Contents

Chapter 3

Data Preprocessing

Today's real-world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size (often several gigabytes or more) and their likely origin from multiple, heterogenous sources. Low-quality data will lead to low-quality mining results. *"How can the data be preprocessed in order to help improve the quality of the data and, consequently, of the mining results? How can the data be preprocessed so as to improve the efficiency and ease of the mining process?"*

There are a number of data preprocessing techniques. *Data cleaning* can be applied to remove noise and correct inconsistencies in the data. *Data integration* merges data from multiple sources into a coherent data store, such as a data warehouse. *Data reduction* can reduce the data size by aggregating, eliminating redundant features, or clustering, for instance. *Data transformations*, such as normalization, may be applied, where data are scaled to fall within a smaller range like 0.0 to 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements. These techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a *date* field to a common format. In Chapter 2, we learned about the different attribute types and how to use basic statistical descriptions to study characteristics of the data. These can help identify erroneous values and outliers, which will be useful in the data cleaning and integration steps. Data processing techniques, when applied before mining, can substantially improve the overall quality of the patterns mined and/or the time required for the actual mining.

In this chapter, we introduce the basic concepts of data preprocessing in Section 3.1. The methods for data preprocessing are organized into the following categories: data cleaning (Section 3.2), data integration (Section 3.3), data reduction (Section 3.4), and data transformation (Section 3.5).

3.1 Data Preprocessing: An Overview

This section presents an overview of data preprocessing. Section 3.1.1 illustrates the many elements defining data quality. This provides the incentive behind data preprocessing. Section 3.1.2 outlines the major tasks in data preprocessing.

3.1.1 Data Quality: Why Preprocess the Data?

Data has quality if it satisfies the requirements of its intended use. There are many factors comprising **data quality**. These include: *accuracy, completeness, consistency, timeliness, believability*, and *interpretability*.

Imagine that you are a manager at *AllElectronics* and have been charged with analyzing the company's data with respect to the sales at your branch. You immediately set out to perform this task. You carefully inspect the company's database and data warehouse, identifying and selecting the attributes or dimensions to be included in your analysis, such as *item*, *price*, and *units_sold*. Alas! You notice that several of the attributes for various tuples have no recorded value. For your analysis, you would like to include information as to whether each item purchased was advertised as on sale, yet you discover that this information has not been recorded. Furthermore, users of your database system have reported errors, unusual values, and inconsistencies in the data recorded for some transactions. In other words, the data you wish to analyze by data mining techniques are *incomplete* (lacking attribute values or certain attributes of interest, or containing only aggregate data), *inaccurate* or *noisy* (containing errors, or values that deviate from the expected), and *inconsistent* (e.g., containing discrepancies in the department codes used to categorize items). Welcome to the real world!

This scenario illustrates three of the elements defining data quality - **accuracy**, **completeness**, and **consistency**. Inaccurate, incomplete, and inconsistent data are commonplace properties of large real-world databases and data warehouses. There are many possible reasons for inaccurate data (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Users may purposely submit incorrect data values for mandatory fields when they do not wish to submit personal information, e.g., by choosing the default value 'January 1' displayed for birthday. (This is known as *disguised missing data*.) Errors in data transmission can also occur. There may be technology limitations, such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or data codes used, or inconsistent formats for input fields, such as *date*. Duplicate tuples also require data cleaning.

Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because they were not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding, or because of equipment malfunctions. Data that were

inconsistent with other recorded data may have been deleted. Furthermore, the recording of the history or modifications to the data may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

Recall that data quality depends on the intended use of the data. Two different users may have very different assessments of the quality of a given database. For example, a marketing analyst may need to access the database mentioned above for a list of customer addresses. Some of the addresses are outdated or incorrect, yet overall, 80% of the addresses are accurate. The marketing analyst considers this to be a large customer database for target marketing purposes and is pleased with the accuracy of the database, although, as sales manager, you found the data inaccurate.

Timeliness also affects data quality. Suppose that you are overseeing the distribution of monthly sales bonuses to the top sales representatives at *AllElectronics*. Several sales representatives, however, fail to submit their sales records on time at the end of the month. There are also a number of corrections and adjustments that flow in after the month's end. For a period of time following each month, the data stored in the database is incomplete. However, once all of the data is received, it is correct. The fact that the month-end data is not updated in a timely fashion has a negative impact on the data quality.

Two other factors affecting data quality are believability and interpretability. **Believability** reflects how much the data are trusted by users, while **interpretability** reflects how easy the data are understood. Suppose that a database, at one point, had several errors, all of which have since been corrected. The past errors, however, had caused many problems for users in the sales department, and so they no longer trust the data. The data also use many accounting codes, which the sales department does not know how to interpret. Even though such a database is now accurate, complete, consistent, and timely, users from the sales department may regard it as of low quality due to poor believability and interpretability.

3.1.2 Major Tasks in Data Preprocessing

In this section, we look at the major steps involved in data preprocessing, namely, data cleaning, data integration, data reduction, and data transformation.

Data cleaning routines work to "clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. If users believe the data are dirty, they are unlikely to trust the results of any data mining that has been applied to it. Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output. Although most mining routines have some procedures for dealing with incomplete or noisy data, they are not always robust. Instead, they may concentrate on avoiding overfitting the data to the function being modeled. Therefore, a useful preprocessing step is to run your data through some data cleaning routines. Section 3.2 discusses methods for cleaning up your data.

Getting back to your task at *AllElectronics*, suppose that you would like to include data from multiple sources in your analysis. This would involve integrating multiple databases, data cubes, or files, that is, **data integration**. Yet some attributes representing a given concept may have different names in different databases, causing inconsistencies and redundancies. For example, the attribute for customer identification may be referred to as *customer_id* in one data store and *cust_id* in another. Naming inconsistencies may also occur for attribute values. For example, the same first name could be registered as "Bill" in one database, but "William" in another, and "B." in the third. Furthermore, you suspect that some attributes may be inferred from others (e.g., annual revenue). Having a large amount of redundant data may slow down or confuse the knowledge discovery process. Clearly, in addition to data cleaning, steps must be taken to help avoid redundancies during data integration. Typically, data cleaning and data integration are performed as a preprocessing step when preparing the data for a data warehouse. Additional data cleaning can be performed to detect and remove redundancies that may have resulted from data integration.

"Hmmm," you wonder, as you consider your data even further. *"The data set I have selected for analysis is HUGE, which is sure to slow down the mining process. Is there a way I can reduce the size of my data set without jeopardizing the data mining results?"* **Data reduction** obtains a reduced representation of the data set that is much smaller in volume, yet produces the same (or almost the same) analytical results. Data reduction strategies include *dimensionality reduction* and *numerosity reduction*. In **dimensionality reduction**, data encoding schemes are applied so as to obtain a reduced or "compressed" representation of the original data. Examples include data compression techniques (such as *wavelet transforms* and *principal components analysis*) as well as *attribute subset selection* (e.g., removing irrelevant attributes), and *attribute construction* (e.g., where a small set of more useful attributes is derived from the original set). In **numerosity reduction**, the data are replaced by alternative, smaller representations using parametric models (such as *regression* or *log-linear models*) or nonparametric models (such as with *histograms, clusters, sampling*, or *data aggregation*). Data reduction is the topic of Section 3.4.

Getting back to your data, you have decided, say, that you would like to use a distance-based mining algorithm for your analysis, such as neural networks, nearest-neighbor classifiers, or clustering.[1] Such methods provide better results if the data to be analyzed have been *normalized*, that is, scaled to a smaller range such as [0.0, 1.0]. Your customer data, for example, contain the attributes *age* and *annual salary*. The *annual salary* attribute usually takes much larger values than *age*. Therefore, if the attributes are left unnormalized, the distance measurements taken on *annual salary* will generally outweigh distance measurements taken on *age*. *Discretization* and *concept hierarchy generation* can also be useful, where raw data values for attributes are replaced

---

[1] Neural networks and nearest-neighbor classifiers are described in Chapter 8, and clustering is discussed in Chapters 10 and 11.

Data cleaning

Data integration

Data transformation $-2, 32, 100, 59, 48 \longrightarrow -0.02, 0.32, 1.00, 0.59, 0.48$

Data reduction   attributes   attributes

	A1	A2	A3	...	A126
T1					
T2					
T3					
T4					
...					
T2000					

transactions

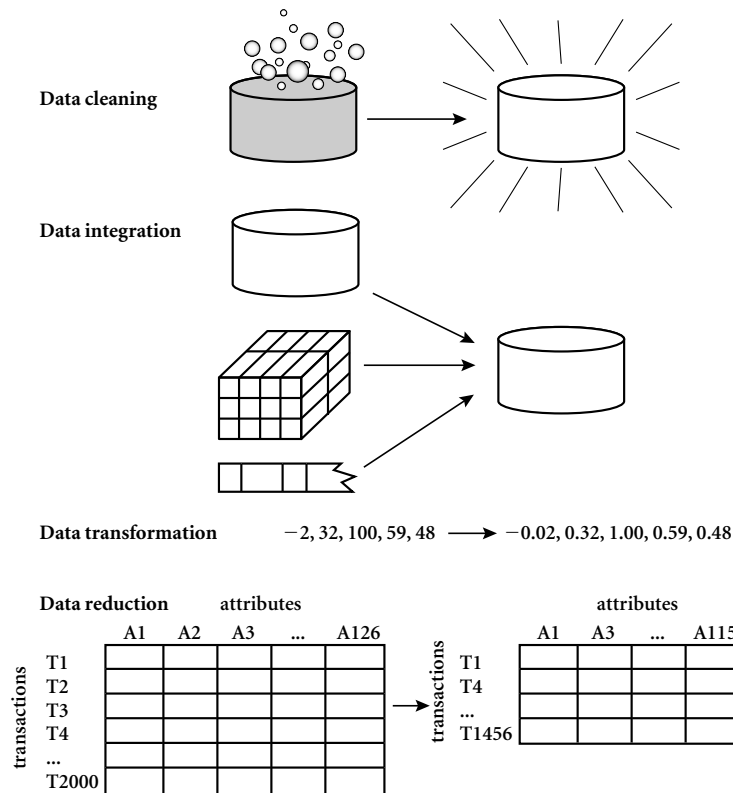|	A1	A3	...	A115
T1				
T4				
...				
T1456				

transactions

Figure 3.1: Forms of data preprocessing. NOTE to editor: Figure needs to be redone so that data reduction comes before data transformation. Thanks.

by ranges or higher conceptual levels. For example, raw values for *age* may be replaced by higher-level concepts, such as *youth*, *adult*, or *senior*. Discretization and concept hierarchy generation are powerful tools for data mining in that they allow the mining of data at multiple levels of abstraction. Normalization, data discretization, and concept hierarchy generation are forms of **data transformation**. You soon realize such data transformation operations are additional data preprocessing procedures that would contribute toward the success of the mining process. Data integration and data discretization are discussed in Sections 3.5.

Figure 3.1 summarizes the data preprocessing steps described here. Note that the above categorization is not mutually exclusive. For example, the removal of redundant data may be seen as a form of data cleaning, as well as data reduction.

In summary, real-world data tend to be dirty, incomplete, and inconsistent. Data preprocessing techniques can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process. Data pre-

processing is an important step in the knowledge discovery process, because quality decisions must be based on quality data. Detecting data anomalies, rectifying them early, and reducing the data to be analyzed can lead to huge payoffs for decision making.

3.2 Data Cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. *Data cleaning* (or *data cleansing*) routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. In this section, you will study basic methods for data cleaning. Section 3.2.1 looks at ways of handling missing values. Section 3.2.2 explains data smoothing techniques. Section 3.2.3 discusses approaches to data cleaning as a process.

3.2.1 Missing Values

Imagine that you need to analyze *AllElectronics* sales and customer data. You note that many tuples have no recorded value for several attributes, such as customer *income*. How can you go about filling in the missing values for this attribute? Let's look at the following methods:

1. **Ignore the tuple**: This is usually done when the class label is missing (assuming the mining task involves classification). This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably. By ignoring the tuple, we do not make use of the remaining attributes values in the tuple. Such data could have been useful to the task at hand.

2. **Fill in the missing value manually**: In general, this approach is time-consuming and may not be feasible given a large data set with many missing values.

3. **Use a global constant to fill in the missing value**: Replace all missing attribute values by the same constant, such as a label like *"Unknown"* or $-\infty$. If missing values are replaced by, say, *"Unknown,"* then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common—that of *"Unknown."* Hence, although this method is simple, it is not foolproof.

4. **Use a measure of central tendency for the attribute (such as the mean or median) to fill in the missing value**: Chapter 2 discussed measures of central tendency, which indicate the "middle" value of a data distribution. For normal (symmetric) data distributions, the mean can be used, while skewed data distribution should employ the median (Section 2.2). For example, suppose that the data distribution regarding

the income of *AllElectronics* customers is symmetric and that the average income is $56,000. Use this value to replace the missing value for *income*.

5. **Use the attribute mean or median for all samples belonging to the same class as the given tuple**: For example, if classifying customers according to *credit_risk*, we may replace the missing value with the average *income* value for customers in the same credit risk category as that of the given tuple. If the data distribution for a given class is skewed, the median value is a better choice.

6. **Use the most probable value to fill in the missing value**: This may be determined with regression, inference-based tools using a Bayesian formalism, or decision tree induction. For example, using the other customer attributes in your data set, you may construct a decision tree to predict the missing values for *income*. Decision trees and Bayesian inference are described in detail in Chapters 8 and 9, respectively, while regression is introduced in Section 3.4.5.

Methods 3 to 6 bias the data. The filled-in value may not be correct. Method 6, however, is a popular strategy. In comparison to the other methods, it uses the most information from the present data to predict missing values. By considering the values of the other attributes in its estimation of the missing value for *income*, there is a greater chance that the relationships between *income* and the other attributes are preserved.

It is important to note that, in some cases, a missing value may not imply an error in the data! For example, when applying for a credit card, candidates may be asked to supply their driver's license number. Candidates who do not have a driver's license may naturally leave this field blank. Forms should allow respondents to specify values such as "not applicable". Software routines may also be used to uncover other null values, such as "don't know", "?", or "none". Ideally, each attribute should have one or more rules regarding the *null* condition. The rules may specify whether or not nulls are allowed, and/or how such values should be handled or transformed. Fields may also be intentionally left blank if they are to be provided in a later step of the business process. Hence, although we can try our best to clean the data after it is seized, good design of databases and of data entry procedures should help minimize the number of missing values or errors in the first place.

3.2.2 Noisy Data

"What is noise?" **Noise** is a random error or variance in a measured variable. In Chapter 2, we saw how some basic statistical description techniques (such as boxplots and scatter plots) and methods of data visualization can be used to identify outliers, which may represent noise. Given a numeric attribute such as, say, *price*, how can we "smooth" out the data to remove the noise? Let's look at the following data smoothing techniques:

1. **Binning:** Binning methods smooth a sorted data value by consulting its "neighborhood," that is, the values around it. The sorted values are dis-

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

Partition into (equal-frequency) bins:

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

Smoothing by bin means:

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

Smoothing by bin boundaries:

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

Figure 3.2: Binning methods for data smoothing.

tributed into a number of "buckets," or *bins*. Because binning methods consult the neighborhood of values, they perform *local* smoothing. Figure 3.2 illustrates some binning techniques. In this example, the data for *price* are first sorted and then partitioned into *equal-frequency* bins of size 3 (i.e., each bin contains three values). In **smoothing by bin means**, each value in a bin is replaced by the mean value of the bin. For example, the mean of the values 4, 8, and 15 in Bin 1 is 9. Therefore, each original value in this bin is replaced by the value 9. Similarly, **smoothing by bin medians** can be employed, in which each bin value is replaced by the bin median. In **smoothing by bin boundaries**, the minimum and maximum values in a given bin are identified as the *bin boundaries*. Each bin value is then replaced by the closest boundary value. In general, the larger the width, the greater the effect of the smoothing. Alternatively, bins may be *equal-width*, where the interval range of values in each bin is constant. Binning is also used as a discretization technique and is further discussed in Section 3.5.

2. **Regression:** Data smoothing can also be done by conforming data values to a function, a technique known as regression. *Linear regression* involves finding the "best" line to fit two attributes (or variables), so that one attribute can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface. Regression is further described in Section 3.4.5.

3. **Outlier analysis**: Outliers may be detected by clustering, for example,
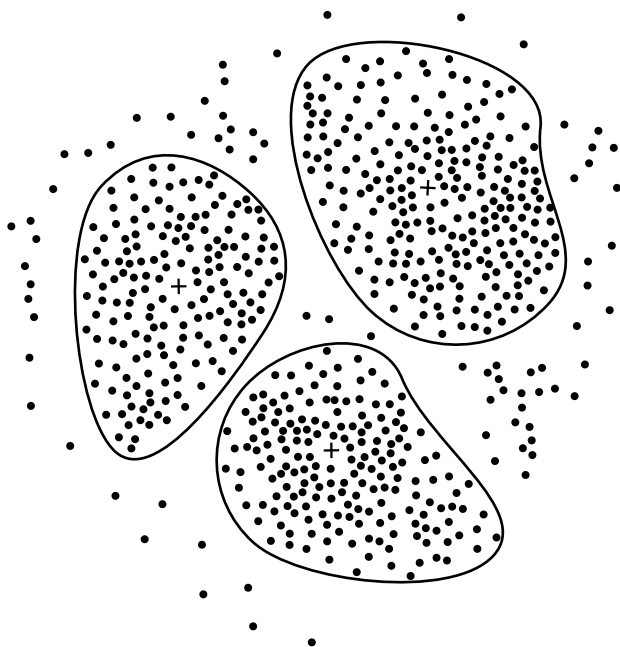
Figure 3.3: A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster centroid is marked with a "+", representing the average point in space for that cluster. Outliers may be detected as values that fall outside of the sets of clusters.

where similar values are organized into groups, or "clusters." Intuitively, values that fall outside of the set of clusters may be considered outliers (Figure 3.3). Chapter 11 is dedicated to the topic of outlier analysis.

Many methods for data smoothing are also methods for data discretization (a form of data transformation) and data reduction. For example, the binning techniques described above reduce the number of distinct values per attribute. This acts as a form of data reduction for logic-based data mining methods, such as decision tree induction, which repeatedly make value comparisons on sorted data. Concept hierarchies are a form of data discretization that can also be used for data smoothing. A concept hierarchy for *price*, for example, may map real *price* values into *inexpensive, moderately_priced*, and *expensive*, thereby reducing the number of data values to be handled by the mining process. Data discretization is discussed in Section 3.5. Some methods of classification, such as neural networks, have built-in data smoothing mechanisms. Classification is the topic of Chapters 8 and 9.

3.2.3   Data Cleaning as a Process

Missing values, noise, and inconsistencies contribute to inaccurate data. So far, we have looked at techniques for handling missing data and for smoothing data. *"But data cleaning is a big job. What about data cleaning as a process? How exactly does one proceed in tackling this task? Are there any tools out there to help?"*

The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and the inconsistent use of codes. Errors in instrumentation devices that record data, and system errors, are another source of discrepancies. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).[2]

"So, how can we proceed with discrepancy detection?" As a starting point, use any knowledge you may already have regarding properties of the data. Such knowledge or "data about data" is referred to as **metadata**. This is where we can make use of the knowledge we gained about our data in Chapter 2. For example, what are the data type and domain of each attribute? What are the acceptable values for each attribute? The basic statistical data descriptions discussed in Section 2.2 are useful here to grasp data trends and identify anomalies. For example, find the mean, median, and mode values. Are the data symmetric or skewed? What is the range of values? Do all values fall within the expected range? What is the standard deviation of each attribute? Values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. Are there any known dependencies between attributes? In this step, you may write your own scripts and/or use some of the tools that we discuss further below. From this, you may find noise, outliers, and unusual values that need investigation.

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

The data should also be examined regarding unique rules, consecutive rules, and null rules. A **unique rule** says that each value of the given attribute must be different from all other values for that attribute. A **consecutive rule** says that there can be no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers). A **null rule** specifies the use of blanks, question marks, special characters, or

---

[2]Data integration and the removal of redundant data that can result from such integration are further described in Section 3.3.

other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled. As mentioned in Section 3.2.1, reasons for missing values may include (1) the person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a *license-number* attribute left blank by nondrivers); (2) the data entry person does not know the correct value; or (3) the value is to be provided by a later step of the process. The null rule should specify how to record the null condition, for example, such as to store zero for numeric attributes, a blank for character attributes, or any other conventions that may be in use (such as that entries like "don't know" or "?" should be transformed to blank).

There are a number of different commercial tools that can aid in the step of discrepancy detection. **Data scrubbing tools** use simple domain knowledge (e.g., knowledge of postal addresses, and spell-checking) to detect errors and make corrections in the data. These tools rely on parsing and fuzzy matching techniques when cleaning data from multiple sources. **Data auditing tools** find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions. They are variants of data mining tools. For example, they may employ statistical analysis to find correlations, or clustering to identify outliers. They may also use the basic statistical data descriptions presented in Section 2.2.

Some data inconsistencies may be corrected manually using external references. For example, errors made at data entry may be corrected by performing a paper trace. Most errors, however, will require *data transformations*. That is, once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them.

Commercial tools can assist in the data transformation step. **Data migration tools** allow simple transformations to be specified, such as to replace the string *"gender"* by *"sex"*. **ETL (extraction/transformation/loading) tools** allow users to specify transforms through a graphical user interface (GUI). These tools typically support only a restricted set of transforms so that, often, we may also choose to write custom scripts for this step of the data cleaning process.

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

New approaches to data cleaning emphasize increased interactivity. Potter's Wheel, for example, is a publicly available data cleaning tool that integrates discrepancy detection and transformation. Users gradually build a series of transformations by composing and debugging individual transformations, one step at a time, on a spreadsheet-like interface. The transformations can be specified graphically or by providing examples. Results are shown immediately on the records that are visible on the screen. The user can choose to undo the transformations, so that transformations that introduced additional errors can be "erased." The tool performs discrepancy checking automatically in the background on the latest transformed view of the data. Users can gradually develop and refine transformations as discrepancies are found, leading to more effective and efficient data cleaning.

Another approach to increased interactivity in data cleaning is the development of declarative languages for the specification of data transformation operators. Such work focuses on defining powerful extensions to SQL and algorithms that enable users to express data cleaning specifications efficiently.

As we discover more about the data, it is important to keep updating the metadata to reflect this knowledge. This will help speed up data cleaning on future versions of the same data store.

3.3   Data Integration

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

The semantic heterogeneity and structure of data pose great challenges in data integration. How can we match schema and objects from different sources? This is the essence of the *entity identification problem*, described in Section 3.3.1. Are any attributes correlated? Section 3.3.2 presents correlation tests for numeric and nominal data. Tuple duplication is described in Section 3.3.3. Finally, Section 3.3.4 touches on the detection and resolution of data value conflicts.

3.3.1   The Entity Identification Problem

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

There are a number of issues to consider during data integration. *Schema integration* and *object matching* can be tricky. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the **entity identification problem**. For example, how can the data analyst or the computer be sure that *customer_id* in one database and *cust_number* in another refer to the same attribute? Examples of metadata for each attribute

include the name, meaning, data type, and range of values permitted for the attribute, and null rules for handling blank, zero, or null values (Section 3.2). Such metadata can be used to help avoid errors in schema integration. The metadata may also be used to help transform the data (e.g., where data codes for *pay_type* in one database may be *"H"* and *"S"*, and *1* and *2* in another). Hence, this step also relates to data cleaning, as described earlier.

When matching attributes from one database to another during integration, special attention must be paid to the *structure* of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. For example, in one system, a *discount* may be applied to the order, whereas in another system it is applied to each individual line item within the order. If this is not caught before integration, items in the target system may be improperly discounted.

3.3.2 Redundancy and Correlation Analysis

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

Some redundancies can be detected by **correlation analysis**. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For nominal data, we use the χ^2 (*chi-square*) test. For numeric attributes, we can use the *correlation coefficient* and *covariance*, both of which access how one attribute's values vary with those of another.

χ^2 Correlation Test for Nominal Data

For nominal data, a correlation relationship between two attributes, A and B, can be discovered by a χ^2 (**chi-square**) test. Suppose A has c distinct values, namely $a_1, a_2, \ldots a_c$. B has r distinct values, namely $b_1, b_2, \ldots b_r$. The data tuples described by A and B can be shown as a **contingency table**, with the c values of A making up the columns and the r values of B making up the rows. Let (A_i, B_j) denote the joint event that attribute A takes on value a_i and attribute B takes on value b_j, that is, where $(A = a_i, B = b_j)$. Each and every possible (A_i, B_j) joint event has its own cell (or slot) in the table. The χ^2 value (also known as the *Pearson χ^2 statistic*) is computed as:

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{3.1}$$

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

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{n}, \tag{3.2}$$

Table 3.1: A 2×2 contingency table for the data of Example 2.1. Are *gender* and *preferred_Reading* correlated?

	male	*female*	Total
fiction	250 (90)	200 (360)	450
non_fiction	50 (210)	1000 (840)	1050
Total	300	1200	1500

where n is the number of data tuples, $count(A = a_i)$ is the number of tuples having value a_i for A, and $count(B = b_j)$ is the number of tuples having value b_j for B. The sum in Equation (3.1) is computed over all of the $r \times c$ cells. Note that the cells that contribute the most to the χ^2 value are those whose actual count is very different from that expected.

The χ^2 statistic tests the hypothesis that A and B are *independent*, that is, there is no correlation between them. The test is based on a significance level, with $(r - 1) \times (c - 1)$ degrees of freedom. We will illustrate the use of this statistic in an example below. If the hypothesis can be rejected, then we say that A and B are statistically correlated.

Let's look at a concrete example.

Example 3.1 **Correlation analysis of nominal attributes using χ^2.** Suppose that a group of 1,500 people was surveyed. The gender of each person was noted. Each person was polled as to whether their preferred type of reading material was fiction or nonfiction. Thus, we have two attributes, *gender* and *preferred_reading*. The observed frequency (or count) of each possible joint event is summarized in the contingency table shown in Table 3.1, where the numbers in parentheses are the expected frequencies. The expected frequencies are calculated based on the data distribution for both attributes using Equation (3.2).

Using Equation (3.2), we can verify the expected frequencies for each cell. For example, the expected frequency for the cell (male, fiction) is

$$e_{11} = \frac{count(male) \times count(fiction)}{n} = \frac{300 \times 450}{1500} = 90,$$

and so on. Notice that in any row, the sum of the expected frequencies must equal the total observed frequency for that row, and the sum of the expected frequencies in any column must also equal the total observed frequency for that column. Using Equation (3.1) for χ^2 computation, we get

$$\begin{aligned}
\chi^2 &= \frac{(250 - 90)^2}{90} + \frac{(50 - 210)^2}{210} + \frac{(200 - 360)^2}{360} + \frac{(1000 - 840)^2}{840} \\
&= 284.44 + 121.90 + 71.11 + 30.48 = 507.93.
\end{aligned}$$

For this 2×2 table, the degrees of freedom are $(2 - 1)(2 - 1) = 1$. For 1 degree of freedom, the χ^2 value needed to reject the hypothesis at the 0.001 significance level is 10.828 (taken from the table of upper percentage points of

the χ^2 distribution, typically available from any textbook on statistics). Since our computed value is above this, we can reject the hypothesis that *gender* and *preferred_reading* are independent and conclude that the two attributes are (strongly) correlated for the given group of people.

Correlation Coefficient for Numeric Data

For numeric attributes, we can evaluate the correlation between two attributes, A and B, by computing the **correlation coefficient** (also known as **Pearson's product moment coefficient**, named after its inventer, Karl Pearson). This is

$$r_{A,B} = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{n\sigma_A\sigma_B} = \frac{\sum_{i=1}^{n}(a_ib_i) - n\bar{A}\bar{B}}{n\sigma_A\sigma_B}, \tag{3.3}$$

where n is the number of tuples, a_i and b_i are the respective values of A and B in tuple i, $\bar{A}$ and $\bar{B}$ are the respective mean values of A and B, σ_A and σ_B are the respective standard deviations of A and B (as defined in Section 2.2.8), and $\Sigma(a_ib_i)$ is the sum of the AB cross-product (that is, for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \leq r_{A,B} \leq +1$. If $r_{A,B}$ is greater than 0, then A and B are *positively correlated*, meaning that the values of A increase as the values of B increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that A (or B) may be removed as a redundancy. If the resulting value is equal to 0, then A and B are *independent* and there is no correlation between them. If the resulting value is less than 0, then A and B are *negatively correlated*, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes (Section 2.2.12). For example, the scatter plots of Figure 2.9 respectively show positively correlated data and negatively correlated data, while Figure 2.10 displays uncorrelated data.

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

Covariance of Numeric Data

In probability theory and statistics, correlation and covariance are two similar measures for assessing how much two attributes change together. Consider two numeric attributes A and B, and a set of n observations $\{(a_1, b_1), \ldots, (a_n, b_n)\}$. The mean values of A and B, respectively, are also known as the **expected**

Time point	*AllElectronics*	*HighTech*
t1	6	20
t2	5	10
t3	4	14
t4	3	5
t5	2	5

Table 3.2: Stock prices for *AllElectronics* and *HighTech*.

values on A and B, that is,

$$E(A) = \bar{A} = \frac{\sum_{i=1}^{n} a_i}{n}$$

and

$$E(B) = \bar{B} = \frac{\sum_{i=1}^{n} b_i}{n}.$$

The **covariance** between A and B is defined as

$$Cov(A, B) = E((A - \bar{A})(B - \bar{B})) = \frac{\sum_{i=1}^{n} (a_i - \bar{A})(b_i - \bar{B})}{n}. \qquad (3.4)$$

If we compare Equation 3.3 for $r_{A,B}$ (correlation coefficient) with Equation 3.4 for covariance, we see that

$$r_{A,B} = \frac{Cov(A, B)}{\sigma_A \sigma_B} \qquad (3.5)$$

where σ_A and σ_B are the standard deviations of A and B, respectively. It can also be shown that

$$Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B}. \qquad (3.6)$$

This equation may simplify calculations.

For two attributes A and B that tend to change together, if A is larger than $\bar{A}$ (the expected value of A), then B is likely to be larger than $\bar{B}$ (the expected value of B). Therefore, the covariance between A and B is *positive*. On the other hand, if one of the attributes tends to be above its expected value when the other attribute is below its expected value, then the covariance of A and B is *negative*.

If A and B are *independent*, that is, they do not have correlation, then $E(A \cdot B) = E(A) \cdot E(B)$. Therefore, the covariance is $Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B} = E(A) \cdot E(B) - \bar{A}\bar{B} = 0$. However, the converse is not true. Some pairs of random variables (attributes) may have a covariance of 0 but are not independent. Only under some additional assumptions (such as that the data follow multivariate normal distributions) does a covariance of 0 imply independence.

Example 3.2 **Covariance analysis of numeric attributes.** Consider Table 3.2, which presents a simplified example of stock prices observed at five time points for

AllElectronics and *HighTech*, some high tech company. If the stocks are affected by the same industry trends, will their prices rise or fall together?

$$E(AllElectronics) = \frac{6+5+4+3+2}{5} = \frac{20}{5} = \$4$$

and

$$E(HighTech) = \frac{20+10+14+5+5}{5} = \frac{54}{5} = \$10.8.$$

Thus, using Equation 3.4, we compute

$$Cov(AllElectroncis, HighTech) = \frac{6\times20+5\times10+4\times14+3\times5+2\times5}{5} - 4\times10.8$$
$$= 50.2 - 43.2 = 7.$$

Therefore, given the positive covariance we can say that stock prices for both companies rise together.

Variance is a special case of covariance, where the two attributes are identical (that is, the covariance of an attribute with itself). Variance was discussed in Chapter 2.

3.3.3 Tuple Duplication

In addition to detecting redundancies between attributes, duplication should also be detected at the tuple level (e.g., where there are two or more identical tuples for a given unique data entry case). The use of denormalized tables (often done to improve performance by avoiding joins) is another source of data redundancy. Inconsistencies often arise between various duplicates, due to inaccurate data entry or updating some but not all of the occurrences of the data. For example, if a purchase order database contains attributes for the purchaser's name and address instead of a key to this information in a purchaser database, discrepancies can occur, such as the same purchaser's name appearing with different addresses within the purchase order database.

3.3.4 Detection and Resolution of Data Value Conflicts

Data integration also involves the *detection and resolution of data value conflicts*. For example, for the same real-world entity, attribute values from different sources may differ. This may be due to differences in representation, scaling, or encoding. For instance, a *weight* attribute may be stored in metric units in one system and British imperial units in another. For a hotel chain, the *price* of rooms in different cities may involve not only different currencies but also different services (such as free breakfast) and taxes. When exchanging information between schools, each school may have its own curriculum and grading scheme. One university may adopt a quarter system, offer three courses on database systems, and assign grades from A+ to F, whereas another may adopt a semester system, offer two courses on databases, and assign grades from

1 to 10. It is difficult to work out precise course-to-grade transformation rules between the two universities, making information exchange difficult.

Attributes may also differ on the level of abstraction, where an attribute in one system is recorded at, say, a lower level of abstraction than the "same" attribute in another. For example, the *total_sales* in one database may refer to one branch of *All_Electronics*, while an attribute of the same name in another database may refer to the total sales for *All_Electronics* stores in a given region. The topic of discrepancy detection is further described in Section 3.2.3 on data cleaning as a process.

3.4   Data Reduction

Imagine that you have selected data from the *AllElectronics* data warehouse for analysis. The data set will likely be huge! Complex data analysis and mining on huge amounts of data can take a long time, making such analysis impractical or infeasible.

Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results. In this section, we first present an overview of data reduction strategies, followed by a closer look at individual techniques.

3.4.1   Overview of Data Reduction Strategies

Data reduction strategies include *dimensionality reduction*, *numerosity reduction*, and *data compression*.

Dimensionality reduction is the process of reducing the number of random variables or attributes under consideration. Dimensionality reduction methods include *wavelet transforms* (Section 3.4.2) and *principal components analysis* (Section 3.4.3), which transform or project the original data onto a smaller space. *Attribute subset selection* is a method of dimensionality reduction in which irrelevant, weakly relevant, or redundant attributes or dimensions are detected and removed (Section 3.4.4).

Techniques of **numerosity reduction** replace the original data volume by alternative, smaller forms of data representation. These techniques may be parametric or nonparametric. For *parametric methods*, a model is used to estimate the data, so that typically only the data parameters need to be stored, instead of the actual data. (Outliers may also be stored.) Regression and log-linear models (Section 3.4.5) are examples. *Nonparametric methods* for storing reduced representations of the data include *histograms* (Section 3.4.6), *clustering* (Section 3.4.7), *sampling* (Section 3.4.8), and *data cube aggregation* (Section 3.4.9).

In **data compression**, transformations are applied so as to obtain a reduced or "compressed" representation of the original data. If the original data can be *reconstructed* from the compressed data without any loss of information,

the data reduction is called **lossless**. If, instead, we can reconstruct only an approximation of the original data, then the data reduction is called **lossy**. There are several lossless algorithms for string compression, however, they typically allow only limited manipulation of the data. Dimensionality reduction and numerosity reduction techniques can also be considered forms of data compression.

There are many other ways of organizing methods of data reduction. The computational time spent on data reduction should not outweigh or "erase" the time saved by mining on a reduced data set size.

3.4.2 Wavelet Transforms

The **discrete wavelet transform (DWT)** is a linear signal processing technique that, when applied to a data vector X, transforms it to a numerically different vector, X', of **wavelet coefficients**. The two vectors are of the same length. When applying this technique to data reduction, we consider each tuple as an n-dimensional data vector, that is, $X = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n database attributes.[3]

"How can this technique be useful for data reduction if the wavelet transformed data are of the same length as the original data?" The usefulness lies in the fact that the wavelet transformed data can be truncated. A compressed approximation of the data can be retained by storing only a small fraction of the strongest of the wavelet coefficients. For example, all wavelet coefficients larger than some user-specified threshold can be retained. All other coefficients are set to 0. The resulting data representation is therefore very sparse, so that operations that can take advantage of data sparsity are computationally very fast if performed in wavelet space. The technique also works to remove noise without smoothing out the main features of the data, making it effective for data cleaning as well. Given a set of coefficients, an approximation of the original data can be constructed by applying the *inverse* of the DWT used.

The DWT is closely related to the *discrete Fourier transform (DFT)*, a signal processing technique involving sines and cosines. In general, however, the DWT achieves better lossy compression. That is, if the same number of coefficients is retained for a DWT and a DFT of a given data vector, the DWT version will provide a more accurate approximation of the original data. Hence, for an equivalent approximation, the DWT requires less space than the DFT. Unlike the DFT, wavelets are quite localized in space, contributing to the conservation of local detail.

There is only one DFT, yet there are several families of DWTs. Figure 3.4 shows some wavelet families. Popular wavelet transforms include the Haar-2, Daubechies-4, and Daubechies-6 transforms. The general procedure for applying a discrete wavelet transform uses a hierarchical *pyramid algorithm* that halves the data at each iteration, resulting in fast computational speed. The method is as follows:

---

[3]In our notation, any variable representing a vector is shown in bold italic font; measurements depicting the vector are shown in italic font.

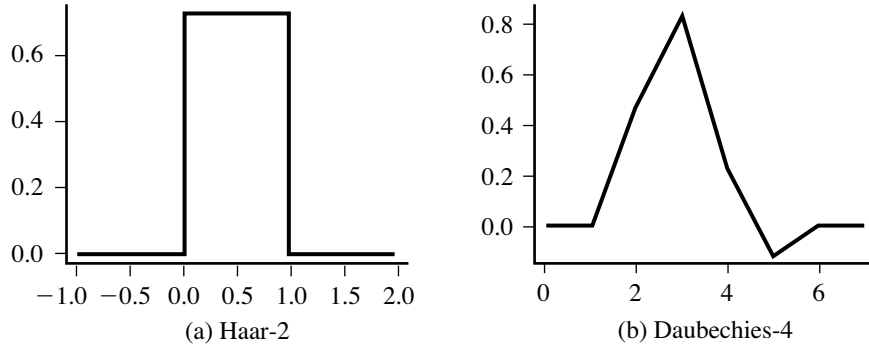(a) Haar-2   (b) Daubechies-4

Figure 3.4: Examples of wavelet families. The number next to a wavelet name is the number of *vanishing moments* of the wavelet. This is a set of mathematical relationships that the coefficients must satisfy and is related to the number of coefficients.

1. The length, L, of the input data vector must be an integer power of 2. This condition can be met by padding the data vector with zeros as necessary ($L \geq n$).

2. Each transform involves applying two functions. The first applies some data smoothing, such as a sum or weighted average. The second performs a weighted difference, which acts to bring out the detailed features of the data.

3. The two functions are applied to pairs of data points in X, that is, to all pairs of measurements (x_{2i}, x_{2i+1}). This results in two sets of data of length $L/2$. In general, these represent a smoothed or low-frequency version of the input data and the high-frequency content of it, respectively.

4. The two functions are recursively applied to the sets of data obtained in the previous loop, until the resulting data sets obtained are of length 2.

5. Selected values from the data sets obtained in the above iterations are designated the wavelet coefficients of the transformed data.

Equivalently, a matrix multiplication can be applied to the input data in order to obtain the wavelet coefficients, where the matrix used depends on the given DWT. The matrix must be **orthonormal**, meaning that the columns are unit vectors and are mutually orthogonal, so that the matrix inverse is just its transpose. Although we do not have room to discuss it here, this property allows the reconstruction of the data from the smooth and smooth-difference data sets. By factoring the matrix used into a product of a few sparse matrices, the resulting *"fast DWT"* algorithm has a complexity of $O(n)$ for an input vector of length n.

Wavelet transforms can be applied to multidimensional data, such as a data cube. This is done by first applying the transform to the first dimension, then

to the second, and so on. The computational complexity involved is linear with respect to the number of cells in the cube. Wavelet transforms give good results on sparse or skewed data and on data with ordered attributes. Lossy compression by wavelets is reportedly better than JPEG compression, the current commercial standard. Wavelet transforms have many real-world applications, including the compression of fingerprint images, computer vision, analysis of time-series data, and data cleaning.

3.4.3 Principal Components Analysis

In this subsection we provide an intuitive introduction to principal components analysis as a method of dimesionality reduction. A detailed theoretical explanation is beyond the scope of this book. For additional references, please see the bibliographic notes at the end of this chapter.

Suppose that the data to be reduced consist of tuples or data vectors described by n attributes or dimensions. **Principal components analysis**, or **PCA** (also called the Karhunen-Loeve, or K-L, method), searches for k n-dimensional orthogonal vectors that can best be used to represent the data, where $k \leq n$. The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. Unlike attribute subset selection (Section 3.4.4), which reduces the attribute set size by retaining a subset of the initial set of attributes, PCA "combines" the essence of attributes by creating an alternative, smaller set of variables. The initial data can then be projected onto this smaller set. PCA often reveals relationships that were not previously suspected and thereby allows interpretations that would not ordinarily result.

The basic procedure is as follows:

1. The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.

2. PCA computes k orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that each point in a direction perpendicular to the others. These vectors are referred to as the *principal components*. The input data are a linear combination of the principal components.

3. The principal components are sorted in order of decreasing "significance" or strength. The principal components essentially serve as a new set of axes for the data, providing important information about variance. That is, the sorted axes are such that the first axis shows the most variance among the data, the second axis shows the next highest variance, and so on. For example, Figure 3.5 shows the first two principal components, Y_1 and Y_2, for the given set of data originally mapped to the axes X_1 and X_2. This information helps identify groups or patterns within the data.

4. Because the components are sorted according to the decreasing order of "significance," the size of the data can be reduced by eliminating the
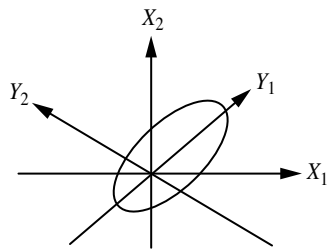
Figure 3.5: Principal components analysis. $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ are the first two principal components for the given data. NOTE: Figure needs to be corrected so that $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ are orthogonal.

weaker components, that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data.

PCA can be applied to ordered and unordered attributes, and can handle sparse data and skewed data. Multidimensional data of more than two dimensions can be handled by reducing the problem to two dimensions. Principal components may be used as inputs to multiple regression and cluster analysis. In comparison with wavelet transforms, PCA tends to be better at handling sparse data, whereas wavelet transforms are more suitable for data of high dimensionality.

3.4.4   Attribute Subset Selection

Data sets for analysis may contain hundreds of attributes, many of which may be irrelevant to the mining task or redundant. For example, If the task is to classify customers based on a probability of positive reaction on a discount offer of a music CD, attributes such as the customer's telephone number are likely to be irrelevant, unlike attributes such as *age* or *music_taste*. Although it may be possible for a domain expert to pick out some of the useful attributes, this can be a difficult and time-consuming task, especially when the behavior of the data is not well known (hence, a reason behind its analysis!). Leaving out relevant attributes or keeping irrelevant attributes may be detrimental, causing confusion for the mining algorithm employed. This can result in discovered patterns of poor quality. In addition, the added volume of irrelevant or redundant attributes can slow down the mining process.

Attribute subset selection[4] reduces the data set size by removing irrelevant or redundant attributes (or dimensions). The goal of attribute subset selection is to find a minimum set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes. Mining on a reduced set of attributes

---

[4]In machine learning, attribute subset selection is known as *feature subset selection*.
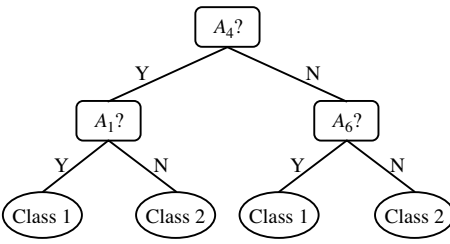
Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: {} => $\{A_1\}$ => $\{A_1, A_4\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$

Figure 3.6: Greedy (heuristic) methods for attribute subset selection.

has an additional benefit. It reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand.

"How can we find a 'good' subset of the original attributes?" For n attributes, there are 2^n possible subsets. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as n and the number of data classes increase. Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection. These methods are typically **greedy** in that, while searching through attribute space, they always make what looks to be the best choice at the time. Their strategy is to make a locally optimal choice in the hope that this will lead to a globally optimal solution. Such greedy methods are effective in practice and may come close to estimating an optimal solution.

The "best" (and "worst") attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another. Many other attribute evaluation measures can be used, such as the *information gain* measure used in building decision trees for classification.[5]

Basic heuristic methods of attribute subset selection include the following techniques, some of which are illustrated in Figure 3.6.

1. **Stepwise forward selection**: The procedure starts with an empty set of attributes as the reduced set. The best of the original attributes is determined and added to the reduced set. At each subsequent iteration or step, the best of the remaining original attributes is added to the set.

2. **Stepwise backward elimination**: The procedure starts with the full set of attributes. At each step, it removes the worst attribute remaining in the set.

---

[5]The information gain measure is described in detail in Chapter 8.

3. **Combination of forward selection and backward elimination**: The stepwise forward selection and backward elimination methods can be combined so that, at each step, the procedure selects the best attribute and removes the worst from among the remaining attributes.

4. **Decision tree induction**: Decision tree algorithms, such as ID3, C4.5, and CART, were originally intended for classification. Decision tree induction constructs a flowchart-like structure where each internal (nonleaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf) node denotes a class prediction. At each node, the algorithm chooses the "best" attribute to partition the data into individual classes.

   When decision tree induction is used for attribute subset selection, a tree is constructed from the given data. All attributes that do not appear in the tree are assumed to be irrelevant. The set of attributes appearing in the tree form the reduced subset of attributes.

The stopping criteria for the methods may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

In some cases, we may want to create new attributes based on others. Such **attribute construction**[6] can help improve accuracy and understanding of structure in high-dimensional data. For example, we may wish to add the attribute *area* based on the attributes *height* and *width*. By combining attributes, attribute construction can discover missing information about the relationships between data attributes that can be useful for knowledge discovery.

3.4.5   Regression and Log-Linear Models: Parametric Data Reduction

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

$$y = wx + b, \tag{3.7}$$

where the variance of y is assumed to be constant. In the context of data mining, x and y are numeric database attributes. The coefficients, w and b (called *regression coefficients*), specify the slope of the line and the Y-intercept, respectively. These coefficients can be solved for by the *method of least squares*, which minimizes the error between the actual line separating the data and the estimate of the line. **Multiple linear regression** is an extension of (simple) linear regression, which allows a response variable, y, to be modeled as a linear function of two or more predictor variables.

---

[6]In the machine learning literature, attribute construction is known as *feature construction*.

Log-linear models approximate discrete multidimensional probability distributions. Given a set of tuples in n dimensions (e.g., described by n attributes), we can consider each tuple as a point in an n-dimensional space. Log-linear models can be used to estimate the probability of each point in a multidimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations. This allows a higher-dimensional data space to be constructed from lower-dimensional spaces. Log-linear models are therefore also useful for dimensionality reduction (since the lower-dimensional points together typically occupy less space than the original data points) and data smoothing (since aggregate estimates in the lower-dimensional space are less subject to sampling variations than the estimates in the higher-dimensional space).

Regression and log-linear models can both be used on sparse data, although their application may be limited. While both methods can handle skewed data, regression does exceptionally well. Regression can be computationally intensive when applied to high-dimensional data, whereas log-linear models show good scalability for up to 10 or so dimensions.

Several software packages exist to solve regression problems. Examples include SAS (*www.sas.com*), SPSS (*www.spss.com*), and S-Plus (*www.insightful.com*). Another useful resource is the book *Numerical Recipes in C,* by Press, Flannery, Teukolsky, and Vetterling, and its associated source code.

3.4.6 Histograms

Histograms use binning to approximate data distributions and are a popular form of data reduction. Histograms were introduced in Section 2.2.9. A **histogram** for an attribute, A, partitions the data distribution of A into disjoint subsets, or *buckets*. If each bucket represents only a single attribute-value/frequency pair, the buckets are called *singleton buckets*. Often, buckets instead represent continuous ranges for the given attribute.

Example 3.3 **Histograms.** The following data are a list of prices of commonly sold items at *AllElectronics* (rounded to the nearest dollar). The numbers have been sorted: 1, 1, 5, 5, 5, 5, 5, 8, 8, 10, 10, 10, 10, 12, 14, 14, 14, 15, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.

Figure 3.7 shows a histogram for the data using singleton buckets. To further reduce the data, it is common to have each bucket denote a continuous range of values for the given attribute. In Figure 3.8, each bucket represents a different $10 range for *price*.

"How are the buckets determined and the attribute values partitioned?" There are several partitioning rules, including the following:

- **Equal-width**: In an equal-width histogram, the width of each bucket range is uniform (such as the width of $10 for the buckets in Figure 3.8).
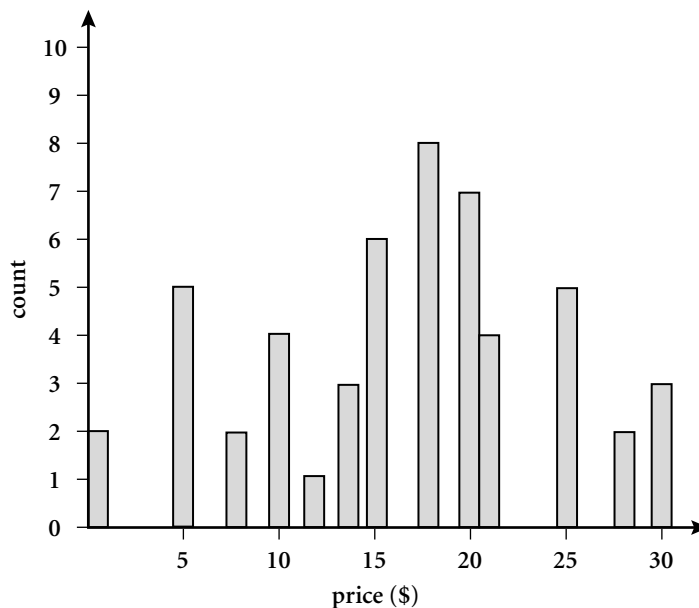
Figure 3.7: A histogram for *price* using singleton buckets—each bucket represents one price-value/ frequency pair.

- **Equal-frequency** (or equidepth): In an equal-frequency histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (that is, each bucket contains roughly the same number of contiguous data samples).

Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data. The histograms described above for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. Such histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for high dimensionalities.

Singleton buckets are useful for storing outliers with high frequency.

3.4.7   Clustering

Clustering techniques consider data tuples as objects. They partition the objects into groups or *clusters*, so that objects within a cluster are "similar" to one another and "dissimilar" to objects in other clusters. Similarity is commonly defined in terms of how "close" the objects are in space, based on a distance function. The "quality" of a cluster may be represented by its *diameter*, the maximum distance between any two objects in the cluster. *Centroid distance* is an alternative measure of cluster quality and is defined as the average distance
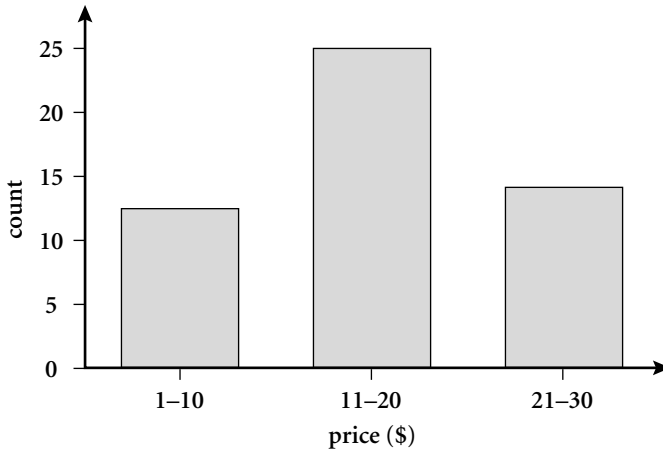
Figure 3.8: An equal-width histogram for *price*, where values are aggregated so that each bucket has a uniform width of $10.

of each cluster object from the cluster centroid (denoting the "average object," or average point in space for the cluster). Figure 3.3 of Section 3.2.2 shows a 2-D plot of customer data with respect to customer locations in a city, where the centroid of each cluster is shown with a "+". Three data clusters are visible.

In data reduction, the cluster representations of the data are used to replace the actual data. The effectiveness of this technique depends on the nature of the data. It is much more effective for data that can be organized into distinct clusters than for smeared data.

There are many measures for defining clusters and cluster quality. Clustering methods are further described in Chapters 10 and 11.

3.4.8   Sampling

Sampling can be used as a data reduction technique because it allows a large data set to be represented by a much smaller random sample (or subset) of the data. Suppose that a large data set, D, contains N tuples. Let's look at the most common ways that we could sample D for data reduction, as illustrated in Figure 3.9.

- **Simple random sample without replacement (SRSWOR) of size** s: This is created by drawing s of the N tuples from D ($s < N$), where the probability of drawing any tuple in D is $1/N$, that is, all tuples are equally likely to be sampled.

- **Simple random sample with replacement (SRSWR) of size** s: This is similar to SRSWOR, except that each time a tuple is drawn from D, it is recorded and then *replaced*. That is, after a tuple is drawn, it is placed back in D so that it may be drawn again.
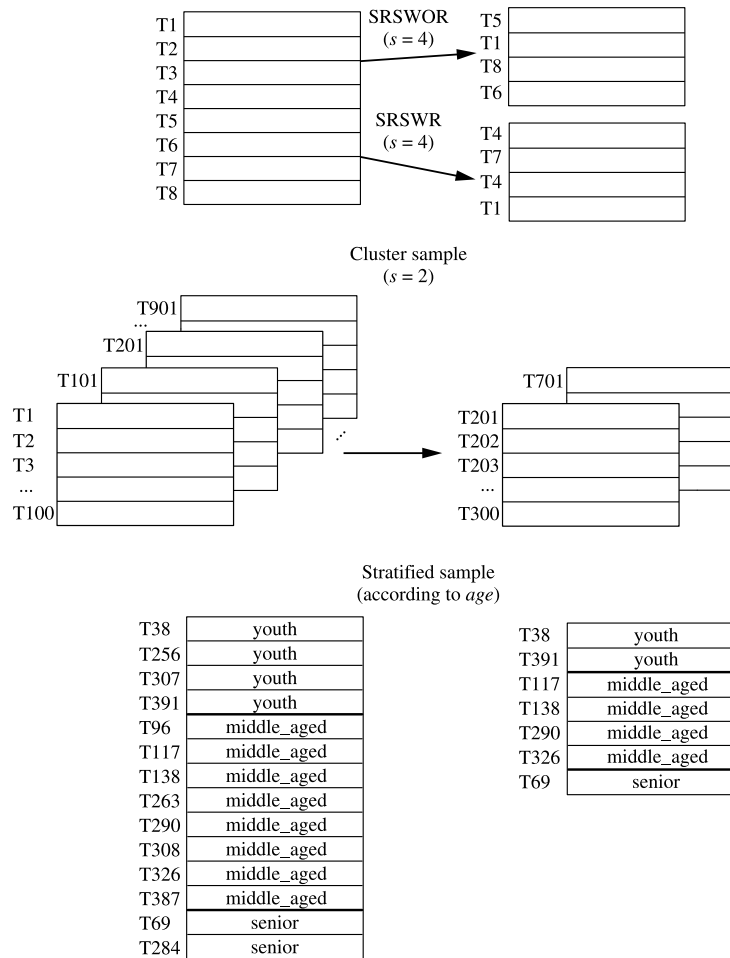
Figure 3.9: Sampling can be used for data reduction.

- **Cluster sample**: If the tuples in D are grouped into M mutually disjoint "clusters," then an SRS of s clusters can be obtained, where $s < M$. For example, tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster. A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples. Other clustering criteria conveying rich semantics can also be explored. For example, in a spatial database, we may choose to define clusters geographically based on how closely different areas are located.

- **Stratified sample**: If D is divided into mutually disjoint parts called *strata,* a stratified sample of D is generated by obtaining an SRS at each

> stratum. This helps ensure a representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where a stratum is created for each customer age group. In this way, the age group having the smallest number of customers will be sure to be represented.

An advantage of sampling for data reduction is that the cost of obtaining a sample *is proportional to the size of the sample*, s, as opposed to N, the data set size. Hence, sampling complexity is potentially *sublinear* to the size of the data. Other data reduction techniques can require at least one complete pass through D. For a fixed sample size, sampling complexity increases only linearly as the number of data dimensions, n, increases, whereas techniques using histograms, for example, increase exponentially in n.

When applied to data reduction, sampling is most commonly used to estimate the answer to an aggregate query. It is possible (using the central limit theorem) to determine a sufficient sample size for estimating a given function within a specified degree of error. This sample size, s, may be extremely small in comparison to N. Sampling is a natural choice for the progressive refinement of a reduced data set. Such a set can be further refined by simply increasing the sample size.

3.4.9 Data Cube Aggregation

Imagine that you have collected the data for your analysis. These data consist of the *AllElectronics* sales per quarter, for the years 2008 to 2010. You are, however, interested in the annual sales (total per year), rather than the total per quarter. Thus the data can be *aggregated* so that the resulting data summarize the total sales per year instead of per quarter. This aggregation is illustrated in Figure 3.10. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

Data cubes are discussed in detail in Chapter 4 on data warehousing and Chapter 5 on advanced data cube technology. We briefly introduce some concepts here. Data cubes store multidimensional aggregated information. For example, Figure 3.11 shows a data cube for multidimensional analysis of sales data with respect to annual sales per item type for each *AllElectronics* branch. Each cell holds an aggregate data value, corresponding to the data point in multidimensional space. (For readability, only some cell values are shown.) *Concept hierarchies* may exist for each attribute, allowing the analysis of data at multiple levels of abstraction. For example, a hierarchy for *branch* could allow branches to be grouped into regions, based on their address. Data cubes provide fast access to precomputed, summarized data, thereby benefiting online analytical processing as well as data mining.

The cube created at the lowest level of abstraction is referred to as the *base cuboid*. The base cuboid should correspond to an individual entity of interest, such as *sales* or *customer*. In other words, the lowest level should be usable, or useful for the analysis. A cube at the highest level of abstraction is the *apex*

Year 2004	
Quarter	Sales

Year 2003	

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

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

Figure 3.10: Sales data for a given branch of *AllElectronics* for the years 2002 to 2004. On the left, the sales are shown per quarter. On the right, the data are aggregated to provide the annual sales. NOTE TO EDITOR: Please update figure by replacing years 2002, 2003, 2004 with 2008, 2009, 2010, respectively. Thanks.

cuboid. For the sales data of Figure 3.11, the apex cuboid would give one total—the total *sales* for all three years, for all item types, and for all branches. Data cubes created for varying levels of abstraction are often referred to as *cuboids*, so that a data cube may instead refer to a *lattice of cuboids*. Each higher level of abstraction further reduces the resulting data size. When replying to data mining requests, the *smallest* available cuboid relevant to the given task should be used. This issue is also addressed in Chapter 4.

3.5   Data Transformation and Data Discretization

This section presents methods of data transformation. In this preprocessing step, the data are transformed or consolidated so that the resulting mining process may be more efficient, and the patterns found may be easier to understand. Data discretization, a form of data transformation, is also discussed.

3.5.1   Overview of Data Transformation Strategies

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Strategies for data transformation include the following:

1. **Smoothing**, which works to remove noise from the data. Such techniques include binning, regression, and clustering.

2. **Attribute construction** (or *feature construction*), where new attributes are constructed and added from the given set of attributes to help the mining process.
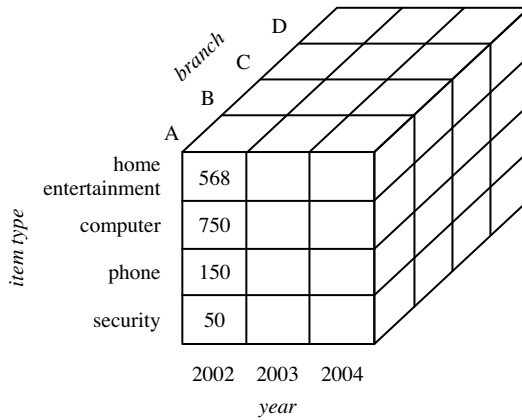
Figure 3.11: A data cube for sales at *AllElectronics*. NOTE TO EDITOR: Please update figure by replacing years 2002, 2003, 2004 with 2008, 2009, 2010, respectively. Thanks.

3. **Aggregation**, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for analysis of the data at multiple levels of abstraction.

4. **Normalization**, where the attribute data are scaled so as to fall within a smaller range, such as −1.0 to 1.0, or 0.0 to 1.0.

5. **Discretization**, where the raw values of a numeric attribute (such as *age*) are replaced by interval labels (e.g., 0-10, 11-20, and so on) or conceptual labels (e.g., *youth, adult*, and *senior*). The labels, in turn, can be recursively organized into higher-level concepts, resulting in a *concept hierarchy*
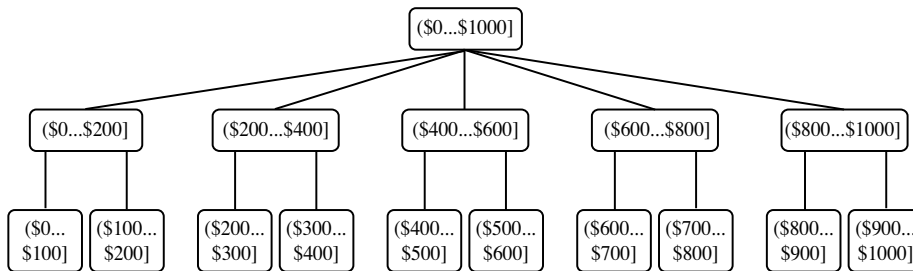

Figure 3.12: A concept hierarchy for the attribute *price*, where an interval ($X . . . $Y] denotes the range from $X (exclusive) to $Y (inclusive).

for the numeric attribute. Figure 3.12 shows a concept hierarchy for the attribute *price*. More than one concept hierarchy can be defined for the same attribute in order to accommodate the needs of various users.

6. **Concept hierarchy generation for nominal data**, where attributes such as *street* can be generalized to higher-level concepts, like *city* or *country*. Many hierarchies for nominal attributes are implicit within the database schema and can be automatically defined at the schema definition level.

Recall that there is much overlap between the major data preprocessing tasks. The first three of the above strategies were discussed earlier in this chapter. Smoothing is a form of data cleaning and was addressed in Section 3.2.2. Section 3.2.3 on the data cleaning process also discussed ETL tools, where users specify transformations to correct data inconsistencies. Attribute construction and aggregation were discussed in Section 3.4 on data reduction. In this section, we therefore concentrate on the latter three strategies.

Discretization techniques can be categorized based on how the discretization is performed, such as whether it uses class information or which direction it proceeds (i.e., top-down vs. bottom-up). If the discretization process uses class information, then we say it is *supervised discretization*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretization* or *splitting*. This contrasts with *bottom-up discretization* or *merging*, which starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals.

Data discretization and concept hierarchy generation are also forms of data reduction. The raw data are replaced by a smaller number of interval or concept labels. This simplifies the original data and makes the mining more efficient. The resulting patterns mined are typically easier to understand. Concept hierarchies are also useful for mining at multiple levels of abstraction.

The rest of this section is organized as follows. First, normalization techniques are presented in Section 3.5.2. We then describe several techniques for data discretization, each of which can be used to generate concept hierarchies for numeric attributes. The techniques include *binning* (Section 3.5.3), *histogram analysis* (Section 3.5.4), as well as *cluster analysis*, *decision-tree analysis*, and *correlation analysis* (Section 3.5.5). Finally, Section 3.5.6 describes the automatic generation of concept hierarchies for nominal data.

3.5.2   Data Transformation by Normalization

The measurement unit used can affect the data analysis. For example, changing measurement units from meters to inches for *height*, or from kilograms to pounds for *weight*, may lead to very different results. In general, expressing an attribute

in smaller units will lead to a larger range for that attribute, and thus tend to give such an attribute greater effect or "weight". To help avoid dependence on the choice of measurement units, the data should be *normalized* or *standardized*. This involves transforming the data to fall within a smaller or common range, such as $[-1, 1]$ or $[0.0, 1.0]$. (The terms "standardize" and "normalize" are used interchangeably in data preprocessing, although in statistics, the latter term also has other connotations.)

Normalizing the data attempts to give all attributes an equal weight. Normalization is particularly useful for classification algorithms involving neural networks, or distance measurements such as nearest-neighbor classification and clustering. If using the neural network backpropagation algorithm for classification mining (Chapter 8), normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. For distance-based methods, normalization helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). It is also useful when given no prior knowledge of the data.

There are many methods for data normalization. We study *min-max normalization, z-score normalization,* and *normalization by decimal scaling.* For our discussion, let A be a numeric attribute with n observed values, $v_1, v_2, \ldots, v_n$.

Min-max normalization performs a linear transformation on the original data. Suppose that min_A and max_A are the minimum and maximum values of an attribute, A. Min-max normalization maps a value, v_i, of A to v_i' in the range $[new\_min_A, new\_max_A]$ by computing

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A. \qquad (3.8)$$

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

Example 3.4 **Min-max normalization.** Suppose that the minimum and maximum values for the attribute *income* are \$12,000 and \$98,000, respectively. We would like to map *income* to the range $[0.0, 1.0]$. By min-max normalization, a value of \$73,600 for *income* is transformed to $\frac{73,600-12,000}{98,000-12,000}(1.0 - 0) + 0 = 0.716$.

In **z-score normalization** (or *zero-mean normalization*), the values for an attribute, A, are normalized based on the mean (i.e., average) and standard deviation of A. A value, v_i, of A is normalized to v_i' by computing

$$v_i' = \frac{v_i - \bar{A}}{\sigma_A}, \qquad (3.9)$$

where $\bar{A}$ and σ_A are the mean and standard deviation, respectively, of attribute A. The mean and standard deviation were discussed in Section 2.2 of this book, where $\bar{A} = \frac{1}{n}(v_1 + v_2 + \cdots + v_n)$ and σ_A is computed as the square root of the

variance of A (see Equation (2.6)). This method of normalization is useful when the actual minimum and maximum of attribute A are unknown, or when there are outliers that dominate the min-max normalization.

Example 3.5 **z-score Normalization** Suppose that the mean and standard deviation of the values for the attribute *income* are \$54,000 and \$16,000, respectively. With z-score normalization, a value of \$73,600 for *income* is transformed to $\frac{73,600-54,000}{16,000} = 1.225$.

A variation of the above z-score normalization replaces the standard deviation of Equation 3.9 by the *mean absolute deviation* of A. The *mean absolute deviation* of A, denoted s_A, is:

$$s_A = \frac{1}{n}(|v_1 - \bar{A}| + |v_2 - \bar{A}| + \cdots + |v_n - \bar{A}|). \qquad (3.10)$$

Thus, z-score normalization using the mean absolute deviation is:

$$v_i' = \frac{v_i - \bar{A}}{s_A}. \qquad (3.11)$$

The mean absolute deviation, s_A, is more robust to outliers than the standard deviation, σ_A. When computing the mean absolute deviation, the deviations from the mean (i.e., $|x_i - \bar{x}|$) are not squared; hence, the effect of outliers is somewhat reduced.

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A. The number of decimal points moved depends on the maximum absolute value of A. A value, v_i, of A is normalized to v_i' by computing

$$v_i' = \frac{v_i}{10^j}, \qquad (3.12)$$

where j is the smallest integer such that $Max(|v_i'|) < 1$.

Example 3.6 **Decimal scaling.** Suppose that the recorded values of A range from -986 to 917. The maximum absolute value of A is 986. To normalize by decimal scaling, we therefore divide each value by $1,000$ (i.e., $j = 3$) so that -986 normalizes to -0.986 and 917 normalizes to 0.917.

Note that normalization can change the original data quite a bit, especially when using z-score normalization or decimal scaling. It is also necessary to save the normalization parameters (such as the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

3.5.3   Discretization by Binning

Binning is a top-down splitting technique based on a specified number of bins. Section 3.2.2 discussed binning methods for data smoothing. These methods

are also used as discretization methods for data reduction and concept hierarchy generation. For example, attribute values can be discretized by applying equal-width or equal-frequency binning, and then replacing each bin value by the bin mean or median, as in *smoothing by bin means* or *smoothing by bin medians*, respectively. These techniques can be applied recursively to the resulting partitions in order to generate concept hierarchies.

Binning does not use class information and is therefore an unsupervised discretization technique. It is sensitive to the user-specified number of bins, as well as the presence of outliers.

3.5.4   Discretization by Histogram Analysis

Like binning, histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms were introduced in Section 2.2.9. A histogram partitions the values of an attribute, A, into disjoint ranges called *buckets*.

Various partitioning rules can be used to define histograms (Section 3.4.6). In an *equal-width* histogram, for example, the values are partitioned into equal-sized partitions or ranges (such as in Figure 3.8 for *price*, where each bucket has a width of \$10). With an *equal-frequency* histogram, the values are partitioned so that, ideally, each partition contains the same number of data tuples. The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A *minimum interval size* can also be used per level to control the recursive procedure. This specifies the minimum width of a partition, or the minimum number of values for each partition at each level. Histograms can also be partitioned based on cluster analysis of the data distribution, as described below.

3.5.5   Discretization by Cluster, Decision Tree, and Correlation Analyses

Clustering, decision tree analysis, and correlation analysis can be used for data discretization. We briefly study each of these approaches.

Cluster analysis is a popular data discretization method. A clustering algorithm can be applied to discretize a numeric attribute, A, by partitioning the values of A into clusters or groups. Clustering takes the distribution of A into consideration, as well as the closeness of data points, and therefore is able to produce high-quality discretization results.

Clustering can be used to generate a concept hierarchy for A by following either a top-down splitting strategy or a bottom-up merging strategy, where each cluster forms a node of the concept hierarchy. In the former, each initial cluster or partition may be further decomposed into several subclusters, forming a lower level of the hierarchy. In the latter, clusters are formed by repeatedly grouping neighboring clusters in order to form higher-level concepts. Clustering methods for data mining are studied in Chapters 10 and 11.

Techniques to generate decision trees for classification (Chapter 8) can be applied to discretization. Such techniques employ a top-down splitting approach. Unlike the other methods mentioned so far, decision tree approaches to discretization are supervised, that is, they make use of class label information. For example, we may have a data set of patient symptoms (the attributes), where each patient has an associated *diagnosis* class label. Class distribution information is used in the calculation and determination of split-points (data values for partitioning an attribute range). Intuitively, the main idea is to select split-points so that a given resulting partition contains as many tuples of the same class as possible. *Entropy* is the most commonly used measure for this purpose. To discretize a numeric attribute, A, the method selects the value of A that has the minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization. Such discretization forms a concept hierarchy for A.

Because decision-tree-based discretization uses class information, it is more likely that the interval boundaries (split-points) are defined to occur in places that may help improve classification accuracy. Decision trees and the entropy measure are described in greater detail in Section 8.3.2.

Measures of correlation can be used for discretization. *ChiMerge* is a χ^2-based discretization method. The discretization methods that we have studied up to this point have all employed a top-down, splitting strategy. This contrasts with ChiMerge, which employs a bottom-up approach by finding the best neighboring intervals and then merging these to form larger intervals, recursively. As with decision tree analysis, ChiMerge is supervised in that it uses class information. The basic notion is that for accurate discretization, the relative class frequencies should be fairly consistent within an interval. Therefore, if two adjacent intervals have a very similar distribution of classes, then the intervals can be merged. Otherwise, they should remain separate.

ChiMerge proceeds as follows. Initially, each distinct value of a numeric attribute A is considered to be one interval. χ^2 tests are performed for every pair of adjacent intervals. Adjacent intervals with the least χ^2 values are merged together, because low χ^2 values for a pair indicate similar class distributions. This merging process proceeds recursively until a predefined stopping criterion is met.

3.5.6 Concept Hierarchy Generation for Nominal Data

We now look at data transformation for nominal data. In particular, we study the generation of concept hierarchies for nominal attributes. Nominal attributes have a finite (but possibly large) number of distinct values, with no ordering among the values. Examples include *geographic location*, *job category*, and *item type*.

Manual definition of concept hierarchies can be a tedious and time-consuming task for a user or a domain expert. Fortunately, many hierarchies are implicit within the database schema and can be automatically defined at the schema definition level. The concept hierarchies can be used to transform the data into

multiple levels of granularity. For example, data mining patterns regarding sales may be found relating to specific regions or countries, in addition to individual branch locations.

We study four methods for the generation of concept hierarchies for nominal data.

1. **Specification of a partial ordering of attributes explicitly at the schema level by users or experts:** Concept hierarchies for nominal attributes or dimensions typically involve a group of attributes. A user or an expert can easily define a concept hierarchy by specifying a partial or total ordering of the attributes at the schema level. For example, suppose that a relational database contains the following group of attributes: *street, city, province_or_state*, and *country*. Similarly, a *location* dimension of a data warehouse may contain the same attributes. A hierarchy can be defined by specifying the total ordering among these attributes at the schema level, such as *street < city < province_or_state < country*.

2. **Specification of a portion of a hierarchy by explicit data grouping:** This is essentially the manual definition of a portion of a concept hierarchy. In a large database, it is unrealistic to define an entire concept hierarchy by explicit value enumeration. On the contrary, we can easily specify explicit groupings for a small portion of intermediate-level data. For example, after specifying that *province* and *country* form a hierarchy at the schema level, a user could define some intermediate levels manually, such as "{*Alberta, Saskatchewan, Manitoba*} ⊂ *prairies_Canada*" and "{*British Columbia, prairies_Canada*} ⊂ *Western_Canada*".

3. **Specification of a *set of attributes*, but not of their partial ordering:** A user may specify a set of attributes forming a concept hierarchy, but omit to explicitly state their partial ordering. The system can then try to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy.

   "Without knowledge of data semantics, how can a hierarchical ordering for an arbitrary set of nominal attributes be found?" Consider the following observation that since higher-level concepts generally cover several subordinate lower-level concepts, an attribute defining a high concept level (e.g., *country*) will usually contain a smaller number of distinct values than an attribute defining a lower concept level (e.g., *street*). Based on this observation, a concept hierarchy can be automatically generated based on the number of distinct values per attribute in the given attribute set. The attribute with the most distinct values is placed at the lowest level of the hierarchy. The lower the number of distinct values an attribute has, the higher it is in the generated concept hierarchy. This heuristic rule works well in many cases. Some local-level swapping or adjustments may be applied by users or experts, when necessary, after examination of the generated hierarchy.

country   **15 distinct values**

province_or_state   **365 distinct values**

city   **3,567 distinct values**

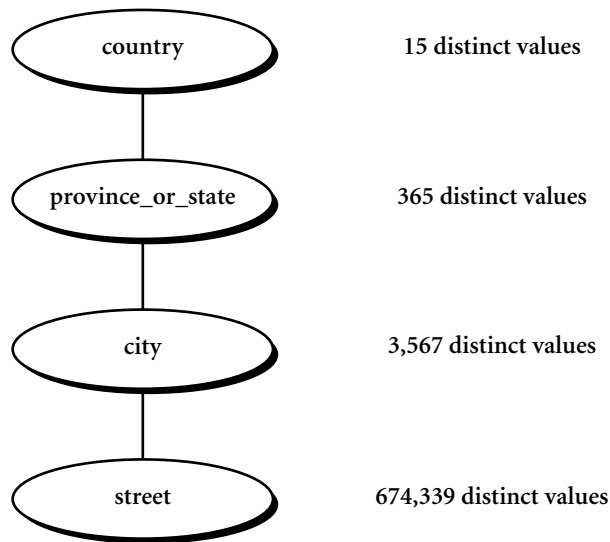street   **674,339 distinct values**

Figure 3.13: Automatic generation of a schema concept hierarchy based on the number of distinct attribute values.

Let's examine an example of this third method.

Example 3.7 **Concept hierarchy generation based on the number of distinct values per attribute.** Suppose a user selects a set of location-oriented attributes, *street, country, province_or_state*, and *city*, from the *AllElectronics* database, but does not specify the hierarchical ordering among the attributes.

A concept hierarchy for *location* can be generated automatically, as illustrated in Figure 3.13. First, sort the attributes in ascending order based on the number of distinct values in each attribute. This results in the following (where the number of distinct values per attribute is shown in parentheses): *country* (15), *province_or_state* (365), *city* (3567), and *street* (674,339). Second, generate the hierarchy from the top down according to the sorted order, with the first attribute at the top level and the last attribute at the bottom level. Finally, the user can examine the generated hierarchy, and when necessary, modify it to reflect desired semantic relationships among the attributes. In this example, it is obvious that there is no need to modify the generated hierarchy.

Note that this heuristic rule is not foolproof. For example, a time dimension in a database may contain 20 distinct years, 12 distinct months, and 7 distinct days of the week. However, this does not suggest that the time hierarchy should be "*year < month < days_of_the_week*", with *days_of_the_week* at the top of the hierarchy.

4. **Specification of only a partial set of attributes:** Sometimes a user can be careless when defining a hierarchy, or have only a vague idea about

what should be included in a hierarchy. Consequently, the user may have included only a small subset of the relevant attributes in the hierarchy specification. For example, instead of including all of the hierarchically relevant attributes for *location*, the user may have specified only *street* and *city*. To handle such partially specified hierarchies, it is important to embed data semantics in the database schema so that attributes with tight semantic connections can be pinned together. In this way, the specification of one attribute may trigger a whole group of semantically tightly linked attributes to be "dragged in" to form a complete hierarchy. Users, however, should have the option to override this feature, as necessary.

Example 3.8 **Concept hierarchy generation using prespecified semantic connections.** Suppose that a data mining expert (serving as an administrator) has pinned together the five attributes *number, street, city, province_or_state*, and *country*, because they are closely linked semantically regarding the notion of *location*. If a user were to specify only the attribute *city* for a hierarchy defining *location*, the system can automatically drag in all of the above five semantically related attributes to form a hierarchy. The user may choose to drop any of these attributes, such as *number* and *street*, from the hierarchy, keeping *city* as the lowest conceptual level in the hierarchy.

In summary, information at the schema level and of attribute-value counts can be used to generate concept hierarchies for nominal data. Transforming nominal data with the use of concept hierarchies allows higher-level knowledge patterns to be found. It allows mining at multiple levels of abstraction, which is a common requirement for data mining applications.

3.6 Summary

- **Data quality** is defined in terms of *accuracy, completeness, consistency, timeliness, believability*, and *interpretabilty*. These qualities are assessed based on the intended use of the data.

- **Data cleaning** routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. Data cleaning is usually performed as an iterative two-step process consisting of discrepancy detection and data transformation.

- **Data integration** combines data from multiple sources to form a coherent data store. The resolution of semantic heterogeneity, metadata, correlation analysis, tuple duplication detection, and data conflict detection contribute toward smooth data integration.

- **Data reduction** techniques obtain a reduced representation of the data while minimizing the loss of information content. These include methods of *dimensionality reduction*, *numerosity reduction*, and *data compression*. **Dimensionality reduction** reduces the number of random variables or

attributes under consideration. Methods include *wavelet transforms, principal components analysis, attribute subset selection*, and *attribute creation*. **Numerosity reduction** methods use parametric or nonparatmetric models to obtain smaller representations of the original data. Parametric models store only the model parameters instead of the actual data. Examples include regression and log-linear models. Nonparamteric methods include histograms, clustering, sampling, and data cube aggregation. **Data compression** methods apply transformations to obtain a reduced or "compressed" representation of the original data. The data reduction is *lossless* if the original data can be reconstructed from the compressed data without any loss of information; otherwise, it is *lossy*.

- **Data transformation** routines convert the data into appropriate forms for mining. For example, in **normalization**, attribute data are scaled so as to fall within a small range such as 0.0 to 1.0. Other examples are **data discretization** and **concept hierarchy generation**.

- **Data discretization** transforms numeric data by mapping values to interval or concept labels. Such methods can be used to automatically generate *concept hierarchies* for the data, which allows for mining at multiple levels of granularity. Discretization techniques include binning, histogram analysis, cluster analysis, decision-tree analysis, and correlation analysis. For nominal data, **concept hierarchies** may be generated based on schema definitions as well as the number of distinct values per attribute.

- Although numerous methods of data preprocessing have been developed, data preprocessing remains an active area of research, due to the huge amount of inconsistent or dirty data and the complexity of the problem.

3.7   Exercises

1. *Data quality* can be assessed in terms of several issues, including accuracy, completeness, and consistency. For each of the above three issues, discuss how the assessment of data quality can depend on the *intended use* of the data, giving examples Propose two other dimensions of data quality.

2. In real-world data, tuples with *missing values* for some attributes are a common occurrence. Describe various methods for handling this problem.

3. Exercise 2.2 gave the following data (in increasing order) for the attribute *age*: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

   (a) Use *smoothing by bin means* to smooth the above data, using a bin depth of 3. Illustrate your steps. Comment on the effect of this technique for the given data.

   (b) How might you determine *outliers* in the data?

(c) What other methods are there for *data smoothing*?

4. Discuss issues to consider during *data integration*.

5. What are the value ranges of the following *normalization methods*?

   (a) min-max normalization
   (b) z-score normalization
   (c) z-score normalization using the mean absolute deviation instead of standard deviation
   (d) normalization by decimal scaling

6. Use the methods below to *normalize* the following group of data:

$$200, 300, 400, 600, 1000$$

   (a) min-max normalization by setting $min = 0$ and $max = 1$
   (b) z-score normalization
   (c) z-score normalization using the mean absolute deviation instead of standard deviation
   (d) normalization by decimal scaling

7. Using the data for *age* given in Exercise 3.3, answer the following:

   (a) Use min-max normalization to transform the value 35 for *age* onto the range $[0.0, 1.0]$.
   (b) Use z-score normalization to transform the value 35 for *age*, where the standard deviation of *age* is 12.94 years.
   (c) Use normalization by decimal scaling to transform the value 35 for *age*.
   (d) Comment on which method you would prefer to use for the given data, giving reasons as to why.

8. Using the data for *age* and *body fat* given in Exercise 2.4, answer the following:

   (a) Normalize the two attributes based on *z-score normalization*.
   (b) Calculate the *correlation coefficient* (Pearson's product moment coefficient). Are these two attributes positively or negatively correlated? Compute their covariance.

9. Suppose a group of 12 *sales price* records has been sorted as follows:

$$5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215.$$

Partition them into three bins by each of the following methods.

    (a) equal-frequency (equidepth) partitioning

    (b) equal-width partitioning

    (c) clustering

10. Use a flowchart to summarize the following procedures for *attribute subset selection*:

    (a) stepwise forward selection

    (b) stepwise backward elimination

    (c) a combination of forward selection and backward elimination

11. Using the data for *age* given in Exercise 3.3,

    (a) Plot an equal-width histogram of width 10.

    (b) Sketch examples of each of the following sampling techniques: SR-SWOR, SRSWR, cluster sampling, stratified sampling. Use samples of size 5 and the strata "youth", "middle-aged", and "senior".

12. ChiMerge [Ker92] is a supervised, bottom-up (i.e., merge-based) *data discretization* method. It relies on χ^2 analysis: adjacent intervals with the least χ^2 values are merged together till the chosen stopping criterion satisfies.

    (a) Briefly describe how ChiMerge works.

    (b) Take the IRIS data set, obtained from the UC-Irvine Machine Learning Data Repository (*http://www.ics.uci.edu/∼mlearn/MLRepository.html*), as a data set to be discretized. Perform data discretization for each of the four numerical attributes using the ChiMerge method. (Let the stopping criteria be: *max-interval* = 6). You need to write a small program to do this to avoid clumsy numerical computation. Submit your simple analysis and your test results: split points, final intervals, and your documented source program.

13. Propose an algorithm, in pseudocode or in your favorite programming language, for the following:

    (a) The automatic generation of a concept hierarchy for categorical data based on the number of distinct values of attributes in the given schema

    (b) The automatic generation of a concept hierarchy for numerical data based on the *equal-width* partitioning rule

    (c) The automatic generation of a concept hierarchy for numerical data based on the *equal-frequency* partitioning rule

14. Robust data loading poses a challenge in database systems because the input data are often dirty. In many cases, an input record may miss multiple values, some records could be *contaminated*, with some data values out of range or of a different data type than expected. Work out an automated *data cleaning and loading* algorithm so that the erroneous data will be marked, and contaminated data will not be mistakenly inserted into the database during data loading.

3.8 Bibliographic Notes

Data preprocessing is discussed in a number of textbooks, including English [Eng99], Pyle [Pyl99], Loshin [Los01], Redman [Red01], and Dasu and Johnson [DJ03]. More specific references to individual preprocessing techniques are given below.

For discussion regarding data quality, see Redman [Red92], Wang, Storey, and Firth [WSF95], Wand and Wang [WW96], Ballou and Tayi [BT99], and Olson [Ols03]. Potter's Wheel (`control.cx.berkely.edu/abc`), the interactive data cleaning tool described in Section 3.2.3, is presented in Raman and Hellerstein [RH01]. An example of the development of declarative languages for the specification of data transformation operators is given in Galhardas et al. [GFS⁺01]. The handling of missing attribute values is discussed in Friedman [Fri77], Breiman, Friedman, Olshen, and Stone [BFOS84], and Quinlan [Qui89]. Hua and Pei [HP07] presented a heuristic approach to clean *disguised missing data*, where such data is captured when users falsely select default values on forms (such as 'January 1' for *birthdate*) when they do not want to disclose personal information. A method for the detection of outlier or "garbage" patterns in a handwritten character database is given in Guyon, Matic, and Vapnik [GMV96]. Binning and data normalization are treated in many texts, including [KLV⁺98], [WI98], [Pyl99]. Systems that include attribute (or feature) construction include BACON by Langley, Simon, Bradshaw, and Zytkow [LSBZ87], Stagger by Schlimmer [Sch86], FRINGE by Pagallo [Pag89], and AQ17-DCI by Bloedorn and Michalski [BM98]. Attribute construction is also described in Liu and Motoda [LM98, Le98]. Dasu, et al. built a BELLMAN system and proposed a set of interesting methods for building a data quality browser by mining database structures [DJMS02].

A good survey of data reduction techniques can be found in Barbará et al. [BDF⁺97]. For algorithms on data cubes and their precomputation, see [SS94, AAD⁺96, HRU96, RS97, ZDN97]. Attribute subset selection (or *feature subset selection*) is described in many texts, such as Neter, Kutner, Nachtsheim, and Wasserman [NKNW96], Dash and Liu [DL97], and Liu and Motoda [LM98, LM98b]. A combination forward selection and backward elimination method was proposed in Siedlecki and Sklansky [SS88]. A wrapper approach to attribute selection is described in Kohavi and John [KJ97]. Unsupervised attribute subset selection is described in Dash, Liu, and Yao [DLY97]. For a description of wavelets for dimensionality reduction, see Press, Teukolosky,

Vetterling, and Flannery [PTVF96]. A general account of wavelets can be found in Hubbard [Hub96]. For a list of wavelet software packages, see Bruce, Donoho, and Gao [BDG96]. Daubechies transforms are described in Daubechies [Dau92]. The book by Press, et al. [PTVF96] includes an introduction to singular value decomposition for principal components analysis. Routines for PCA are included in most statistical software packages, such as SAS (`http://www.sas.com/SASHome.html`).

An introduction to regression and log-linear models can be found in several textbooks, such as [Jam85, Dob90, JW92, Dev95, NKNW96]. For log-linear models (known as *multiplicative models* in the computer science literature), see Pearl [Pea88]. For a general introduction to histograms, see Barbará et al. [BDF$^+$97] and Devore and Peck [DP97]. For extensions of single attribute histograms to multiple attributes, see Muralikrishna and DeWitt [MD88] and Poosala and Ioannidis [PI97]. Several references to clustering algorithms are given in Chapter 7 of this book, which is devoted to the topic. A survey of multidimensional indexing structures is given in Gaede and Günther [GG98]. The use of multidimensional index trees for data aggregation is discussed in Aoki [Aok98]. Index trees include R-trees (Guttman [Gut84]), quad-trees (Finkel and Bentley [FB74]), and their variations. For discussion on sampling and data mining, see Kivinen and Mannila [KM94] and John and Langley [JL96].

There are many methods for assessing attribute relevance. Each has its own bias. The information gain measure is biased towards attributes with many values. Many alternatives have been proposed, such as gain ratio (Quinlan [Qui93]), which considers the probability of each attribute value. Other relevance measures include the gini index (Breiman, Friedman, Olshen, and Stone [BFOS84]), the χ^2 contingency table statistic, and the uncertainty coefficient (Johnson and Wichern [JW92]). For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [BN92]. For additional methods, see Liu and Motoda [LM98]b, Dash and Liu [DL97], and Almuallim and Dietterich [AD91].

Liu et al. [LHTD02] performed a comprehensive survey of data discretization methods. Entropy-based discretization with the C4.5 algorithm is described in Quinlan [Qui93]. In Catlett [Cat91], the D-2 system binarizes a numerical feature recursively. ChiMerge by Kerber [Ker92] and Chi2 by Liu and Setiono [LS95] are methods for the automatic discretization of numerical attributes that both employ the χ^2 statistic. Fayyad and Irani [FI93] apply the minimum description length principle to determine the number of intervals for numerical discretization. Concept hierarchies and their automatic generation from categorical data are described in Han and Fu [HF94].

Bibliography

[AAD+96] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, pages 506–521, Bombay, India, Sept. 1996.

[AD91] H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proc. 1991 Nat. Conf. Artificial Intelligence (AAAI'91)*, pages 547–552, Anaheim, CA, July 1991.

[Aok98] P. M. Aoki. Generalizing "search" in generalized search trees. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 380–389, Orlando, FL, Feb. 1998.

[BDF+97] D. Barbará, W. DuMouchel, C. Faloutos, P. J. Haas, J. H. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Servcik. The New Jersey data reduction report. *Bull. Technical Committee on Data Engineering*, 20:3–45, Dec. 1997.

[BDG96] A. Bruce, D. Donoho, and H.-Y. Gao. Wavelet analysis. In *IEEE Spectrum*, pages 26–35, Oct 1996.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[BM98] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. 11th Conf. on Computational Learning Theory (COLT' 98)*, pages 92–100, Madison, WI, 1998.

[BN92] W. L. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[BT99] D. P. Ballou and G. K. Tayi. Enhancing data quality in data warehouse environments. *Comm. ACM*, 42:73–78, 1999.

[Cat91] J. Catlett. *Megainduction: Machine Learning on Very large Databases*. Ph.D. Thesis, University of Sydney, 1991.

[Dau92]   I. Daubechies. *Ten Lectures on Wavelets*. Capital City Press, 1992.

[Dev95]   J. L. Devore. *Probability and Statistics for Engineering and the Science* (4th ed.). Duxbury Press, 1995.

[DJ03]   T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, 2003.

[DJMS02] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or how to build a data quality browser. In *Proc. 2002 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'02)*, pages 240–251, Madison, WI, June 2002.

[DL97]   M. Dash and H. Liu. Feature selection methods for classification. *Intelligent Data Analysis*, 1:131–156, 1997.

[DLY97]   M. Dash, H. Liu, and J. Yao. Dimensionality reduction of unsupervised data. In *Proc. 1997 IEEE Int. Conf. Tools with AI (IC-TAI'97)*, pages 532–539, IEEE Computer Society, 1997.

[Dob90]   A. J. Dobson. *An Introduction to Generalized Linear Models*. Chapman and Hall, 1990.

[DP97]   J. Devore and R. Peck. *Statistics: The Exploration and Analysis of Data*. Duxbury Press, 1997.

[Eng99]   L. English. *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits*. John Wiley & Sons, 1999.

[FB74]   R. A. Finkel and J. L. Bentley. Quad-trees: A data structure for retrieval on composite keys. *ACTA Informatica*, 4:1–9, 1974.

[FI93]   U. Fayyad and K. Irani. Multi-interval discretization of continuous-values attributes for classification learning. In *Proc. 1993 Int. Joint Conf. Artificial Intelligence (IJCAI'93)*, pages 1022–1029, Chambery, France, 1993.

[Fri77]   J. H. Friedman. A recursive partitioning decision rule for nonparametric classifiers. *IEEE Trans. Computer*, 26:404–408, 1977.

[GFS+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proc. 2001 Int. Conf. on Very Large Data Bases (VLDB'01)*, pages 371–380, Rome, Italy, Sept. 2001.

[GG98]   V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.

[GMV96] I. Guyon, N. Matic, and V. Vapnik. Discoverying informative patterns and data cleaning. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 181–203. AAAI/MIT Press, 1996.

[Gut84] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proc. 1984 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'84)*, pages 47–57, Boston, MA, June 1984.

[HF94] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, July 1994.

[HP07] M. Hua and J. Pei. Cleaning disguised missing data: A heuristic approach. In *Proc. 2007 ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD'07)*, San Jose, CA, Aug. 2007.

[HRU96] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, pages 205–216, Montreal, Canada, June 1996.

[Hub96] B. B. Hubbard. *The World According to Wavelets*. A. K. Peters, 1996.

[Jam85] M. James. *Classification Algorithms*. John Wiley & Sons, 1985.

[JL96] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 367–370, Portland, OR, Aug. 1996.

[JW92] R. A. Johnson and D. A. Wichern. *Applied Multivariate Statistical Analysis* (3rd ed.). Prentice Hall, 1992.

[Ker92] R. Kerber. Discretization of numeric attributes. In *Proc. 1992 Nat. Conf. Artificial Intelligence (AAAI'92)*, pages 123–128, AAAI/MIT Press, 1992.

[KJ97] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324, 1997.

[KLV+98] R. L Kennedy, Y. Lee, B. Van Roy, C. D. Reed, and R. P. Lippman. *Solving Data Mining Problems Through Pattern Recognition*. Prentice Hall, 1998.

[KM94] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proc. 13th ACM Symp. Principles of Database Systems*, pages 77–85, Minneapolis, MN, May 1994.

[Le98]   H. Liu and H. Motoda (eds.). *Feature Extraction, Construction, and Selection: A Data Mining Perspective*. Kluwer Academic, 1998.

[LHTD02]   H. Liu, F. Hussain, C. L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6:393–423, 2002.

[LM98]   H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic, 1998.

[Los01]   D. Loshin. *Enterprise Knowledge Management: The Data Quality Approach*. Morgan Kaufmann, 2001.

[LS95]   H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proc. 1995 IEEE Int. Conf. Tools with AI (ICTAI'95)*, pages 388–391, Washington, DC, Nov. 1995.

[LSBZ87]   P. Langley, H. A. Simon, G. L. Bradshaw, and J. M. Zytkow. *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT Press, 1987.

[MD88]   M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for extimating selectivity factors for multi-dimensional queries. In *Proc. 1988 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'88)*, pages 28–36, Chicago, IL, June 1988.

[NKNW96]   J. Neter, M. H. Kutner, C. J. Nachtsheim, and L. Wasserman. *Applied Linear Statistical Models* (4th ed.). Irwin, 1996.

[Ols03]   J. E. Olson. *Data Quality: The Accuracy Dimension*. Morgan Kaufmann, 2003.

[Pag89]   G. Pagallo. Learning DNF by decision trees. In *Proc. 1989 Int. Joint Conf. Artificial Intelligence (IJCAI'89)*, pages 639–644, Morgan Kaufmann, 1989.

[Pea88]   J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kauffman, 1988.

[PI97]   V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97)*, pages 486–495, Athens, Greece, Aug. 1997.

[PTVF96]   W. H. Press, S. A. Teukolosky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1996.

[Pyl99]   D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui89] J. R. Quinlan. Unknown attribute values in induction. In *Proc. 1989 Int. Conf. Machine Learning (ICML'89)*, pages 164–168, Ithaca, NY, June 1989.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Red92] T. Redman. *Data Quality: Management and Technology*. Bantam Books, 1992.

[Red01] T. Redman. *Data Quality: The Field Guide*. Digital Press (Elsevier), 2001.

[RH01] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proc. 2001 Int. Conf. on Very Large Data Bases (VLDB'01)*, pages 381–390, Rome, Italy, Sept. 2001.

[RS97] K. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97)*, pages 116–125, Athens, Greece, Aug. 1997.

[Sch86] J. C. Schlimmer. Learning and representation change. In *Proc. 1986 Nat. Conf. Artificial Intelligence (AAAI'86)*, pages 511–515, Philadelphia, PA, 1986.

[SS88] W. Siedlecki and J. Sklansky. On automatic feature selection. *Int. J. Pattern Recognition and Artificial Intelligence*, 2:197–220, 1988.

[SS94] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 1994 Int. Conf. Data Engineering (ICDE'94)*, pages 328–336, Houston, TX, Feb. 1994.

[WI98] S. M. Weiss and N. Indurkhya. *Predictive Data Mining*. Morgan Kaufmann, 1998.

[WSF95] R. Wang, V. Storey, and C. Firth. A framework for analysis of data quality research. *IEEE Trans. Knowledge and Data Engineering*, 7:623–640, 1995.

[WW96] Y. Wand and R. Wang. Anchoring data quality dimensions in ontological foundations. *Comm. ACM*, 39:86–95, 1996.

[ZDN97] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, pages 159–170, Tucson, AZ, May 1997.