# Chapter 14

# Mining Time Series Data

"*The only reason for time is so that everything doesn't happen at once.*—Albert Einstein

## 14.1 Introduction

Temporal data is common in data mining applications. Typically, this is a result of continuously occurring processes in which the data is collected by hardware or software monitoring devices. The diversity of domains is quite significant and extends from the medical to the financial domain. Some examples of such data are as follows:

- *Sensor data:* Sensor data is often collected by a wide variety of hardware and other monitoring devices. Typically, this data contains continuous readings about the underlying data objects. For example, environmental data is commonly collected with different kinds of sensors that measure temperature, pressure, humidity, and so on. Sensor data is the most common form of time series data.
- *Medical devices:* Many medical devices such as electrocardiogram (ECG) and electroencephalogram (EEG) produce continuous streams of time series data. These represent measurements of the functioning of the human body, such as the heart beat, pulse rate, blood pressure, etc. Real-time data is also collected from patients in intensive care units (ICU) to monitor their condition.
- *Financial market data:* Financial data, such as stock prices, is often temporal. Other forms of temporal data include commodity prices, industrial trends, and economic indicators.

In general, temporal data may be either *discrete* or *continuous*. For example, Web log data contains a series of *discrete* events corresponding to user clicks, whereas environmental data may contain a series of continuous *values* such as temperature. Continuous temporal data sets are referred to as *time series*, whereas discrete temporal data sets are referred to as *sequences*. This chapter focuses on continuous time series data. The next chapter

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_14

studies data mining methods for discrete sequence data. While time series and discrete sequence data are conceptually similar, there are significant differences in the algorithmic methodologies used in each domain. However, in many cases, time series data is converted to discrete sequence data through discretization to facilitate the application of rich classes of sequence mining techniques. This chapter also discusses such cases.

Unlike multidimensional data, in which all attributes are treated equally, time series data are viewed as *contextual data* representations. In contextual data representations, the attributes are of two types:

- *Contextual attribute(s):* These represent the attributes that provide the *context* in which the measurements are made. In other words, the contextual attributes provide the reference points at which the behavioral values are measured. For the case of time series data, the *single* contextual attribute corresponds to the time dimension. Some data types, such as spatial data, may contain multiple contextual attributes corresponding to spatial coordinates. The time stamps could correspond to actual time values at which the data points are measured, or they could correspond to *consecutive indices* (or *ticks*) at which these values are measured.

- *Behavioral attribute(s):* These represent the behavioral values at the reference points. For example, in an environmental sensor, this could correspond to the temperature attribute. In general, each contextual attribute value (e.g., time stamp) has a corresponding behavioral attribute value (e.g., temperature). The behavioral attributes are usually the interesting ones from an application-specific perspective, but they cannot be properly interpreted without the knowledge of the contextual attributes. When more than one behavioral attribute is associated with each series, the corresponding series is referred to as a *multivariate* time series.

The analysis of contextual data types is more difficult because behavioral attribute values cannot be interpreted effectively without using the contextual attribute. For example, a sudden change of the behavioral attribute between successive time stamps (contextual attribute) is often indicative of outlier behavior. Thus, unlike multidimensional data, problem definitions are dependent on a combination of the interrelationships between contextual and behavioral attributes. Thus, problems such as clustering, classification, and outlier detection need to be significantly modified to account for the impact of the contextual attribute. Several data types discussed in subsequent chapters fall within this class. Other examples include sequence data and spatial data.

The greater complexity of time series data enables a larger number of problem definitions. Most of the models can be categorized into one of two types:

1. *Real-time analysis:* In real-time analysis, the data points in one or more series are analyzed in real time, to make predictions. Typically, a small window of recent history is used over the different data streams for the analysis. Examples of such analysis include forecasting, deviation detection, or event detection. When multiple series are available, they are typically analyzed in a temporally synchronized way. Even in cases where data mining applications such as clustering are applied to these problems, the analysis is typically performed in real time.

2. *Retrospective analysis:* In retrospective analysis, the time series data is already available, and subsequently analyzed. The analysis of different time series within a database is sometimes not synchronized over time. For example, in a time series database of ECG readings, the data may have been recorded over different periods.

Both these forms of analysis are useful in different kinds of applications. Furthermore, these two scenarios have different interpretations for the same applications such as clustering or outlier detection. These issues are discussed in more detail in later sections.

This chapter is organized as follows. The next section presents methods for time series preparation and similarity. Because the methods for time series similarity have already been discussed in detail in Chap. 3, they are summarized only briefly in this chapter. The reader is referred to the relevant sections of Chap. 3 for the different time series similarity measures. The problem of time series forecasting is discussed in Sect. 14.3. Time series motif discovery is discussed in Sect. 14.4. Section 14.5 addresses the problem of clustering time series. Outlier detection is discussed in Sect. 14.6. Time series classification is discussed in Sect. 14.7. The summary of the chapter is presented in Sect. 14.8.

## 14.2 Time Series Preparation and Similarity

Time series data may be either univariate or multivariate. In univariate time series data, a single behavioral attribute is associated with each time instant. In multivariate time series data, multiple behavioral attributes are associated with each time instant. The dimensionality of the time series, therefore, refers to the number of behavioral attributes being tracked.

**Definition 14.2.1 (Multivariate Time Series Data)** *A time series of length $n$ and dimensionality $d$ contains $d$ numeric features at each of $n$ timestamps $t_1 \ldots t_n$. Each timestamp contains a component for each of the $d$ series. Therefore, the set of values received at timestamp $t_i$ is $\overline{Y_i} = (y_i^1 \ldots y_i^d)$. The value of the $j$th series at timestamp $t_i$ is $y_i^j$*

In a univariate time series, the value of $d$ is 1. In such cases, a series of length $n$ is represented as a set of scalar behavioral values $y_1 \ldots y_n$, associated with the timestamps $t_1 \ldots t_n$.

### 14.2.1 Handling Missing Values

It is common for time series data to contain missing values. Furthermore, the values of the series may not be synchronized in time when they are collected by independent sensors. It is often convenient to have time series values that are equally spaced and synchronized across different behavioral attributes for data processing. The most common methodology used for handling missing, unequally spaced, or unsynchronized values is linear interpolation. The idea is to create *estimated* values at the desired time stamps. These can be used to generate multivariate time series that are synchronized, equally spaced, and have no missing values.

Consider the scenario where $y_i$ and $y_j$ are values of the time series at times $t_i$ and $t_j$, respectively, where $i < j$. Let $t$ be a time drawn from the interval $(t_i, t_j)$. Then, the interpolated value of the series is given by:

$$y = y_i + \left(\frac{t - t_i}{t_j - t_i}\right) \cdot (y_j - y_i) \tag{14.1}$$

This is simple linear interpolation, although other more complex methods, such as polynomial interpolation or spline interpolation, are possible. However, such methods require a larger number of data points in a time window for the estimation. In many cases, such methods do not provide significantly superior results over the straightforward linear interpolation method.

### 14.2.2 Noise Removal

Noise-prone hardware, such as sensors, are often used for time series data collection. The approach used by most of the noise removal methods is to remove *short-term fluctuations*. It should be pointed out that the distinction between noise and *interesting* outliers is often a difficult one to make. Interesting outliers are fluctuations, caused by specific aspects of the data *generation* process, rather than artifacts of the data *collection* process. Therefore, such cleaning and smoothing methods are sometimes not appropriate for problems such as outlier detection. Two methods, referred to as *binning* and *smoothing*, are often used for noise removal.

#### Binning

The method of binning divides the data into time intervals of size $k$ denoted by $[t_1, t_k], [t_{k+1}, t_{2k}]$, etc. It is assumed that the timestamps are equally spaced apart. Therefore, each bin is of the same size, and it contains an equal number of points. The average value of the data points in each interval are reported as the smoothed values. Let $y_{i \cdot k+1} \ldots y_{i \cdot k+k}$ be the values at timestamps $t_{i \cdot k+1} \ldots t_{i \cdot k+k}$. Then, the new binned value will be $y'_{i+1}$, where

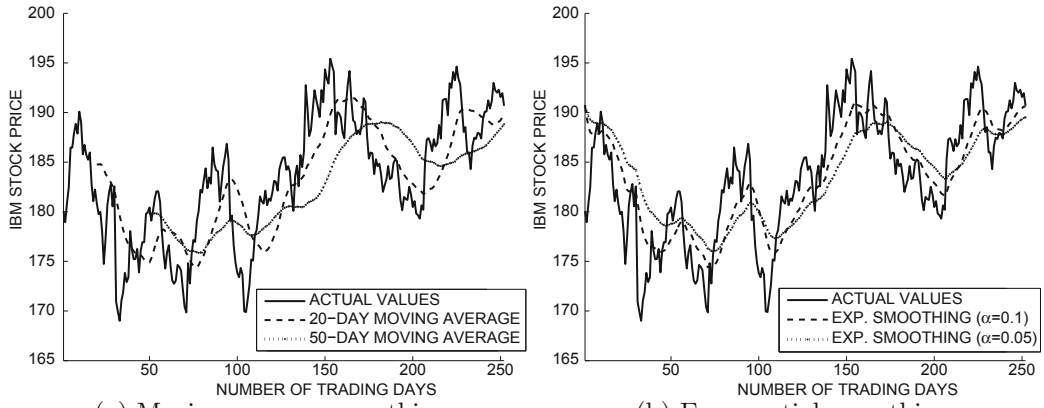
$$y'_{i+1} = \frac{\sum_{r=1}^{k} y_{i \cdot k+r}}{k}$$

Therefore, this approach uses the mean of the values in the bins. It is also possible to use the median of the behavioral attribute values. Typically, the median provides more robust estimates than the mean because the outlier points do not affect the median in a disproportionate way. The main problem with binning is that it reduces the number of available data points by a factor of $k$. Binning is also referred to as piecewise aggregate approximation (PAA). Such an approach can be rather lossy for large values of $k$, although it can also be advantageous for fast distance computations [309] because it provides a compressed representation.

#### Moving-Average Smoothing

Moving-average methods reduce the loss in binning by using overlapping bins, over which the averages are computed. As in the case of binning, averages are computed over windows of the time series. The main difference is that a bin is constructed starting at *each* timestamp in the series rather than only the timestamps at the boundaries of the bins. Therefore, the bin intervals are chosen to be $[t_1, t_k], [t_2, t_{k+1}]$, etc. This results in a set of overlapping intervals. The time series values are averaged over each of these intervals. Moving averages are also referred to as *rolling averages* and they reduce the noise in the time series because of the smoothing effect of averages.

In a real-time application, the moving average becomes available only after the *last* timestamp of the window. Therefore, moving averages introduce lags into the analysis and also lose some points at the beginning of the series because of boundary effects. Furthermore, short-term trends are sometimes lost because of smoothing. Larger bin sizes result in greater smoothing and lag. Because of the impact of lag, it is possible for the moving average to contain troughs (or downtrends) where there are peaks (or uptrends) in the original series, and vice versa. This can sometimes lead to a misleading understanding of recent trends.




(a) Moving average smoothing (b) Exponential smoothing

Figure 14.1: Various smoothing methods applied to IBM stock price from September 5, 2013 to September 4, 2014

**Exponential Smoothing**

In exponential smoothing, the smoothed value $y'_i$ is defined as a linear combination of the current value $y_i$, and the previously smoothed value $y'_{i-1}$. The smoothing parameter $\alpha \in (0, 1)$ is used for this purpose.

$$y'_i = \alpha \cdot y_i + (1 - \alpha) \cdot y'_{i-1} \tag{14.2}$$

The value of $y'_0$ is typically set to the first point in the series. When the value of $\alpha$ is 1, there are no smoothing effects, and the smoothed series is the same as the original series. When the value of $\alpha$ is 0, the entire series becomes smoothed to the constant value of $y'_0$. The approach is referred to as exponential smoothing because the value of $y'_i$ can be expressed as an exponentially decayed sum of the series values. By recursively substituting the aforementioned equation into itself, the following can be shown:

$$y'_i = (1 - \alpha)^i \cdot y'_0 + \alpha \cdot \sum_{j=1}^{i} y_j \cdot (1 - \alpha)^{i-j}. \tag{14.3}$$

The choice of $\alpha$ regulates the decay factor. Unlike moving averages, exponential smoothing provides more importance to recent data points. Data points are not lost at the beginning of the series, and the impact of the lag is reduced for the same level of smoothing. Examples of moving average and exponential smoothing are illustrated in Fig. 14.1a, b, respectively. It is evident that exponential smoothing does not lose any points at the beginning of the series and generally provides slightly better smoothing for lower lag.

### 14.2.3 Normalization

Time series typically need to be normalized, especially when multiple series are analyzed simultaneously. For example, one series might measure temperature, whereas another might measure pressure. Because these values are measured on different scales, they cannot be compared meaningfully. Therefore, two normalization methods are commonly used to adjust for such variations.

1. *Range-based normalization:* In range-based normalization, the minimum and maximum value of the time series are determined. Let these values be denoted by $min$ and $max$, respectively. Then, the time series value $y_i$ is mapped to the new value $y_i'$ in the range $(0, 1)$ as follows:
$$y_i' = \frac{y_i - min}{max - min}. \tag{14.4}$$

2. *Standardization:* In standardization, the mean and standard deviation of the series are used for normalization. This is essentially the $Z$-value of the time series. Let $\mu$ and $\sigma$ represent the mean and standard deviation of the values in the time series. Then, the time series value $y_i$ is mapped to a new value $z_i$ as follows:
$$z_i = \frac{y_i - \mu}{\sigma}. \tag{14.5}$$

Standardization is generally the preferred method. However, it does not guarantee a specific range of the time series values.

### 14.2.4 Data Transformation and Reduction

A variety of preprocessing methods exist for transforming and reducing the time series data into a reduced representation. Some of these methods transform the data into a smaller number of numeric coefficients, whereas other methods transform the data into discrete values.

#### 14.2.4.1 Discrete Wavelet Transform

The discrete wavelet transform ($DWT$) converts a time series to multidimensional data. While time series can also be considered as multidimensional data by viewing[1] the values at the different timestamps as dimensions, the values in successive timestamps are highly related to one another. A direct application of multidimensional methods ignores the temporal continuity in data values. In wavelets, the coefficients describe properties of different *contiguous temporal regions* of the series. Each coefficient is equal to half the difference in the average value of the behavioral attribute between a pair of carefully chosen contiguous segments of the series. The resulting representation can be more easily analyzed like multidimensional data because temporal locality is already incorporated within the coefficients. By using only the largest coefficients for representation, it is possible to reconstruct the entire time series accurately. Typically, the number of retained coefficients is much smaller than the length of the original time series. Thus, the approach is a dimensionality reduction method as well. $DWT$ is described in detail in Sect. 2.4.4.1 of Chap. 2.

#### 14.2.4.2 Discrete Fourier Transform

Wavelets are most effective when most of the variations in the series can be captured in specific local regions of the series. In cases where the series contain global periodicity, the discrete Fourier transform ($DFT$) is more effective. Examples of scenarios in which either of these methods would perform well are provided in Fig. 14.2. The basic idea is that any series

[1]The concept of "dimension" can be defined in two ways for time series data. Each behavioral attribute in a multivariate series can be viewed as a dimension. Alternatively, the different values in a univariate time series can be viewed as dimensions. The usage is often dependent on the semantics of the application at hand.

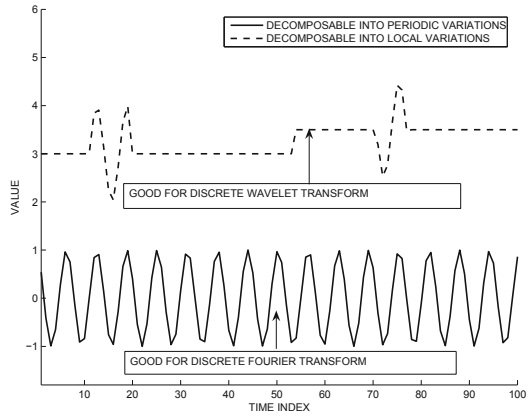



Figure 14.2: Preferred scenarios for *DFT* and *DWT*

of length $n$ can be expressed as a linear combination of smooth periodic sinusoidal series. Along with a single constant term, the $n-1$ sinusoidal series have periodicity drawn from $n, n/2, n/3, \ldots n/(n-1)$. The data can be reduced using this decomposition because only a small number of these constituent series have large enough contributions to be included. Consider a time series $x_0 \ldots x_{n-1}$. Each coefficient $X_k$ of the Fourier transform is a *complex* value which is defined as follows:

$$X_k = \sum_{r=0}^{n-1} x_r \cdot e^{-ir\omega k} = \sum_{r=0}^{n-1} x_r \cdot \cos(r\omega k) - i \sum_{r=0}^{n-1} x_r \cdot \sin(r\omega k) \quad \forall k \in \{0 \ldots n-1\}. \tag{14.6}$$

Here, $\omega$ is set to $\frac{2\pi}{n}$ radians, and the notation $i$ denotes the imaginary number $\sqrt{-1}$. Therefore, $X_k$ is a complex value. One property of the Fourier coefficients is that $X_{n-k}$ can be derived from $X_k$ by flipping the sign of the imaginary part for $k \geq 1$ (see Exercise 7). Therefore, only the first $n/2$ complex coefficients need to be retained. Furthermore, only the coefficients $X_k = a_k + ib_k$ with large energy $a_k^2 + b_k^2$ need to be retained. The top-$m$ retained coefficients (together with their index $k$) can be used to approximate the time series in a compact way. Both the real and imaginary parts of the coefficients can be stored in a real-valued vector data structure. This vector provides the reduced representation of the series. The original series can be reconstructed from the coefficients as follows:

$$x_r = \frac{1}{n} \sum_{k=0}^{n-1} X_k \cdot e^{ir\omega k} = \frac{1}{n} \left( \sum_{k=0}^{n-1} X_k \cdot \cos(r\omega k) + i \sum_{k=0}^{n-1} X_k \cdot \sin(r\omega k) \right) \quad \forall r \in \{0 \ldots n-1\}.$$

Note that each $X_k$ is a complex value. However, the imaginary part of the right-hand side of this equation will always evaluate to zero to yield the real series value $x_r$.

*DFT* has several properties, which make it useful for data mining applications. It satisfies the *additivity property*; the Fourier coefficients of the sum (or difference) of two series can be obtained as the sum (or difference) of their Fourier coefficients. It also satisfies Parseval's theorem, which states that if $X_k = a_k + ib_k$ is the $k$th Fourier coefficient, then we have $\sum_{r=0}^{n-1} x_r^2 = \frac{1}{n} \sum_{k=0}^{n-1} (a_k^2 + b_k^2)$. Because of these properties, one can compute the (scaled) Euclidean distance between two time series by computing the Euclidean distance between their Fourier coefficients. Like *DWT*, *DFT* can also be viewed as the transformation of the time series to a new (rotated) orthogonal basis system, except that each basis vector $\overline{B_k} =$

$[1, e^{i\omega k}, e^{2i\omega k}, \ldots, e^{(n-1)i\omega k}]$ of the Fourier coefficient $X_k$ is a complex vector. Therefore, the time series may be decomposed in terms of the mutually orthogonal basis vectors $\overline{B_0} \ldots \overline{B_{n-1}}$ as follows:

$$(x_0 \ldots x_{n-1}) = \frac{1}{n} \sum_{k=0}^{n-1} X_k \overline{B_k} \tag{14.7}$$

Typically, off-the-shelf mathematical packages are available to compute the coefficients with the use of the fast Fourier transform (FFT). A closely related transform, known as the discrete cosine transform (DCT), provides even better compression.

#### 14.2.4.3 Symbolic Aggregate Approximation (SAX)

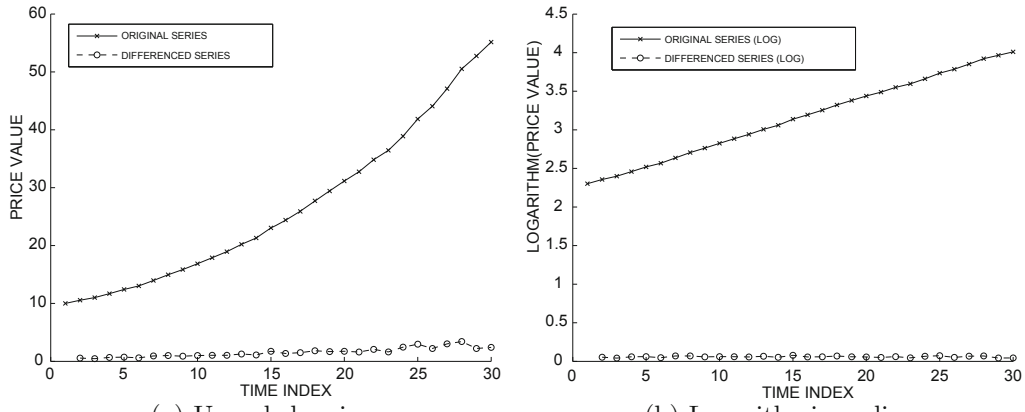
This approach converts a time series to discrete sequence data. The basic idea is to determine piecewise aggregate approximates by averaging behavioral attribute values over successive and equally-spaced windows of the time series. The resulting continuous values are then discretized into a small number of discrete values. Depending on the application, the number of breakpoints may vary between 3 and 10. The approach selects the break points of the discretization, so that each of the symbolic values has an approximately equal frequency of representation. One possibility is to use equi-depth discretization of the continuous values, though this can be impractical or infeasible for long series or streaming series. For long series or streaming series, a Gaussian distribution assumption of the resulting averages is used to determine the discretization breakpoints. The idea is to select points on the Gaussian curve, so that the area between successive breakpoints is equal, and therefore the different symbols have approximately the same frequency.

### 14.2.5 Time Series Similarity Measures

Time series similarity measures are typically designed with application-specific goals in mind. The most common methods for time series similarity computation are Euclidean distance and dynamic time warping ($DTW$). The Euclidean distance is defined in an identical way to multidimensional data where the behavioral attribute values at the different timestamps are interpreted as dimensions. The Euclidean distance can be used only when the two series have the same length, and a one-to-one correspondence exists between the data points. This is not appropriate in unsynchronized time series where the data may be generated at different rates over different portions of the time series. The $DTW$ method stretches and shrinks the time dimension differently in different portions of one of the series to create an optimal matching. As discussed in Sect. 16.3.4.1 of Chap. 16, $DTW$ can also be extended to multivariate time series such as trajectory data. Two other similarity/distance functions include the Edit Distance and the Longest Common Subsequence. These measures are used more commonly for discrete sequences, rather than continuous time series. All these measures are described in detail in Sect. 3.4.1 of Chap. 3.

## 14.3 Time Series Forecasting

Forecasting is one of the most common applications of time series analysis. The prediction of future trends has applications in retail sales, economic indicators, weather forecasting, stock markets, and many other application scenarios. In this case, we have one or more series of data values, and it is desirable to predict the future values of the series using the history of previous values.




(a) Unscaled series (b) Logarithmic scaling

Figure 14.3: Impact of different operations on stationary and non-stationary series

Time series can be either *stationary* or *nonstationary*. A stationary stochastic process is one whose parameters, such as the mean and variance, do not change with time. A nonstationary process is one whose parameters change with time. Some kinds of time series such as white noise are stationary. White noise is the strongest form of stationarity with zero mean, constant variance, and zero covariance between series values separated by a fixed lag. On the other hand, consider the case, where the behavioral attribute corresponds to the price level of an industrial commodity such as crude oil. This is typically nonstationary because the average price level may increase over time as a result of inflation. In fact, most time series in real applications are nonstationary. A stationary series will usually be characterized as a noisy series with a level trend, constant variance, and zero covariance between different series values. For example, in Fig. 14.3a, both the series are nonstationary because the average values increase with time. On the other hand, in Fig. 14.3b, the dashed curve is stationary because the trends do not change significantly with time. A *strictly stationary* time series is defined as follows:

**Definition 14.3.1 (Strictly Stationary Time Series)** *A strictly stationary time series is one in which the probabilistic distribution of the values in any time interval* $[a, b]$ *is identical to that in the shifted interval* $[a + h, b + h]$ *for any value of the time shift* $h$.

In other words, all multivariate distributions of subsets of variables must match with their shifted counterparts. The window-based statistical parameters of a stationary time series can be estimated in a meaningful way because the parameters do not vary over different time windows. In such cases, the estimated statistical parameters are good predictors of future behavior. On the other hand, the *current* mean, variances, and statistical correlations of the series are not necessarily good predictors of *future* behavior in regression-based forecasting models for nonstationary series. Therefore, it is often advantageous to convert nonstationary series to stationary ones before forecasting analysis. After the forecasting has been performed on the stationary series, the predicted values are transformed back to the original representation, using the inverse transformation. The strict stationarity concept of Definition 14.3.1 is, however, too restrictive to be meaningfully used in real applications. For example, it is difficult even to determine whether or not a time series is strictly stationary from a single instance because one must comprehensively characterize all multivariate distributions of subsets of variables.

A key observation is that it is much easier to either obtain or convert to series that exhibit *weak* stationarity properties. In such cases, unlike white noise, the mean of the series, and the covariance between approximately adjacent time series values may be non-zero but constant over time. This is referred to as *covariance stationarity*. This kind of weak stationarity can be assessed relatively easily and is also useful for forecasting models that are dependent on specific parameters such as the mean and covariance. In other nonstationary series, the average value of the series can be described by a *trend-line* that is not necessarily horizontal, as required by a stationary series. Periodically, the series will deviate from the trend line, possibly because of some changes in the generative process, and then return to the trend line. This is referred to as a *trend stationary* series. Such weak forms of stationarity are also very useful for creating effective forecasting models. In the following, some practical methods that are commonly used to convert nonstationary series to stationary series will be discussed.

**Differencing**

A common approach used for converting time series to stationary forms is differencing. In differencing, the time series value $y_i$ is replaced by the difference between it and the previous value. Therefore, the new value $y'_i$ is as follows:

$$y'_i = y_i - y_{i-1}. \tag{14.8}$$

If the series is stationary after differencing, then an appropriate model for the data is:

$$y_{i+1} = y_i + e_{i+1}. \tag{14.9}$$

Here, $e_{i+1}$ corresponds to white noise with zero mean. A differenced time series would have $t-1$ values for a series of length $t$ because it is not possible for the first value to be reflected in the transformed series.

Higher order differencing can be used to achieve stationarity in second order changes. Therefore, the higher order differenced value $y''_i$ is defined as follows:

$$y''_i = y'_i - y'_{i-1} \tag{14.10}$$

$$= y_i - 2 \cdot y_{i-1} + y_{i-2} \tag{14.11}$$

This model allows the series to drift over time, since the noise has non-zero mean. The corresponding model is as follows:

$$y_{i+1} = y_i + c + e_{i+1}. \tag{14.12}$$

Here, $c$ is a non-zero constant that accounts for the drift. Generally, it is rare to use differences beyond the second order.

A different approach is to use seasonal differences when it is known that the series is stationary after seasonal differencing. The seasonal differences are defined as follows:

$$y'_i = y_i - y_{i-m} \tag{14.13}$$

Here $m$ is an integer greater than 1.

In some cases, such as geometrically increasing series, the logarithm function is applied to the values in the series, before the differencing operation. For example, consider a time series of prices that increase at an approximately constant inflation factor. In such cases,

it may be useful to apply the logarithm function to the time series values, before the differencing operation. An example is provided in Fig. 14.3a, where the variation in inflation is illustrated with time. It is evident that the differencing operation does not help in making the series stationary. In Fig. 14.3b, the logarithm function is applied to the series before the differencing operation. In this case, the series becomes stationary after the differencing operation.

In the following, a number of univariate time series forecasting models will be discussed. These models work effectively under different assumptions on the time series patterns. Some of these models assume a stationary time series, whereas others do not.

### 14.3.1 Autoregressive Models

Univariate time series contain a single variable that is predicted using *autocorrelations*. Autocorrelations represent the correlations between adjacently located timestamps in a series. Typically, the behavioral attribute values at adjacently located timestamps are positively correlated. The autocorrelations in a time series are defined with respect to a particular value of the lag $L$. Thus, for a time series $y_1, \ldots y_n$, the autocorrelation at lag $L$ is defined as the Pearson coefficient of correlation between $y_t$ and $y_{t+L}$.

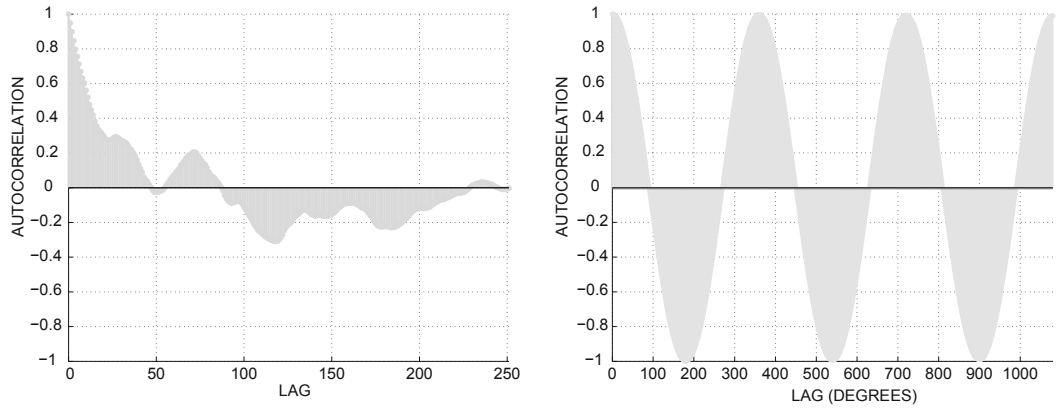
$$\text{Autocorrelation}(L) = \frac{\text{Covariance}_t(y_t, y_{t+L})}{\text{Variance}_t(y_t)}. \tag{14.14}$$

The autocorrelation always lies in the range $[-1, 1]$, although the value is almost always positive for very small values of $L$, and gradually drops off with increasing lag $L$. The positive correlation is a result of the fact that adjacent values of most time series are very similar, though the similarity drops off with increasing distance. High (absolute) values of the autocorrelation imply that the value at a given position in the series can be predicted as a function of the values in the immediately preceding window. This is, in fact, the key property that enables the use of the autoregressive model. For example, the variation in autocorrelation with lag for the IBM stock example (Fig. 14.1) is illustrated in Fig. 14.4a. Such a figure is referred to as the *autocorrelation plot* and is used commonly in AR models. While the autocorrelation is usually positive and falls off with lag, the precise behavior is highly application-specific. For periodic series, the autocorrelation may be periodic and negative at certain lag intervals. An example of the autocorrelations for a periodic sine wave is illustrated in Fig. 14.4b.

In the autoregressive model, the value of $y_t$ at time $t$ is defined as a linear combination of the values in the immediately preceding window of length $p$.

$$y_t = \sum_{i=1}^{p} a_i \cdot y_{t-i} + c + \epsilon_t \tag{14.15}$$

A model that uses the preceding window of length $p$ is referred to as an $AR(p)$ model. The values of the regression coefficients $a_1 \ldots a_p, c$ need to be learned from the training data. The larger the value of $p$, the greater the lag that one is willing to incorporate in the autocorrelations. The choice of $p$ should be guided by the level of autocorrelation of Eq. 14.14. Because the autocorrelation often reduces with increasing values of the lag $L$, a value of $p$ should be selected, so that the autocorrelation at lag $L = p$ is small. In such cases, increasing the window of regression further may not help the accuracy of the modeling process, and may sometimes result in overfitting. Typically, the autocorrelation plot (Fig. 14.4) is used to identify the window. Instead of using a window of coefficients in




(a) IBM stock (b) Sine wave

Figure 14.4: Autocorrelation plots for various series

Eq. 14.15, it is also possible to select coefficients with specific lag values. In particular, lag values with high absolute autocorrelation in the autocorrelation plot may be selected. Such an approach is also helpful for forecasting periodic series.

Each timestamp in the past history of the time series data creates a linear equation between the time series variables. A set of linear equations between the coefficients can be created by using the value at each timestamp in the training data, along with its immediately preceding window of length $p$. When the number of timestamps available is much larger than $p$, this is an over-determined system of equations, which is infeasible. Therefore, any (infeasible) solution will have an error associated with it. The coefficients $a_1, \ldots a_p, c$ can be approximated with *least-squares regression*, to minimize the square-error of the over-determined system (cf. Sect. 11.5 of Chap. 11). Note that the model can be used effectively for forecasting *future* values, only if the key properties of the time series, such as the mean, variance, and autocorrelation do not change significantly with time. Many off-the-shelf commercial solvers are available for these models. The effectiveness of the forecasting model may be quantified by using the noise level in the estimated coefficients. Specifically, the $R^2$-value, which is also referred to as the *coefficient of determination*, measures the ratio of the white noise to the series variance:

$$R^2 = 1 - \frac{\text{Mean}_t(\epsilon_t^2)}{\text{Variance}_t(y_t)} \tag{14.16}$$

The coefficient of determination quantifies the fraction of variability in the series that is explained by the regression, as opposed to random noise. It is therefore desirable for this coefficient to be as close to 1 as possible.

### 14.3.2 Autoregressive Moving Average Models

While autocorrelation is a useful predictive property of time series, it does not always explain all the variations. In fact, the unexpected component of the variations (shocks), does impact future values of the time series. This component can be captured with the use of a moving average model (MA). The autoregressive model can therefore be made more robust by combining it with an MA. Before discussing the autoregressive moving average model (ARMA), the MA will be introduced.

The moving average model predicts subsequent series values on the basis of the past history of deviations from predicted values. A deviation from a predicted value can be viewed as white noise, or a shock. This model is best used in scenarios where the behavioral attribute value at a timestamp is dependent on the history of shocks in the time series, rather than the actual series values. The moving average model is defined as follows:

$$y_t = \sum_{i=1}^{q} b_i \cdot \epsilon_{t-i} + c + \epsilon_t$$

The aforementioned model is also referred to as $MA(q)$. The parameter $c$ is the mean of the time series. The values of $b_1 \ldots b_q$ are the coefficients that need to be learned from the data. The moving average model is quite different from the autoregressive model, in that it relates the current value to the mean of the series and the previous history of *deviations from forecasts*, rather than the actual values. Here the values of $\epsilon_t$ are assumed to be white noise error terms that are uncorrelated with one another. A problem here is that the error terms $\epsilon_t$ are not part of observed data, but also need to be derived from the forecasting model. This circularity implies that the system of equations is inherently nonlinear when expressed purely in terms of the coefficients and the observed values $y_i$. Typically, iterative nonlinear fitting procedures are used instead of the linear least-squares approach to determine a solution to the moving average model. It is rare that the series values can be predicted in terms of *only* the shocks, and not the autocorrelations. Autocorrelations are extremely important in time series analysis because of the inherent temporal continuity of time series data. At the same time, the history of shocks do impact the future values of the series. Therefore, neither the autoregressive nor the moving average model can fully capture all the correlations needed for forecasting in isolation.

A more general model may be obtained by combining the power of both the autoregressive model and the moving average model. The idea is to learn the appropriate impact of *both* the autocorrelations and the shocks in predicting time series values. The two models can be combined with $p$ autoregressive terms and $q$ moving average terms. This model is referred to as the $ARMA$ model. In this case, the relationships between the different terms may be expressed as follows:

$$y_t = \sum_{i=1}^{p} a_i \cdot y_{t-i} + \sum_{i=1}^{q} b_i \cdot \epsilon_{t-i} + c + \epsilon_t$$

The aforementioned model is the $ARMA(p, q)$ model. A key question here is about the choice of the parameters $p$ and $q$ in these models. If the values of $p$ and $q$ are set to be too small, then the model will not fit the data well. On the other hand if the values of $p$ and $q$ are set to be too large, then the model is likely to overfit the data. In general, it is advisable to select the values of $p$ and $q$ as small as possible, so that the model fits the data well. As in the previous case, autoregressive moving average models are best used with stationary data.

In many cases, nonstationary data can be addressed by combining differencing with the autoregressive moving average model. This results in the *autoregressive integrated moving average model (ARIMA)*. In principle, differences of any order may be used, although first- and second-order differences are most commonly used. Consider the case where the first order differenced value $y'_t$ is used. Then, the $ARIMA$ model can be expressed as follows:

$$y'_t = \sum_{i=1}^{p} a_i \cdot y'_{t-i} + \sum_{i=1}^{q} b_i \cdot \epsilon_{t-i} + c + \epsilon_t$$

Thus, this model is virtually identical to the $ARMA(p, q)$ model, except that differencing is used within the model. If the order of the differencing is $d$, then this model is referred to as the $ARIMA(p, d, q)$ model.

### 14.3.3 Multivariate Forecasting with Hidden Variables

All the aforementioned models are designed for a single time series. In practice, a given application may have thousands of time series, and there may be significant correlations both across different series and across time. Therefore, models are required that can combine the autoregressive correlations with the cross-series correlations for making forecasts.

While there are many different ways of multivariate forecasting, hidden variables are often used to achieve this goal. This is because the hidden variable approach is able to cleanly separate out the cross-series correlations from the autoregressive correlations in the modeling process. The idea in hidden variable modeling is to transform the large number of cross-correlated time series into a small number of uncorrelated time series. Typically, principal component analysis (PCA) is used for this transformation. Because these different series are uncorrelated with one another, it is possible to use any of the $AR$, $ARMA$ or $ARIMA$ models individually on the series to predict the hidden values. Then, the predicted values are mapped back to their original representation. This provides the forecasted values for all the different series with the use of a small number of hidden variable predictions. Readers are advised to revisit Sect. 2.4.3.1 of Chap. 2 for the discussion on $PCA$ before reading further.

It is assumed that there are $d$ synchronized time series of length $n$. The $d$ different time series values received at the $i$th timestamp are denoted by $\overline{Y_i} = (y_i^1 \ldots y_i^d)$. The goal is to predict $\overline{Y_{n+1}}$ from $\overline{Y_1} \ldots \overline{Y_n}$. The steps of the multivariate forecasting approach are as follows:

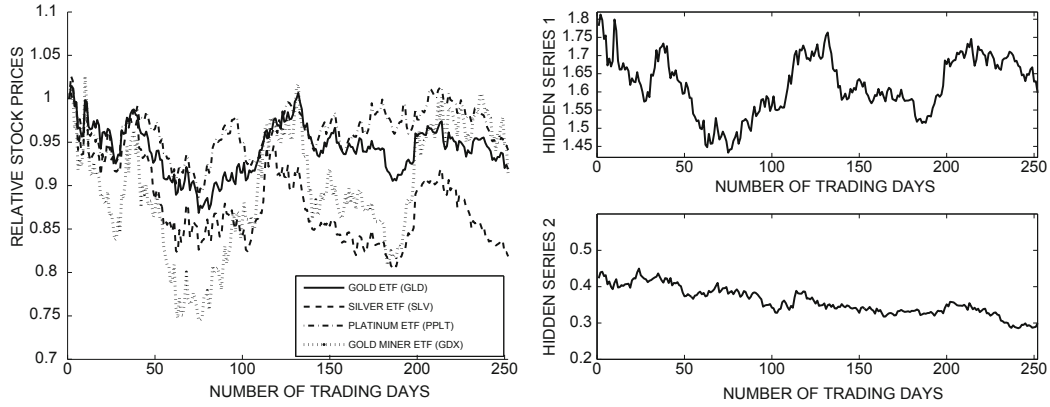
1. Construct the $d \times d$ covariance matrix of the multidimensional time series. Let the $d \times d$ covariance matrix be denoted by $C$. The $(i, j)$th entry of $C$ is the covariance between the $i$th and $j$th series. This step is identical to the case of multidimensional data, and the temporal ordering among the different values of $\overline{Y_i}$ is not used at this stage. Thus, the covariance matrix only captures information about correlations across series, rather than correlations across time. Note that covariance matrices can also be maintained incrementally in the streaming setting, using an approach discussed in Sect. 20.3.1.4 of Chap. 20.

2. Determine the eigenvectors of the covariance matrix $C$ as follows:

$$C = P \Lambda P^T \tag{14.17}$$

   Here, $P$ is a $d \times d$ matrix, whose $d$ columns contain the orthonormal eigenvectors. The matrix $\Lambda$ is a diagonal matrix containing the eigenvalues. Let $P_{truncated}$ be a $d \times p$ matrix obtained by selecting the $p \ll d$ columns of $P$ with the largest eigenvalues. Typically, the value of $p$ is much smaller than $d$. This represents a basis for the hidden series with the greatest variability.

3. A new multivariate time series with $p$ hidden time series variables is created. Each $d$-dimensional time series data point $\overline{Y_i}$ at the $i$th timestamp is expressed in terms of a $p$-dimensional hidden series data point. This is achieved by using the $p$ basis vectors




(a) Correlated stock prices (b) Uncorrelated hidden variables

Figure 14.5: Normalized prices of four precious metal exchange traded funds (*ETFs*) from September 5, 2013 to September 4, 2014 and corresponding uncorrelated hidden variables

derived in the previous step. Therefore, the $p$-dimensional hidden value $\overline{Z_i} = (z_i^1 \ldots z_i^p)$ is derived as follows:

$$\overline{Z_i} = \overline{Y_i} P_{truncated} \tag{14.18}$$

The value of $\overline{Z_i}$ represents the $p$ different values for the hidden series variables at the $i$th timestamp. Thus, this step creates $p$ different hidden variable time series that are approximately independent of one another. Note that the other $(d-p)$ hidden variables in $\overline{Y_i}P$ are approximately constant over time because of their small eigenvalues (variance). The means of these $(d-p)$ approximately constant values are noted as well. No predictive modeling is required for the vast majority of these hidden variables with constant values. In Fig. 14.5a, the stock prices of four precious metal-related exchange traded funds (ETFs) are illustrated for a period of 1 year. Each series was multiplicatively scaled to a relative value starting at 1. The top two hidden variable series are illustrated in Fig. 14.5b. Note that these derived series are uncorrelated and the first hidden variable has much higher variance than the second. The remaining two hidden variables are not shown because their variance is even smaller. In fact, each of the four correlated series in Fig. 14.5a can be approximately expressed as a different linear combination of the two hidden-variable series in Fig. 14.5b. Therefore, forecasting the hidden variables yields approximate forecasts of the original series.

4. For each of the $p$ uncorrelated and high-variance series, use any univariate forecasting model to predict the values of the $p$ hidden variables at the $(n+1)$th timestamp. A univariate approach can be used effectively because the different hidden variables are uncorrelated by design. This provides a set of values $\overline{Z_{n+1}} = (z_{n+1}^1 \ldots z_{n+1}^p)$. Append the means of the approximately constant values of the remaining $(d-p)$ hidden series to $\overline{Z_{n+1}}$ to create a new $d$-dimensional hidden variable vector $\overline{W_{n+1}}$.

5. Transform back the predicted hidden variables $\overline{W_{n+1}}$ to the original $d$-dimensional representation by using the reverse transformation. This provides the forecasted values of the original series:

$$\overline{Y_{n+1}} = \overline{W_{n+1}} P^T \tag{14.19}$$

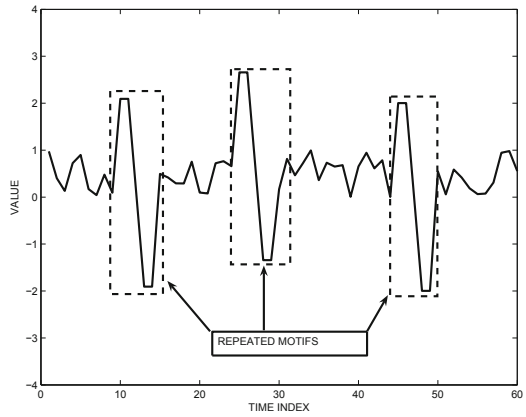



Figure 14.6: Repeated motif in a single time series

The aforementioned description is a simplified version of the *SPIRIT* framework. It reduces the computational effort of prediction because simplified univariate modeling is performed only on a small number $p \ll d$ of independent time series. On the other hand, it does incur the overhead of computing eigenvectors. The hidden-variable series is a linear combination of many different series. Therefore, the noise effects of individual series are often smoothed out within the hidden variables, which increases the robustness of the forecasting process.

## 14.4 Time Series Motifs

A *motif* is a frequently occurring pattern or shape in the time series. Motif discovery can be formulated in a wide variety of ways, depending on application-specific requirements. These different formulations vary in terms of the input data and the nature of the motifs discovered. These variations are as follows:

1. *Single series versus database of many series:* In the first case, a single series is available, and the frequently occurring shapes in specific windows of the series are determined. For example, in Fig. 14.6, the highlighted shape appears three times in the same series and therefore has a count of 3. A different formulation is one in which we have $N$ different series, and the occurrence of a shape *at least once* in a particular series is given a credit of exactly 1. The frequency is therefore computed in terms of the number of series in which the pattern occurs. The second formulation is much closer to sequential pattern mining in discrete data. Different applications may require different definitions of motif discovery.

2. *Contiguous versus noncontiguous motifs:* Contiguous motifs require that the shapes are discovered over a contiguous window of the time series. Noncontiguous motifs may allow gaps between different elements of the motif. Much of the work in time series analysis assumes that the motifs are defined over contiguous windows. Non-contiguous motifs are more common in discrete sequence analysis. Nevertheless, noncontiguous motifs may have utility in some applications.

3. *Multigranularity motifs:* Many formulations fix the window size in which the motifs are discovered. However, in practice, the frequent motifs may occur over windows of

different sizes. Such motifs are very useful in many application-specific scenarios. For example, in the financial-market series of Fig. 14.11a, an important motif is caused by a "flash crash" event over the course of a day. On the other hand, in Fig. 14.11b, the (recessionary) trend occurs over several months. In the second case, it may be needed to smooth out the local variations to discover motifs. Thus, different techniques are required to discover different types of motifs.

When does a motif belong to a time series? Two methods are typically used by different applications.

1. *Distance-based support:* A particular segment of a sequence is said to support a motif when the distance between the segment and the motif is less than a particular threshold.

2. *Transformation to sequential pattern mining:* A variety of discretizations can be used to convert time series into discrete sequences. After the conversion, motifs can be defined as discrete subsequences of the sequences.

The latter method lends itself to richer classes of algorithms from sequential pattern mining. Furthermore, the approach used for discretization can be varied to discover motifs of different kinds. It also allows the discovery of noncontiguous patterns because sequential pattern mining algorithms do not assume contiguity by default. This section will discuss both kinds of methods. In addition, the notion of periodic patterns will be introduced.

### 14.4.1 Distance-Based Motifs

Distance-based motifs are always defined on *contiguous* segments of the time series. First, the concept of approximate distance match between a motif and a contiguous segment in a time series needs to be defined.

**Definition 14.4.1 (Approximate Distance Match)** *A sequence (or motif) $S = s_1 \ldots s_w$ of real values is said to approximately match a contiguous subsequence of length $w$ in the time series $(y_1, \ldots y_n)$ $(w \leq n)$ starting at position $i$, if the distance between $(s_1, \ldots s_w)$ and $(y_i, \ldots y_{i+w-1})$ is at most $\epsilon$.*

A wide variety of distance functions may be used, and the Euclidean distance function is a common choice. The aforementioned definition assumes that the two subsequences being matched have the same length. This is a conservative assumption that allows the use of distance functions such as the Euclidean function. However, if other distance functions, such as dynamic time warping, are used, it may not be necessary for both the matched motifs to have the same length.

The number of occurrences of the motif in a single long series is used to quantify the frequency of the motif. In addition to the series itself, the window length $w$ and the approximation threshold $\epsilon$ are the two main inputs to the algorithm.

**Definition 14.4.2 (Motif Count)** *The number of matches of a time series window $S = s_1 \ldots s_w$ to the time series $(y_1 \ldots y_n)$ at threshold level $\epsilon$, is equal to the number of windows of length $w$ in $(y_1 \ldots y_n)$, for which the distance between the corresponding subsequences is at most $\epsilon$.*

The goal is to discover the top $k$ motifs for a user-defined parameter $k$. Furthermore, to ensure that the $k$ motifs discovered are very different from one another, a constraint is

```
Algorithm FindBestMotif(Time Series: y_1 ... y_n, Window: w
            Distance Threshold: ε)
begin
  for i = 1 to n − w + 1 do begin
    Candidate-Motif= (y_i, ... y_{i+w−1});
    for j = 1 to n − w + 1 do begin
      Comparison-Motif= (y_j ... y_{j+w−1});
      D = ComputeDistance(Candidate-Motif, Comparison-Motif);
      if (D ≤ ε) and (non-trivial match)
                then increment count of Candidate-Motif by 1;
    endfor
    if Candidate-Motif has highest count found so far
      then update Best-Candidate to Candidate-Motif;
  endfor
  return Best-Candidate;
end
```

Figure 14.7: Determining the most frequent motif

imposed; the distances between any pairs of motifs discovered among the top-$k$ motifs must be at least $2 \cdot \epsilon$. In the following, the discovery of the most frequent occurring single motif will be described. The generalization to the case of top $k$ motifs is relatively straightforward. The overall approach [356] uses a nested-loop algorithm to discover the most frequent motif. The approach is described in Fig. 14.7.

The approach extracts all of the candidate motifs of length $w$ from a time series and computes the distances to all of the windows of length $w$. The number of windows over which the match occurs is counted. Care is taken to exclude *trivial matches* in the count. Trivial matches are defined as those matches where approximately the same (overlapping) window is being matched. For example, the case when $i = j$ is a trivial match. Furthermore, in the case where $i < j$, if the window starting at $i$ matches with all windows starting at $i+1, i+2 \ldots j$, then the match at $j$ is trivial as well. In the case where $i > j$, if the window starting at $i$ matches with all windows starting at $i-1, i-2 \ldots j$, then the match at $j$ is trivial. Therefore, this condition is explicitly checked in the counting. The best candidate is tracked over the course of the algorithm, and reported at termination. As evident from Fig. 14.7, the approach requires a nested loop, and the number of iterations in each loop is almost equal to the size of the series $n$. Thus, the approach requires $O(n^2)$ distance computations. In principle, any time series distance function, such as $DTW$, can be used for the computation, although it is generally more expensive.

The majority of the time is spent in distance computations. In many cases, a fast computation of the lower bound on the distance can be used to speed up the approach. If the computed lower bound between a pair of windows is greater than $\epsilon$, then the pair is guaranteed to be irrelevant for adding to the candidate motif count. Therefore, the distance computation does not need to be explicitly performed. The piecewise aggregate approximation (PAA) can be used to speed up the distance computations. Consider a scenario where the PAA has been performed over windows of length $m$. The resulting series has been compressed by a factor of $m$, and therefore the distance computations are much faster. If the

series $X'$ be the PAA of $X = (x_1 \ldots x_n)$, and $Y'$ be the PAA of $Y = (y_1 \ldots y_n)$, then it can be shown that:

$$Dist(X, Y) \geq \sqrt{m} \cdot Dist(X', Y') \tag{14.20}$$

The proof of this result is as follows. Consider the time series $Z = X - Y$. Over any window of $m$ data points, the second moment of elements of $Z$ in that window, is at least[2] equal to $m$ times the square of the mean of the same elements. Other faster methods for approximation exist, such as the use of the SAX representation. When the SAX representation is used, a table of precomputed distances can be maintained for all pairs of discrete values, and a simple table lookup is required for lower bounding. Furthermore, some other time series distance functions such as dynamic time warping can also be bounded from below. The bibliographic notes contain pointers to some of these bounds. Many variations of the basic approach are possible by adding another layer of nesting, which accounts for variations in the window size.

### 14.4.2 Transformation to Sequential Pattern Mining

A particularly convenient method for discovering motifs in time series is to transform the problem to the sequential pattern mining problem. The setting for this case is somewhat different, where a database of $N$ series is available, and it is desired to determine all frequent motifs at a specified minimum support level. Since motif (pattern) mining is more naturally defined in the discrete case, this transformation facilitates the use of a wide variety of tools available for the discrete scenario. Furthermore, such an approach can also enable the discovery of noncontiguous patterns in the time series. This is because the subsequences in sequential pattern mining are allowed to be noncontiguous.

The first step is to convert the time series into discrete sequences, by discretizing the behavioral attribute value at each timestamp into categorical values. It is possible to combine discretization with binning to create a robust sequence representation. It should be pointed out that there are many different ways of converting a time series to discrete sequences, depending on application-specific goals. For example, the discretization of the difference of the behavioral attribute values between successive timestamps is equivalent to using discretized wavelet coefficients of the most detailed level of granularity. Lower order wavelet coefficients will provide insights into trends over larger segments of the time series. Thus, it is even possible to perform multiresolution motif analysis by using discretized wavelet coefficients of different orders, and creating separate base sequences for wavelets of each order. In general, the approach for converting time series to discrete sequences will heavily influence the nature of the motifs found.

For all these methods, the final result of the discretization is a sequence of discrete values for each of the $N$ time series in the database. After this new database of sequences has been constructed, any sequential pattern mining algorithm can be applied. The *GSP* algorithm is described in Sect. 15.2 of Chap. 15. It is important to note that the algorithms in Chap. 15 allow gaps between successive elements of the sequence. However, these algorithms can be trivially generalized to the contiguous case, by adding a maximum gap constraint in the sequential pattern mining algorithm. Constrained sequential pattern mining algorithms are briefly discussed in Sect. 15.2.2 of Chap. 15. It should be pointed out that the different constraints discussed in Sect. 15.2.2 correspond to different kinds of motifs. Because of the wide variation in the kinds of motifs that can be found by varying either the discretization approach or the sequential pattern mining approach, this methodology is very flexible, and it can be tailored to many different application scenarios.

[2] The mean of the squares is always no less than the square of the mean for any set of numeric elements. The difference between the two is equal to the variance, which is always nonnegative.

### 14.4.3 Periodic Patterns

Just as *DWT* is used for discovering *local* patterns in a time series, *DFT* is often used for discovering *periodic* patterns. Recall from Sect. 14.2.4.2 that the $r$th component of a time series $x_0 \ldots x_{n-1}$ can be expressed in terms of $n$ complex Fourier coefficients $X_0 \ldots X_{n-1}$ as follows:

$$x_r = \frac{1}{n}\left(\sum_{k=0}^{n-1} X_k \cdot \cos(r\omega k) + i \sum_{k=0}^{n-1} X_k \cdot \sin(r\omega k)\right) \quad \forall r \in \{0 \ldots n-1\}$$

Here $\omega$ is set to $\frac{2\pi}{n}$ radians. Since the imaginary part of this summation is always 0 for real values of $x_r$, let us expand the real part by assuming $X_k = a_k + ib_k$:

$$x_r = \frac{1}{n}\left(\sum_{k=0}^{n-1} (a_k + ib_k) \cdot \cos(r\omega k) + i \sum_{k=0}^{n-1} (a_k + ib_k) \cdot \sin(r\omega k)\right) \quad \forall r \in \{0 \ldots n-1\}$$

By ignoring the imaginary part, we obtain:

$$\begin{aligned} x_r &= \frac{1}{n}\left(\sum_{k=0}^{n-1} a_k \cdot \cos(r\omega k) - \sum_{k=0}^{n-1} b_k \cdot \sin(r\omega k)\right) \quad \forall r \in \{0 \ldots n-1\} \\ &= \frac{1}{n}\sqrt{a_k^2 + b_k^2} \cdot \sum_{k=0}^{n-1} \cos(r\omega k + \theta_k) \quad \forall r \in \{0 \ldots n-1\} \end{aligned}$$

Here, we have $\theta_k = \cos^{-1}\left(\frac{a_k}{\sqrt{a_k^2+b_k^2}}\right)$. All terms with $k \geq 1$ are periodic. In other words, *the time series can be decomposed into $n-1$ periodic sinusoidal components, so that the $k$th component has a periodicity of $\frac{n}{k}$ and amplitude of $\sqrt{a_k^2 + b_k^2}$.* Therefore, if a periodic component has very high amplitude relative to other components, the entire series will be dominated by its periodic behavior. In order to detect such components, the mean and standard deviation of all the $n$ amplitudes are determined. Any amplitude $\sqrt{a_k^2 + b_k^2}$, which is at least $\delta$ standard deviations greater than the mean is flagged. Such a component has periodicity $\frac{n}{k}$, and its periodicity will be apparent in the series because of its high amplitude. Note that the smaller Fourier coefficients are also discarded in the case of dimensionality reduction. However, when the threshold $\delta$ is chosen more aggressively (i.e., very large positive values such as 3), only 2 or 3 coefficients remain, and the periodicity of the residual series becomes apparent. Furthermore, only values of $k \in (\beta, \frac{n}{\alpha})$ are relevant for discovering patterns that have period at least $\alpha \geq 2$ and have appeared at least $\beta \geq 2$ times in the series. The bibliographic notes contain pointers to methods for discovering *partial* periodic patterns.

## 14.5 Time Series Clustering

Time series data clustering can be defined in two different ways, depending on the application-specific scenario.

1. In the first approach, real-time clustering is performed on time series that are received simultaneously in time. For example, in a financial market application, it may be desirable to segment the series into groups that coevolve over time. In this case, the

values in the different time series are compared to one another in an approximately synchronized way. Typically, the analysis is performed on a small window of the recent history. The time series are clustered into groups based on correlations between series in the window. Furthermore, the clustering is performed in online fashion, and the different series may move across different clusters. For example, a stock ticker for IBM may move along with Microsoft on one day, but not the next.

2. In the second approach, a database of time series is available. These different time series may or may not have been collected at the same instant. It is desirable to cluster these series, on the basis of their *shapes*. For example, in an application containing electrocardiogram (ECG) time series, the different patients may have contributed a time series to the database at different instants. Shape matching typically requires the use of time series similarity functions discussed in Sect. 3.4.1 of Chap. 3. Thus, both the contextual attribute and the behavioral attribute(s) may be warped or scaled, depending on the nature of the similarity function. In such cases, the different time series may not even be of equal length.

In this section, the different kinds of clustering methods will be discussed in detail. The problem becomes much more difficult when shape-based clustering is applied to multivariate time series. One solution is to generalize the similarity functions to the multivariate case. Time series similarity functions can be generalized to the multivariate case, though a full discussion of this topic is beyond the scope of this book. Relevant pointers may be found in the bibliographic notes.

For shape-based clustering, the special case of bivariate and trivariate series can also be addressed with the use of trajectory clustering. An example of how multivariate series may be converted to trajectory data is found in Sect. 1.3.2.3 of Chap. 1. Methods for trajectory clustering are discussed in Sect. 16.3.4 of Chap. 16.

### 14.5.1 Online Clustering of Coevolving Series

The problem of online clustering of coevolving series is based on determining correlations across the series, in online fashion. This is useful in many real-time applications such as financial markets because it provides an understanding of the aggregate trends in the series. In these cases, the time series are clustered based on their *correlations* in a window of length $p$. Because of the use of correlations to define similarity, the approach is referred to as *time series correlation clustering*. The *ORCLUS* correlation clustering algorithm for multidimensional data was discussed in Chap. 7. A similar principle applies to time series data, except that the correlation is measured between different components (behavioral dimensions) of the multivariate time series. The same temporal window is used for the different time series in order to compute the correlations. Therefore, the analysis of the different streams is temporally synchronized.

A natural approach is to use regression-based similarity functions to compute the similarities between different streams. It is not necessary for the two streams to be positively correlated. Rather, the streams may be highly negatively correlated. The key issue here is the *predictability* of the different time series with respect to each other. For example, in Fig. 14.8, the series A and B are very similar because they are perfectly negatively correlated with one another. This is because these two series can be predicted from one another. On the other hand, series C is very different, and has low predictability with respect to either stream, and it is useful in applications where it is desired to maximize the predictive power of cluster representatives. An example is sensor selection, where a subset of sensors

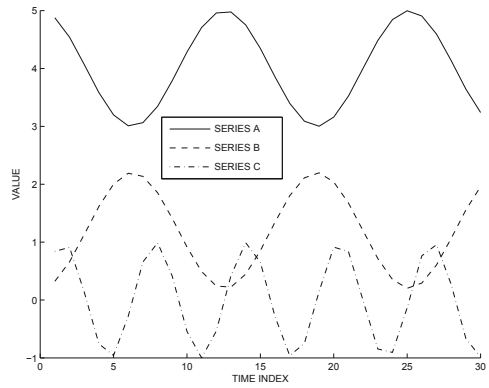



Figure 14.8: Time series correlation clustering

need to be selected which maximize the ability to predict the values of all other sensors. Because prediction is one of the most fundamental problems in real-time time series analysis, the use of regression-based similarity is natural in such scenarios. This is different from offline shape-based analysis where more conventional time series similarity functions, such as *DTW*, are used. A method that directly uses regression analysis for real-time time series clustering is referred to as *online time series correlation clustering*.

For ease in discussion, we will treat the $d$ time series as a single multivariate series with $d$ behavioral attributes. The multivariate time series of length $t$ is denoted by $\overline{Y_1} \ldots \overline{Y_t}$. The value $\overline{Y_t}$ of each of the $d$ streams at the $t$th tick is $(y_t^1 \ldots y_t^d)$. The goal is to therefore always to maintain a partition of the $d$ series, so that highly correlated components are assigned to the same partition. A representative-based approach is used for clustering. The basic idea is to incrementally maintain a set of $k$ representative time series from the $d$ series in real-time. This representative set, denoted by $J$, is similar to the representative set of a $k$-medoids algorithm. After the representatives have been determined, all of the time series streams can be assigned to one of the representatives with the use of a time series similarity function. Each series can be assigned to its closest representative. This similarity function will be discussed later in more detail.

A natural approach is to incrementally maintain the representatives, and add or drop streams to the set $J$ where necessary. The clustering is implicitly defined by assignment of the $d$ time series to their closest representatives. Therefore, when a new time series data point arrives, the current set of representatives $J$ need to be *updated*. Streams are iteratively exchanged between the current cluster representatives and the non-representatives to optimize a quality criterion based on minimizing the error. The similarity between a representative stream $i$ and nonrepresentative stream $j$, is the regression error of predicting stream $j$ from stream $i$. The idea is that true cluster representatives can be used to accurately predict the other streams. To predict the stream $j$ from stream $i$, a similar model as the autoregressive model is used, except that the elements of stream $i$ are used to predict stream $j$, instead of its own elements. Thus, the regression model is as follows:

$$y_t^j = \sum_{r=1}^{p} a_r \cdot y_{t-r}^i + c + \epsilon_t$$

This is similar to the $AR(p)$ model, except that the elements of stream $i$ are being used to predict those of stream $j$. As in the case of the $AR(p)$ model, least-squares regression

**Algorithm** *UpdateClusters*(Multivariate Stream: $\overline{Y_1} \ldots \overline{Y_t} \ldots$
Current Set of Representatives: $J$)
**begin**
Receive next time-stamp $\overline{Y_t}$ of multivariate stream;
**repeat**
Add a stream to $J$ that leads to the maximum decrease in regression error of the clustering;
Drop the stream from $J$ that leads to the least increase in regression error of the clustering;
Assign each series to closest representative in $J$ to create the clustering $\mathcal{C}$;
**until**($J$ did not change in previous iteration);
**return**($J$, $\mathcal{C}$);
**end**

Figure 14.9: Dynamically maintaining cluster representatives

can be used to learn $p$ coefficients. Furthermore, the training data is restricted to a window of size $w > p$ to allow for stream evolution. The squared average of the white noise error terms, or the $R^2$-statistic over the window of size $w > p$, can be used as the distance (similarity) between the two streams. Note that the regression coefficients can also be maintained incrementally because they are already known at the previous timestamp, and the model simply needs to be updated with the addition of a single data point. Most iterative optimization methods for least-squares regression, such as gradient descent, converge very fast when starting with a near-optimal solution.

This regression-based similarity function is not symmetric because the error of predicting stream $j$ from stream $i$ is different from the error of predicting stream $i$ from stream $j$. The representative set $J$ can also be used to create a clustering $\mathcal{C}$ of the streams, by assigning each stream to the representative, that best predicts it. Thus, at each step, the set of representatives $J$ and clusters $\mathcal{C}$ can be incrementally reported, after updating the model. The pseudocode for the online stream clustering approach is illustrated in Fig. 14.9. This approach can be useful for trend analysis in financial markets, where a representative set of stocks needs be tracked from the vast universe of stocks. Another relevant application is that of *sensor selection*, where a subset of representative sensors need to be determined to lower the operational costs of sensor networks. The description in this section is based on a simplification of a cost-based dynamic sensor selection algorithm [50].

### 14.5.2 Shape-Based Clustering

The second type of clustering is derived from shape-based clustering. In this case, the different time series may not be synchronized in time. The time series are clustered on the basis of the similarity of the shape of the overall series. A first step is the design of a shape-based similarity function. A major challenge in this case is that the different series may be scaled, translated, or stretched differently. This issue was discussed in Sect. 3.4.1 of Chap. 3. The illustration of Fig. 3.7 is replicated in Fig. 14.10. This figure illustrates different hypothetical stock tickers. In these cases, the three stocks show similar patterns, but with different scaling and random variations. Furthermore, in some cases, the time dimension may also be warped. For example, in Fig. 14.10, the entire set of values for

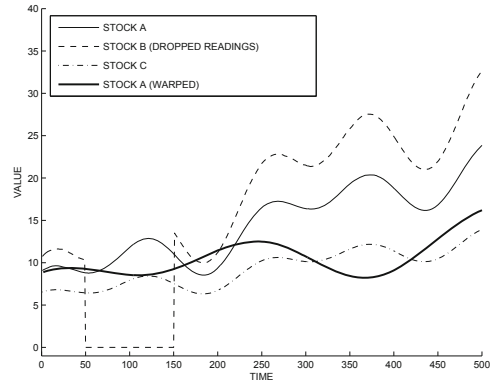



Figure 14.10: Impact of scaling, translation and noise on clustering (revisiting Fig. 3.7)

stock $A$ was stretched because the time-granularity information was not available to the analyst. This is referred to as *time warping*. Fortunately, the *dynamic time warping (DTW)* similarity function, discussed in Sect. 3.4.1 of Chap. 3, can address these issues. The design of an effective similarity function is, therefore, one of the most crucial steps in time series clustering.

Many existing methods can be adapted to shape-based time series clustering with the use of different time series similarity functions. The $k$-medoids and graph-based methods can be used with almost any similarity function. Methods such as $k$-means can also be used, though in a more limited way. This is because the different time series need to be of the same length in order for the mean of a cluster to be defined meaningfully.

#### 14.5.2.1 $k$-Means

The $k$-means method for multidimensional data is discussed in Sect. 6.3.1 of Chap. 6. This method can be adapted to time series data, by changing the similarity function and the computation of the means of the time series. The computation of the similarity function can be adapted from Sect. 3.4.1 of Chap. 3. The precise choice of the similarity function may depend on the application at hand, though the $k$-means approach is optimized for the Euclidean distance function. This is because the $k$-means approach can be viewed as an iterative solution to an optimization problem, in which the objective function is constructed with the Euclidean distance. This aspect is discussed in more detail in Chap. 6.

The Euclidean distance function on a time series is defined in the same way as multidimensional data. The means of the different time series are also defined in a similar way to multidimensional data. The $k$-means method is best used for databases of series of the same length with a one-to-one correspondence between the time points. Thus, the centroid for each time point can be defined using the correspondence. Time warping is typically difficult to use in $k$-means algorithms in a meaningful way, because of the assumption of one-to-one correspondence between the time series data points. For more generic distance functions such as $DTW$, other methods for time series clustering are more appropriate.

#### 14.5.2.2 $k$-Medoids

The main problem with the $k$-means approach is the fact that it cannot incorporate arbitrary similarity (or distance) functions. The $k$-medoids approach can be used more effectively in

this case because it does not make any assumptions on the relative lengths of the different time series. The approach is described in detail in Sect. 6.3.4 of Chap. 6. The main difference from the description provided in this section is that of the choice of the similarity function. Any of the similarity functions described in Sect. 3.4.1 of Chap. 3 may be used. The *CLARANS* method discussed in Sect. 7.3.1 of Chap. 7 can also be generalized to this case.

#### 14.5.2.3 Hierarchical Methods

The hierarchical methods, discussed in Sect. 6.4 of Chap. 6, can also be generalized to any data type because they work with pairwise distances between the different data objects. In these methods, the main challenge is that distance computations between all pairs of time series are required. Many time series distance and similarity functions require expensive dynamic programming methods. This is a major disadvantage in the use of hierarchical methods. Nevertheless, the approach can still be used quite effectively in cases where the total number of time series is small.

#### 14.5.2.4 Graph-Based Methods

Graph-based methods provide a transformational approach to time series data clustering. The idea is to transform the time series data set into a single large graph, on which community detection algorithms can be applied. As discussed in Sect. 2.2.2.9 of Chap. 2, any data type can be converted to a similarity graph, once a similarity function has been defined. Each node in this graph corresponds to a data object. Each node is connected to its $k$-nearest neighbors, and the weight of the edge is equal to the similarity between the corresponding pair of objects. Once a similarity graph has been defined, any of the graph clustering algorithms discussed in Sect. 19.3 of Chap. 19 can be used to determine node clusters. The spectral method of Sect. 19.3.4 is most commonly used. The clusters (communities) of nodes can then be mapped back to clusters of time series by using the correspondence between nodes and time series data objects.

## 14.6 Time Series Outlier Detection

As in the case of time series clustering, the problem of outlier detection in time series can be defined in two different ways.

1. *Point outliers:* A point outlier is a sudden change in a time series value at a given timestamp. This problem is closely related to forecasting, because an outlier is defined as a significant deviation from expected (or *forecasted*) values. Such outliers are referred to as *contextual* outliers because they are outliers in the *context* of their immediate history.

2. *Shape outliers:* In this case, a *consecutive pattern* of data points in a contiguous window may be defined as an anomaly. For example, in an ECG series, an irregular heart beat may be considered an anomaly *when considered together*, although no individual point in the series may be considered an anomaly. Such outliers are referred to as *collective outliers* because they are defined by combining the patterns from multiple data items.

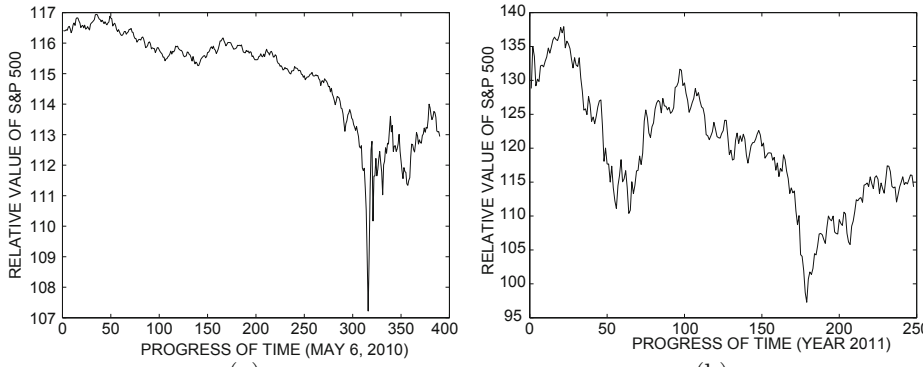



Figure 14.11: Behavior of the $S\&P$ 500 on the day of the flash crash (May 6, 2010) (**a**), and year 2001 (**b**)

To illustrate the distinction between these two kinds of anomalies, an example from financial markets will be used. The two cases illustrated in Fig. 14.11a, b show the behavior[3] of the $S\&P$ 500 over different periods in time. Figure 14.11a illustrates the movement of the $S\&P$ 500 on May 16, 2010. This was the date of the stock market flash crash. This is a very unusual event *both* from the perspective of the *point* deviation at the time of the drop, *and* from the perspective of the *shape* of the drop. A different scenario is illustrated in Fig. 14.11b. Here, the variation of the $S\&P$ 500 during the year 2001 is illustrated. There are two significant drops over the course of the year, both because of stock market weakness, and also because of the 9/11 terrorist attacks. While the specific timestamps of drop may be considered somewhat abnormal based on deviation analysis over specific windows, the actual shape of these time series is not unusual because it is frequently encountered during bear markets (periods of market weakness). Thus, these two kinds of outliers require dedicated methods for analysis. It should be pointed out that a similar distinction between the two kinds of outliers can be defined in many contextual data types such as discrete sequence data. These are referred to as point outliers and combination outliers, respectively, for the case of discrete sequence data. The combination outliers in discrete sequence data are analogous to shape outliers in continuous time series data. This is discussed in greater detail in Chap. 15.

### 14.6.1 Point Outliers

Point outliers are closely related to the problem of forecasting in time series data. A data point is considered an outlier if it deviates significantly from its expected (or *forecasted*) value. Such point outliers correspond to unsupervised *events* in the underlying data. Event detection is often considered a synonym for temporal outlier detection when it performed in real time.

Point outliers can be defined for either univariate or multivariate data. The case of univariate data and multivariate data is almost identical. Therefore, the more general case of multivariate data will be discussed. As in previous sections, assume that the multivariate series on which the outliers are to be detected is denoted by $\overline{Y_1} \ldots \overline{Y_n}$. The overall approach comprises four steps:

[3]The tracking Exchange Traded Fund (ETF) SPY was used.

1. Determine the forecasted values of the time series at each timestamp. Depending on the nature of the underlying series, any of the univariate or multivariate methodologies discussed in Sect. 14.3 may be used. Let the forecasted value at the $r$th timestamp $t_r$ be denoted by $\overline{W_r}$

2. Compute the (possibly multivariate) time series of deviations $\overline{\Delta_1} \ldots \overline{\Delta_r} \ldots$. In other words, for the $r$th timestamp $t_r$, the deviation is computed as follows:

$$\overline{\Delta_r} = \overline{W_r} - \overline{Y_r}. \tag{14.21}$$

3. Let the $d$ different components of $\overline{\Delta_r}$ be denoted by $(\delta_r^1 \ldots \delta_r^d)$. These can be separated out into $d$ different univariate series of deviations along each dimension. The values of the $i$th series are denoted by $\delta_1^i \ldots \delta_n^i$. Let the mean and standard deviation of the $i$th series of deviations be denoted by $\mu_i$ and $\sigma_i$.

4. Compute the normalized deviations $\delta z_r^i$ as follows:

$$\delta z_r^i = \frac{\delta_r^i - \mu_i}{\sigma_i}. \tag{14.22}$$

   The resulting deviation is essentially equal to the $Z$-value of a normal distribution. This approach provides a continuous alarm level of outlier scores for each of the $d$ dimensions of the multivariate time series. Unusual time instants can be detected by using thresholding on these scores. Because of the $Z$-value interpretation, an absolute threshold of 3 is usually considered sufficient.

In some cases, it is desirable to create a unified alarm level of deviation scores rather than creating a separate alarm level for each of the series. This problem is closely related to that of outlier ensemble analysis that is discussed in Sect. 9.4 of Chap. 9. The unified alarm level $U_r$ at timestamp $r$ can be reported as the maximum of the scores across the different components of the multivariate series:

$$U_r = \max_{i \in \{1 \ldots d\}} \delta z_r^i. \tag{14.23}$$

The score across different detectors can be combined in other ways, such as by using the average or squared aggregate over different series.

### 14.6.2 Shape Outliers

One of the earliest methods for finding shape-based outliers is the *Hotsax* approach. In this approach, outliers are defined on windows of the time series. A $k$-nearest neighbor method is used to determine the outlier scores. Specifically, the Euclidean distance of a data point to its $k$th-nearest neighbors is used to define the outlier score.

The outlier analysis is performed over windows of length $W$. Therefore, the approach reports windows of unusual shapes from a time series of data points. The first step is to extract all windows of length $W$ from the time series by using a sliding-window approach. The analysis is then performed over these newly created data objects. For each extracted window, its Euclidean distance to the other *nonoverlapping* windows is computed. The windows with the highest $k$-nearest neighbor distance values are reported as outliers. The reason for using nonoverlapping windows is to minimize the impact of trivial matches to overlapping windows. While a brute-force $k$-nearest neighbor approach can determine the

outliers, the complexity will scale with the square of the number of data points. Therefore, a pruning method is used for improving the efficiency. While this method optimizes the efficiency, and it does not affect the final result reported by the method.

The general principle of pruning for more efficient outlier detection in nearest neighbor methods was introduced in Sect. 8.5.1.2 of Chap. 8. The algorithm examines the candidate subsequences iteratively in an outer loop. For each such candidate subsequence, the $k$-nearest neighbors are computed progressively in an inner loop with distance computations to other subsequences. Each candidate subsequence is either included in the current set of best $n$ outlier estimates at the end of an outer loop iteration, or discarded via *early abandonment* of the inner loop without computing the *exact* value of the $k$-nearest neighbor. This inner loop can be terminated early when the currently approximated $k$-nearest neighbor distance for that candidate subsequence is less than the score for the $n$th best outlier found so far. Clearly, such a subsequence cannot be an outlier. To obtain the best pruning results, the subsequences need to be heuristically ordered so that the earliest candidate subsequences examined in the outer loop have the greatest tendency to be outliers. Furthermore, the pruning performance is also most effective when the true outliers are found early. It remains to explain, how the heuristic orderings required for good pruning are achieved.

Pruning is facilitated by an approach that can measure the clustering behavior of the underlying subsequences. Clustering has a well known relationship of complementarity with outlier analysis. Therefore it is useful to examine those subsequences first in the outer loop that are members of clusters containing very few (or one) members. The SAX representation is used to create a simple mapping of the subsequences into clusters. Subsequences that map to the same SAX word, are assumed to belong to a single cluster. The piecewise aggregate approximations of SAX are performed over windows of length $w < W$. Therefore, the length of a SAX word is $W/w$, and the number of distinct possibilities for a SAX word is small if $W/w$ is small. These distinct words correspond to the different clusters. Multiple subsequences map to the same cluster. Therefore, the ordering of the candidates is based on the number of data objects in the same cluster. Candidates in clusters with fewer objects are examined first because they are more likely to be outliers.

This cluster-based ordering is used to design an efficient pruning mechanism for outlier analysis. The candidates in the clusters are examined one by one in an outer loop. The $k$-nearest neighbor distances to these candidates are computed in an inner loop. For each candidate subsequence, those subsequences that map to the same word as the candidate may be considered first for computing the nearest neighbor distances in the inner loop. This provides quick and tight upper bounds on the nearest neighbor distances. As these distances are computed one by one, a tighter and tighter upper bound on the nearest neighbor distance is computed over the progression of the inner loop. *A candidate can be pruned when an upper bound on its nearest neighbor distance is guaranteed to be smaller (i.e., more similar) than the nth best outlier distance found so far.* Therefore, for any given candidate series, it is not necessary to determine its exact nearest neighbor by comparing to all subsequences. Rather, early termination of the inner loop is often possible during the computation of the nearest neighbor distance. This forms the core of the pruning methodology of *Hotsax*, and is similar in principle to the nested-loop pruning methodology discussed in Sect. 8.5.1.2 of Chap. 8 on multidimensional outlier analysis. The main difference is in terms of how the SAX representation is used both for ordering the candidates in the outer loop, and ordering the distance computations in the inner loop.

## 14.7 Time Series Classification

Time series classification can be defined in several ways, depending on the association of the underlying class labels to either individual timestamps, or the whole series.

1. *Point labels:* In this case, the class labels are associated with individual timestamps. In most cases, the class of interest is rare in nature and corresponds to unusual activity at that timestamp. This problem is also referred to as *event detection*. This version of the event detection problem can be distinguished from the unsupervised outlier detection problem discussed in Sect. 14.6, in that it is *supervised* with labels.

2. *Whole-series labels:* In this case, the class labels are associated with the full series. Therefore, the series needs to be classified on the basis of the shapes inside it.

Both these problems will be discussed in this chapter.

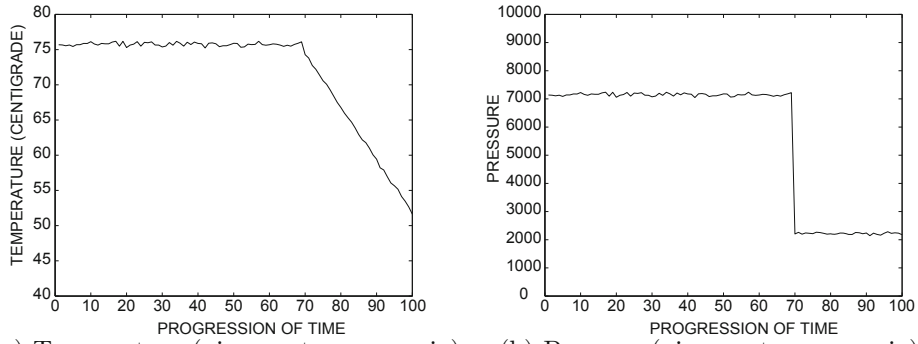
### 14.7.1 Supervised Event Detection

The problem of supervised event detection is one in which the class labels are associated with the timestamps rather than the full series. In most cases, one or more of the class labels are rare, and the remaining labels correspond to the "normal" periods. While it is possible in principle to define the problem with a balanced distribution of labels, this is rarely the case in application-specific settings. Therefore, the discussion in this subsection will focus only on the imbalanced label distribution scenario.

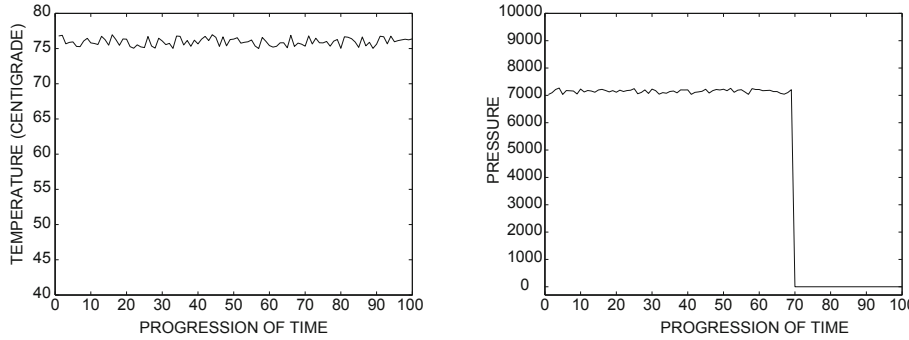
These rare class labels correspond to the events in the underlying data. For example, consider a scenario, in which the performance of a machine is tracked using sensors. In some cases, a rare event, such as the malfunctioning of the machine, may cause unusual sensor readings. Such unusual events need to be tracked *in a timely fashion*. Therefore, this problem is similar to point anomaly detection, except that it is done in a supervised way.

In many application-specific scenarios, the time series data collection is inherently designed in such a way that the unusual events are reflected in unexpected deviations of the time series. This is particularly true of many sensor-based collection mechanisms. While this can be captured by unsupervised methods, the addition of supervision helps in the removal of spurious events that may have different underlying causes. For example, consider the case of an environmental monitoring application. Many deviations may be the result of the failure of the sensor equipment, or another spurious event that causes deviations in sensor values. This may not necessarily reflect an anomaly of interest. While anomalous events often correspond to extreme deviations in sensor stream values, the precise causality of different kinds of deviations may be quite different. These other *noisy* or *spurious* abnormalities may not be of any interest to an analyst. For example, consider the case illustrated in Fig. 14.12, in which temperature and pressure values inside pressurized pipes containing heating fluids are illustrated. Figures 14.12 a and b illustrate values on two sensors in a pipe rupture scenario. Figures 14.12 c and d illustrate the values of the two sensors in a situation where the pressure sensor malfunctions, and this results in a value of 0 at each timestamp in the pressure sensor. In the first case, the readings of both pressure and temperature sensors are affected by the malfunction, though the final pressure values are not zero, but they reflect the pressure in the external surroundings. The readings on the temperature sensor are not affected at all in the second scenario, since the malfunction is specific to the pressure sensor.

Thus, the key is to *differentiate* among the deviations of different behavioral attributes in a multivariate scenario. The use of supervision is very helpful because it can be used




(a) Temperature (pipe rupture scenario) (b) Pressure (pipe rupture scenario)




(c) Temperature (sensor failure scenario) (d) Pressure (sensor failure scenario)

Figure 14.12: Behavior of temperature and pressure sensors due to pipe rupture (**a, b)**, and failure of pressure sensor (**c, d)**

to determine the *differential* behavior of the deviations across different streams. In the aforementioned pipe rupture scenario, the relative deviations in the two events are quite different. In the labeling input, it is assumed that most of the timestamps are labeled "normal." A few *ground truth timestamps*, $T_1 \ldots T_r$, are labeled "rare." These are used for supervision. These are referred to as *primary abnormal events*. In addition, spurious events may also cause large deviations. These timestamps are referred to as *secondary abnormal events*. In some application-specific scenarios, the timestamps for the secondary abnormal events may be provided, though this is not assumed here. The bibliographic notes contain pointers to these enhanced methods.

It is assumed that a total of $d$ different time series data streams are available, and the differential patterns in the $d$ streams are used to detect the abnormal events. The overall process of event prediction is to create a composite alarm level from the error terms in the time series prediction. The first step is to use a univariate time series prediction model to determine the error terms at a given timestamp. Any of the models discussed in Sect. 14.3 may be used. These are then combined together to create a composite alarm level with the use of coefficients $\alpha_1 \ldots \alpha_d$ for the $d$ different time series data streams. The values of $\alpha_1 \ldots \alpha_d$ are learned from the training data in an offline (or periodic batch) phase, so as to best discriminate the true event from the normal periods. The actual prediction can be performed using an online approach in real time. Therefore, the steps may be summarized as follows:

1. (Offline Batch) Learn the coefficients $\alpha_1 \ldots \alpha_d$ that best distinguish between the true and normal periods. The details of this step are discussed later in this section.

2. (Real Time) Determine the (absolute) deviation level for each timeseries data stream, with the use of any forecasting method discussed in Sect. 14.3. These correspond the absolute values of the white noise error terms. Let the absolute deviation level of stream $j$ at timestamp $n$ be denoted by $z_n^j$.

3. (Real Time) Combine the deviation levels for the different streams as follows, to create the composite alarm level:

$$Z_n = \sum_{i=1}^{d} \alpha_i z_n^i \tag{14.24}$$

   The value of $Z_n$ is reported as the alarm level at timestamp $n$. Thresholding can be used on the alarm level to generate discrete labels.

The main step in the last section, which has not yet been discussed, is the determination of the *discrimination coefficients* $\alpha_1 \ldots \alpha_d$. These should be selected in the training phase, so as to maximize the differences in the alarm level between the primary events and the normal periods.

To learn the coefficients $\alpha_1 \ldots \alpha_d$ in the training phase, the composite alarm level is averaged at the timestamps $T_1 \ldots T_r$ for all primary events of interest. Note that the composite alarm level at each timestamp $T_i$ is an algebraic expression, which is a linear function of the coefficients $\alpha_1 \ldots \alpha_d$ according to Eq. 14.24. These expressions are added up over the time stamps $T_1 \ldots T_r$ to create an alarm level $Q^p(\alpha_1 \ldots \alpha_d)$ which is a function of $(\alpha_1, \ldots \alpha_d)$.

$$Q^p(\alpha_1 \ldots \alpha_d) = \frac{\sum_{i=1}^{r} Z_{T_i}}{r}. \tag{14.25}$$

A similar algebraic expression for the normal alarm level $Q^n(\alpha_1 \ldots \alpha_d)$ is also computed by using all of the available timestamps, the majority of which are assumed to be normal.

$$Q^n(\alpha_1 \ldots \alpha_d) = \frac{\sum_{i=1}^{n} Z_i}{n} \tag{14.26}$$

As in the case of the event signature, the normal alarm level is also a linear function of $\alpha_1 \ldots \alpha_d$. Then, the optimization problem is that of determining the optimal values of $\alpha_i$ that increase the differential signature between the primary events and the normal alarm level. This optimization problem is as follows:

$$\text{Maximize } Q^p(\alpha_1 \ldots \alpha_d) - Q^n(\alpha_1 \ldots \alpha_d)$$
$$\text{subject to: } \sum_{i=1}^{d} \alpha_i^2 = 1$$

This optimization problem can be solved using any off-the-shelf iterative optimization solver. In practice, the online event detection and offline learning processes are executed simultaneously, as new events are encountered. In such cases, the values of $\alpha_i$ can be updated incrementally within the iterative optimization solver. The composite alarm level can be reported as an event score. Alternatively, thresholding on the alarm level can be used to generate discrete timestamps at which the events are predicted. The choice of threshold will regulate the trade-off between the precision and recall of the predicted events.

### 14.7.2 Whole Series Classification

In whole-series classification, the labels are associated with the entire series, rather than events associated with individual timestamps. It is assumed that a database of $N$ different series is available, and each series has a length of $n$. Each of the series is associated with a class label drawn from $\{1 \ldots k\}$.

Many proximity-based classifiers are designed with the help of time series similarity functions. Thus, the effective design of similarity functions is crucial in classification, as is the case in many other time series data mining applications.

In the following, three classification methods will be discussed. Two of these methods are *inductive* methods, in which only the training instances are used to build a model. These are then used for classification. The third method is a *transductive semisupervised method*, in which the training and test instances are used together for classification. The semisupervised approach is a graph-based method in which the unlabeled test instances are leveraged for more effective classification.

#### 14.7.2.1 Wavelet-Based Rules

A major challenge in time series classification is that much of the series may be noisy and irrelevant. The classification properties may be exhibited only in temporal segments of varying length in the series. For example, consider the scenario where the series in Fig. 14.11 are presented to a learner with labels. In the case where the label corresponds to a recession (Fig. 14.11a), it is important for a learner to analyze the trends for a period of a few weeks or months in order to determine the correct labels. On the other hand, where the label corresponds to the occurrence of a flash crash (Fig. 14.11b), it is important for a learner to be able to extract out the trends over the period of a day.

For a given learning problem, it may not be known *a priori* what level of granularity should be used for the learning process. The Haar wavelet method provides a multigranularity decomposition of the time series data to handle such scenarios. As discussed in Sect. 14.4 on time series motifs, wavelets are an effective way to determine frequent trends over varying levels of granularity. It is therefore natural to combine multigranular motif discovery with associative classifiers.

Readers are advised to refer to Sect. 2.4.4.1 of Chap. 2 for a discussion of wavelet decomposition methods. The Haar wavelet coefficient of order $i$ analyzes trends over a time period, which is proportional to $2^{-i} \cdot n$, where $n$ is the full length of the series. Specifically, the coefficient value is equal to half the difference between the average values of the first half and second half of the time period of length $2^{-i} \cdot n$. Because the Haar wavelet represents the coefficients of different orders in the transformation, it automatically accounts for trends of different granularity. In fact, an arbitrary shape in any particular window of the series can usually be well approximated by an appropriate subset of wavelet coefficients. These can be considered signatures that are specific to a particular class label. The goal of the rule-based method is to discover signatures that are specific to particular class labels. Therefore, the overall training approach in the rule-based method is as follows:

1. Generate wavelet representation of each of the $N$ time series to create $N$ numeric multidimensional representations.

2. Discretize wavelet representation to create categorical representations of the time series wavelet transformation. Thus, each categorical attribute value represents a range of numeric values of the wavelet coefficients.

3. Generate rule set using any rule-based classifier described in Sect. 10.4 of Chap. 10. The combination of wavelet coefficients in the rule antecedent correspond to the "signature" shapes in the time series, which are relevant to classification.

Once the rule set has been generated, it can be used to classify arbitrary time series. A given test series is converted into its wavelet representation. The rules that are fired by this series are determined. These are used to classify the test instance. The methods for using a rule set to classify a test instance are discussed in Sect. 10.4 of Chap. 10. When it is known that the class labels are sensitive to periodicity rather than local trends, this approach should be used with Fourier coefficients instead of wavelet coefficients.

#### 14.7.2.2 Nearest Neighbor Classifier

Nearest neighbor classifiers are introduced in Sect. 10.8 of Chap. 10. The nearest neighbor classifier can be used with virtually any data type, as long as an appropriate distance function is available. Distance functions for time series data have already been introduced in Sect. 3.4.1 of Chap. 3. Any of these distance (similarity) functions may be used, depending on the domain-specific scenario. The basic approach is the same as in the case of multidimensional data. For any test instance, its $k$-nearest neighbors in the training data are determined. The dominant label from these $k$-nearest neighbors is reported as the relevant class label. The optimal value of $k$ may be determined by using leave-one-out cross-validation.

#### 14.7.2.3 Graph-Based Methods

Similarity graphs can be used for clustering and classification of virtually any data type. The use of similarity graphs for semisupervised classification was introduced in Sect. 11.6.3 of Chap. 11. The basic approach constructs a similarity graph from *both* the training and test instances. Thus, this approach is a transductive method because the test instances are used along with the training instances for classification. A graph $G = (N, A)$ is constructed, in which a node in $N$ corresponds to each of the training and test instances. A subset of nodes in $G$ is labeled. These correspond to instances in the training data, whereas the unlabeled nodes correspond to instances in the test data. Each node in $N$ is connected to its $k$-nearest neighbors with an undirected edge in $A$. The similarity is computed using any of the distance functions discussed in Sect. 3.4.1 or 3.4.2 of Chap. 3. The specified labels of nodes in $N$ are then used to derive labels for nodes where they are unknown. This problem is referred to as collective classification. Numerous methods for collective classification are discussed in Sect. 19.4 of Chap. 19.

## 14.8 Summary

Time series data is common in many domains, such as sensor networking, healthcare, and financial markets. Typically, time series data needs to be normalized, and missing values need to be imputed for effective processing. Numerous data reduction techniques such as Fourier and wavelet transforms are used in time series analysis. The choice of similarity function is the most crucial aspect of time series analysis, because many data mining applications such as clustering, classification, and outlier detection are dependent on this choice.

Forecasting is an important problem in time series analysis because it can be used to make predictions about data points in the future. Most time series applications use either point-wise or shape-wise analysis. For example, in the case of clustering, point-wise analysis

results in temporal correlation clusters, where a cluster contains many different series that move together. On the other hand, shape-wise analysis is focused on determining groups of time series with approximately similar shapes.

The problem of point-wise outlier detection is closely related to forecasting. A time series data point is an outlier if it differs significantly from its expected (or *forecasted*) value. A shape outlier is defined in time series data with the use of similarity functions. When supervision is incorporated in point-wise outlier detection, the problem is referred to as *event detection*. Many existing classification techniques can be extended to shape-based classification.

## 14.9 Bibliographic Notes

The problem of time series analysis has been studied extensively by statisticians and computer scientists. Detailed books on temporal data mining and time series analysis may be found in [134, 467, 492]. Data preparation and normalization are important aspects of time series analysis. The binning approach is also referred to as piecewise aggregate approximation (PAA) [309]. The SAX approach is described in [355]. The *DWT*, *DFT*, and *DCT* transforms are discussed in [134, 467, 475, 492]. Time series similarity measures are discussed in detail in Chap. 3 of this book, and in an earlier tutorial by Gunopulos and Das [241].

The problem of time series motif discovery has been discussed in [151, 394, 395, 418, 524]. The distance-based motif discussion in this chapter is based on the description in [356]. A wavelet-based approach for multiresolution motif discovery is discussed in [51]. The discovered motifs are used for classification. Further discussions on periodic pattern mining may be found in [251, 411, 467]. The problem of time series forecasting is discussed in detail in [134]. The lower bounding of distance functions is useful for fast pruning and indexing. The lower bounding on PAA has been shown in [309]. It has been shown how to perform lower bounding on *DTW* in [308].

A recent survey on time series data clustering may be found in [324]. The problem of online clustering time series data streams is related to the problem of sensor selection. The Selective MUSCLES method was introduced in [527] that can potentially be used to select representatives from a set of time series. The online correlation method, discussed in this chapter, is based on the discussion in [50]. A survey of representative selection algorithms for sensor data may be found in [414]. Many of these algorithms may also be used for online correlation clustering.

A survey on outlier detection for temporal data may be found in [237]. A chapter on temporal outlier detection may also be found in a recent outlier detection book [5]. The online detection of *timestamps* is referred to as event detection. The supervised version of this problem is related to rare class detection. The supervised event detection method discussed in Sect. 14.7.1 was proposed in [52]. The *Hotsax* approach discussed in this book was proposed in [306]. A wavelet-based approach for classification of sequences is discussed in [51]. This approach has been adapted for time series data in this chapter. Surveys on temporal data classification may be found in [33, 516]. The latter survey is on sequence classification, although it also discusses many aspects of time series classification.

## 14.10 Exercises

**1.** For the time series $(2, 7, 5, 3, 3, 5, 5, 3)$, determine the binned time series where the bins are chosen to be of length 2.

**2.** For the time series of Exercise 1, construct the rolling average series for a window size of 2 units. Compare the results to those obtained in the previous exercise.

**3.** For the time series of Exercise 1, construct the exponentially smoothed series, with a smoothing parameter $\alpha = 0.5$. Set the initial smoothed value $y_0$ to the first point in the series.

**4.** Implement the binning, moving average, and exponential smoothing methods.

**5.** Consider a series, in which consecutive values are related as follows:

$$y_{i+1} = y_i \cdot (1 + R_i) \tag{14.27}$$

Here $R_i$ is a random variable drawn from $[0.01, 0.05]$. What transformation would you apply to make this series stationary?

**6.** Consider the series in which $y_i$ is defined as follows:

$$y_i = 1 + i + i^2 + R_i \tag{14.28}$$

Here $R_i$ is a random variable drawn from $[0.01, 0.05]$. What transformation would you apply to make this series stationary?

**7.** For a real-valued time series $x_0 \ldots x_{n-1}$ with Fourier coefficients $X_0 \ldots X_{n-1}$, show that $X_k + X_{n-k}$ is real-valued for each $k \in \{1 \ldots n-1\}$.

**8.** Suppose that you wanted to implement the $k$-means algorithm for a set of time series, and you were given the same subset of complex Fourier coefficients for each dimensionality-reduced series. How would the implementation be different from that of using $k$-means on the original time series?

**9.** Use Parseval's theorem and additivity to show that the dot product of two series is proportional to the sum of the dot products of the real parts and the dot products of the imaginary parts of the Fourier coefficients of the two series. What is the proportionality factor?

**10.** Implement a shape-based $k$-nearest neighbor classifier for time series data.

**11.** Generalize the distance-based motif discovery algorithm, discussed in this chapter to the case where the motifs are allowed to be of any length $[a, b]$, and the Manhattan segmental distance is used for distance comparison. The Manhattan segmental distance between a pair of series is the same as the Manhattan distance, except that it divides the distance with the motif length for normalization.

**12.** Suppose you have a database of $N$ series, and the frequency of motifs are counted, so that their occurrence once in any series is given a credit of one. Discuss the details of an algorithm that can use wavelets to determine motifs at different resolutions.