, city, province or state, country. This constraint may introduce some redundancy. For example, "Vancouver" and "Victoria" are both cities in the Canadian province of British Columbia. Entries for such cities in the location dimension table will create redundancy among the attributes province or state and country, i.e., (Vancouver, British Columbia, Canada) and (Victoria, British Columbia, Canada). Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order).

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are normalized, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake.

The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form. Such a table is easy to maintain and saves storage space because a large dimension table can become enormous when the dimensional structure is included as columns. However, this saving of space is negligible in comparison to the typical magnitude of the fact table. Furthermore, the snowflake structure can reduce the effectiveness of browsing since more joins will be needed to execute a query. Consequently, the system performance may be adversely impacted. Hence, the snowflake schema is not as popular as the star schema in data warehouse design.
An example of a snowflake schema for AllElectronics sales is given in Figure 1. Here, the sales fact table is identical to that of the star schema. The main difference between the two schemas is in the definition of dimension tables. The single dimension table for item in the star schema is normalized in the snowflake schema, resulting in new item and supplier tables. For example, the item dimension table now contains the attributes supplier key, type, brand, item name, and item key, where supplier key is linked to the supplier dimension table, containing supplier type and supplier key information. Similarly, the single dimension table for location in the star schema can be normalized into two new tables: location and city. The city key in the new location table links to the city dimension. Notice that further normalization can be performed on province or state and country in the snowflake schema.

A compromise between the star schema and the snowflake schema is to adopt a mixed schema where only the very large dimension tables are normalized. Normalizing large dimension tables saves storage space, while keeping small dimension tables unnormalized may reduce the cost and performance degradation due to joins on multiple dimension tables. Doing both may lead to an overall performance gain. However, careful performance tuning could be required to determine which dimension tables should be normalized and split into multiple tables.
Fact constellation: Sophisticated applications may require multiple fact tables to share dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a galaxy schema or a fact constellation.

An example of a fact constellation schema is shown. This schema specifies two fact tables, sales and shipping. The sales table definition is identical to that of the star schema. The shipping table has five dimensions, or keys: time key, item key, shipper key, from location, and to location, and two measures: dollars cost and units shipped. A fact constellation schema allows dimension tables to be shared between fact tables. For example, the dimensions tables for time, item, and location, are shared between both the sales and shipping fact tables.

In data warehousing, there is a distinction between a data warehouse and a data mart. A data warehouse collects information about subjects that span the entire organization, such as customers, items, sales, assets, and personnel, and thus its scope is enterprise-wide. For data warehouses, the fact constellation schema is commonly used since it can model multiple, interrelated subjects. A data mart, on the other hand, is a department subset of the data warehouse that focuses on selected subjects, and thus its scope is department-wide. For data marts, the star or snowflake schema are popular since each is geared towards modeling single subjects.

Examples for defining star, snowflake, and fact constellation schemas

"How can I define a multidimensional schema for my data?"

Just as relational query languages like SQL can be used to specify relational
queries, a data mining query language can be used to specify data mining tasks. In particular, we examine an SQL-based data mining query language called DMQL which contains,
language primitives for defining data warehouses and data marts. Language primitives for specifying other data mining tasks, such as the mining of concept/class descriptions, associations, classifications, and so on.

define dimension #dimension name as (attribute or subdimension list)

Let's look at examples of how to define the star, snowflake and constellations schemas of to using DMQL. DMQL keywords are displayed in sans serif font.

The star schema of is defined in DMQL as follows. define
cube sales star [time, item, branch, location]:
   dollars sold = sum(sales in dollars), units sold = count(*)
define dimension time as (time key, day, day of week, month, quarter, year) define dimension item as (item key, item name, brand, type, supplier type) define dimension branch as (branch key, branch name, branch type) define dimension location as (location key, street, city, province or state, country)

The define cube statement defines a data cube called sales star, which corresponds to the central sales fact table. This command specifies the keys to the dimension tables, and the two measures, dollars sold and units sold. The data cube has four dimensions, namely time, item, branch, and location. A define dimension statement is used to define each of the dimensions.

The snowflake schema is defined in DMQL as follows.

define cube sales snowflake [time, item, branch, location]:
   dollars sold = sum(sales in dollars), units sold = count(*)
define dimension time as (time key, day, day of week, month, quarter, year) define dimension item as (item key, item name, brand, type, supplier (supplier key, supplier type)) define dimension branch as (branch key, branch name, branch type) define dimension location as (location key, street, city (city key, city, province or state, country))

This definition is similar to that of sales star, except that, here, the item and location dimensions tables are normalized. For instance, the item dimension of the sales star data cube has been normalized in the sales snowflake cube into two dimension tables, item and supplier. Note that the dimension de-nition for supplier is specified within the definition for item. Defining supplier in this way implicitly creates a supplier key in the item dimension table definition. Similarly, the location dimension of the sales star data cube has been normalized in the sales snowflake cube into two dimension tables, location and city. The dimension definition for city is specified within the definition for location. In this way, a city key is implicitly created in the location dimension table definition.
Finally, a fact constellation schema can be defined as a set of interconnected cubes. The fact constellation schema of is defined in DMQQL as follows.

de define cube sales [time, item, branch, location]:
dollars sold = sum(sales in dollars), units sold = count(*)
de define dimension time as (time key, day, day of week, month, quarter, year)
de define dimension item as (item key, item name, brand, type, supplier type)
de define dimension branch as (branch key, branch name, branch type)
de define dimension location as (location key, street, city, province or state, country)
de define cube shipping [time, item, shipper, from location, to location]:
dollars cost = sum(cost in dollars), units shipped = count(*)
de define dimension time as time in cube sales

Measures: their categorization and computation

Measures can be organized into three categories, based on the kind of aggregate functions used.

distributive: An aggregate function is distributive if it can be computed in a distributed manner as follows: Suppose the data is partitioned into n sets. The computation of the function on each partition derives one aggregate value. If the result derived by applying the function to the n aggregate values is the same as that derived by applying the function on all the data without partitioning, the function can be computed in a distributed manner. For example, \text{count()} can be computed for a data cube by first partitioning the cube into a set of subcubes, computing \text{count()} for each subcube, and then summing up the counts obtained for each subcube. Hence \text{count()} is a distributive aggregate function. For the same reason, \text{sum()}, \text{min()}, and \text{max()} are distributive aggregate functions. A measure is distributive if it is obtained by applying a distributive aggregate function.

algebraic: An aggregate function is algebraic if it can be computed by an algebraic function with M arguments (where M is a bounded integer), each of which is obtained by applying a distributive aggregate function. For example, \text{avg()} (average) can be computed by \text{sum()}/\text{count()} where both \text{sum()} and \text{count()} are distributive aggregate functions. Similarly, it can be shown that \text{min()}, \text{max()}, and \text{standard deviation()} are algebraic aggregate functions. A measure is algebraic if it is obtained by applying an algebraic aggregate function.

distributive aggregate functions. Similarly, it can be shown that \text{min()}, \text{max()}, and \text{standard deviation()} are algebraic aggregate functions. A measure is algebraic if it is obtained by applying an algebraic aggregate function.

holistic: An aggregate function is holistic if there is no constant bound on the storage size needed to describe a subaggregate. That is, there does not exist an algebraic function with M arguments (where M is a constant) that characterizes the computation. Common examples of holistic functions include \text{median()}, \text{mode()} (i.e., the most frequently occurring item(s)), and \text{rank()}. A measure is holistic if it is obtained by applying a holistic aggregate function.
Once the selected cuboids have been materialized, it is important to take advantage of them during query processing. This involves determining the relevant cuboid(s) from among the candidate materialized cuboids, how to use available index structures on the materialized cuboids, and how to transform the OLAP operations on to the selected cuboid(s).

Finally, during load and refresh, the materialized cuboids should be updated efficiently. Parallelism and incremental update techniques for this should be explored. Multiway array aggregation in the computation of data cubes in order to ensure fast on-line analytical processing, however, we may need to pre-compute all of the cuboids for a given data cube. Cuboids may be stored on secondary storage, and accessed when necessary. Hence, it is important to explore efficient methods for computing all of the cuboids making up a data cube, that is, for full materialization.

These methods must take into consideration the limited amount of main memory available for cuboid computation, as well as the time required for such computation. To simplify matters, we may exclude the cuboids generated by climbing up existing hierarchies along each dimension. Since Relational OLAP (ROLAP) uses tuples and relational tables as its basic data structures, while the basic data structure used in multidimensional OLAP (MOLAP) is the multidimensional array, one would expect that ROLAP and MOLAP each explore very different cube computation techniques. ROLAP cube computation uses the following major optimization techniques.

Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples. Grouping is performed on some sub-aggregates as a "partial grouping step". These partial groupings may be used to speed up the computation of other sub-aggregates.

Aggregates may be computed from previously computed aggregates, rather than from the base fact tables. "How do these optimization techniques apply to MOLAP?" ROLAP uses value-based addressing, where dimension values are accessed by key-based addressing search strategies. In contrast, MOLAP uses direct array addressing, where dimension values are accessed via the position or index of their corresponding array locations. Hence, MOLAP cannot perform the value-based reordering of the first optimization technique listed above for ROLAP. Therefore, a different approach should be developed for the array-based cube construction of MOLAP, such as the following.
1. Partition the array into chunks. A chunk is a sub cube that is small enough to fit into the memory available for cube computation. Chunking is a method for dividing an n-dimensional array into small n-dimensional chunks, where each chunk is stored as an object on disk. The chunks are compressed so as to remove wasted space resulting from empty array cells (i.e., cells that do not contain any valid data). For instance, “chunkID + offset” can be used as a cell addressing mechanism to compress a sparse array structure and when searching for cells within a chunk. Such a compression technique is powerful enough to handle sparse cubes, both on disk and in memory.

2. Compute aggregates by visiting (i.e., accessing the values at) cube cells. The order in which cells are visited can be optimized so as to minimize the number of times that each cell must be revisited, thereby reducing memory access and storage costs. The trick is to exploit this ordering so that partial aggregates can be computed simultaneously, and any unnecessary revisiting of cells is avoided. Since this chunking technique involves “overlapping” some of the aggregation computations, it is referred to as multi-way array aggregation in data cube computation.

Summary

· A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data organized in support of management decision making.
· A multidimensional data model is typically used for the design of corporate data warehouses and departmental data marts.
· Concept hierarchies organize the values of attributes or dimensions into gradual levels of abstraction. They are useful in mining at multiple levels of abstraction.

On-line analytical processing (OLAP) can be performed in data warehouses/marts using the multidimensional data model. Typical OLAP operations include roll-up, drill-(down, cross, through), slice-and-dice, pivot (rotate), as well as statistical operations such as ranking, computing moving averages and growth rates, etc. OLAP operations can be implemented efficiently using the data cube structure.

· Data warehouses often adopt three-tier architecture. The bottom tier is a warehouse database server, which is typically a relational database system.
· The middle tier is an OLAP server, and the top tier is a client, containing query and reporting tools. OLAP servers may use Relational OLAP (ROLAP), or Multidimensional OLAP (MOLAP), or Hybrid OLAP (HOLAP).
PART A(2MARKS):

1. When can we say the association rules are interesting?

Association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Users or domain experts can set such thresholds.

2. Explain Association rule in mathematical notations.

Let I={i1,i2, ...,im} be a set of items
Let D, the task relevant data be a set of database transactions T is a set of items
An association rule is an implication of the form A=>B where A C I, B C I, and A \cap B = \emptyset. The rule A=>B contains in the transaction set D with support s,

where s is the percentage of transaction s in D that contain AUB. The Rule A=>B has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contain B.

3. Define support and confidence in Association rule mining.

Support S is the percentage of transactions in D that contain AUB. Confidence c is the percentage of transactions in D containing A that also contain B. Support (A=>B) = P(AUB) Confidence (A=>B) = P(B/A)

4. How are association rules mined from large databases? I step: Find all frequent item sets: II step: Generate strong association rules from frequent item sets

5. Describe the different classifications of Association rule mining. Based on types of values handled in the Rule
   i. Boolean association rule
   ii. Quantitative association rule
Based on the dimensions of data involved
   i. Single dimensional association rule
   ii. Multidimensional association rule
Based on the levels of abstraction involved
   i. Multilevel association rule
   ii. Single level association rule
   Based on various extensions
   i. Correlation analysis
   ii. Mining max pattern

6. What is the purpose of Apriori Algorithm?

Apriori algorithm is an influential algorithm for mining frequent item sets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent item set properties.

7. Define anti-monotone property.

If a set cannot pass a test, all of its supersets will fail the same test as well.

8. How to generate association rules from frequent item sets?
Association rules can be generated as follows.
For each frequent item set $1$, generate all non-empty subsets of $1$. For every non-empty subset $s$ of $1$, output the rule $S \Rightarrow (1 - s)$ if $\text{Support count}(1) = \text{min_conf}$, $\text{Support count}(s)$ where $\text{min_conf}$ is the minimum confidence threshold.

9. Give few techniques to improve the efficiency of Apriori algorithm.

- Hash based technique
- Transaction Reduction
- Portioning
- Sampling
- Dynamic item counting

10. What are the things suffering the performance of Apriori candidate generation technique. Need to generate a huge number of candidate sets

Need to repeatedly scan the database and check a large set of candidates by pattern matching

11. Describe the method of generating frequent item sets without candidate generation.

Frequent-pattern growth (or FP Growth) adopts divide-and-conquer strategy.
Steps:
- Compress the database representing frequent items into a frequent pattern tree
- Divide the compressed database into a set of conditional database
- Mine each conditional database separately

12. What are multidimensional association rules?

Association rules that involve two or more dimensions or predicates Interdimension association rule: Multidimensional association rule with no repeated predicate or dimension. Hybrid-dimension association rule: Multidimensional association rule with multiple occurrences of some predicates or dimensions.

13. Define constraint-Based Association Mining.

Mining is performed under the guidance of various kinds of constraints provided by the user. The constraints include the following
Knowledge type constraints Data constraints Dimension/level constraints Interestingness constraints Rule constraints.

14. Define the concept of classification.

Two step process
- A model is built describing a predefined set of data classes or concepts.
  - The model is constructed by analyzing database tuples described by attributes. The model is used for classification.

15. What is Decision tree?

A decision tree is a flow chart like tree structures, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions. The topmost in a tree is the root node.

16. What is Attribute Selection Measure?
The information Gain measure is used to select the test attribute at each node in the decision tree. Such a measure is referred to as an attribute selection measure or a measure of the goodness of split.

17. Describe Tree pruning methods.
   □ Pre pruning
   □ Post pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outlier. Tree pruning methods address this problem of overfitting the data approaches.

18. Define Pre Pruning

A tree is pruned by halting its construction early. Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset samples.

19. Define Post Pruning.

Post pruning removes branches from a “Fully grown” tree. A tree node is pruned by removing its branches. Eg: Cost Complexity Algorithm.

PART B (10 MARKS):

1. Explain the various primitives for specifying Data mining Task.
2. Describe the various descriptive statistical measures for data mining.
3. Discuss about different types of data and functionalities.
4. Describe in detail about Interestingness of patterns.
5. Explain in detail about data mining task primitives.
6. Discuss about different Issues of data mining.
7. Explain in detail about data preprocessing. How data mining system are classified? Discuss each classification with an example.
8. How data mining system can be integrated with a data warehouse? Discuss with an example.
9. Explain data mining applications for Telecommunication industry.
UNIT-III

MINING FREQUENT PATTERNS

Frequent patterns:
patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (frequent) sequential pattern. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences.

Association Rule Mining:

- Association rule mining is a popular and well researched method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness.

Problem Definition:
The problem of association rule mining is defined as:

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of binary attributes called items.

Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions called the database.

Each transaction in $D$ has a unique transaction ID and contains a subset of the items in $I$.

A rule is defined as an implication of the form

$$X \Rightarrow Y$$

where $X, Y \subseteq I$ and $X \cap Y = \emptyset$.

The sets of items (for short itemsets) $X$ and $Y$ are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively.

Example:
To illustrate the concepts, we use a small example from the supermarket domain. The set of items is $I = \{milk, bread, butter, bear\}$ and a small database containing the items (1 codes presence and 0 absence of an item in a transaction) is shown in the table.

An example rule for the supermarket could be "butter and bread \Rightarrow milk" meaning that if butter and bread are bought, customers also buy milk.

Example database with 4 items and 5 transactions

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>milk</th>
<th>bread</th>
<th>butter</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Important concepts of Association Rule Mining:

- **Support** of an itemset is defined as the proportion of transactions in the dataset which contain the itemset. In the example database, the itemset has a support of since it occurs in 20% of all transactions (1 out of 5 transactions).

- **Confidence** of a rule is defined as

$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$$

containing butter and bread the rule is correct (100% of the times a customer buys butter and bread, milk is bought as well). Confidence can be interpreted as an estimate of the probability \( P(Y|X) \), the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS.

- **Lift** of a rule is defined as

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \times \text{supp}(Y)}$$

or the ratio of the observed support to that expected if X and Y were independent. The rule \( \{\text{milk, bread}\} \Rightarrow \{\text{butter}\} \) has a lift of

\[ \frac{0.2}{0.4 \times 0.4} = 1.25 \]

- **Conviction** of a rule is defined as

$$\text{conv}(X \Rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \Rightarrow Y)}$$

The rule \( \{\text{milk, bread}\} \Rightarrow \{\text{butter}\} \) has a conviction of

\[ \frac{1 - 0.4}{1 - 0.5} = \frac{1}{0.2} = 5 \]

and can be interpreted as the ratio of the expected frequency that X occurs without Y (that is to say, the frequency that the rule makes an incorrect prediction) if X and Y were independent divided by the observed frequency of incorrect predictions.

**Market basket analysis:**

This process analyzes customer buying habits by finding associations between the different items that customers place in their shopping baskets. The discovery of such association scan help retailers develop marketing strategies by gaining insight into which items are frequently...
purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket. Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

Example:

If customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help increase the sales of both items. In an alternative strategy, placing hardware and software at opposite ends of the store may entice customers who purchase such items to pick up other items along the way. For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading toward the software display to purchase antivirus software and may decide to purchase a home security system as well. Market basket analysis can also help retailers plan which items to put on sale at reduced prices. If customers tend to purchase computers and printers together, then having a sale on printers may encourage the sale of printers as well as computers.

Frequent Pattern Mining:
Frequent pattern mining can be classified in various ways, based on the following criteria:

1. Based on the completeness of patterns to be mined:
   - We can mine the complete set of frequent item sets, the closed frequent item sets, and the maximal frequent item sets, given a minimum support threshold.
   - We can also mine constrained frequent item sets, approximate frequent item sets, near-match frequent item sets, to p-k frequent item sets and so on.

2. Based on the levels of abstraction involved in the rule set:

Some methods for association rule mining can find rules at differing levels of abstraction.
For example, suppose that a set of association rules mined includes the following rules where X is a variable representing a customer:

\[ \text{buys}(X, \text{―computer‖}) \Rightarrow \text{buys}(X, \text{―HP printer‖}) \]  \hspace{1cm} (1)

\[ \text{buys}(X, \text{―laptop computer‖}) \Rightarrow \text{buys}(X, \text{―HP printer‖}) \]  \hspace{1cm} (2)

In rule (1) and (2), the items bought are referenced at different levels of abstraction (e.g., ―computer‖ is a higher-level abstraction of ―laptop computer‖).

3. Based on the number of data dimensions involved in the rule:
   • If the items or attributes in an association rule reference only one dimension, then it is a single-dimensional association rule.
     \[ \text{buys}(X, \text{―computer‖}) \Rightarrow \text{buys}(X, \text{―antivirus software‖}) \]
   • If a rule references two or more dimensions, such as the dimensions age, income, and buys, then it is a multidimensional association rule. The following rule is an example of a multidimensional rule:
     \[ \text{age}(X, \text{―30,31…39‖}) \land \text{income}(X, \text{―42K,…48K‖}) \Rightarrow \text{buys}(X, \text{―high resolution TV‖}) \]

4. Based on the types of values handled in the rule:
   • If a rule involves associations between the presence or absence of items, it is a Boolean association rule.
   • If a rule describes associations between quantitative items or attributes, then it is a quantitative association rule.

5. Based on the kinds of rules to be mined:
   • Frequent pattern analysis can generate various kinds of rules and other interesting relationships.
   • Association rule mining can generate a large number of rules, many of which are redundant or do not indicate a correlation relationship among item sets.
   • The discovered associations can be further analyzed to uncover statistical correlations, leading to correlation rules.

6. Based on the kinds of patterns to be mined:
   • Many kinds of frequent patterns can be mined from different kinds of data sets. Sequential pattern mining searches for frequent subsequences in a sequence data set, where a sequence records an ordering of events.
   • For example, with sequential pattern mining, we can study the order in which items are frequently purchased. For instance, customers may tend to first buy a PC, followed by a digital camera, and then a memory card.
data set. Single items are the simplest form of structure.
Each element of an item set may contain a subsequence, a subtree, and so on.
Therefore, structured pattern mining can be considered as the most general form of frequent pattern mining.

Efficient Frequent Itemset Mining Methods:

Finding Frequent Itemsets Using Candidate Generation: The Apriori Algorithm

- Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules.
- The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties.
- Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k+1)-itemsets.
- First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted L1. Next, L1 is used to find L2, the set of frequent 2-itemsets, which is used to find L3, and so on, until no more frequent k-itemsets can be found.
- The finding of each Lk requires one full scan of the database.
- A two-step process is followed in Apriori consisting of join and prune action.

Example:

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T10</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T20</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T30</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T40</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T50</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T60</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T70</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T80</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T90</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>

There are nine transactions in this database, that is, \(|D| = 9\).
Steps:

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C1. The algorithm simply scans all of the transactions in order to count the number of occurrences of each item.

2. Suppose that the minimum support count required is 2, that is, min sup = 2. The set of frequent 1-itemsets, L1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C1 satisfy minimum support.

3. To discover the set of frequent 2-itemsets, L2, the algorithm uses the join L1 on L1 to generate a candidate set of 2-itemsets, C2. No candidates are removed from C2 during the prune step because each subset of the candidates is also frequent.

4. Next, the transactions in D are scanned and the support count of each candidate itemset in C2 is accumulated.

5. The set of frequent 2-itemsets, L2, is then determined, consisting of those candidate 2-itemsets in C2 having minimum support.

6. The generation of the set of candidate 3-itemsets, C3, from the join step, we first get C3 = L2 x L2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent.

7. The transactions in D are scanned in order to determine L3, consisting of those candidate 3-itemsets in C3 having minimum support.

8. The algorithm uses L3 x L3 to generate a candidate set of 4-itemsets, C4.
Generating Association Rules from Frequent Itemsets:

Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them.
The conditional probability is expressed in terms of itemset support count, where \( support \_ count(A \cup B) \) is the number of transactions containing the itemsets \( A \cup B \), and \( support \_ count(A) \) is the number of transactions containing the itemset \( A \). Based on this equation, association rules can be generated as follows:

- For each frequent itemset \( I \), generate all nonempty subsets of \( I \).
- For every nonempty subset \( s \) of \( I \), output the rule \( "s \Rightarrow (I - s)" \) if \( \frac{support \_ count(I)}{support \_ count(s)} \geq min \_ conf \), where \( min \_ conf \) is the minimum confidence threshold.

Example:

Generating association rules. Let’s try an example based on the transactional data for AllElectronics shown in Table 5.1. Suppose the data contain the frequent itemset \( I = \{11, 12, 15\} \). What are the association rules that can be generated from \( I \)? The nonempty subsets of \( I \) are \( \{11, 12\}, \{11, 15\}, \{12, 15\}, \{11\}, \{12\}, \) and \( \{15\} \). The resulting association rules are as shown below, each listed with its confidence:

- \( 11 \land 12 \Rightarrow 15, \) confidence = 2/4 = 50%
- \( 11 \land 15 \Rightarrow 12, \) confidence = 2/2 = 100%
- \( 12 \land 15 \Rightarrow 11, \) confidence = 2/2 = 100%
- \( 11 \Rightarrow 12 \land 15, \) confidence = 2/6 = 33%
- \( 12 \Rightarrow 11 \land 15, \) confidence = 2/7 = 29%
- \( 15 \Rightarrow 11 \land 12, \) confidence = 2/2 = 100%

**FP-Growth Method: Mining Frequent Itemsets without Candidate Generation**

As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain.

An interesting method in this attempt is called frequent-pattern growth, or simply FP-growth, which adopts a divide-and-conquer strategy as follows. First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases (a special kind of projected database), each associated with one frequent item or —pattern fragment, and mines each such database separately. You’ll see how it works with the following example.

FP-growth (finding frequent itemsets without candidate generation). We re-examine the mining of transaction database, \( D \), of Table 5.1 in Example 5.3 using the frequent pattern growth approach.
The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or list is denoted \( L \). Thus, we have \( L = \{\{I_2: 7\}, \{I_1: 6\}, \{I_3: 6\}, \{I_4: 2\}, \{I_5: 2\}\} \).

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with “null.” Scan database \( D \) a second time. The items in each transaction are processed in \( L \) order (i.e., sorted according to descending support count), and a branch is created for each transaction. For example, the scan of the first transaction, “T100: I_1, I_2, I_5,” which contains three items (I_2, I_1, I_5 in \( L \) order), leads to the construction of the first branch of the tree with three nodes, \( \langle I_2: 1 \rangle \), \( \langle I_1: 1 \rangle \), and \( \langle I_5: 1 \rangle \), where I_2 is linked as a child of the root, I_1 is linked to I_2, and I_5 is linked to I_1. The second transaction, T200, contains the items I_2 and I_4 in \( L \) order, which would result in a branch where I_2 is linked to the root and I_4 is linked to I_2. However, this branch would share a common prefix, I_2, with the existing path for T100. Therefore, we instead increment the count of the I_2 node by 1, and create a new node, \( \langle I_4: 1 \rangle \), which is linked as a child of \( \langle I_2: 2 \rangle \). In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

![FP-tree diagram](image)

Mining the FP-tree by creating conditional (sub-)pattern bases.

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional Pattern Base</th>
<th>Conditional FP-tree</th>
<th>Frequent Patterns Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_5</td>
<td>{{I_2, I_1: 1}, {I_2, I_1, I_3: 1}}</td>
<td>(\langle I_2: 2, I_1: 2\rangle)</td>
<td>{I_2, I_5: 2}, {I_1, I_5: 2}, {I_2, I_1, I_5: 2})</td>
</tr>
<tr>
<td>I_4</td>
<td>{{I_2, I_1: 1}, {I_2: 1}}</td>
<td>(\langle I_2: 2\rangle)</td>
<td>{I_2, I_4: 2})</td>
</tr>
<tr>
<td>I_3</td>
<td>{{I_2, I_1: 2}, {I_2: 2}, {I_1: 2}}</td>
<td>(\langle I_2: 4, I_1: 2, I_2: 1\rangle)</td>
<td>{I_2, I_3: 4}, {I_1, I_3: 4}, {I_2, I_1, I_3: 2})</td>
</tr>
<tr>
<td>I_1</td>
<td>{{I_2: 4}}</td>
<td>(\langle I_2: 4\rangle)</td>
<td>{I_2, I_1: 4})</td>
</tr>
</tbody>
</table>

Mining of the FP-tree is summarized in Table 5.2 and detailed as follows. We first consider I_5, which is the last item in \( L \), rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I_5 occurs in two branches of the FP-tree of Figure 5.7. (The occurrences of I_5 can easily be found by following its chain of node-links.) The paths formed by these branches are \( \langle I_2, I_1, I_5: 1 \rangle \) and \( \langle I_2, I_1, I_3, I_5: 1 \rangle \). Therefore, considering I_5 as a suffix, its corresponding two prefix paths are \( \langle I_2, I_1: 1 \rangle \) and \( \langle I_2, I_1, I_3: 1 \rangle \), which form its conditional pattern base. Its conditional FP-tree contains only a single path, \( \langle I_2: 2, I_1: 2 \rangle \); I_3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: \{I_2, I_5: 2\}, \{I_1, I_5: 2\}, \{I_2, I_1, I_5: 2\}\. 

For I_4, its two prefix paths form the conditional pattern base, \{\{I_2, I_1: 1\}, \{I_2: 1\}\}, which generates a single-node conditional FP-tree, \(\langle I_2: 2\rangle\), and derives one frequent
Mining Multilevel Association Rules:

- For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data at those levels.
- Strong associations discovered at high levels of abstraction may represent common sense knowledge.
- Therefore, data mining systems should provide capabilities for mining association rules at multiple levels of abstraction, with sufficient flexibility for easy traversal among different abstraction spaces.
- Association rules generated from mining data at multiple levels of abstraction are called multiple-level or multilevel association rules.
- Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework.
- In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level 1 and working downward in the hierarchy toward the more specific concept levels, until no more frequent itemsets can be found. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Data can be generalized by replacing low-level concepts within the data by their higher-level concepts, or ancestors, from a concept hierarchy.
The concept hierarchy has five levels, respectively referred to as levels 0 to 4, starting with level 0 at the root node for all.

- Level 1 includes computer, software, printer & camera, and computer accessory.
- Level 2 includes laptop computer, desktop computer, office software, antivirus software.
- Level 3 includes IBM desktop computer, Microsoft office software, and so on.
- Level 4 is the most specific abstraction level of this hierarchy.

Approaches For Mining Multilevel Association Rules:

1. Uniform Minimum Support:
   - The same minimum support threshold is used when mining at each level of abstraction. When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold.

   The uniform support approach, however, has some difficulties. It is unlikely that items at lower levels of abstraction will occur as frequently as those at higher levels of abstraction.
   - If the minimum support threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels.
2. Reduced Minimum Support:

- Each level of abstraction has its own minimum support threshold.
- The deeper the level of abstraction, the smaller the corresponding threshold is.
- For example, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, “computer,” “laptop computer,” and “desktop computer” are all considered frequent.

3. Group-Based Minimum Support:

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

Mining Multidimensional Association Rules from Relational Databases and Data Warehouses:

- Single dimensional or intradimensional association rule contains a single distinct predicate (e.g., buys) with multiple occurrences. i.e., the predicate occurs more than once within the rule.

\[ \text{buys}(X, \text{—digital camera}) \Rightarrow \text{buys}(X, \text{—HP printer}) \]

- Association rules that involve two or more dimensions or predicates can be referred to as multidimensional association rules.

\[ \text{age}(X, \text{“20…29”})^\mathcal{O} \text{occup}^\mathcal{O} \text{ation}(X, \text{“student”}) \Rightarrow \text{buys}(X, \text{“laptop”}) \]
Above Rule contains three predicates (age, occupation, and buys), each of which occurs only once in the rule. Hence, we say that it has no repeated predicates.

Multidimensional association rules with no repeated predicates are called interdimensional association rules.

We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called hybrid-dimensional association rules. An example of such a rule is the following, where the predicate buys is repeated:

\[ \text{age}(X, -20...29\text{\textendash})^\text{buys}(X, -\text{laptop}\text{\textendash}) \Rightarrow \text{buys}(X, -\text{HP printer}\text{\textendash}) \]

Mining Quantitative Association Rules:

- Quantitative association rules are multidimensional association rules in which the numeric attributes are dynamically discretized during the mining process so as to satisfy some mining criteria, such as maximizing the confidence or compactness of the rules mined.

- In this section, we focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side of the rule and one categorical attribute on the right-hand side of the rule. That is

\[ \text{Aquan1} \land \text{Aquan2} \Rightarrow \text{Acat} \]

where Aquan1 and Aquan2 are tests on quantitative attribute interval

Acat tests a categorical attribute from the task-relevant data.

- Such rules have been referred to as two-dimensional quantitative association rules, because they contain two quantitative dimensions.

- For instance, suppose you are curious about the association relationship between pairs of quantitative attributes, like customer age and income, and the type of television (such as high-definition TV, i.e., HDTV) that customers like to buy. An example of such a 2-D quantitative association rule is

\[ \text{age}(X, -30...39\text{\textendash})^\text{income}(X, -42K...48K\text{\textendash}) \Rightarrow \text{buys}(X, -\text{HDTV}\text{\textendash}) \]
From Association Mining to Correlation Analysis:

• A correlation measure can be used to augment the support-confidence framework for association rules. This leads to correlation rules of the form A => B [support, confidence, correlation]

• That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets A and B. There are many different correlation measures from which to choose. In this section, we study various correlation measures to determine which would be good for mining large data sets.

• Lift is a simple correlation measure that is given as follows. The occurrence of itemset A is independent of the occurrence of itemset B if \( P(A \cup B) = P(A)P(B) \); otherwise, itemsets A and B are dependent and correlated as events. This definition can easily be extended to more than two item sets.

The lift between the occurrence of A and B can be measured by computing

\[
\text{lift}(A, B) = \frac{P(A \cup B)}{P(A)P(B)}.
\]

• If the \( \text{lift}(A, B) \) is less than 1, then the occurrence of A is negatively correlated with the occurrence of B.

• If the resulting value is greater than 1, then A and B are positively correlated, meaning that the occurrence of one implies the occurrence of the other.

• If the resulting value is equal to 1, then A and B are independent and there is no correlation between them.

From Association Mining to Correlation Analysis:

Most association rule mining algorithms employ a support-confidence framework. Often, many interesting rules can be found using low support thresholds. Although minimum support and confidence thresholds help weed out or exclude the exploration of a good number of uninteresting rules, many rules so generated are still not interesting to the users. Unfortunately, this is especially true when mining at low support thresholds or mining for long patterns. This has been one of the major bottlenecks for successful application of association rule mining.

1) Strong Rules Are Not Necessarily Interesting: An Example

Whether or not a rule is interesting can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics—behind the data, can be used as one step toward the goal of weeding out uninteresting rules from presentation to the user. The support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be used to augment the support-confidence framework for association rules. This leads to correlation rules of the form

\[ A \Rightarrow B \text{ [support, confidence, correlation]} \].
Constraint-Based Association Mining

A data mining process may uncover thousands of rules from a given set of data, most of which end up being unrelated or uninteresting to the users. Often, users have a good sense of which —direction of mining may lead to interesting patterns and the —form of the patterns or rules they would like to find. Thus, a good heuristic is to have the users specify such intuition or expectations as constraints to confine the search space. This strategy is known as constraint-based mining.

The constraints can include the following:

1) Metarule-Guided Mining of Association Rules

“How are metarules useful?” Meta rules allow users to specify the syntactic form of rules that they are interested in mining. The rule forms can be used as constraints to help improve the efficiency of the mining process. Meta rules may be based on the analyst’s experience, expectations, or intuition regarding the data or may be automatically generated based on the database schema.

Metarule-guided mining: Suppose that as a market analyst for AllElectronics, you have access to the data describing customers (such as customer age, address, and credit rating) as well as the list of customer transactions. You are interested in finding associations between customer traits and the items that customers buy. However, rather than finding all of the association rules reflecting these relationships, you are particularly interested only in determining which pairs of customer traits promote the sale of office software. A metarule can be used to specify this information describing the form of rules you are interested in finding. An example of such a metarule is

\[ P_1(X, Y) \land P_2(X, W) \Rightarrow buys(X, “office software”), \]

where P1 and P2 are predicate variables that are instantiated to attributes from the given database during the mining process. X is a variable representing a customer, and Y and W take on values of the attributes assigned to P1 and P2, respectively. Typically, a user will specify a list of attributes to be considered for instantiation with P1 and P2. Otherwise, a default set may be used.

2) Constraint Pushing: Mining Guided by Rule Constraints

Rule constraints specify expected set/subset relationships of the variables in the mined rules, constant initiation of variables, and aggregate functions. Users typically employ their knowledge of the application or data to specify rule constraints for the mining task. These rule constraints may be used together with, or as an alternative to, metarule-guided mining. In this section, we examine rule constraints as to how they can be used to make the mining process more efficient.

Let’s study an example where rule constraints are used to mine hybrid-dimensional association rules. Our association mining query is to “Find the sales of which cheap items (where the sum of the prices is less than $100) may promote the sales of which expensive items (where the minimum price is $500) of the same group for Chicagocustomers in 2004.” This can be expressed in the DMQL data mining query language as follows,
Classification and Prediction:

Data analysis task is classification, where a model or classifier is constructed to predict 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. Suppose that the marketing manager would like 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 categorical label. This model is a predictor.

Data classification is a two-step process, as shown for the loan application data of Figure 6.1. (The data are simplified for illustrative purposes.) 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 = (x1, x2, ..., xn), depicting n measurements made on the tuple from n database attributes, respectively, A1, A2, ..., An. Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute. 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),
Issues Regarding Classification and Prediction:

1. Preparing the Data for Classification and Prediction:

The following preprocessing steps may be applied to the data to help improve the accuracy, efficiency, and scalability of the classification or prediction process.

(i) Data cleaning:
- This refers to the preprocessing of data in order to remove or reduce noise (by applying smoothing techniques) and the treatment of missing values (e.g., by replacing a missing value). Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

(ii) Relevance analysis:
- Many of the attributes in the data may be redundant.
- Correlation analysis can be used to identify whether any two given attributes are statistically related.
- For example, a strong correlation between attributes A1 and A2 would suggest that one of the two could be removed from further analysis.
- A database may also contain irrelevant attributes. Attribute subset selection can be used in these cases to find a reduced set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes.
- Hence, relevance analysis, in the form of correlation analysis and attribute subset selection, can be used to detect attributes that do not contribute to the classification or prediction task.
- Such analysis can help improve classification efficiency and scalability.

(iii) Data Transformation And Reduction
- The data may be transformed by normalization
  Normalization involves scaling all values for a given attribute so that they fall within a small specified range, such as -1 to +1 or 0 to 1.
- The data can also be transformed by generalizing it to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous valued attributes.
Comparing Classification and Prediction Methods:

Accuracy:

- The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples with out class label information).
- The accuracy of a predictor refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data.

Speed:

This refers to the computational costs involved in generating and using the given classifier or predictor.

Robustness:

This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.

Scalability:

This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.

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.

Classification by Decision Tree Induction:

- A decision tree is a flowchart-like tree structure, where
each internal node denotes a test on an attribute. Each branch represents an outcome of the test. Each leaf node holds a class label.
The topmost node in a tree is the root node.
The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery.

- Decision trees can handle high dimensional data.
- In general, decision tree classifiers have good accuracy.
- Decision tree induction algorithms have been used for classification in many application areas, such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology.

Algorithm For Decision Tree Induction:

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 splittingCriterion;
8. if splitting_attribute is discrete-valued and
   multiway splits allowed then // not restricted to binary trees
   attribute_list ← attribute_list - splitting_attribute; // remove splitting_attribute
9. for each outcome $j$ of splitting criterion
   // partition the tuples and grow subtrees for each partition
   let $D_j$ be the set of data tuples in $D$ satisfying outcome $j$; // a partition
10. if $D_j$ is empty then
11. attach a leaf labeled with the majority class in $D$ to node $N$;
12. else attach the node returned by Generate_decision_tree($D_j$, attribute_list) to node $N$;
13. endfor
14. return $N$;
The algorithm is called with three parameters:

- Data partition
- Attribute list

Attribute selection method

- 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.
- The tree starts as a single node, N, representing the training tuples in D.
- If the tuples in D are all of the same class, then node N becomes a leaf and is labeled with that class.
- All of the 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.

There are three possible scenarios. Let A be the splitting attribute. A has v distinct values, \{a_1, a_2, ..., a_v\}, based on the training data.

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.

2. A is continuous-valued:
   - In this case, the test at node N has two possible outcomes, corresponding to the conditions A <= split point and A > split point, respectively.

3. A is discrete-valued and a binary tree must be produced:
   - If A is discrete-valued and a binary tree must be produced, then the test is of the form A <= SA, where SA is the Splitting of A.
If A is discrete valued (b) or A is continuous valued (c) or A is discrete-valued and a binary tree must be produced:

(b) 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. Information gain ID3 uses information gain as its attribute selection measure. The attribute with the highest information gain is chosen as the splitting attribute for node N.

The expected information needed to classify a tuple in D is given by

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

where $p_i$ is the probability that an arbitrary tuple in D belongs to class $C_i$ Info(D) is also known as the entropy of D.

How much more information would we still need (after the partitioning) in order to arrive at an exact classification? This amount is measured by

$$\text{Info}_A(D) = \sum_{j=1}^{x} \frac{|D_j|}{|D|} \times \text{Info}(D_j),$$

The term $|D_j|$ acts as the weight of the jth partition

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is, $\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$:
Induction of a decision tree using information gain. Table 6.1 presents a training set, D, of class-labeled tuples. The class label attribute, buys computer, has two distinct values (namely, yes, no); therefore, there are two distinct classes (that is, \( m = 2 \)). Let class \( C_1 \) correspond to yes and class \( C_2 \) correspond to no. There are nine tuples of class yes and five tuples of class no. A (root) node \( N \) is created for the tuples in \( D \) to compute the expected information needed to classify a tuple in \( D \):

the expected information needed to classify a tuple in \( D \) if the tuples are partitioned according to age is

\[
\text{Info}(D) = -\frac{9}{14} \log_2 \left( \frac{9}{14} \right) - \frac{5}{14} \log_2 \left( \frac{5}{14} \right) = 0.940 \text{ bits}
\]

Gain ratio

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. The attribute with the maximum gain ratio is selected as the splitting attribute.

Example 6.2 Computation of gain ratio for the attribute income. A test on income splits the data of Table 6.1 into three partitions, namely low, medium, and high, containing four, six, and four tuples, respectively. To compute the gain ratio of income, we first use Equation (6.5) to obtain

\[
\begin{align*}
\text{Gain}(age) & = \text{Info}(D) - \text{Info}_\text{age}(D) \\
& = 0.940 - 0.694 = 0.246 \text{ bits}.
\end{align*}
\]
From Example 6.1, we have Gain(income) = 0.029. Therefore, GainRatio(income) = 0.029/0.926 = 0.031.

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

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

where \( p_i \) is the probability that a tuple in D belongs to class \( C_i \) and is estimated by \( \frac{j_{C_i,D}}{j_D} \). The sum is computed over \( m \) classes. If income has three possible values, namely flow, medium, highg, then the possible subsets are flow, medium, highg, flow, mediumg, flow, highg, medium, flow, mediumg, highg, flowg, mediumg, and {} if a binary split on A partitions D into D1 and D2, the Gini index of D given that partitioning is

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

\[ \Delta Gini(A) = Gini(D) - Gini_A(D). \]

Induction of a decision tree using gini index. Let D be the training data of Table 6.1 where there are nine tuples belonging to the class buys computer = yes and the remaining five tuples belong to the class buys computer = no. A (root) node N is created for the tuples in D attribute income and consider each of the possible splitting subsets. Consider the subset flow, mediumg. This would result in 10 tuples in partition D1 satisfying the condition “income \( \leq \) flow, medium

\[ Gini_{income \in \{low,medium\}}(D) \]
\[ = \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \]
\[ = \frac{10}{14} \left( 1 - \left( \frac{6}{10} \right)^2 - \left( \frac{4}{10} \right)^2 \right) + \frac{4}{14} \left( 1 - \left( \frac{1}{4} \right)^2 - \left( \frac{3}{4} \right)^2 \right) \]
\[ = 0.450 \]
\[ = Gini_{income \in \{high\}}(D). \]

Similarly, the Gini index values for splits on the remaining subsets are: 0.315 (for the subsets flow, highg and mediumg) and 0.300 (for the subsets medium, highg and flowg). Therefore, the best binary split for attribute income is on medium, highg (or flowg) because it minimizes the gini index. Evaluating the attribute, we obtain fyouth, seniorg (or fmiddle agedg) as the best split for age with a Gini index of 0.375.

Bayesian Classification:

- Bayesian classifiers are statistical classifiers.
- They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class.
- Bayesian classification is based on Bayes’ theorem.
Bayes’ Theorem:

- Let X be a data tuple. In Bayesian terms, X is considered —evidence— and it is described by measurements made on a set of n attributes.

- For classification problems, we want to determine \( P(H|X) \), the probability that the hypothesis H holds given the —evidence— or observed data tuple X.

- \( P(H|X) \) is the posterior probability, or a posteriori probability, of H conditioned on X.

- Bayes’ theorem is useful in that it provides a way of calculating the posterior probability, \( P(H|X) \), from \( P(H) \), \( P(X|H) \), and \( P(X) \).

Naïve Bayesian Classification:

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, \( X = (x_1, x_2, ..., x_n) \), depicting n measurements made on the tuple from n attributes, respectively, \( A_1, A_2, ..., A_n \).

2. Suppose that there are m classes, \( C_1, C_2, ..., C_m \). Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X.

That is, the naïve Bayesian classifier predicts that tuple X belongs to the class \( C_i \) if and only if

\[
P(C_i|X) > P(C_j|X) \quad \text{for } 1 \leq j \leq m, j \neq i.
\]

Thus we maximize \( P(C_i|X) \). The class \( C_i \) for which \( P(C_i|X) \) is maximized is called the maximum posteriori hypothesis. By Bayes’ theorem

\[
P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}.
\]

3. As \( P(X) \) is constant for all classes, only \( P(X|C_i)P(C_i) \) need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, \( P(C_1) = P(C_2) = ... = P(C_m) \), and we would therefore maximize \( P(X|C_i) \). Otherwise, we maximize \( P(X|C_i)P(C_i) \).

4. Given data sets with many attributes, it would be extremely computationally expensive to compute \( P(X|C_i) \). In order to reduce computation in evaluating \( P(X|C_i) \), the
naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple. Thus,

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

\[
= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i).
\]

We can easily estimate the probabilities \(P(x_1|C_i), P(x_2|C_i), \ldots, P(x_n|C_i)\) from the training tuples. For each attribute, we look at whether the attribute is categorical or continuous-valued. For instance, to compute \(P(X|C_i)\), we consider the following:

If \(A_k\) is categorical, then \(P(x_k|C_i)\) is the number of tuples of class \(C_i\) in \(D\) having the value \(x_k\) for \(A_k\), divided by \(|C_i|\) the number of tuples of class \(C_i\) in \(D\).

If \(A_k\) is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward.

A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean \(\mu\) and standard deviation \(\sigma\), defined by

\[
g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},
\]

\[
P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}).
\]

5. In order to predict the class label of \(X\), \(P(X|C_i)P(C_i)\) is evaluated for each class \(C_i\). The classifier predicts that the class label of tuple \(X\) is the class \(C_i\) if and only if

\[
P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \quad \text{for} \quad 1 \leq j \leq m, j \neq i.
\]

The training data are in Table 6.1. The data tuples are described by the attributes age, income, student, and credit rating. The class label attribute, buys computer, has two distinct values (namely, yes, no). Let \(C_1\) correspond to the class buys computer = yes and \(C_2\) correspond to buys computer = no. The tuple we wish to classify is \(X = (\text{age} = \text{youth}, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit rating} = \text{fair})\). We need to maximize \(P(X|C_i)P(C_i)\), for \(i = 1, 2\). The prior probability of each class, can be computed based on the training tuples:

\[
P(\text{buys computer} = \text{yes}) = \frac{9}{14} = 0.643
\]

\[
P(\text{buys computer} = \text{no}) = \frac{5}{14} = 0.357
\]

To compute \(P(X|C_i)\), for \(i = 1, 2\), we compute the following conditional probabilities:

\[
P(\text{age} = \text{youth} | \text{buys computer} = \text{yes}) = \frac{2}{9} = 0.222
\]

\[
P(\text{age} = \text{youth} | \text{buys computer} = \text{no}) = \frac{3}{5} = 0.600
\]

\[
P(\text{income} = \text{medium} | \text{buys computer} = \text{yes}) = \frac{4}{9} = 0.444
\]

\[
P(\text{income} = \text{medium} | \text{buys computer} = \text{no}) = \frac{2}{5} = 0.400
\]

\[
P(\text{student} = \text{yes} | \text{buys computer} = \text{yes}) = \frac{6}{9} = 0.667
\]

\[
P(\text{student} = \text{yes} | \text{buys computer} = \text{no}) = \frac{1}{5} = 0.200
\]

\[
P(\text{credit rating} = \text{fair} | \text{buys computer} = \text{yes}) = \frac{6}{9} = 0.667
\]

\[
P(\text{credit rating} = \text{fair} | \text{buys computer} = \text{no}) = \frac{2}{5} = 0.400
\]

Using the above probabilities, we obtain:

\[
P(X|\text{buys computer} = \text{yes}) = P(\text{age} = \text{youth} | \text{buys computer} = \text{yes})
\]

\[
P(\text{income} = \text{medium} | \text{buys computer} = \text{yes})
\]
\[ P(\text{student} = \text{yes} \mid \text{buys computer} = \text{yes}) = 0.222 \]
\[ P(\text{credit rating} = \text{fair} \mid \text{buys computer} = \text{yes}) = 0.444 \]
\[ = 0.667 \times 0.667 = 0.44. \]

Similarly,
\[ P(X \mid \text{buys computer} = \text{no}) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019. \]

To find the class, \( C_i \), that maximizes \( P(X \mid C_i)P(C_i) \), we compute
\[ P(X \mid \text{buys computer} = \text{yes})P(\text{buys computer} = \text{yes}) = 0.044 \times 0.643 = 0.028 \]
\[ P(X \mid \text{buys computer} = \text{no})P(\text{buys computer} = \text{no}) = 0.019 \times 0.357 = 0.007 \]

Therefore, the naïve Bayesian classifier predicts \( \text{buys computer} = \text{yes} \) for tuple \( X \).

**Bayesian Belief Networks**

Bayesian belief networks specify joint conditional probability distributions. Trained Bayesian belief networks can be used for classification. Bayesian belief networks are also known as belief networks, Bayesian networks, and probabilistic networks. For brevity, we will refer to them as belief networks. A belief network is defined by two components—a directed acyclic graph and a set of conditional probability tables (Figure 6.11). Each node in the directed acyclic graph represents a random variable. The variables may be discrete or continuous-valued. Each arc represents a probabilistic dependence. If an arc is drawn from a node \( Y \) to a node \( Z \), then \( Y \) is a parent or immediate predecessor of \( Z \), and \( Z \) is a descendant of \( Y \).

**PositiveXRay** is independent of whether the patient has a family history of lung cancer or is a smoker, given that we know the patient has lung cancer. In other words, once we know the outcome of the variable \( \text{LungCancer} \), then the variables \( \text{FamilyHistory} \) and \( \text{Smoker} \) do not provide any additional information regarding \( \text{PositiveXRay} \). The arcs also show that the variable \( \text{LungCancer} \) is conditionally independent of \( \text{Emphysema} \), given its parents, \( \text{FamilyHistory} \) and \( \text{Smoker} \). Figure 6.11(b) shows a CPT for the variable \( \text{LungCancer} \). The conditional probability for each known value of \( \text{LungCancer} \) is given for each possible combination of values of its parents. For instance, from the upper leftmost and bottom rightmost entries, respectively, we see that
\[ P(\text{LungCancer} = \text{yes} \mid \text{FamilyHistory} = \text{yes}, \text{Smoker} = \text{yes}) = 0.8 \]
\[ P(\text{LungCancer} = \text{no} \mid \text{FamilyHistory} = \text{no}, \text{Smoker} = \text{no}) = 0.9 \]
Rule-Based classification:

Using IF-THEN Rules for classification:
An IF-THEN rule is an expression of the form
IF condition THEN conclusion.

An example is rule R1,
R1: IF age = youth AND student = yes THEN buys computer = yes.

The “IF”-part (or left-hand side) of a rule is known as the rule antecedent or precondition. The “THEN”-part (or right-hand side) is the rule consequent. In the rule antecedent, the condition consists of one or more attribute tests (such as age = youth, and student = yes) that are logically ANDed. The rule’s consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). R1 can also be written as
R1: (age = youth) ^ (student = yes)(buys computer = yes).

If the condition (that is, all of the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule antecedent is satisfied (or simply, that the rule is satisfied) and that the rule covers the tuple. A rule R can be assessed by its coverage and accuracy. Given a tuple, X, from a class-labeled data set, D, let \( n_{\text{covers}} \) be the number of tuples covered by R; \( n_{\text{correct}} \) be the number of tuples correctly classified by R; and \( |D| \) be the number of tuples in D. We can define the coverage and accuracy of R as

\[
\text{coverage}(R) = \frac{n_{\text{covers}}}{|D|} \\
\text{accuracy}(R) = \frac{n_{\text{correct}}}{n_{\text{covers}}}.
\]

Table 6.1. A customer will buy a computer. Consider rule R1 above, which covers 2 of the 14 tuples. It can correctly classify both tuples. Therefore, coverage(R1) = 2/14 = 14.28% and accuracy
(R1) = 2/2 = 100%.

If a rule is satisfied by X, the rule is said to be triggered. For example, suppose we have X=
(age = youth, income = medium, student = yes, credit rating = fair).

We would like to classify X according to buys computer. X satisfies R1, which triggers the rule. If R1 is the only rule satisfied, then the rule fires by returning the class prediction for X.

If more than one rule is triggered, we need a conflict resolution strategy to figure out which rule gets to fire and assigns its class prediction to X. There are many possible strategies. We look at two, namely size ordering and rule ordering. The size ordering scheme assigns the highest priority to the triggering rule that has the “toughest” requirements, where toughness is measured by the rule antecedent size. That is, the triggering rule with the most attribute tests is fired.

The rule ordering scheme prioritizes the rules beforehand. The ordering may be class-based or rule-based. With class-based ordering, the classes are sorted in order of decreasing “importance,” such as by decreasing order of prevalence. That is, all of the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on.

With rule-based ordering, the rules are organized into one long priority list, according to some measure of rule quality such as accuracy, coverage, or size (number of attribute tests in the rule antecedent), or based on advice from domain experts. When rule ordering is used, the rule set is known as a decision list. With rule ordering, the triggering rule that appears earliest in the list has highest priority, and so it gets to fire its class prediction.
Rule Extraction from a Decision Tree

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).

Example Extracting classification rules from a decision tree. The decision tree of Figure 6.2 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Figure 6.2 are
R1: IF age = youth AND student = no THEN buys computer = no
R2: IF age = youth AND student = yes THEN buys computer = yes
R3: IF age = middle aged THEN buys computer = yes
R4: IF age = senior AND credit rating = excellent THEN buys computer = yes
R5: IF age = senior AND credit rating = fair THEN buys computer = no

Rule Induction Using a Sequential Covering Algorithm

IF-THEN rules can be extracted directly from the training data (i.e., without having to generate a decision tree first) using a sequential covering algorithm. The name comes from the notion that the rules are learned sequentially (one at a time), where each rule for a given class will ideally cover many of the tuples of that class (and hopefully none of the tuples of other classes).

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

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:
(1) Rule set = $\emptyset$; // initial set of rules learned is empty
(2) for each class $c$ do
(3) repeat
(4) Rule = Learn One Rule(D, Att vals, $c$);
(5) remove tuples covered by Rule from D;
(6) until terminating condition;
(7) Rule set = Rule set +Rule; // add new rule to rule set
(8) endfor
(9) return Rule Set;

Figure

Basic sequential covering algorithm.

Classification by Back propagation:

“What is backpropagation?” Backpropagation is a neural network learning algorithm. A neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples. Neural network learning is also referred to as connectionist learning due to the connections between units.

A Multilayer Feed Forward Neural Network:
The backpropagation algorithm performs learning on a multilayer feed-forward neural network. It iteratively learns a set of weights for prediction of the class label of tuples. A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer. An example of a multilayer feed-forward network.

Each layer is made up of units. The inputs to the network correspond to the attributes measured for each training tuple. The inputs are fed simultaneously into the units making up the input layer. These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of “neuronlike” units, known as a hidden layer. The outputs of the hidden layer units can be input to another hidden layer, and so on. The number of hidden layers is arbitrary, although in practice, usually only one is used. The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network’s prediction for a given tuple. The units in the input layer are called input units. The units in the hidden layers and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units. The multilayer neural network shown in Figure 6.15 has two layers of output units. Therefore, we say that it is a two-layer neural network.

The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer. It is fully connected in that each unit provides input to each unit in the next forward layer.

Classification by Backpropagation:

- Backpropagation is a neural network learning algorithm.
- A neural network is a set of connected input/output units in which each connection has a weight associated with it.
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples.
- Neural network learning is also referred to as connectionist learning due to the connections between units.
- Neural networks involve long training times and are therefore more suitable for applications where this is feasible.
- Backpropagation learns by iteratively processing a data set of training tuples, comparing the network’s prediction for each tuple with the actual known target value.
The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for prediction).

For each training tuple, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual target value. These modifications are made in the “backwards” direction, that is, from the output layer, through each hidden layer down to the first hidden layer hence the name is backpropagation.

Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops.

Advantages:

- It includes their high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained.
- They can be used when you may have little knowledge of the relationships between attributes and classes.
- They are well-suited for continuous-valued inputs and outputs, unlike most decision tree algorithms.
- They have been successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text.
- Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process.

Process:

Initialize the weights:

The weights in the network are initialized to small random numbers ranging from -1.0 to 1.0, or -0.5 to 0.5. Each unit has a bias associated with it. The biases are similarly initialized to small random numbers.

Each training tuple, $X$, is processed by the following steps.

Propagate the inputs forward:

First, the training tuple is fed to the input layer of the network. The inputs pass through the input units, unchanged. That is, for an input unit $j$, its output, $O_j$, is equal to its input value, $I_j$. Next, the net input and output of each unit in the hidden and output layers are computed. The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs.

Each such unit has anumber of inputs to it that are, in fact, the outputs of the units connected to
it in the previous layer. Each connection has a weight. To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed.

\[ l_j = \sum_i w_{ij} o_i + \theta_j, \]

where \( w_{ij} \) is the weight of the connection from unit \( i \) in the previous layer to unit \( j \); \( o_i \) is the output of unit \( i \) from the previous layer; \( \theta_j \) is the bias of the unit & it acts as a threshold in that it serves to vary the activity of the unit.

Each unit in the hidden and output layers takes its net input and then applies an activation function to it.

Backpropagate the error:

The error is propagated backward by updating the weights and biases to reflect the error of the network’s prediction. For a unit \( j \) in the output layer, the error \( Err_j \) is computed by

\[ Err_j = o_j(1 - o_j)(T_j - o_j) \]

where \( o_j \) is the actual output of unit \( j \), and \( T_j \) is the known target value of the given training tuple.

The error of a hidden layer unit \( j \) is

\[ Err_j = o_j(1 - o_j) \sum_k Err_k w_{jk} \]

where \( w_{jk} \) is the weight of the connection from unit \( j \) to a unit \( k \) in the next higher layer, and \( Err_k \) is the error of unit \( k \).

Weights are updated by the following equations, where \( D_{wj} \) is the change in weight \( w_{ij} \):
Support Vector Machines

Support Vector Machines, a promising new method for the classification of both linear and nonlinear data. In a nutshell, a support vector machine (or SVM) is an algorithm that works as follows. It uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane (that is, a “decision boundary” separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors (“essential” training tuples) and margins.

The Case When the Data Are Linearly Separable

Let the data set D be given as \((X_1, y_1), (X_2, y_2), \ldots, (X_{|D|}, y_{|D|})\), where \(X_i\) is the set of training tuples with associated class labels, \(y_i\). Each \(y_i\) can take one of two values, either +1 or -1 (i.e., \(y_i\) belongs to \(+1, -1\)), corresponding to the classes buys computer = yes and buys computer = no, respectively. To aid in visualization, let’s consider an example based on two input attributes, \(A_1\) and \(A_2\), as shown in Figure 6.20. From the graph, we see that the 2-D data are linearly separable (or “linear,” for short) because a straight line can be drawn to separate all of the tuples of class +1 from all of the tuples of class -1.

There are an infinite number of separating lines that could be drawn. We want to find the “best” one, that is, one that (we hope) will have the minimum classification error on previously unseen tuples. How can we find this best line? Note that if our data were 3-D (i.e., with three attributes), we would want to find the best separating plane. Generalizing to \(n\) dimensions, we want to find the best hyperplane. We will use the term “hyperplane” to refer to the decision boundary that we are seeking, regardless of the number of input attributes. So, in other words, how can we find the best hyperplane? An SVM approaches this problem by searching for the maximum marginal hyperplane.
Hyper plane with the larger margin to be more accurate at classifying future data tuples than the hyperplane with the smaller margin. This is why (during the learning or training phase), the SVM searches for the hyperplane with the largest margin, that is, the maximum marginal hyper plane (MMH). The associated margin gives the largest separation between classes. Getting to an informal definition of margin, we can say that the shortest distance from a hyperplane to one side of its margin is equal to the shortest distance from the hyper plane to the other side of its margin, where the “sides” of the margin are parallel to the hyperplane. When dealing with the MMH, this distance is, in fact, the shortest distance from the MMH to the closest training tuple of either class. A separating hyper plane can be written as

\[ W \cdot X + b = 0; \]

where \( W \) is a weight vector, namely, \( W = w_1, w_2, \ldots, w_n \); \( n \) is the number of attributes; and \( b \) is a scalar, often referred to as a bias. To aid in visualization, let's consider two input attributes, \( A_1 \) and \( A_2 \).

Training tuples are 2-D, e.g., \( X = (x_1, x_2) \), where \( x_1 \) and \( x_2 \) are the values of attributes \( A_1 \) and \( A_2 \), respectively, for \( X \). If we think of \( b \) as an additional weight, \( w_0 \), we can rewrite the above separating hyper plane as

\[ w_0 + w_1 x_1 + w_2 x_2 = 0; \]

Thus, any point that lies above the separating hyper plane satisfies \( w_0 + w_1 x_1 + w_2 x_2 > 0 \):

Similarly, any point that lies below the separating hyper plane satisfies \( w_0 + w_1 x_1 + w_2 x_2 < 0 \):
Lazy Learners (or Learning from Your Neighbors):

The classification methods is a decision tree induction, Bayesian classification, rule-based classification, classification by back propagation, support vector machines, and classification based on association rule mining—are all examples of eager learners. Eager learners, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify.

**k-Nearest-Neighbor Classifiers**

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple...
with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all of the training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a k-nearest-neighbor classifier searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k “nearest neighbors” of the unknown tuple. “Closeness” is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples, say, X1 = (x11, x12, : : : , x1n) and X2 = (x21, x22, : : : , x2n), is

\[ \text{dist}(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}. \]

In other words, for each numeric attribute, we take the difference between the corresponding values of that attribute in tuple X1 and in tuple X2, square this difference, and accumulate it. The square root is taken of the total accumulated distance count.

Typically, we normalize the values of each attribute before using Equation (6.45). This helps prevent attributes with initially large ranges (such as income) from out weighing attributes with initially smaller ranges (such as binary attributes). Min-max normalization, for example, can be used to transforms value v of a numeric attribute A to v0 in the range [0, 1] by computing

\[ v' = \frac{v - \text{min}_A}{\text{max}_A - \text{min}_A}. \]

where minA and maxA are the minimum and maximum values of attribute A.

For k-nearest-neighbor classification, the unknown tuple is assigned the most common class among its k nearest neighbors. When k = 1, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the k nearest neighbors of the unknown tuple.

“But how can distance be computed for attributes that not numeric, but categorical, such as color?”

The above discussion assumes that the attributes used to describe the tuples are all numeric. For categorical attributes, a simple method is to compare the corresponding value of the attribute in tuple X1 with that in tuple X2. If the two are identical (e.g., tuples X1 and X2 both have the color blue), then the difference between the two is taken as 0. If the two are different (e.g., tuple X1 is blue but tuple X2 is red), then the difference is considered to be 1.

“What about missing values?” In general, if the value of a given attribute A is missing in tuple X1 and/or in tuple X2, we assume the maximum possible difference. Suppose that each of the attributes have been mapped to the range [0, 1]. For categorical attributes, we take the difference value to be 1 if either one or both of the corresponding values of A are missing. If A is numeric and missing from both tuples X1 and X2, then the difference is also taken to be 1.

Case-Based Reasoning

Case-based reasoning (CBR) classifiers use a database of problem solutions to solve new problems. Unlike nearest-neighbor classifiers, which store training tuples as points in Euclidean space, CBR stores the tuples or “cases” for problem solving. Business applications of CBR include problem resolution for customer service help desks, where cases describe product-related diagnostic problems. CBR has also