Contents

Chapter 3

Data Preprocessing

Today's real-world databases are highly susceptible to noise, missing, and inconsistent data due to their typically huge size, often several gigabytes or more. 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 or a data cube. *Data transformations*, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements. *Data reduction* can reduce the data size by aggregating, eliminating redundant features, or clustering, for instance. These data processing techniques, when applied prior to mining, can substantially improve the overall data mining results.

In this chapter, you will learn methods for data preprocessing. These methods are organized into the following categories: data cleaning, data integration and transformation, and data reduction. The use of concept hierarchies for data discretization, an alternative form of data reduction, is also discussed. Concept hierarchies can be further used to promote mining at multiple levels of abstraction. You will study how concept hierarchies can be generated automatically from the given data.

3.1 Why preprocess the data?

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 study/inspect the company's database or data warehouse, identifying and selecting the attributes or dimensions to be included in your analysis, such as *item*, *price*, and *units_sold*. Alas! You note 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), **noisy** (containing errors, or *outlier* values which deviate from the expected), and **inconsistent** (e.g., containing discrepancies in the department codes used to categorize items). Welcome to the real world!

Incomplete, noisy, and inconsistent data are commonplace properties of large, real-world databases and data warehouses. 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 it was 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.

Data can be noisy, having incorrect attribute values, owing to the following. The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. 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. Duplicate tuples also require data cleaning.

Data cleaning routines work to "clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. Dirty data can cause confusion for the mining procedure. 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, 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* is 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 "derived" or 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 may be performed to detect and remove redundancies that may have resulted from data integration.

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. Such methods provide better results if the data to be analyzed have been *normalized*, that is, scaled to a specific range such as [0, 1.0]. Your customer data, for example, contains the attributes *age*, and *annual salary*. The *annual salary* attribute can take many more values than *age*. Therefore, if the attributes are left un-normalized, then distance measurements taken on *annual salary* will generally outweigh distance measurements taken on *age*. Furthermore, it would be useful for your analysis to obtain aggregate information as to the sales per customer region — something which is not part of any precomputed data cube in your data warehouse. You soon realize that **data transformation** operations, such as normalization and aggregation, are additional data preprocessing procedures that would contribute towards the success of the mining process. Data integration and data transformation are discussed in Section 3.3.

"Hmmm", you wonder, as you consider your data even further. *"The data set I have selected for analysis is huge — it is sure to slow or wear down the mining process. Is there any 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. There are a number of strategies for data reduction. These include *data aggregation* (e.g., building a data cube), *dimension reduction* (e.g., removing irrelevant attributes through correlation analysis), *data compression* (e.g., using encoding schemes such as minimum length encoding or wavelets), and *numerosity reduction* (e.g., "replacing" the data by alternative, smaller representations such as clusters, or parametric models). Data can also be "reduced" by *generalization*, where low level concepts such as *city* for customer location, are replaced with higher level concepts, such as *region* or *province_or_state*. A concept hierarchy is used to organize the concepts into varying levels of abstraction. Data reduction is the topic of Section 3.4. Since concept hierarchies are so useful in mining at multiple levels of abstraction, we devote a separate section to the automatic generation of this important data structure. Section 3.5 discusses concept hierarchy generation, a form of data reduction by data discretization.

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 preprocessing is therefore an important step in the knowledge discovery process, since 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 pay-offs for decision making.

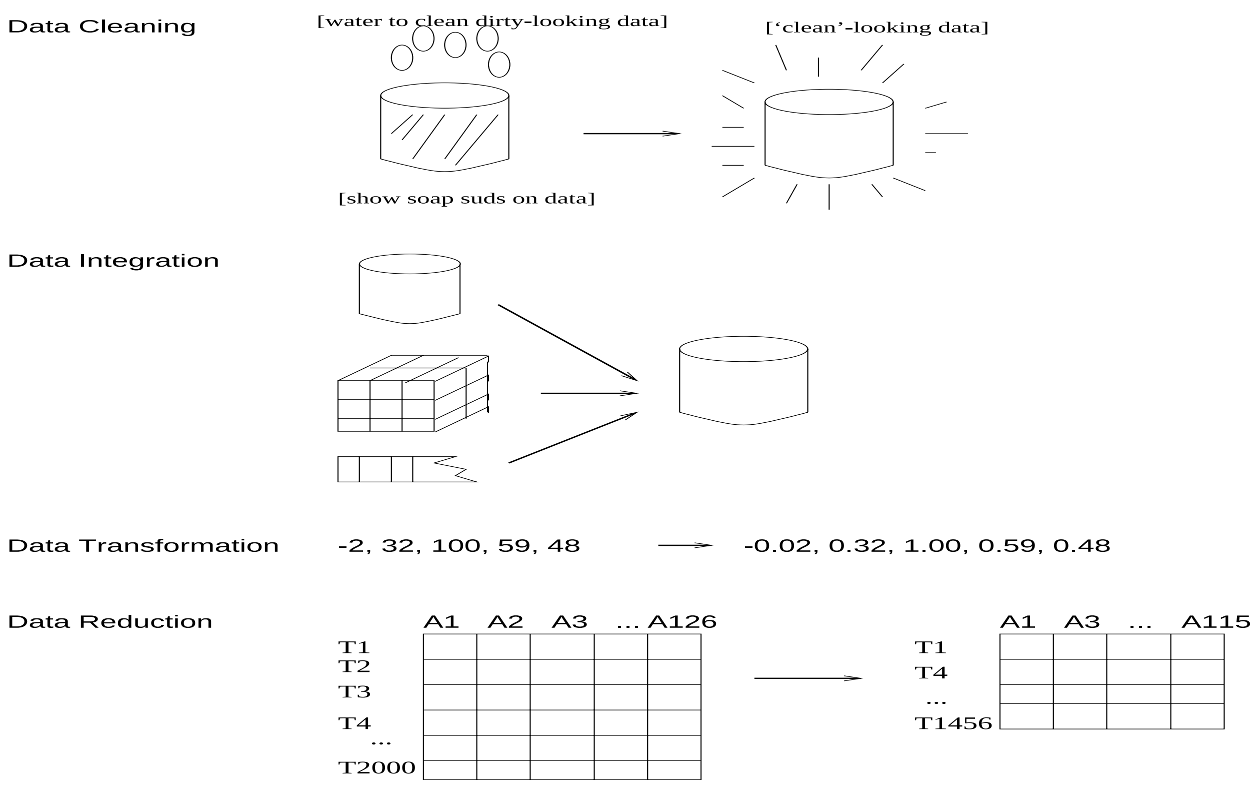


Figure 3.1: Forms of data preprocessing.

3.2 Data cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. *Data cleaning* 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.

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 or description). 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.

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 recommended.

4. **Use the attribute mean to fill in the missing value**: For example, suppose that the average income of *AllElectronics* customers is $28,000. Use this value to replace the missing value for *income*.

5. **Use the attribute mean for all samples belonging to the same class as the given tuple**: For example, if classifying customers according to *credit_risk*, replace the missing value with the average *income* value for

- **Sorted data for *price* (in dollars)**: 4, 8, 15, 21, 21, 24, 25, 28, 34
- **Partition into (equi-width) 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.

customers in the same credit risk category as that of the given tuple.

6. **Use the most probable value to fill in the missing value**: This may be determined with 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 are described in detail in Chapter 7.

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.

3.2.2 Noisy data

"What is noise?" **Noise** is a random error or variance in a measured variable. 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 methods:** Binning methods smooth a sorted data value by consulting the "neighborhood", or 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 partitioned into *equi-depth* bins (of depth 3). 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 equi-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, and in Chapter 6 on association rule mining.

2. **Clustering:** Outliers may be detected by clustering, where similar values are organized into groups or "clusters". Intuitively, values which fall outside of the set of clusters may be considered outliers (Figure 3.3). Chapter 9 is dedicated to the topic of clustering.

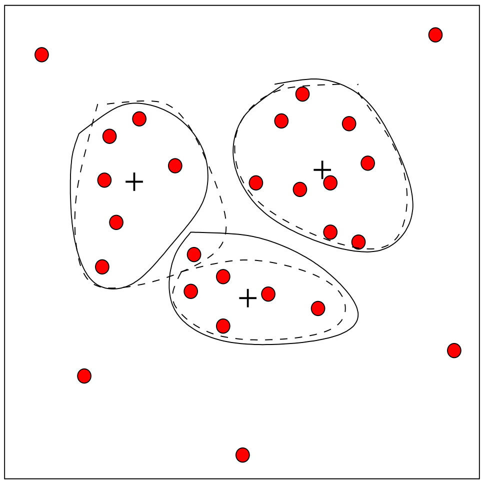


Figure 3.3: Outliers may be detected by clustering analysis.

3. **Combined computer and human inspection:** Outliers may be identified through a combination of computer and human inspection. In one application, for example, an information-theoretic measure was used to help identify outlier patterns in a handwritten character database for classification. The measure's value reflected the "surprise" content of the predicted character label with respect to the known label. Outlier patterns may be informative (e.g., identifying useful data exceptions, such as different versions of the characters "0" or "7"), or "garbage" (e.g., mislabeled characters). Patterns whose surprise content is above a threshold are output to a list. A human can then sort through the patterns in the list to identify the actual garbage ones. This is much faster than having to manually search through the entire database. The garbage patterns can then be removed from the (training) database.

4. **Regression:** Data can be smoothed by fitting the data to a function, such as with regression. *Linear regression* involves finding the "best" line to fit two variables, so that one variable can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two variables are involved and the data are fit to a multidimensional surface. Using regression to find a mathematical equation to fit the data helps smooth out the noise. Regression is further described in Section 3.4.4, as well as in Chapter 7.

Many methods for data smoothing are also methods of data reduction involving discretization. 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 which can also be used for data smoothing. A concept hierarchy for *price*, for example, may map *price* real 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 Chapter 7.

3.2.3 Inconsistent data

There may be inconsistencies in the data recorded for some transactions. 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. This may be coupled with routines designed to help correct the inconsistent use of codes. Knowledge engineering tools may also be used to detect the violation of known data constraints. For example, known functional dependencies between attributes can be used to find values contradicting the functional constraints.

There may also be inconsistencies due to data integration, where a given attribute can have different names in different databases. Redundancies may also result. Data integration and the removal of redundant data are described in Section 3.3.1.

3.3 Data integration and transformation

3.3.1 Data integration

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* can be tricky. How can like 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 entity? Databases and data warehouses typically have metadata - that is, data about the data. Such metadata can be used to help avoid errors in schema integration.

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

Some redundancies can be detected by **correlation analysis**. For example, given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. The correlation between attributes A and B can be measured by

$$\frac{P(A \wedge B)}{P(A)P(B)}. \tag{3.1}$$

If the resulting value of Equation (3.1) is greater than 1, then A and B are positively correlated. The higher the value, the more each attribute implies the other. Hence, a high value may indicate that A (or B) may be removed as a redundancy. If the resulting value is equal to 1, then A and B are independent and there is no correlation between them. If the resulting value is less than 1, then A and B are negatively correlated. This means that each attribute discourages the other. Equation (3.1) may detect a correlation between the *customer_id* and *cust_number* attributes described above. Correlation analysis is further described in Chapter 6 (Section 6.5.2 on mining correlation rules).

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.

A third important issue in data integration is 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. The *price* of different hotels may involve not only different currencies but also different services (such as free breakfast) and taxes. Such semantic heterogeneity of data poses great challenges in data integration.

Careful integration of the data from multiple sources 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.

3.3.2 Data transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Data transformation can involve the following:

1. **Normalization**, where the attribute data are scaled so as to fall within a small specified range, such as -1.0 to 1.0, or 0 to 1.0.

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

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 granularities.

4. **Generalization** of the data, where low level or 'primitive' (raw) data are replaced by higher level concepts through the use of concept hierarchies. For example, categorical attributes, like *street*, can be generalized to higher level concepts, like *city* or *county*. Similarly, values for numeric attributes, like *age*, may be mapped to higher level concepts, like *young, middle-aged*, and *senior*.

In this section, we discuss normalization. Smoothing is a form of data cleaning, and was discussed in Section 3.2.2. Aggregation and generalization also serve as forms of data reduction, and are discussed in Sections 3.4 and 3.5, respectively.

An attribute is normalized by scaling its values so that they fall within a small specified range, such as 0 to 1.0. 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 7), normalizing the input values for each attribute measured in the training samples 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). There are many methods for data normalization. We study three: min-max normalization, z-score normalization, and normalization by decimal scaling.

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 of A to v' in the range $[new\_min_A, new\_max_A]$ by computing

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A. \tag{3.2}$$

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 3.1 Suppose that the maximum and minimum values for the attribute *income* are \$98,000 and \$12,000, respectively. We would like to map *income* to the range $[0, 1]$. By min-max normalization, a value of \$73,600 for *income* is transformed to $\frac{73{,}600-12{,}000}{98{,}000-12{,}000}(1-0)+0 = 0.716$. □

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

$$v' = \frac{v - mean_A}{stand\_dev_A} \tag{3.3}$$

where $mean_A$ and $stand\_dev_A$ are the mean and standard deviation, respectively, of attribute A. This method of normalization is useful when the actual minimum and maximum of attribute A are unknown, or when there are outliers which dominate the min-max normalization.

Example 3.2 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 $\frac{73{,}600-54{,}000}{16{,}000} = 1.225$. □

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 of A is normalized to v' by computing

$$v' = \frac{v}{10^j}, \tag{3.4}$$

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

Example 3.3 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. □

Note that normalization can change the original data quite a bit, especially the latter two of the methods shown above. 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.

3.4 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 may take a very long time, making such analysis impractical or infeasible. Is there any way to "reduce" the size of the data set without jeopardizing the data mining results?

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

Strategies for data reduction include the following.

1. **Data cube aggregation**, where aggregation operations are applied to the data in the construction of a data cube.
2. **Dimension reduction**, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.
3. **Data compression**, where encoding mechanisms are used to reduce the data set size.
4. **Numerosity reduction**, where the data are replaced or estimated by alternative, smaller data representations such as parametric models (which need store only the model parameters instead of the actual data), or non-parametric methods such as clustering, sampling, and the use of histograms.
5. **Discretization and concept hierarchy generation**, where raw data values for attributes are replaced by ranges or higher conceptual levels. Concept hierarchies allow the mining of data at multiple levels of abstraction, and are a powerful tool for data mining. We therefore defer the discussion of automatic concept hierarchy generation to Section 3.5 which is devoted entirely to this topic.

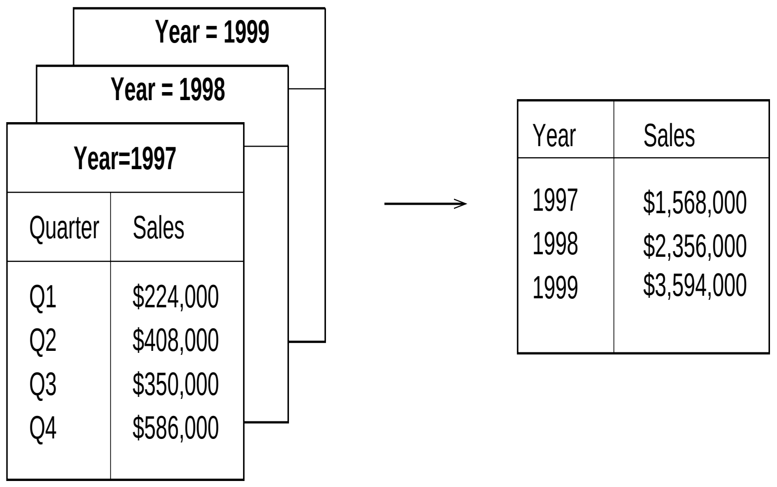
Strategies 1 to 4 above are discussed in the remainder of this section. The computational time spent on data reduction should not outweight or "erase" the time saved by mining on a reduced data set size.

3.4.1 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 1997 to 1999. 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 3.4. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

Data cubes were discussed in Chapter 2. For completeness, we briefely review some of that material here. Data cubes store multidimensional aggregated information. For example, Figure 3.5 shows a data cube for multidimensional analysis of sales data with respect to annual sales per item type for each *AllElectronics* branch. Each cells holds an aggregate data value, corresponding to the data point in multidimensional space. 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 on-line analytical processing as well as data mining.

The cube created at the lowest level of abstraction is referred to as the base cuboid. A cube for the highest level of abstraction is the apex cuboid. For the sales data of Figure 3.5, 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 sometimes 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.



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

Year	Sales
1997	$1,568,000
1998	$2,356,000
1999	$3,594,000

Figure 3.4: Sales data for a given branch of *AllElectronics* for the years 1997 to 1999. In the data on the left, the sales are shown per quarter. In the data on the right, the data are aggregated to provide the *annual_sales*.

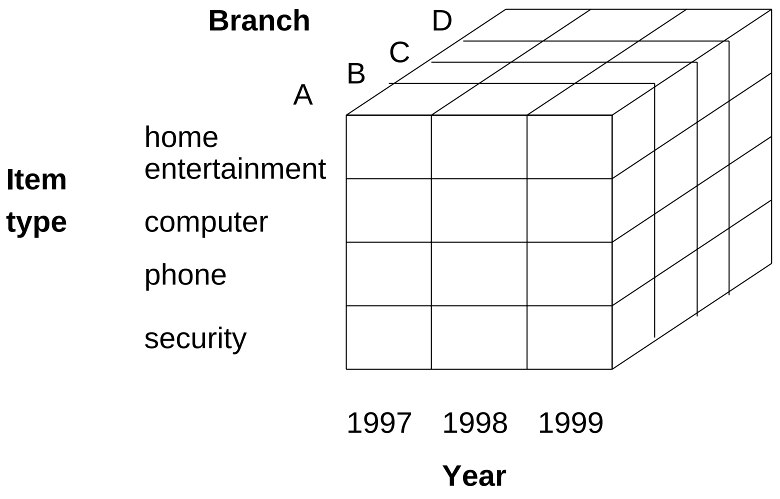


Figure 3.5: A data cube for sales at *AllElectronics*.

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. Since data cubes provide fast accessing to precomputed, summarized data, they should be used when possible to reply to queries regarding aggregated information. When replying to such OLAP queries or data mining requests, the *smallest* available cuboid relevant to the given task should be used. This issue is also addressed in Chapter 2.

3.4.2 Dimensionality reduction

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 as to whether or not they are likely to purchase a popular new CD at *AllElectronics* when notified of a sale, 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.

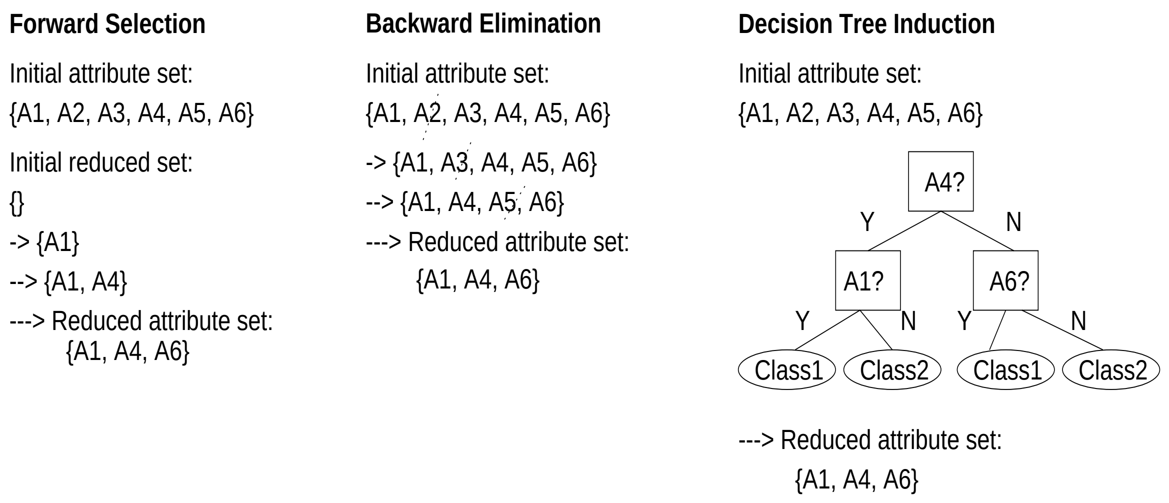


Figure 3.6: Greedy (heuristic) methods for attribute subset selection.

Dimensionality reduction reduces the data set size by removing such attributes (or dimensions) from it. Typically, methods of attribute subset selection are applied. 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 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?" There are 2^d possible subsets of d attributes. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as d and the number of data classes increase. Therefore, heuristic methods which 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 selected 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[1].

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

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

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

3. **Combination forward selection and backward elimination**: The step-wise forward selection and backward elimination methods can be combined, where at each step one selects the best attribute and removes the worst from among the remaining attributes.

The stopping criteria for methods 1 to 3 may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

4. **Decision tree induction**: Decision tree algorithms, such as ID3 and C4.5, were originally intended for classification. Decision tree induction constructs a flow-chart-like structure where each internal (non-leaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf)

[1]The information gain measure is described in Chapters 5 and 7.

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. This method of attribute selection is visited again in greater detail in Chapter 5 on concept description.

3.4.3 Data compression

In *data compression*, data encoding or 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 compression technique used is called *lossless*. If, instead, we can reconstruct only an approximation of the original data, then the data compression technique is called *lossy*. There are several well-tuned algorithms for string compression. Although they are typically lossless, they allow only limited manipulation of the data. In this section, we instead focus on two popular and effective methods of lossy data compression: *wavelet transforms*, and *principal components analysis*.

Wavelet transforms

The **discrete wavelet transform (DWT)** is a linear signal processing technique that, when applied to a data vector D, transforms it to a numerically different vector, D', of **wavelet coefficients**. The two vectors are of the same length.

"Hmmm", you wonder. "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. The remaining 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 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 are 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. Unlike DFT, wavelets are quite localized in space, contributing to the conservation of local detail.

There is only one DFT, yet there are several DWTs. The general algorithm for a discrete wavelet transform is as follows.

1. The length, L, of the input data vector must be an integer power of two. This condition can be met by padding the data vector with zeros, as necessary.

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.

3. The two functions are applied to pairs of the input data, resulting in two sets of data of length $L/2$. In general, these respectively represent a smoothed version of the input data, and the high-frequency content of it.

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 desired length.

5. A selection of 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. For example, given an input vector of length 4 (represented as the column vector $[x_0, x_1, x_2, x_3]$), the 4-point **Haar**

transform of the vector can be obtained by the following matrix multiplication:

$$\begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.5}$$

The matrix on the left is *orthonormal*, meaning that the columns are unit vectors (multiplied by a constant) 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. Other popular wavelet transforms include the Daubechies-4 and the Daubechies-6 transforms.

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 data with ordered attributes.

Principal components analysis

Herein, we provide an intuitive introduction to principal components analysis as a method of data compression. A detailed theoretical explanation is beyond the scope of this book.

Suppose that the data to be compressed consists of N tuples or data vectors, from k-dimensions. **Principal components analysis (PCA)** searches for c k-dimensional orthogonal vectors that can best be used to represent the data, where $c << N$. The original data is thus projected onto a much smaller space, resulting in data compression. PCA can be used as a form of dimensionality reduction. However, unlike attribute subset selection, 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.

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 N orthonormal vectors which 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. This information helps identify groups or patterns within the data.
4. Since the components are sorted according to decreasing order of "significance", the size of the data can be reduced by eliminating the weaker components, i.e., 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. For example, a 3-D data cube for sales with the dimensions *item_type, branch*, and *year* must first be reduced to a 2-D cube, such as with the dimensions *item_type*, and *branch* $\times$ *year*.

3.4.4 Numerosity reduction

"Can we reduce the data volume by choosing alternative, 'smaller' forms of data representation?" Techniques of *numerosity reduction* can indeed be applied for this purpose. These techniques may be parametric or non-parametric. For *parametric methods*, a model is used to estimate the data, so that typically only the data parameters need be stored, instead of the actual data. (Outliers may also be stored). Log-linear models, which estimate discrete multidimensional probability distributions, are an example. *Non-parametric methods* for storing reduced representations of the data include histograms, clustering, and sampling.

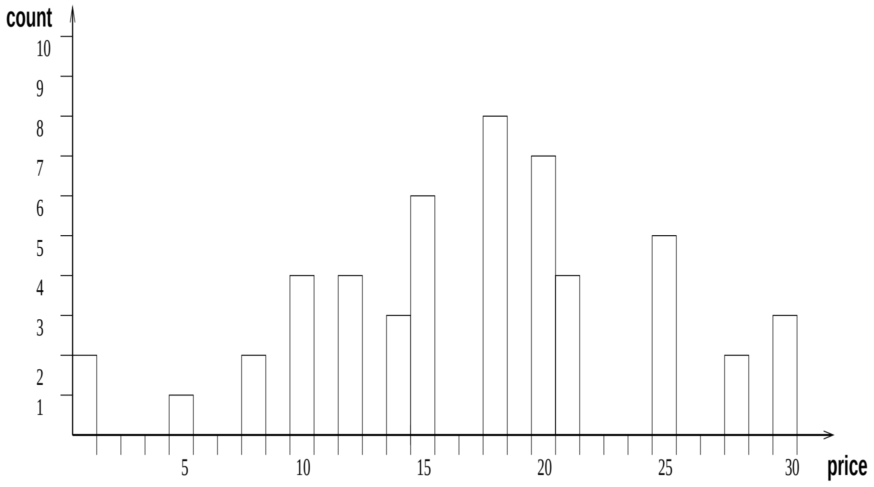


Figure 3.7: A histogram for *price* using singleton buckets - each bucket represents one price-value/frequency pair.

Let's have a look at each of the numerosity reduction techniques mentioned above.

Regression and log-linear models

Regression and log-linear models can be used to approximate the given data. In **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

$$Y = \alpha + \beta X, \tag{3.6}$$

where the variance of Y is assumed to be constant. The coefficients α and β (called *regression coefficients*) specify the Y-intercept and slope of the line, 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 regression** is an extension of linear regression allowing a response variable Y to be modeled as a linear function of a multidimensional feature vector.

Log-linear models approximate discrete multidimensional probability distributions. The method can be used to estimate the probability of each cell in a base cuboid for a set of discretized attributes, based on the smaller cuboids making up the data cube lattice. This allows higher order data cubes to be constructed from lower order ones. Log-linear models are therefore also useful for data compression (since the smaller order cuboids together typically occupy less space than the base cuboid) and data smoothing (since cell estimates in the smaller order cuboids are less subject to sampling variations than cell estimates in the base cuboid). Regression and log-linear models are further discussed in Chapter 7 (Section 7.8 on Prediction).

Histograms

Histograms use binning to approximate data distributions and are a popular form of data reduction. A **histogram** for an attribute A partitions the data distribution of A into disjoint subsets, or *buckets*. The buckets are displayed on a horizontal axis, while the height (and area) of a bucket typically reflects the average frequency of the values represented by the bucket. 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 3.4 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.

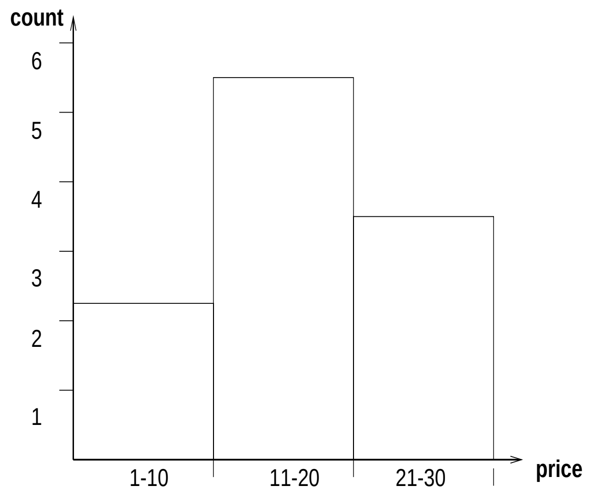


Figure 3.8: A histogram for *price* where values are aggregated so that each bucket has a uniform width of \$10.

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.

1. **Equi-width**: In an equi-width histogram, the width of each bucket range is constant (such as the width of \$10 for the buckets in Figure 3.8).
2. **Equi-depth** (or equi-height): In an equi-depth 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).
3. **V-Optimal**: If we consider all of the possible histograms for a given number of buckets, the V-optimal histogram is the one with the least variance. Histogram variance is a weighted sum of the original values that each bucket represents, where bucket weight is equal to the number of values in the bucket.
4. **MaxDiff**: In a MaxDiff histogram, we consider the difference between each pair of adjacent values. A bucket boundary is established between each pair for pairs having the $\beta - 1$ largest differences, where β is user-specified.

V-Optimal and MaxDiff histograms tend to be the most accurate and practical. 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 very high dimensions. Singleton buckets are useful for storing outliers with high frequency. Histograms are further described in Chapter 5 (Section 5.6 on mining descriptive statistical measures in large databases).

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 of each cluster object from the cluster centroid (denoting the "average object", or average point in space for the cluster). Figure 3.9 shows

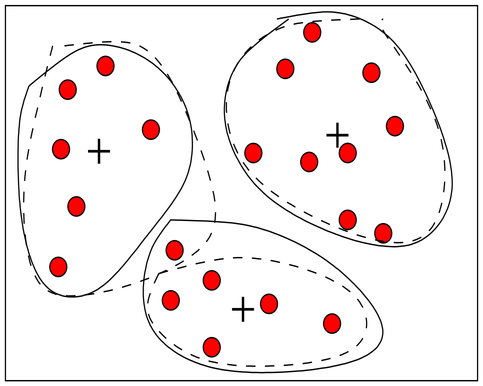


Figure 3.9: A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster centroid is marked with a "+".

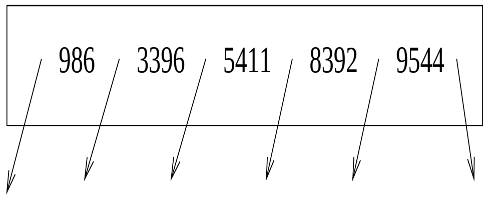


Figure 3.10: The root of a B+-tree for a given set of data.

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.

In database systems, **multidimensional index trees** are primarily used for providing fast data access. They can also be used for hierarchical data reduction, providing a multiresolution clustering of the data. This can be used to provide approximate answers to queries. An index tree recursively partitions the multidimensional space for a given set of data objects, with the root node representing the entire space. Such trees are typically balanced, consisting of internal and leaf nodes. Each parent node contains keys and pointers to child nodes that, collectively, represent the space represented by the parent node. Each leaf node contains pointers to the data tuples they represent (or to the actual tuples).

An index tree can therefore store aggregate and detail data at varying levels of resolution or abstraction. It provides a hierarchy of clusterings of the data set, where each cluster has a label that holds for the data contained in the cluster. If we consider each child of a parent node as a bucket, then an index tree can be considered as a *hierarchical histogram*. For example, consider the root of a B+-tree as shown in Figure 3.10, with pointers to the data keys 986, 3396, 5411, 8392, and 9544. Suppose that the tree contains 10,000 tuples with keys ranging from 1 to 9,999. The data in the tree can be approximated by an equi-depth histogram of 6 buckets for the key ranges 1 to 985, 986 to 3395, 3396 to 5410, 5411 to 8392, 8392 to 9543, and 9544 to 9999. Each bucket contains roughly 10,000/6 items. Similarly, each bucket is subdivided into smaller buckets, allowing for aggregate data at a finer-detailed level. The use of multidimensional index trees as a form of data resolution relies on an ordering of the attribute values in each dimension. Multidimensional index trees include R-trees, quad-trees, and their variations. They are well-suited for handling both sparse and skewed data.

There are many measures for defining clusters and cluster quality. Clustering methods are further described in Chapter 8.

Sampling

Sampling can be used as a data reduction technique since 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 have a look at some possible samples for D.

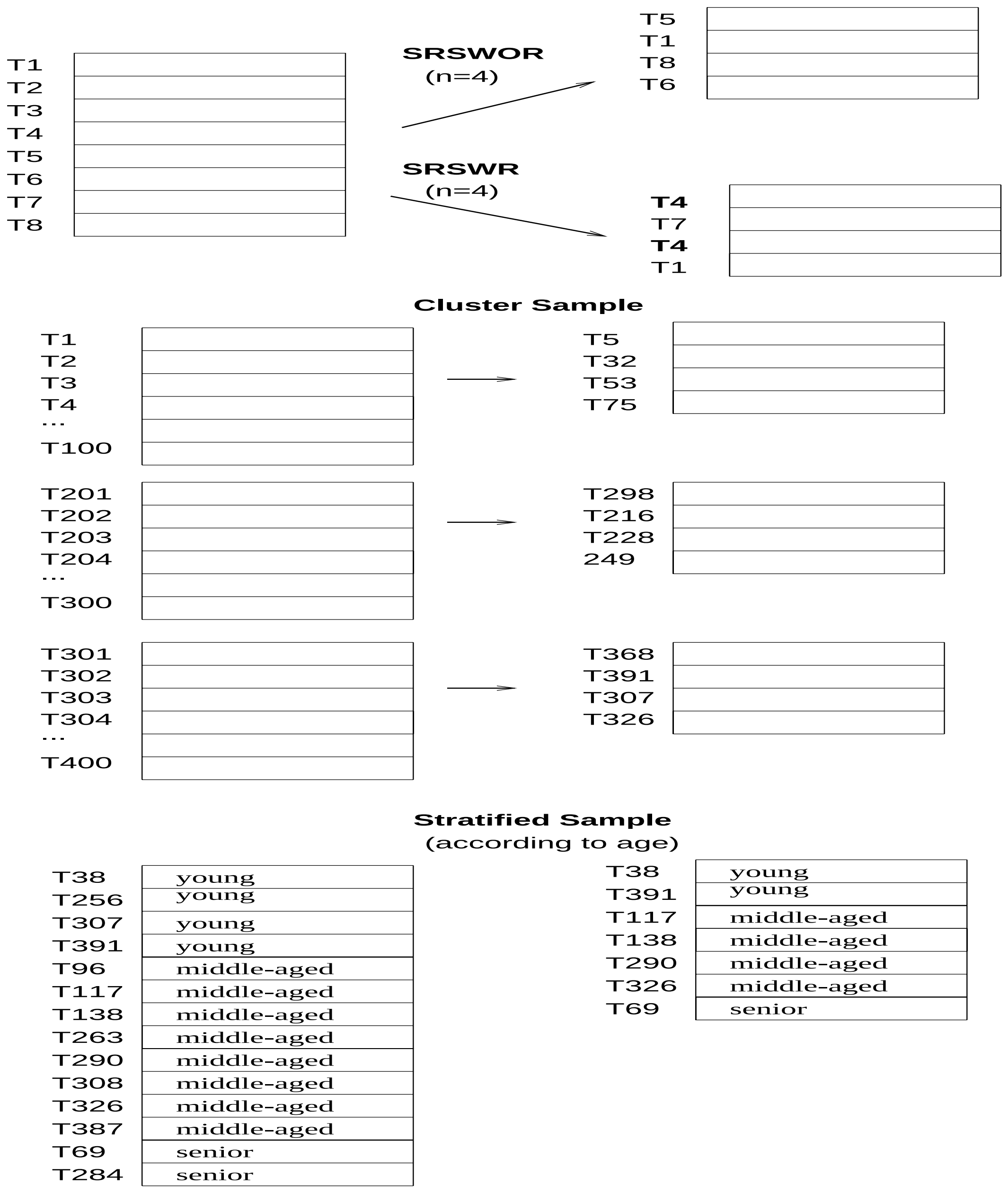


Figure 3.11: Sampling can be used for data reduction.

1. **Simple random sample without replacement (SRSWOR)** of size n: This is created by drawing n of the N tuples from D ($n < N$), where the probably of drawing any tuple in D is $1/N$, i.e., all tuples are equally likely.

2. **Simple random sample with replacement (SRSWR) of size** n: 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.

3. **Cluster sample**: If the tuples in D are grouped into M mutually disjoint "clusters", then a SRS of m clusters can be obtained, where $m < M$. 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.

4. **Stratified sample**: If D is divided into mutually disjoint parts called "strata", a stratified sample of D is generated by obtaining a SRS at each stratum. This helps to ensure a representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where 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.

These samples are illustrated in Figure 3.11. They represent the most commonly used forms of sampling for data reduction.

An advantage of sampling for data reduction is that the cost of obtaining a sample *is proportional to the size of the sample*, n, as opposed to N, the data set size. Hence, sampling complexity is potentially *sub-linear* 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, d, increases, while techniques using histograms, for example, increase exponentially in d.

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, n, 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.

3.5 Discretization and concept hierarchy generation

Discretization techniques can be used to reduce the number of values for a given continuous attribute, by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values. Reducing the number of values for an attribute is especially beneficial if decision tree-based methods of classification mining are to be applied to the preprocessed data. These methods are typically recursive, where a large amount of time is spent on sorting the data at each step. Hence, the smaller the number of distinct values to sort, the faster these methods should be. Many discretization techniques can be applied recursively in order to provide a hierarchical, or multiresolution partitioning of the attribute values, known as a concept hierarchy. Concept hierarchies were introduced in Chapter 2. They are useful for mining at multiple levels of abstraction.

A concept hierarchy for a given numeric attribute defines a discretization of the attribute. Concept hierarchies can be used to reduce the data by collecting and replacing low level concepts (such as numeric values for the attribute *age*) by higher level concepts (such as *young, middle-aged*, or *senior*). Although detail is lost by such data generalization, the generalized data may be more meaningful and easier to interpret, and will require less space than the original data. Mining on a reduced data set will require fewer input/output operations and be more efficient than mining on a larger, ungeneralized data set. An example of a concept hierarchy for the attribute *price* is given in Figure 3.12. More than one concept hierarchy can be defined for the same attribute in order to accommodate the needs of the various users.

Manual definition of concept hierarchies can be a tedious and time-consuming task for the user or domain expert. Fortunately, many hierarchies are implicit within the database schema, and can be defined at the schema definition level. Concept hierarchies often can be automatically generated or dynamically refined based on statistical analysis of the data distribution.

Let's look at the generation of concept hierarchies for numeric and categorical data.

3.5.1 Discretization and concept hierarchy generation for numeric data

It is difficult and tedious to specify concept hierarchies for numeric attributes due to the wide diversity of possible data ranges and the frequent updates of data values.

Concept hierarchies for numeric attributes can be constructed automatically based on data distribution analysis. We examine five methods for numeric concept hierarchy generation. These include *binning*, *histogram analysis*,

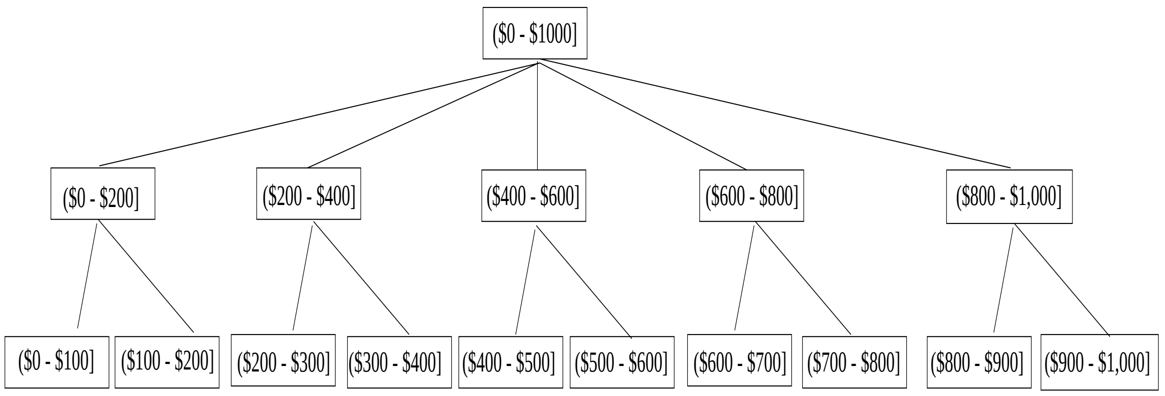


Figure 3.12: A concept hierarchy for the attribute *price*.

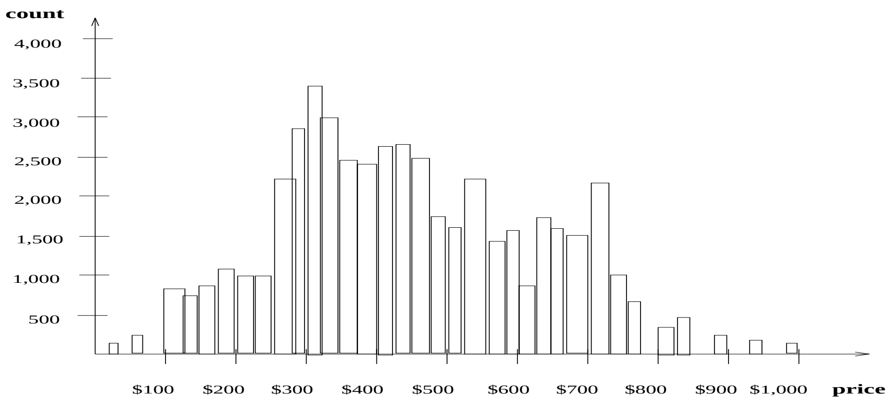


Figure 3.13: Histogram showing the distribution of values for the attribute *price*.

clustering analysis, *entropy-based discretization*, and *data segmentation by "natural partitioning"*.

1. **Binning.**

   Section 3.2.2 discussed binning methods for data smoothing. These methods are also forms of discretization. For example, attribute values can be discretized by 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.

2. **Histogram analysis.**

   Histograms, as discussed in Section 3.4.4, can also be used for discretization. Figure 3.13 presents a **histogram** showing the data distribution of the attribute *price* for a given data set. For example, the most frequent price range is roughly $300-$325. Partitioning rules can be used to define the ranges of values. For instance, in an *equi-width* histogram, the values are partitioned into equal sized partions or ranges (e.g., ($0-$100], ($100-$200], ..., ($900-$1,000]). With an *equi-depth* histogram, the values are partitioned so that, ideally, each partition contains the same number of data samples. 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 pre-specified 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. A concept hierarchy for *price*, generated from the data of Figure 3.13 is shown in Figure 3.12.

3. **Clustering analysis.**

   A clustering algorithm can be applied to partition data into clusters or groups. Each cluster forms a node of a concept hierarchy, where all nodes are at the same conceptual level. Each cluster may be further decomposed into several subclusters, forming a lower level of the hierarchy. Clusters may also be grouped together in order to form a higher conceptual level of the hierarchy. Clustering methods for data mining are studied in Chapter 8.

4. **Entropy-based discretization.**

   An information-based measure called "entropy" can be used to recursively partition the values of a numeric attribute A, resulting in a hierarchical discretization. Such a discretization forms a numerical concept hierarchy for the attribute. Given a set of data tuples, S, the basic method for entropy-based discretization of A is as follows.

   - Each value of A can be considered a potential interval boundary or threshold T. For example, a value v of A can partition the samples in S into two subsets satisfying the conditions $A < v$ and $A \geq v$, respectively, thereby creating a binary discretization.
   - Given S, the threshold value selected is the one that maximizes the information gain resulting from the subsequent partitioning. The information gain is:

     $$I(S,T) = \frac{|S_1|}{|S|}Ent(S_1) + \frac{|S_2|}{|S|}Ent(S_2), \tag{3.7}$$

     where S_1 and S_2 correspond to the samples in S satisfying the conditions $A < T$ and $A \geq T$, respectively. The entropy function Ent for a given set is calculated based on the class distribution of the samples in the set. For example, given m classes, the entropy of S_1 is:

     $$Ent(S_1) = -\sum_{i=1}^{m} p_i log_2(p_i), \tag{3.8}$$

     where p_i is the probability of class i in S_1, determined by dividing the number of samples of class i in S_1 by the total number of samples in S_1. The value of $Ent(S_2)$ can be computed similarly.
   - The process of determining a threshold value is recursively applied to each partition obtained, until some stopping criterion is met, such as

     $$Ent(S) - I(S,T) > \delta \tag{3.9}$$

   Experiments show that entropy-based discretization can reduce data size and may improve classification accuracy. The information gain and entropy measures described here are also used for decision tree induction. These measures are revisited in greater detail in Chapter 5 (Section 5.4 on analytical characterization) and Chapter 7 (Section 7.3 on decision tree induction).

5. **Segmentation by natural partitioning**.

   Although binning, histogram analysis, clustering and entropy-based discretization are useful in the generation of numerical hierarchies, many users would like to see numerical ranges partitioned into relatively uniform, easy-to-read intervals that appear intuitive or "natural". For example, annual salaries broken into ranges like [\$50,000, \$60,000) are often more desirable than ranges like [\$51263.98, \$60872.34), obtained by some sophisticated clustering analysis.

   The **3-4-5 rule** can be used to segment numeric data into relatively uniform, "natural" intervals. In general, the rule partitions a given range of data into either 3, 4, or 5 relatively equi-length intervals, recursively and level by level, based on the value range at the most significant digit. The rule is as follows.

   (a) If an interval covers 3, 6, 7 or 9 distinct values at the most significant digit, then partition the range into 3 intervals (3 equi-width intervals for 3, 6, 9, and three intervals in the grouping of 2-3-2 for 7);

   (b) if it covers 2, 4, or 8 distinct values at the most significant digit, then partition the range into 4 equi-width intervals; and

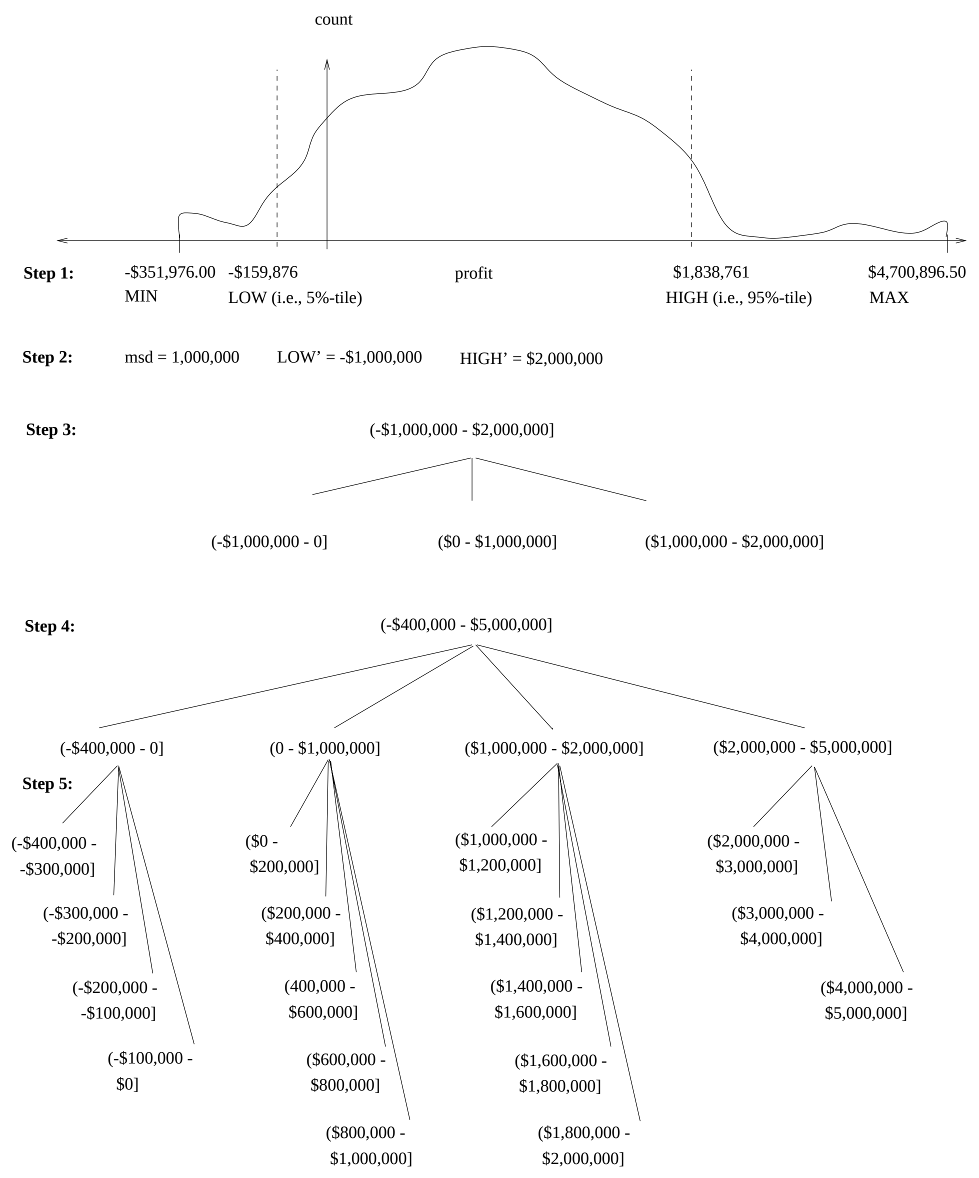


Figure 3.14: Automatic generation of a concept hierarchy for *profit* based on the 3-4-5 rule.

(c) if it covers 1, 5, or 10 distinct values at the most significant digit, then partition the range into 5 equi-width intervals.

The rule can be recursively applied to each interval, creating a concept hierarchy for the given numeric attribute. Since there could be some dramatically large positive or negative values in a data set, the top level segmentation, based merely on the minimum and maximum values, may derive distorted results. For example, the assets of a few people could be several orders of magnitude higher than those of others in a data set. Segmentation based on the maximal asset values may lead to a highly biased hierarchy. Thus the top level segmentation can be performed based on the range of data values representing the majority (e.g., 5%-tile to 95%-tile) of the given data. The extremely high or low values beyond the top level segmentation will form distinct interval(s) which can be handled separately, but in a similar manner.

The following example illustrates the use of the 3-4-5 rule for the automatic construction of a numeric hierarchy.

Example 3.5 Suppose that profits at different branches of *AllElectronics* for the year 1997 cover a wide range, from $-$351,976.00 to $4,700,896.50. A user wishes to have a concept hierarchy for *profit* automatically generated. For improved readability, we use the notation $(l — r]$ to represent the interval $(l, r]$. For example, (−\$1,000,000 — \$0] denotes the range from −\$1,000,000 (exclusive) to \$0 (inclusive).

Suppose that the data within the 5%-tile and 95%-tile are between −\$159,876 and \$1,838,761. The results of applying the 3-4-5 rule are shown in Figure 3.14.

- *Step 1:* Based on the above information, the minimum and maximum values are: $MIN = -\$351,976.00$, and $MAX = \$4,700,896.50$. The low (5%-tile) and high (95%-tile) values to be considered for the top or first level of segmentation are: $LOW = -\$159,876$, and $HIGH = \$1,838,761$.
- *Step 2:* Given LOW and HIGH, the most significant digit is at the million dollar digit position (i.e., $msd =$ 1,000,000). Rounding LOW down to the million dollar digit, we get $LOW' = -\$1,000,000$; and rounding $HIGH$ up to the million dollar digit, we get $HIGH' = +\$2,000,000$.
- *Step 3:* Since this interval ranges over 3 distinct values at the most significant digit, i.e., $(2,000,000 - (-1,000,000))/1,000,000 = 3$, the segment is partitioned into 3 equi-width subsegments according to the 3-4-5 rule: (−\$1,000,000 — \$0], (\$0 — \$1,000,000], and (\$1,000,000 — \$2,000,000]. This represents the top tier of the hierarchy.
- *Step 4:* We now examine the MIN and MAX values to see how they "fit" into the first level partitions.

  Since the first interval, $(-\$1,000,000$ — $\$0]$ covers the MIN value, i.e., $LOW' < MIN$, we can adjust the left boundary of this interval to make the interval smaller. The most significant digit of MIN is the hundred thousand digit position. Rounding MIN down to this position, we get $MIN' = -\$400,000$. Therefore, the first interval is redefined as $(-\$400,000$ — $0]$.

  Since the last interval, (\$1,000,000 — \$2,000,000] does not cover the MAX value, i.e., $MAX > HIGH'$, we need to create a new interval to cover it. Rounding up MAX at its most significant digit position, the new interval is (\$2,000,000 — \$5,000,000]. Hence, the top most level of the hierarchy contains four partitions, (−\$400,000 — \$0], (\$0 — \$1,000,000], (\$1,000,000 — \$2,000,000], and (\$2,000,000 — \$5,000,000].
- *Step 5:* Recursively, each interval can be further partitioned according to the 3-4-5 rule to form the next lower level of the hierarchy:
  - The first interval (−\$400,000 — \$0] is partitioned into 4 sub-intervals: (−\$400,000 — −\$300,000], (−\$300,000 — −\$200,000], (−\$200,000 — −\$100,000], and (−\$100,000 — \$0].
  - The second interval, (\$0 — \$1,000,000], is partitioned into 5 sub-intervals: (\$0 — \$200,000], (\$200,000 — \$400,000], (\$400,000 — \$600,000], (\$600,000 — \$800,000], and (\$800,000 — \$1,000,000].
  - The third interval, (\$1,000,000 — \$2,000,000], is partitioned into 5 sub-intervals: (\$1,000,000 — \$1,200,000], (\$1,200,000 — \$1,400,000], (\$1,400,000 — \$1,600,000], (\$1,600,000 — \$1,800,000], and (\$1,800,000 — \$2,000,000].
  - The last interval, (\$2,000,000 — \$5,000,000], is partitioned into 3 sub-intervals: (\$2,000,000 — \$3,000,000], (\$3,000,000 — \$4,000,000], and (\$4,000,000 — \$5,000,000].

  Similarly, the 3-4-5 rule can be carried on iteratively at deeper levels, as necessary. □

3.5.2 Concept hierarchy generation for categorical data

Categorical data are discrete data. Categorical 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*. There are several methods for the generation of concept hierarchies for categorical data.

1. **Specification of a partial ordering of attributes explicitly at the schema level by users or experts.**

   Concept hierarchies for categorical 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, a relational database or a dimension *location* of a data warehouse may contain the following group of attributes: *street, city, province_or_state*, and *country*. 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*.

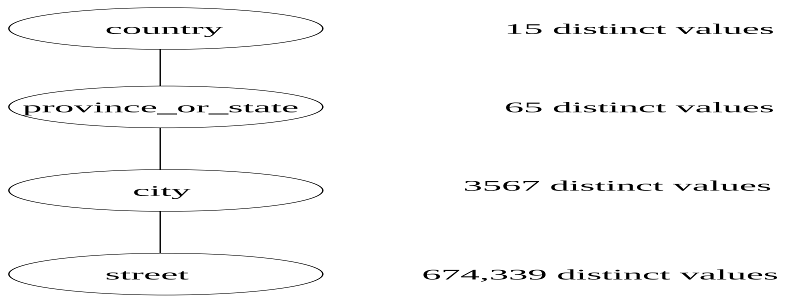


Figure 3.15: Automatic generation of a schema concept hierarchy based on the number of distinct attribute values.

2. **Specification of a portion of a hierarchy by explicit data grouping.**

   This is essentially the manual definition of a portion of concept hierarchy. In a large database, it is unrealistic to define an entire concept hierarchy by explicit value enumeration. However, it is realistic to 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, one may like to add some intermediate levels manually, such as defining explicitly, "$\{$*Alberta, Saskatchewan, Manitoba*$\} \subset$ *prairies_Canada*", and "$\{$*British Columbia, prairies_Canada*$\}$ $\subset$ *Western_Canada*".

3. **Specification of a** *set of attributes*, **but not of their partial ordering**.

   A user may simply group a set of attributes as a preferred dimension or hierarchy, but may omit stating their partial order explicitly. This may require the system to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy. Without knowledge of data semantics, it is difficult to provide an ideal hierarchical ordering for an arbitrary set of attributes. However, an important observation is that since higher level concepts generally cover several subordinate lower level concepts, an attribute defining a high concept level will usually contain a smaller number of distinct values than an attribute defining a lower concept level. 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 lesser the number of distinct values an attribute has, the higher it is in the generated concept hierarchy. This heuristic rule works fine in many cases. Some local level swapping or adjustments may be performed by users or experts, when necessary, after examination of the generated hierarchy.

   Let's examine an example of this method.

   Example 3.6 Suppose a user selects a set of attributes, *street, country, province_or_state*, and *city*, for a dimension *location* from the database *AllElectronics*, but does not specify the hierarchical ordering among the attributes.

   The concept hierarchy for *location* can be generated automatically as follows. 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* (*65*), *city* (*3567*), and *street* (*674,339*). Second, generate the hierarchy from top down according to the sorted order, with the first attribute at the top-level and the last attribute at the bottom-level. The resulting hierarchy is shown in Figure 3.15. Finally, the user examines the generated hierarchy, and when necessary, modifies it to reflect desired semantic relationship among the attributes. In this example, it is obvious that there is no need to modify the generated hierarchy. □

   Note that this heristic rule cannot be pushed to the extreme since there are obvious cases which do not follow such a heuristic. 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 sloppy when defining a hierarchy, or may 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 a hierarchy specification. For example, instead of including all the hierarchically relevant attributes for *location*, one may specify 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 over-ride this feature, as necessary.

Example 3.7 Suppose that a database system 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 may automatically drag 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. □

3.6 Summary

- **Data preparation** is an important issue for both data warehousing and data mining, as real-world data tends to be incomplete, noisy, and inconsistent. Data preparation includes data cleaning, data integration, data transformation, and data reduction.
- **Data cleaning** routines can be used to fill in missing values, smooth noisy data, identify outliers, and correct data inconsistencies.
- **Data integration** combines data from multiples sources to form a coherent data store. Metadata, correlation analysis, data conflict detection, and the resolution of semantic heterogeneity contribute towards smooth data integration.
- **Data transformation** routines conform the data into appropriate forms for mining. For example, attribute data may be **normalized** so as to fall between a small range, such as 0 to 1.0.
- **Data reduction** techniques such as data cube aggregation, dimension reduction, data compression, numerosity reduction, and discretization can be used to obtain a reduced representation of the data, while minimizing the loss of information content.
- **Concept hierarchies** organize the values of attributes or dimensions into gradual levels of abstraction. They are a form a discretization that is particularly useful in multilevel mining.
- **Automatic generation of concept hierarchies** for categoric data may be based on the number of distinct values of the attributes defining the hierarchy. For numeric data, techniques such as data segmentation by partition rules, histogram analysis, and clustering analysis can be used.
- Although several methods of data preparation have been developed, data preparation remains an active area of research.

Exercises

1. Data quality can be assessed in terms of accuracy, completeness, and consistency. 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. Suppose that the data for analysis includes the attribute *age*. The *age* values for the data tuples are (in increasing order):

   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. Using the data for *age* given in Question 3, answer the following:
   (a) Use min-max normalization to transform the value 35 for *age* onto the range $[0, 1]$.
   (b) Use z-score normalization to transform the value 35 for *age*, where the standard deviation of *age* is ??.
   (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.

6. Use a flow-chart to illustrate the following procedures for attribute subset selection:
   (a) step-wise forward selection.
   (b) step-wise backward elimination
   (c) a combination of forward selection and backward elimination.

7. Using the data for *age* given in Question 3:
   (a) Plot an equi-width histogram of width 10.
   (b) Sketch examples of each of the following sample techniques: SRSWOR, SRSWR, cluster sampling, stratified sampling.

8. Propose a concept hierarchy for the attribute *age* using the 3-4-5 partition rule.

9. Propose an algorithm, in pseudo-code or in your favorite programming language, for
   (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 numeric data based on the *equi-width* partitioning rule, and
   (c) the automatic generation of a concept hierarchy for numeric data based on the *equi-depth* partitioning rule.

Bibliographic Notes

Data preprocessing is discussed in a number of textbooks, including Pyle [28], Kennedy et al. [21], and Weiss and Indurkhya [37]. More specific references to individual preprocessing techniques are given below.

For discussion regarding data quality, see Ballou and Tayi [3], Redman [31], Wand and Wang [35], and Wang, Storey and Firth [36]. The handling of missing attribute values is discussed in Quinlan [29], Breiman et al. [5], and Friedman [11]. A method for the detection of outlier or "garbage" patterns in a handwritten character database is given in Guyon, Matic, and Vapnik [14]. Binning and data normalization are treated in several texts, including [28, 21, 37].

A good survey of data reduction techniques can be found in Barbará et al. [4]. For algorithms on data cubes and their precomputation, see [33, 16, 1, 38, 32]. Greedy methods for attribute subset selection (or *feature subset selection*) are described in several texts, such as Neter et al. [24], and John [18]. A combination forward selection and backward elimination method was proposed in Siedlecki and Sklansky [34]. For a description of wavelets for data compression, see Press et al. [27]. Daubechies transforms are described in Daubechies [6]. The book by Press et al. [27] also contains an introduction to singular value decomposition for principal components analysis.

An introduction to regression and log-linear models can be found in several textbooks, such as [17, 9, 20, 8, 24]. For log-linear models (known as *multiplicative models* in the computer science literature), see Pearl [25]. For a general introduction to histograms, see [7, 4]. For extensions of single attribute histograms to multiple attributes, see Muralikrishna and DeWitt [23], and Poosala and Ioannidis [26]. Several references to clustering algorithms are given in Chapter 7 of this book, which is devoted to this topic. A survey of multidimensional indexing structures is in Gaede and Günther [12]. The use of multidimensional index trees for data aggregation is discussed in Aoki [2]. Index trees include R-trees (Guttman [13]), quad-trees (Finkel and Bentley [10]), and their variations. For discussion on sampling and data mining, see John and Langley [19], and Kivinen and Mannila [22].

Entropy and information gain are described in Quinlan [30]. Concept hierarchies, and their automatic generation from categorical data are described in Han and Fu [15].

Bibliography

[1] 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*, pages 506–521, Bombay, India, Sept. 1996.

[2] P. M. Aoki. Generalizing "search" in generalized search trees. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, April 1998.

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

[4] D. Barbará et al. The new jersey data reduction report. *Bulletin of the Technical Committee on Data Engineering*, 20:3–45, December 1997.

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

[6] I. Daubechies. *Ten Lectures on Wavelets.* Capital City Press, Montpelier, Vermont, 1992.

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

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

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

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

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

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

[13] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proc. 1984 ACM-SIGMOD Int. Conf. Management of Data*, June 1984.

[14] 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.

[15] 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.

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

[17] M. James. *Classification Algorithms.* John Wiley, 1985.

[18] G. H. John. *Enhancements to the Data Mining Process.* Ph.D. Thesis, Computer Science Dept., Stanford Univeristy, 1997.

[19] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 367–370, Portland, OR, Aug. 1996.

[20] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[21] R. L Kennedy, Y. Lee, B. Van Roy, C. D. Reed, and R. P. Lippman. *Solving Data Mining Problems Through Pattern Recognition.* Upper Saddle River, NJ: Prentice Hall, 1998.

[22] 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.

[23] 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*, pages 28–36, Chicago, IL, June 1988.

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

[25] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Palo Alto, CA: Morgan Kauffman, 1988.

[26] V. Poosala and Y. Ioannidis. Selectivity estimationwithout the attribute value independence assumption. In *Proc. 23rd Int. Conf. on Very Large Data Bases*, pages 486–495, Athens, Greece, Aug. 1997.

[27] 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, Cambridge, MA, 1996.

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

[29] J. R. Quinlan. Unknown attribute values in induction. In *Proc. 6th Int. Workshop on Machine Learning*, pages 164–168, Ithaca, NY, June 1989.

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

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

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

[33] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 1994 Int. Conf. Data Engineering*, pages 328–336, Feb. 1994.

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

[35] Y. Wand and R. Wang. Anchoring data quality dimensions ontological foundations. *Communications of ACM*, 39:86–95, 1996.

[36] 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.

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

[38] 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*, pages 159–170, Tucson, Arizona, May 1997.