**Data warehouse implementation& Data mining:** Efficient Data Cubecomputation: An overview, Indexing OLAP Data: Bitmap index and join index,Efficient processing of OLAP Queries, OLAP server Architecture ROLAP versusMOLAP Versus HOLAP**:** Introduction: What is data mining, Challenges, DataMining Tasks, and Data: Types of Data, Data Quality, Data Preprocessing, Measures of Similarity and Dissimilarity -

-------------------------------------------------------------------------------------------------------------

Data warehouses contain huge volumes of data. OLAP servers demand that decision support queries be answered in the order of seconds. Therefore, it is crucial for data warehouse systems to support highly efficient cube computation techniques, access methods, and query processing techniques. In this section, we present an overview of methods for the efficient implementation of data warehouse systems.

## Efficient Data Cube computation: An overview

Data cube computation is an essential task in data warehouse implementation. The pre computation of all or part of a data cube can greatly reduce the response time and enhance the performance of on-line analytical processing. At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions. In SQL terms, these aggregations are referred to as group-by's. Each group-by can be represented by a *cuboid*, where the set of group-by's forms a lattice of cuboids defining a data cube. In this subsection, we explore issues relating to the efficient computation of data cubes.

**The compute cube Operator and the Curse of Dimensionality:**

One approach to cube computation extends SQL so as to include a compute cube operator.The compute cube operator computes aggregates over all subsets of the dimensions specified in the operation. This can require excessive storage space, especially for large numbers of dimensions. We start with an intuitive look at what is involved in the efficient computation of data cubes.

**Example 4.6 A data cube is a lattice of cuboids.** Suppose that you want to create a data cube for *AllElectronics* sales that contains the following: *city*, *item*, *year*, and *sales in dollars*. You want to be able to analyze the data, with queries such as the following:

"*Compute the sum of sales, grouping by city and item.*"

"*Compute the sum of sales, grouping by city*."

"*Compute the sum of sales, grouping by item.*"

What is the total number of cuboids, or group-by's, that can be computed for this data cube? Taking the three attributes, *city*, *item*, and *year*, as the dimensions for the data cube, and *sales in dollars* as the measure, the total number of cuboids, or groupby's, that can be computed for this data cube is $2^3 = 8$. The possible group-by's are the following: {(*city*, *item*, *year*), (*city*, *item*), (*city*, *year*), (*item*, *year*), (*city*), (*item*), (*year*),()}, where () means that the group-by is empty (i.e., the dimensions are not grouped). These group-by's form a lattice of cuboids for the data cube, as shown in Figure 4.14.

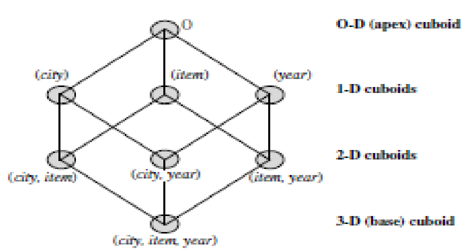



**Figure 4.14** Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains *city*, *item*, and *year* dimensions.

The **base cuboid** contains all three dimensions, *city*, *item*, and *year*. It can returnthe total sales for any combination of the three dimensions. The **apex cuboid**, or 0-Dcuboid, refers to the case where the group-by is empty. It contains the total sum of all sales. The base cuboid is the least generalized (most specific) of the cuboids. The apex cuboid is the most generalized (least specific) of the cuboids, and is often denoted as all. If we start at the apex cuboid and explore downward in the lattice, this is equivalent to drilling down within the data cube. If we start at the base cuboid and explore upward, this is akin to rolling up. An SQL query containing no group-by (e.g., "*compute the sum of total sales*") is a *zero dimensional operation*. An SQL query containing one group-by (e.g., *"compute the sum of sales, group-by city*") is a *one-dimensional operation*. A cube operator on $n$ dimensions is equivalent to a collection of group-by statements, one for each subset of the $n$ dimensions. Therefore, the cube operator is the $n$-dimensional generalization of the group-by operator. Similar to the SQL syntax, the data cube in Example 4.1 could be defined as define cube sales cube [city, item, year]: sum(sales in dollars)
For a cube with $n$ dimensions, there are a total of $2n$ cuboids, including the base cuboid.
A statement such as compute cube sales cube would explicitly instruct the system to compute the sales aggregate cuboids for all eight subsets of the set f*city, item, year*g, including the empty subset. A cube computation operator was first proposed and studied by Gray et al. [GCBC97].
Online analytical processing may need to access different cuboids for different queries. Therefore, it may seem like a good idea to compute in advance all or at least some of the cuboids in a data cube. Precomputation leads to fast response time and avoids some redundant computation. Most, if not all, OLAP products resort to some degree of precomputation of multidimensional aggregates. A major challenge related to this precomputation, however, is that the required storage space may explode if all the cuboids in a data cube are precomputed, especially when the cube has many dimensions. The storage requirements are even more excessive when many of the dimensions have associated concept hierarchies, each with multiple levels. This problem is referred to as the **curse of dimensionality**. The extent of the curse of dimensionality is illustrated here. "***How many cuboids are there in an* n*-dimensional data cube***?" If there were no hierarchies associated with each dimension, then the total number of cuboids for an $n$-dimensional data cube, as we have seen, is $2^n$. However, in practice, many dimensions do have hierarchies. For example, *time* is usually explored not at only one conceptual level (e.g., *year*), but rather at multiple conceptual levels such as in the hierarchy "*day* < *month* < *quarter* < *year*." For an $n$-dimensional data cube, the total number of cuboids=

The **base cuboid** contains all three dimensions, *city*, *item*, and *year*. It can return the total sales for any combination of the three dimensions. The **apex cuboid**, or 0-D cuboid, refers to the case where the group-by is empty. It contains the total sum of all sales. The base cuboid is the least generalized (most specific) of the cuboids. The apex cuboid is the most generalized (least specific) of the cuboids, and is often denoted as all. If we start at the apex cuboid and explore downward in the lattice, this is equivalent to drilling down within the data cube. If we start at the base cuboid and explore upward, this is akin to rolling up.
An SQL query containing no group-by (e.g., "*compute the sum of total sales*") is a *zero dimensional operation*. An SQL query containing one group-by (e.g., *"compute the sum of sales, group-by city*") is a *one-dimensional operation*. A cube operator on $n$ dimensions is equivalent to a collection of group-by statements, one for each subset of the $n$ dimensions. Therefore, the cube operator is the $n$-dimensional generalization of the group-by operator.

Similar to the SQL syntax, the data cube in Example 4.1 could be defined as

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

For a cube with *n* dimensions, there are a total of $2^n$ cuboids, including the base cuboid. A statement such as

**compute cube sales cube**

would explicitly instruct the system to compute the sales aggregate cuboids for all eight subsets of the set *city, item, year*, including the empty subset. A cube computation operator was first proposed and studied by Gray et al.. Online analytical processing may need to access different cuboids for different queries. Therefore, it may seem like a good idea to compute in advance all or at least some of the cuboids in a data cube. Precomputation leads to fast response time and avoids some redundant computation. Most, if not all, OLAP products resort to some degree of precomputation of multidimensional aggregates. A major challenge related to this precomputation, however, is that the required storage space may explode if all the cuboids in a data cube are precomputed, especially when the cube has many dimensions. The storage requirements are even more excessive when many of the dimensions have associated concept hierarchies, each with multiple levels.

This problem is referred to as the **curse of dimensionality**. The extent of the curse of dimensionality is illustrated here. "*How many cuboids are there in an* n-*dimensional data cube?*" If there were no hierarchies associated with each dimension, then the total number of cuboids for an *n*-dimensional data cube, as we have seen, is 2*n*. However, in practice, many dimensions do have hierarchies. For example, *time* is usually explored not at only one conceptual level (e.g., *year*), but rather at multiple conceptual levels such as in the hierarchy "*day* <*month* < *quarter* < *year*." For an *n*-dimensional data cube, the total number of cuboids that can be generated (including the cuboids generated by climbing up the hierarchies along each dimension) is

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

where *Li* is the number of levels associated with dimension *i*.

This formula is based on the fact that, at most, one abstraction level in each dimension will appear in a cuboid. By now, you probably realize that it is unrealistic to precompute and materialize all of the cuboids that can possibly be generated for a data cube (i.e., from a base cuboid). If there are many cuboids, and these cuboids are large in size, a more reasonable option is *partial materialization*; that is, to materialize only *some* of the possible cuboids that can be generated

**Partial Materialization: Selected Computationof Cuboids**

There are three choices for data cube materialization given a base cuboid:

**1. No materialization**: Do not precompute any of the "nonbase" cuboids. This leads to computing expensive multidimensional aggregates on-the-fly, which can be extremely slow.

**2. Full materialization**: Precompute all of the cuboids. The resulting lattice of computed cuboids is referred to as the *full cube*. This choice typically requires huge amounts of memory space in order to store all of the precomputed cuboids.

**3. Partial materialization**: Selectively compute a proper subset of the whole set of possible cuboids. Alternatively, we may compute a subset of the cube, which contains only those cells that satisfy some user-specified criterion, such as where the tuple count of each cell is above some threshold. We will use the term *subcube* to refer to the latter case, where only some of the cells may be precomputed for various cuboids. Partial materialization represents an interesting trade-off between storage space and response time.

The partial materialization of cuboids or subcubes should consider three factors: (1) identify the subset of cuboids or subcubes to materialize; (2) exploit the materialized cuboids or subcubes during query processing; and (3) efficiently update the materialized cuboids or subcubes during load and refresh

**Indexing OLAP Data: Bitmap Index and Join Index**

To facilitate efficient data accessing, most data warehouse systems support index structures and materialized views.

The bitmap indexing method is popular in OLAP products because it allows quick searching in data cubes. The bitmap index is an alternative representation of the*record ID (RID)* list. In the bitmap index for a given attribute, there is a distinct bitvector, ***Bv***, for each value *v* in the attribute's domain. If a given attribute's domain consistsof *n* values, then *n* bits are needed for each entry in the bitmap index (i.e., there are*n* bit vectors). If the attribute has the value *v* for a given row in the data table, then thebit representing that value is set to 1 in the corresponding row of the bitmap index. Allother bits for that row are set to 0.

Bitmap indexing is advantageous compared to hash and tree indices. It is especiallyuseful for low-cardinality domains because comparison, join, and aggregation operationsare then reduced to bit arithmetic, which substantially reduces the processing time.Bitmap indexing leads to significant reductions in space and input/output (I/O) since astring of characters can be represented by a single bit. For higher-cardinality domains,the method can be adapted using compression techniques.

**Bitmap indexing.** In the *AllElectronics* data warehouse, suppose the dimension *item* at the top level has four values (representing item types): *"home entertainment," "computer," "phone,"* and *"security."* Each value (e.g., *"computer"*) is represented by a bit vector in the *item* bitmap index table. Suppose that the cube is stored as a relation table with 100,000 rows. Because the domain of *item* consists of four values, the bitmap index table requires four bit vectors (or lists), each with 100,000 bits. Figure shows a base (data) table containing the dimensions *item* and *city*, and its mapping to bitmap index tables for each of the dimensions. ■

Base table

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

*item* bitmap index table

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

*city* bitmap index table

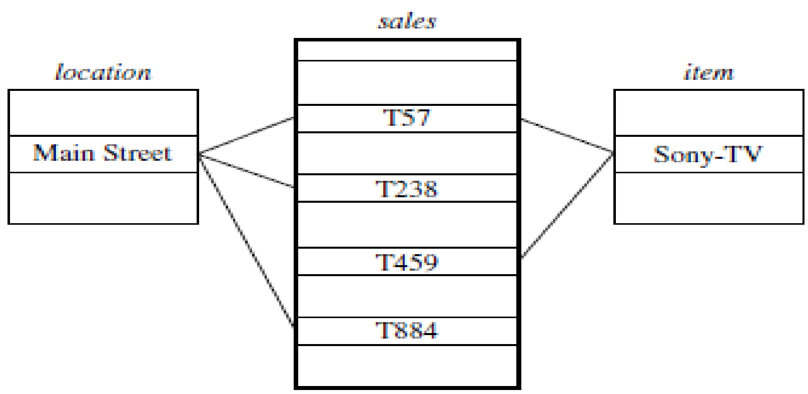
| *RID* | V | T |
|---|---|---|
| R1 | 1 | 0 |
| R2 | 1 | 0 |
| R3 | 1 | 0 |
| R4 | 1 | 0 |
| R5 | 0 | 1 |
| R6 | 0 | 1 |
| R7 | 0 | 1 |
| R8 | 0 | 1 |

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

Indexing OLAP data using bitmap indices.

The **join indexing** method gained popularity from its use in relational database query processing. Traditional indexing maps the value in a given column to a list of rows having that value. In contrast, join indexing registers the joinable rows of two relations from a relational database. For example, if two relations *R.RID*, *A*/ and *S.B*, *SID*/ join on the attributes *A* and *B*, then the join index record contains the pair .*RID*, *SID*/, where *RID* and *SID* are record identifiers from the *R* and *S* relations, respectively. Hence, the join index records can identify joinable tuples without performing costly join operations. Join indexing is especially useful for maintaining the relationship between a foreign key and its matching primary keys, from the joinable relation. Join indices may span multiple dimensions to form composite join indices. We can use join indices to identify sub cubes that are of interest.

**Join indexing.** In Example , we defined a star schema for *AllElectronics* of the form "*sales_star* [*time, item, branch, location*]: *dollars_sold* = sum (*sales_in_dollars*)." An example of a join index relationship between the *sales* fact table and the *location* and *item* dimension tables is shown in Figure For example, the "Main Street" value in the *location* dimension table joins with tuples T57, T238, and T884 of the *sales* fact table. Similarly, the "Sony-TV" value in the *item* dimension table joins with tuples T57 and T459 of the *sales* fact table. The corresponding join index tables are shown in Figure




Linkages between a *sales* fact table and *location* and *item* dimension tables.

Join index table for *location/sales*

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

Join index table for *item/sales*

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

Join index table linking *location* and *item* to *sales*

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

**Figure 4.17** Join index tables based on the linkages between the *sales* fact table and the *location* and *item* dimension tables shown in Figure 4.16.

**Efficient Processing of OLAP Queries**

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:** This involves transforming 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 may correspond to selection and/or projection operations on a materialized cuboid.

**2. Determine to which materialized cuboid(s) the relevant operations should be applied:** This involves identifying all of the materialized cuboids that may potentially be used to answer the query, pruning the set using knowledge of "dominance "relationships among the cuboids, estimating the costs of using the remaining materialized cuboids, and selecting the cuboid with the least cost.

**OLAP query processing.** Suppose that we define a data cube for *AllElectronics* of the form "*sales_cube* [*time, item, location*]: sum(*sales_in_dollars*)." The dimension hierarchies used are "*day* < *month* < *quarter* < *year*" for *time*; "*item_name* < *brand* < *type*" for *item*; and "*street* < *city* < *province_or_state* < *country*" for *location*.

Suppose that the query to be processed is on {*brand, province_or_state*}, with the selection constant "*year* = *2010*." Also, suppose that there are four materialized cuboids available, as follows:

- cuboid 1: {*year, item_name, city*}
- cuboid 2: {*year, brand, country*}
- cuboid 3: {*year, brand, province_or_state*}
- cuboid 4: {*item_name, province_or_state*}, where *year* = 2010

"*Which of these four cuboids should be selected to process the query?*" Finer-granularity data cannot be generated from coarser-granularity data. Therefore, cuboid 2 cannot be used because *country* is a more general concept than *province_or_state*. Cuboids 1, 3, and 4 can be used to process the query because (1) they have the same set or a superset of the dimensions in the query, (2) the selection clause in the query can imply the selection in the cuboid, and (3) the abstraction levels for the *item* and *location* dimensions in these cuboids are at a finer level than *brand* and *province_or_state*, respectively.

"*How would the costs of each cuboid compare if used to process the query?*" It is likely that using cuboid 1 would cost the most because both *item_name* and *city* are at a lower level than the *brand* and *province_or_state* concepts specified in the query. If there are not many *year* values associated with *items* in the cube, but there are several *item_names* for each *brand*, then cuboid 3 will be smaller than cuboid 4, and thus cuboid 3 should be chosen to process the query. However, if efficient indices are available for cuboid 4, then cuboid 4 may be a better choice. Therefore, some cost-based estimation is required to decide which set of cuboids should be selected for query processing. ■

## OLAP Server Architectures: ROLAP versus MOLAP versus HOLAP

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

**Multidimensional OLAP (MOLAP) servers:** These servers support multidimensional data views through *array-based multidimensional storage engines*. They map multidimensional views directly to data cube array structures. The advantage of using a data cube is that it allows fast indexing to pre computed summarized data.

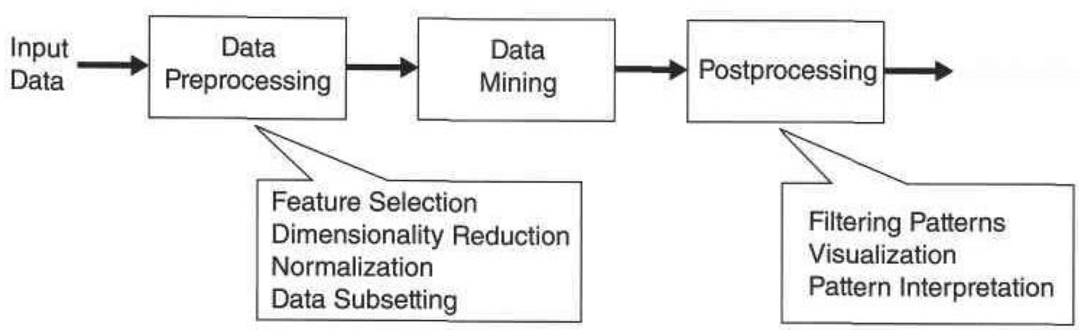
**Hybrid OLAP (HOLAP) servers:** The hybrid OLAP approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may allow large volumes of detailed data to be stored in a relational database, while aggregations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

# What is data mining?

**Data mining is the process of automatically discovering useful information in large data repositories.** Data mining techniques are deployed to scour large databases in order to find novel and useful patterns that might otherwise remain unknown. They also provide capabilities to predict the outcome of a future observation, such as predicting whether a newly arrived. Customer will spend more than $100 at a department store. Not all information discovery tasks are considered to be data mining. For example, Looking up individual records using a database management system or finding particular Web pages via a query to an Internet search engine are tasks related to the area of information retrieval. Although such tasks are important and may involve the use of the sophisticated algorithms and data structures, they rely on traditional computer science techniques and obvious features of the data to create index structures for efficiently organizing and retrieving information. Nonetheless, data mining techniques have been used to enhance information retrieval systems. Data Mining and Knowledge Discovery Data mining is an integral part of knowledge discovery in databases (KDD), which is the overall process of converting raw data into useful information, as shown in Figure 1.1. This process consists of a series of transformation steps, from data preprocessing to post processing of data mining results.

- Extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) patterns or knowledge from huge amount of data

# Knowledge Discovery (KDD) Process




Data mining is an integral part of knowledge discovery in databases (KDD), which is the overall process of converting raw data into useful information, as shown in Figure 1.1. This process consists of a series of transformation steps, from data preprocessing to post processing of data mining results.

The input data can be stored in a variety of formats (flat files, spreadsheets, or relational tables) and may reside in a centralized data repository or be distributed across multiple sites. The purpose of preprocessing is to transform the raw input data into an appropriate format for subsequent analysis. The steps involved in data preprocessing include fusing data from multiple sources, cleaning data to remove noise and duplicate observations, and selecting records and features that are relevant to the data mining task at hand. Because of the many ways data can be collected and stored, data preprocessing is perhaps the most laborious and time-consuming step in the overall knowledge discovery process.

“Closing the loop" is the phrase often used to refer to the process of integrating data mining results into decision support systems. For example, in business applications, the insights offered by data mining results can be integrated with campaign management tools so that effective marketing promotions can be conducted and tested. Such integration requires a post processing step that ensures that only valid and useful results are incorporated into the decision support system. An example of post processing is visualization, which allows analysts to explore the data and the data mining results from a variety of viewpoints. Statistical measures or hypothesis testing methods can also be applied during post processing to eliminate spurious data mining results.

## Types of Data:

**A Data set** is a Collection of data objects and their attributes.

A **data object** is also known as record, point, case, sample, entity, or instance.

An **attribute** is a property or characteristic of an object. Attribute is also known as variable, field, characteristic, or feature.

**There are different types of attributes:**

**Nominal** Examples: ID numbers, eye color, zip codes

**Ordinal** Examples: rankings (e.g., taste of potato chips on a scale from 1-10), grades, height in {tall, medium, short}

**Interval** Examples: calendar dates, temperatures in Celsius or Fahrenheit.

**Ratio** Examples: temperature in Kelvin, length, time, counts.

| Attribute Type | Description | Examples | Operations |
|---|---|---|---|
| Nominal | The values of a nominal attribute are just different names, i.e., nominal attributes provide only enough information to distinguish one object from another. ($=, \neq$) | zip codes, employee ID numbers, eye color, sex: {*male, female*} | mode, entropy, contingency correlation, $\chi^2$ test |
| Ordinal | The values of an ordinal attribute provide enough information to order objects. ($<, >$) | hardness of minerals, {*good, better, best*}, grades, street numbers | median, percentiles, rank correlation, run tests, sign tests |
| Interval | For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. ($+, -$) | calendar dates, temperature in Celsius or Fahrenheit | mean, standard deviation, Pearson's correlation, $t$ and $F$ tests |
| Ratio | For ratio variables, both differences and ratios are meaningful. ($*, /$) | temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current | geometric mean, harmonic mean, percent variation |

**Nominal and ordinal attributes are collectively referred to as categorical or qualitative attributes.** As the name suggests, qualitative attributes, such as employee ID, lack most of the properties of numbers. Even if they are represented by numbers, i.e., integers, they should be treated more like symbols. The remaining two types of attributes, **interval and ratio, are collectively referred to as quantitative or numeric attributes**. Quantitative attributes are represented by numbers and have most of the properties of numbers. Note that quantitative attributes can be integer-valued or continuous

### Describing Attributes by the Number of Values

An independent way of distinguishing between attributes is by the number of values they can take.

Discrete: A **discrete attribute has a finite or countably infinite set of values**. Such attributes can be categorical, such as zip codes or ID numbers, or numeric, such as counts. Discrete attributes are often represented using integer variables. **Binary attributes are a special case of discrete attributes and assume only two values**, e.g., true/false, yes/no, male/female, or 0 or 1. Binary attributes are often represented as Boolean variables, or as integer variables that only take the values 0 or 1.

Continuous A **continuous attribute is one whose values are real numbers**. Examples include attributes such as temperature, height, or weight. Continuous attributes are typically represented as floating-point variables. Practically, real values can only be measured and represented with limited precision.

Binary attributes where **only non-zero values are important are called asymmetric binary attributes**. This type of attribute is particularly important for association analysis

It is also possible to have **discrete or continuous asymmetric features**. For instance, if the number of credits associated with each course is recorded, then the resulting data set will consist of asymmetric discrete or continuous attributes.

## Types of Data Sets

There are many types of data sets, and as the field of data mining develops and matures, a greater variety of data sets become available for analysis. In this section, we describe some of the most common types. For convenience, **we have grouped the types of data sets into three groups:** record data, graph based data, and ordered data. These categories do not cover all possibilities and other groupings are certainly possible.

### General Characteristics of Data Sets

Before providing details of specific kinds of data sets, we discuss **three characteristics** that apply to many data sets and have a significant impact on the data mining techniques that are used: **dimensionality, sparsity, and resolution.**

**Dimensionality** The **dimensionality of a data set is the number of attributes that the objects in the data set possess.** Data with a small number of dimensions tends to be qualitatively different than moderate or high-dimensional data. Indeed, the difficulties associated with analyzing high-dimensional data are sometimes referred to as the **curse of dimensionality**. Because of this, an important motivation in preprocessing the data is dimensionality reduction..

**Sparsity** For some data sets, such as those with asymmetric features, most attributes of an object have values of 0; in many cases less than 1% of the entries are non-zero. In practical terms, **sparsity is an advantage because usually only the non-zero values need to be stored and manipulated.** This results in significant savings with respect to computation time and storage. Furthermore some data mining algorithms work well only for sparse data.

**Resolution** It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For instance, the surface of the Earth seems very uneven at a resolution of a few meters, but is relatively smooth at a resolution of tens of kilometers. The patterns in the data also depend on the level of resolution. If the resolution is too fine, a pattern may not be visible or may be buried in noise; if the resolution is too coarse, the pattern may disappear. For 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

## 1. Record Data

Much data mining work assumes that the data set is a collection of records (data objects), each of which consists of a fixed set of data fields (attributes). See Figure 2.2(a). For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes. Record data is usually stored either in flat files or in relational databases. Relational databases are certainly more than a collection of records, but data mining often does not use any of the additional information available in a relational database. Rather, the database serves as a convenient place to find records. Different types of record data are described below and are illustrated in Figure 2.2.

### Transaction or Market Basket Data

Transaction data is a special type of record data, where each record (transaction) involves a set of items. Consider a grocery store. The set of products purchased by a customer during one shopping trip constitutes a transaction, while the individual products that were purchased are the items. This type of data is called market basket data because the items in each record are the products in a person's "market basket."

| Tid | Refund | Marital Status | Taxable Income | Defaulted Borrower |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| 4 | Yes | Married | 120K | No |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married | 60K | No |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 9 | No | Married | 75K | No |
| 10 | No | Single | 90K | Yes |

(a) Record data.

| TID | ITEMS |
|---|---|
| 1 | Bread, Soda, Milk |
| 2 | Beer, Bread |
| 3 | Beer, Soda, Diaper, Milk |
| 4 | Beer, Bread, Diaper, Milk |
| 5 | Soda, Diaper, Milk |

(b) Transaction data.

| Projection of x Load | Projection of y Load | Distance | Load | Thickness |
|---|---|---|---|---|
| 10.23 | 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 |

(c) Data matrix.

| | team | coach | play | ball | score | game | win | lost | timeout | season |
|---|---|---|---|---|---|---|---|---|---|---|
| Document 1 | 3 | 0 | 5 | 0 | 2 | 6 | 0 | 2 | 0 | 2 |
| Document 2 | 0 | 7 | 0 | 2 | 1 | 0 | 0 | 3 | 0 | 0 |
| Document 3 | 0 | 1 | 0 | 0 | 1 | 2 | 2 | 0 | 3 | 0 |

(d) Document-term matrix.

**Figure 2.2.** Different variations of record data.

### The Data Matrix

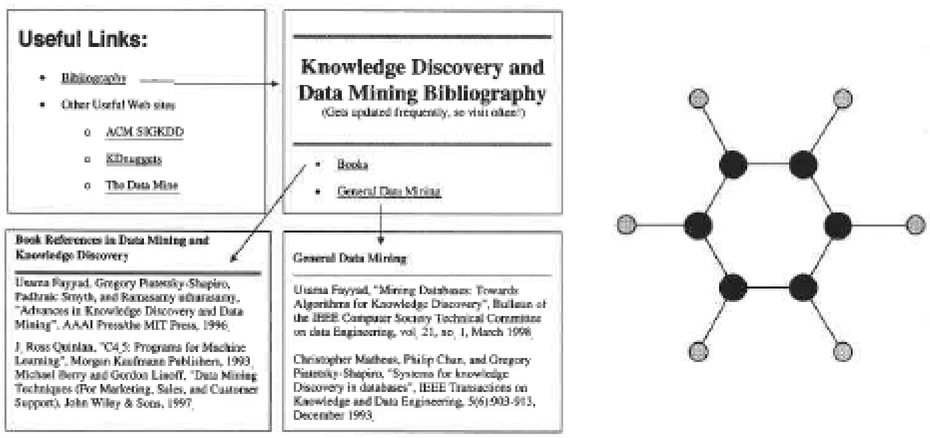
If the data objects in a collection of data all have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object. A set of such data objects can be interpreted as an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute. This matrix is called a data matrix or a pattern matrix. A data matrix is a variation of record data, but because it consists of numeric attributes, standard matrix operation can be applied to transform and manipulate the data. Therefore, the data matrix is the standard data format for most statistical data.

### The Sparse Data Matrix

A sparse data matrix is a special case of a data matrix in which the attributes are of the same type and are **asymmetric; i.e., only non-zero values are important**. Transaction data is an example of a sparse data matrix that has only 0, 1 entries. Another common example is document data. In particular, if the order of the terms (words) in a document is ignored then a document can be represented as a **term vector,** where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation of a collection of documents is often called a **document-term matrix**. Figure 2.2(d) shows a sample document-term matrix. The documents are the rows of this matrix, while the terms are the columns. In practice, only the non-zero entries of sparse data matrices are stored.

## 2. Graph-Based Data

A graph can sometimes be a convenient and powerful representation for data. We consider two specific cases: (1) the graph captures relationships among data objects and (2) the data objects themselves are represented as graphs.




(a) Linked Web pages. (b) Benzene molecule.

**Figure 2.3.** Different variations of graph data.

**Data with Relationships among Objects** The relationships among objects frequently convey important information. In such cases, the data is often represented as a graph. In particular, the data objects are mapped to nodes of the graph, while the relationships among objects are captured by the links between objects and link properties, such as direction and weight. Consider Web pages on the **World Wide Web**, which contain both text and links to other pages. In order to process search queries, Web search engines collect and process Web pages to extract their contents. It is well known, however, that the links to and from each page provide a great deal of information about the relevance of a Web page to a query, and thus, must also be taken into consideration. Figure 2.3(a) shows a set of linked Web pages.

**Data with Objects** That Are Graphs If objects have structure, that is, the objects contain sub objects that have relationships, then such objects are frequently represented as graphs. For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds. Figure 2.3(b) shows a ball-and-stick diagram of the chemical compound **benzene,** which contains atoms of carbon (black) and hydrogen (gray). A graph representation makes it possible to determine which substructures occur frequently in a set of compounds and to ascertain whether the presence of any of these substructures is associated with the presence or absence of certain chemical properties, such as melting point or heat of formation.

## 3. Ordered Data

For some types of data, the attributes have relationships that involve order in time or space. Different types of ordered data are described next and are shown in Figure 2.4. **Sequential Data** Sequential data, also referred to as **temporal data**, can be thought of as an extension of record data, where each record has a **time associated with it**. Consider a retail transaction data set that also stores the time at which the transaction took place. This time information makes it possible to find patterns such as "candy sales peak before Halloween." A time can also be associated with each attribute. For example, each record could be the purchase history of a customer, with a listing of items purchased at different times. Using this information, it is possible to find patterns such as "people who buy DVD players tend to buy DVDs in the period immediately following the purchase." Figure 2.a shows an example of sequential transaction data. There are fi.ve different times- t1, t2, t3, t4, and t5; three different customers-Cl, C2, and C3; and five different items A, B, C, D, and E. In the top table, each row corresponds to the items purchased at a particular time by each customer. For instance, at time t3, customer C2 purchased items A and D. In the bottom table, the same information is displayed, but each row corresponds to a particular customer. Each row contains information on each transaction involving the customer, where a transaction is considered to be a set of items and the time at which those items were purchased. For example, customer C3 bought items A and C at time t2.

**Sequence Data**

Sequence data consists of a data set that is a sequence of individual entities, such as a sequence of words or letters. It is quite similar to sequential data, except that there are no time stamps; instead, there are positions in an ordered sequence. For example, the **genetic information of plants and animals can be represented in the form of sequences of nucleotides that are known as genes**. Many of the problems associated with genetic sequence data involve predicting similarities in the structure and function of genes from similarities in nucleotide sequences. Figure 2.4(b) shows a section of the human genetic code expressed using the four nucleotides from which all DNA is constructed: A, T, G, and C.

**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**. For example, a financial data set might contain objects that are time series of the daily prices of various stocks. As another example, consider Figure 2.4(c), which shows a time series of the average monthly temperature for

Minneapolis during the years 1982 to 1994. When working with **temporal data**, it is important to consider temporal autocorrelation; i.e., if two measurements are close in time, then the values of those measurements are often very similar.

### Spatial Data

Some objects have **spatial attributes, such as positions or areas,** as well as other types of attributes. An example of spatial data is **weather data (precipitation, temperature, pressure)** that is collected for a variety of geographical locations. An important aspect of spatial data is spatial autocorrelation; i.e., objects that are physically close tend to be similar in other ways as well. Thus, two points on the Earth that are close to each other usually have similar values for temperature and rainfall.

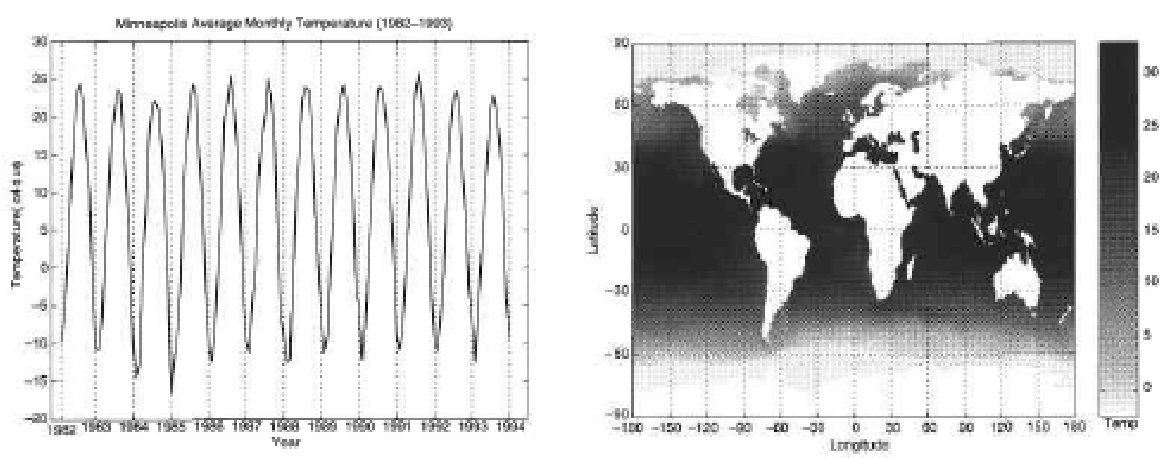
| Time | Customer | Items Purchased |
|---|---|---|
| t1 | C1 | A, B |
| t2 | C3 | A, C |
| t2 | C1 | C, D |
| t3 | C2 | A, D |
| t4 | C2 | E |
| t5 | C1 | A, E |

| 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) |

(a) Sequential transaction data.

GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG

(b) Genomic sequence data.




(c) Temperature time series. (d) Spatial temperature data.

Figure 2.4. Different variations of ordered data.

## Task of Data Mining

Data mining tasks are generally divided into **two major categories**:

**Predictive tasks.** The objective of these tasks is to predict the value of a particular attribute based on the values of other attributes. The attribute to be predicted is commonly known as the target or dependent variable, while the attributes used for making the prediction are known as the explanatory or independent variables.

**Descriptive tasks**. Here, the objective is to derive patterns (correlations, trends, clusters, trajectories, and anomalies) that summarize the underlying relationships in data. Descriptive data mining tasks are often exploratory in nature and frequently require postprocessing techniques to validate and explain the results.

**Predictive modeling** refers to the task of building a model for the target variable as a function of the explanatory variables. There are **two types of predictive modeling tasks: classification, which is used for discrete target variables, and regression, which is used for continuous target variables**. For example, predicting whether a Web user will make a purchase at an online bookstore is a classification task because the target variable is binary-valued. On the other hand, forecasting the future price of a stock is a regression task because price is a continuous-valued attribute. The goal of both tasks is to learn a model that minimizes the error between the predicted and true values of the target variable.

**Example 1.1 (Predicting the Type of a Flower).** Consider the task of predicting a species of flower based on the characteristics of the flower. In particular, consider classifying an Iris flower as to whether it belongs to one of the following **three Iris species: Setosa, Versicolour, or Virginica**. To perform this task, we need a data set containing the characteristics of various flowers of these three species. In addition to the species of a flower, this data set contains **four other attributes: sepal width, sepal length, petal length, and petal width.** Figure 1.4 shows a plot of petal width versus petal length for the 150 flowers in the Iris data set. **Petal width is broken into the categories low, medium, and high**, which correspond to the **intervals [0, 0.75), [0.75, 1.75), [1.75, oo)**, respectively. Also**, petal length is broken into categories low, medium, and high, which correspond to the intervals [0,2.5), [2.5,5), [5, oo),** respectively. Based on these categories of petal width and length, the following rules can be derived**:**

**Petal width low and petal length low implies Setosa.**
**Petal width medium and petal length medium implies Versicolour.**
**Petal width high and petal length high implies Virginica.**

While these rules do not classify all the flowers, they do a good (but not perfect) job of classifying most of the flowers. Note that flowers from the Setosa species are well separated from the Versicolour and Virginica species with respect to petal width and length, but the latter two species overlap somewhat with respect to these attributes.

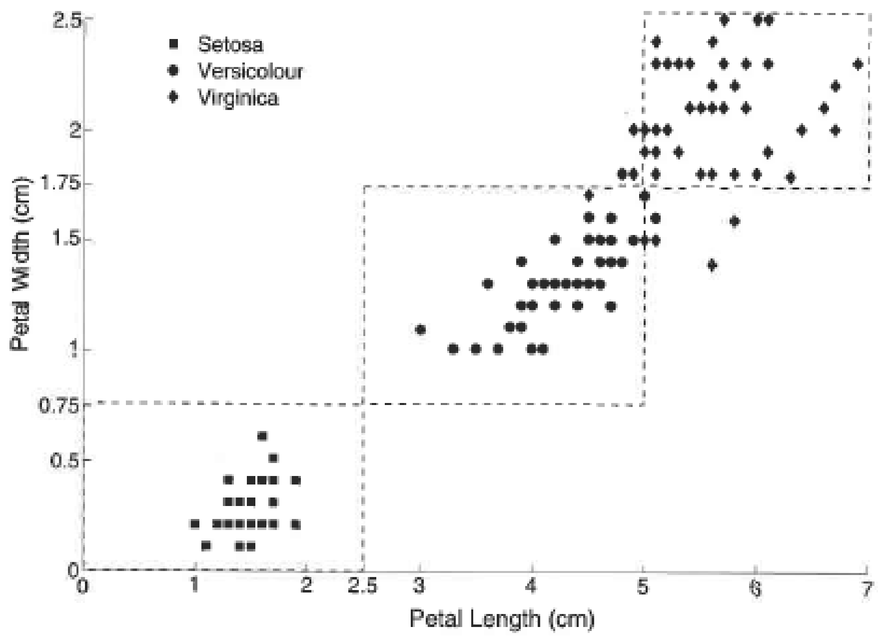



**Figure 1.4.** Petal width versus petal length for 150 Iris flowers.

**Association analysis** is used to discover patterns that describe strongly associated features in the data. The discovered patterns are typically represented in the form of implication rules or feature subsets. Because of the exponential size of its search space, the goal of association analysis is to extract the most interesting patterns in an efficient manner. Useful applications of association analysis include finding groups of genes that have related functionality, identifying Web pages that are accessed together, or understanding the relationships between different elements of Earth's climate system.

**Example 1.2 (Market Basket Analysis).** The transactions shown in Table 1.1 illustrate point-of-sale data collected at the checkout counters of a grocery store. Association analysis can be applied to find items that are frequently bought together by customers. For example, we may discover the rule {Diapers} → {Milk}, which suggests that customers who buy diapers also tend to buy milk. This type of rule can be used to identify potential cross-selling opportunities among related items.

**Table 1.1.** Market basket data.

| Transaction ID | Items |
|---|---|
| 1 | {Bread, Butter, Diapers, Milk} |
| 2 | {Coffee, Sugar, Cookies, Salmon} |
| 3 | {Bread, Butter, Coffee, Diapers, Milk, Eggs} |
| 4 | {Bread, Butter, Salmon, Chicken} |
| 5 | {Eggs, Bread, Butter} |
| 6 | {Salmon, Diapers, Milk} |
| 7 | {Bread, Tea, Sugar, Eggs} |
| 8 | {Coffee, Sugar, Chicken, Eggs} |
| 9 | {Bread, Diapers, Milk, Salt} |
| 10 | {Tea, Eggs, Cookies, Diapers, Milk} |

**Cluster analysis** seeks to find groups of closely related observations so that observations that belong to the same cluster are more similar to each other than observations that belong to other clusters. Clustering has been used to group sets of related customers, find areas of the ocean that have a significant impact on the Earth's climate, and compress data.

**Example 1.3 (Document Clustering)**. The collection of news articles shown in Table 1.2 can be grouped based on their respective topics. Each article is represented as a set of word-frequency pairs (r, "), where tu is a word and c is the number of times the word appears in the article. There are two natural clusters in the data set. The first cluster consists of the first four articles, which correspond to news about the economy, while the second cluster contains the last four articles, which correspond to news about health care. A good clustering algorithm should be able to identify these two clusters based on the similarity between words that appear in the articles.

**Table 1.2.** Collection of news articles.

| Article | Words |
|---|---|
| 1 | dollar: 1, industry: 4, country: 2, loan: 3, deal: 2, government: 2 |
| 2 | machinery: 2, labor: 3, market: 4, industry: 2, work: 3, country: 1 |
| 3 | job: 5, inflation: 3, rise: 2, jobless: 2, market: 3, country: 2, index: 3 |
| 4 | domestic: 3, forecast: 2, gain: 1, market: 2, sale: 3, price: 2 |
| 5 | patient: 4, symptom: 2, drug: 3, health: 2, clinic: 2, doctor: 2 |
| 6 | pharmaceutical: 2, company: 3, drug: 2, vaccine: 1, flu: 3 |
| 7 | death: 2, cancer: 4, drug: 3, public: 4, health: 3, director: 2 |
| 8 | medical: 2, cost: 3, increase: 2, patient: 2, health: 3, care: 1 |

**Anomaly detection** is the task of identifying observations whose characteristics are significantly different from the rest of the data. Such observations are known as **anomalies or outliers**. The goal of an anomaly detection algorithm is to discover the real anomalies and avoid falsely labeling normal objects as anomalous. In other words, a good anomaly detector must have a high detection rate and a low false alarm rate. Applications of anomaly detection include the detection of fraud, network intrusions, unusual patterns of disease, and ecosystem disturbances.

**Example 1.4 (Credit Card Fraud Detection).** A credit card company records the transactions made by every credit card holder, along with personal information such as credit limit, age, annual income, and address. since the number of fraudulent cases is relatively small compared to the number of legitimate transactions, anomaly detection techniques can be applied to build a profile of legitimate transactions for the users. When a new transaction arrives, it is compared against the profile of the user. If the characteristics of the transaction are very different from the previously created profile, then the transaction is flagged as potentially fraudulent.

## The Origins of Data Mining

Brought together by the goal of meeting the challenges of the previous section, researchers from different disciplines began to focus on developing more efficient and scalable tools that could handle diverse types of data. This work, which culminated in the field of data mining, built upon the methodology and algorithms that researchers had previously used. **In particular, data mining draws upon ideas, such as (1) sampling, estimation, and hypothesis testing from statistics and (2) search algorithms, modeling techniques, and learning theories from artificial intelligence, pattern recognition, and machine learning.** Data mining has also been quick to adopt ideas from other areas, including **optimization, evolutionary computing, information theory, signal processing, visualization, and information retrieval**. A number of other areas also play key supporting roles. In particular, database systems are needed to provide support for efficient storage, indexing, and query processing. Techniques from high performance (parallel) computing are often important in addressing the massive size of some data sets. Distributed techniques can also help address the issue of size and are essential when the data cannot be gathered in one location

Figure 1.2 shows the relationship of data mining to other areas.

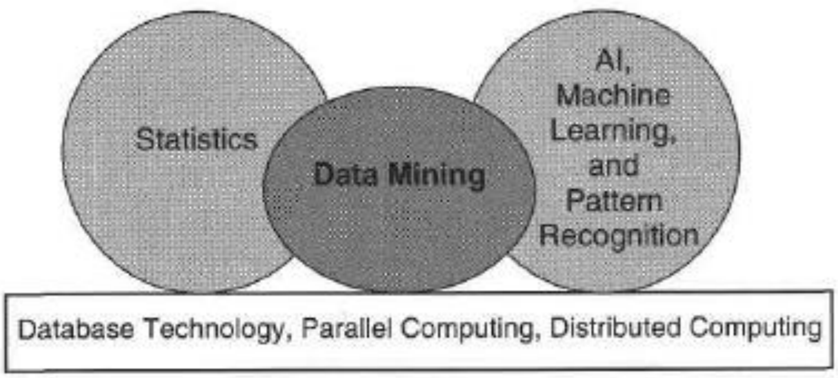



**Figure 1.2.** Data mining as a confluence of many disciplines.

## Motivating Challenges of Data Mining:

Traditional data analysis techniques have often encountered practical difficulties in meeting the challenges posed by new data sets. The following are some of the specific challenges that motivated the development of data mining.

**Scalability** Because of advances in data generation and collection, data sets with sizes of gigabytes, terabytes, or even petabytes are becoming common. If data mining algorithms are to handle these massive data sets, then they must be scalable. Many data mining algorithms employ special search strategies to handle exponential search problems. Scalability may also require the implementation of novel data structures to access individual records in an efficient manner. For instance, out-of-core algorithms may be necessary when processing data sets that cannot fit into main memory. Scalability can also be improved by using sampling or developing parallel and distributed algorithms.

**High Dimensionality** It is now common to encounter data sets with hundreds or thousands of attributes instead of the handful common a few decades ago. In bioinformatics, progress in microarray technology has produced gene expression data involving thousands of features. Data sets with temporal or spatial components also tend to have high dimensionality. For example, consider a data set that contains measurements of temperature at various locations. If the temperature measurements are taken repeatedly for an extended period, the number of dimensions (features) increases in proportion to the number of measurements taken. Traditional data analysis techniques that were developed for low-dimensional data often do not work well for such high dimensional data. Also, for some data analysis algorithms, the computational complexity increases rapidly as the dimensionality (the number of features) increases.

**Heterogeneous and Complex Data** Traditional data analysis methods often deal with data sets containing attributes of the same type, either continuous or categorical. As the role of data mining in business, science, medicine, and other fields has grown, so has the need for techniques that can handle heterogeneous attributes. Recent years have also seen the emergence of more complex data objects. Examples of such non-traditional types of data include collections of Web pages containing semi-structured text and hyperlinks; DNA data with sequential and three-dimensional structure; and climate data that consists of time series measurements (temperature, pressure, etc.) at various locations on the Earth's surface. Techniques developed for mining such complex objects should take into consideration relationships in the data, such as temporal and spatial autocorrelation, graph connectivity, and parent-child relationships between the elements in semi-structured text and XML documents.

**Data ownership and Distribution** Sometimes, the data needed for an analysis is not stored in one location or owned by one organization. Instead, the data is geographically distributed among resources belonging to multiple entities. This requires the development of distributed data mining techniques. Among the key challenges faced by distributed data mining algorithms include (1) how to reduce the amount of communication needed to perform the distributed computation, (2) how to effectively consolidate the data mining results obtained from multiple sources, and (3) how to address data security issues.

**Non-traditional Analysis** The traditional statistical approach is based on a hypothesize-and-test paradigm. In other words, a hypothesis is proposed, an experiment is designed to gather the data, and then the data is analyzed with respect to the hypothesis. Unfortunately, this process is extremely labor intensive. Current data analysis tasks often require the generation and evaluation of thousands of hypotheses, and consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation. Furthermore, the data sets analyzed in data mining are typically not the result of a carefully designed experiment and often represent opportunistic samples of the data, rather than random samples. Also, the data sets frequently involve non-traditional types of data and data distributions.

# Data Preprocessing:

Data preprocessing steps should be applied to make the data more suitable for data mining. Data preprocessing is a broad area and consists of a number of different strategies and techniques that are interrelated in complex ways. We will present some of the most important ideas and approaches, and try to point out the interrelationships among them. Specifically, we will discuss the following topics:

- Aggregation
- Sampling
- Dimensionality reduction
- Feature subset selection
- Feature creation
- Discretization and binarization
- Variable transformation

**Aggregation**

Sometimes "less is more" and this is the case with aggregation, the combining of two or more objects into a single object. Consider a data set consisting of transactions (data objects) recording the daily sales of products in various store locations (Minneapolis, Chicago, Paris, ) for different days over the course of a year. See Table 2.4. One way to aggregate transactions for this data set is to replace all the transactions of a single store with a single storewide transaction. This reduces the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction, and the number of data objects is reduced to the number of stores.

**Table 2.4.** Data set containing information about customer purchases.

| Transaction ID | Item | Store Location | Date | Price | ... |
|---|---|---|---|---|---|
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | |
| 101123 | Watch | Chicago | 09/06/04 | $25.99 | ... |
| 101123 | Battery | Chicago | 09/06/04 | $5.99 | ... |
| 101124 | Shoes | Minneapolis | 09/06/04 | $75.00 | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | |

**Sampling**

**Sampling is a commonly used approach for selecting a subset of the data objects to be analyzed.** In statistics, it has long been used for both the preliminary investigation of the data and the final data analysis. Sampling can also be very useful in data mining. However, the motivations for sampling in statistics and data mining are often different. ***Statisticians use sampling because obtaining the entire set of data of interest is too expensive or time consuming***, while ***data miners sample because it is too expensive or time consuming to process all the data***. In some cases, using a sampling algorithm can reduce the data size to the point where a better, but more expensive algorithm can be used.

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**. In turn, a sample is representative if it has approximately the same property (of interest) as the original set of data. If the mean (average) of the data objects is the property of interest, then a sample is representative if it has a mean that is close to that of the original data. Because sampling is a statistical process, the representativeness of any particular sample will vary, and the best that we can do is choose a sampling scheme that guarantees a high probability of getting a representative sample. As discussed next, thisinvolves choosing the appropriate sample size and sampling techniques

### Sampling Approaches

There are many sampling techniques, but only a few of the most basic ones and their variations will be covered here. The simplest type of sampling is **simple random sampling**. For this type of sampling, there is an equal probability of selecting any particular item. There are two variations on random sampling (and other sampling techniques as well): (1) **sampling without replacement**—as each item is selected, it is removed from the set of all objects that together constitute the **population**, and (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.

When the population consists of different types of objects, with widely different numbers of objects, 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 types. 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 prespecified 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

The proper sample size can be difficult to determine, so **adaptive** or **progressive sampling** schemes are sometimes used. These approaches start with a small sample, and then increase the sample size until a sample of sufficient size has been obtained. While this technique eliminates the need to determine the correct sample size initially, it requires that there be a way to evaluate the sample to judge if it is large enough.

**Dimensionality Reduction**

- **Curse of dimensionality**
    - When dimensionality increases, data becomes increasingly sparse
    - Density and distance between points, which is critical to clustering, outlier analysis, becomes less meaningful
    - The possible combinations of subspaces will grow exponentially
- **Dimensionality reduction**
    - Avoid the curse of dimensionality
    - Help eliminate irrelevant features and reduce noise
    - Reduce time and space required in data mining
    - Allow easier visualization
- **Dimensionality reduction techniques**
    - Wavelet transforms
    - Principal Component Analysis
    - Supervised and nonlinear techniques (e.g., feature selection)

### 2.3.4 Feature Subset Selection

Another way to reduce the dimensionality is to use only a subset of the features. While it might seem that such an approach would lose information, this is not the case if redundant and irrelevant features are present. **Redundant features** duplicate much or all of the information contained in one or more other attributes. For example, the purchase price of a product and the amount of sales tax paid contain much of the same information. **Irrelevant features** contain almost no useful information for the data mining task at hand. For instance, students' ID numbers are irrelevant to the task of predicting students' grade point averages. Redundant and irrelevant features can reduce classification accuracy and the quality of the clusters that are found.

While some irrelevant and redundant attributes can be eliminated immediately by using common sense or domain knowledge, selecting the best subset of features frequently requires a systematic approach. The ideal approach to feature selection is to try all possible subsets of features as input to the data mining algorithm of interest, and then take the subset that produces the best results. This method has the advantage of reflecting the objective and bias of the data mining algorithm that will eventually be used. Unfortunately, since the number of subsets involving $n$ attributes is $2^n$, such an approach is impractical in most situations and alternative strategies are needed. There are three standard approaches to feature selection: embedded, filter, and wrapper.

**Embedded approaches** Feature selection occurs naturally as part of the data mining algorithm. Specifically, during the operation of the data mining algorithm, the algorithm itself decides which attributes to use and which to ignore. Algorithms for building decision tree classifiers, which are discussed in Chapter 4, often operate in this manner.

**Filter approaches** Features are selected before the data mining algorithm is run, using some approach that is independent of the data mining task. For example, we might select sets of attributes whose pairwise correlation is as low as possible.

**Wrapper approaches** These methods use the target data mining algorithm as a black box to find the best subset of attributes, in a way similar to that of the ideal algorithm described above, but typically without enumerating all possible subsets.

Since the embedded approaches are algorithm-specific, only the filter and wrapper approaches will be discussed further here.

### An Architecture for Feature Subset Selection

It is possible to encompass both the filter and wrapper approaches within a common architecture. The feature selection process is viewed as consisting of four parts: a measure for evaluating a subset, a search strategy that controls the generation of a new subset of features, a stopping criterion, and a validation procedure. Filter methods and wrapper methods differ only in the way in which they evaluate a subset of features. For a wrapper method, subset evaluation uses the target data mining algorithm, while for a filter approach, the evaluation technique is distinct from the target data mining algorithm. The following discussion provides some details of this approach, which is summarized in Figure 2.11.

Conceptually, feature subset selection is a search over all possible subsets of features. Many different types of search strategies can be used, but the search strategy should be computationally inexpensive and should find optimal or near optimal sets of features. It is usually not possible to satisfy both requirements, and thus, tradeoffs are necessary.

An integral part of the search is an evaluation step to judge how the current subset of features compares to others that have been considered. This requires an evaluation measure that attempts to determine the goodness of a subset of attributes with respect to a particular data mining task, such as classification

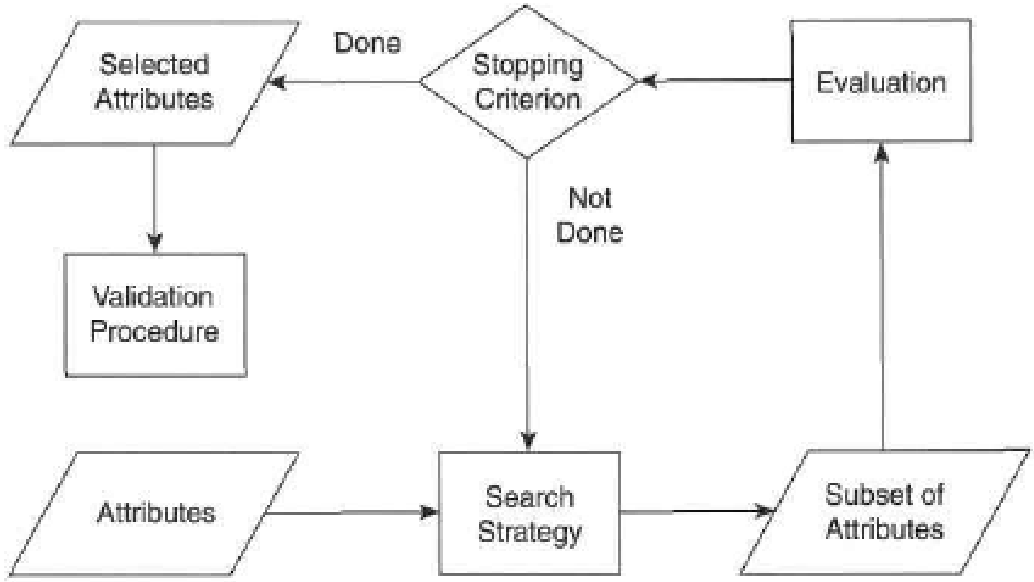



**Figure 2.11.** Flowchart of a feature subset selection process.

or clustering. For the filter approach, such measures attempt to predict how well the actual data mining algorithm will perform on a given set of attributes. For the wrapper approach, where evaluation consists of actually running the target data mining application, the subset evaluation function is simply the criterion normally used to measure the result of the data mining.

Because the number of subsets can be enormous and it is impractical to examine them all, some sort of stopping criterion is necessary. This strategy is usually based on one or more conditions involving the following: the number of iterations, whether the value of the subset evaluation measure is optimal or exceeds a certain threshold, whether a subset of a certain size has been obtained, whether simultaneous size and evaluation criteria have been achieved, and whether any improvement can be achieved by the options available to the search strategy.

Finally, once a subset of features has been selected, the results of the target data mining algorithm on the selected subset should be validated. A straightforward evaluation approach is to run the algorithm with the full set of features and compare the full results to results obtained using the subset of features. Hopefully, the subset of features will produce results that are better than or almost as good as those produced when using all features. Another validation approach is to use a number of different feature selection algorithms to obtain subsets of features and then compare the results of running the data mining algorithm on each subset.

### 2.3.6 Discretization and Binarization

Some data mining algorithms, especially certain classification algorithms, require that the data be in the form of categorical attributes. Algorithms that find association patterns require that the data be in the form of binary attributes. Thus, it is often necessary to transform a continuous attribute into a categorical attribute (**discretization**), and both continuous and discrete attributes may need to be transformed into one or more binary attributes (**binarization**). Additionally, if a categorical attribute has a large number of values (categories), or some values occur infrequently, then it may be beneficial for certain data mining tasks to reduce the number of categories by combining some of the values.

**Table 2.6.** Conversion of a categorical attribute to five asymmetric binary attributes.

| Categorical Value | Integer Value | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|---|
| *awful* | 0 | 1 | 0 | 0 | 0 | 0 |
| *poor* | 1 | 0 | 1 | 0 | 0 | 0 |
| *OK* | 2 | 0 | 0 | 1 | 0 | 0 |
| *good* | 3 | 0 | 0 | 0 | 1 | 0 |
| *great* | 4 | 0 | 0 | 0 | 0 | 1 |

#### Discretization of Continuous Attributes

Discretization is typically applied to attributes that are used in classification or association analysis. In general, the best discretization depends on the algorithm being used, as well as the other attributes being considered. Typically, however, the discretization of an attribute is considered in isolation.

Transformation of a continuous attribute to a categorical attribute involves two subtasks: deciding how many categories to have and determining how to map the values of the continuous attribute to these categories. In the first step, after the values of the continuous attribute are sorted, they are then divided into $n$ intervals by specifying $n-1$ **split points**. In the second, rather trivial step, all the values in one interval are mapped to the same categorical value. Therefore, the problem of discretization is one of deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals $\{(x_0, x_1], (x_1, x_2], \ldots, (x_{n-1}, x_n)\}$, where $x_0$ and $x_n$ may be $+\infty$ or $-\infty$, respectively, or equivalently, as a series of inequalities $x_0 < x \leq x_1, \ldots, x_{n-1} < x < x_n$.

### 2.3.7 Variable Transformation

A **variable transformation** refers to a transformation that is applied to all the values of a variable. (We use the term variable instead of attribute to adhere to common usage, although we will also refer to attribute transformation on occasion.) In other words, for each object, the transformation is applied to the value of the variable for that object. For example, if only the magnitude of a variable is important, then the values of the variable can be transformed by taking the absolute value. In the following section, we discuss two important types of variable transformations: simple functional transformations and normalization.

#### Simple Functions

For this type of variable transformation, a simple mathematical function is applied to each value individually. If $x$ is a variable, then examples of such transformations include $x^k$, $\log x$, $e^x$, $\sqrt{x}$, $1/x$, $\sin x$, or $|x|$. In statistics, variable transformations, especially *sqrt*, *log*, and $1/x$, are often used to transform data that does not have a Gaussian (normal) distribution into data that does. While this can be important, other reasons often take precedence in data mining. Suppose the variable of interest is the number of data bytes in a session, and the number of bytes ranges from 1 to 1 billion. This is a huge range, and it may be advantageous to compress it by using a $\log_{10}$ transformation. In this case, sessions that transferred $10^8$ and $10^9$ bytes would be more similar to each other than sessions that transferred 10 and 1000 bytes ($9 - 8 = 1$ versus $3 - 1 = 2$). For some applications, such as network intrusion detection, this may be what is desired, since the first two sessions most likely represent transfers of large files, while the latter two sessions could be two quite distinct types of sessions.

Variable transformations should be applied with caution since they change the nature of the data. While this is what is desired, there can be problems if the nature of the transformation is not fully appreciated. For instance, the transformation $1/x$ reduces the magnitude of values that are 1 or larger, but increases the magnitude of values between 0 and 1. To illustrate, the values $\{1, 2, 3\}$ go to $\{1, \frac{1}{2}, \frac{1}{3}\}$, but the values $\{1, \frac{1}{2}, \frac{1}{3}\}$ go to $\{1, 2, 3\}$. Thus, for all sets of values, the transformation $1/x$ reverses the order. To help clarify the effect of a transformation, it is important to ask questions such as the following: Does the order need to be maintained? 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? Exercise 17 on page 92 explores other aspects of variable transformation.

### Normalization or Standardization

Another common type of variable transformation is the **standardization** or **normalization** of a variable. (In the data mining community the terms are often used interchangeably. In statistics, however, the term normalization can be confused with the transformations used for making a variable **normal**, i.e., **Gaussian**.) The goal of standardization or normalization is to make an entire set of values have a particular property. A traditional example is that of "standardizing a variable" in statistics. If $\overline{x}$ is the mean (average) of the attribute values and $s_x$ is their standard deviation, then the transformation $x' = (x - \overline{x})/s_x$ creates a new variable that has a mean of 0 and a standard deviation of 1. If different variables are to be combined in some way, then such a transformation is often necessary to avoid having a variable with large values dominate the results of the calculation. To illustrate, consider comparing people based on two variables: age and income. For any two people, the difference in income will likely be much higher in absolute terms (hundreds or thousands of dollars) than the difference in age (less than 150). If the differences in the range of values of age and income are not taken into account, then the comparison between people will be dominated by differences in income. In particular, if the similarity or dissimilarity of two people is calculated using the similarity or dissimilarity measures defined later in this chapter, then in many cases, such as that of Euclidean distance, the income values will dominate the calculation.

The mean and standard deviation are strongly affected by outliers, so the above transformation is often modified. First, the mean is replaced by the **median**, i.e., the middle value. Second, the standard deviation is replaced by the **absolute standard deviation**. Specifically, if $x$ is a variable, then the absolute standard deviation of $x$ is given by $\sigma_A = \sum_{i=1}^{m} |x_i - \mu|$, where $x_i$ is the $i^{th}$ value of the variable, $m$ is the number of objects, and $\mu$ is either the mean or median. Other approaches for computing estimates of the location (center) and spread of a set of values in the presence of outliers are described in Sections 3.2.3 and 3.2.4, respectively. These measures can also be used to define a standardization transformation.

# Measuring Data Similarity and Dissimilarity

## Definitions

Informally, the **similarity** between two objects is a numerical measure of the degree to which the two objects are alike. Consequently, similarities are *higher* for pairs of objects that are more alike. Similarities are usually non-negative and are often between 0 (no similarity) and 1 (complete similarity).

The **dissimilarity** between two objects is a numerical measure of the degree to which the two objects are different. Dissimilarities are *lower* for more similar pairs of objects. Frequently, the term **distance** is used as a synonym for dissimilarity, although, as we shall see, distance is often used to refer to a special class of dissimilarities. Dissimilarities sometimes fall in the interval [0, 1], but it is also common for them to range from 0 to $\infty$.

**Distance**

**Euclidian formula:**

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2},$$

The Euclidean distance measure given in Equation 2.1 is generalized by the **Minkowski** distance metric shown in Equation 2.2,

$$d(\mathbf{x}, \mathbf{y}) = \left( \sum_{k=1}^{n} |x_k - y_k|^r \right)^{1/r}, \tag{2.2}$$

where $r$ is a parameter. The following are the three most common examples of Minkowski distances.

- $r = 1$. City block (Manhattan, taxicab, $L_1$ norm) distance. A common example is the **Hamming distance**, which is the number of bits that are different between two objects that have only binary attributes, i.e., between two binary vectors.
- $r = 2$. Euclidean distance ($L_2$ norm).
- $r = \infty$. Supremum ($L_{max}$ or $L_\infty$ norm) distance. This is the maximum difference between any attribute of the objects. More formally, the $L_\infty$ distance is defined by Equation 2.3

$$d(\mathbf{x}, \mathbf{y}) = \lim_{r \to \infty} \left( \sum_{k=1}^{n} |x_k - y_k|^r \right)^{1/r}. \tag{2.3}$$

## Similarity between object:

### Similarity Measures for Binary Data

Similarity measures between objects that contain only binary attributes are called **similarity coefficients**, and typically have values between 0 and 1. A value of 1 indicates that the two objects are completely similar, while a value of 0 indicates that the objects are not at all similar. There are many rationales for why one coefficient is better than another in specific instances.

Let $\mathbf{x}$ and $\mathbf{y}$ be two objects that consist of $n$ binary attributes. The comparison of two such objects, i.e., two binary vectors, leads to the following four quantities (frequencies):

$f_{00}$ = the number of attributes where $\mathbf{x}$ is 0 and $\mathbf{y}$ is 0
$f_{01}$ = the number of attributes where $\mathbf{x}$ is 0 and $\mathbf{y}$ is 1
$f_{10}$ = the number of attributes where $\mathbf{x}$ is 1 and $\mathbf{y}$ is 0
$f_{11}$ = the number of attributes where $\mathbf{x}$ is 1 and $\mathbf{y}$ is 1

**Simple Matching Coefficient** One commonly used similarity coefficient is the **simple matching coefficient** ($SMC$), which is defined as

$$SMC = \frac{\text{number of matching attribute values}}{\text{number of attributes}} = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}}. \tag{2.5}$$

**Jaccard Coefficient** Suppose that $\mathbf{x}$ and $\mathbf{y}$ are data objects that represent two rows (two transactions) of a transaction matrix (see Section 2.1.2). If each asymmetric binary attribute corresponds to an item in a store, then a 1 indicates that the item was purchased, while a 0 indicates that the product was not purchased. Since the number of products not purchased by any customer far outnumbers the number of products that were purchased, a similarity measure such as $SMC$ would say that all transactions are very similar. As a result, the Jaccard coefficient is frequently used to handle objects consisting of asymmetric binary attributes. The **Jaccard coefficient**, which is often symbolized by $J$, is given by the following equation:

$$J = \frac{\text{number of matching presences}}{\text{number of attributes not involved in 00 matches}} = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}. \tag{2.6}$$

**Example 2.17 (The SMC and Jaccard Similarity Coefficients).** To illustrate the difference between these two similarity measures, we calculate $SMC$ and $J$ for the following two binary vectors.

$\mathbf{x} = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
$\mathbf{y} = (0, 0, 0, 0, 0, 0, 1, 0, 0, 1)$

$f_{01} = 2$ the number of attributes where $\mathbf{x}$ was 0 and $\mathbf{y}$ was 1
$f_{10} = 1$ the number of attributes where $\mathbf{x}$ was 1 and $\mathbf{y}$ was 0
$f_{00} = 7$ the number of attributes where $\mathbf{x}$ was 0 and $\mathbf{y}$ was 0
$f_{11} = 0$ the number of attributes where $\mathbf{x}$ was 1 and $\mathbf{y}$ was 1

$SMC = \frac{f_{11}+f_{00}}{f_{01}+f_{10}+f_{11}+f_{00}} = \frac{0+7}{2+1+0+7} = 0.7$

$J = \frac{f_{11}}{f_{01}+f_{10}+f_{11}} = \frac{0}{2+1+0} = 0$ ■

**The cosine similary is one of the most common measure of document similarity.If x and y are two document vector ,then**

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|}, \tag{2.7}$$

where $\cdot$ indicates the vector dot product, $\mathbf{x} \cdot \mathbf{y} = \sum_{k=1}^{n} x_k y_k$, and $\|\mathbf{x}\|$ is the length of vector $\mathbf{x}$, $\|\mathbf{x}\| = \sqrt{\sum_{k=1}^{n} x_k^2} = \sqrt{\mathbf{x} \cdot \mathbf{x}}$.

**Example 2.18 (Cosine Similarity of Two Document Vectors).** This example calculates the cosine similarity for the following two data objects, which might represent document vectors:

$\mathbf{x} = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$
$\mathbf{y} = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$

$\mathbf{x} \cdot \mathbf{y} = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5$
$\|\mathbf{x}\| = \sqrt{3*3 + 2*2 + 0*0 + 5*5 + 0*0 + 0*0 + 0*0 + 2*2 + 0*0 + 0*0} = 6.48$
$\|\mathbf{y}\| = \sqrt{1*1 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 1*1 + 0*0 + 2*2} = 2.24$
$\cos(\mathbf{x}, \mathbf{y}) = \mathbf{0.31}$

### Extended Jaccard Coefficient (Tanimoto Coefficient)

The extended Jaccard coefficient can be used for document data and that reduces to the Jaccard coefficient in the case of binary attributes. The extended Jaccard coefficient is also known as the Tanimoto coefficient. (However, there is another coefficient that is also known as the Tanimoto coefficient.) This coefficient, which we shall represent as $EJ$, is defined by the following equation:

$$EJ(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \mathbf{x} \cdot \mathbf{y}}. \tag{2.9}$$

### Correlation

The correlation between two data objects that have binary or continuous variables is a measure of the linear relationship between the attributes of the objects. (The calculation of correlation between attributes, which is more common, can be defined similarly.) More precisely, **Pearson's correlation** coefficient between two data objects, $\mathbf{x}$ and $\mathbf{y}$, is defined by the following equation:

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\text{covariance}(\mathbf{x}, \mathbf{y})}{\text{standard\_deviation}(\mathbf{x}) * \text{standard\_deviation}(\mathbf{y})} = \frac{s_{xy}}{s_x\ s_y}, \tag{2.10}$$

where we are using the following standard statistical notation and definitions:

$$\text{covariance}(\mathbf{x}, \mathbf{y}) = s_{xy} = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \overline{x})(y_k - \overline{y}) \tag{2.11}$$

$$\text{standard\_deviation}(\mathbf{x}) = s_x = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(x_k - \overline{x})^2}$$

$$\text{standard\_deviation}(\mathbf{y}) = s_y = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(y_k - \overline{y})^2}$$

$$\overline{x} = \frac{1}{n}\sum_{k=1}^{n} x_k \text{ is the mean of } \mathbf{x}$$

$$\overline{y} = \frac{1}{n}\sum_{k=1}^{n} y_k \text{ is the mean of } \mathbf{y}$$