Chapter 2

Data Preparation

"*Success depends upon previous preparation, and without such preparation there is sure to be failure.*"—Confucius

2.1 Introduction

The raw format of real data is usually widely variable. Many values may be missing, inconsistent across different data sources, and erroneous. For the analyst, this leads to numerous challenges in using the data effectively. For example, consider the case of evaluating the interests of consumers from their activity on a social media site. The analyst may first need to determine the types of activity that are valuable to the mining process. The activity might correspond to the interests entered by the user, the comments entered by the user, and the set of friendships of the user along with their interests. All these pieces of information are diverse and need to be collected from different databases within the social media site. Furthermore, some forms of data, such as raw logs, are often not directly usable because of their unstructured nature. In other words, useful features need to be extracted from these data sources. Therefore, a *data preparation phase* is needed.

The data preparation phase is a multistage process that comprises several individual steps, some or all of which may be used in a given application. These steps are as follows:

1. *Feature extraction and portability:* The raw data is often in a form that is not suitable for processing. Examples include raw logs, documents, semistructured data, and possibly other forms of heterogeneous data. In such cases, it may be desirable to derive meaningful features from the data. Generally, features with good semantic interpretability are more desirable because they simplify the ability of the analyst to understand intermediate results. Furthermore, they are usually better tied to the goals of the data mining application at hand. In some cases where the data is obtained from multiple sources, it needs to be integrated into a single database for processing. In addition, some algorithms may work only with a specific data type, whereas the data may contain heterogeneous types. In such cases, *data type portability* becomes

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_2

important where attributes of one type are transformed to another. This results in a more homogeneous data set that can be processed by existing algorithms.

2. *Data cleaning:* In the data cleaning phase, missing, erroneous, and inconsistent entries are removed from the data. In addition, some missing entries may also be estimated by a process known as *imputation*.

3. *Data reduction, selection, and transformation:* In this phase, the size of the data is reduced through data subset selection, feature subset selection, or data transformation. The gains obtained in this phase are twofold. First, when the size of the data is reduced, the algorithms are generally more efficient. Second, if irrelevant features or irrelevant records are removed, the quality of the data mining process is improved. The first goal is achieved by generic sampling and dimensionality reduction techniques. To achieve the second goal, a highly problem-specific approach must be used for feature selection. For example, a feature selection approach that works well for clustering may not work well for classification.

Some forms of feature selection are tightly integrated with the problem at hand. Later chapters on specific problems such as clustering and classification will contain detailed discussions on feature selection.

This chapter is organized as follows. The feature extraction phase is discussed in Sect. 2.2. The data cleaning phase is covered in Sect. 2.3. The data reduction phase is explained in Sect. 2.4. A summary is given in Sect. 2.5.

2.2 Feature Extraction and Portability

The first phase of the data mining process is creating a set of features that the analyst can work with. In cases where the data is in raw and unstructured form (e.g., raw text, sensor signals), the relevant features need to be extracted for processing. In other cases where a heterogeneous mixture of features is available in different forms, an "off-the-shelf" analytical approach is often not available to process such data. In such cases, it may be desirable to transform the data into a uniform representation for processing. This is referred to as *data type porting*.

2.2.1 Feature Extraction

The first phase of feature extraction is a crucial one, though it is very application specific. In some cases, feature extraction is closely related to the concept of data type portability, where low-level features of one type may be transformed to higher-level features of another type. The nature of feature extraction depends on the domain from which the data is drawn:

1. *Sensor data:* Sensor data is often collected as large volumes of low-level signals, which are massive. The low-level signals are sometimes converted to higher-level features using wavelet or Fourier transforms. In other cases, the time series is used directly after some cleaning. The field of signal processing has an extensive literature devoted to such methods. These technologies are also useful for porting time-series data to multidimensional data.

2. *Image data:* In its most primitive form, image data are represented as pixels. At a slightly higher level, color histograms can be used to represent the features in different segments of an image. More recently, the use of *visual words* has become more

popular. This is a semantically rich representation that is similar to document data. One challenge in image processing is that the data are generally very high dimensional. Thus, feature extraction can be performed at different levels, depending on the application at hand.

3. *Web logs:* Web logs are typically represented as text strings in a prespecified format. Because the fields in these logs are clearly specified and separated, it is relatively easy to convert Web access logs into a multidimensional representation of (the relevant) categorical and numeric attributes.

4. *Network traffic:* In many intrusion-detection applications, the characteristics of the network packets are used to analyze intrusions or other interesting activity. Depending on the underlying application, a variety of features may be extracted from these packets, such as the number of bytes transferred, the network protocol used, and so on.

5. *Document data:* Document data is often available in raw and unstructured form, and the data may contain rich linguistic relations between different entities. One approach is to remove stop words, stem the data, and use a bag-of-words representation. Other methods use *entity extraction* to determine linguistic relationships.

   Named-entity recognition is an important subtask of information extraction. This approach locates and classifies atomic elements in text into predefined expressions of names of persons, organizations, locations, actions, numeric quantities, and so on. Clearly, the ability to identify such atomic elements is very useful because they can be used to understand the structure of sentences and complex events. Such an approach can also be used to populate a more conventional database of relational elements or as a sequence of atomic entities, which is more easily analyzed. For example, consider the following sentence:

   ```
   Bill Clinton lives in Chappaqua.
   ```

   Here, "Bill Clinton" is the name of a person, and "Chappaqua" is the name of a place. The word "lives" denotes an action. Each type of entity may have a different significance to the data mining process depending on the application at hand. For example, if a data mining application is mainly concerned with mentions of specific locations, then the word "Chappaqua" needs to be extracted.

   Popular techniques for named entity recognition include linguistic grammar-based techniques and statistical models. The use of grammar rules is typically very effective, but it requires work by experienced computational linguists. On the other hand, statistical models require a significant amount of training data. The techniques designed are very often domain-specific. The area of named entity recognition is vast in its own right, which is outside the scope of this book. The reader is referred to [400] for a detailed discussion of different methods for entity recognition.

Feature extraction is an art form that is highly dependent on the skill of the analyst to choose the features and their representation that are best suited to the task at hand. While this particular aspect of data analysis typically belongs to the domain expert, it is perhaps the most important one. If the correct features are not extracted, the analysis can only be as good as the available data.

2.2.2 Data Type Portability

Data type portability is a crucial element of the data mining process because the data is often heterogeneous, and may contain multiple types. For example, a demographic data set may contain both numeric and mixed attributes. A time-series data set collected from an electrocardiogram (ECG) sensor may have numerous other meta-information and text attributes associated with it. This creates a bewildering situation for an analyst who is now faced with the difficult challenge of designing an algorithm with an arbitrary combination of data types. The mixing of data types also restricts the ability of the analyst to use off-the-shelf tools for processing. Note that porting data types does lose representational accuracy and expressiveness in some cases. Ideally, it is best to customize the algorithm to the particular combination of data types to optimize results. This is, however, time-consuming and sometimes impractical.

This section will describe methods for converting between various data types. Because the numeric data type is the simplest and most widely studied one for data mining algorithms, it is particularly useful to focus on how different data types may be converted to it. However, other forms of conversion are also useful in many scenarios. For example, for similarity-based algorithms, it is possible to convert virtually any data type to a graph and apply graph-based algorithms to this representation. The following discussion, summarized in Table 2.1, will discuss various ways of transforming data across different types.

2.2.2.1 Numeric to Categorical Data: Discretization

The most commonly used conversion is from the numeric to the categorical data type. This process is known as *discretization*. The process of discretization divides the ranges of the numeric attribute into ϕ ranges. Then, the attribute is assumed to contain ϕ different categorical labeled values from 1 to ϕ, depending on the range in which the original attribute lies. For example, consider the age attribute. One could create ranges $[0, 10]$, $[11, 20]$, $[21, 30]$, and so on. The *symbolic* value for any record in the range $[11, 20]$ is "2" and the symbolic value for a record in the range $[21, 30]$ is "3". Because these are symbolic values, no ordering is assumed between the values "2" and "3". Furthermore, variations within a range are not distinguishable after discretization. Thus, the discretization process does lose some information for the mining process. However, for some applications, this loss of information is not too debilitating. One challenge with discretization is that the data may be nonuniformly distributed across the different intervals. For example, for the case of the salary attribute, a large subset of the population may be grouped in the $[40,000,\ 80,000]$ range, but very few will be grouped in the $[1,040,000,\ 1,080,000]$ range. Note that both ranges have the same size. Thus, the use of ranges of equal size may not be very helpful in discriminating between different data segments. On the other hand, many attributes, such as age, are not as nonuniformly distributed, and therefore ranges of equal size may work reasonably well. The discretization process can be performed in a variety of ways depending on application-specific goals:

1. *Equi-width ranges:* In this case, each range $[a, b]$ is chosen in such a way that $b - a$ is the same for each range. This approach has the drawback that it will not work for data sets that are distributed nonuniformly across the different ranges. To determine the actual values of the ranges, the minimum and maximum values of each attribute are determined. This range $[min, max]$ is then divided into ϕ ranges of equal length.

2. *Equi-log ranges:* Each range $[a, b]$ is chosen in such a way that $\log(b) - \log(a)$ has the same value. This kinds of range selection has the effect of geometrically increasing

Table 2.1: Portability of different data types

Source data type	Destination data type	Methods
Numeric	Categorical	Discretization
Categorical	Numeric	Binarization
Text	Numeric	Latent semantic analysis (LSA)
Time series	Discrete sequence	SAX
Time series	Numeric multidimensional	DWT, DFT
Discrete sequence	Numeric multidimensional	DWT, DFT
Spatial	Numeric multidimensional	2-d DWT
Graphs	Numeric multidimensional	MDS, spectral
Any type	Graphs	Similarity graph (Restricted applicability)

ranges $[a, a \cdot \alpha]$, $[a \cdot \alpha, a \cdot \alpha^2]$, and so on, for some $\alpha > 1$. This kind of range may be useful when the attribute shows an exponential distribution across a range. In fact, if the attribute frequency distribution for an attribute can be modeled in functional form, then a natural approach would be to select ranges $[a, b]$ such that $f(b) - f(a)$ is the same for some function $f(\cdot)$. The idea is to select this function $f(\cdot)$ in such a way that each range contains an approximately similar number of records. However, in most cases, it is hard to find such a function $f(\cdot)$ in closed form.

3. *Equi-depth ranges:* In this case, the ranges are selected so that each range has an equal number of records. The idea is to provide the same level of granularity to each range. An attribute can be divided into equi-depth ranges by first sorting it, and then selecting the division points on the sorted attribute value, such that each range contains an equal number of records.

The process of discretization can also be used to convert time-series data to discrete sequence data.

2.2.2.2 Categorical to Numeric Data: Binarization

In some cases, it is desirable to use numeric data mining algorithms on categorical data. Because binary data is a special form of both numeric and categorical data, it is possible to convert the categorical attributes to binary form and then use numeric algorithms on the binarized data. If a categorical attribute has ϕ different values, then ϕ different binary attributes are created. Each binary attribute corresponds to one possible value of the categorical attribute. Therefore, exactly one of the ϕ attributes takes on the value of 1, and the remaining take on the value of 0.

2.2.2.3 Text to Numeric Data

Although the vector-space representation of text can be considered a sparse numeric data set with very high dimensionality, this special numeric representation is not very amenable to conventional data mining algorithms. For example, one typically uses specialized similarity functions, such as the cosine, rather than the Euclidean distance for text data. This is the reason that text mining is a distinct area in its own right with its own family of specialized algorithms. Nevertheless, it is possible to convert a text collection into a form

that is more amenable to the use of mining algorithms for numeric data. The first step is to use *latent semantic analysis (LSA)* to transform the text collection to a nonsparse representation with lower dimensionality. Furthermore, after transformation, each document $\overline{X} = (x_1 \ldots x_d)$ needs to be scaled to $\frac{1}{\sqrt{\sum_{i=1}^{d} x_i^2}}(x_1 \ldots x_d)$. This scaling is necessary to ensure that documents of varying length are treated in a uniform way. After this scaling, traditional numeric measures, such as the Euclidean distance, work more effectively. *LSA* is discussed in Sect. 2.4.3.3 of this chapter. Note that *LSA* is rarely used in conjunction with this kind of scaling. Rather, traditional text mining algorithms are directly applied to the reduced representation obtained from *LSA*.

2.2.2.4 Time Series to Discrete Sequence Data

Time-series data can be converted to discrete sequence data using an approach known as *symbolic aggregate approximation (SAX)*. This method comprises two steps:

1. *Window-based averaging:* The series is divided into windows of length w, and the average time-series value over each window is computed.

2. *Value-based discretization:* The (already averaged) time-series values are discretized into a smaller number of approximately *equi-depth* intervals. This is identical to the equi-depth discretization of numeric attributes that was discussed earlier. The idea is to ensure that each symbol has an approximately equal frequency in the time series. The interval boundaries are constructed by assuming that the time-series values are distributed with a Gaussian assumption. The mean and standard deviation of the (windowed) time-series values are estimated in the data-driven manner to instantiate the parameters of the Gaussian distribution. The quantiles of the Gaussian distribution are used to determine the boundaries of the intervals. This is more efficient than sorting all the data values to determine quantiles, and it may be a more practical approach for a long (or streaming) time series. The values are discretized into a small number (typically 3 to 10) of intervals for the best results. Each such equi-depth interval is mapped to a symbolic value. This creates a symbolic representation of the time series, which is essentially a discrete sequence.

Thus, *SAX* might be viewed as an equi-depth discretization approach after window-based averaging.

2.2.2.5 Time Series to Numeric Data

This particular transformation is very useful because it enables the use of multidimensional algorithms for time-series data. A common method used for this conversion is the discrete wavelet transform (*DWT*). The wavelet transform converts the time series data to multidimensional data, as a set of coefficients that represent averaged differences between different portions of the series. If desired, a subset of the largest coefficients may be used to reduce the data size. This approach will be discussed in Sect. 2.4.4.1 on data reduction. An alternative method, known as the discrete Fourier transform (*DFT*), is discussed in Sect. 14.2.4.2 of Chap. 14. The common property of these transforms is that the various coefficients are no longer as dependency oriented as the original time-series values.

2.2.2.6 Discrete Sequence to Numeric Data

This transformation can be performed in two steps. The first step is to convert the discrete sequence to a *set* of (binary) time series, where the number of time series in this set is equal to the number of distinct symbols. The second step is to map each of these time series into a multidimensional vector using the wavelet transform. Finally, the features from the different series are combined to create a single multidimensional record.

To convert a sequence to a binary time series, one can create a binary string in which the value denotes whether or not a particular symbol is present at a position. For example, consider the following nucleotide sequence, which is drawn on four symbols:

```
ACACACTGTGACTG
```

This series can be converted into the following set of four binary time series corresponding to the symbols A, C, T, and G, respectively:

```
10101000001000
01010100000100
00000010100010
00000001010001
```

A wavelet transformation can be applied to each of these series to create a multidimensional set of features. The features from the four different series can be appended to create a single numeric multidimensional record.

2.2.2.7 Spatial to Numeric Data

Spatial data can be converted to numeric data by using the same approach that was used for time-series data. The main difference is that there are now two contextual attributes (instead of one). This requires modification of the wavelet transformation method. Section 2.4.4.1 will briefly discuss how the one-dimensional wavelet approach can be generalized when there are two contextual attributes. The approach is fairly general and can be used for any number of contextual attributes.

2.2.2.8 Graphs to Numeric Data

Graphs can be converted to numeric data with the use of methods such as multidimensional scaling (*MDS*) and spectral transformations. This approach works for those applications where the edges are weighted, and represent similarity or distance relationships between nodes. The general approach of *MDS* can achieve this goal, and it is discussed in Sect. 2.4.4.2. A spectral approach can also be used to convert a graph into a multidimensional representation. This is also a dimensionality reduction scheme that converts the structural information into a multidimensional representation. This approach will be discussed in Sect. 2.4.4.3.

2.2.2.9 Any Type to Graphs for Similarity-Based Applications

Many applications are based on the notion of similarity. For example, the clustering problem is defined as the creation of groups of similar objects, whereas the outlier detection problem is defined as one in which a subset of objects differing significantly from the remaining objects are identified. Many forms of classification models, such as nearest neighbor classifiers, are also dependent on the notion of similarity. The notion of pairwise similarity can

be best captured with the use of a *neighborhood graph*. For a given set of data objects $\mathcal{O} = \{O_1 \ldots O_n\}$, a neighborhood graph is defined as follows:

1. A single node is defined for each object in $\mathcal{O}$. This is defined by the node set N, containing n nodes where the node i corresponds to the object O_i.

2. An edge exists between O_i and O_j, if the distance $d(O_i, O_j)$ is less than a particular threshold ϵ. Alternatively, the k-nearest neighbors of each node may be used. Because the k-nearest neighbor relationship is not symmetric, this results in a directed graph. The directions on the edges are ignored, and the parallel edges are removed. The weight w_{ij} of the edge (i, j) is equal to a kernelized function of the distance between the objects O_i and O_j, so that larger weights indicate greater similarity. An example is the *heat kernel*:
$$w_{ij} = e^{-d(O_i,O_j)^2/t^2} \tag{2.1}$$
Here, t is a user-defined parameter.

A wide variety of data mining algorithms are available for network data. All these methods can also be used on the similarity graph. Note that the similarity graph can be crisply defined for data objects of any type, as long as an appropriate distance function can be defined. This is the reason that distance function design is so important for virtually any data type. The issue of distance function design will be addressed in Chap. 3. Note that this approach is useful only for applications that are based on the notion of similarity or distances. Nevertheless, many data mining problems are directed or indirectly related to notions of similarity and distances.

2.3 Data Cleaning

The data cleaning process is important because of the errors associated with the data collection process. Several sources of missing entries and errors may arise during the data collection process. Some examples are as follows:

1. Some data collection technologies, such as sensors, are inherently inaccurate because of the hardware limitations associated with collection and transmission. Sometimes sensors may drop readings because of hardware failure or battery exhaustion.

2. Data collected using scanning technologies may have errors associated with it because optical character recognition techniques are far from perfect. Furthermore, speech-to-text data is also prone to errors.

3. Users may not want to specify their information for privacy reasons, or they may specify incorrect values intentionally. For example, it has often been observed that users sometimes specify their birthday incorrectly on automated registration sites such as those of social networks. In some cases, users may choose to leave several fields empty.

4. A significant amount of data is created manually. Manual errors are common during data entry.

5. The entity in charge of data collection may not collect certain fields for some records, if it is too costly. Therefore, records may be incompletely specified.

The aforementioned issues may be a significant source of inaccuracy for data mining applications. Methods are needed to remove or correct missing and erroneous entries from the data. There are several important aspects of data cleaning:

1. *Handling missing entries:* Many entries in the data may remain unspecified because of weaknesses in data collection or the inherent nature of the data. Such missing entries may need to be estimated. The process of estimating missing entries is also referred to as *imputation*.

2. *Handling incorrect entries:* In cases where the same information is available from multiple sources, *inconsistencies* may be detected. Such inconsistencies can be removed as a part of the analytical process. Another method for detecting the incorrect entries is to use domain-specific knowledge about what is already known about the data. For example, if a person's height is listed as 6 m, it is most likely incorrect. More generally, data points that are inconsistent with the remaining data distribution are often noisy. Such data points are referred to as *outliers*. It is, however, dangerous to assume that such data points are always caused by errors. For example, a record representing credit card fraud is likely to be inconsistent with respect to the patterns in most of the (normal) data but should not be removed as "incorrect" data.

3. *Scaling and normalization:* The data may often be expressed in very different scales (e.g., age and salary). This may result in some features being inadvertently weighted too much so that the other features are implicitly ignored. Therefore, it is important to normalize the different features.

The following sections will discuss each of these aspects of data cleaning.

2.3.1 Handling Missing Entries

Missing entries are common in databases where the data collection methods are imperfect. For example, user surveys are often unable to collect responses to all questions. In cases where data contribution is voluntary, the data is almost always incompletely specified. Three classes of techniques are used to handle missing entries:

1. Any data record containing a missing entry may be eliminated entirely. However, this approach may not be practical when most of the records contain missing entries.

2. The missing values may be estimated or imputed. However, errors created by the imputation process may affect the results of the data mining algorithm.

3. The analytical phase is designed in such a way that it can work with missing values. Many data mining methods are inherently designed to work robustly with missing values. This approach is usually the most desirable because it avoids the additional biases inherent in the imputation process.

The problem of estimating missing entries is directly related to the classification problem. In the classification problem, a single attribute is treated specially, and the other features are used to estimate its value. In this case, the missing value can occur on any feature, and therefore the problem is more challenging, although it is fundamentally not different. Many of the methods discussed in Chaps. 10 and 11 for classification can also be used for missing value estimation. In addition, the matrix completion methods discussed in Sect. 18.5 of Chap. 18 may also be used.

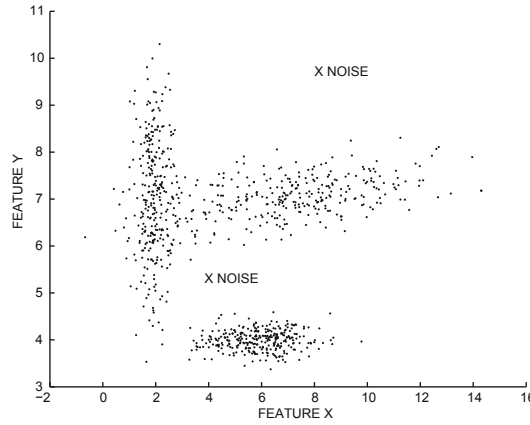


Figure 2.1: Finding noise by data-centric methods

In the case of dependency-oriented data, such as time series or spatial data, missing value estimation is much simpler. In this case, the behavioral attribute values of contextually nearby records are used for the imputation process. For example, in a time-series data set, the average of the values at the time stamp just before or after the missing attribute may be used for estimation. Alternatively, the behavioral values at the last n time-series data stamps can be linearly interpolated to determine the missing value. For the case of spatial data, the estimation process is quite similar, where the average of values at neighboring spatial locations may be used.

2.3.2 Handling Incorrect and Inconsistent Entries

The key methods that are used for removing or correcting the incorrect and inconsistent entries are as follows:

1. *Inconsistency detection:* This is typically done when the data is available from different sources in different formats. For example, a person's name may be spelled out in full in one source, whereas the other source may only contain the initials and a last name. In such cases, the key issues are duplicate detection and inconsistency detection. These topics are studied under the general umbrella of data integration within the database field.

2. *Domain knowledge:* A significant amount of domain knowledge is often available in terms of the ranges of the attributes or rules that specify the relationships across different attributes. For example, if the country field is "United States," then the city field cannot be "Shanghai." Many *data scrubbing* and *data auditing* tools have been developed that use such domain knowledge and constraints to detect incorrect entries.

3. *Data-centric methods:* In these cases, the *statistical behavior* of the data is used to detect outliers. For example, the two isolated data points in Fig. 2.1 marked as "noise" are outliers. These isolated points might have arisen because of errors in the data collection process. However, this may not always be the case because the anomalies may be the result of interesting behavior of the underlying system. Therefore, any detected outlier may need to be manually examined before it is discarded. The use of

data-centric methods for cleaning can sometimes be dangerous because they can result in the removal of useful knowledge from the underlying system. The outlier detection problem is an important analytical technique in its own right, and is discussed in detail in Chaps. 8 and 9.

The methods for addressing erroneous and inconsistent entries are generally highly domain specific.

2.3.3 Scaling and Normalization

In many scenarios, the different features represent different scales of reference and may therefore not be comparable to one another. For example, an attribute such as age is drawn on a very different scale than an attribute such as salary. The latter attribute is typically orders of magnitude larger than the former. As a result, any aggregate function computed on the different features (e.g., Euclidean distances) will be dominated by the attribute of larger magnitude.

To address this problem, it is common to use *standardization*. Consider the case where the jth attribute has mean μ_j and standard deviation σ_j. Then, the jth attribute value x_i^j of the ith record $\overline{X_i}$ may be normalized as follows:

$$z_i^j = \frac{x_i^j - \mu_j}{\sigma_j} \tag{2.2}$$

The vast majority of the normalized values will typically lie in the range $[-3, 3]$ under the normal distribution assumption.

A second approach uses *min-max scaling* to map all attributes to the range $[0, 1]$. Let min_j and max_j represent the minimum and maximum values of attribute j. Then, the jth attribute value x_i^j of the ith record $\overline{X_i}$ may be scaled as follows:

$$y_i^j = \frac{x_i^j - min_j}{max_j - min_j} \tag{2.3}$$

This approach is not effective when the maximum and minimum values are extreme value outliers because of some mistake in data collection. For example, consider the age attribute where a mistake in data collection caused an additional zero to be appended to an age, resulting in an age value of 800 years instead of 80. In this case, most of the scaled data along the age attribute will be in the range $[0, 0.1]$, as a result of which this attribute may be de-emphasized. Standardization is more robust to such scenarios.

2.4 Data Reduction and Transformation

The goal of data reduction is to represent it more compactly. When the data size is smaller, it is much easier to apply sophisticated and computationally expensive algorithms. The reduction of the data may be in terms of the number of rows (records) or in terms of the number of columns (dimensions). Data reduction does result in some loss of information. The use of a more sophisticated algorithm may sometimes compensate for the loss in information resulting from data reduction. Different types of data reduction are used in various applications:

1. *Data sampling:* The records from the underlying data are sampled to create a much smaller database. Sampling is generally much harder in the streaming scenario where the sample needs to be dynamically maintained.

2. *Feature selection:* Only a subset of features from the underlying data is used in the analytical process. Typically, these subsets are chosen in an application-specific way. For example, a feature selection method that works well for clustering may not work well for classification and vice versa. Therefore, this section will discuss the issue of feature subsetting only in a limited way and defer a more detailed discussion to later chapters.

3. *Data reduction with axis rotation:* The correlations in the data are leveraged to represent it in a smaller number of dimensions. Examples of such data reduction methods include principal component analysis (PCA), singular value decomposition (SVD), or latent semantic analysis (LSA) for the text domain.

4. *Data reduction with type transformation:* This form of data reduction is closely related to data type portability. For example, time series are converted to multidimensional data of a smaller size and lower complexity by discrete wavelet transformations. Similarly, graphs can be converted to multidimensional representations by using embedding techniques.

Each of the aforementioned aspects will be discussed in different segments of this section.

2.4.1 Sampling

The main advantage of sampling is that it is simple, intuitive, and relatively easy to implement. The type of sampling used may vary with the application at hand.

2.4.1.1 Sampling for Static Data

It is much simpler to sample data when the entire data is already available, and therefore the number of base data points is known in advance. In the unbiased sampling approach, a predefined fraction f of the data points is selected and retained for analysis. This is extremely simple to implement, and can be achieved in two different ways, depending upon whether or not replacement is used.

In sampling without replacement from a data set $\mathcal{D}$ with n records, a total of $\lceil n \cdot f \rceil$ records are randomly picked from the data. Thus, no duplicates are included in the sample, unless the original data set $\mathcal{D}$ also contains duplicates. In sampling with replacement from a data set $\mathcal{D}$ with n records, the records are sampled *sequentially and independently* from the *entire* data set $\mathcal{D}$ for a total of $\lceil n \cdot f \rceil$ times. Thus, duplicates are possible because the same record may be included in the sample over sequential selections. Generally, most applications do not use replacement because unnecessary duplicates can be a nuisance for some data mining applications, such as outlier detection. Some other specialized forms of sampling are as follows:

1. *Biased sampling:* In biased sampling, some parts of the data are intentionally emphasized because of their greater importance to the analysis. A classical example is that of temporal-decay bias where more recent records have a larger chance of being included in the sample, and stale records have a lower chance of being included. In exponential-decay bias, the probability $p(\overline{X})$ of sampling a data record $\overline{X}$, which was generated

δt time units ago, is proportional to an exponential decay function value regulated by the decay parameter λ:

$$p(\overline{X}) \propto e^{-\lambda \cdot \delta t} \tag{2.4}$$

Here e is the base of the natural logarithm. By using different values of λ, the impact of temporal decay can be regulated appropriately.

2. *Stratified sampling:* In some data sets, important parts of the data may not be sufficiently represented by sampling because of their rarity. A stratified sample, therefore, first partitions the data into a set of desired strata, and then independently samples from each of these strata based on predefined proportions in an application-specific way.

   For example, consider a survey that measures the economic diversity of the lifestyles of different individuals in the population. Even a sample of 1 million participants may not capture a billionaire because of their relative rarity. However, a stratified sample (by income) will independently sample a predefined fraction of participants from each income group to ensure greater robustness in analysis.

Numerous other forms of biased sampling are possible. For example, in density-biased sampling, points in higher-density regions are weighted less to ensure greater representativeness of the rare regions in the sample.

2.4.1.2 Reservoir Sampling for Data Streams

A particularly interesting form of sampling is that of *reservoir sampling* for data streams. In reservoir sampling, a sample of k points is *dynamically* maintained from a data stream. Recall that a stream is of an extremely large volume, and therefore one cannot store it on a disk to sample it. Therefore, for each incoming data point in the stream, one must use a set of efficiently implementable operations to *maintain* the sample.

In the static case, the probability of including a data point in the sample is k/n where k is the sample size, and n is the number of points in the "data set." In this case, the "data set" is not static and cannot be stored on disk. Furthermore, the value of n is constantly increasing as more points arrive and previous data points (outside the sample) have already been discarded. Thus, the sampling approach works with *incomplete knowledge* about the previous history of the stream at any given moment in time. In other words, for each incoming data point in the stream, we need to *dynamically* make two simple *admission control* decisions:

1. What sampling rule should be used to decide whether to include the newly incoming data point in the sample?

2. What rule should be used to decide how to eject a data point from the sample to "make room" for the newly inserted data point?

Fortunately, it is relatively simple to design an algorithm for reservoir sampling in data streams [498]. For a reservoir of size k, the first k data points in the stream are used to initialize the reservoir. Subsequently, for the nth incoming stream data point, the following two admission control decisions are applied:

1. Insert the nth incoming stream data point into the reservoir with probability k/n.

2. If the newly incoming data point was inserted, then eject one of the old k data points at random to make room for the newly arriving point.

It can be shown that the aforementioned rule maintains an unbiased reservoir sample from the data stream.

Lemma 2.4.1 *After n stream points have arrived, the probability of any stream point being included in the reservoir is the same, and is equal to k/n.*

Proof: This result is easy to show by induction. At initialization of the first k data points, the theorem is trivially true. Let us (inductively) assume that it is also true after $(n-1)$ data points have been received, and therefore the probability of each point being included in the reservoir is $k/(n-1)$. The probability of the arriving point being included in the stream is k/n, and therefore the lemma holds true for the arriving data point. It remains to prove the result for the remaining points in the data stream. There are two *disjoint* case events that can arise for an incoming data point, and the final probability of a point being included in the reservoir is the sum of these two cases:

I: The incoming data point is not inserted into the reservoir. The probability of this is $(n-k)/n$. Because the original probability of any point being included in the reservoir by the inductive assumption, is $k/(n-1)$, the overall probability of a point being included in the reservoir *and* Case I event, is the multiplicative value of $p_1 = \frac{k(n-k)}{n(n-1)}$.

II: The incoming data point is inserted into the reservoir. The probability of Case II is equal to insertion probability k/n of incoming data points. Subsequently, existing reservoir points are retained with probability $(k-1)/k$ because exactly one of them is ejected. Because the inductive assumption implies that any of the earlier points in the data stream was originally present in the reservoir with probability $k/(n-1)$, it implies that the probability of a point being included in the reservoir *and* Case II event is given by the product p_2 of the three aforementioned probabilities:

$$p_2 = \left(\frac{k}{n}\right)\left(\frac{k-1}{k}\right)\left(\frac{k}{n-1}\right) = \frac{k(k-1)}{n(n-1)} \tag{2.5}$$

Therefore, the total probability of a stream point being retained in the reservoir after the nth data point arrival is given by the sum of p_1 and p_2. It can be shown that this is equal to k/n. ■

It is possible to extend reservoir sampling to cases where temporal bias is present in the data stream. In particular, the case of exponential bias has been addressed in [35].

2.4.2 Feature Subset Selection

A second method for data preprocessing is feature subset selection. Some features can be discarded when they are known to be irrelevant. Which features are relevant? Clearly, this decision depends on the application at hand. There are two primary types of feature selection:

1. *Unsupervised feature selection:* This corresponds to the removal of noisy and redundant attributes from the data. Unsupervised feature selection is best defined in terms of its impact on clustering applications, though the applicability is much broader. It is difficult to comprehensively describe such feature selection methods without using the clustering problem as a proper context. Therefore, a discussion of methods for unsupervised feature selection is deferred to Chap. 6 on data clustering.

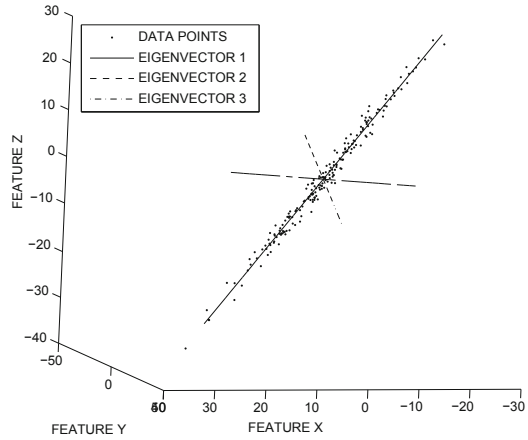


Figure 2.2: Highly correlated data represented in a small number of dimensions in an axis system that is rotated appropriately

2. *Supervised feature selection:* This type of feature selection is relevant to the problem of data classification. In this case, only the features that can predict the class attribute effectively are the most relevant. Such feature selection methods are often closely integrated with analytical methods for classification. A detailed discussion is deferred to Chap. 10 on data classification.

Feature selection is an important part of the data mining process because it defines the quality of the input data.

2.4.3 Dimensionality Reduction with Axis Rotation

In real data sets, a significant number of correlations exist among different attributes. In some cases, hard constraints or rules between attributes may uniquely define some attributes in terms of others. For example, the date of birth of an individual (represented quantitatively) is perfectly correlated with his or her age. In most cases, the correlations may not be quite as perfect, but significant dependencies may still exist among the different features. Unfortunately, real data sets contain many such redundancies that escape the attention of the analyst during the initial phase of data creation. These correlations and constraints correspond to implicit redundancies because they imply that knowledge of some subsets of the dimensions can be used to predict the values of the other dimensions. For example, consider the 3-dimensional data set illustrated in Fig. 2.2. In this case, if the axis is rotated to the orientation illustrated in the figure, the correlations and redundancies in the newly transformed feature values are removed. As a result of this redundancy removal, the entire data can be (approximately) represented along a 1-dimensional line. Thus, the *intrinsic dimensionality* of this 3-dimensional data set is 1. The other two axes correspond to the low-variance dimensions. If the data is represented as coordinates in the new axis system illustrated in Fig. 2.2, then the coordinate values along these low-variance dimensions will not vary much. Therefore, after the axis system has been rotated, these dimensions can be removed without much information loss.

A natural question arises as to how the correlation-removing axis system such as that in Fig. 2.2 may be determined in an automated way. Two natural methods to achieve this goal

are those of *principal component analysis (PCA)* and *singular value decomposition (SVD)*. These two methods, while not exactly identical at the definition level, are closely related. Although the notion of principal component analysis is intuitively easier to understand, *SVD* is a more general framework and can be used to perform *PCA* as a special case.

2.4.3.1 Principal Component Analysis

PCA is generally applied after subtracting the mean of the data set from each data point. However, it is also possible to use it without mean centering, as long as the mean of the data is separately stored. This operation is referred to as *mean centering*, and it results in a data set centered at the origin. The goal of *PCA* is to rotate the data into an axis-system where the greatest amount of variance is captured in a small number of dimensions. It is intuitively evident from the example of Fig. 2.2 that such an axis system is affected by the correlations between attributes. An important observation, which we will show below, is that the variance of a data set along a particular direction can be expressed directly in terms of its covariance matrix.

Let C be the $d \times d$ symmetric covariance matrix of the $n \times d$ data matrix D. Thus, the (i, j)th entry c_{ij} of C denotes the covariance between the ith and jth columns (dimensions) of the data matrix D. Let μ_i represent the mean along the ith dimension. Specifically, if x_k^m be the mth dimension of the kth record, then the value of the covariance entry c_{ij} is as follows:

$$c_{ij} = \frac{\sum_{k=1}^{n} x_k^i x_k^j}{n} - \mu_i \mu_j \quad \forall i, j \in \{1 \ldots d\} \tag{2.6}$$

Let $\overline{\mu} = (\mu_1 \ldots \mu_d)$ is the d-dimensional row vector representing the means along the different dimensions. Then, the aforementioned $d \times d$ computations of Eq. 2.6 for different values of i and j can be expressed compactly in $d \times d$ matrix form as follows:

$$C = \frac{D^T D}{n} - \overline{\mu}^T \overline{\mu} \tag{2.7}$$

Note that the d diagonal entries of the matrix C correspond to the d variances. The covariance matrix C is positive semi-definite, because it can be shown that for any d-dimensional column vector $\overline{v}$, the value of $\overline{v}^T C \overline{v}$ is equal to the variance of the 1-dimensional projection $D\overline{v}$ of the data set D on $\overline{v}$.

$$\overline{v}^T C \overline{v} = \frac{(D\overline{v})^T D\overline{v}}{n} - (\overline{\mu}\,\overline{v})^2 = \text{Variance of 1-dimensional points in } D\overline{v} \geq 0 \tag{2.8}$$

In fact, the goal of *PCA* is to successively determine orthonormal vectors $\overline{v}$ maximizing $\overline{v}^T C \overline{v}$. How can one determine such directions? Because the covariance matrix is symmetric and positive semidefinite, it can be diagonalized as follows:

$$C = P \Lambda P^T \tag{2.9}$$

The columns of the matrix P contain the orthonormal eigenvectors of C, and Λ is a diagonal matrix containing the nonnegative eigenvalues. The entry Λ_{ii} is the eigenvalue corresponding to the ith eigenvector (or column) of the matrix P. These eigenvectors represent successive orthogonal solutions[1] to the aforementioned optimization model maximizing the variance $\overline{v}^T C \overline{v}$ along the unit direction $\overline{v}$.

[1]Setting the gradient of the Lagrangian relaxation $\overline{v}^T C \overline{v} - \lambda(||\overline{v}||^2 - 1)$ to 0 is equivalent to the eigenvector condition $C\overline{v} - \lambda\overline{v} = 0$. The variance along an eigenvector is $\overline{v}^T C \overline{v} = \overline{v}^T \lambda \overline{v} = \lambda$. Therefore, one should include the orthonormal eigenvectors in decreasing order of eigenvalue λ to maximize preserved variance in reduced subspace.

An interesting property of this diagonalization is that both the eigenvectors and eigenvalues have a geometric interpretation in terms of the underlying data distribution. Specifically, if the axis system of data representation is rotated to the orthonormal set of eigenvectors in the columns of P, then it can be shown that all $\binom{d}{2}$ covariances of the newly transformed feature values are zero. In other words, *the greatest variance-preserving directions are also the correlation-removing directions*. Furthermore, the eigenvalues represent the variances of the data along the corresponding eigenvectors. In fact, the diagonal matrix Λ is the new covariance matrix after axis rotation. Therefore, eigenvectors with large eigenvalues preserve greater variance, and are also referred to as *principal components*. Because of the nature of the optimization formulation used to derive this transformation, a new axis system containing only the eigenvectors with the largest eigenvalues is optimized to *retaining the maximum variance in a fixed number of dimensions*. For example, the scatter plot of Fig. 2.2 illustrates the various eigenvectors, and it is evident that the eigenvector with the largest variance is all that is needed to create a variance-preserving representation. It generally suffices to retain only a small number of eigenvectors with large eigenvalues.

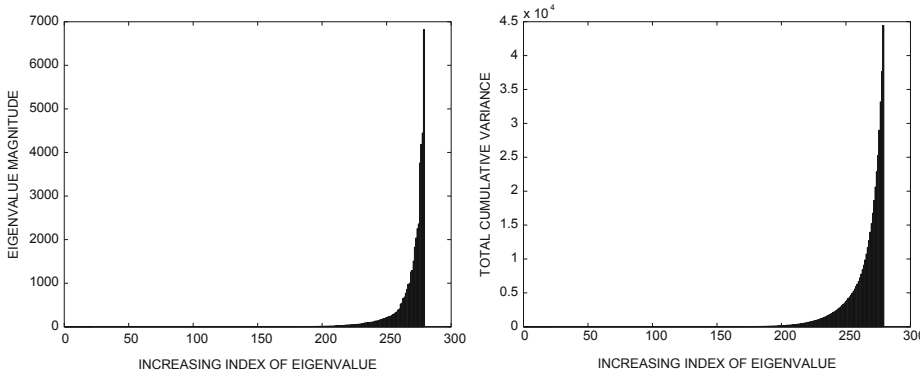
Without loss of generality, it can be assumed that the columns of P (and corresponding diagonal matrix Λ) are arranged from left to right in such a way that they correspond to decreasing eigenvalues. Then, the transformed data matrix D' in the new coordinate system after axis rotation to the orthonormal columns of P can be algebraically computed as the following linear transformation:

$$D' = DP \tag{2.10}$$

While the transformed data matrix D' is also of size $n \times d$, only its first (leftmost) $k \ll d$ columns will show significant variation in values. Each of the remaining $(d - k)$ columns of D' will be approximately equal to the mean of the data in the rotated axis system. For mean-centered data, the values of these $(d - k)$ columns will be almost 0. Therefore, the dimensionality of the data can be reduced, and only the first k columns of the transformed data matrix D' may need to be retained[2] for representation purposes. Furthermore, it can be confirmed that the covariance matrix of the transformed data $D' = DP$ is the diagonal matrix Λ by applying the covariance definition of Eq. 2.7 to DP (transformed data) and $\overline{\mu}P$ (transformed mean) instead of D and $\overline{\mu}$, respectively. The resulting covariance matrix can be expressed in terms of the original covariance matrix C as $P^T C P$. Substituting $C = P \Lambda P^T$ from Eq. 2.9 shows equivalence because $P^T P = P P^T = I$. In other words, correlations have been removed from the transformed data because Λ is diagonal.

The variance of the data set defined by projections along top-k eigenvectors is equal to the sum of the k corresponding eigenvalues. In many applications, the eigenvalues show a precipitous drop-off after the first few values. For example, the behavior of the eigenvalues for the 279-dimensional *Arrythmia* data set from the *UCI Machine Learning Repository* [213] is illustrated in Fig. 2.3. Figure 2.3a shows the absolute magnitude of the eigenvalues in increasing order, whereas Fig. 2.3b shows the total amount of variance retained in the top-k eigenvalues. Figure 2.3b can be derived by using the cumulative sum of the smallest eigenvalues in Fig. 2.3a. It is interesting to note that the 215 smallest eigenvalues contain less than 1 % of the total variance in the data and can therefore be removed with little change to the results of similarity-based applications. Note that the *Arrythmia* data set is not a very strongly correlated data set along many pairs of dimensions. Yet, the dimensionality reduction is drastic because of the *cumulative* effect of the correlations across many dimensions.

[2]The means of the remaining columns also need be stored if the data set is not mean centered.



(a) Magnitude of Eigenvalues (Increasing Index): *Arrythmia*

(b) Variance in smallest k Eigenvalues: *Arrythmia*

Figure 2.3: Variance retained with increasing number of eigenvalues for the *Arrythmia* data set

The eigenvectors of the matrix C may be determined by using any numerical method discussed in [295] or by an off-the-shelf eigenvector solver. *PCA* can be extended to discovering nonlinear embeddings with the use of a method known as the *kernel trick*. Refer to Sect. 10.6.4.1 of Chap. 10 for a brief description of kernel *PCA*.

2.4.3.2 Singular Value Decomposition

Singular value decomposition (*SVD*) is closely related to principal component analysis (*PCA*). However, these distinct methods are sometimes confused with one another because of the close relationship. Before beginning the discussion of *SVD*, we state how it is related to *PCA*. *SVD* is more general than *PCA* because it provides *two* sets of basis vectors instead of one. *SVD* provides basis vectors of both the rows and columns of the data matrix, whereas *PCA* only provides basis vectors of the rows of the data matrix. Furthermore, *SVD* provides the same basis as *PCA* for the rows of the data matrix in certain special cases:

SVD provides the same basis vectors and data transformation as PCA for data sets in which the mean of each attribute is 0.

The basis vectors of *PCA* are invariant to mean-translation, whereas those of *SVD* are not. When the data are not mean centered, the basis vectors of *SVD* and *PCA* will not be the same, and qualitatively different results may be obtained. *SVD* is often applied without mean centering to sparse nonnegative data such as document-term matrices. A formal way of defining *SVD* is as a decomposable product of (or *factorization* into) three matrices:

$$D = Q\Sigma P^T \tag{2.11}$$

Here, Q is an $n \times n$ matrix with orthonormal columns, which are the *left singular vectors*. Σ is an $n \times d$ diagonal matrix containing the *singular values*, which are always nonnegative and, by convention, arranged in nonincreasing order. Furthermore, P is a $d \times d$ matrix with orthonormal columns, which are the *right singular vectors*. Note that the diagonal matrix Σ is rectangular rather than square, but it is referred to as diagonal because only entries of the

form Σ_{ii} are nonzero. It is a fundamental fact of linear algebra that such a decomposition always exists, and a proof may be found in [480]. The number of nonzero diagonal entries of Σ is equal to the rank of the matrix D, which is at most $\min\{n, d\}$. Furthermore, because of the orthonormality of the singular vectors, both $P^T P$ and $Q^T Q$ are identity matrices. We make the following observations:

1. The columns of matrix Q, which are also the left singular vectors, are the orthonormal eigenvectors of DD^T. This is because $DD^T = Q\Sigma(P^T P)\Sigma^T Q^T = Q\Sigma\Sigma^T Q^T$. Therefore, the square of the nonzero singular values, which are diagonal entries of the $n \times n$ diagonal matrix $\Sigma\Sigma^T$, represent the nonzero eigenvalues of DD^T.

2. The columns of matrix P, which are also the right singular vectors, are the orthonormal eigenvectors of $D^T D$. The square of the nonzero singular values, which are represented in diagonal entries of the $d \times d$ diagonal matrix $\Sigma^T\Sigma$, are the nonzero eigenvalues of $D^T D$. Note that the nonzero eigenvalues of DD^T and $D^T D$ are the same. The matrix P is particularly important because it provides the basis vectors, which are analogous to the eigenvectors of the covariance matrix in *PCA*.

3. Because the covariance matrix of mean-centered data is $\frac{D^T D}{n}$ (cf. Eq. 2.7) and the right singular vectors of *SVD* are eigenvectors of $D^T D$, it follows that the eigenvectors of *PCA* are the same as the right-singular vectors of *SVD* for mean-centered data. Furthermore, the squared singular values in *SVD* are n times the eigenvalues of *PCA*. This equivalence shows why *SVD* and *PCA* can provide the same transformation for mean-centered data.

4. Without loss of generality, it can be assumed that the diagonal entries of Σ are arranged in decreasing order, and the columns of matrix P and Q are also ordered accordingly. Let P_k and Q_k be the truncated $d \times k$ and $n \times k$ matrices obtained by selecting the first k columns of P and Q, respectively. Let Σ_k be the $k \times k$ square matrix containing the top k singular values. Then, the *SVD* factorization yields an *approximate* d-dimensional data representation of the original data set D:

$$D \approx Q_k \Sigma_k P_k^T \tag{2.12}$$

   The columns of P_k represent a k-dimensional basis system for a reduced representation of the data set. The dimensionality reduced data set in this k-dimensional basis system is given by the $n \times k$ data set $D'_k = DP_k = Q_k\Sigma_k$, as in Eq. 2.10 of *PCA*. Each of the n rows of D'_k contain the k coordinates of each transformed data point in this new axis system. Typically, the value of k is much smaller than both n and d. Furthermore, unlike *PCA*, the rightmost $(d-k)$ columns of the full d-dimensional transformed data matrix $D' = DP$ will be approximately 0 (rather than the data mean), whether the data are mean centered or not. In general, *PCA* projects the data on a low-dimensional hyperplane passing through the data mean, whereas *SVD* projects the data on a low-dimensional hyperplane passing through the origin. *PCA* captures as much of the variance (or, squared Euclidean distance about the *mean*) of the data as possible, whereas *SVD* captures as much of the aggregate squared Euclidean distance about the *origin* as possible. This method of approximating a data matrix is referred to as *truncated SVD*.

In the following, we will show that truncated *SVD* maximizes the aggregate squared Euclidean distances (or *energy*) of the transformed data points about the origin. Let $\overline{v}$ be a

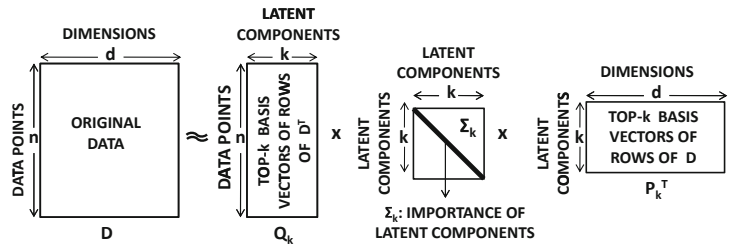


Figure 2.4: Complementary basis properties of matrix factorization in *SVD*

d-dimensional column vector and $D\overline{v}$ be the projection of the data set D on $\overline{v}$. Consider the problem of determining the *unit* vector $\overline{v}$ such that the sum of squared Euclidean distances $(D\overline{v})^T(D\overline{v})$ of the projected data points from the origin is maximized. Setting the gradient of the Lagrangian relaxation $\overline{v}^T D^T D\overline{v} - \lambda(||\overline{v}||^2 - 1)$ to 0 is equivalent to the eigenvector condition $D^T D\overline{v} - \lambda\overline{v} = 0$. Because the right singular vectors are eigenvectors of $D^T D$, it follows that the eigenvectors (right singular vectors) with the k largest eigenvalues (squared singular values) provide a basis that maximizes the preserved energy in the transformed and reduced data matrix $D'_k = DP_k = Q_k\Sigma_k$. Because the *energy*, which is the sum of squared Euclidean distances from the origin, is invariant to axis rotation, the energy in D'_k is the same as that in $D'_k P_k^T = Q_k\Sigma_k P_k^T$. Therefore, *$k$-rank SVD is a maximum energy-preserving factorization*. This result is known as the *Eckart–Young theorem*.

The total preserved energy of the projection $D\overline{v}$ of the data set D along unit right-singular vector $\overline{v}$ with singular value σ is given by $(D\overline{v})^T(D\overline{v})$, which can be simplified as follows:

$$(D\overline{v})^T(D\overline{v}) = \overline{v}^T(D^T D\overline{v}) = \overline{v}^T(\sigma^2\overline{v}) = \sigma^2$$

Because the energy is defined as a linearly separable sum along orthonormal directions, the preserved energy in the data projection along the top-k singular vectors is equal to the sum of the squares of the top-k singular values. Note that the total energy in the data set D is always equal to the sum of the squares of all the nonzero singular values. It can be shown that maximizing the preserved energy is the same as minimizing the squared error[3] (or *lost* energy) of the k-rank approximation. This is because the sum of the energy in the preserved subspace and the lost energy in the complementary (discarded) subspace is always a constant, which is equal to the energy in the original data set D.

When viewed purely in terms of eigenvector analysis, *SVD* provides two different perspectives for understanding the transformed and reduced data. The transformed data matrix can either be viewed as the *projection* DP_k of the data matrix D on the top k basis eigenvectors P_k of the $d \times d$ *scatter matrix* $D^T D$, or it can *directly* be viewed as the scaled eigenvectors $Q_k\Sigma_k = DP_k$ of the $n \times n$ *dot-product similarity matrix* DD^T. While it is generally computationally expensive to extract the eigenvectors of an $n \times n$ similarity matrix, such an approach also generalizes to nonlinear dimensionality reduction methods where notions of linear basis vectors do not exist in the original space. In such cases, the dot-product similarity matrix is replaced with a more complex similarity matrix in order to extract a nonlinear embedding (cf. Table 2.3).

SVD is more general than *PCA* and can be used to simultaneously determine a subset of k basis vectors for the data matrix and its transpose with the maximum energy. The latter can be useful in understanding complementary transformation properties of D^T.

[3]The squared error is the sum of squares of the entries in the error matrix $D - Q_k\Sigma_k P_k^T$.

The orthonormal columns of Q_k provide a k-dimensional basis system for (approximately) transforming "data points" corresponding to the rows of D^T, and the matrix $D^T Q_k = P_k \Sigma_k$ contains the corresponding coordinates. For example, in a user-item ratings matrix, one may wish to determine either a reduced representation of the users, or a reduced representation of the items. *SVD* provides the basis vectors for both reductions. Truncated *SVD* expresses the data in terms of k dominant *latent components*. The ith latent component is expressed in the ith basis vectors of both D and D^T, and its relative importance in the data is defined by the ith singular value. By decomposing the matrix product $Q_k \Sigma_k P_k^T$ into column vectors of Q_k and P_k (i.e., dominant basis vectors of D^T and D), the following additive sum of the k latent components can be obtained:

$$Q_k \Sigma_k P_k^T = \sum_{i=1}^{k} \overline{q_i} \sigma_i \overline{p_i}^T = \sum_{i=1}^{k} \sigma_i (\overline{q_i}\ \overline{p_i}^T) \tag{2.13}$$

Here $\overline{q_i}$ is the ith column of Q, $\overline{p_i}$ is the ith column of P, and σ_i is the ith diagonal entry of Σ. Each latent component $\sigma_i (\overline{q_i}\ \overline{p_i}^T)$ is an $n \times d$ matrix with rank 1 and energy σ_i^2. This decomposition is referred to as *spectral decomposition*. The relationships of the reduced basis vectors to *SVD* matrix factorization are illustrated in Fig. 2.4.

An example of a rank-2 truncated *SVD* of a toy 6×6 matrix is illustrated below:

$$D = \begin{pmatrix} 2 & 2 & 1 & 2 & 0 & 0 \\ 2 & 3 & 3 & 3 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 2 & 1 & 2 \end{pmatrix} \approx Q_2 \Sigma_2 P_2^T$$

$$\approx \begin{pmatrix} -0.41 & 0.17 \\ -0.65 & 0.31 \\ -0.23 & 0.13 \\ -0.56 & -0.20 \\ -0.10 & -0.46 \\ -0.19 & -0.78 \end{pmatrix} \begin{pmatrix} 8.4 & 0 \\ 0 & 3.3 \end{pmatrix} \begin{pmatrix} -0.41 & -0.49 & -0.44 & -0.61 & -0.10 & -0.12 \\ 0.21 & 0.31 & 0.26 & -0.37 & -0.44 & -0.68 \end{pmatrix}$$

$$= \begin{pmatrix} 1.55 & 1.87 & \underline{1.67} & 1.91 & 0.10 & 0.04 \\ 2.46 & 2.98 & 2.66 & 2.95 & 0.10 & -0.03 \\ 0.89 & 1.08 & 0.96 & 1.04 & 0.01 & -0.04 \\ 1.81 & 2.11 & 1.91 & 3.14 & 0.77 & 1.03 \\ 0.02 & -0.05 & -0.02 & 1.06 & 0.74 & 1.11 \\ 0.10 & -0.02 & 0.04 & 1.89 & 1.28 & 1.92 \end{pmatrix}$$

Note that the rank-2 matrix is a good approximation of the original matrix. The entry with the largest error is underlined in the final approximated matrix. Interestingly, this entry is also *inconsistent* with the structure of the remaining matrix in the *original* data (why?). Truncated *SVD* often tries to correct inconsistent entries, and this property is sometimes leveraged for noise reduction in error-prone data sets.

2.4.3.3 Latent Semantic Analysis

Latent semantic analysis (*LSA*) is an application of the *SVD* method to the text domain. In this case, the data matrix D is an $n \times d$ document-term matrix containing normalized

word frequencies in the n documents, where d is the size of the lexicon. No mean centering is used, but the results are approximately the same as *PCA* because of the sparsity of D. The sparsity of D implies that most of the entries in D are 0, and the mean values of each column are much smaller than the nonzero values. In such scenarios, it can be shown that the covariance matrix is approximately proportional to $D^T D$. The sparsity of the data set also results in a low intrinsic dimensionality. Therefore, in the text domain, the reduction in dimensionality from *LSA* is rather drastic. For example, it is not uncommon to be able to represent a corpus drawn on a lexicon of 100,000 dimensions in fewer than 300 dimensions.

LSA is a classical example of how the "loss" of information from discarding some dimensions can actually result in an *improvement* in the quality of the data representation. The text domain suffers from two main problems corresponding to *synonymy* and *polysemy*. Synonymy refers to the fact that two words may have the same meaning. For example, the words "*comical*" and "*hilarious*" mean approximately the same thing. Polysemy refers to the fact that the same word may mean two different things. For example, the word "*jaguar*" could refer to a car or a cat. Typically, the significance of a word can only be understood in the context of other words in the document. This is a problem for similarity-based applications because the computation of similarity with the use of word frequencies may not be completely accurate. For example, two documents containing the words "*comical*" and "*hilarious*," respectively, may not be deemed sufficiently similar in the original representation space. The two aforementioned issues are a direct result of synonymy and polysemy effects. The truncated representation after *LSA* typically removes the noise effects of synonymy and polysemy because the (high-energy) singular vectors represent the directions of correlation in the data, and the appropriate context of the word is implicitly represented along these directions. The variations because of individual differences in usage are implicitly encoded in the low-energy directions, which are truncated anyway. It has been observed that significant *qualitative* improvements [184, 416] for text applications may be achieved with the use of *LSA*. The improvement[4] is generally greater in terms of synonymy effects than polysemy. This noise-removing behavior of *SVD* has also been demonstrated in general multidimensional data sets [25].

2.4.3.4 Applications of PCA and SVD

Although *PCA* and *SVD* are primarily used for data reduction and compression, they have many other applications in data mining. Some examples are as follows:

1. *Noise reduction:* While removal of the smaller eigenvectors/singular vectors in *PCA* and *SVD* can lead to information loss, it can also lead to *improvement* in the quality of data representation in surprisingly many cases. The main reason is that the variations along the small eigenvectors are often the result of noise, and their removal is generally beneficial. An example is the application of *LSA* in the text domain where the removal of the smaller components leads to the enhancement of the semantic characteristics of text. *SVD* is also used for deblurring noisy images. These text- and image-specific results have also been shown to be true in arbitrary data domains [25]. Therefore, the data reduction is not just space efficient but actually provides *qualitative* benefits in many cases.

[4]Concepts that are not present predominantly in the collection will be ignored by truncation. Therefore, alternative meanings reflecting infrequent concepts in the collection will be ignored. While this has a robust effect on the *average*, it may not always be the correct or complete disambiguation of polysemous words.

2. *Data imputation: SVD* and *PCA* can be used for data imputation applications [23], such as collaborative filtering, because the *reduced* matrices Q_k, Σ_k, and P_k can be estimated for small values of k even from incomplete data matrices. Therefore, the entire matrix can be approximately reconstructed as $Q_k \Sigma_k P_k^T$. This application is discussed in Sect. 18.5 of Chap. 18.

3. *Linear equations:* Many data mining applications are optimization problems in which the solution is recast into a system of linear equations. For any linear system $A\overline{y} = 0$, any right singular vector of A with 0 singular value will satisfy the system of equations (see Exercise 14). Therefore, any linear combination of the 0 singular vectors will provide a solution.

4. *Matrix inversion: SVD* can be used for the inversion of a square $d \times d$ matrix D. Let the decomposition of D be given by $Q\Sigma P^T$. Then, the inverse of D is $D^{-1} = P\Sigma^{-1}Q^T$. Note that Σ^{-1} can be trivially computed from Σ by inverting its diagonal entries. The approach can also be generalized to the determination of the *Moore–Penrose pseudoinverse* D^+ of a rank-k matrix D by inverting only the nonzero diagonal entries of Σ. The approach can even be generalized to non-square matrices by performing the additional operation of transposing Σ. Such matrix inversion operations are required in many data mining applications such as least-squares regression (cf. Sect. 11.5 of Chap. 11) and social network analysis (cf. Chap. 19).

5. *Matrix algebra:* Many network mining applications require the application of algebraic operations such as the computation of the powers of a matrix. This is common in random-walk methods (cf. Chap. 19), where the kth powers of the symmetric adjacency matrix of an undirected network may need to be computed. Such symmetric adjacency matrices can be decomposed into the form $Q\Delta Q^T$. The kth power of this decomposition can be efficiently computed as $D^k = Q\Delta^k Q^T$. In fact, any polynomial function of the matrix can be computed efficiently.

SVD and *PCA* are extraordinarily useful because matrix and linear algebra operations are ubiquitous in data mining. *SVD* and *PCA* facilitate such matrix operations by providing convenient decompositions and basis representations. *SVD* has rightly been referred to [481] as "absolutely a high point of linear algebra."

2.4.4 Dimensionality Reduction with Type Transformation

In these methods, dimensionality reduction is coupled with type transformation. In most cases, the data is *transformed* from a more complex type to a less complex type, such as multidimensional data. Thus, these methods serve the dual purpose of data reduction and type portability. This section will study two such transformation methods:

1. *Time series to multidimensional:* A number of methods, such as the discrete Fourier transform and discrete wavelet transform are used. While these methods can also be viewed as a rotation of an axis system defined by the various time stamps of the contextual attribute, the data are no longer dependency oriented after the rotation. Therefore, the resulting data set can be processed in a similar way to multidimensional data. We will study the Haar wavelet transform because of its intuitive simplicity.

2. *Weighted graphs to multidimensional:* Multidimensional scaling and spectral methods are used to embed weighted graphs in multidimensional spaces, so that the similarity or distance values on the edges are captured by a multidimensional embedding.

Table 2.2: An example of wavelet coefficient computation

Granularity (order k)	**Averages** (Φ values)	**DWT coefficients** (ψ values)
$k = 4$	(8, 6, 2, 3, 4, 6, 6, 5)	–
$k = 3$	(7, 2.5, 5, 5.5)	(1, −0.5, −1, 0.5)
$k = 2$	(4.75, 5.25)	(2.25, −0.25)
$k = 1$	(5)	(−0.25)

This section will discuss each of these techniques.

2.4.4.1 Haar Wavelet Transform

Wavelets are a well-known technique that can be used for multigranularity decomposition and summarization of time-series data into the multidimensional representation. The *Haar* wavelet is a particularly popular form of wavelet decomposition because of its intuitive nature and ease of implementation. To understand the intuition behind wavelet decomposition, an example of sensor temperatures will be used.

Suppose that a sensor measured the temperatures over the course of 12 h from the morning until the evening. Assume that the sensor samples temperatures at the rate of 1 sample/s. Thus, over the course of a single day, a sensor will collect $12 \times 60 \times 60 = 43,200$ readings. Clearly, this will not scale well over many days and many sensors. An important observation is that many adjacent sensor readings will be very similar, causing this representation to be very wasteful. So, how can we represent this data approximately in a small amount of space? How can we determine the key regions where "variations" in readings occur, and store these variations instead of repeating values?

Suppose we only stored the average over the entire day. This provides some idea of the temperature but not much else about the variation over the day. Now, if the difference in average temperature between the first half and second half of the day is also stored, we can derive the averages for both the first and second half of the day from these two values. This principle can be applied recursively because the first half of the day can be divided into the first quarter of the day and the second quarter of the day. Thus, with four stored values, we can perfectly reconstruct the averages in four quarters of the day. This process can be applied recursively right down to the level of granularity of the sensor readings. These "difference values" are used to derive wavelet coefficients. Of course, we did not yet achieve any data *reduction* because the number of such coefficients can be shown to be exactly equal to the length of the original time series.

It is important to understand that large difference values tell us more about the *variations* in the temperature values than the small ones, and they are therefore more important to store. Therefore, larger coefficient values are stored after a normalization for the level of granularity. This normalization, which is discussed later, has a bias towards storing coefficients representing longer time scales because trends over longer periods of time are more informative for (global) series reconstruction.

More formally, the wavelet technique creates a decomposition of the time series into a set of coefficient-weighted wavelet basis vectors. Each of the coefficients represents the rough variation of the time series between the two halves of a particular time range. The

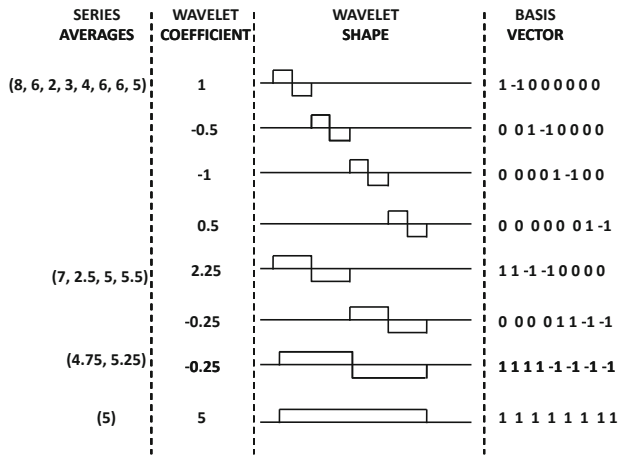


Figure 2.5: Illustration of the wavelet decomposition

wavelet basis vector is a time series that represents the temporal range of this variation in the form of a simple step function. The wavelet coefficients are of different *orders*, depending on the length of the time-series segment analyzed, which also represents the granularity of analysis. The higher-order coefficients represent the broad trends in the series because they correspond to larger ranges. The more localized trends are captured by the lower-order coefficients. Before providing a more notational description, a simple recursive description of wavelet decomposition of a time series segment S is provided below in two steps:

1. Report half the average difference of the behavioral attribute values between the first and second temporal halves of S as a wavelet coefficient.

2. Recursively apply this approach to first and second temporal halves of S.

At the end of the process, a reduction process is performed, where larger (normalized) coefficients are retained. This normalization step will be described in detail later.

A more formal and notation-intensive description will be provided at this point. For ease in discussion, assume that the length q of the series is a power of 2. For each value of $k \geq 1$, the Haar wavelet decomposition defines 2^{k-1} coefficients of order k. Each of these 2^{k-1} coefficients corresponds to a contiguous portion of the time series of length $q/2^{k-1}$. The ith of these 2^{k-1} coefficients corresponds to the segment in the series starting from position $(i-1) \cdot q/2^{k-1} + 1$ to the position $i \cdot q/2^{k-1}$. Let us denote this coefficient by ψ_k^i and the corresponding time-series segment by S_k^i. At the same time, let us define the average value of the first half of the S_k^i by a_k^i and that of the second half by b_k^i. Then, the value of ψ_k^i is given by $(a_k^i - b_k^i)/2$. More formally, if Φ_k^i denote the average value of the S_k^i, then the value of ψ_k^i can be defined recursively as follows:

$$\psi_k^i = (\Phi_{k+1}^{2 \cdot i-1} - \Phi_{k+1}^{2 \cdot i})/2 \tag{2.14}$$

The set of Haar coefficients is defined by all the coefficients of order 1 to $\log_2(q)$. In addition, the global average Φ_1^1 is required for the purpose of perfect reconstruction. The total number of coefficients is exactly equal to the length of the original series, and the dimensionality reduction is obtained by discarding the smaller (normalized) coefficients. This will be discussed later.

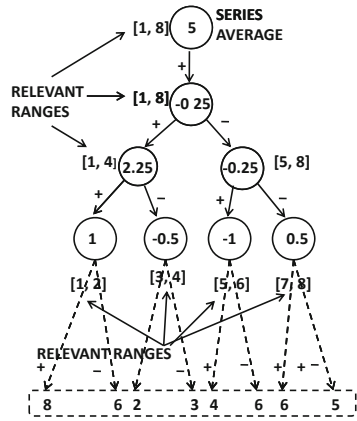


Figure 2.6: The error tree from the wavelet decomposition

The coefficients of different orders provide an understanding of the major trends in the data at a particular level of granularity. For example, the coefficient ψ_k^i is half the quantity by which the first half of the segment S_k^i is larger than the second half of the same segment. Because larger values of k correspond to geometrically reducing segment sizes, one can obtain an understanding of the basic trends at different levels of granularity. This definition of the Haar wavelet makes it very easy to compute by a sequence of averaging and differencing operations. Table 2.2 shows the computation of the wavelet coefficients for the sequence $(8, 6, 2, 3, 4, 6, 6, 5)$. This decomposition is illustrated in graphical form in Fig. 2.5. Note that each value in the original series can be represented as a sum of $\log_2(8) = 3$ wavelet coefficients with a positive or negative sign attached in front. In general, the entire decomposition may be represented as a tree of depth 3, which represents the hierarchical decomposition of the entire series. This is also referred to as the *error tree*. In Fig. 2.6, the error tree for the wavelet decomposition in Table 2.2 is illustrated. The nodes in the tree contain the values of the wavelet coefficients, except for a special *super-root* node that contains the series average.

The number of wavelet coefficients in this series is 8, which is also the length of the original series. The original series has been replicated just below the error tree in Fig. 2.6, and can be reconstructed by adding or subtracting the values in the nodes along the path leading to that value. Each coefficient in a node should be added, if we use the left branch below it to reach to the series values. Otherwise, it should be subtracted. This natural decomposition means that an entire contiguous range along the series can be reconstructed by using only the portion of the error tree which is relevant to it.

As in all dimensionality reduction methods, smaller coefficients are ignored. We will explain the process of discarding coefficients with the help of the notion of the *basis vectors* associated with each coefficient:

The wavelet representation is a decomposition of the original time series of length q into the weighted sum of a set of q "simpler" time series (or wavelets) that are orthogonal to one another. These "simpler" time series are the basis vectors, and the wavelet coefficients represent the weights of the different basis vectors in the decomposition.

Figure 2.5 shows these "simpler" time series along with their corresponding coefficients. The number of wavelet coefficients (and basis vectors) is equal to the length of the series q.

The length of the time series representing each basis vector is also q. Each basis vector has a $+1$ or -1 value in the contiguous time-series segment from which a particular coefficient was derived by a differencing operation. Otherwise, the value is 0 because the wavelet is not related to variations in that region of the time series. The first half of the nonzero segment of the basis vector is $+1$, and the second half is -1. This gives it the shape of a *wavelet* when it is plotted as a time series, and also reflects the differencing operation in the relevant time-series segment. Multiplying a basis vector with the coefficient has the effect of creating a weighted time series in which the difference between the first half and second half reflects the average difference between the corresponding segments in the original time series. Therefore, by adding up all these weighted wavelets over different levels of granularity in the error tree, it is possible to reconstruct the original series. The list of basis vectors in Fig. 2.5 are the rows of the following matrix:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Note that the dot product of any pair of basis vectors is 0, and therefore these series are orthogonal to one another. The most detailed coefficients have only one $+1$ and one -1, whereas the most coarse coefficient has four $+1$ and -1 entries. In addition, the vector (11111111) is needed to represent the series average.

For a time series T, let $\overline{W_1} \ldots \overline{W_q}$ be the corresponding basis vectors. Then, if $a_1 \ldots a_q$ are the wavelet coefficients for the basis vectors $\overline{W_1} \ldots \overline{W_q}$, the time series T can be represented as follows:

$$T = \sum_{i=1}^{q} a_i \overline{W_i} = \sum_{i=1}^{q} (a_i ||\overline{W_i}||) \frac{\overline{W_i}}{||\overline{W_i}||} \tag{2.15}$$

The coefficients represented in Fig. 2.5 are unnormalized because the underlying basis vectors do not have unit norm. While a_i is the unnormalized value from Fig. 2.5, the values $a_i||\overline{W_i}||$ represent normalized coefficients. The values of $||\overline{W_i}||$ are different for coefficients of different orders, and are equal to $\sqrt{2}$, $\sqrt{4}$, or $\sqrt{8}$ in this particular example. For example, in Fig. 2.5, the broadest level unnormalized coefficient is -0.25, whereas the corresponding normalized value is $-0.25\sqrt{8}$. After normalization, the basis vectors $\overline{W_1} \ldots \overline{W_q}$ are orthonormal, and, therefore, the sum of the squares of the corresponding (normalized) coefficients is equal to the retained energy in the approximated time series. Because the normalized coefficients provide a new coordinate representation after axis rotation, Euclidean distances between time series are preserved in this new representation if coefficients are not dropped. It can be shown that by retaining the coefficients with the largest normalized values, the error loss from the wavelet representation is minimized.

The previous discussion focused on the approximation of a single time series. In practice, one might want to convert a database of N time series into N multidimensional vectors. When a database of multiple time series is available, then two strategies can be used:

1. The coefficient for the same basis vector is selected for each series to create a meaningful multidimensional database of low dimensionality. Therefore, the basis vectors

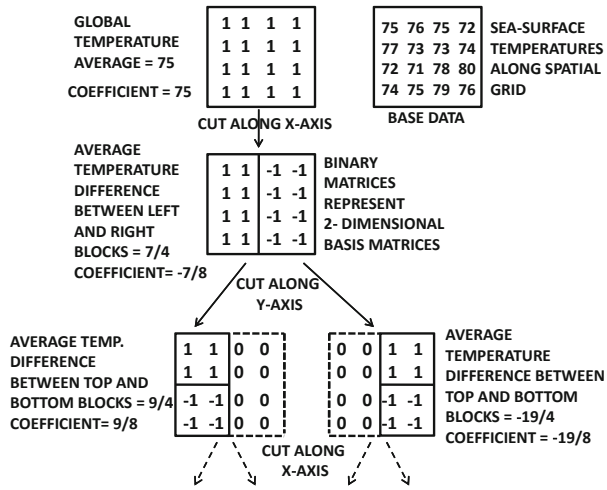


Figure 2.7: Illustration of the top levels of the wavelet decomposition for spatial data in a grid containing sea-surface temperatures

that have the largest *average* normalized coefficient across the N different series are selected.

2. The full dimensionality of the wavelet coefficient representation is retained. However, for each time series, the largest normalized coefficients (in magnitude) are selected individually. The remaining values are set to 0. This results in a sparse database of high dimensionality, in which many values are 0. A method such as *SVD* can be applied as a second step to further reduce the dimensionality. The second step of this approach has the disadvantage of losing interpretability of the features of the wavelet transform. Recall that the Haar wavelet is one of the few dimensionality reduction transforms where the coefficients do have some interpretability in terms of specific trends across particular time-series segments.

The wavelet decomposition method provides a natural method for dimensionality reduction (and data-type transformation) by retaining only a small number of coefficients.

Wavelet Decomposition with Multiple Contextual Attributes

Time-series data contain a single contextual attribute, corresponding to the time value. This helps in simplification of the wavelet decomposition. However, in some cases such as spatial data, there may be two contextual attributes corresponding to the X-coordinate and the Y-coordinate. For example, sea-surface temperatures are measured at spatial locations that are described with the use of two coordinates. How can wavelet decomposition be performed in such cases? In this section, a brief overview of the extension of wavelets to multiple contextual attributes is provided.

Assume that the spatial data is represented in the form of a 2-dimensional grid of size $q \times q$. Recall that in the 1-dimensional case, differencing operations were applied over contiguous segments of the time series by successive division of the time series in hierarchical fashion. The corresponding basis vectors have $+1$ and -1 at the relevant positions. The 2-dimensional case is completely analogous where contiguous *areas* of the spatial grid are used

by successive divisions. These divisions are alternately performed along the different axes. The corresponding basis vectors are 2-dimensional *matrices* of size $q \times q$ that regulate how the differencing operations are performed.

An example of the strategy for 2-dimensional decomposition is illustrated in Fig. 2.7. Only the top two levels of the decomposition are illustrated in the figure. Here, a 4×4 grid of spatial temperatures is used as an example. The first division along the X-axis divides the spatial area into two blocks of size 4×2 each. The corresponding two-dimensional binary basis matrix is illustrated into the same figure. The next phase divides each of these 4×2 blocks into blocks of size 2×2 during the hierarchical decomposition process. As in the case of 1-dimensional time series, the wavelet coefficient is half the difference in the average temperatures between the two halves of the relevant block being decomposed. The alternating process of division along the X-axis and the Y-axis can be carried on to the individual data entries. This creates a hierarchical wavelet error tree, which has many similar properties to that created in the 1-dimensional case. The overall principles of this decomposition are almost identical to the 1-dimensional case, with the major difference in terms of how the cuts along different dimensions are performed by alternating at different levels. The approach can be extended to the case of $k > 2$ contextual attributes with the use of a round-robin rotation in the axes that are selected at different levels of the tree for the differencing operation.

2.4.4.2 Multidimensional Scaling

Graphs are a powerful mechanism for representing relationships between objects. In some data mining scenarios, the data type of an object may be very complex and heterogeneous such as a time series annotated with text and other numeric attributes. However, a crisp notion of distance between several pairs of data objects may be available based on application-specific goals. How can one visualize the inherent similarity between these objects? How can one visualize the "nearness" of two individuals connected in a social network? A natural way of doing so is the concept of multidimensional scaling (*MDS*). Although *MDS* was originally proposed in the context of spatial visualization of graph-structured distances, it has much broader applicability for embedding data objects of arbitrary types in multidimensional space. Such an approach also enables the use of multidimensional data mining algorithms on the embedded data.

For a graph with n nodes, let $\delta_{ij} = \delta_{ji}$ denote the specified distance between nodes i and j. It is assumed that all $\binom{n}{2}$ pairwise distances between nodes are specified. It is desired to map the n nodes to n different k-dimensional vectors denoted by $\overline{X_1} \ldots \overline{X_n}$, so that the distances in multidimensional space closely correspond to the $\binom{n}{2}$ distance values in the distance graph. In *MDS*, the k coordinates of each of these n points are treated as variables that need to be optimized, so that they can *fit* the current set of pairwise distances. Metric *MDS*, also referred to as classical *MDS*, attempts to solve the following optimization (minimization) problem:

$$O = \sum_{i,j:i<j} (||\overline{X_i} - \overline{X_j}|| - \delta_{ij})^2 \tag{2.16}$$

Here $|| \cdot ||$ represents Euclidean norm. In other words, each node is represented by a multidimensional data point, such that the Euclidean distances between these points reflect the graph distances as closely as possible. In other forms of *nonmetric MDS*, this objective function might be different. This optimization problem therefore has $n \cdot k$ variables, and it scales with the size of the data n and the desired dimensionality k of the embedding. The

Table 2.3: Scaled eigenvectors of various similarity matrices yield embeddings with different properties

Method	Relevant similarity matrix
PCA	Dot product matrix DD^T after mean centering D
SVD	Dot product matrix DD^T
Spectral embedding (Symmetric Version)	Sparsified/normalized similarity matrix $\Lambda^{-1/2}W\Lambda^{-1/2}$ (cf. Sect. 19.3.4 of Chap. 19)
MDS/ISOMAP	Similarity matrix derived from distance matrix Δ with cosine law $S = -\frac{1}{2}(I - \frac{U}{n})\Delta(I - \frac{U}{n})$
Kernel PCA	Centered kernel matrix $S = (I - \frac{U}{n})K(I - \frac{U}{n})$ (cf. Sect. 10.6.4.1 of Chap. 10)

objective function O of Eq. 2.16 is usually divided by $\sum_{i,j:i<j} \delta_{ij}^2$ to yield a value in $(0, 1)$. The square root of this value is referred to as *Kruskal stress*.

The basic assumption in classical *MDS* is that the distance matrix $\Delta = [\delta_{ij}^2]_{n \times n}$ is generated by computing pairwise Euclidean distances in some hypothetical data matrix D for which the entries and dimensionality are unknown. The matrix D can never be recovered completely in classical *MDS* because Euclidean distances are invariant to mean translation and axis rotation. The appropriate conventions for the data mean and axis orientation will be discussed later. While the optimization of Eq. 2.16 requires numerical techniques, a direct solution to classical *MDS* can be obtained by eigen decomposition *under the assumption that the specified distance matrix is Euclidean*:

1. Any pairwise (squared) distance matrix $\Delta = [\delta_{ij}^2]_{n \times n}$ can be converted into a symmetric dot-product matrix $S_{n \times n}$ with the help of the *cosine law* in Euclidean space. In particular, if $\overline{X_i}$ and $\overline{X_j}$ are the embedded representations of the ith and jth nodes, the dot product between $\overline{X_i}$ and $\overline{X_j}$ can be related to the distances as follows:

$$\overline{X_i} \cdot \overline{X_j} = -\frac{1}{2}\left[||\overline{X_i} - \overline{X_j}||^2 - (||\overline{X_i}||^2 + ||\overline{X_j}||^2)\right] \quad \forall i, j \in \{1 \ldots n\} \tag{2.17}$$

   For a *mean-centered* embedding, the value of $||\overline{X_i}||^2 + ||\overline{X_j}||^2$ can be expressed (see Exercise 9) in terms of the entries of the distance matrix Δ as follows:

$$||\overline{X_i}||^2 + ||\overline{X_j}||^2 = \frac{\sum_{p=1}^n ||\overline{X_i} - \overline{X_p}||^2}{n} + \frac{\sum_{q=1}^n ||\overline{X_j} - \overline{X_q}||^2}{n} - \frac{\sum_{p=1}^n \sum_{q=1}^n ||\overline{X_p} - \overline{X_q}||^2}{n^2} \tag{2.18}$$

   A mean-centering assumption is necessary because the Euclidean distance is mean invariant, whereas the dot product is not. By substituting Eq. 2.18 in Eq. 2.17, it is possible to express the dot product $\overline{X_i} \cdot \overline{X_j}$ fully in terms of the entries of the distance matrix Δ. Because this condition is true for all possible values of i and j, we can conveniently express it in $n \times n$ matrix form. Let U be the $n \times n$ matrix of all 1s, and let I be the identity matrix. Then, our argument above shows that the dot-product matrix S is equal to $-\frac{1}{2}(I - \frac{U}{n})\Delta(I - \frac{U}{n})$. Under the Euclidean assumption, the matrix S is always positive semidefinite because it is equal to the $n \times n$ dot-product matrix DD^T of the *unobserved* data matrix D, which has unknown dimensionality. Therefore, it is desired to determine a high-quality factorization of S into the form $D_k D_k^T$, where D_k is an $n \times k$ matrix of dimensionality k.

2. Such a factorization can be obtained with eigen decomposition. Let $S \approx Q_k \Sigma_k^2 Q_k^T = (Q_k \Sigma_k)(Q_k \Sigma_k)^T$ represent the approximate diagonalization of S, where Q_k is an $n \times k$ matrix containing the largest k eigenvectors of S, and Σ_k^2 is a $k \times k$ diagonal matrix containing the eigenvalues. The embedded representation is given by $D_k = Q_k \Sigma_k$. Note that *SVD* also derives the optimal embedding as the scaled eigenvectors of the dot-product matrix of the original data. Therefore, the squared error of representation is minimized by this approach. This can also be shown to be equivalent to minimizing the Kruskal stress.

The optimal solution is not unique, because we can multiply $Q_k \Sigma_k$ with any $k \times k$ matrix with orthonormal columns, and the pairwise Euclidean distances will not be affected. In other words, any representation of $Q_k \Sigma_k$ in a rotated axis system is optimal as well. *MDS* finds an axis system like *PCA* in which the individual attributes are uncorrelated. In fact, if classical *MDS* is applied to a distance matrix Δ, which is constructed by computing the pairwise Euclidean distances in an actual data set, then it will yield the same embedding as the application of *PCA* on that data set. *MDS* is useful when such a data set is not available to begin with, and only the distance matrix Δ is available.

As in all dimensionality reduction methods, the value of the dimensionality k provides the trade-off between representation size and accuracy. Larger values of the dimensionality k will lead to lower stress. A larger number of data points typically requires a larger dimensionality of representation to achieve the same stress. The most crucial element is, however, the inherent structure of the distance matrix. For example, if a $10,000 \times 10,000$ distance matrix contains the pairwise driving distance between 10,000 cities, it can usually be approximated quite well with just a 2-dimensional representation. This is because driving distances are an approximation of Euclidean distances in 2-dimensional space. On the other hand, an arbitrary distance matrix may not be Euclidean and the distances may not even satisfy the triangle inequality. As a result, the matrix S might not be positive semidefinite. In such cases, it is sometimes still possible to use the metric assumption to obtain a high-quality embedding. Specifically, only those positive eigenvalues may be used, whose magnitude exceeds that of the most negative eigenvalue. This approach will work reasonably well if the negative eigenvalues have small magnitude.

MDS is commonly used in nonlinear dimensionality reduction methods such as *ISOMAP* (cf. Sect. 3.2.1.7 of Chap. 3). It is noteworthy that, in conventional *SVD*, the scaled eigenvectors of the $n \times n$ dot-product similarity matrix DD^T yield a low-dimensional embedded representation of D just as the eigenvectors of S yield the embedding in *MDS*. The eigen decomposition of similarity matrices is fundamental to many linear and nonlinear dimensionality reduction methods such as *PCA*, *SVD*, *ISOMAP*, *kernel PCA*, and spectral embedding. The specific properties of each embedding are a result of the choice of the similarity matrix and the scaling used on the resulting eigenvectors. Table 2.3 provides a preliminary comparison of these methods, although some of them are discussed in detail only in later chapters.

2.4.4.3 Spectral Transformation and Embedding of Graphs

Whereas *MDS* methods are designed for preserving global distances, spectral methods are designed for preserving local distances for applications such as clustering. Spectral methods work with *similarity graphs* in which the weights on the edges represent similarity rather than distances. When distance values are available they are converted to similarity values with kernel functions such as the heat kernel discussed earlier in this chapter. The notion

of similarity is natural to many real Web, social, and information networks because of the notion of *homophily*. For example, consider a bibliographic network in which nodes correspond to authors, and the edges correspond to co-authorship relations. The weight of an edge represents the number of publications between authors and therefore represents one possible notion of similarity in author publications. Similarity graphs can also be constructed between arbitrary data types. For example, a set of n time series can be converted into a graph with n nodes, where a node represents each time series. The weight of an edge is equal to the similarity between the two nodes, and only edges with a "sufficient" level of similarity are retained. A discussion of the construction of the similarity graph is provided in Sect. 2.2.2.9. Therefore, if a similarity graph can be transformed to a multidimensional representation that preserves the similarity structure between nodes, it provides a transformation that can port virtually any data type to the easily usable multidimensional representation. The caveat here is that such a transformation can only be used for similarity-based applications such as clustering or nearest neighbor classification because the transformation is designed to preserve the *local* similarity structure. The local similarity structure of a data set is nevertheless fundamental to many data mining applications.

Let $G = (N, A)$ be an undirected graph with node set N and edge set A. It is assumed that the node set contains n nodes. A symmetric $n \times n$ weight matrix $W = [w_{ij}]$ represents the similarities between the different nodes. Unlike *MDS*, which works with a complete graph of *global* distances, this graph is generally a *sparsified* representation of the similarity of each object to its k *nearest objects* (cf. Sect. 2.2.2.9). The similarities to the remaining objects are not distinguished from one another and set to 0. This is because spectral methods preserve only the local similarity structure for applications such as clustering. All entries in this matrix are assumed to be nonnegative, and higher values indicate greater similarity. If an edge does not exist between a pair of nodes, then the corresponding entry is assumed to be 0. It is desired to embed the nodes of this graph into a k-dimensional space so that the similarity structure of the data is preserved.

First, let us discuss the much simpler problem of mapping the nodes onto a 1-dimensional space. The generalization to the k-dimensional case is relatively straightforward. We would like to map the nodes in N into a set of 1-dimensional real values $y_1 \ldots y_n$ on a line, so that the distances between these points reflect the edge connectivity among the nodes. Therefore, it is undesirable for nodes that are connected with high-weight edges, to be mapped onto distant points on this line. Therefore, we would like to determine values of y_i that minimize the following objective function O:

$$O = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(y_i - y_j)^2 \tag{2.19}$$

This objective function penalizes the distances between y_i and y_j with weight proportional to w_{ij}. Therefore, when w_{ij} is very large (more similar nodes), the data points y_i and y_j will be more likely to be closer to one another in the embedded space. The objective function O can be rewritten in terms of the *Laplacian matrix* L of weight matrix W. The Laplacian matrix L is defined as $\Lambda - W$, where Λ is a diagonal matrix satisfying $\Lambda_{ii} = \sum_{j=1}^{n} w_{ij}$. Let the n-dimensional column vector of embedded values be denoted by $\overline{y} = (y_1 \ldots y_n)^T$. It can be shown after some algebraic simplification that the minimization objective function O can be rewritten in terms of the Laplacian matrix:

$$O = 2\overline{y}^T L \overline{y} \tag{2.20}$$

The matrix L is positive semidefinite with nonnegative eigenvalues because the sum-of-squares objective function O is always nonnegative. We need to incorporate a scaling constraint to ensure that the trivial value of $y_i = 0$ for all i, is not selected by the optimization solution. A possible scaling constraint is as follows:

$$\overline{y}^T \Lambda \overline{y} = 1 \tag{2.21}$$

The use of the matrix Λ in the constraint of Eq. 2.21 is essentially a normalization constraint, which is discussed in detail in Sect. 19.3.4 of Chap. 19.

It can be shown that the value of O is optimized by selecting $\overline{y}$ as the smallest eigenvector of the relationship $\Lambda^{-1} L \overline{y} = \lambda \overline{y}$. However, the smallest eigenvalue is always 0, and it corresponds to the trivial solution where the node embedding $\overline{y}$ is proportional to the vector containing only 1s. This trivial eigenvector is non-informative because it corresponds to an embedding in which every node is mapped to the same point. Therefore, it can be discarded, and it is not used in the analysis. The second-smallest eigenvector then provides an optimal solution that is more informative.

This solution can be generalized to finding an optimal k-dimensional embedding by determining successive directions corresponding to eigenvectors with increasing eigenvalues. After discarding the first trivial eigenvector $\overline{e_1}$ with eigenvalue $\lambda_1 = 0$, this results in a set of k eigenvectors $\overline{e_2}, \overline{e_3} \ldots \overline{e_{k+1}}$, with corresponding eigenvalues $\lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_{k+1}$. Each eigenvector is of length n and contains one coordinate value for each node. The ith value along the jth eigenvector represents the jth coordinate of the ith node. This creates an $n \times k$ matrix, corresponding to the k-dimensional embedding of the n nodes.

What do the small magnitude eigenvectors intuitively represent in the new transformed space? By using the ordering of the nodes along a small magnitude eigenvector to create a cut, the weight of the edges across the cut is likely to be small. Thus, this represents a cluster in the space of nodes. In practice, the k smallest eigenvectors (after ignoring the first) are selected to perform the reduction and create a k-dimensional embedding. This embedding typically contains an excellent representation of the underlying similarity structure of the nodes. The embedding can be used for virtually any similarity-based application, although the most common application of this approach is spectral clustering. Many variations of this approach exist in terms of how the Laplacian L is normalized, and in terms of how the final clusters are generated. The spectral clustering method will be discussed in detail in Sect. 19.3.4 of Chap. 19.

2.5 Summary

Data preparation is an important part of the data mining process because of the sensitivity of the analytical algorithms to the quality of the input data. The data mining process requires the collection of raw data from a variety of sources that may be in a form which is unsuitable for direct application of analytical algorithms. Therefore, numerous methods may need to be applied to extract features from the underlying data. The resulting data may have significant missing values, errors, inconsistencies, and redundancies. A variety of analytical methods and data scrubbing tools exist for imputing the missing entries or correcting inconsistencies in the data.

Another important issue is that of data heterogeneity. The analyst may be faced with a multitude of attributes that are distinct, and therefore the direct application of data mining algorithms may not be easy. Therefore, data type portability is important, wherein some subsets of attributes are converted to a predefined format. The multidimensional

format is often preferred because of its simplicity. Virtually, any data type can be converted to multidimensional representation with the two-step process of constructing a similarity graph, followed by multidimensional embedding.

The data set may be very large, and it may be desirable to reduce its size both in terms of the number of rows and the number of dimensions. The reduction in terms of the number of rows is straightforward with the use of sampling. To reduce the number of columns in the data, either feature subset selection or data transformation may be used. In feature subset selection, only a smaller set of features is retained that is most suitable for analysis. These methods are closely related to analytical methods because the relevance of a feature may be application dependent. Therefore, the feature selection phase need to be tailored to the specific analytical method.

There are two types of feature transformation. In the first type, the axis system may be rotated to align with the correlations of the data and retain the directions with the greatest variance. The second type is applied to complex data types such as graphs and time series. In these methods, the size of the representation is reduced, and the data is also transformed to a multidimensional representation.

2.6 Bibliographic Notes

The problem of feature extraction is an important one for the data mining process but it is highly application specific. For example, the methods for extracting named entities from a document data set [400] are very different from those that extract features from an image data set [424]. An overview of some of the promising technologies for feature extraction in various domains may be found in [245].

After the features have been extracted from different sources, they need to be integrated into a single database. Numerous methods have been described in the conventional database literature for data integration [194, 434]. Subsequently, the data needs to be cleaned and missing entries need to be removed. A new field of probabilistic or uncertain data has emerged [18] that models uncertain and erroneous records in the form of probabilistic databases. This field is, however, still in the research stage and has not entered the mainstream of database applications. Most of the current methods either use tools for missing data analysis [71, 364] or more conventional data cleaning and data scrubbing tools [222, 433, 435].

After the data has been cleaned, its size needs to be reduced either in terms of numerosity or in terms of dimensionality. The most common and simple numerosity reduction method is sampling. Sampling methods can be used for either static data sets or dynamic data sets. Traditional methods for data sampling are discussed in [156]. The method of sampling has also been extended to data streams in the form of reservoir sampling [35, 498]. The work in [35] discusses the extension of reservoir sampling methods to the case where a biased sample needs to be created from the data stream.

Feature selection is an important aspect of the data mining process. The approach is often highly dependent on the particular data mining algorithm being used. For example, a feature selection method that works well for clustering may not work well for classification. Therefore, we have deferred the discussion of feature selection to the relevant chapters on the topic on clustering and classification in this book. Numerous books are available on the topic of feature selection [246, 366].

The two most common dimensionality reduction methods used for multidimensional data are *SVD* [480, 481] and *PCA* [295]. These methods have also been extended to text in

the form of *LSA* [184, 416]. It has been shown in many domains [25, 184, 416] that the use of methods such as *SVD*, *LSA*, and *PCA* unexpectedly improves the quality of the underlying representation after performing the reduction. This improvement is because of reduction in noise effects by discarding the low-variance dimensions. Applications of *SVD* to data imputation are found in [23] and Chap. 18 of this book. Other methods for dimensionality reduction and transformation include Kalman filtering [260], Fastmap [202], and nonlinear methods such as *Laplacian eigenmaps* [90], *MDS* [328], and *ISOMAP* [490].

Many dimensionality reduction methods have also been proposed in recent years that simultaneously perform type transformation together with the reduction process. These include wavelet transformation [475] and graph embedding methods such as *ISOMAP* and *Laplacian eigenmaps* [90, 490]. A tutorial on spectral methods for graph embedding may be found in [371].

2.7 Exercises

1. Consider the time-series $(-3, -1, 1, 3, 5, 7, *)$. Here, a missing entry is denoted by $*$. What is the estimated value of the missing entry using linear interpolation on a window of size 3?

2. Suppose you had a bunch of text documents, and you wanted to determine all the personalities mentioned in these documents. What class of technologies would you use to achieve this goal?

3. Download the *Arrythmia* data set from the *UCI Machine Learning Repository* [213]. Normalize all records to a mean of 0 and a standard deviation of 1. Discretize each numerical attribute into (a) 10 equi-width ranges and (b) 10 equi-depth ranges.

4. Suppose that you had a set of arbitrary objects of different types representing different characteristics of widgets. A domain expert gave you the similarity value between every pair of objects. How would you convert these objects into a multidimensional data set for clustering?

5. Suppose that you had a data set, such that each data point corresponds to sea-surface temperatures over a square mile of resolution 10×10. In other words, each data record contains a 10×10 grid of temperature values with spatial locations. You also have some text associated with each 10×10 grid. How would you convert this data into a multidimensional data set?

6. Suppose that you had a set of discrete biological protein sequences that are annotated with text describing the properties of the protein. How would you create a multidimensional representation from this heterogeneous data set?

7. Download the *Musk* data set from the *UCI Machine Learning Repository* [213]. Apply *PCA* to the data set, and report the eigenvectors and eigenvalues.

8. Repeat the previous exercise using *SVD*.

9. For a mean-centered data set with points $\overline{X_1} \ldots \overline{X_n}$, show that the following is true:

$$||\overline{X_i}||^2 + ||\overline{X_j}||^2 = \frac{\sum_{p=1}^{n} ||\overline{X_i} - \overline{X_p}||^2}{n} + \frac{\sum_{q=1}^{n} ||\overline{X_j} - \overline{X_q}||^2}{n} - \frac{\sum_{p=1}^{n} \sum_{q=1}^{n} ||\overline{X_p} - \overline{X_q}||^2}{n^2} \tag{2.22}$$

10. Consider the time series $1, 1, 3, 3, 3, 3, 1, 1$. Perform wavelet decomposition on the time series. How many coefficients of the series are nonzero?

11. Download the *Intel Research Berkeley data set*. Apply a wavelet transformation to the temperature values in the first sensor.

12. Treat each quantitative variable in the *KDD CUP 1999 Network Intrusion Data Set* from the *UCI Machine Learning Repository* [213] as a time series. Perform the wavelet decomposition of this time series.

13. Create samples of size $n = 1, 10, 100, 1000, 10000$ records from the data set of the previous exercise, and determine the average value e_i of each quantitative column i using the sample. Let μ_i and σ_i be the global mean and standard deviation over the entire data set. Compute the number of standard deviations z_i by which e_i varies from μ_i.

$$z_i = \frac{|e_i - \mu_i|}{\sigma_i}$$

How does z_i vary with n?

14. Show that any right singular vector $\overline{y}$ of A with 0 singular value satisfies $A\overline{y} = 0$.

15. Show that the diagonalization of a square matrix is a specialized variation of *SVD*.