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 several data preprocessing techniques. *Data cleaning* can be applied to remove noise and correct inconsistencies in data. *Data integration* merges data from multiple sources into a coherent data store such as a data warehouse. *Data reduction* can reduce data size by, for instance, aggregating, eliminating redundant features, or clustering. *Data transformations* (e.g., 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 data characteristics. 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 have quality if they satisfy the requirements of the intended use. There are many factors comprising **data quality**, including *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 your branch's sales. 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 (e.g., *item*, *price*, and *units_sold*) to be included in your analysis. 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 (i.e., 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, or inconsistent formats for input fields (e.g., *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 data history or modifications 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 before 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 database's accuracy, 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 are incomplete. However, once all of the data are received, it is correct. The fact that the month-end data are 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 sales department users, 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 the database is now accurate, complete, consistent, and timely, sales department users 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. 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 data cleaning.

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 (i.e., **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, "William" in another, and "B." in a 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 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 (e.g., *wavelet transforms* and *principal components analysis*), *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 (e.g., *regression* or *log-linear models*) or nonparametric models (e.g., *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 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 data mining at multiple abstraction levels. Normalization, data

[1] Neural networks and nearest-neighbor classifiers are described in Chapter 9, and clustering is discussed in Chapters 10 and 11.

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 previous 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 data quality, thereby helping to improve the accuracy and efficiency of the subsequent mining process. Data preprocessing 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.

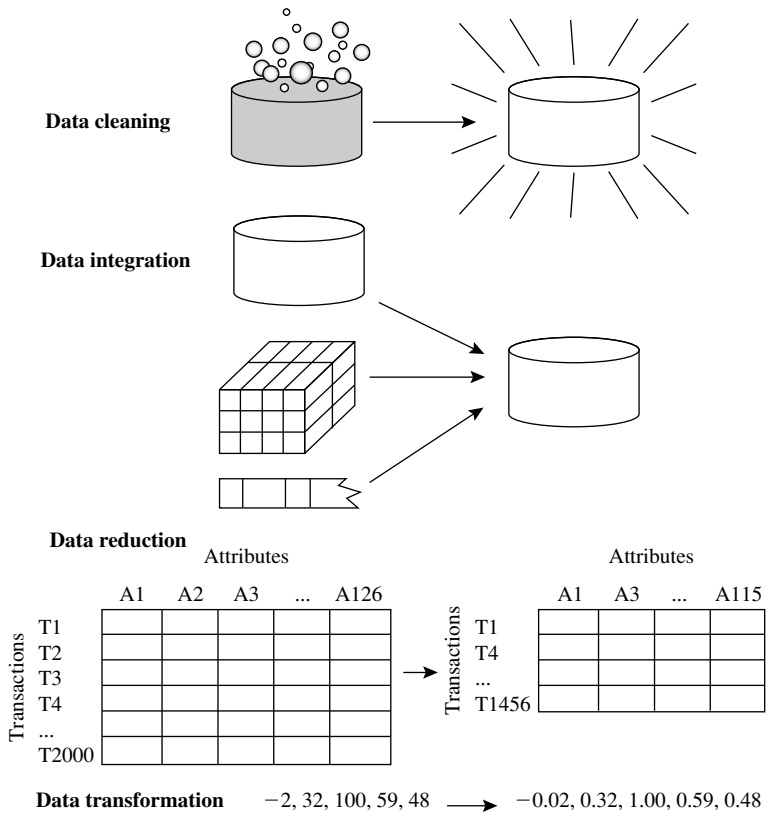

Figure 3.1 Forms of data preprocessing.

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 (e.g., 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 mean 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 mean *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 through 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 other attributes' values 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 (e.g., "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 database and data entry procedure design 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 (e.g., 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.

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

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.

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.

Regression: Data smoothing can also be done by regression, a technique that conforms data values to a function. *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.

Outlier analysis: Outliers may be detected by clustering, for example, 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 12 is dedicated to the topic of outlier analysis.

Many data smoothing methods are also used for data discretization (a form of data transformation) and data reduction. For example, the binning techniques described before 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 makes 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

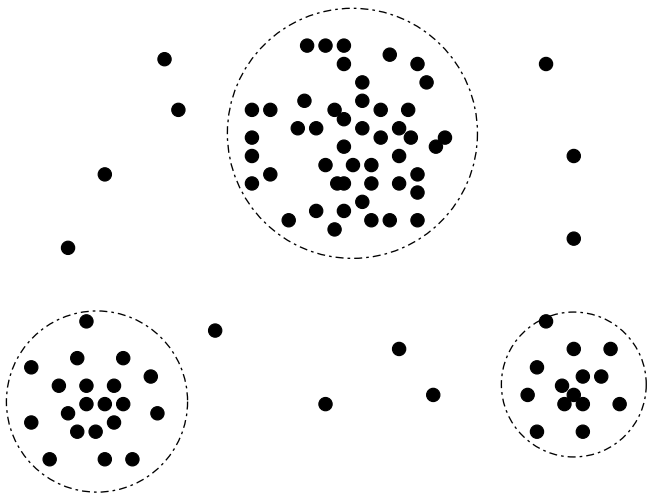

Figure 3.3 A 2-D customer data plot with respect to customer locations in a city, showing three data clusters. Outliers may be detected as values that fall outside of the cluster sets.

process. Data discretization is discussed in Section 3.5. Some methods of classification (e.g., 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 inconsistent use of codes. Other sources of discrepancies include errors in instrumentation devices that record data and system errors. 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]

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

"*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 later. 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 (e.g., "2010/12/25" and "25/12/2010" for *date*). **Field overloading** is another error source that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., an unused bit of an attribute that has a value range that uses only, say, 31 out of 32 bits).

The data should also be examined regarding unique rules, consecutive rules, and null rules. A **unique rule** says that each value of the given attribute must be different from all other values for that attribute. A **consecutive rule** says that there can be no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers). A **null rule** specifies the use of blanks, question marks, special characters, or other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled. As mentioned in Section 3.2.1, reasons for missing values may include (1) the person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a *license_number* attribute left blank by 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 (e.g., entries like "don't know" or "?" should be transformed to blank).

There are a number of different commercial tools that can aid in the discrepancy detection step. **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 mistakenly created. 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 automatically performs discrepancy checking 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 data 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 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*" but *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 from 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

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

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 Eq. (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 for which the 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 illustrate the use of this statistic in Example 3.1. If the hypothesis can be rejected, then we say that A and B are statistically correlated.

Example 3.1 **Correlation analysis of nominal attributes using χ^2.** Suppose that a group of 1500 people was surveyed. The gender of each person was noted. Each person was polled as to whether his or her 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 Eq. (3.2).

Using Eq. (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.

Table 3.1 Example 2.1's 2 × 2 Contingency Table Data

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

Note: Are *gender* and *preferred_reading* correlated?

Using Eq. (3.1) for χ^2 computation, we get

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

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

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

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 2.2.2), and $\Sigma(a_i b_i)$ is the sum of the AB cross-product (i.e., for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \le r_{A,B} \le +1$. 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 2.2.3). For example, Figure 2.8's

scatter plots respectively show positively correlated data and negatively correlated data, while Figure 2.9 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 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

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

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

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

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

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

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* (i.e., 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

Table 3.2 Stock Prices for *AllElectronics* and *HighTech*

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

(e.g., 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*, a 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

$$E(HighTech) = \frac{20+10+14+5+5}{5} = \frac{54}{5} = \$10.80.$$

Thus, using Eq. (3.4), we compute

$$Cov(AllElectroncis, HighTech) = \frac{6\times 20+5\times 10+4\times 14+3\times 5+2\times 5}{5} - 4\times 10.80$$
$$= 50.2 - 43.2 = 7.$$

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 (i.e., 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 data occurrences. 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 Data Value Conflict Detection and Resolution

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 (e.g., free breakfast) and taxes. When exchanging information between schools, for example, 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 abstraction level, where an attribute in one system is recorded at, say, a lower abstraction level 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*