TDT4300 Compendium

Jacob Clements

May 2024

1 Introduction

1.1 Content

The course deals with methods and theory for developing data warehouses and performing data analysis using data mining:

1. Data quality and techniques for pre-processing data
2. Modelling and design of data warehouses
3. Algorithms for classification, clustering and association rule detection
4. Practical use of data analysis software

1.2 Main points for this course

- Data is growing at a rapid pace. However, our capability to analyse it is limited
- Key challenge in analyzing massive amounts of data is scalability
- Data management systsems offer the capability to organize the data effectively for efficient data analysis.
- Objective of data mining is to discover insights into Big Data either through user-driven exploration or automatically through algorithms.

This knowledge discovery process can be summarized as follows:

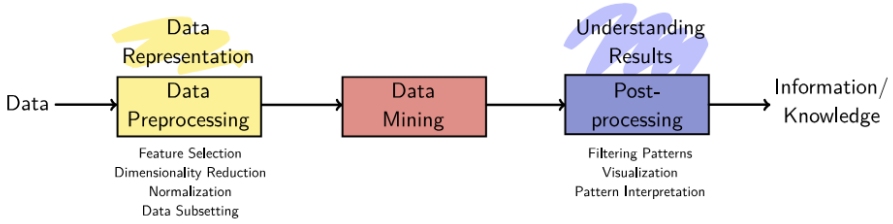


Figure 1: Data preprocessing, data mining, and post-processing pipeline.

1.2.1 Note:

The first step of KDD process takes the most amount of time as as it requires cleaning and a determination of proper representation so as for the data mining algorithms to be applicable.

2 Data Warehousing

2.1 Part 1: Origins of Data Warehouses

2.1.1 Data analysis and visualizations

Data analysis:

- Data analysis applications look for unusual patterns in data.
- Data analysis applications typically aggregate data across many dimensions looking for unusual patterns.
- They categorize data values and trends, extract statistical information, and then contrast one category with another.
- Formulating a query that extracts relevant data from a large database.
- Extracting the aggregated data from the database into a file or table.
- Visualizing the results in a graphical way.
- Analyzing the results and formulating a new query.

Visualization:

- Visualization tools display data trends, clusters, and differences.
- Visualization focuses on presenting new graphical metaphors that allow people to discover data trends and anomalies.
- These visualization and data analysis tools represent the dataset as an N-dimensional space.
- Visualization tools render two and three-dimensional sub-slabs of this space as 2D or 3D objects.
- Color and time (motion) add two more dimensions to the display giving the potential for a 5D display.

Thus, visualization as well as data analysis tools do "dimensionality reduction", often by summarizing data along the dimensions that are left out. Along with summarization and dimensionality reduction, data analysis applications use constructs such as histogram, cross-tabulation, subtotals, roll-up and drill-down extensively.

2.1.2 Problems with SQL aggregation constructs

Certain common form of data anlysis are difficult with SQL aggregation constructs. Four common problems are:

1. Histograms
2. Roll-up
3. Totals and subtotals for drill-downs
4. cross tabulations

2.1.3 The answer: Data cubes

- The generalization of group by, roll-up and cross-tab ideas seems obvious: Figure 2 shows the concept for aggregation up to 3-dimensions.
- The traditional GROUP BY generates the N-dimensional data cube core.
- The N-1 lower-dimensional aggregates appear as points, lines, planes, cubes, or hyper-cubes hanging off the data cube core.
- Creating a data cube requires generating the power set (set of all subsets) of the aggregation columns.

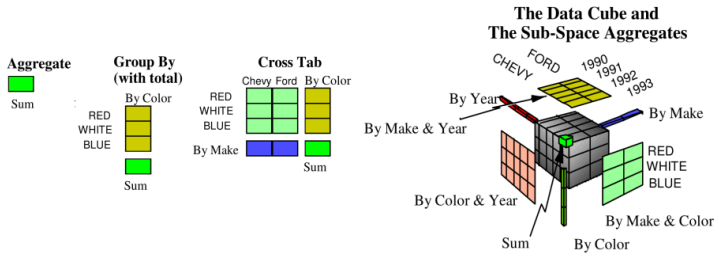


Figure 2: 3-D data cube

Cube operator: The cube operator is the N-dimnesional generalization of simple aggregate functions. The 0-D data cube is a point. The 1-D data cube is a line with a point. The 2-D data cube is a cross tabulation, a plane, two lines, and a point. The 3-D data cube is a cube with 3 intersecting 2-D cross tabs.

2.2 Part 2: Data Cubes

- A data cube allows data to be modeled and viewed in multiple dimensions.
- It is defined by dimensions and facts.

- Dimensions are the perspectives or entities with respect to which an organization wants to keep records. Each dimension may have a table associated with it, called a dimension table, which further describes the dimension.
- A multidimensional data model is typically organized around a central theme, such as sales. This theme is represented by a fact table.
- Facts are numeric measures. Think of them as the quantities by which we want to analyze relationships between dimensions.
- The fact table contains the names of the facts, or measures, as well as keys to each of the related dimension tables.
- A data cube is often referred to as a cuboid.
- Given a set of dimensions, we can generate a cuboid for each of the possible subsets of the given dimensions.
- The result would form a lattice of cuboids, each showing the data at a different level of summarization, or group-by.
- The lattice of cuboids is then referred to as a data cube.

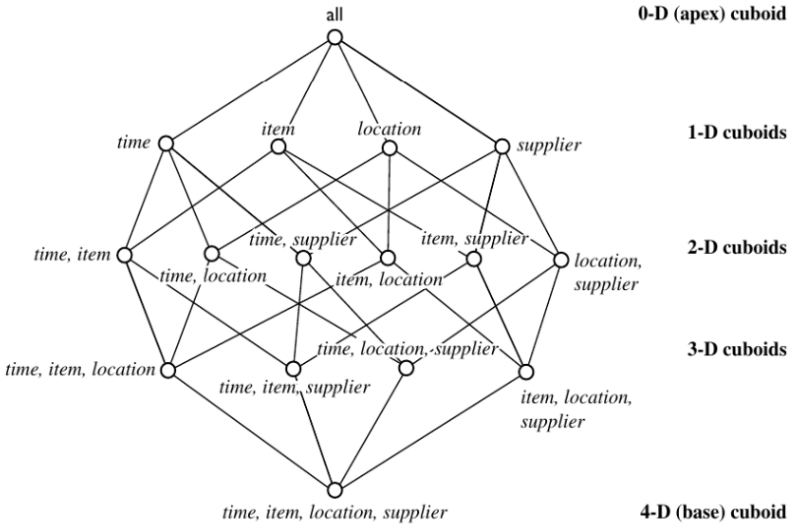


Figure 3: Lattice of cuboids, making up a 4-D data cube for time, item, location and supllier. Each cuboid represents a different degree of summarization

2.2.1 Dimensions and Concept Hierarchies

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

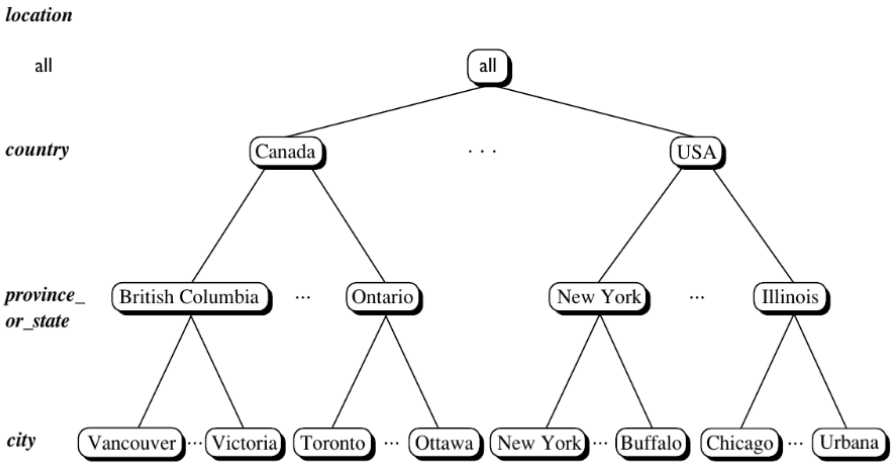


Figure 4: A concept hierarchy for location. Due to space limitations not all of the hierarchy nodes are shown, indicated by the ellipses between nodes

Concept hierarchies may also be defined by discretizing or grouping values for a given dimension or attribute, resulting in a set-grouping hierarchy.

2.2.2 Schema designs

- The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities and the relationships between them.
- Such a data model is appropriate for online transaction processing.
- A data warehouse, however, requires a concise, subject-oriented schema that facilitates online data analysis.
- The most popular data model for a data warehouse is a multidimensional model, which can exist in the form of:
    1. a star schema,
    2. a snowflake schema, or
    3. a fact constellation schema.

Star schema: A star schema contains:

1. a large central table (fact table) containing the bulk of the data, with no redundancy

2. a set of smaller attendent 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.
Illustration:

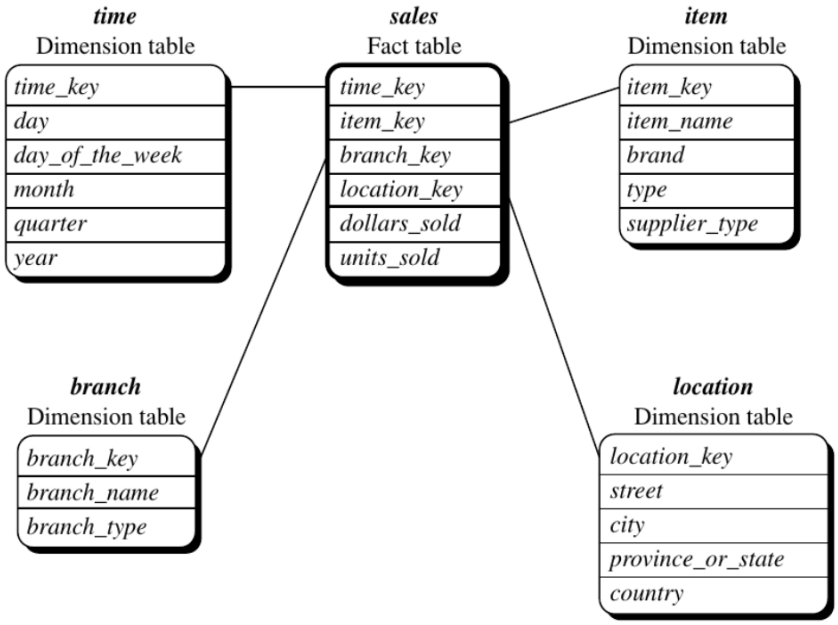


Figure 5: Star schema of sales: data warehouse

Concisely elucidated: fact table and dimension tables:

- **Fact table:** contains the main quantatative data (facts) of the business process and foreign keys to dimension tables. It focuses on numeric measures that are typically subject to analysis.

- **Dimension tables:** contains descriptive attributes (meta data) about the dimensions that contextualizes the facts. Each dimension table is linked to the fact table via a primary key that is referenced as a foreign key in the fact table. These tables are not typically used to perform major calculations but provide filtering, grouping, and labelling capabilities for reporting and analytics.

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are nomralized, therby further splitting the

data into additional tables. The resulting schema graph forms a shape similar to a snowflake.

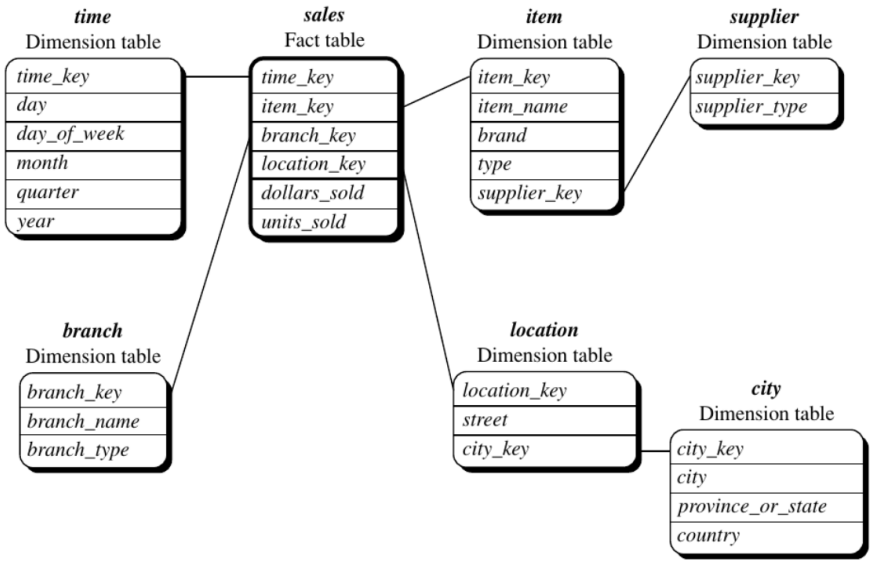


Figure 6: Snowflake schema of sales: data warehouse

Star vs. snoflake schemas:

- 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 to reduce redundancies.
- Such a table is easy to maintain and saves storage space.
- However, this space savings 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, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design.

Fact constellation schema: 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 schema.

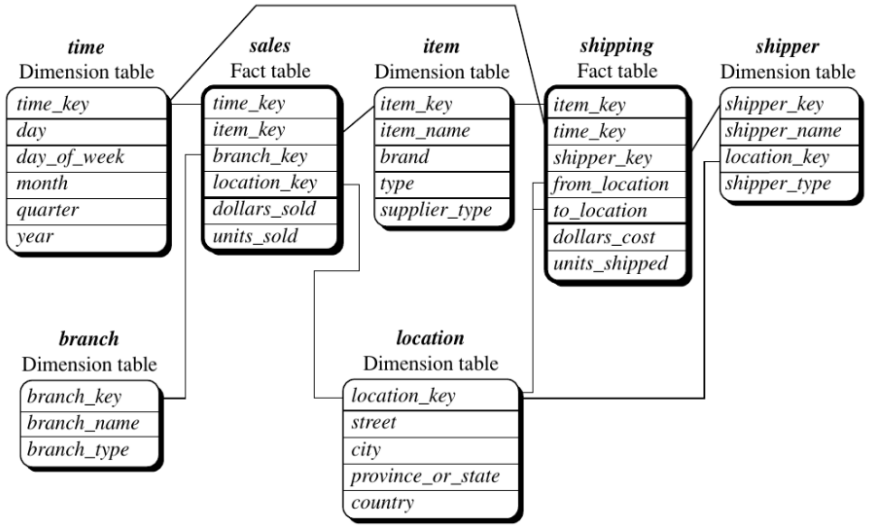


Figure 7: Fact constellation schema of sales and shipping: data warehouse

2.2.3 Operators

A number of data cube operations exist to materialize these different views, allowing interactive querying and analysis of the data at hand.

1. **Drill Down:** is the process of moving data from summary data to more detailed data by increasing the granularity. It involves navigating into deeper layers of dimension hierarchies or adding more dimensions to the analysis. An example is drill down on time from quarters to months.

2. **Roll Up:** is the opposite of drill-down. It involves summarizing data by climbing up a dimension hierarchy or decreasing the level of detail. An example is roll-up on location from cities to countries.

3. **Dice:** Dicing means creating a sub-cube by selecting specific values of dimensions, which effectively filters the data cube on two or more dimensions. An example is dice for (location = "Toronto" or "Vancouver") and (time = "Q1" or "Q2) and (item = "home entertainment" or "computer"). This would create a sub-cube.

4. **Pivot:** Pivoting the data cube allows you to reorient the cube to see different dimensions along the axes.

5. **Slice:** Slicing is similar to dicing but involves cutting through the data cube along one dimension, which reduces the cubes dimensionality. This operation selects a single value for one of the dimensions, creating a slice of the cube.

2.2.4 Measures

- CUBE operator proposed by Gray et al. allows aggregation function to be expressed over attributes or dimensions in the argument list.
- These aggregate functions or measures that are computed over dimensions or attributes can be of three types so that they can be efficiently computed over multi-dimensional data:
  1. Distributive.
  2. Algebraic.
  3. Holistic.

Distributive Functions

- Distributive functions are those where the result of applying the function to a large set can be broken down into smaller parts, computed separately, and then combined to get the final result.
- Examples include `COUNT()`, `MIN()`, `MAX()`, and `SUM()`.
- Essentially, for these functions, the same operation can be used both on individual parts and on the combined results.
- For the `COUNT()` function, combining results involves summing up the counts from each subset.

Algebraic Functions

- Algebraic functions work by summarizing data using a fixed-size summary (like a pair of numbers), which can then be combined to produce the final result.
- Examples include `AVERAGE()`, `STD-DIV`, `MaxN()`, `MinN()`, and `CENTER-OF-MASS()`.
- For example, to find an `AVERAGE`, you first keep track of both the total sum and the count of items in each subset, then combine these to calculate the overall average.
- These functions use summaries that stay the same size, no matter how much data you have.

Holistic Functions

- Holistic functions are those where you can't summarize the data with a fixed-size summary. The storage needed to describe the result can grow with the amount of data.
- Examples include `Median()`, `MostFrequent()` (also known as `Mode()`), and `Rank()`.
- For these functions, you need all or most of the data to accurately compute the result, as no fixed-size summary can capture everything needed.

2.3 Part 3: Data Warehouse Concepts

2.3.1 Definition:

"A data warehouse is a subject-oriented, integrated, time-invariant, and nonvolatile collection of data in support of management's decision making process.

- Data warehouses are subject-oriented:
  - Organized around major subjects (e.g., customer, supplier, product, and sales).
  - Focus on the modeling and analysis of data for decision makers (as opposed to day-to-day operations and transaction processing of an organization)
  - Provide a simple and concise view of particular subject issues by excluding data that are not useful in the decision support process.
- Data warehouses are integrated:
  - Usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and online transaction records.
  - Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on.
- Data warehouses are time-variant:
  - Data are stored to provide information from a historic perspective (e.g., the past 5–10 years).
  - Every key structure in the data warehouse contains, either implicitly or explicitly, a time element.
- Data warehouses are nonvolatile:
  - Always a physically separate store of data transformed from the application data found in the operational environment.
  - A data warehouse does not require transaction processing, recovery, and concurrency control mechanisms.
  - It usually requires only two operations in data accessing:
    - Initial loading of data.
    - Access of data.

2.3.2 Applications

- Organizations use this information to support business decision-making activities:

1. Increasing customer focus (i.e., analysis of customer buying patterns)
   - Buying preference,
   - Buying time,
   - Budget cycles,
   - Appetites for spending
2. Comparing the performance of sales by quarter, by year, and by geographic regions in order to fine-tune production strategies.
3. Analyzing operations and looking for sources of profit.
4. Managing customer relationships.
5. Making environmental corrections.
6. Managing the cost of corporate assets.

2.3.3 Databases versus data warehouses

Feature	*OLTP*	*OLAP*
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long-term informational requirements decision support
DB design	ER-based, application-oriented	star/snowflake, subject-oriented
Data	current, guaranteed up-to-date	historic, accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
Number of records accessed	tens	millions
Number of users	thousands	hundreds
DB size	GB to high-order GB	$\geq$ TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Figure 8: Comparison of database and data warehouse

2.3.4 Data Warehouse Architecture

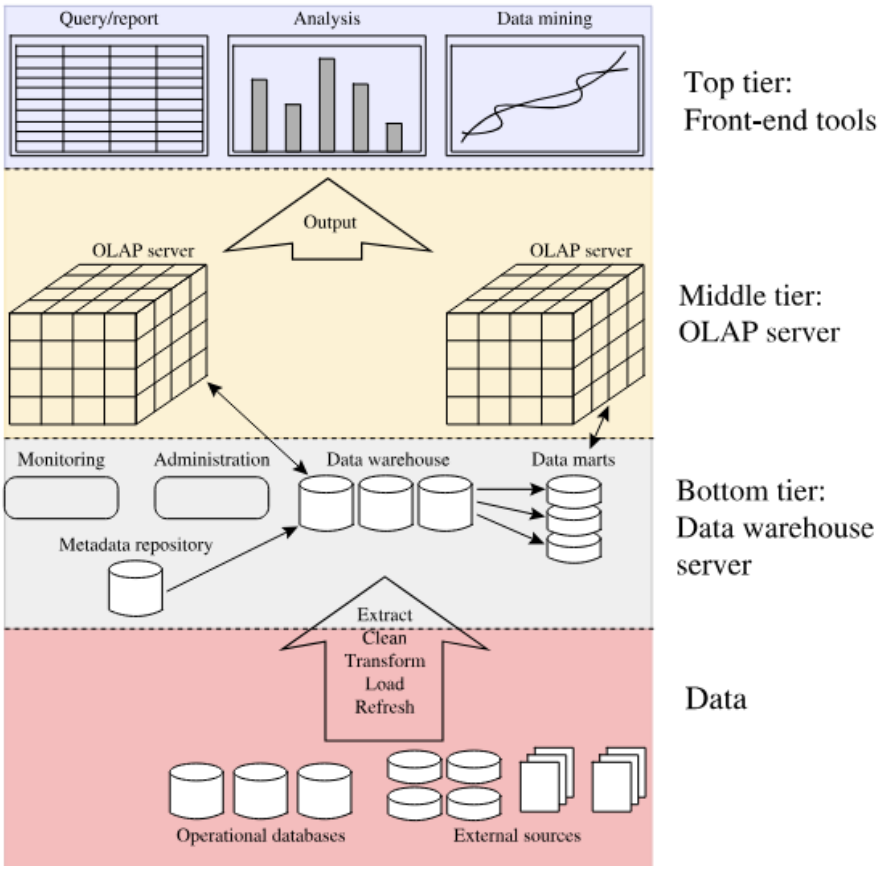


Figure 9: A three-tier data warehousing architecture

Bottom tier:

- Always a relational database system.
- Back-end tools and utilities are used to feed data from operational databases or other external sources.
- Back-end tools and utilities perform data extraction, cleaning, and transformation.
- Also contains a metadata repository, which stores information about the data warehouse and its contents.

Middel tier:

- Is an OLAP server.
- Can be a relational OLAP (ROLAP) model
  - An extended relational DBMS that maps operations on multidimensional data to standard relational operations.
- Can be a multi-dimensional OLAP (MOLAP) model
  - A special-purpose server that directly implements multidimensional data and operations.

Top tier:

- Is a front-end client layer that contains:
  - Query and reporting tools.
  - Analysis tools.
  - Data mining tools (e.g., trend analysis, prediction, and so on).

2.3.5 Data Warehouse Models

We have three type of data warehouse models:

- Enterprise warehouse
- Data mart
- Virtual warehouse

Enterprise warehouse:

- Contains information about subjects spanning the entire organization.
- Provides corporate-wide data integration from
  1. Operational systems
  2. External information providers
- Scope: cross-functional.
- Contains detailed data as well as summarized data.
- Size: between gigabytes to terabytes (or beyond).
- Implementation on: traditional mainframes, computer superservers, or parallel architecture platforms.
- Implementation cycle: years (requires extensive business modeling).

Data mart:

- Contains a subset of corporate-wide data that is of value to a specific group of users.
- Scope: specific selected subjects.
  - For example, a marketing data mart may confine its subjects to customer, item, and sales.
- Data contained tends to be summarized.
- Implementation on: low-cost departmental servers.
- Implementation cycle: weeks.
- Independent data marts:
  - Data captured from:
    * Operational systems
    * External information providers
    * Data generated locally within a department or geographic area.
- Dependent data marts:
  - Data captured from: enterprise data warehouses.

Virtual warehouse:

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

Data warehouse development:

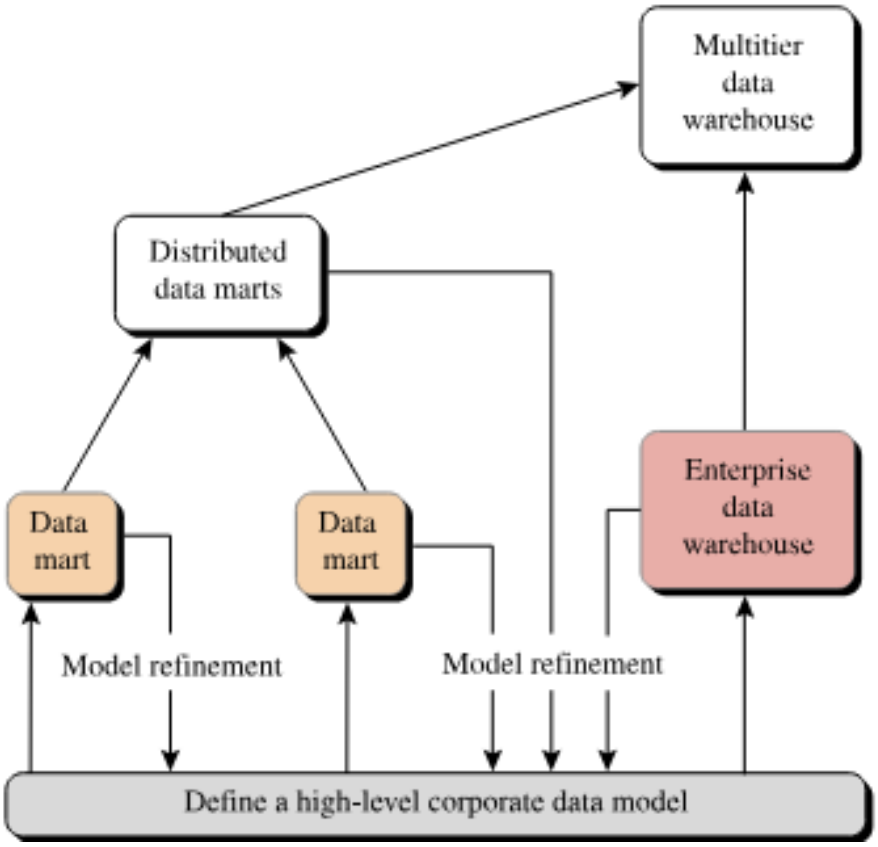


Figure 10: A recommended approach for data warehouse development

- Implement the warehouse in an incremental and evolutionary manner.
- High-level corporate data model is defined.
- Independent data marts can be implemented in parallel.
- Distributed data marts can be constructed to integrate different data marts via hub servers.

2.3.6 ETL process

- Data warehouse systems use back-end tools and utilities to populate and refresh their data.

    1. Data Extraction: gathers data from multiple, heterogeneous, and external sources.

2. Data Cleaning: detects errors in the data and rectifies them when possible.
3. Data Transformation: converts data from legacy or host format to warehouse format.
4. Load: sorts, summarizes, consolidates, computes views, checks integrity, and builds indices and partitions.
5. Refresh: propagates the updates from the data sources to the warehouse.

2.3.7 Metadata repository

Metadata:

- Metadata are data about data.
- Metadata are the data that define warehouse objects.
- Metadata are created and captured for:
  1. Timestamping any extracted data.
  2. Source of the extracted data.
  3. Missing fields that have been added by data cleaning or integration processes.

Metadata repository:

- Description of the data warehouse structure:
  1. Warehouse schema.
  2. View.
  3. Dimensions.
  4. Hierarchies.
  5. Derived data definitions.
  6. Data mart locations and contents.
- Operational metadata:
  1. Data lineage (history of migrated data and the sequence of transformations applied to it).
  2. Currency of data (active, archived, or purged).
  3. Monitoring information (warehouse usage statistics, error reports, and audit trails).
- Algorithms used for summarization:
  1. Measure and dimension definition algorithms.

2. Data on granularity.
3. Partitions.
4. Subject areas.
5. Aggregation.
6. Summarization.
7. Predefined queries and reports.

- Mapping from the operational environment to the data warehouse:
  1. Source databases and their contents.
  2. Gateway descriptions.
  3. Data partitions.
  4. Data extraction, cleaning, transformation rules and defaults.
  5. Data refresh and purging rules.
  6. Security (user authorization and access control).
- Data related to system performance:
  1. Indices and profiles that improve data access and retrieval performance.
  2. Rules for the timing and scheduling of refresh, update, and replication cycles.
- Business metadata:
  1. Business terms and definitions.
  2. Data ownership information.
  3. Charging policies.

2.4 Part 4: Data Warehouse Implementation

2.4.1 Cube Operator

A data cube is a lattice of cuboids.

- Base cuboid contains all three dimensions, city, item, and year. Can return the total sales for any combination of the three dimensions.
- Apex cuboid, or 0-D cuboid contains the total sum of all sales.
- Base cuboid is the least generalized (most specific) of the cuboids.
- Apex cuboid is the most generalized (least specific) of the cuboids, and is often denoted as `ALL`.

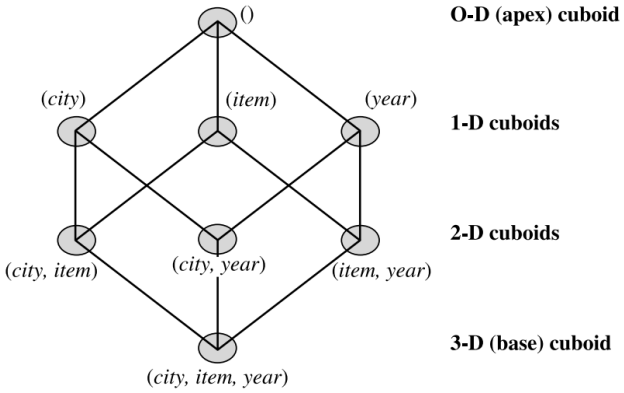


Figure 11: Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains city, item and year dimensions

- Drill Down within Data Cube: start at the apex cuboid and explore downward in the lattice.
- Roll Up within Data Cube: start at the base cuboid and explore upward.
- A cube operator on n dimensions is equivalent to a collection of group-by statements, one for each subset of the n dimensions. Hence, the cube operator is a n-dimensional generalization of the group-by operator.
- For a cube with n dimensions, there are a total of 2^n cuboids, including the base cuboid.
- Good idea to compute in advance all or at least some of the cuboids in the data cube.
- Precomputation leads to fast response time and avoids some redundant computation

2.4.2 Cube Operator Precomputation

Given that we have n dimensions where each i-th dimension has L_i levels, the total number of cuboids would be:

$$\text{Total number of cuboids} = \prod_{1}^{n}(L_i + 1)$$

The number one is added to account for the ALL value.

Note: It is unrealistic to precompute all cuboids that can possibly be generated from the data cube. Hence, a more realistic approach is partial materialization.

Note: Materialization

- No Materialization: compute expensive multidimensional aggregates on-the-fly (very slow).
- Full Materialization: precompute all of the cuboids (full cube). Requires large memory space.
- Partial Materialization: selectively compute a proper subset of the whole set of possible cuboids.

Note: Partial Materialization

- Selection of subset of cuboids to materialize may depend on:
  - Query Workload: frequencies and access cost.
  - Workload Characteristics: update cost and storage requirements.
  - Physical Database Design: generation and selection of indices.
  - Heuristics: based on frequently referenced cuboids.
- Iceberg Cube: which is a data cube that stores only those cube cells with an aggregate value (e.g., count) that is above some minimum support threshold.
- Shell Cube: precomputes cuboids for a small number of dimensions (e.g., three to five). Queries on additional combinations of the dimensions can be computed on-the-fly.

2.4.3 Indexing OLAP Data

Indexing overview:

- Fact tables are typically very large and are used in the majority of queries.
- Therefore, a fact table often has many indices such that most queries can benefit from one or more of them.
- Typically, a separate index is built on each dimension key.
- If the DBMS supports index intersection, the indices can then be used in combination when answering a query.
- It can also be a good idea, however, to create indices on combinations of dimension keys for those combinations that are often used together by queries.
- The combinations for which to create indices thus have to be chosen carefully based on the typical usage.
- In general, the key referencing the Date dimension should be put first in combinations since most queries refer to dates.

- Virtually any DBMS supports the B-tree index which is a tree structure where the leaves contain lists of row IDs (where a row ID can be a physical location of the row or something else the system can use to identify and find a certain row).
- To find rows that have a given value for the indexed attribute, the system traverses the tree and obtains a list of row IDs. The row IDs are then used to retrieve the rows.
- The B-tree is an efficient index.
- However, when the indexed attribute has low cardinality (i.e., holds few different values), another representation can be a better choice.

Bitmap indexing: Bitmap indexing is a type of database indexing technique that uses bit arrays (bitmaps) to represent the presence or absence of a value or a range of values. Each bit in the bitmap corresponds to a row in the database table, and its value indicates whether the row contains the specific value of interest.

How is Bitmap Indexing Done?

- **Bitmap Creation**:
  - For each distinct value in the column to be indexed, a bitmap is created.
  - Each bitmap is a bit array where the length of the array is equal to the number of rows in the table.
  - If a row contains the value, the corresponding bit in the bitmap is set to 1; otherwise, it is set to 0.
- **Bitmap Storage**:
  - The bitmaps are stored in a compressed form to save space, as bitmaps tend to be sparse.
  - Compression techniques like Run-Length Encoding (RLE) are often used.
- **Query Processing**:
  - When a query is executed, the bitmaps for the relevant values are retrieved and combined using bitwise operations (AND, OR, NOT).
  - The result of these operations is another bitmap that identifies the rows satisfying the query conditions.

When is Bitmap Indexing Done?

Bitmap indexing is typically used in scenarios where:

- The data set is large and mostly read-only.

- The columns to be indexed have a low cardinality (few distinct values), such as gender, yes/no, or status fields.
- The query workload involves complex and ad-hoc queries, particularly those with multiple AND/OR conditions.

Why is Bitmap Indexing Beneficial?

- **Efficient Query Processing**:
  - Bitmap indexing allows for rapid query processing through bitwise operations, which are extremely fast and efficient.
  - It is especially effective for complex queries involving multiple conditions.
- **Space Efficiency**:
  - Although bitmaps can be large, they can be highly compressed, significantly reducing the storage space required compared to traditional indexing methods.
- **Reduced I/O Operations**:
  - Since the bitmaps are compact and can be processed in memory, bitmap indexing reduces the number of I/O operations needed to execute a query.
- **Scalability**:
  - Bitmap indexes can efficiently handle large data sets and are scalable, making them suitable for data warehousing and analytics applications.

In summary, bitmap indexing is a powerful technique for speeding up query processing in databases, particularly for large, read-intensive data sets with low-cardinality columns. It leverages the efficiency of bitwise operations and compression to provide fast and space-efficient indexing.

Join indexing: Join indexing is a database indexing technique designed to improve the efficiency of join operations, particularly in relational databases. It creates an index that explicitly represents the join relationships between two or more tables, allowing for faster retrieval of related records.

How is Join Indexing Done?

- **Index Creation**:
  - A join index is created by identifying the columns involved in the join condition across multiple tables.
  - The index stores pairs (or tuples) of row identifiers (RIDs) from the involved tables that satisfy the join condition.

- **Index Storage**:
  - The join index is stored separately from the actual data tables.
  - It may include additional metadata to facilitate quick lookups and updates.
- **Query Processing**:
  - When a query involving a join is executed, the join index is used to quickly locate and retrieve the related rows from the involved tables.
  - This reduces the need to perform a full table scan or compute the join on-the-fly.

When is Join Indexing Done?
Join indexing is typically used in scenarios where:

- The database workload involves frequent and complex join operations.
- The tables being joined are large, making full table scans costly.
- The relationships between tables are relatively stable, meaning the join conditions do not change frequently.

Why is Join Indexing Beneficial?

- **Efficient Join Operations**:
  - Join indexing significantly speeds up join operations by directly linking related rows across tables.
  - This is particularly beneficial for complex queries involving multiple joins.
- **Reduced Computation Cost**:
  - By precomputing and storing the join relationships, the database can avoid the costly computation of joins at query runtime.
- **Improved Query Performance**:
  - Join indexes can lead to faster query response times, making the system more responsive and efficient.
- **Optimal for Data Warehousing**:
  - Join indexing is especially useful in data warehousing environments, where complex queries and large datasets are common.

In summary, join indexing is a powerful technique for optimizing join operations in databases. By precomputing and storing the relationships between tables, it enhances query performance and reduces computational overhead, making it particularly valuable for complex queries and large datasets.

Efficient Processing of OLAP Operations:

- The purpose of materializing cuboids and constructing OLAP index structures is to speed up query processing in data cubes.
- Given materialized views, query processing should proceed as follows:
    1. Determine which operations should be performed on the available cuboids:
        - Transform any selection, projection, roll-up (group-by), and drill-down operations specified in the query into corresponding SQL and/or OLAP operations.
        - For example, slicing and dicing a data cube corresponds to selection and/or projection operations on a materialized cuboid.
    2. Determine to which materialized cuboid(s) the relevant operations should be applied:
        - Identifying materialized cuboids with query answering potential and selecting the cuboid with the least cost.

2.4.4 OLAP Server Architectures

ROLAP:

- Relational OLAP (ROLAP) servers:
    - Intermediate servers that stand in between a relational back-end server and client front-end tools.
    - 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 back end, implementation of aggregation navigation logic, and additional tools and services.
    - ROLAP technology tends to have greater scalability than MOLAP technology.

MOLAP:

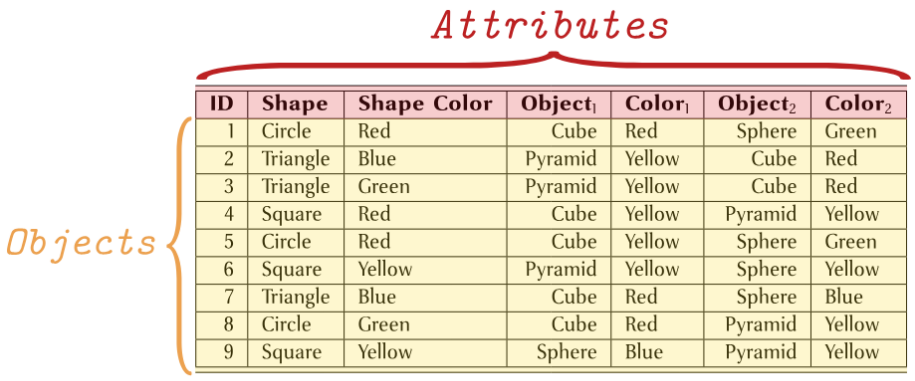
- Multidimensional OLAP (MOLAP) servers:
    - These servers support multidimensional data views through array-based multidimensional storage engines.
    - They map multi-dimensional views directly to data cube array structures.
    - The advantage of using a data cube is that it allows fast indexing 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.
- MOLAP servers adopt a two-level storage representation to handle dense and sparse data sets: Denser subcubes are identified and stored as array structures, whereas sparse subcubes employ compression technology for efficient storage.

HOLAP:

- 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 detailed data to be stored in a relational database, while aggregations are kept in a separate MOLAP store.
- Specialized SQL servers:
  - Provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

3 Data



ID	Shape	Shape Color	$Object_1$	$Color_1$	$Object_2$	$Color_2$
1	Circle	Red	Cube	Red	Sphere	Green
2	Triangle	Blue	Pyramid	Yellow	Cube	Red
3	Triangle	Green	Pyramid	Yellow	Cube	Red
4	Square	Red	Cube	Yellow	Pyramid	Yellow
5	Circle	Red	Cube	Yellow	Sphere	Green
6	Square	Yellow	Pyramid	Yellow	Sphere	Yellow
7	Triangle	Blue	Cube	Red	Sphere	Blue
8	Circle	Green	Cube	Red	Pyramid	Yellow
9	Square	Yellow	Sphere	Blue	Pyramid	Yellow

1. Collection of observations (that are data objects and their attributes).
2. An attribute is a property or characteristic of an object.

- Attribute is also known as variable, field, characteristic, dimension, or feature.
- Examples: eye color of a person, temperature, etc.

3. A collection of attributes describe an object.
   - Object is also known as record, point, case, sample, entity, or instance.

3.1 Attributes

- The type of an attribute depends on which of the following properties/operations it possesses:
  - Distinctness $(=, \neq)$.
  - Order $(>, <)$.
  - Differences are meaningful $(+, -)$.
  - Ratios are meaningful $(\times, \div)$.
- Nominal attribute: distinctness.
- Ordinal attribute: distinctness and order. (Non-uniform)
- Interval attribute: distinctness, order and meaningful differences. (No true zero)
- Ratio attribute: all 4 properties/operations.

3.1.1 Attribute types based on number values:

- Discrete Attribute:
  - Has only a finite or countably infinite set of values.
  - Examples: zip codes, counts, or the set of words in a collection of documents.
  - Often represented as integer variables.
  - Note: binary attributes are a special case of discrete attributes.
- Continuous Attribute:
  - Has real numbers as attribute values.
  - Examples: temperature, height, or weight.
  - Practically, real values can only be measured and represented using a finite number of digits.
  - Often represented as floating-point variables.

3.1.2 Attribute types based on importance:

- Binary variable has either two outcomes: 1 (positive/present) or 0 (negative/absent).
- If there is no preference for which outcome should be coded as 0 and which as 1, the binary variable is called symmetric.
- Example: binary variable for "is evergreen?". A plant has the possible states "loses leaves in winter" and "does not lose leaves in winter."
- Both are equally valuable and carry the same weight when similarity is computed.
- If the outcomes of a binary variable are not equally important, the binary variable is called asymmetric.
- Presence or absence of a relatively rare attribute.
- Example: binary variable for "is color-blind?". Two people being color-blind is an important observation versus two people who are not color-blind.
- The most important outcome is usually coded as 1 (present) and the other is coded as 0 (absent).
- Agreement of two 1's is more significant than the agreement of two 0's.
- Usually, the negative match is treated as irrelevant.

3.2 Data characteristics

We have 3 data characteristics:

1. Dimensionality
2. Sparsity
3. Resolution

3.2.1 Dimensionality:

- Dimensionality (number of attributes):
  - Data with a small number of dimensions tends to be qualitatively different than moderate or high-dimensional data.
  - Curse of Dimensionality: Difficulties associated with analyzing high-dimensional data.
  - Curse of Dimensionality can be overcome with the help of dimensionality reduction techniques.

3.2.2 Sparsity:

- Sparsity:
  - For some data sets (e.g., with asymmetric features) fewer than 1% of the entries are non-zero.
  - Sparsity is an advantage: only non-zero values need to be stored and manipulated.
  - Results in significant savings with respect to computation time and storage.

3.2.3 Resolution:

- Resolution:
  - Patterns in the data depend on the level of resolution.
  - Properties of data are different at different resolutions.
  - Fine Resolution: a pattern may not be visible or may be buried in noise.
  - Coarse Resolution: pattern may disappear.
  - Example: Variations in atmospheric pressure on a scale of hours reflect the movement of storms and other weather systems. On a scale of months, such phenomena are not detectable.

3.3 Data set types

- Record
  - Data Matrix
  - Document Data
  - Transaction Data
- Graph
  - World Wide Web
  - Molecular Structures
- Ordered
  - Spatial Data
  - Temporal Data
  - Sequential Data
  - Genetic Sequence Data

3.3.1 Record data:

- Data mining algorithms assume the data set is a collection of records (data objects).
- Each record consists of a fixed set of data fields (attributes).
- Often there is no explicit relationship among records or data fields.
- Every record (object) has the same set of attributes.
- Record data is usually stored either in flat files or in relational databases.
- Database serves as a convenient place to find records.

Example 1: Transaction data:

T-ID	Items
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Soda, Diaper, Milk

- Each record (transaction) involves a set of items.
- Can be viewed as a set of records whose fields are asymmetric attributes.
- Often attributes are binary (item was purchased / not purchased), can also be discrete or continuous (# items purchased or $ spent on items).
- Grocery Store Example:
  - Set of products purchased by a customer in one transaction.
  - Individual products are items.
  - Also known as market basket data.

Example 2: Data matrix

x-Load	y-Load	Distance	Load	Thickness
10.26	5.27	15.22	27	1.2
12.65	6.25	16.22	22	1.1
13.54	7.23	17.34	23	1.2
14.27	8.43	18.45	25	0.9

- If all attributes are numeric, then the records can be thought of as points (vectors) in a multidimensional space.
- That is, a $m \times n$ matrix, where there are m rows (data objects) and n columns (numeric attributes).
- Data matrix is the standard data format for most statistical data.

Example 3: Document data

	$\mathbf{word}_1$	$\mathbf{word}_2$	$\mathbf{word}_3$	...	$\mathbf{word}_{\|\mathcal{V}\|}$
$\mathbf{document}_1$	10	0	0	...	5
$\mathbf{document}_2$	1	2	0	...	0
⋮	⋮	⋮	⋮	...	⋮
$\mathbf{document}_n$	0	1	0	...	0

- Each document becomes a term vector.
- Each term is a component (attribute) of the vector.
- The value of each component is the number of times the corresponding term occurs in the document.
- Example of sparse data matrix.

3.3.2 Graph data:

- Traditional paradigm in data analysis assumes independence between data instances.
- Often data instances (nodes) may be connected or linked together via relationships (edges).
- What emerges is a network or graph.
- Both the nodes and edges in the graph may have several attributes that may be numerical or categorical, or even more complex (e.g., time series data).
- Examples:
  1. World Wide Web (with its Web pages and hyperlinks)
  2. Social Networks (wikis, blogs, tweets etc.)
  3. Semantic Networks (ontologies)
  4. Citation Networks for scientific literature
  5. Biological Networks (protein interactions, gene regulation networks, metabolic pathways)
- According to Tan et al. graph data can be of two types:

1. Data with Relationships among Objects.
2. Data with Objects that are Graphs.
   - Example: chemical compound structures can be represented by a graph.
   - Substructure Mining: determine which substructures occur frequently in a set of compounds to ascertain chemical properties (e.g., melting point).

3.3.3 Ordered data:

- For some types of data, the attributes have relationships that involve order in time or space.
  - Sequential Data
  - Sequence Data
  - Time Series Data
  - Spatial Data

Example: Sequential data Sequential data also referred to as temporal data has that each record has a time associated with it:

Time	Customer	Items Purchased
t1	C1	{A,B}
t2	C3	{A,C}
t2	C1	{C,D}
t3	C2	{A,D}
t4	C2	{E}
t5	C1	{A,E}

Customer	Time and Items Purchased
C1	⟨ (t1: {A,B}), (t2:{C,D}), (t5:{A,E})⟩
C2	⟨ (t3: {A,D}), (t4: {E}) ⟩
C3	⟨ (t2: {A,C}) ⟩

Example: Sequence data Sequence of individual entities, such as a sequence of words or letters. Similar to sequential data, except there are no time stamps; there are positions in an ordered sequence:

```
  1 ATTAAAGGTT TATACCTTCC CAGGTAACAA ACCAACCAAC TTTCGATCTC TTGTAGATCT
 61 GTTCTCTAAA CGAAATTTAA AATCTGTGTG GCTGTCACTC GGCTGCATGC TTAGTGCACT
121 CACGCAGTAT AATTAATAAC TAATTACTGT CGTTGACAGG ACACGAGTAA CTCGTCTATC
181 TTCTGCAGGC TGCTTACGGT TTCGTCCGTG TTGCAGCCGA TCATCAGCAC ATCTAGGTTT
241 CGTCCGGGTG TGACCGAAAG GTAAGATGGA GAGCCTTGTC CCTGGTTTCA ACGAGAAAAC
301 ACACGTCCAA CTCAGTTTGC CTGTTTTACA GGTTCGCGAC GTGCTCGTAC GTGGCTTTGG
361 AGACTCCGTG GAGGAGGTCT TATCAGAGGC ACGTCAACAT CTTAAAGATG GCACTTGTGG
421 CTTAGTAGAA GTTGAAAAAG GCGTTTTGCC TCAACTTGAA CAGCCCTATG TGTTCATCAA
481 ACGTTCGGAT GCTCGAACTG CACCTCATGG TCATGTTATG GTTGAGCTGG TAGCAGAACT
```

Example: Time series data Time series data is a special type of sequential data in which each record is a time series, i.e. a series of measurements taken

over time. We have temporal autocorrelation if two measurements are close in time, then the values of those measurements are often very similar. For illustration: consider the price of a stock over time.

Example: Spatial data Some objects have spatial attributes, such as positions or areas, as well as other types of attributes. We have spatial autocorrelation when objects that are physically close tend to be similar in other ways as well. For spatial data visualization: think of an image of the weather over some area, where you for example show temperature, pressure and probability of rain.

3.3.4 Data analysis perspectives:

We can view data analysis from different perspectives:

1. Database perspective
2. Geometric and algebraic perspective
3. Probablistic perspective

3.4 Data quality

- Poor data quality negatively affects many data processing efforts.
- Data mining example:

Examples of data quality problems:

- Noise and outliers
- Wrong data
- Fake data
- Missing values
- Duplicate data

3.4.1 Noise:

- Noise is the random component of a measurement error.
- It may involve the distortion of a value or the addition of spurious objects.
- The term noise is often used in connection with data that has a spatial or temporal component.
- In such cases, techniques from signal or image processing can frequently be used to reduce noise and thus, help to discover patterns (signals) that might be "lost in the noise."
- Robust algorithms: data mining methods that produce acceptable results even when noise is present.

3.4.2 Outliers:

- Outliers are:
  1. Data objects that have characteristics different from most other objects in the data set.
  2. Values of an attribute that are unusual with respect to the typical values for that attribute.

3.4.3 Outliers vs noise:

Outliers can be legitimate data objects or values and may sometimes be of interest.

3.4.4 Missing values:

- Reasons for missing values
  - Information is not collected (e.g., people decline to give their age and weight).
  - Attributes may not be applicable to all cases (e.g., annual income is not applicable to children).
  - Handling missing values.
    * Eliminate data objects or variables.
    * Estimate missing values (e.g., time series of temperature or census results).
    * Ignore the missing value during analysis.

3.4.5 Duplicate data:

- Data set may include data objects that are duplicates, or almost duplicates of one another.
  - Major issue when merging data from heterogeneous sources.
- Example: same person with multiple email addresses.
- Data Cleaning: process of dealing with duplicate data issues.
- When should duplicate data not be removed?

3.5 Data preprocessing

- Data Preprocessing: processing the data so as to improve the data mining analysis with respect to time, cost, and quality.
- This done by: selecting data objects and attributes for the analysis or creating/changing the attributes.

- Typical data preprocessing steps:
  - Aggregation
  - Sampling
  - Discretization and Binarization
  - Attribute Transformation
  - Dimensionality Reduction
  - Feature Subset Selection
  - Feature Creation

3.5.1 Aggregation:

- Combining two or more attributes (or objects) into a single attribute (or object).
- Purpose:
  - Data Reduction: reduce the number of attributes or objects.
  - Change of scale:
    - Cities aggregated into regions, states, countries, etc.
    - Days aggregated into weeks, months, or years.
  - More stable data: aggregated data tends to have less variability.

3.5.2 Sampling:

- Sampling is the main technique employed for data reduction.
- It is often used for both the preliminary investigation of the data and the final data analysis.
- Statisticians often sample because obtaining the entire set of data of interest is too expensive or time consuming.
- Sampling is typically used in data mining because processing the entire set of data of interest is too expensive or time consuming.
- The key principle for effective sampling is the following:
  - Using a sample will work almost as well as using the entire data set, if the sample is representative.
  - A sample is representative if it has approximately the same properties (of interest) as the original set of data.
- Sampling Methods:
  - Simple Random Sampling.

- Stratified Sampling.
- Progressive Sampling.

Simple random sampling (SRS):

- For this type of sampling, there is an equal probability of selecting any particular item.
- Two variations on SRS:
  1. Sampling without replacement: as each item is selected, it is removed from the set of all objects that together constitute the population.
  2. Sampling with replacement: objects are not removed from the population as they are selected for the sample.
- In sampling with replacement, the same object can be picked more than once.
- The samples produced by the two methods are not much different when samples are relatively small compared to the data set size, but sampling with replacement is simpler to analyze since the probability of selecting any object remains constant during the sampling process.

Stratified sampling:

- Problems with SRS:
  - When the population consists of different classes of objects, with widely different numbers of objects per class, then simple random sampling can fail to adequately represent those types of objects that are less frequent.
  - This can cause problems when the analysis requires proper representation of all object classes.
  - For example, when building classification models for rare classes, it is critical that the rare classes be adequately represented in the sample.
- Hence, a sampling scheme that can accommodate differing frequencies for the items of interest is needed.
- Stratified Sampling: which starts with pre-specified groups of objects, is such an approach.
- In the simplest version, equal numbers of objects are drawn from each group even though the groups are of different sizes. In another variation, the number of objects drawn from each group is proportional to the size of that group.

Progressive sampling:

- Progressive Sampling Schemes: these approaches start with a small sample, and then increase the sample size until a sample of sufficient size has been obtained.
- Eliminates the need to determine the correct sample size initially.
- However, it requires that there be a way to evaluate the sample to judge if it is large enough.
- This can be done with respect to the application at hand.
- Example: a progressive sampling is used to learn a predictive model.
- The accuracy of predictive models increases as the sample size increases up to some point.
- After which the increase in accuracy levels off.
- We want to stop increasing the sample size at this leveling-off point.

3.5.3 Curse of dimensionality:

The term curse of dimensionality was coined by Bellman (1961) to indicate that the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with the number of variables, that is, with the dimensionality of the function.

- When dimensionality increases, data becomes increasingly sparse in the space that it occupies.
- Definitions of density and distance between points, which are critical for clustering and outlier detection, become less meaningful.

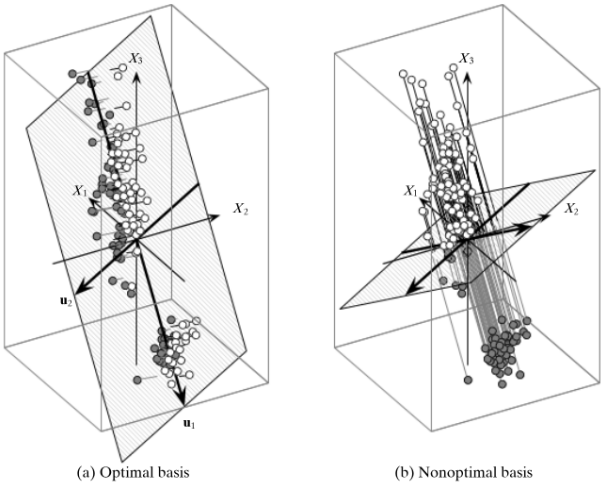
Dimensionality reduction:

- Purpose:
  - Avoid curse of dimensionality.
  - Reduce amount of time and memory required by data mining algorithms.
  - Allow data to be more easily visualized.
  - May help to eliminate irrelevant features or reduce noise.
- Techniques:
  - Principal Components Analysis (PCA).
  - Singular Value Decomposition.
  - Others: supervised and non-linear techniques.

Principal components analysis (PCA):

- Principal Components Analysis (PCA) is a linear algebra technique for continuous attributes that finds new attributes (principal components) that are:
  1. linear combinations of the original attributes,
  2. orthogonal (perpendicular) to each other, and
  3. capture the maximum amount of variation in the data.

Illustrated:



(a) Optimal basis (b) Nonoptimal basis

3.5.4 Visualizing higher dimensions:

The easiest way to visualize objects in higher dimensions is by projecting the object on pairs (2-D) of dimensions selected from the total dimensions of the objects. For n-dimensional data we get $nC2$ pairwise combinations.

3.5.5 Feature selection:

- Another way to reduce dimensionality of data.
- Redundant features:
  - Duplicate much or all of the information contained in one or more other attributes.
  - Example: purchase price of a product and the amount of sales tax paid.
- Irrelevant features:
  - Contain no information that is useful for the data mining task at hand.

  - Example: students' ID is often irrelevant to the task of predicting students' GPA.
  - Many techniques developed, especially for classification.

Ideal approach:

- Irrelevant and redundant attributes can be eliminated by using common sense domain knowledge.
- A more systematic approach is however required.
- Ideal Approach:
  - Enumerate all possible subsets of features.
  - For each subset obtain performance results from the data mining algorithm being used.
- Pros: reflects the objective and bias of the data mining algorithm that will eventually be used.
- Cons: number of subsets involving n attributes is 2^n

Other approaches:

- Embedded Approaches: algorithm itself decides which attributes to use and which to ignore (e.g., decision trees).
- Filter Approaches: features are selected independently and separately from the algorithm being run (e.g., where pairwise correlation is as low as possible).
- Wrapper Approaches: same as Ideal Approach but without enumerating all possible subsets.

3.5.6 Feature creation:

- Create new attributes that can capture the important information in a data set much more efficiently than the original attributes.
- Three general methodologies:
  - Feature extraction.
    - Example: extracting edges from images.
  - Feature construction.
    - Example: dividing mass by volume to get density.
  - Mapping data to new space.
    - Example: Fourier and wavelet analysis.

3.5.7 Discretization and binarization:

- Certain classification algorithms require data in the form of categorical attributes.
- Algorithms that find association patterns require that the data be in the form of binary attributes.
- **Discretization**: transform a continuous attribute into a categorical attribute.
- **Binarization**: Both continuous and discrete attributes may need to be transformed into one or more binary attributes.

Binaraization:

- A simple approach to binarize m categorical values:
  1. Assign a unique integer in the interval $[0, m-1]$. Maintain order during assignment if the attribute is ordinal.
  2. Convert each of these m integers to a binary number.
  3. $n = \lceil \log_2(m) \rceil$ binary attributes are required to represent these integers.
- Problems: unintended correlations introduced.
- For association problems asymmetrical attributes are required.
- Therefore necessary to introduce one binary attribute for each categorical value. This is known as one-hot encoding

Discretization:

- Transformation of a continuous attribute to a categorical attribute involves two subtasks:
  1. **First step**: continuous attribute are sorted and then are divided into n intervals by specifying $n-1$ split points.
  2. **Second step**: all the values in one interval are mapped to the same categorical value.
- **Unsupervised Approaches**: label associated with the objects are not used.
  - Example: equal width, equal frequency / equal depth, and k-means.
- **Supervised Approaches**: labels associated with the objects are used.
  - Example: a simple approach for partitioning a continuous attribute starts by bisecting the initial values so that the resulting two intervals give minimum entropy.

3.5.8 Attribute transformations:

An attribute transformation is a function that maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identified with one of the new values.

Using Simple Functions:

- Using transformations such as x^k, $\log(x)$, e^x, $\sqrt{x}$, $1/x$, or $|x|$.
- Variable transformations should be applied with caution since they change the nature of the data.
- Does the transformation apply to all values?
- Especially negative values and 0?
- What is the effect of the transformation on the values between 0 and 1?

Normalization or Standardization:

- Normalization refers to various techniques to adjust to differences among attributes in terms of frequency of occurrence, mean, variance, and range.
- Take out unwanted, common signal, e.g., seasonality.
- In statistics, standardization refers to subtracting off the means and dividing by the standard deviation.
- $$x' = \frac{x - \bar{x}}{s_x}$$

3.5.9 Similarity and Dissimilarity:

- Similarity between two objects:
  - A numerical measure indicating degree to which the two objects are alike.
  - Usually non-negative and are often between 0 (no similarity) and 1 (complete similarity).
- Dissimilarity between two objects:
  - A numerical measure of the degree to which the two objects are different.
  - Dissimilarities are lower for more similar pairs of objects.
  - Dissimilarities sometimes fall in the interval [0,1], but it is also common for them to range from 0 to ∞.
  - Distances, which are dissimilarities with certain properties.

The following table shows the similarity and dissimilarity between two objects x and y, with respect to a single, simple attribute:

Attribute Type	**Dissimilarity**	**Similarity**
Nominal	$d = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$	$s = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$
Ordinal	$d = \lvert x - y \rvert / (n-1)$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - d$
Interval or Ratio	$d = \lvert x - y \rvert$	$s = -d$, $s = \frac{1}{1+d}$, $s = e^{-d}$, $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$

Distances (Dissimilarities with certain properties):

- Minkowski Distance — generalization of Euclidean distance:

$$d(x, y) = \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{1/r} \tag{1}$$

- where, r is a parameter, n is the number of dimensions (attributes) and x_k and y_k are, respectively, the k^{th} attributes (components) or data objects x and y.
- $r = 1$. City block (Manhattan, taxicab, L1 norm) distance.
  - A common example of this for binary vectors is the Hamming distance, which is just the number of bits that are different between two binary vectors.
- $r = 2$. Euclidean distance.
- $r \to \infty$. "supremum" ($L_{\max}$ norm, L_∞ norm, Uniform norm, and Chebyshev) distance.
  - This is the **maximum difference between any component of the vectors**.
  - That is,

$$d(x, y) = \lim_{r \to \infty} \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{1/r} = \max_k |x_k - y_k|. \tag{2}$$

- Do not confuse r with n, i.e., all these distances are defined for all numbers of dimensions.

Common properties of distance:

- **Distances**, such as the Euclidean distance, have some **well known properties**.

- **Positivity**: $d(x, y) \geq 0$ for all x and y and $d(x, y) = 0$ if and only if $x = y$.
- **Symmetry**: $d(x, y) = d(y, x)$ for all x and y.
- **Triangle Inequality**: $d(x, y) \leq d(x, z) + d(z, y)$ for all points x, y, and z.

- A **distance** that satisfies these properties is a **metric**.

Similarity between binary vectors:

- Common situation is that objects, x and y, have only binary attributes.
- Compute similarities using the following quantities:
  - f_{00} = the number of attributes where x was 0 and y was 0.
  - f_{01} = the number of attributes where x was 0 and y was 1.
  - f_{10} = the number of attributes where x was 1 and y was 0.
  - f_{11} = the number of attributes where x was 1 and y was 1.
- **Simple Matching Coefficient**:

$$SMC = \frac{\text{number of matches}}{\text{number of attributes}} = \frac{f_{11} + f_{00}}{f_{00} + f_{01} + f_{10} + f_{11}}. \tag{5}$$

- **Jaccard Coefficient**:

$$J = \frac{\text{number of 11 matches}}{\text{number of non-zero attributes}} = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}. \tag{6}$$

Cosine similarity:

- If x and y are two document vectors, then:

$$\cos(x, y) = \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|} = \frac{\sum_{k=1}^{n} x_k \cdot y_k}{\sqrt{\sum_{k=1}^{n} x_k^2} \cdot \sqrt{\sum_{k=1}^{n} y_k^2}}.$$

- where, $\langle x, y \rangle$ indicates the inner product or vector dot product of vectors, x and y, and $\|x\|$ is the length of vector x.

3.5.10 Correlation:

- **Correlation between two data objects** that have binary or continuous variables is a *measure of the linear relationship between the attributes of the objects.*
- **Perfect Correlation**: correlation is always in the range -1 to 1. A correlation of 1 (-1) means that x and y have a perfect positive (negative) linear relationship.

- That is, $x_k = a \cdot y_k + b$, where a and b are constants.

Pearssons correlation coefficient:

- **Pearson's Correlation Coefficient** between two data objects, x and y is defined by the following equation:

$$\text{corr}(x, y) = \frac{\text{covariance}(x, y)}{\text{std\_dev}(x) \cdot \text{std\_dev}(y)} = \frac{s_{xy}}{s_x \cdot s_y}. \tag{3}$$

- where, the standard statistical notation and definitions are:

$$\text{covariance}(x, y) = s_{xy} = \frac{1}{n-1} \cdot \sum_{k=1}^{n} (x_k - \bar{x})(y_k - \bar{y}). \tag{4}$$

$$\text{std\_dev}(x) = s_x = \sqrt{\frac{1}{n-1} \cdot \sum_{k=1}^{n} (x_k - \bar{x})^2}. \tag{5}$$

$$\bar{x} = \frac{1}{n} \sum_{k=1}^{n} x_k \text{ is the mean of } x. \tag{6}$$

4 Association Rule Mining

4.1 Introduction

Given a set of transactions, find rules that will predict the occurence of other items in the transaction.

T-ID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Cola
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Cola

Table 1: Transactions and Items

Examples of association rules:

- $\{Diaper\} \rightarrow \{Beer\}$
- $\{Milk, Bread\} \rightarrow \{Eggs, Cola\}$
- $\{Beer, Bread\} \rightarrow \{Milk\}$

Note that implication means co-occurence and not causality.

4.1.1 Main points

1. Association analysis: discovering interesting relationsships hidden in large data sets.
2. Uncovered relationsships can be represented in the form of assocaition rules of sets of frequent items
3. When applied to retail transactions, commonly known as market basket analysis

4.2 Definitions

4.2.1 Binary representation:

Presence of an item in a transaction is often considered more important than its absence, therefore an item is represented as an asymmetric binary variable.

Example of transforming transaction data table to a binary representation:

T-ID	Bread	Milk	Diapers	Beers	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Figure 12: Binary Representation of Transactions

4.2.2 Notation:

- **Set of all transactions:** $T = \{t_1, t_2, \ldots, t_N\}$.
- **Set of all Items:** $I = \{i_1, i_2, \ldots, i_d\}$.
- **Each transaction t_i contains a subset of items chosen from I.**
- **Itemset:** a collection of zero or more items.
- k–**Itemset:** itemset containing k items.
  Example: $\{Beer, Diapers, Milk\}$ is a 3-itemset.
- **Support Count** ($\sigma(X)$): number of transactions that contain a particular itemset.
$$\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}| \tag{7}$$
- **Example:** $\sigma(\{Beer, Diapers, Milk\}) = 2$.

- **Association Rule**: is an implication expression of the form $X \rightarrow Y$, where, $X \cap Y = \emptyset$.
- **Support** $s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$.
- **Example:** $s(\{Milk, Diaper\} \rightarrow \{Beer\}) = \frac{\sigma(\{Milk, Diaper, Beer\})}{|T|} = \frac{2}{5} = 0.4$
- **Confidence** $c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$.
- **Example:** $c(\{Milk, Diaper\} \rightarrow \{Beer\}) = \frac{\sigma(\{Milk, Diaper, Beer\})}{\sigma(\{Milk, Diaper\})} = \frac{2}{3} = 0.\overline{6}$

4.2.3 Problem definition:

Given a set of transactions T, the goal of association rule mining is to find all rules fulfulling the requirements:

- Support $\geq$ minimum support threshold
- Confidence $\geq$ minimum confidence threshold

Minimum support (minsup) and confidence (minconf) represent thresholds on interesting association rules.

4.2.4 Why support and confidence?:

Support is an important measure because a rule that has very low support might occur simply by chance. Also, from a business perspective a low support rule is unlikely to be interesting because it might not be profitable to promote items that customers seldom buy together. For these reasons, we are interested in finding rules whose support is greater than some user-defined threshold. Support also has a desirable property that can be exploited for the efficient discovery of association rules. Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence also provides an estimate of the conditional probability of Y given X.

4.3 Brute-Force approach:

1. List all possible association rules
2. compute the support and confidence for each rule
3. prune results that do not satisfy the minsup and minconf thresholds

Brute-force is computationally prohibitive.

4.3.1 Analysis

- Consider, we have d items
- total number of itemsets 2^d
- Total number of possible association rules: $R = 3^d - 2^d + 1$

4.4 First step to improvement:

Rules originating from the same itemset have identical support but can have different confidence. Thus, we may decouple the support and confidence requirements. Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. Frequent itemset generation; whose objective is to find all the itemsets that satisfy minsup threshold
2. Rule generation; whose objective is to extract all high confidence rules from frequent itemsets found in the previous step. These rules are called strong rules.

4.5 Frequent Itemset Generation

A lattice structure can be used to enumerate the list of all possible itemsets. The figure below shows an itemset lattice for I = {a, b, c, d, e}. In general, a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.

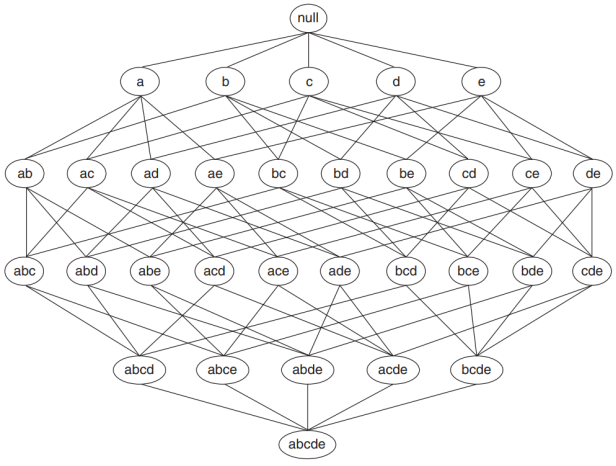


Figure 13: Itemset lattice

4.5.1 Brute-force:

- Each itemset in the lattice is a candidate frequent itemset.
- Count the support of each candidate by scanning the database.
- Match each transaction against every candidate.
- Complexity — $\mathcal{O}(NMw)$ — very expensive since $M = 2^d$.

N is the number of transactions, M is the list of candidates and w is the width of the transactions.

There are 3 main approaches for reducing the computational complexity of frequent itemset generation:

1. **Reduce the number of candidate itemsets** (M). The *Apriori* principle, described in the next section, is an effective way to eliminate some of the candidate itemsets without counting their support values.
2. **Reduce the number of comparisons**. Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set.
3. **Reduce the number of transactions** (N). As the size of candidate itemsets increases, fewer transactions will be supported by the itemsets.

4.5.2 The Apriori Principle:

If an itemset is frequent then all of its subsets must ble frequent.

$$\forall X, Y : (X \subseteq Y) \implies s(X) \geq s(Y) \tag{8}$$

Conversely, if an itemset is infrequent then all if its supersets must be infrequent.

4.5.3 Anti-monotone property:

A measure f posses the anti-monotone property if for every itemset X that is a proper subset of itemset Y, i.e. $X \subset Y$, we have $f(Y) \leq f(X)$.
In layman terms; support of an itemset never exceeds the support of its subsets.

4.5.4 Apriori Algorithm:

- F_k: frequent k-itemsets.
- L_k: candidate k-itemsets.
- Algorithm

1. Let $k = 1$
2. Generate $F_1 = \{\text{frequent 1-itemsets}\}$
3. Repeat until F_k is empty
    1. Candidate Generation: Generate L_{k+1} from F_k.
    2. Pruning: Prune candidate itemsets in L_{k+1} containing subsets of length k that are infrequent.
    3. Support Counting: Count the support of each candidate in L_{k+1} by scanning the DB.
    4. Candidate Elimination: Eliminate candidates in L_{k+1} that are infrequent, leaving only those that are frequent $\implies$ F_{k+1}.

4.5.5 Candidate generation step: $F_{k-1} \times F_{k-1}$ method:

Merge two frequent (k-1)-itemsets if their first (k-2) items are identical.

4.5.6 Computational complexity: Apriori Algorithm

- **Choice of minimum support threshold:**
  - Lowering support threshold results in more frequent itemsets.
  - This may increase number of candidates and max length of frequent itemsets.
- **Dimensionality (number of items) of the data set.**
  - More space is needed to store support count of itemsets.
  - If number of frequent itemsets also increases, both computation and I/O costs may also increase.
- **Size of database.**
  - Run time of algorithm increases with number of transactions.
- **Average transaction width.**
  - Transaction width increases the max length of frequent itemsets.
  - Number of subsets in a transaction increases with its width, increasing computation time for support counting.

4.5.7 Rule Generation:

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

- Example: if $\{A, B, C, D\}$ is a frequent itemset, candidate rules:

$$\begin{array}{cccc} ABC \rightarrow D & ABD \rightarrow C & ACD \rightarrow B & BCD \rightarrow A \\ A \rightarrow BCD & B \rightarrow ACD & C \rightarrow ABD & D \rightarrow ABC \\ AB \rightarrow CD & AC \rightarrow BD & AD \rightarrow BC & BC \rightarrow AD \\ BD \rightarrow AC & CD \rightarrow AB & & \end{array}$$

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

- In general, confidence does not have an anti-monotone property:

$$c(ABC \rightarrow D) \text{ can be larger or smaller than } c(AB \rightarrow D).$$

- But confidence of rules generated from the same itemset has an anti-monotone property.

- Example: suppose $\{A, B, C, D\}$ is a frequent 4-itemset:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD).$$

- Confidence is anti-monotone w.r.t. number of items on the RHS of the rule.

- The Apriori algorithm uses a level-wise approach for generating association rules.

- Each level corresponds to the number of items that belong to the rule consequent.

- Initially, extract all high confidence rules that have one item in rule consequent.

- These rules are then joined to generate new candidate rules.

- Example: if $\{ACD\} \rightarrow \{B\}$ and $\{ABD\} \rightarrow \{C\}$ are high confidence rules, then the candidate rule $\{AD\} \rightarrow \{BC\}$ is generated by merging the consequents of both rules.

- If any node in the lattice has low confidence, entire subgraph spanned by the node can be pruned immediately.

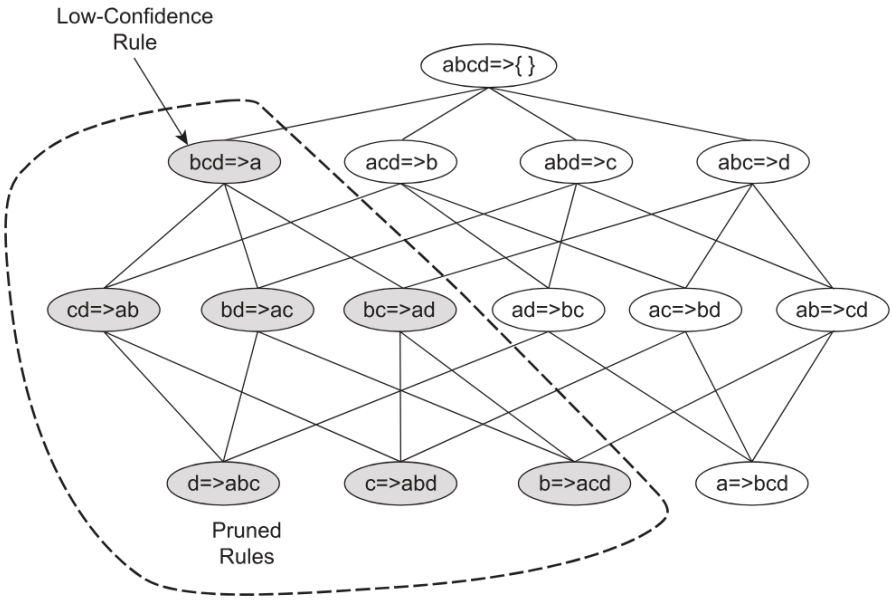


Figure 14: Pruning of association rules using the confidence measure.

4.6 Compact representation of frequent itemsets

In practice, the number of frequent itemsets produced from a transaction data set can be very large. It is useful to identify a small representative set of frequent itemsets from which all other frequent itemsets can be derived. The reason we can do this is because some itemsets are redundant as they have the same support as their supersets. Hence, we need a more compact representation. Two such representations are presented in this section in the form of maximal and closed frequent itemsets.

4.6.1 Maximal frequent itemsets:

A frequent itemset is maximal if non of its immediate supersets are frequent.

Maximal frequent itemsets effectively provide a compact representation of frequent itemsets. In other words, they form the smallest set of itemsets from which all frequent itemsets can be derieved.

4.6.2 Closed itemsets:

An itemset X is closed if none of its immediate supersets has exactly the same support count as X.

Closed itemsets provide a minimal representation of all itemsets without losing the support information.

4.6.3 Closed frequent itemsets

An itemset is a closed frequent itemset if it is closed an its support is greater than or equal to minsup.

4.6.4 Algorithm: Support counting using closed frequent itemsets:

Algorithm 1 Calculation of Support for All Frequent Itemsets

1: Let C denote the set of closed frequent itemsets and F denote the set of all frequent itemsets.
2: Let $k_{\max}$ denote the maximum size of closed frequent itemsets
3: $F_{k_{\max}} = \{f | f \in C, |f| = k_{\max}\}$ {Find all frequent itemsets of size $k_{\max}$.}
4: **for** $k = k_{\max} - 1$ down to 1 **do**
5: $F_k = \{f | f \in F, |f| = k\}$ {Find all frequent itemsets of size k.}
6: **for** each $f \in F_k$ **do**
7: **if** $f \notin C$ **then**
8: $f.\text{support} = \max\{f'.\text{support} | f' \in F_{k+1}, f \subseteq f'\}$
9: **end if**
10: **end for**
11: **end for**

4.6.5 Relations between compact representations:

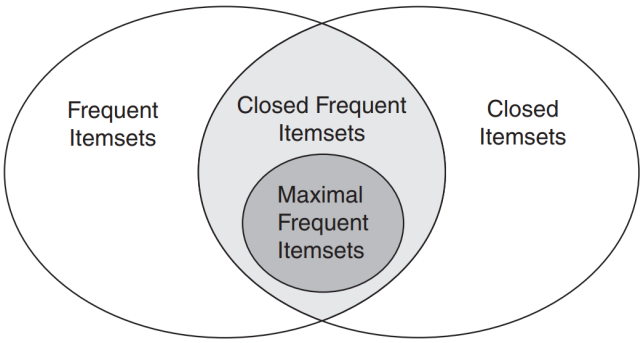


Figure 15: Relationships among frequent, closed, closed frequent, and maximal frequent itemsets.

Note that all maximal frequent itemsets are closed because none of the maximal frequent itemsets can have the same count as their immediate supersets.

4.6.6 Benefits and Drawbacks of Maximal Frequent Itemsets and Closed Frequent Itemsets

Maximal Frequent Itemsets:

Benefits:

- **Compact Representation**: Maximal frequent itemsets provide a highly compact representation as they only include the largest itemsets without any frequent supersets.
- **Efficient Storage**: Reduces storage requirements since fewer itemsets are stored compared to all frequent itemsets.

Drawbacks:

- **Loss of Support Information**: Maximal frequent itemsets do not retain the support counts of their subsets, which can be crucial for some analyses.
- **Additional Processing**: To determine the support of a subset, an additional pass over the data or recomputation may be required, increasing computational overhead.

Closed Frequent Itemsets:

Benefits:

- **Retains Support Information**: Closed frequent itemsets retain the exact support counts for each itemset, which can be valuable for detailed analysis and decision-making.
- **Efficient in Some Cases**: Can be more efficient to use when the focus is on support counts and their applications, such as association rule mining.

Drawbacks:

- **Larger Representation**: While more compact than the full set of frequent itemsets, closed frequent itemsets still include more itemsets than maximal frequent itemsets, leading to higher storage requirements.
- **Complexity**: The process of finding closed frequent itemsets can be more complex than finding maximal frequent itemsets.

4.7 Drawbacks of Apriori Algorithm

- **Apriori candidate generate-and-test method** significantly reduces the size of candidate sets, leading to good performance gain.
- However, it can suffer from two nontrivial costs:
    1. Need to generate a huge number of candidate sets.
        - For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.

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

4.7.1 Representation of transaction data set:

There are many ways to represent a transaction data set. The choice of representation can affect the I/O costs incurred when computing the support of candidate itemsets. The figure below shows two different ways of representing market basket transactions. The representation on the left is called a horizontal data layout, which is adopted by many association rule mining algorithms, including Apriori. Another possibility is to store the list of transaction identifiers (TID-list) associated with each item. Such a representation is known as the vertical data layout. The support for each candidate itemset is obtained by intersecting the TID-lists of its subset items. The length of the TID-lists shrinks as we progress to larger sized itemsets. However, one problem with this approach is that the initial set of TID-lists might be too large to fit into main memory, thus requiring more sophisticated techniques to compress the TID-lists.

Horizontal Data Layout

TID	Items
1	a,b,e
2	b,c,d
3	c,e
4	a,c,d
5	a,b,c,d
6	a,e
7	a,b
8	a,b,c
9	a,c,d
10	b

Vertical Data Layout

a	b	c	d	e
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

Figure 16: Horisontal and vertical data layout

4.8 FP-Growth algorithm

- **FP-Growth mines the complete set of frequent itemsets** without such a costly candidate generation process.
- FP-growth, which adopts a divide-and-conquer strategy as follows:
    1. Compress the database representing frequent items into a frequent pattern tree, or FP-tree, which retains the itemset association information.
    2. Divides the compressed database into a set of conditional databases (projected trees) each associated with one "pattern fragment,"

and mines each database separately. For each "pattern fragment," only its associated data sets need to be examined.

- **FP-Growth substantially reduce the size of the data sets** to be searched, along with the "growth" of patterns being examined.

4.8.1 Step-by-step guide: Constructing FP-tree:

1. Obtain the transaction database.
2. Sort the items in the transaction database by their support. Remember to discard the items that do not meet the minimum support threshold
3. Sort the order of the items in each transaction of the database with descending order of support.
4. Construct the FP-Tree (call it R) step-by-step by adding each of the transactions (where items in them are reordered with descending support).
    - The FP-tree (R) serves as an index in lieu of the original database.
5. Given the FP-Tree (R), projected (conditional) FP-trees (databases) are built for each frequent item i in R in increasing order of support.

4.8.2 Step-by-step guide: Finding frequent itemsets from FP-tree

1. Intitalize the FP-tree from the database D
2. List all items an their support counts. We do this in a table with decreasing order (should be the same as the one we used for the construction of FP-tree).
3. Process Each Item in Increasing Order of Support:
   For each item i:
    (a) Find conditional pattern base for i:
        - Traverse the FP-tree to find all paths that contain item i
        - Extract the prefix paths that lead to i, excluding i itself. Order the conditional pattern base in increasing order.

        Construct the conditional FP-tree for item i:
        - Using the conditional pattern base, construct a conditional FP-tree R_i. This is done just as for the creating the regular FP-tree, where each pattern is tantamount to a transaction for the FP-tree.
        - Remove infrequent items with support count less than minsup.

        Generate frequent itemsets for item i:

- Combine the current prefix with all frequent itemsets found in the conditional FP-tree.
- If the conditional FP-tree R_i is a single path, generate all subsets off the path and add to the set of frequent itemsets F. We use the total number of occurences of our items counted from the conditional pattern base as our count.
- If the conditional FP-tree R_i is not a single path, we create subsets of each elements with the current prefix (i.e. 2-itemsets). Then, due to the recursive call, we have to analyse all these subsets as prefixes in the same way as for a regular prefix. Our method of finding conditional pattern bases etc. is then based on the conditional FP-tree we found for the prefix that lead to the recursive call. When all of these subsets have been mined, we move onto the next item.

4.8.3 Pseudocode:

Algorithm 8.5: Algorithm FPGROWTH

```
// Initial Call:  R ← FP-tree(D),  P ← ∅,  F ← ∅
FPGROWTH (R, P, F, minsup):
Remove infrequent items from R
if ISPATH(R) then // insert subsets of  R into F
    foreach Y ⊆ R do
        X ← P ∪ Y
        sup(X) ← min_{x∈Y}{cnt(x)}
        F ← F ∪ {(X, sup(X))}
else  // process projected FP-trees for each frequent item  i
    foreach i ∈ R in increasing order of sup(i) do
        X ← P ∪ {i}
        sup(X) ← sup(i) // sum of  cnt(i) for all nodes labeled  i
        F ← F ∪ {(X, sup(X))}
        R_X ← ∅ // projected FP-tree for  X
        foreach path ∈ PATHFROMROOT(i) do
            cnt(i) ← count of i in path
            Insert path, excluding i, into FP-tree R_X with count cnt(i)
        if R_X ≠ ∅ then FPGROWTH (R_X, X, F, minsup)
```

Figure 17: Entire algorithm for frequent itemset generation by FP-trees

4.9 ECLAT-algorithm:

- Advantage: Very fast support counting
- Disadvantage: Intermediate tid-lists may become too large for memory

4.9.1 Algorithm:

- For each item, store a list of transaction ID-s (TIDS) (For the actual algorithm this is done as a vertical data layout); For employing it on

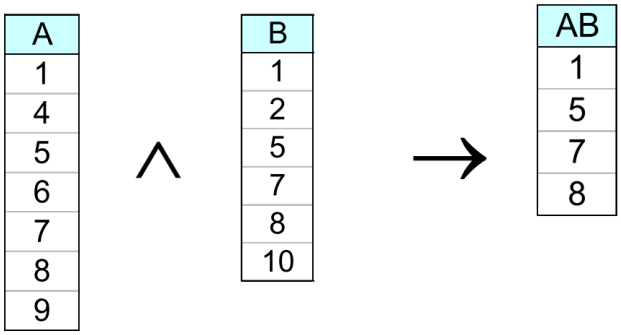


Figure 18: Candiate generation process for ECLAT algorithm

paper, perhaps prudent to use boolean matrix for ease of computation

- Determine support of any k-itemset by intersection tid-lists of two of its (k-1) subsets
- 3 traversal approaches; top-down, bottum-up and hybrid.

4.10 Comparing association rule algorithms:

- **Apriori**:
  - Best suited for simpler, smaller datasets due to its straightforward implementation and ease of understanding.
  - It uses a breadth-first search (BFS) strategy and generates candidate itemsets by joining previously found frequent itemsets. However, this can lead to a large number of candidate sets and multiple database scans, making it less efficient for larger datasets.
  - Ideal when memory resources are limited since it does not require complex data structures.
- **FP-tree (Frequent Pattern Tree)**:
  - Efficient for larger datasets, especially when the data contains many frequent patterns. It utilizes a tree structure to compress the dataset, which helps in reducing the size of the database and the number of scans required.
  - The algorithm avoids the costly candidate generation process by building a compact data structure that represents the itemset frequencies.
  - Generally requires more memory than Apriori but is faster due to its depth-first search (DFS) strategy and reduced database scans.

- **ECLAT (Equivalence Class Transformation)**:
  - Particularly effective for dense datasets where many items are frequently occurring together. It uses a vertical data format, representing the database with transaction ID sets (TID sets) for each item, which allows efficient support counting through intersection operations.
  - ECLAT can quickly compute the support of itemsets by intersecting TID sets, making it efficient for datasets with dense itemsets.
  - The memory requirement can be high as it needs to store TID sets for all frequent itemsets, which can become large in dense datasets.

Additional Considerations

- **Memory Usage vs. Speed**:
  - FP-tree and ECLAT generally require more memory than Apriori, but they offer significant speed advantages, especially for larger datasets.
  - The choice between these algorithms should also consider the available computational resources, such as memory and processing power.
- **Simplicity vs. Efficiency**:
  - Apriori is simpler to implement and understand, making it a good choice for educational purposes or scenarios where interpretability and ease of implementation are crucial.
  - FP-tree and ECLAT are more complex but offer better performance for large-scale data mining tasks.

When choosing the appropriate algorithm, it's essential to evaluate the specific characteristics of your dataset (size, density) and the computational resources at your disposal. Additionally, the nature of your mining task (e.g., the importance of speed versus memory usage) can significantly influence the decision.

4.11 Interestingsness Measures

- Association rule algorithms often produce too many rules.
- Many of them are uninteresting or redundant.
- For example: $\{A, B, C\} \rightarrow \{D\}$ and $\{A, B\} \rightarrow \{D\}$ may convey redundant information and also have the same support and confidence.
- "Interestingness Measures" can be used to prune and rank the resulting rules.
- We only used support and confidence measures.

- Other examples measures:

$$\text{Lift} = \frac{c(A \to B)}{s(B)} \tag{9}$$

4.12 Note for algorithms done by hand:

When we compute frequent itemsets for a transaction table, the method is exactly the same for the Apriori method as it is for the ECLAT method.

5 Clustering

5.1 Introduction

Cluster analysis groups data objects based on information found only in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.

5.2 Types of clusters:

5.2.1 Well-separated

A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster.

Figure 19: Well separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster

5.2.2 Prototype-based

A cluster is a set of objects such that an object in a cluster is closer (more similar) to the prototype or "center" of a cluster, than to the center of any other cluster. The center of a cluster is often a centroid, the average of all the points in the cluster, or a medoid, "the most representative point" of a cluster.

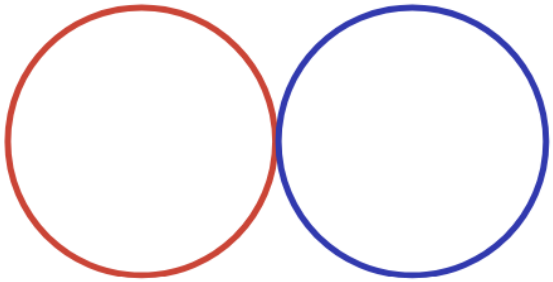


Figure 20: Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster

5.2.3 Contiguity-based

Contiguous Cluster (Nearest neighbor or Transitive): A cluster is a set of points such that a point in a cluster is closer (or more similar) to one or more other points in the cluster than to any point not in the cluster.

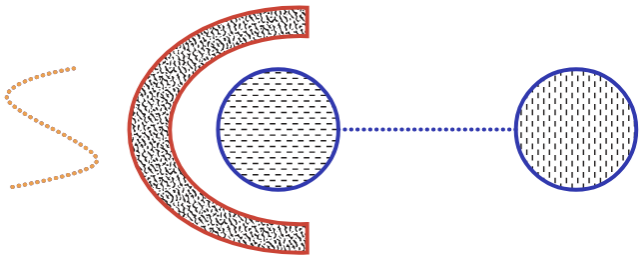


Figure 21: Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster

5.2.4 Density-based

A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density. Used when the clusters are irregular or intertwined, and when noise and outliers are present.

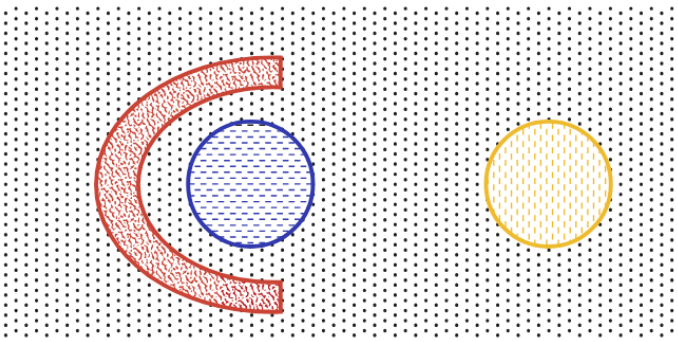


Figure 22: Density-based clusters. Clusters are regions of high density separated by regions of low density

5.2.5 Shared property / Conceptual clusters

Find clusters that share some common property or represent a particular concept.

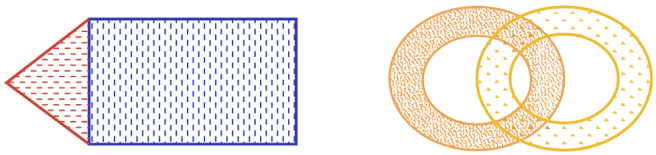


Figure 23: Conceptual clusters. Points in a cluster share some common property that derives from the entire set of points

5.3 Note: Clustering algorithms

This course will look at these clustering algorithms: K-means and its variants, hierarchical clustering and density-based clustering.

5.4 K-means clustering

Is a partitional clustering approach. We must define the number of clusters K. Each cluster is associated with a centroid. Each point is assigned to the cluster with the closest centroid. Very simple algorithm. Algorithm:

Algorithm 7.1 Basic K-means algorithm.

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

5.4.1 Details

- Simple iterative algorithm.
- Initial centroids are often chosen randomly. Therefore, clusters produced can vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster, but other definitions are possible (based on different similarity functions).
- K-means will converge for common proximity measures with appropriately defined centroid (based on different similarity functions).
- Most of the convergence happens in the first few iterations.
  - Often the stopping condition is changed to: until relatively few points change clusters.
- Time complexity is: $O(n \cdot K \cdot l \cdot d)$.
- Space complexity is: $O((n + K) \cdot d)$.
  - n = number of points.
  - K = number of clusters.
  - l = number of iterations.
  - d = number of attributes.

5.4.2 Evaluating K-Means clusters

- Most common measure is Sum of Squared Error (SSE).
  - For each point, the error is the distance to the nearest cluster center.
  - To get SSE, we square these errors and sum them.

$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} (\text{dist}(m_i, x))^2.$$

  - x is a data point in cluster C_i and m_i is the representative point for cluster C_i.

  - We can show that m_i corresponds to the center (mean) of the cluster:

$$c_i = \frac{1}{n_i} \sum_{x \in C_i} x.$$

  - where, n_i is the number of objects in the i^{th} cluster.

- One easy way to reduce SSE is to increase K, the number of clusters.
  - However, a good clustering with smaller K can have a lower SSE than a poor clustering with higher K.

5.4.3 Selecting K

- We seek from error curve: value K where the rate of decline decreases.
- Error curve should look something like an arm in typing position: it slopes down rapidly from shoulder to elbow, and then slower from the elbow to the wrist.
- Want K to be located exactly at the elbow.
- Easier to identify when compared to a similar MSE error plot for random centers, since the relative rate of error reduction for random centers should be analogous to what we see past the elbow. The slow downward drift is telling us the extra clusters are not doing anything special for us.

5.4.4 Solutions to initial centroids problem

1. Multiple runs: helps, but probability is not on your side.
2. Sample and use hierarchical clustering to determine initial centroids.
3. Select more than K initial centroids and then select among these initial centroids. Select most widely separated.
4. K-Means++.
5. Bisecting K-means.
   - Not as susceptible to initialization issues.
6. Post-processing.

5.4.5 K-means++

- This approach can be slower than random initialization, but very consistently produces better results in terms of SSE.
- Algorithm to select initial centroids C:
  1. Select an initial point at random to be the first centroid.
  2. For $k-1$ steps:
     (a) For each of the N points x_i, $1 \leq i \leq N$, find the minimum squared distance to the currently selected centroids, $C_1, \ldots, C_j$, $1 \leq j < k$, i.e., $\min_j (d(C_j, x_i))^2$.
     (b) Randomly select a new centroid by choosing a point with probability proportional to $\frac{\min_j (d(C_j, x_i))^2}{(\sum_i d(C_j, x_i))^2}$.
  3. End For.
- Afterwards, continue K-Means as usual.

5.4.6 Bisecting K-means

- Bisecting K-means algorithm: variant of K-means that can produce a partitional or a hierarchical clustering.
- Straightforward extension of the basic K-means algorithm:
  1. To obtain K clusters, split the set of all points into two clusters.
  2. Select one of these clusters to split, and so on, until K clusters have been produced.

Algorithm 7.3 Bisecting K-means algorithm.

```
1: Initialize the list of clusters to contain the cluster consisting of all points.
2: repeat
3:    Remove a cluster from the list of clusters.
4:    {Perform several "trial" bisections of the chosen cluster.}
5:    for i = 1 to number of trials do
6:       Bisect the selected cluster using basic K-means.
7:    end for
8:    Select the two clusters from the bisection with the lowest total SSE.
9:    Add these two clusters to the list of clusters.
10: until The list of clusters contains K clusters.
```

5.4.7 Pre- and post-processing

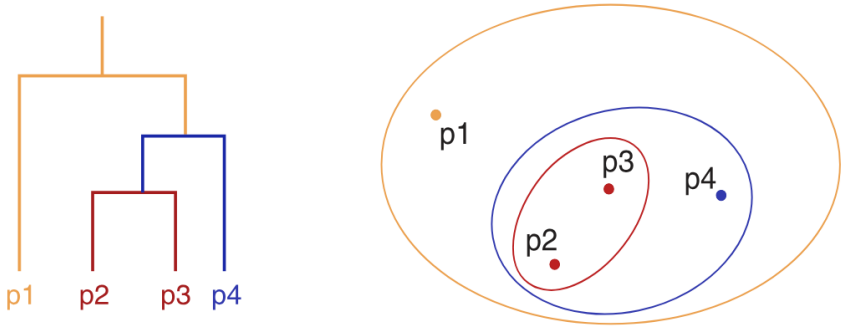
- Pre-processing:
  - Normalize the data.
  - Eliminate outliers.
- Post-processing:
  - Eliminate small clusters that may represent outliers.
  - Split 'loose' clusters, i.e., clusters with relatively high SSE.
  - Merge clusters that are 'close' and that have relatively low SSE.
  - Can use these steps during the clustering process.

5.4.8 Limitations of K-means

- K-means has problems when clusters are of differing:
  - Sizes
  - Densities
  - Non-globular shapes
- K-means has problems when the data contains outliers.

5.5 Hierarchical clustering

Is a clustering technique that produces a set of nested clusters organized as a hierarchical tree. It can additionally be visualized as a dendrogram: a tree like diagram that records the sequences of merges or splits.



(a) Dendrogram. (b) Nested cluster diagram.

Figure 24: A hierarchical clusering of four points visualized as a dendrogram and nested clusters

Strengths:

- Do not have to assume any particular number of clusters.
- Any desired number of clusters can be obtained by 'cutting' the dendogram at the proper level.
- They may correspond to meaningful taxonomies.
- For example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, ...).

5.5.1 Details

- Two main types of hierarchical clustering:
  1. Agglomerative:
     - Start with the points as individual clusters.
     - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left.
  2. Divisive:
     - Start with one, all-inclusive cluster.
     - At each step, split a cluster until each cluster contains a point (or there are k clusters).
- Traditional hierarchical algorithms use a similarity or distance matrix.
  - Merge or split one cluster at a time.

5.5.2 Agglomerative clustering

- Most popular hierarchical clustering technique.
- Basic algorithm is straightforward:
  1. Compute the proximity matrix.
  2. Let each data point be a cluster.
  3. Repeat
     (a) Merge the two closest clusters.
     (b) Update the proximity matrix.
  4. Until only a single cluster remains.
- Key operation is the computation of the proximity of two clusters.
- Different approaches to defining the distance between clusters distinguish the different algorithms.

5.5.3 How to define inter-cluster distance

This is a problem that can take many approaches and helps us with defining new distances in the proximity matrix after merging clusters. In this course we look at the methods; MIN (single-link), MAX(complete-linkage) and group average.

5.5.4 MIN (single-link)

The proximity of two clusters is based on the two closest points in the different clusters. This entails that the distance is determined by one pair of points, i.e. by one link in the proximity graph.
Strength: can handle non-elliptical shapes.
Weakness: sensitive to noise and outliers

5.5.5 MAX (complete linkage)

Proximity of two clusters is based on the two most distant points in the different clusters. This entails that the distance is determined by all pair of points in the two clusters.
Strengts: less susceptible to noise and outliers.
Weakness: Tends to break large clusters, and is biased towards globular clusters.

5.5.6 Group average

Proximity of two clusters is the average of pairwise proximity between points in the two clusters

$$\text{proximity}(\text{cluster}_1, \text{cluster}_2) = \frac{\sum_{p_i \in \text{cluster}_i} \sum_{p_j \in \text{cluster}_j} \text{proximity}(p_i, p_j)}{|\text{cluster}_i| \cdot |\text{cluster}_j|}.$$

We need to use average connectivity for scalability since total proximity patently favours large clusters. We can see group average as a compromise between single and complete link.
Strength: less susceptible to noise and outliers
Weakness: biased towards globular clusters

5.5.7 Time and space complexity

Space:
$O(N^2)$ space since it uses the proximity matrix. (N is the number of points)
Time:
$O(N^3)$ time in many cases. There are N steps and at each step the size, N^2, proximity matrix must be updated and searched. Complexity can be reduced to $O(N^2 log(N))$ time with some cleverness.

5.5.8 Problems and limitations

- Once a decision is made to combine two clusters, it cannot be undone.
- No global objective function is directly minimized.
- Different schemes have problems with one or more of the following:
  - Sensitivity to noise and outliers
  - Difficulty handling clusters of different sizes and non-globular shapes
  - Breaking large clusters

5.6 Density based clustering

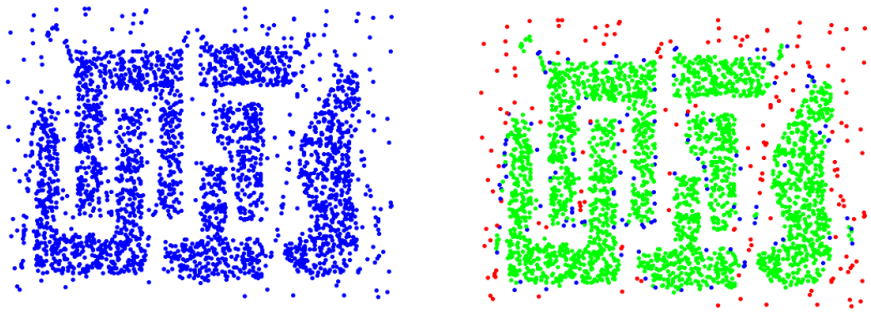
Clusters are regions of high density that are separeted from one another by regions on low density.

5.6.1 DBSCAN Algorithm

DBSCAN is a density-based algorithm.

- Density = number of points within a specified radius (Eps).
  - These are points that are at the interior of a cluster.
  - Counts the point itself
- A boarder point is not a core point, but is in the neighbourhood of a core point. I.e. they are within the MinEps of a core point, but do not themselves satisfy the MinPts condition to be a core point.
- A noise point is any point that is not a core boint or a border point.

Illustration:



Algorithm:

1. Label all points as core, border and noise points
2. Eliminate noise points
3. Put an edge between all core points within a distance Eps of each other
4. Make each group of connected core points into a separate cluster
5. Assign each border point to one of the clusters of its associated core points
6. Thus, we have clusters, made by both core and border points.

Strengts: resistant to noise and can handle clusters of different shapes and sizes.
Weakness: Varying densities and high dimensional data.

5.6.2 Determining parameters

The basic approach is to look at the behavior of the distance from a point to its k-th nearest neighbor, which we will call the k-dist. For points that belong to some cluster, the value of k-dist will be small if k is not larger than the cluster size. Note that there will be some variation, depending on the density of the cluster and the random distribution of points, but on average, the range of variation will not be huge if the cluster densities are not radically different. However, for points that are not in a cluster, such as noise points, the k-dist will be relatively large. Therefore, if we compute the k-dist for all the data points for some k, sort them in increasing order, and then plot the sorted values, we expect to see a sharp change at the value of k-dist that corresponds to a suitable

value of Eps. If we select this distance as the Eps parameter and take the value of k as the MinPts parameter, then points for which k-dist is less than Eps will be labeled as core points, while other points will be labeled as noise or border points. As a rule of thumb we choose the parameter k as $k >= D + 1$, where D is the dimension of the dataset.

5.6.3 Time and space complexity

Time:

O(m · time to find points in the Eps-neighbourhood), where m is the number of points. Worst case, this complexity is $O(m^2)$. Using indexes time complexity can be as low as $O(m \cdot log(m))$.

Space:

Space requirement is $O(m)$. Only a small amount of data for each point is needed (i.e., the cluster label and the identification of each point as a core, border, or noise point).

5.7 Evaluating clustering and clusters

We want to evaluate clusters for several reasons:

- To avoid finding patterns in noise
- To compare clustering algorithms
- To compare two sets of clusters
- to compare two clusters

In this course we will look at unsupervised cluster evaluation.

5.7.1 Different aspects of cluster validation

1. Determining the clustering tendency of a set of data, i.e. distinguishing wheter non-random structure actually exists in the data.
2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.
3. Evaluating how well the results of a cluster analysis fit the data without reference to external information. (use only data).
4. Comparing the results of two different sets of cluster analyses to determine which is better
5. Determine the "correct" number of clusters

For 2,3 and 4, we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

5.7.2 Cohesion and separation

Cluster cohesion: Measures how closely related objects are in a cluster. (SSE for instance).
Cluster separation: Measures how distinct or well-separated a cluster is from other clusters.
Example SSE:

- Cohesion is measured by the within cluster sum of squares (SSE):

$$\text{SSE} = \sum_i \sum_{x \in C_i} (x - m_i)^2.$$

- Separation is measured by the between cluster sum of squares (SSB):

$$\text{SSB} = \sum_i |C_i|(m - m_i)^2,$$

- where, $|C_i|$ is the size of cluster i.

5.7.3 Silhouette coefficient

- Silhouette Coefficient: combines ideas of both cohesion and separation, but for individual points, as well as clusters and clusterings. It is used to evaluate the goodness of fit for a point in a cluster, and is a metric for both how similar a point is to others in a cluster and how dissimilar it is to points in other clusters.
- For an individual point, i
  - Calculate a = average distance of i to the points in its cluster.
  - Calculate b = min(average distance of i to points in another cluster).
  - The silhouette coefficient for a point is then given by

$$s = \frac{(b - a)}{\max(a, b)}$$

- Value can vary between -1 and 1
- Typically ranges between 0 and 1.
- The closer to 1 the better.
- Can calculate the average silhouette coefficient for a cluster or a clustering.

5.7.4 Correlation

- Two matrices:
  1. Proximity Matrix
     - A similarity matrix can also be obtained by transforming / normalizing the distances using the formula:

$$s = 1 - \frac{d - d_{\min}}{d_{\max} - d_{\min}}$$

  2. Ideal Similarity Matrix
     - One row and one column for each data point.
     - An entry is 1 if the associated pair of points belong to the same cluster.
     - An entry is 0 if the associated pair of points belong to different clusters.

- Compute the correlation between the two matrices.
  - Since the matrices are symmetric, only the correlation between $\frac{n(n-1)}{2}$ entries needs to be calculated.

- High magnitude of correlation indicates that points that belong to the same cluster are close to each other.
  - Correlation may be positive or negative depending on whether the similarity matrix is a similarity or dissimilarity matrix.

- Not a good measure for some density or contiguity based clusters.

5.7.5 Cophenetic correlation

- Cophenetic Distance between two objects is the proximity at which an agglomerative hierarchical clustering technique puts the objects in the same cluster for the first time.

- Cophenetic distance matrix, the entries are the cophenetic distances between each pair of objects.

- Cophenetic Correlation Coefficient (CPCC) is the correlation between the entries of this matrix and the original dissimilarity matrix.

- It is a standard measure of how well a hierarchical clustering fits the data.

- Common use is to evaluate which type of hierarchical clustering is best for a particular type of data.

5.7.6 Clustering tendency - Hopkins statistic

Hopkins Statistic measures the clustering tendency. Clustering tendency is a measure of the existence of actual clusters / patterns in the dataset as opposed to random clusters / patterns. The insight behind Hopkins statistic is that it measures the ratio of the distances between points from a sample of points drawn from the dataset and distances between points in a sample of randomly generated points. Based on this ratio it can be determined if there are actual patterns / clusters in the dataset.

1. Generate p points that are randomly distributed across the data space.
2. Sample p actual data points.
3. For both sets of points, we find the distance to the nearest neighbor in the original data set.
4. Let the u_i be the nearest neighbor distances of the artificially generated points, while the w_i are the nearest neighbor distances of the sample of points from the original data set.
5. The Hopkins statistic H is then defined by:

$$H = \frac{\sum_{i=1}^{p} w_i}{\sum_{i=1}^{p} u_i + \sum_{i=1}^{p} w_i}$$

Meaning of the statistic:

- $H \approx 0.5$: randomly generated points and the sample of data points have roughly the same nearest neighbor distances.
- $H \approx 1.0$: data that is regularly distributed in the data space.
- $H \approx 0.0$: data that is highly clustered.

5.7.7 Framework for cluster validity

- Need a framework to interpret any measure.
  - For example, if our measure of evaluation has the value, 10, is that good, fair, or poor?
- Statistics provide a framework for cluster validity.
  - The more "atypical" a clustering result is, the more likely it represents valid structure in the data.
  - Can compare the values of an index that result from random data or clusterings to those of a clustering result.
  - If the value of the index is unlikely, then the cluster results are valid.

- For comparing the results of two different sets of cluster analyses, a framework is less necessary.
  - However, there is the question of whether the difference between two index values is significant.

6 Classification

- Given a collection of records (*training set*):
  - Each record is characterized by a tuple (x, y), where x is the attribute set and y is the class label:
    - x: attribute, predictor, independent variable, or input.
    - y: class, response, dependent variable, or output.
- Task:
  - Learn a model that maps each attribute set x into one of the predefined class labels y.

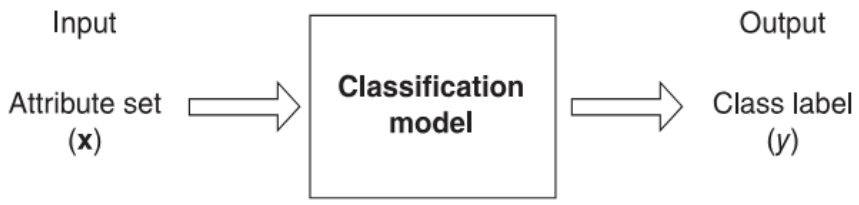


Figure 25: A schematic illustration of a classification task

In our curriculum we will learn about decision tree based methods and nearest-neighbour.

6.1 Decision trees

6.1.1 Basic setup

A decision tree has 3 types of nodes:

- A root node, with no incoming links and zero or more outgoing links
- Internal nodes, each of which has exactly one incoming link and two or more outgoing links
- Leaf or terminal nodes, each of which has exactly one incoming link and no outgoing links

Every leaf node in a decision tree is associated with a class label. The non-terminal nodes, which include the root and internal nodes, contain attribute test conditions that are typically defined using a single attribute. Each possible outcome of the attribute test condition is associated with exactly one child of this node
Given a decision tree, classifying a test instance is straightforward. Starting from the root node, we apply its attribute test condition and follow the appropriate branch based on the outcome of the test. This will lead us either to another internal node, for which a new attribute test condition is applied, or to a leaf node.

6.1.2 Hunt's algorithm

1. Let D_t be the set of training records that reach a node t.
2. General Procedure:
    - If D_t contains records that belong the same class y_t, then t is a leaf node labeled as y_t.
    - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

6.1.3 Design issues of decision tree induction

- How should training records be split?
    - Method for expressing test condition.
        - Depending on attribute types.
    - Measure for evaluating the goodness of a test condition.
- How should the splitting procedure stop?
    - Stop splitting if all the records belong to the same class or have identical attribute values.
    - Early termination.

6.1.4 Methods for expressing test conditions

These methods depend on the attribute types:

- Binary
- Nominal
- Ordinal
- Continuous

Nominal attributes:
There are 2 common ways to implement splits for these attributes:

1. Multi-way Split:
    - Use as many partitions as distinct values.
2. Binary Split:
    - Divides values into two subsets.

For ordinal attributes we must also preserve order property among the attribute values.

Continuous attributes:

- Different ways of handling.
    - Discretization to form an ordinal categorical attribute.
    - Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
        * Static – discretize once at the beginning.
        * Dynamic – repeat at each node.
- Binary Decision: $(A < v)$ or $(A \neq v)$.
    - Consider all possible splits and find the best cut.
    - Can be more compute intensive.

6.1.5 Split method for different attributes:

Binary attributes:
Splits into two partitions (child nodes)

Categorical attributes:

- For each distinct value, gather counts for each class in the dataset.
- Use the count matrix to make decisions.

Continuous attributes:

- Use Binary Decisions based on one value.
- Several Choices for the splitting value:
    - Number of possible splitting values = Number of distinct values.

- Each splitting value has a count matrix associated with it.
  - Class counts in each of the partitions, $A \leq v$ and $A > v$.
- for efficient computation, for each attribute:
  1. Sort the attribute on values.
  2. Linearly scan these values, each time updating the count matrix and computing gini index.
  3. Choose the split position that has the least gini index.

6.1.6 How to determine best split

We have a greedy approach: we choose to split on the attribute that gives the purest class distribution.

6.1.7 Measure of node impurity

- Gini Index:

$$\text{Gini Index} = 1 - \sum_{i=0}^{c-1} \left[p_i(t)\right]^2 . \tag{10}$$

- Entropy:

$$\text{Entropy} = -\sum_{i=0}^{c-1} p_i(t) \cdot \log_2 \left[p_i(t)\right] . \tag{11}$$

- Misclassification Error:

$$\text{Classification Error} = 1 - \max \left[p_i(t)\right] . \tag{12}$$

- Where $p_i(t)$ is the frequency of class i at node t, and c is the total number of classes.

6.1.8 Finding the best split

1. Compute impurity measure (P) before splitting.
2. Compute impurity measure (M) after splitting.
   - Compute impurity measure of each child node.
   - M is the weighted impurity of child nodes.
3. Choose the attribute test condition that produces the highest gain.

$$\text{Gain} = P - M.$$

4. Or, equivalently, lowest impurity measure after splitting (M).

6.1.9 GINI

- Gini Index for a given node t

$$\text{Gini Index} = 1 - \sum_{i=0}^{c-1} \left[p_i(t)\right]^2 \tag{5}$$

- Where $p_i(t)$ is the frequency of class i at node t, and c is the total number of classes.
- Maximum of $1 - \frac{1}{c}$ when records are equally distributed among all classes, implying the least beneficial situation for classification.
- Minimum of 0 when all records belong to one class, implying the most beneficial situation for classification.
- When a node p is split into k partitions (children):

$$\text{GINI}_{\text{split}} = \sum_{i=1}^{k} \frac{n_i}{n} \text{GINI}(i). \tag{9}$$

- Where n_i is the number of records at child i and n is the number of records at parent node p.

6.1.10 Entropy

- Entropy:

$$\text{Entropy} = -\sum_{i=0}^{c-1} p_i(t) \cdot \log_2 \left[p_i(t)\right]. \tag{10}$$

- Where $p_i(t)$ is the frequency of class i at node t, and c is the total number of classes.
- Maximum of $\log_2(c)$ when records are equally distributed among all classes, implying the least beneficial situation for classification.
- Minimum of 0 when all records belong to one class, implying most beneficial situation for classification.
- Entropy based computations are quite similar to the GINI index computations.
-

$$\text{Entropy}_{\text{split}} = \sum_{i=1}^{k} \frac{n_i}{n} \text{Entropy}(i)$$

- Parent Node, p is split into k partitions (children) n is number of records in child node i.

- Information Gain:

$$\text{Gain}_{\text{split}} = \text{Entropy}(p) - \sum_{i=1}^{k} \frac{n_i}{n} \text{Entropy}(i).$$

- Choose the split that achieves most reduction (maximizes GAIN).
- Information gain is the mutual information between the class variable and the splitting variable.

Gain ratio:

- Gain Ratio:

$$\text{Gain Ratio} = \frac{\text{Gain}_{\text{split}}}{\text{Split Info}} \tag{13}$$

$$\text{Split Info} = -\sum_{i=1}^{k} \frac{n_i}{n} \log_2 \left[\frac{n_i}{n} \right]. \tag{14}$$

- Parent Node, p is split into k partitions (children), n_i is the number of records in child node i.
- Adjusts Information Gain by the entropy of the partitioning (Split Info).
  - Higher entropy partitioning (large number of small partitions) is penalized!
- Designed to overcome the disadvantage of Information Gain.

6.1.11 Classification error

- Classification error at a node t:

$$\text{Error}(t) = 1 - \max_i [p_i(t)].$$

- Maximum of $1 - \frac{1}{c}$ when records are equally distributed among all classes, implying the least interesting situation.
- Minimum of 0 when all records belong to one class, implying the most interesting situation.

6.1.12 Algorithm for decision tree induction

Pseudocode:

Algorithm 3.1 A skeleton decision tree induction algorithm.

```
TreeGrowth (E, F)
 1: if stopping_cond(E,F) = true then
 2:    leaf = createNode().
 3:    leaf.label = Classify(E).
 4:    return leaf.
 5: else
 6:    root = createNode().
 7:    root.test_cond = find_best_split(E, F).
 8:    let V = {v|v is a possible outcome of root.test_cond }.
 9:    for each v ∈ V do
10:       E_v = {e | root.test_cond(e) = v and e ∈ E}.
11:       child = TreeGrowth(E_v, F).
12:       add child as descendent of root and label the edge (root → child) as v.
13:    end for
14: end if
15: return root.
```

The input to this algorithm is a set of training instances E along with the attribute set F. The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the nodes of the tree (Steps 11 and 12) until the stopping criterion is met (Step 1). The details of this algorithm are explained below.

1. The `createNode()` function extends the decision tree by creating a new node. A node in the decision tree either has a test condition, denoted as `node.test_cond`, or a class label, denoted as `node.label`.

2. The `find_best_split()` function determines the attribute test condition for partitioning the training instances associated with a node. The splitting attribute chosen depends on the impurity measure used. The popular measures include entropy and the Gini index.

3. The `Classify()` function determines the class label to be assigned to a leaf node. For each leaf node t, let $p(i|t)$ denote the fraction of training instances from class i associated with the node t. The label assigned is the class with the highest $p(i|t)$. to the leaf node is typically the one that occurs most frequently in the training instances that are associated with this node.

$$\text{leaf.label} = \arg\max_i \; p(i|t), \tag{3.10}$$

   where the arg max operator returns the class i that maximizes $p(i|t)$. Besides providing the information needed to determine the class label of a leaf node, $p(i|t)$ can also be used as a rough estimate of the probability that an instance assigned to the leaf node t belongs to class i.

4. The `stopping_cond()` function is used to terminate the tree-growing process by checking whether all the instances have identical class label or attribute values. Since decision tree classifiers employ a top-down, recursive partitioning approach for building a model, the number of training instances associated with a node decreases as the depth of the tree increases. As a result, a leaf node may contain too few training instances to make a statistically significant decision about its class label. This is known as the *data fragmentation* problem. One way to avoid this problem is to disallow splitting of a node when the number of instances associated with the node fall below a certain threshold.

6.1.13 Advantages of decision trees

- Inexpensive to construct.
- Extremely fast at classifying unknown records.
- Easy to interpret for small-sized trees.
- Accuracy is comparable to other classification techniques for many simple data sets.

6.2 Underfitting and overfitting

6.2.1 Underfitting

When a model is too simple, both training error and test errors are large.

6.2.2 Overfitting

When a model is too complex, training error is small but test error is large.
Decision trees: Overfitting results in complex and large decision trees. Hence, they are difficult to comprehend and result in bad generalization, resulting in poor overall performance when employed on unseen instances.

6.2.3 How to address overfitting

- Increasing the size of training data reduces the difference between training and testing errors at a given size of model.
- Pre-Pruning (Early Stopping Rule):
  - Stop the algorithm before it becomes a fully-grown tree.
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class.
    - Stop if all the attribute values are the same.
  - More restrictive conditions:

    - Stop if number of instances is less than some user-specified threshold.
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test).
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

- Post-pruning:
  - Grow decision tree to its entirety.
  - Trim the nodes of the decision tree in a bottom-up fashion.
  - If generalization error improves after trimming, replace sub-tree by a leaf node.
  - Class label of leaf node is determined from majority class of instances in the sub-tree.

6.3 Evaluation

6.3.1 Evaluation metrics

We have numerous evaluation metrics. Here are some of the most important based, where the formulas are based on this table: (Also known as a confusion matrix)

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	a (True Positive)	b (False Negative)
	Class = No	c (False Positive)	d (True Negative)

Here are our most used metrics in this course:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{a + d}{a + b + c + d}.$$

$$\text{Error Rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{FN + FP}{TP + TN + FP + FN} = \frac{b + c}{a + b + c + d}.$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{a}{a + c}.$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{a}{a + b}.$$

$$\text{F-Measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot a}{2 \cdot a + b + c}.$$

6.3.2 Methods of estimation

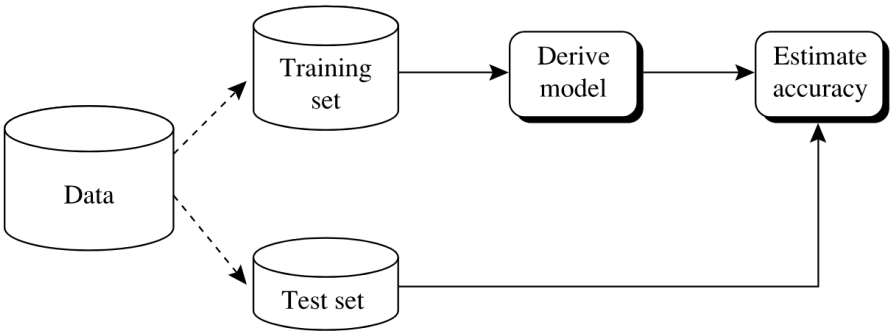


Figure 26: Estimating accuracy with the holdout method

- Holdout:
  - Reserve 2/3 for training and 1/3 for testing.
- Random subsampling:
  - Repeated holdout.
- Cross validation:
  - Partition data into k disjoint subsets.
  - k-fold: train on $k-1$ partitions, test on the remaining one.
  - This is process is repeated k-times, each time using a different fold as the validation set
  - Then we average the results to see how the classification model performed
- Bootstrap:
  - Sampling with replacement.

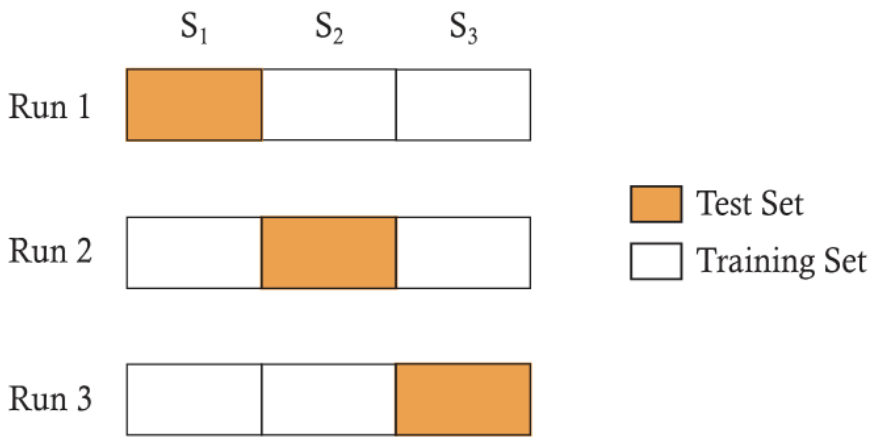


Figure 27: Example demonstrating the technique of 3-fold cross-validation

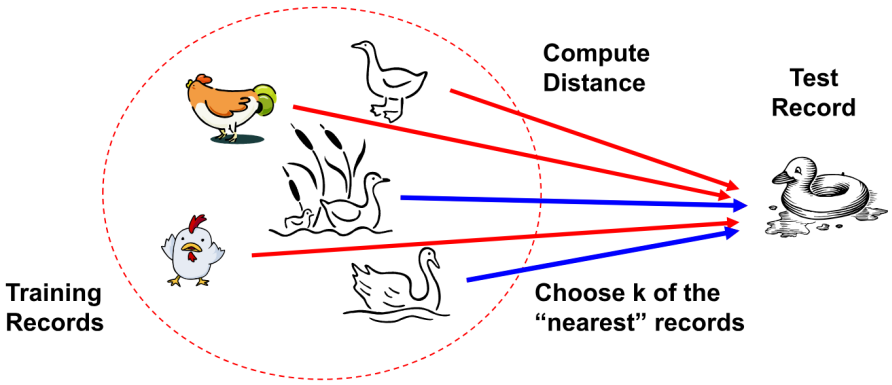
6.4 Nearest neighbour classifiers

6.4.1 Eager and lazy learners

- **Eager Learners**: Learn a model that maps the input attributes to the class label from the training data.
  - Decision Trees.
- **Lazy Learners**: Delay modeling the training data until classification of test examples is needed.
  - **Rote Classifier**: Memorize entire training data and perform classification only if the attributes of a test instance match one of the training examples exactly.

6.4.2 Nearest neighbour classifiers

The basic idea is that "If it walks like a duck, quacks like a duck, then it's probably a duck."



For us to be able to use this method we require:

- A set of labeled records.
- Proximity metric to compute distance/similarity between a pair of records (e.g., Euclidean distance).
- The value of k, the number of nearest neighbors to retrieve.
- A method for using class labels of k nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote).

6.4.3 How to Determine the Class Label of a Test Sample?

- Take the majority vote of class labels among the k-nearest neighbors.
- Weight the vote according to distance (e.g., with a weight factor, $w = \frac{1}{d^2}$ or something different depending on the problem.)

6.4.4 Key bottleneck of K-NN

The key bottleneck in k-nearest neighbor classifiers is the repeated and inefficient distance computation during the classification. To alleviate this kd-trees or R-trees can be utilize to speed up the retrieval of matching instance within the proximity of the test instance

6.4.5 Choice of proximity measure matters

The choice of proximity measure significantly impacts the performance of machine learning algorithms. Different types of data and applications benefit from different proximity measures. Here are some examples:

Documents
For documents, cosine similarity is often better than correlation or Euclidean

distance. This is because cosine similarity measures the orientation of the document vectors rather than their magnitude, making it more suitable for text data where the length of the document might not be as important as the content itself.

- **Cosine Similarity**: Measures the cosine of the angle between two non-zero vectors.
- **Correlation**: Measures the linear relationship between two variables.
- **Euclidean Distance**: Measures the straight-line distance between two points in Euclidean space.

Image Data

For image data, mean squared error (MSE) might be more appropriate.

- **Mean Squared Error (MSE)**: Measures the average of the squares of the errors between the pixels of the two images.

Categorical Data

For categorical data, Jaccard index might be more suitable.

- **Jaccard Index**: Measures the similarity between finite sample sets, defined as the size of the intersection divided by the size of the union of the sample sets.

Time-Series Data

For time-series data, Pearson correlation might be more effective.

- **Pearson Correlation**: Measures the linear correlation between two sequences.

Genomic Data

For genomic data, measures like Jaccard index are often used.

- **Jaccard Index**: Similar to its use in categorical data, it measures the similarity between sets of genomic data.

The choice of proximity measure should be based on the nature of the data and the specific requirements of the application to ensure optimal performance of the machine learning models.

6.4.6 Nearest neighbour classification final points

- Data preprocessing is often required:
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes.
  - Example:
    - Height of a person may vary from 1.5 m to 1.8 m.
    - Weight of a person may vary from 90 lb to 300 lb.
    - Income of a person may vary from 10 K to 1 M.
  - Time series are often standardized to have 0 mean and a standard deviation of 1.
- Choosing the value of k:
  - If k is too small, sensitive to noise points.
  - If k is too large, neighborhood may include points from other classes.
- Nearest neighbour classifiers are local classifiers
- They can produce decision boundaries of arbitrary shapes

6.5 Comparing decision boundaries:

Decision trees perform classification via recursive partition of the data set that results in axis-parallel hyperplanes as decision boundaries. While for nearest-neighbours the decision boundaries can have arbitrary shapes.

6.5.1 Illustrations:

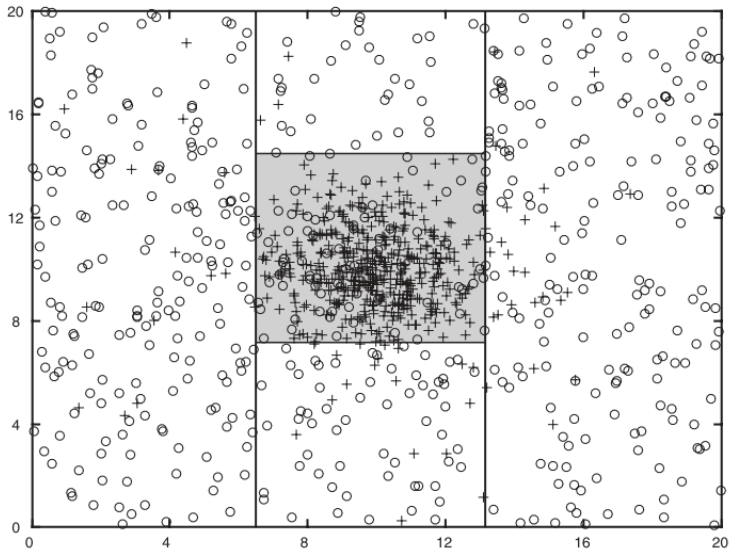


Figure 28: Decision tree boundary with 5 leaf nodes, representing the 4 axis-parallel hyperplanes

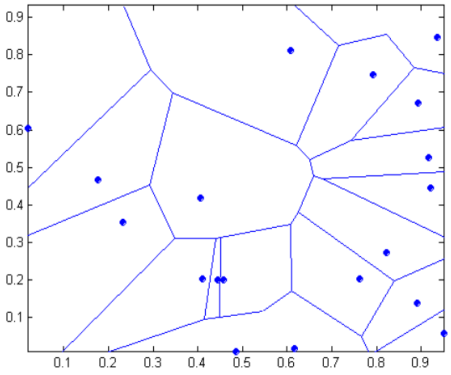


Figure 29: 1-NN decision boundary is a Vornoi diagram

7 Web Usage Mining

7.1 Web usage data

7.1.1 Data:

- Data from e-commerce, Web services, and Web-based information systems:
  - Volumes of Clickstream
  - Transaction Data
  - User Profile Data
- Analyzing such data can help organizations determine:
  - Value of clients.
  - Marketing strategies across products.
  - Effectiveness of promotional campaigns.
  - Optimize Web-based applications.
  - Personalized content.

7.1.2 Data mining process:

Web usage mining processes have 3 phases:

1. Data collection and pre-processing
2. Pattern discovery
3. Pattern analysis

7.1.3 Data collection:

- Web Usage Data:
  - Server Log Files (e.g., Web Server Access Logs and Application Server Logs)
  - Site Files
  - Metadata
  - Operational Databases
  - Application Templates
  - Domain Knowledge
  - Demographics Data
- Four primary groups of data sources:
  1. Usage Data (Server Access Logs)

2. Content Data (HTML/XML pages)
3. Structure Data (Hyperlink Structure as Sitemaps)
4. User Data (User Profiles)

Usage data

1	2006-02-01 00:08:43 1.2.3.4 - GET /classes/cs589/papers.html - 200 9221 HTTP/1.1 maya.cs.depaul.edu RESOURCE Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1;+.NET+CLR+2.0.50727) http://dataminingresources.blogspot.com/
2	2006-02-01 00:08:46 1.2.3.4 - GET /classes/cs589/papers/cms-tai.pdf - 200 4096 HTTP/1.1 maya.cs.depaul.edu IP ADDRESS Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1;+.NET+CLR+2.0.50727) http://maya.cs.depaul.edu/~classes/cs589/papers.html
3	2006-02-01 08:01:28 2.3.4.5 - GET /classes/ds575/papers/hyperlink.pdf - 200 318814 HTTP/1.1 maya.cs.depaul.edu Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1) REFERRER FIELD http://www.google.com/search?hl=en&lr=&q=hyperlink+analysis+for+the+web+survey
4	2006-02-02 19:34:45 3.4.5.6 - GET /classes/cs480/announce.html - 200 3794 HTTP/1.1 maya.cs.depaul.edu Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1) AGENT FIELD http://maya.cs.depaul.edu/~classes/cs480/
5	2006-02-02 19:34:45 3.4.5.6 - GET /classes/cs480/styles2.css - 200 1636 HTTP/1.1 maya.cs.depaul.edu Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1) http://maya.cs.depaul.edu/~classes/cs480/announce.html
6	2006-02-02 19:34:45 3.4.5.6 - GET /classes/cs480/header.gif - 200 6027 HTTP/1.1 maya.cs.depaul.edu Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1) http://maya.cs.depaul.edu/~classes/cs480/announce.html

- Usage data needs to be transformed and aggregated at different levels of abstraction for different analysis:
  1. Pageview — aggregation of resources representing user events (e.g., viewing a page).
  2. Session — sequence of pageviews by a single user during a single visit.
- Key steps:
  - Data Fusion and Cleaning
  - Pageview Identification
  - User Identification
  - Sessionization

7.1.4 Data Pre-processing:

Data fusion and cleaning:

- Data cleaning is site-specific, involving tasks such as:
  - Removing extraneous references (e.g., style, graphics, and sound files).
  - Removing useless data fields (e.g., HTTP protocol).
  - Removing references due to crawler navigations.

Pageview identification:

- Each pageview is a collection of resources representing a specific "user event," e.g.,
  - Clicking on a link.
  - Viewing a product page.
  - Adding a product to the shopping cart.
- Static single frame site: each HTML file has a one-to-one correspondence with a pageview.
- Multi-framed sites: several files make up a given pageview.
- Dynamic sites: combination of static templates and content generated by application servers.

User identification:

- Authentication mechanisms.
- Client-side cookies.
- Using a combination of IP addresses and other information such as user agents and referrers.

Method	Description	Privacy Concerns	Advantages	Disadvantages
IP Address + Agent	Assume each unique IP address / Agent pair is a unique user	Low	Always available. No additional technology required.	Not guaranteed to be unique. Defeated by rotating IPs.
Embedded Session Ids	Use dynamically generated pages to associate ID with every hyperlink	Low to medium	Always available. Independent of IP addresses.	Cannot capture repeat visitors. Additional overhead for dynamic pages.
Registration	User explicitly logs in to the site.	Medium	Can track individuals not just browsers	Many users won't register. Not available before registration.
Cookie	Save ID on the client machine.	Medium to high	Can track repeat visits from same browser.	Can be turned off by users.
Software Agents	Program loaded into browser and sends back usage data.	High	Accurate usage data for a single site.	Likely to be rejected by users.

Time	IP	URL	Ref	Agent
0:01	1.2.3.4	A	-	IE5;Win2k
0:09	1.2.3.4	B	A	IE5;Win2k
0:10	2.3.4.5	C	-	IE6;WinXP;SP1
0:12	2.3.4.5	B	C	IE6;WinXP;SP1
0:15	2.3.4.5	E	C	IE6;WinXP;SP1
0:19	1.2.3.4	C	A	IE5;Win2k
0:22	2.3.4.5	D	B	IE6;WinXP;SP1
0:22	1.2.3.4	A	-	IE6;WinXP;SP2
0:25	1.2.3.4	E	C	IE5;Win2k
0:25	1.2.3.4	C	A	IE6;WinXP;SP2
0:33	1.2.3.4	B	C	IE6;WinXP;SP2
0:58	1.2.3.4	D	B	IE6;WinXP;SP2
1:10	1.2.3.4	E	D	IE6;WinXP;SP2
1:15	1.2.3.4	A	-	IE5;Win2k
1:16	1.2.3.4	C	A	IE5;Win2k
1:17	1.2.3.4	F	C	IE6;WinXP;SP2
1:26	1.2.3.4	F	C	IE5;Win2k
1:30	1.2.3.4	B	A	IE5;Win2k
1:36	1.2.3.4	D	B	IE5;Win2k

User 1

0:01	1.2.3.4	A	-
0:09	1.2.3.4	B	A
0:19	1.2.3.4	C	A
0:25	1.2.3.4	E	C
1:15	1.2.3.4	A	-
1:26	1.2.3.4	F	C
1:30	1.2.3.4	B	A
1:36	1.2.3.4	D	B

User 2

0:10	2.3.4.5	C	-
0:12	2.3.4.5	B	C
0:15	2.3.4.5	E	C
0:22	2.3.4.5	D	B

User 3

0:22	1.2.3.4	A	-
0:25	1.2.3.4	C	A
0:33	1.2.3.4	B	C
0:58	1.2.3.4	D	B
1:10	1.2.3.4	E	D
1:17	1.2.3.4	F	C

Figure 30: Example of user identification using IP + Agent

Sessionization:

- **Sessionization**: segmenting user activity record of each user into sessions, each representing a single visit to the site.
- Without mechanisms such as embedded session ids must rely on heuristics methods for sessionization.
- Heuristic categories:
  1. **Time-Oriented**: global or local time-out estimate to distinguish between consecutive sessions.
  2. **Structure-Oriented**: static site structure or implicit linkage structure captured in the referrer fields of the server logs.

Time	IP	URL	Ref	Agent
0:01	1.2.3.4	A	-	IE5;Win2k
0:09	1.2.3.4	B	A	IE5;Win2k
0:10	2.3.4.5	C	-	IE6;WinXP;SP1
0:12	2.3.4.5	B	C	IE6;WinXP;SP1
0:15	2.3.4.5	E	C	IE6;WinXP;SP1
0:19	1.2.3.4	C	A	IE5;Win2k
0:22	2.3.4.5	D	B	IE6;WinXP;SP1
0:22	1.2.3.4	A	-	IE6;WinXP;SP2
0:25	1.2.3.4	E	C	IE5;Win2k
0:25	1.2.3.4	C	A	IE6;WinXP;SP2
0:33	1.2.3.4	B	C	IE6;WinXP;SP2
0:58	1.2.3.4	D	B	IE6;WinXP;SP2
1:10	1.2.3.4	E	D	IE6;WinXP;SP2
1:15	1.2.3.4	A	-	IE5;Win2k
1:16	1.2.3.4	C	A	IE5;Win2k
1:17	1.2.3.4	F	C	IE6;WinXP;SP2
1:26	1.2.3.4	F	C	IE5;Win2k
1:30	1.2.3.4	B	A	IE5;Win2k
1:36	1.2.3.4	D	B	IE5;Win2k

User 1

0:01	1.2.3.4	A	-
0:09	1.2.3.4	B	A
0:19	1.2.3.4	C	A
0:25	1.2.3.4	E	C
1:15	1.2.3.4	A	-
1:26	1.2.3.4	F	C
1:30	1.2.3.4	B	A
1:36	1.2.3.4	D	B

User 2

0:10	2.3.4.5	C	-
0:12	2.3.4.5	B	C
0:15	2.3.4.5	E	C
0:22	2.3.4.5	D	B

User 3

0:22	1.2.3.4	A	-
0:25	1.2.3.4	C	A
0:33	1.2.3.4	B	C
0:58	1.2.3.4	D	B
1:10	1.2.3.4	E	D
1:17	1.2.3.4	F	C

Figure 31: Example of sessionization with a time-oriented heuristic

Time	IP	URL	Ref	Agent
0:01	1.2.3.4	A	-	IE5;Win2k
0:09	1.2.3.4	B	A	IE5;Win2k
0:10	2.3.4.5	C	-	IE6;WinXP;SP1
0:12	2.3.4.5	B	C	IE6;WinXP;SP1
0:15	2.3.4.5	E	C	IE6;WinXP;SP1
0:19	1.2.3.4	C	A	IE5;Win2k
0:22	2.3.4.5	D	B	IE6;WinXP;SP1
0:22	1.2.3.4	A	-	IE6;WinXP;SP2
0:25	1.2.3.4	E	C	IE5;Win2k
0:25	1.2.3.4	C	A	IE6;WinXP;SP2
0:33	1.2.3.4	B	C	IE6;WinXP;SP2
0:58	1.2.3.4	D	B	IE6;WinXP;SP2
1:10	1.2.3.4	E	D	IE6;WinXP;SP2
1:15	1.2.3.4	A	-	IE5;Win2k
1:16	1.2.3.4	C	A	IE5;Win2k
1:17	1.2.3.4	F	C	IE6;WinXP;SP2
1:26	1.2.3.4	F	C	IE5;Win2k
1:30	1.2.3.4	B	A	IE5;Win2k
1:36	1.2.3.4	D	B	IE5;Win2k

User 1

0:01	1.2.3.4	A	-
0:09	1.2.3.4	B	A
0:19	1.2.3.4	C	A
0:25	1.2.3.4	E	C
1:15	1.2.3.4	A	-
1:26	1.2.3.4	F	C
1:30	1.2.3.4	B	A
1:36	1.2.3.4	D	B

User 2

0:10	2.3.4.5	C	-
0:12	2.3.4.5	B	C
0:15	2.3.4.5	E	C
0:22	2.3.4.5	D	B

User 3

0:22	1.2.3.4	A	-
0:25	1.2.3.4	C	A
0:33	1.2.3.4	B	C
0:58	1.2.3.4	D	B
1:10	1.2.3.4	E	D
1:17	1.2.3.4	F	C

Figure 32: Example of sessionization with a h-ref heuristic

Path completion:

- Client- or proxy-side caching can often result in missing access references to those pages or objects that have been cached.
- Effective path completion requires extensive knowledge of the link structure within the site.
- Referrer information in server logs can also be used in disambiguating the inferred paths.

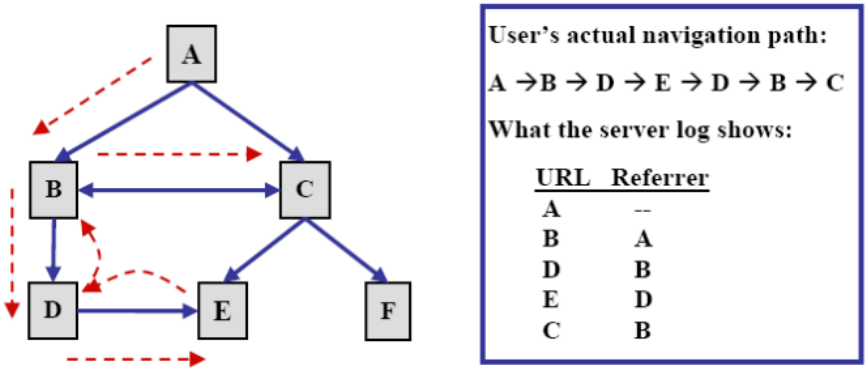
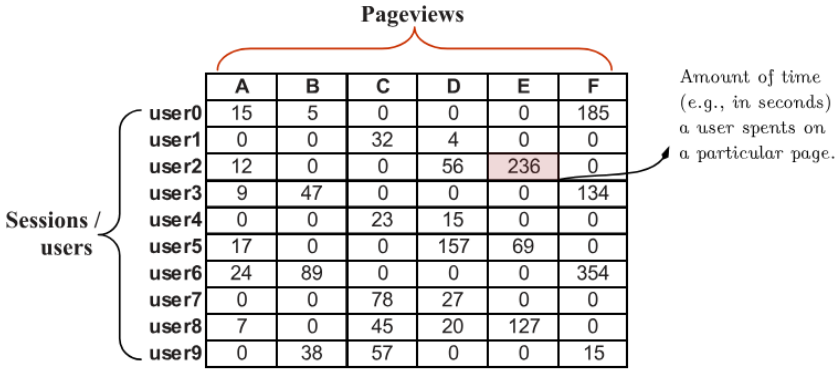


Figure 33: Missing references du to caching

7.2 Data modelling (See example problem in exam 2024:2)

- Data pre-processing results in a set of n pageviews: $P = \{p_1, p_2, ..., p_n\}$.
- Also, a set of m user transactions: $T = \{t_1, t_2, ..., t_m\}$, where $t_i \in T$ is a subset of P.
- Pageviews are semantically meaningful entities to which mining tasks are applied (such as pages or products).
- Set of all user transactions can be viewed as an $m \times n$ user-pageview matrix (also called the transaction matrix), denoted by UPM.



	A	B	C	D	E	F
user0	15	5	0	0	0	185
user1	0	0	32	4	0	0
user2	12	0	0	56	236	0
user3	9	47	0	0	0	134
user4	0	0	23	15	0	0
user5	17	0	0	157	69	0
user6	24	89	0	0	0	354
user7	0	0	78	27	0	0
user8	7	0	45	20	127	0
user9	0	38	57	0	0	15

Figure 34: An example of a user-x-pageview matrix

- We can also integrate other knowledge sources, such as semantic information from the Web page contents for mining.
- Generally, the textual features from the content of Web pages represent the underlying semantics of the site.
- Each pageview p can be represented as a r-dimensional feature vector, where r is the total number of extracted features (words or concepts) from the site in a global dictionary.
- This vector, denoted by p, can be given by:

$$p = (fw^p(f_1), fw^p(f_2), ..., fw^p(f_r)),$$

  where $fw^p(f_j)$ is the weight of the jth feature (i.e., f_j) in pageview p, for $1 \leq j \leq r$.
- For the whole collection of pageviews in the site, we then have an $n \times r$ pageview-feature matrix $PFM = \{p_1, p_2, ..., p_n\}$.
- Goal of transformation: represent user sessions / user profiles as vectors of semantic / textual / concept features rather than as vectors of pageviews.
- Thus, user's session reflects the significance of various concepts / context features relevant to user's interaction.
- Formally, transformation is the multiplication of the user-pageview matrix UPM with the pageview-feature matrix PFM.
- The new matrix is: $TFM = \{t_1, t_2, ..., t_m\}$, where each t_i is a r-dimensional vector over the feature space.

7.3 Data mining

Clustering using a standard clustering algorithm such as k-means, results in three clusters of user transactions. This example indicates that the resulting user segment is clearly interested in items B and F and to a lesser degree in item A:

		A	B	C	D	E	F
Cluster 0	user 1	0	0	1	1	0	0
	user 4	0	0	1	1	0	0
	user 7	0	0	1	1	0	0
Cluster 1	user 0	1	1	0	0	0	1
	user 3	1	1	0	0	0	1
	user 6	1	1	0	0	0	1
	user 9	0	1	1	0	0	1
Cluster 2	user 2	1	0	0	1	1	0
	user 5	1	0	0	1	1	0
	user 8	1	0	1	1	1	0

Aggregated Profile for Cluster 1	
Weight	Pageview
1.00	B
1.00	F
0.75	A
0.25	C

Figure 35: Derivation of aggregate profiles from Web transaction clusters

Clustering enhanced transaction matrix may reveal segments of users that have common interests in different concepts as indicated from their navigational behaviors:

	web	data	mining	business	intelligence	marketing	ecommerce	search	information	retrieval
user1	2	1	1	1	2	2	1	2	3	3
user2	1	1	1	2	3	3	1	1	2	2
user3	2	3	3	1	1	1	2	1	2	2
user4	3	2	2	1	2	2	1	2	4	4
user5	1	1	1	2	3	3	1	1	2	2
user6	3	2	2	1	2	2	1	2	4	4

Figure 36: Content enhanced transaction matrix

Preferences of user are matched on the left-hand side X of each rule (e.g., $X \rightarrow Y$), and the items on the right-hand side of the matching rules can be used as potential recommendations. This also enables Web sites to organize the site content more efficiently, or to provide effective cross-sale product recommendations:

Transactions
A, B, D, E
A, B, E, C, D
A, B, E, C
B, E, B, A, C
D, A, B, E, C

Size 1		Size 2		Size 3		Size 4	
Itemset	Supp.	Itemset	Supp.	Itemset	Supp.	Itemset	Supp.
A	5	A,B	5	A,B,C	4	A,B,C,E	4
B	5	A,C	4	A,B,E	5		
C	4	A,E	5	A,C,E	4		
E	5	B,C	4	B,C,E	4		
		B,E	5				
		C,E	4				

Figure 37: Web transactions and resulting frequent itemsets (minsup = 4)