Chapter 3

Similarity and Distances

"*Love is the power to see similarity in the dissimilar.*"—Theodor Adorno

3.1 Introduction

Many data mining applications require the determination of similar or dissimilar objects, patterns, attributes, and events in the data. In other words, a methodical way of quantifying similarity between data objects is required. Virtually all data mining problems, such as clustering, outlier detection, and classification, require the computation of similarity. A formal statement of the problem of similarity or distance quantification is as follows:

Given two objects O_1 and O_2, determine a value of the similarity $Sim(O_1, O_2)$ (or distance $Dist(O_1, O_2)$) between the two objects.

In similarity functions, larger values imply greater similarity, whereas in distance functions, smaller values imply greater similarity. In some domains, such as spatial data, it is more natural to talk about distance functions, whereas in other domains, such as text, it is more natural to talk about similarity functions. Nevertheless, the principles involved in the design of such functions are generally invariant across different data domains. This chapter will, therefore, use either of the terms "distance function" and "similarity function," depending on the domain at hand. Similarity and distance functions are often expressed in closed form (e.g., Euclidean distance), but in some domains, such as time-series data, they are defined algorithmically and cannot be expressed in closed form.

Distance functions are fundamental to the effective design of data mining algorithms, because a poor choice in this respect may be very detrimental to the quality of the results. Sometimes, data analysts use the Euclidean function as a "black box" without much thought about the overall impact of such a choice. It is not uncommon for an inexperienced analyst to invest significant effort in the algorithmic design of a data mining problem, while treating the distance function subroutine as an afterthought. This is a mistake. As this chapter will elucidate, poor choices of the distance function can sometimes be disastrously misleading

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_3

depending on the application domain. Good distance function design is also crucial for type portability. As discussed in Sect. 2.4.4.3 of Chap. 2, spectral embedding can be used to convert a similarity graph constructed on any data type into multidimensional data.

Distance functions are highly sensitive to the data distribution, dimensionality, and data type. In some data types, such as multidimensional data, it is much simpler to define and compute distance functions than in other types such as time-series data. In some cases, user intentions (or *training feedback* on object pairs) are available to *supervise* the distance function design. Although this chapter will primarily focus on unsupervised methods, we will also briefly touch on the broader principles of using supervised methods.

This chapter is organized as follows. Section 3.2 studies distance functions for multidimensional data. This includes quantitative, categorical, and mixed attribute data. Similarity measures for text, binary, and set data are discussed in Sect. 3.3. Temporal data is discussed in Sect. 3.4. Distance functions for graph data are addressed in Sect. 3.5. A discussion of supervised similarity will be provided in Sect. 3.6. Section 3.7 gives a summary.

3.2 Multidimensional Data

Although multidimensional data are the simplest form of data, there is significant diversity in distance function design across different attribute types such as categorical or quantitative data. This section will therefore study each of these types separately.

3.2.1 Quantitative Data

The most common distance function for quantitative data is the L_p-norm. The L_p-norm between two data points $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$ is defined as follows:

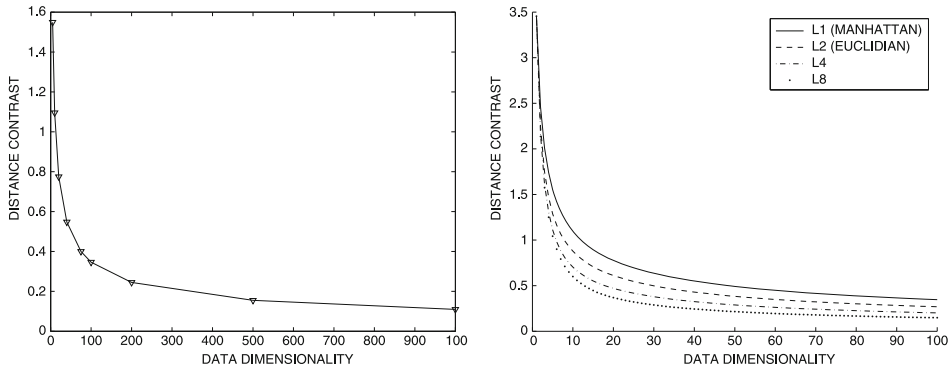
$$Dist(\overline{X}, \overline{Y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}. \tag{3.1}$$

Two special cases of the L_p-norm are the *Euclidean* ($p = 2$) and the *Manhattan* ($p = 1$) metrics. These special cases derive their intuition from spatial applications where they have clear physical interpretability. The Euclidean distance is the straight-line distance between two data points. The Manhattan distance is the "city block" driving distance in a region in which the streets are arranged as a rectangular grid, such as the Manhattan Island of New York City.

A nice property of the Euclidean distance is that it is rotation-invariant because the straight-line distance between two data points does not change with the orientation of the axis system. This property also means that transformations, such as *PCA*, *SVD*, or the wavelet transformation for time series (discussed in Chap. 2), can be used on the data without affecting[1] the distance. Another interesting special case is that obtained by setting $p = \infty$. The result of this computation is to select the dimension for which the two objects are the most distant from one another and report the absolute value of this distance. All other features are ignored.*a*

The L_p-norm is one of the most popular distance functions used by data mining analysts. One of the reasons for its popularity is the natural intuitive appeal and interpretability of L_1- and L_2-norms in spatial applications. The intuitive interpretability of these distances does not, however, mean that they are the most relevant ones, especially for the high-dimensional case. In fact, these distance functions may not work very well when the data

[1]The distances are affected after dimensions are dropped. However, the transformation itself does not impact distances.



(a) Contrasts with dimensionality (b) Contrasts with norms

Figure 3.1: Reduction in distance contrasts with increasing dimensionality and norms

are high dimensional because of the varying impact of data sparsity, distribution, noise, and feature relevance. This chapter will discuss these broader principles in the context of distance function design.

3.2.1.1 Impact of Domain-Specific Relevance

In some cases, an analyst may know which features are more important than others for a particular application. For example, for a credit-scoring application, an attribute such as salary is much more relevant to the design of the distance function than an attribute such as gender, though both may have some impact. In such cases, the analyst may choose to weight the features differently if domain-specific knowledge about the relative importance of different features is available. This is often a heuristic process based on experience and skill. The generalized L_p-distance is most suitable for this case and is defined in a similar way to the L_p-norm, except that a coefficient a_i is associated with the ith feature. This coefficient is used to weight the corresponding feature component in the L_p-norm:

$$Dist(\overline{X}, \overline{Y}) = \left(\sum_{i=1}^{d} a_i \cdot |x_i - y_i|^p \right)^{1/p}. \tag{3.2}$$

This distance is also referred to as the generalized *Minkowski distance*. In many cases, such domain knowledge is not available. Therefore, the L_p-norm may be used as a default option. Unfortunately, without knowledge about the most relevant features, the L_p-norm is susceptible to some undesirable effects of increasing dimensionality, as discussed subsequently.

3.2.1.2 Impact of High Dimensionality

Many distance-based data mining applications lose their effectiveness as the dimensionality of the data increases. For example, a distance-based clustering algorithm may group unrelated data points because the distance function may poorly reflect the intrinsic semantic distances between data points with increasing dimensionality. As a result, distance-based models of clustering, classification, and outlier detection are often *qualitatively* ineffective. This phenomenon is referred to as the "curse of dimensionality," a term first coined by Richard Bellman.

To better understand the impact of the dimensionality curse on distances, let us examine a unit cube of dimensionality d that is fully located in the nonnegative quadrant, with one corner at the origin $\overline{O}$. What is the Manhattan distance of the corner of this cube (say, at the origin) to a randomly chosen point $\overline{X}$ inside the cube? In this case, because one end point is the origin, and all coordinates are nonnegative, the Manhattan distance will sum up the coordinates of $\overline{X}$ over the different dimensions. Each of these coordinates is uniformly distributed in $[0, 1]$. Therefore, if Y_i represents the uniformly distributed random variable in $[0, 1]$, it follows that the Manhattan distance is as follows:

$$Dist(\overline{O}, \overline{X}) = \sum_{i=1}^{d} (Y_i - 0). \tag{3.3}$$

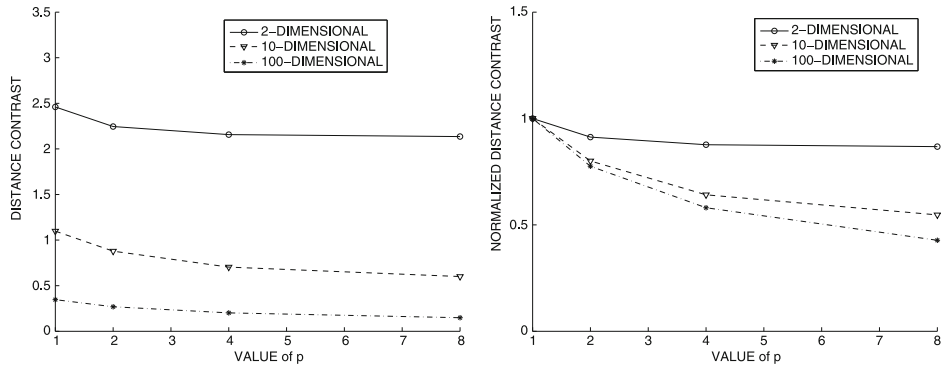
The result is a random variable with a mean of $\mu = d/2$ and a standard deviation of $\sigma = \sqrt{d/12}$. For large values of d, it can be shown by the law of large numbers that the vast majority of randomly chosen points inside the cube will lie in the range $[D_{min}, D_{max}] = [\mu - 3\sigma, \mu + 3\sigma]$. Therefore, most of the points in the cube lie within a distance range of $D_{max} - D_{min} = 6\sigma = \sqrt{3d}$ from the origin. Note that the expected Manhattan distance grows with dimensionality at a rate that is linearly proportional to d. Therefore, the *ratio* of the variation in the distances to the absolute values that is referred to as $Contrast(d)$, is given by:

$$\text{Contrast}(d) = \frac{D_{max} - D_{min}}{\mu} = \sqrt{12/d}. \tag{3.4}$$

This ratio can be interpreted as the distance *contrast* between the different data points, in terms of how different the minimum and maximum distances from the origin might be considered. Because the contrast reduces with $\sqrt{d}$, it means that there is virtually no contrast with increasing dimensionality. Lower contrasts are obviously not desirable because it means that the data mining algorithm will score the distances between all pairs of data points in approximately the same way and will not discriminate well between different pairs of objects with varying levels of semantic relationships. The variation in contrast with increasing dimensionality is shown in Fig. 3.1a. This behavior is, in fact, observed for all L_p-norms at different values of p, though with varying severity. These differences in severity will be explored in a later section. Clearly, with increasing dimensionality, a direct use of the L_p-norm may not be effective.

3.2.1.3 Impact of Locally Irrelevant Features

A more fundamental way of exploring the effects of high dimensionality is by examining the impact of irrelevant features. This is because many features are likely to be irrelevant in a typical high-dimensional data set. Consider, for example, a set of medical records, containing patients with diverse medical conditions and very extensive quantitative measurements about various aspects of an individual's medical history. For a cluster containing diabetic patients, certain attributes such as the blood glucose level are more important for the distance computation. On the other hand, for a cluster containing epileptic patients, a different set of features will be more important. The additive effects of the natural variations in the many attribute values may be quite significant. A distance metric such as the Euclidean metric may unnecessarily contribute a high value from the more noisy components because of its square-sum approach. The key point to understand here is that the precise features that are relevant to the distance computation may sometimes be sensitive to the particular pair of objects that are being compared. This problem cannot be solved by global feature



(a) Contrast (b) Contrast/Manhattan Contrast

Figure 3.2: Impact of p on contrast

subset selection during preprocessing, because the relevance of features is *locally* determined by the pair of objects that are being considered. Globally, all features may be relevant.

When many features are irrelevant, the additive noise effects of the irrelevant features can sometimes be reflected in the concentration of the distances. In any case, such irrelevant features will almost always result in errors in distance computation. Because high-dimensional data sets are often likely to contain diverse features, many of which are irrelevant, the additive effect with the use of a sum-of-squares approach, such as the L_2-norm, can be very detrimental.

3.2.1.4 Impact of Different L_p-Norms

Different L_p-norms do not behave in a similar way either in terms of the impact of irrelevant features or the distance contrast. Consider the extreme case when $p = \infty$. This translates to using only the dimension where the two objects are the most *dissimilar*. Very often, this may be the impact of the natural variations in an irrelevant attribute that is not too useful for a similarity-based application. In fact, for a 1000-dimensional application, if two objects have similar values on 999 attributes, such objects should be considered *very* similar. However, a single irrelevant attribute on which the two objects are very different will throw off the distance value in the case of the L_∞ metric. In other words, local similarity properties of the data are de-emphasized by L_∞. Clearly, this is not desirable.

This behavior is generally true for larger values of p, where the irrelevant attributes are emphasized. In fact, it can also be shown that distance contrasts are also poorer for larger values of p for certain data distributions. In Fig. 3.1b, the distance contrasts have been illustrated for different values of p for the L_p-norm over different dimensionalities. The figure is constructed using the same approach as Fig. 3.1a. While all L_p-norms degrade with increasing dimensionality, the degradation is much faster for the plots representing larger values of p. This trend can be understood better from Fig. 3.2 where the value of p is used on the X-axis. In Fig. 3.2a, the contrast is illustrated with different values of p for data of different dimensionalities. Figure 3.2b is derived from Fig. 3.2a, except that the results show the fraction of the Manhattan performance achieved by higher order norms. It is evident that the *rate* of degradation with increasing p is higher when the dimensionality of the data is large. For 2-dimensional data, there is very little degradation. This is the reason that the value of p matters less in lower dimensional applications.

This argument has been used to propose the concept of fractional metrics, for which $p \in (0, 1)$. Such fractional metrics can provide more effective results for the high-dimensional case. As a rule of thumb, the larger the dimensionality, the lower the value of p. However, no exact rule exists on the precise choice of p because dimensionality is not the only factor in determining the proper value of p. The precise choice of p should be selected in an application-specific way, with the use of benchmarking. The bibliographic notes contain discussions on the use of fractional metrics.

3.2.1.5 Match-Based Similarity Computation

Because it is desirable to select locally relevant features for distance computation, a question arises as to how this can be achieved in a meaningful and practical way for data mining applications. A simple approach that is based on the cumulative evidence of matching many attribute values has been shown to be effective in many scenarios. This approach is also relatively easy to implement efficiently.

A broader principle that seems to work well for high-dimensional data is that the impact of the noisy variation along individual attributes needs to be de-emphasized while counting the cumulative match across many dimensions. Of course, such an approach poses challenges for low-dimensional data, because the cumulative impact of matching cannot be counted in a statistically robust way with a small number of dimensions. Therefore, an approach is needed that can automatically adjust to the dimensionality of the data.

With increasing dimensionality, a record is likely to contain both relevant and irrelevant features. A pair of semantically similar objects may contain feature values that are dissimilar (at the level of one standard deviation along that dimension) because of the noisy variations in irrelevant features. Conversely, a pair of objects are unlikely to have similar values across many attributes, just by chance, unless these attributes were relevant. Interestingly, the Euclidean metric (and L_p-norm in general) achieves exactly the opposite effect by using the squared sum of the difference in attribute values. As a result, the "noise" components from the irrelevant attributes dominate the computation and mask the similarity effects of a large number of relevant attributes. The L_∞-norm provides an extreme example of this effect where the dimension with the largest distance value is used. In high-dimensional domains such as text, similarity functions such as the cosine measure (discussed in Sect. 3.3), tend to emphasize the cumulative effect of matches on many attribute values rather than large distances along individual attributes. This general principle can also be used for quantitative data.

One way of de-emphasizing precise levels of dissimilarity is to use *proximity thresholding* in a dimensionality-sensitive way. To perform proximity thresholding, the data are discretized into equidepth buckets. Each dimension is divided into k_d equidepth buckets, containing a fraction $1/k_d$ of the records. The number of buckets, k_d, is dependent on the data dimensionality d.

Let $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$ be two d-dimensional records. Then, for dimension i, if both x_i and y_i belong to the same bucket, the two records are said to be in proximity on dimension i. The subset of dimensions on which $\overline{X}$ and $\overline{Y}$ map to the same bucket is referred to as the proximity set, and it is denoted by $\mathcal{S}(\overline{X}, \overline{Y}, k_d)$. Furthermore, for each dimension $i \in \mathcal{S}(\overline{X}, \overline{Y}, k_d)$, let m_i and n_i be the upper and lower bounds of the bucket in dimension i, in which the two records are proximate to one another. Then, the

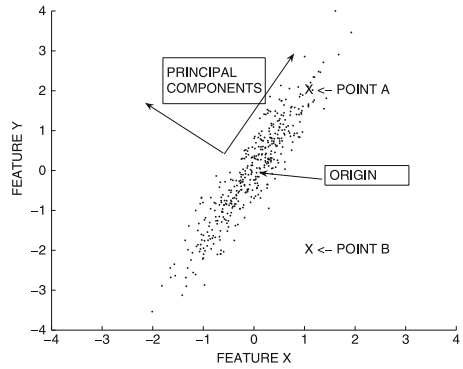


Figure 3.3: Global data distributions impact distance computations

similarity $PSelect(\overline{X}, \overline{Y}, k_d)$ is defined as follows:

$$PSelect(\overline{X}, \overline{Y}, k_d) = \left[\sum_{i \in \mathcal{S}(\overline{X}, \overline{Y}, k_d)} \left(1 - \frac{|x_i - y_i|}{m_i - n_i}\right)^p \right]^{1/p}. \tag{3.5}$$

The value of the aforementioned expression will vary between 0 and $|\mathcal{S}(\overline{X}, \overline{Y}, k_d)|$ because each individual expression in the summation lies between 0 and 1. This is a *similarity* function because larger values imply greater similarity.

The aforementioned similarity function guarantees a nonzero similarity component only for dimensions mapping to the same bucket. The use of equidepth partitions ensures that the probability of two records sharing a bucket for a particular dimension is given by $1/k_d$. Thus, on average, the aforementioned summation is likely to have d/k_d nonzero components. For more similar records, the number of such components will be greater, and each individual component is also likely to contribute more to the similarity value. The degree of dissimilarity on the distant dimensions is ignored by this approach because it is often dominated by noise. It has been shown theoretically [7] that picking $k_d \propto d$ achieves a constant level of contrast in high-dimensional space for certain data distributions. High values of k_d result in more stringent quality bounds for each dimension. These results suggest that in high-dimensional space, it is better to aim for higher quality bounds for each dimension, so that a smaller percentage (not number) of retained dimensions are used in similarity computation. An interesting aspect of this distance function is the nature of its sensitivity to data dimensionality. The choice of k_d with respect to d ensures that for low-dimensional applications, it bears some resemblance to the L_p-norm by using most of the dimensions; whereas for high-dimensional applications, it behaves similar to text domain-like similarity functions by using similarity on matching attributes. The distance function has also been shown to be more effective for a prototypical nearest-neighbor classification application.

3.2.1.6 Impact of Data Distribution

The L_p-norm depends only on the two data points in its argument and is invariant to the global statistics of the remaining data points. Should distances depend on the underlying data distribution of the remaining points in the data set? The answer is yes. To illustrate this point, consider the distribution illustrated in Fig. 3.3 that is centered at the origin. In

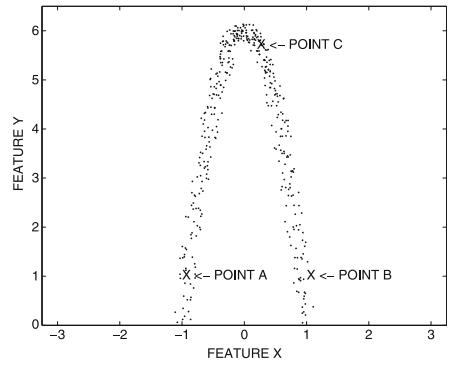


Figure 3.4: Impact of nonlinear distributions on distance computations

addition, two data points A= $(1, 2)$ and B= $(1, -2)$ are marked in the figure. Clearly, A and B are equidistant from the origin according to any L_p-norm. However, a question arises, as to whether A and B should truly be considered equidistant from the origin O. This is because the straight line from O to A is aligned with a *high-variance* direction in the data, and statistically, it is more *likely* for data points to be further away in this direction. On the other hand, many segments of the path from O to B are sparsely populated, and the corresponding direction is a *low-variance* direction. Statistically, it is much less likely for B to be so far away from O along this direction. Therefore, the distance from O to A *ought* to be less than that of O to B.

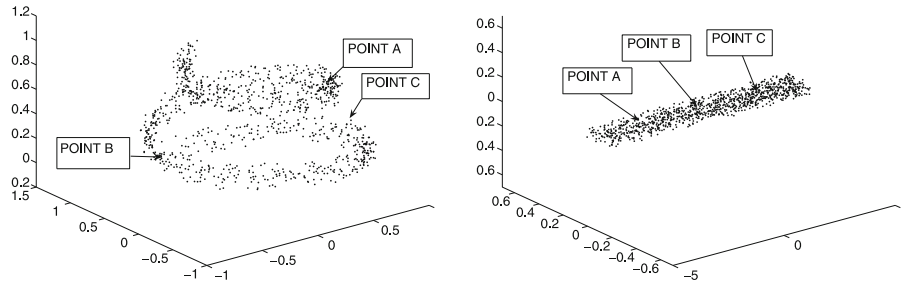
The *Mahalanobis distance* is based on this general principle. Let Σ be its $d \times d$ covariance matrix of the data set. In this case, the (i, j)th entry of the covariance matrix is equal to the covariance between the dimensions i and j. Then, the Mahalanobis distance $Maha(\overline{X}, \overline{Y})$ between two d-dimensional data points $\overline{X}$ and $\overline{Y}$ is as follows:

$$Maha(\overline{X}, \overline{Y}) = \sqrt{(\overline{X} - \overline{Y})\Sigma^{-1}(\overline{X} - \overline{Y})^T}.$$

A different way of understanding the Mahalanobis distance is in terms of principal component analysis (*PCA*). The Mahalanobis distance is similar to the Euclidean distance, except that it normalizes the data on the basis of the interattribute correlations. For example, if the axis system were to be rotated to the principal directions of the data (shown in Fig. 3.3), then the data would have no (second order) interattribute correlations. The Mahalanobis distance is equivalent to the Euclidean distance in such a transformed (axes-rotated) data set *after* dividing each of the transformed coordinate values by the standard deviation of the data along that direction. As a result, the data point B will have a larger distance from the origin than data point A in Fig. 3.3.

3.2.1.7 Nonlinear Distributions: ISOMAP

We now examine the case in which the data contain nonlinear distributions of arbitrary shape. For example, consider the global distribution illustrated in Fig. 3.4. Among the three data points A, B, and C, which pair are the closest to one another? At first sight, it would seem that data points A and B are the closest on the basis of Euclidean distance. However, the global data distribution tells us otherwise. One way of understanding distances is as the shortest length of the path from one data point to another, when using only point-to-point jumps from data points to one of their k-nearest neighbors based on a standard metric



(a) A and C seem close (original data)

(b) A and C are actually far away (*ISOMAP* embedding)

Figure 3.5: Impact of *ISOMAP* embedding on distances

such as the Euclidean measure. The intuitive rationale for this is that only *short* point-to-point jumps can accurately measure minor changes in the generative process for that point. Therefore, the overall sum of the point-to-point jumps reflects the aggregate change (distance) from one point to another (distant) point more accurately than a straight-line distance between the points. Such distances are referred to as *geodesic distances*. In the case of Fig. 3.4, the only way to walk from A to B with short point-to-point jumps is to walk along the entire elliptical shape of the data distribution while passing C along the way. Therefore, A and B are actually the *farthest* pair of data points (from A, B, and C) on this basis! The implicit assumption is that nonlinear distributions are *locally* Euclidean but are *globally* far from Euclidean.

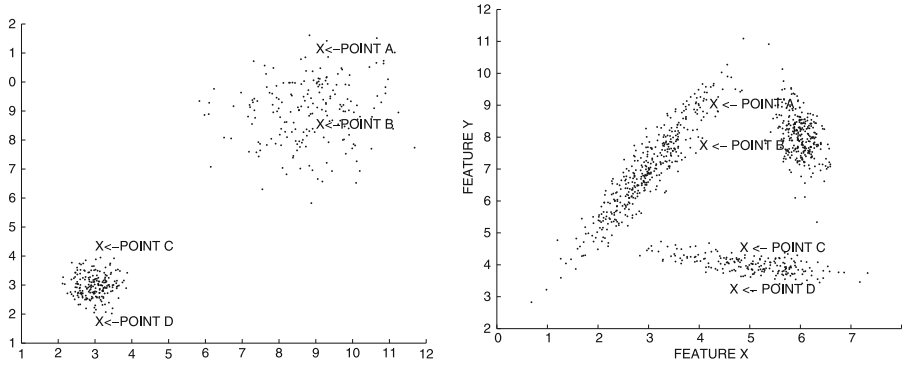
Such distances can be computed by using an approach that is derived from a nonlinear dimensionality reduction and embedding method, known as *ISOMAP*. The approach consists of two steps:

1. Compute the k-nearest neighbors of each point. Construct a weighted graph G with nodes representing data points, and edge weights (costs) representing distances of these k-nearest neighbors.

2. For any pair of points $\overline{X}$ and $\overline{Y}$, report $Dist(\overline{X}, \overline{Y})$ as the shortest path between the corresponding nodes in G.

These two steps are already able to compute the distances without explicitly performing dimensionality reduction. However, an additional step of embedding the data into a multidimensional space makes *repeated* distance computations between many pairs of points much faster, while losing some accuracy. Such an embedding also allows the use of algorithms that work naturally on numeric multidimensional data with predefined distance metrics.

This is achieved by using the all-pairs shortest-path problem to construct the full set of distances between any pair of nodes in G. Subsequently, *multidimensional scaling (MDS)* (cf. Sect. 2.4.4.2 of Chap. 2) is applied to embed the data into a lower dimensional space. The overall effect of the approach is to "straighten out" the nonlinear shape of Fig. 3.4 and embed it into a space where the data are aligned along a flat strip. In fact, a 1-dimensional representation can approximate the data after this transformation. Furthermore, in this new space, a distance function such as the Euclidean metric will work very well as long as metric *MDS* was used in the final phase. A 3-dimensional example is illustrated in Fig. 3.5a, in which the data are arranged along a spiral. In this figure, data points A and C seem much



(a) local density variation (b) local orientation variation

Figure 3.6: Impact of local distributions on distance computations

closer to each other than data point B. However, in the *ISOMAP* embedding of Fig. 3.5b, the data point B is much closer to each of A and C. This example shows the drastic effect of data distributions on distance computation.

In general, high-dimensional data are aligned along nonlinear low-dimensional shapes, which are also referred to as *manifolds*. These manifolds can be "flattened out" to a new representation where metric distances can be used effectively. Thus, this is a data transformation method that facilitates the use of standard metrics. The major computational challenge is in performing the dimensionality reduction. However, after the one-time preprocessing cost has been paid for, repeated distance computations can be implemented efficiently.

Nonlinear embeddings can also be achieved with extensions of *PCA*. *PCA* can be extended to discovering nonlinear embeddings with the use of a method known as the *kernel trick*. Refer to Sect. 10.6.4.1 of Chap. 10 for a brief description of kernel *PCA*.

3.2.1.8 Impact of Local Data Distribution

The discussion so far addresses the impact of global distributions on the distance computations. However, the distribution of the data varies significantly with locality. This variation may be of two types. For example, the absolute density of the data may vary significantly with data locality, or the shape of clusters may vary with locality. The first type of variation is illustrated in Fig. 3.6a, which has two clusters containing the same number of points, but one of them is denser than the other. Even though the absolute distance between (A, B) is identical to that between (C, D), the distance between C and D should be considered greater on the basis of the *local* data distribution. In other words, C and D are much farther away in the *context* of what their local distributions look like. This problem is often encountered in many distance-based methods such as outlier detection. It has been shown that methods that adjust for the local variations in the distances typically perform much better than those that do not adjust for local variations. One of the most well-known methods for outlier detection, known as Local Outlier Factor (*LOF*), is based on this principle.

A second example is illustrated in Fig. 3.6b, which illustrates the impact of varying local orientation of the clusters. Here, the distance between (A, B) is identical to that between (C, D) using the Euclidean metric. However, the local clusters in each region show very different orientation. The high-variance axis of the cluster of data points relevant to (A, B)

is aligned along the path from A to B. This is not true for (C, D). As a result, the intrinsic distance between C and D is much greater than that between A and B. For example, if the *local* Mahalanobis distance is computed using the relevant cluster covariance statistics, then the distance between C and D will evaluate to a larger value than that between A and B.

Shared Nearest-Neighbor Similarity: The first problem can be at least partially alleviated with the use of a shared nearest-neighbor similarity. In this approach, the k-nearest neighbors of each data point are computed in a preprocessing phase. The shared nearest-neighbor similarity is equal to the number of common neighbors between the two data points. This metric is locally sensitive because it depends on the number of common *neighbors*, and not on the absolute values of the distances. In dense regions, the k-nearest neighbor distances will be small, and therefore data points need to be closer together to have a larger number of shared nearest neighbors. Shared nearest-neighbor methods can be used to define a *similarity* graph on the underlying data points in which pairs of data points with at least one shared neighbor have an edge between them. Similarity graph-based methods are almost always locality sensitive because of their local focus on the k-nearest neighbor distribution.

Generic Methods: In generic local distance computation methods, the idea is to divide the space into a set of local regions. The distances are then adjusted in each region using the local statistics of this region. Therefore, the broad approach is as follows:

1. Partition the data into a set of local regions.
2. For any pair of objects, determine the most relevant region for the pair, and compute the pairwise distances using the local statistics of that region. For example, the local Mahalanobis distance may be used in each local region.

A variety of clustering methods are used for partitioning the data into local regions. In cases where each of the objects in the pair belongs to a different region, either the global distribution may be used, or the average may be computed using both local regions. Another problem is that the first step of the algorithm (partitioning process) itself requires a notion of distances for clustering. This makes the solution circular, and calls for an iterative solution. Although a detailed discussion of these methods is beyond the scope of this book, the bibliographic notes at the end of this chapter provide a number of pointers.

3.2.1.9 Computational Considerations

A major consideration in the design of distance functions is the computational complexity. This is because distance function computation is often embedded as a subroutine that is used repeatedly in the application at hand. If the subroutine is not efficiently implementable, the applicability becomes more restricted. For example, methods such as *ISOMAP* are computationally expensive and hard to implement for very large data sets because these methods scale with at least the square of the data size. However, they do have the merit that a one-time transformation can create a representation that can be used efficiently by data mining algorithms. Distance functions are executed repeatedly, whereas the preprocessing is performed only once. Therefore, it is definitely advantageous to use a preprocessing-intensive approach as long as it speeds up later computations. For many applications, sophisticated methods such as *ISOMAP* may be too expensive even for one-time analysis. For such cases, one of the earlier methods discussed in this chapter may need to be used. Among the methods discussed in this section, carefully chosen L_p-norms and match-based techniques are the fastest methods for large-scale applications.

3.2.2 Categorical Data

Distance functions are naturally computed as functions of value differences along dimensions in numeric data, which is *ordered*. However, no ordering exists among the discrete values of categorical data. How can distances be computed? One possibility is to transform the categorical data to numeric data with the use of the binarization approach discussed in Sect. 2.2.2.2 of Chap. 2. Because the binary vector is likely to be sparse (many zero values), similarity functions can be adapted from other sparse domains such as text. For the case of categorical data, it is more common to work with similarity functions rather than distance functions because discrete values can be matched more naturally.

Consider two records $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$. The simplest possible similarity between the records $\overline{X}$ and $\overline{Y}$ is the sum of the similarities on the individual attribute values. In other words, if $S(x_i, y_i)$ is the similarity between the attributes values x_i and y_i, then the overall similarity is defined as follows:

$$Sim(\overline{X}, \overline{Y}) = \sum_{i=1}^{d} S(x_i, y_i).$$

Therefore, the choice of $S(x_i, y_i)$ defines the overall similarity function.

The simplest possible choice is to set $S(x_i, y_i)$ to 1 when $x_i = y_i$ and 0 otherwise. This is also referred to as the *overlap* measure. The major drawback of this measure is that it does not account for the relative frequencies among the different attributes. For example, consider a categorical attribute in which the attribute value is "Normal" for 99 % of the records, and either "Cancer" or "Diabetes" for the remaining records. Clearly, if two records have a "Normal" value for this variable, this does not provide statistically significant information about the similarity, because the majority of pairs are likely to show that pattern just by chance. However, if the two records have a matching "Cancer" or "Diabetes" value for this variable, it provides significant statistical evidence of similarity. This argument is similar to that made earlier about the importance of the global data distribution. Similarities or differences that are unusual are statistically more significant than those that are common.

In the context of categorical data, the *aggregate statistical properties* of the data set should be used in computing similarity. This is similar to how the Mahalanobis distance was used to compute similarity more accurately with the use of global statistics. The idea is that matches on unusual values of a categorical attribute should be weighted more heavily than values that appear frequently. This also forms the underlying principle of many common normalization techniques that are used in domains such as text. An example, which is discussed in the next section, is the use of *inverse document frequency (IDF)* in the information retrieval domain. An analogous measure for categorical data will be introduced here.

The *inverse occurrence frequency* is a generalization of the simple matching measure. This measure weights the similarity between the matching attributes of two records by an inverse function of the frequency of the matched value. Thus, when $x_i = y_i$, the similarity $S(x_i, y_i)$ is equal to the inverse weighted frequency, and 0 otherwise. Let $p_k(x)$ be the fraction of records in which the kth attribute takes on the value of x in the data set. In other words, when $x_i = y_i$, the value of $S(x_i, y_i)$ is $1/p_k(x_i)^2$ and 0 otherwise.

$$S(x_i, y_i) = \begin{cases} 1/p_k(x_i)^2 & \text{if } x_i = y_i \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

A related measure is the *Goodall* measure. As in the case of the inverse occurrence frequency, a higher similarity value is assigned to a match when the value is infrequent. In a simple variant of this measure [104], the similarity on the kth attribute is defined as $1 - p_k(x_i)^2$, when $x_i = y_i$, and 0 otherwise.

$$S(x_i, y_i) = \begin{cases} 1 - p_k(x_i)^2 & \text{if } x_i = y_i \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

The bibliographic notes contain pointers to various similarity measures for categorical data.

3.2.3 Mixed Quantitative and Categorical Data

It is fairly straightforward to generalize the approach to mixed data by adding the weights of the numeric and quantitative components. The main challenge is in deciding how to assign the weights of the quantitative and categorical components. For example, consider two records $\overline{X} = (\overline{X_n}, \overline{X_c})$ and $\overline{Y} = (\overline{Y_n}, \overline{Y_c})$ where $\overline{X_n}$, $\overline{Y_n}$ are the subsets of numerical attributes and $\overline{X_c}$, $\overline{Y_c}$ are the subsets of categorical attributes. Then, the overall similarity between $\overline{X}$ and $\overline{Y}$ is defined as follows:

$$Sim(\overline{X}, \overline{Y}) = \lambda \cdot NumSim(\overline{X_n}, \overline{Y_n}) + (1 - \lambda) \cdot CatSim(\overline{X_c}, \overline{Y_c}). \tag{3.8}$$

The parameter λ regulates the relative importance of the categorical and numerical attributes. The choice of λ is a difficult one. In the absence of domain knowledge about the relative importance of attributes, a natural choice is to use a value of λ that is equal to the fraction of numerical attributes in the data. Furthermore, the proximity in numerical data is often computed with the use of distance functions rather than similarity functions. However, distance values can be converted to similarity values as well. For a distance value of $dist$, a common approach is to use a kernel mapping that yields [104] the similarity value of $1/(1 + dist)$.

Further normalization is required to meaningfully compare the similarity value components on the numerical and categorical attributes that may be on completely different scales. One way of achieving this goal is to determine the standard deviations in the similarity values over the two domains with the use of sample pairs of records. Each component of the similarity value (numerical or categorical) is divided by its standard deviation. Therefore, if σ_c and σ_n are the standard deviations of the similarity values in the categorical and numerical components, then Eq. 3.8 needs to be modified as follows:

$$Sim(\overline{X}, \overline{Y}) = \lambda \cdot NumSim(\overline{X_n}, \overline{Y_n})/\sigma_n + (1 - \lambda) \cdot CatSim(\overline{X_c}, \overline{Y_c})/\sigma_c. \tag{3.9}$$

By performing this normalization, the value of λ becomes more meaningful, as a true *relative weight* between the two components. By default, this weight can be set to be proportional to the number of attributes in each component unless specific domain knowledge is available about the relative importance of attributes.

3.3 Text Similarity Measures

Strictly speaking, text can be considered quantitative multidimensional data when it is treated as a bag of words. The frequency of each word can be treated as a quantitative attribute, and the base lexicon can be treated as the full set of attributes. However, the

structure of text is *sparse* in which most attributes take on 0 values. Furthermore, all word frequencies are nonnegative. This special structure of text has important implications for similarity computation and other mining algorithms. Measures such as the L_p-norm do not adjust well to the varying length of the different documents in the collection. For example, the L_2-distance between two long documents will almost always be larger than that between two short documents even if the two long documents have many words in common, and the short documents are completely disjoint. How can one normalize for such irregularities? One way of doing so is by using the cosine measure. The cosine measure computes the *angle* between the two documents, which is insensitive to the absolute length of the document. Let $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$ be two documents on a lexicon of size d. Then, the cosine measure $\cos(\overline{X}, \overline{Y})$ between $\overline{X}$ and $\overline{Y}$ can be defined as follows:

$$\cos(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}}. \tag{3.10}$$

The aforementioned measure simply uses the raw frequencies between attributes. However, as in other data types, it is possible to use global statistical measures to improve the similarity computation. For example, if two documents match on an uncommon word, it is more indicative of similarity than the case where two documents match on a word that occurs very commonly. The *inverse document frequency* id_i, which is a decreasing function of the number of documents n_i in which the ith word occurs, is commonly used for normalization:

$$id_i = \log(n/n_i). \tag{3.11}$$

Here, the number of documents in the collection is denoted by n. Another common adjustment is to ensure that the excessive presence of single word does not throw off the similarity measure. A damping function $f(\cdot)$, such as the square root or the logarithm, is optionally applied to the frequencies before similarity computation.

$$f(x_i) = \sqrt{x_i}$$
$$f(x_i) = \log(x_i)$$

In many cases, the damping function is not used, which is equivalent to setting $f(x_i)$ to x_i. Therefore, the *normalized* frequency $h(x_i)$ for the ith word may be defined as follows:

$$h(x_i) = f(x_i) \cdot id_i. \tag{3.12}$$

Then, the cosine measure is defined as in Eq. 3.10, except that the normalized frequencies of the words are used:

$$\cos(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} h(x_i) \cdot h(y_i)}{\sqrt{\sum_{i=1}^{d} h(x_i)^2} \cdot \sqrt{\sum_{i=1}^{d} h(y_i)^2}}. \tag{3.13}$$

Another measure that is less commonly used for text is the *Jaccard coefficient* $J(\overline{X}, \overline{Y})$:

$$J(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} h(x_i) \cdot h(y_i)}{\sum_{i=1}^{d} h(x_i)^2 + \sum_{i=1}^{d} h(y_i)^2 - \sum_{i=1}^{d} h(x_i) \cdot h(y_i)}. \tag{3.14}$$

The Jaccard coefficient is rarely used for the text domain, but it is used commonly for sparse binary data sets.

3.3.1 Binary and Set Data

Binary multidimensional data are a representation of set-based data, where a value of 1 indicates the presence of an element in a set. Binary data occur commonly in market-basket domains in which transactions contain information corresponding to whether or not an item is present in a transaction. It can be considered a special case of text data in which word frequencies are either 0 or 1. If S_X and S_Y are two sets with binary representations $\overline{X}$ and $\overline{Y}$, then it can be shown that applying Eq. 3.14 to the raw binary representation of the two sets is equivalent to:

$$J(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sum_{i=1}^{d} x_i^2 + \sum_{i=1}^{d} y_i^2 - \sum_{i=1}^{d} x_i \cdot y_i} = \frac{|S_X \cap S_Y|}{|S_X \cup S_Y|}. \tag{3.15}$$

This is a particularly intuitive measure because it carefully accounts for the number of common and disjoint elements in the two sets.

3.4 Temporal Similarity Measures

Temporal data contain a single contextual attribute representing time and one or more behavioral attributes that measure the properties varying along a particular time period. Temporal data may be represented as continuous time series, or as discrete sequences, depending on the application domain. The latter representation may be viewed as the discrete version of the former. It should be pointed out that discrete sequence data are not always temporal because the contextual attribute may represent placement. This is typically the case in biological sequence data. Discrete sequences are also sometimes referred to as *strings*. Many of the similarity measures used for time series and discrete sequences can be reused across either domain, though some of the measures are more suited to one of the domains. Therefore, this section will address both data types, and each similarity measure will be discussed in a subsection on either continuous series or discrete series, based on its most common use. For some measures, the usage is common across both data types.

3.4.1 Time-Series Similarity Measures

The design of time-series similarity measures is highly application specific. For example, the simplest possible similarity measure between two time series of equal length is the Euclidean metric. Although such a metric may work well in many scenarios, it does not account for several distortion factors that are common in many applications. Some of these factors are as follows:

1. *Behavioral attribute scaling and translation:* In many applications, the different time series may not be drawn on the same scales. For example, the time series representing various stocks prices may show similar patterns of movements, but the absolute values may be very different both in terms of the mean and the standard deviation. For example, the share prices of several different hypothetical stock tickers are illustrated in Fig. 3.7. All three series show similar patterns but with different scaling and some random variations. Clearly, they show similar patterns but cannot be meaningfully compared if the absolute values of the series are used.

2. *Temporal (contextual) attribute translation:* In some applications, such as real-time analysis of financial markets, the different time series may represent the same periods

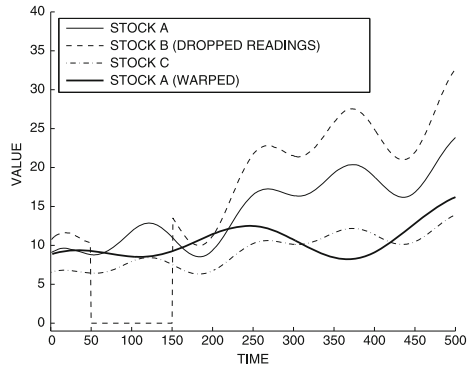


Figure 3.7: Impact of scaling, translation, and noise

in time. In other applications, such as the analysis of the time series obtained from medical measurements, the absolute time stamp of when the reading was taken is not important. In such cases, the temporal attribute value needs to be shifted in at least one of the time series to allow more effective matching.

3. *Temporal (contextual) attribute scaling:* In this case, the series may need to be stretched or compressed along the temporal axis to allow more effective matching. This is referred to as *time warping*. An additional complication is that different temporal segments of the series may need to be warped differently to allow for better matching. In Fig. 3.7, the simplest case of warping is shown where the entire set of values for stock A has been stretched. In general, the time warping can be more complex where different windows in the same series may be stretched or compressed differently. This is referred to as *dynamic* time warping (DTW).

4. *Noncontiguity in matching:* Long time series may have noisy segments that do not match very well with one another. For example, one of the series in Fig. 3.7 has a window of dropped readings because of data collection limitations. This is common in sensor data. The distance function may need to be robust to such noise.

Some of these issues can be addressed by attribute normalization during preprocessing.

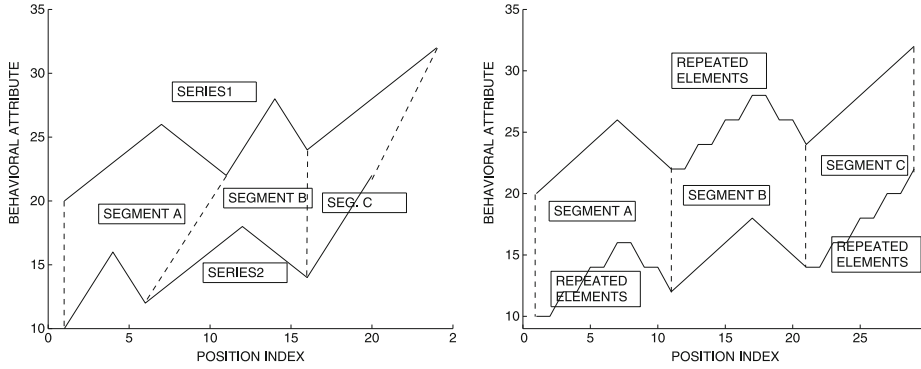
3.4.1.1 Impact of Behavioral Attribute Normalization

The translation and scaling issues are often easier to address for the behavioral attributes as compared to contextual attributes, because they can be addressed by normalization during preprocessing:

1. *Behavioral attribute translation:* The behavioral attribute is mean centered during preprocessing.

2. *Behavioral attribute scaling:* The standard deviation of the behavioral attribute is scaled to 1 unit.

It is important to remember that these normalization issues may not be relevant to every application. Some applications may require only translation, only scaling, or neither of the two. Other applications may require both. In fact, in some cases, the wrong choice of



(a) Original series (b) Warped series

Figure 3.8: Illustration of dynamic time warping by repeating elements

normalization may have detrimental effects on the interpretability of the results. Therefore, an analyst needs to judiciously select a normalization approach depending on application-specific needs.

3.4.1.2 L_p-Norm

The L_p-norm may be defined for two series $\overline{X} = (x_1 \ldots x_n)$ and $\overline{Y} = (y_1 \ldots y_n)$. This measure treats a time series as a multidimensional data point in which each time stamp is a dimension.

$$Dist(\overline{X}, \overline{Y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{3.16}$$

The L_p-norm can also be applied to wavelet transformations of the time series. In the special case where $p = 2$, accurate distance computations are obtained with the wavelet representation, if most of the larger wavelet coefficients are retained in the representation. In fact, it can be shown that if no wavelet coefficients are removed, then the distances are identical between the two representations. This is because wavelet transformations can be viewed as a rotation of an axis system in which each dimension represents a time stamp. Euclidean metrics are invariant to axis rotation. The major problem with L_p-norms is that they are designed for time series of equal length and cannot address distortions on the temporal (contextual) attributes.

3.4.1.3 Dynamic Time Warping Distance

DTW stretches the series along the time axis in a varying (or *dynamic*) way over different portions to enable more effective matching. An example of warping is illustrated in Fig. 3.8a, where the two series have very similar shape in segments A, B, and C, but specific segments in each series need to be stretched appropriately to enable better matching. The DTW measure has been adapted from the field of speech recognition, where time warping was deemed necessary to match different speaking speeds. DTW can be used either for time-series or sequence data, because it addresses only the issue of contextual attribute scaling, and it is unrelated to the nature of the behavioral attribute. The following description is a generic one, which can be used either for time-series or sequence data.

The L_p-metric can only be defined between two time series of equal length. However, DTW, by its very nature, allows the measurement of distances between two series of *different* lengths. In the L_p distance, a one-to-one mapping exists between the time stamps of the two time series. However, in DTW, a many-to-one mapping is allowed to account for the time warping. This many-to-one mapping can be thought of in terms of repeating some of the elements in carefully chosen segments of either of the two time series. This can be used to artificially create two series of the same length that have a one-to-one mapping between them. The distances can be measured on the resulting warped series using any distance measure such as the L_p-norm. For example, in Fig. 3.8b, some elements in a few segments of either series are repeated to create a one-to-one mapping between the two series. Note that the two series now look much more similar than the two series in Fig. 3.8a. Of course, this repeating can be done in many different ways, and the goal is to perform it in an *optimal* way to minimize the DTW distance. The optimal choice of warping is determined using dynamic programming.

To understand how DTW generalizes a one-to-one distance metric such as the L_p-norm, consider the L_1 (Manhattan) metric $M(\overline{X_i}, \overline{Y_i})$, computed on the first i elements of two time series $\overline{X} = (x_1 \ldots x_n)$ and $\overline{Y} = (y_1 \ldots y_n)$ of equal length. The value of $M(\overline{X_i}, \overline{Y_i})$ can be written *recursively* as follows:

$$M(\overline{X_i}, \overline{Y_i}) = |x_i - y_i| + M(\overline{X_{i-1}}, \overline{Y_{i-1}}). \tag{3.17}$$

Note that the indices of *both* series are reduced by 1 in the right-hand side because of the one-to-one matching. In DTW, both indices need not reduce by 1 unit because a many-to-one mapping is allowed. Rather, any one or both indices may reduce by 1, depending on the *best match* between the two time series (or sequences). The index that did *not* reduce by 1 corresponds to the repeated element. The choice of index reduction is naturally defined, recursively, as an optimization over the various options.

Let $DTW(i, j)$ be the optimal distance between the first i and first j elements of two time series $\overline{X} = (x_1 \ldots x_m)$ and $\overline{Y} = (y_1 \ldots y_n)$, respectively. Note that the two time series are of lengths m and n, which may not be the same. Then, the value of $DTW(i, j)$ is defined recursively as follows:

$$DTW(i,j) = distance(x_i, y_j) + \min \begin{cases} DTW(i, j-1) & \text{repeat } x_i \\ DTW(i-1, j) & \text{repeat } y_j \\ DTW(i-1, j-1) & \text{repeat neither} \end{cases}. \tag{3.18}$$

The value of $distance(x_i, y_j)$ may be defined in a variety of ways, depending on the application domain. For example, for continuous time series, it may be defined as $|x_i - y_i|^p$, or by a distance that accounts for (behavioral attribute) scaling and translation. For discrete sequences, it may be defined using a categorical measure. The DTW approach is primarily focused on warping the *contextual* attribute, and has little to do with the nature of the behavioral attribute or distance function. Because of this fact, time warping can easily be extended to multiple behavioral attributes by simply using the distances along multiple attributes in the recursion.

Equation 3.18 yields a natural iterative approach. The approach starts by initializing $DTW(0, 0)$ to 0, $DTW(0, j)$ to ∞ for $j \in \{1 \ldots n\}$, and $DTW(i, 0)$ to ∞ for $i \in \{1 \ldots m\}$. The algorithm computes $DTW(i, j)$ by repeatedly executing Eq. 3.18 with increasing index values of i and j. This can be achieved by a simple nested loop in which the indices i and j increase from 1 to m and 1 to n, respectively:

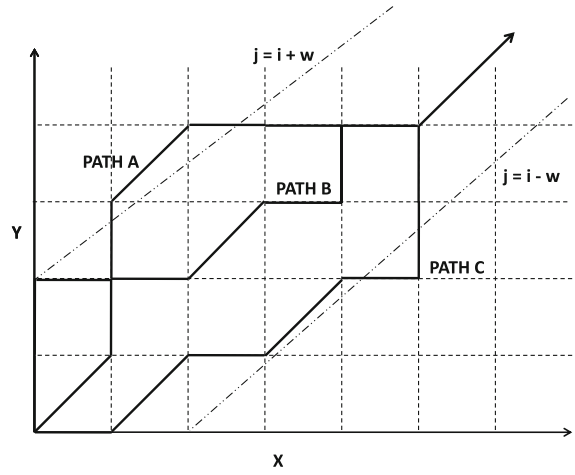


Figure 3.9: Illustration of warping paths

for $i = 1$ to m
  for $j = 1$ to n
    compute $DTW(i, j)$ using Eq. 3.18

The aforementioned code snippet is a nonrecursive and iterative approach. It is also possible to implement a recursive computer program by directly using Eq. 3.18. Therefore, the approach requires the computation of all values of $DTW(i, j)$ for every $i \in [1, m]$ and every $j \in [1, n]$. This is a $m \times n$ grid of values, and therefore the approach may require $O(m \cdot n)$ iterations, where m and n are lengths of the series.

The optimal warping can be understood as an *optimal path* through different values of i and j in the $m \times n$ grid of values, as illustrated in Fig. 3.9. Three possible paths, denoted by A, B, and C, are shown in the figure. These paths only move to the right (increasing i and repeating y_j), upward (increasing j and repeating x_i), or both (repeating neither).

A number of practical constraints are often added to the DTW computation. One commonly used constraint is the *window constraint* that imposes a minimum level w of positional alignment between matched elements. The window constraint requires that $DTW(i, j)$ be computed only when $|i - j| \leq w$. Otherwise, the value may be set to ∞ by default. For example, the paths B and C in Fig. 3.9 no longer need to be computed. This saves the computation of many values in the dynamic programming recursion. Correspondingly, the computations in the inner variable j of the nested loop above can be saved by constraining the index j, so that it is never more than w units apart from the outer loop variable i. Therefore, the inner loop index j is varied from $\max\{0, i - w\}$ to $\min\{n, i + w\}$.

The DTW distance can be extended to multiple behavioral attributes easily, if it is assumed that the different behavioral attributes have the same time warping. In this case, the recursion is unchanged, and the only difference is that $distance(\overline{x_i}, \overline{y_j})$ is computed using a vector-based distance measure. We have used a bar on $\overline{x_i}$ and $\overline{y_j}$ to denote that these are vectors of multiple behavioral attributes. This multivariate extension is discussed in Sect. 16.3.4.1 of Chap. 16 for measuring distances between 2-dimensional trajectories.

3.4.1.4 Window-Based Methods

The example in Fig. 3.7 illustrates a case where dropped readings may cause a gap in the matching. Window-based schemes attempt to decompose the two series into windows and then "stitch" together the similarity measure. The intuition here is that if two series have many contiguous matching segments, they should be considered similar. For long time series, a global match becomes increasingly unlikely. The only reasonable choice is the use of windows for measurement of segment-wise similarity.

Consider two time series $\overline{X}$ and $\overline{Y}$, and let $\overline{X_1} \ldots \overline{X_r}$ and $\overline{Y_1} \ldots \overline{Y_r}$ be temporally ordered and nonoverlapping windows extracted from the respective series. Note that some windows from the base series may not be included in these segments at all. These correspond to the noise segments that are dropped. Then, the overall similarity between $\overline{X}$ and $\overline{Y}$ can be computed as follows:

$$Sim(\overline{X}, \overline{Y}) = \sum_{i=1}^{r} Match(\overline{X_i}, \overline{Y_i}). \tag{3.19}$$

A variety of measures discussed in this section may be used to instantiate the value of $Match(\overline{X_i}, \overline{Y_i})$. It is tricky to determine the proper value of $Match(\overline{X_i}, \overline{Y_i})$ because a contiguous match along a long window is more unusual than many short segments of the same length. The proper choice of $Match(\overline{X_i}, \overline{Y_i})$ may depend on the application at hand. Another problem is that the optimal decomposition of the series into windows may be a difficult task. These methods are not discussed in detail here, but the interested reader is referred to the bibliographic notes for pointers to relevant methods.

3.4.2 Discrete Sequence Similarity Measures

Discrete sequence similarity measures are based on the same general principles as time-series similarity measures. As in the case of time-series data, discrete sequence data may or may not have a one-to-one mapping between the positions. When a one-to-one mapping does exist, many of the multidimensional categorical distance measures can be adapted to this domain, just as the L_p-norm can be adapted to continuous time series. However, the application domains of discrete sequence data are most often such that a one-to-one mapping does not exist. Aside from the DTW approach, a number of other dynamic programming methods are commonly used.

3.4.2.1 Edit Distance

The edit distance defines the distance between two strings as the *least* amount of "effort" (or *cost*) required to transform one sequence into another by using a series of transformation operations, referred to as "edits." The edit distance is also referred to as the *Levenshtein* distance. The edit operations include the use of symbol insertions, deletions, and replacements with specific costs. In many models, replacements are assumed to have higher cost than insertions or deletions, though insertions and deletions are usually assumed to have the same cost. Consider the sequences *abababababab* and *bababababa*, which are drawn on the alphabet $\{a, b\}$. The first string can be transformed to the second in several ways. For example, if every alphabet in the first string was replaced by the other alphabet, it would result in the second string. The cost of doing so is that of ten replacements. However, a more cost-efficient way of achieving the same goal is to delete the leftmost element of the string, and insert the symbol "a" as the rightmost element. The cost of this sequence of operations is only one insertion and one deletion. The edit distance is defined as the optimal cost to

transform one string to another with a sequence of insertions, deletions, and replacements. The computation of the optimal cost requires a dynamic programming recursion.

For two sequences $\overline{X} = (x_1 \ldots x_m)$ and $\overline{Y} = (y_1 \ldots y_n)$, let the edits be performed on sequence $\overline{X}$ to transform to $\overline{Y}$. Note that this distance function is asymmetric because of the directionality to the edit. For example, $Edit(\overline{X}, \overline{Y})$ may not be the same as $Edit(\overline{Y}, \overline{X})$ if the insertion and deletion costs are not identical. In practice, however, the insertion and deletion costs are assumed to be the same.

Let I_{ij} be a binary indicator that is 0 when the ith symbol of $\overline{X}$ and jth symbols of $\overline{Y}$ are the same. Otherwise, the value of this indicator is 1. Then, consider the first i symbols of $\overline{X}$ and the first j symbols of $\overline{Y}$. Assume that these segments are represented by $\overline{X_i}$ and $\overline{Y_j}$, respectively. Let $Edit(i, j)$ represent the optimal matching cost between these segments. The goal is to determine what operation to perform on the last element of $\overline{X_i}$ so that it either matches an element in $\overline{Y_j}$, or it is deleted. Three possibilities arise:

1. The last element of $\overline{X_i}$ is deleted, and the cost of this is $[Edit(i-1, j)+\text{Deletion Cost}]$. The last element of the truncated segment $\overline{X_{i-1}}$ may or may not match the last element of $\overline{Y_j}$ at this point.

2. An element is inserted at the end of $\overline{X_i}$ to match the last element of Y_j, and the cost of this is $[Edit(i, j-1) + \text{Insertion Cost}]$. The indices of the edit term $Edit(i, j-1)$ reflect the fact that the matched elements of both series can now be removed.

3. The last element of $\overline{X_i}$ is flipped to that of $\overline{Y_j}$ *if it is different*, and the cost of this is $[Edit(i-1, j-1) + I_{ij} \cdot (\text{Replacement Cost})]$. In cases where the last elements are the same, the additional replacement cost is not incurred, but progress is nevertheless made in matching. This is because the matched elements (x_i, y_j) of both series need not be considered further, and residual matching cost is $Edit(i-1, j-1)$.

Clearly, it is desirable to pick the minimum of these costs for the optimal matching. Therefore, the optimal matching is defined by the following recursion:

$$Edit(i, j) = \min \begin{cases} Edit(i-1, j) + \text{Deletion Cost} \\ Edit(i, j-1) + \text{Insertion Cost} \\ Edit(i-1, j-1) + I_{ij} \cdot (\text{Replacement Cost}) \end{cases} . \tag{3.20}$$

Furthermore, the bottom of the recursion also needs to be set up. The value of $Edit(i, 0)$ is equal to the cost of i deletions for any value of i, and that of $Edit(0, j)$ is equal to the cost of j insertions for any value of j. This nicely sets up the dynamic programming approach. It is possible to write the corresponding computer program either as a nonrecursive nested loop (as in DTW) or as a recursive computer program that directly uses the aforementioned cases.

The aforementioned discussion assumes general insertion, deletion, and replacement costs. In practice, however, the insertion and deletion costs are usually assumed to be the same. In such a case, the edit function is symmetric because it does not matter which of the two strings is edited to the other. For any sequence of edits from one string to the other, a reverse sequence of edits, with the same cost, will exist from the other string to the first.

The edit distance can be extended to numeric data by changing the primitive operations of *insert*, *delete*, and *replace* to transformation rules that are designed for time series. Such transformation rules can include making basic changes to the shape of the time series in

window segments. This is more complex because it requires one to design the base set of allowed time-series shape transformations and their costs. Such an approach has not found much popularity for time-series distance computation.

3.4.2.2 Longest Common Subsequence

A *subsequence* of a sequence is a set of symbols drawn from the sequence in the same order as the original sequence. A subsequence is different from a *substring* in that the values of the subsequence need not be contiguous, whereas the values in the substring need to be contiguous. Consider the sequences $agbfcgdhei$ and $afbgchdiei$. In this case, ei is a substring of both sequences and also a subsequence. However, $abcde$ and fgi are subsequences of both strings but not substrings. Clearly, subsequences of longer length are indicative of a greater level of matching between the strings. Unlike the edit distance, the longest common subsequence ($LCSS$) is a similarity function because higher values indicate greater similarity. The number of possible subsequences is exponentially related to the length of a string. However, the LCSS can be computed in polynomial time with a dynamic programming approach.

For two sequences $\overline{X} = (x_1 \ldots x_m)$ and $\overline{Y} = (y_1 \ldots y_n)$, consider the first i symbols of $\overline{X}$ and the first j symbols of $\overline{Y}$. Assume that these segments are represented by $\overline{X_i}$ and $\overline{Y_j}$, respectively. Let $LCSS(i, j)$ represent the optimal LCSS values between these segments. The goal here is to either match the last element of $\overline{X_i}$ and $\overline{Y_j}$, or delete the last element in one of the two sequences. Two possibilities arise:

1. The last element of $\overline{X_i}$ matches $\overline{Y_j}$, in which case, it cannot hurt to instantiate the matching on the last element and then delete the last element of both sequences. The similarity value $LCSS(i, j)$ can be expressed recursively as this is $LCSS(i-1, j-1)+1$.

2. The last element does not match. In such a case, the last element of at least one of the two strings needs to be deleted under the assumption that it cannot occur in the matching. In this case, the value of $LCSS(i, j)$ is either $LCSS(i, j-1)$ or $LCSS(i-1, j)$, depending on which string is selected for deletion.

Therefore, the optimal matching can be expressed by enumerating these cases:

$$LCSS(i,j) = \max \begin{cases} LCSS(i-1, j-1) + 1 & \text{only if } x_i = y_j \\ LCSS(i-1, j) & \text{otherwise (no match on } x_i) \\ LCSS(i, j-1) & \text{otherwise (no match on } y_j) \end{cases}. \tag{3.21}$$

Furthermore, the boundary conditions need to be set up. The values of $LCSS(i, 0)$ and $LCSS(0, j)$ are always equal to 0 for any value of i and j. As in the case of the DTW and edit-distance computations, a nested loop can be set up to compute the final value. A recursive computer program can also be implemented that uses the aforementioned recursive relationship. Although the $LCSS$ approach is defined for a discrete sequence, it can also be applied to a continuous time series after discretizing the time-series values into a sequence of categorical values. Alternatively, one can discretize the time-series *movement* between two contiguous time stamps. The particular choice of discretization depends on the goals of the application at hand.

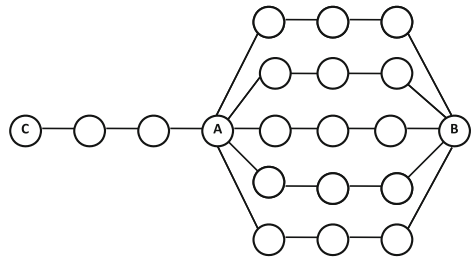


Figure 3.10: Shortest path versus homophily

3.5 Graph Similarity Measures

The similarity in graphs can be measured in different ways, depending on whether the similarity is being measured between two graphs, or between two nodes in a single graph. For simplicity, undirected networks are assumed, though the measures can be easily generalized to directed networks.

3.5.1 Similarity between Two Nodes in a Single Graph

Let $G = (N, A)$ be an undirected network with node set N and edge set A. In some domains, costs are associated with nodes, whereas in others, weights are associated with nodes. For example, in domains such as bibliographic networks, the edges are naturally weighted, and in road networks, the edges naturally have costs. Typically, *distance* functions work with costs, whereas *similarity* functions work with weights. Therefore, it may be assumed that either the cost c_{ij}, or the weight w_{ij} of the edge (i, j) is specified. It is often possible to convert costs into weights (and vice versa) using simple heuristic kernel functions that are chosen in an application-specific way. An example is the heat kernel $K(x) = e^{-x^2/t^2}$.

It is desired to measure the similarity between any pair of nodes i and j. The principle of similarity between two nodes in a single graph is based on the concept of *homophily* in real networks. The principle of homophily is that nodes are typically more similar in a network when they are connected to one another with edges. This is common in many domains such as the Web and social networks. Therefore, nodes that are connected via *short paths* and *many paths* should be considered more similar. The latter criterion is closely related to the concept of connectivity between nodes. The first criterion is relatively easy to implement with the use of the shortest-path algorithm in networks.

3.5.1.1 Structural Distance-Based Measure

The goal here is to measure the distances from any source node s to any other node in the network. Let $SP(s, j)$ be the shortest-path distance from source node s to any node j. The value of $SP(s, j)$ is initialized to 0 for $j = s$ and ∞ otherwise. Then, the distance computation of s to all other nodes in the network may be summarized in a single step that is performed exactly once for each node in the network in a certain order:

- Among all nodes not examined so far, select the node i with the smallest value of $SP(s, i)$ and update the distance labels of each of its neighbors j as follows:

$$SP(s, j) = \min\{SP(s, j), SP(s, i) + c_{ij}\}. \tag{3.22}$$

This is the essence of the well-known Dijkstra algorithm. This approach is linear in the number of edges in the network, because it examines each node and its incident edges exactly once. The approach provides the distances from a single node to all other nodes in a single pass. The final value of $SP(s, j)$ provides a quantification of the structural distance between node s and node j. Structural distance-based measures do not leverage the multiplicity in paths between a pair of nodes because they focus only on the raw structural distances.

3.5.1.2 Random Walk-Based Similarity

The structural measure of the previous section does not work well when pairs of nodes have varying numbers of paths between them. For example, in Fig. 3.10, the shortest-path length between nodes A and B is 4, whereas that between A and C is 3. Yet, node B should be considered more similar to A because the two nodes are more tightly connected with a *multiplicity* of paths. The idea of random walk-based similarity is based on this principle.

In random walk-based similarity, the approach is as follows: Imagine a random walk that starts at source node s, and proceeds to an adjacent node with weighted probability proportional to w_{ij}. Furthermore, at any given node, it is allowed to "jump back" to the source node s with a probability referred to as the *restart probability*. This will result in a probability distribution that is heavily biased toward the source node s. Nodes that are more similar to s will have higher probability of visits. Such an approach will adjust very well to the scenario illustrated in Fig. 3.10 because the walk will visit B more frequently.

The intuition here is the following: If you were lost in a road network and drove randomly, while taking turns randomly, which location are you more *likely* to reach? You are more likely to reach a location that is close by *and* can be reached in multiple ways. The random-walk measure therefore provides a result that is different from that of the shortest-path measure because it also accounts for multiplicity in paths during similarity computation.

This similarity computation is closely related to concept of *PageRank*, which is used to rank pages on the Web by search engines. The corresponding modification for measuring similarity between nodes is also referred to as *personalized PageRank*, and a symmetric variant is referred to as *SimRank*. This chapter will not discuss the details of *PageRank* and *SimRank* computation, because it requires more background on the notion of ranking. Refer to Sect. 18.4 of Chap. 18, which provides a more complete discussion.

3.5.2 Similarity Between Two Graphs

In many applications, multiple graphs are available, and it is sometimes necessary to determine the distances between multiple graphs. A complicating factor in similarity computation is that many nodes may have the same label, which makes them indistinguishable. Such cases arise often in domains such as chemical compound analysis. Chemical compounds can be represented as graphs where nodes are elements, and bonds are edges. Because an element may be repeated in a molecule, the labels on the nodes are not distinct. Determining a similarity measure on graphs is extremely challenging in this scenario, because even the very special case of determining whether the two graphs are identical is hard. The latter problem is referred to as the *graph isomorphism* problem, and is known to the NP-hard [221]. Numerous measures, such as the graph-edit distance and substructure-based similarity, have been proposed to address this very difficult case. The core idea in each of these methods is as follows:

1. *Maximum common subgraph distance:* When two graphs contain a large subgraph in common, they are generally considered more similar. The maximum common subgraph problem and the related distance functions are addressed in Sect. 17.2 of Chap. 17.

2. *Substructure-based similarity:* Although it is difficult to match two large graphs, it is much easier to match smaller substructures. The core idea is to count the frequently occurring substructures between the two graphs and report it as a similarity measure. This can be considered the graph analog of subsequence-based similarity in strings. Substructure-based similarity measures are discussed in detail in Sect. 17.3 of Chap. 17.

3. *Graph-edit distance:* This distance measure is analogous to the string-edit distance and is defined as the number of edits required to transform one graph to the other. Because graph matching is a hard problem, this measure is difficult to implement for large graphs. The graph-edit distance is discussed in detail in Sect. 17.2.3.2 of Chap. 17.

4. *Graph kernels:* Numerous kernel functions have been defined to measure similarity between graphs, such as the shortest-path kernel and the random-walk kernel. This topic is discussed in detail in Sect. 17.3.3 of Chap. 17.

These methods are quite complex and require a greater background in the area of graphs. Therefore, the discussion of these measures is deferred to Chap. 17 of this book.

3.6 Supervised Similarity Functions

The previous sections discussed similarity measures that do not require any understanding of user intentions. In practice, the relevance of a feature or the choice of distance function heavily depends on the domain at hand. For example, for an image data set, should the color feature or the texture feature be weighted more heavily? These aspects cannot be modeled by a distance function without taking the user intentions into account. Unsupervised measures, such as the L_p-norm, treat all features equally, and have little intrinsic understanding of the end user's *semantic* notion of similarity. The only way to incorporate this information into the similarity function is to use explicit feedback about the similarity and dissimilarity of objects. For example, the feedback can be expressed as the following sets of object pairs:

$$\mathcal{S} = \{(O_i, O_j) : O_i \text{ is similar to } O_j\}$$
$$\mathcal{D} = \{(O_i, O_j) : O_i \text{ is dissimilar to } O_j\}.$$

How can this information be leveraged to improve the computation of similarity? Many specialized methods have been designed for supervised similarity computation. A common approach is to assume a specific closed form of the similarity function for which the parameters need to be learned. An example is the weighted L_p-norm in Sect. 3.2.1.1, where the parameters represented by Θ correspond to the feature weights $(a_1 \ldots a_d)$. Therefore, the first step is to create a distance function $f(O_i, O_j, \Theta)$, where Θ is a set of unknown weights. Assume that higher values of the function indicate greater dissimilarity. Therefore, this is a distance function, rather than a similarity function. Then, it is desirable to determine the

parameters Θ, so that the following conditions are satisfied as closely as possible:

$$f(O_i, O_j, \Theta) = \begin{cases} 0 & \text{if } (O_i, O_j) \in \mathcal{S} \\ 1 & \text{if } (O_i, O_j) \in \mathcal{D} \end{cases}. \tag{3.23}$$

This can be expressed as a least squares optimization problem over Θ, with the following error E:

$$E = \sum_{(O_i, O_j) \in \mathcal{S}} (f(O_i, O_j, \Theta) - 0)^2 + \sum_{(O_i, O_j) \in \mathcal{D}} (f(O_i, O_j, \Theta) - 1)^2. \tag{3.24}$$

This objective function can be optimized with respect to Θ with the use of any off-the-shelf optimization solver. If desired, the additional constraint $\Theta \geq 0$ can be added where appropriate. For example, when Θ represents the feature weights $(a_1 \ldots a_d)$ in the Minkowski metric, it is natural to make the assumption of nonnegativity of the coefficients. Such a constrained optimization problem can be easily solved using many nonlinear optimization methods. The use of a closed form such as $f(O_i, O_j, \Theta)$ ensures that the function $f(O_i, O_j, \Theta)$ can be computed efficiently after the one-time cost of computing the parameters Θ.

Where possible, user feedback should be used to improve the quality of the distance function. The problem of learning distance functions can be modeled more generally as that of classification. The classification problem will be studied in detail in Chaps. 10 and 11. Supervised distance function design with the use of Fisher's method is also discussed in detail in the section on instance-based learning in Chap. 10.

3.7 Summary

The problem of distance function design is a crucial one in the context of data mining applications. This is because many data mining algorithms use the distance function as a key subroutine, and the design of the function directly impacts the quality of the results. Distance functions are highly sensitive to the type of the data, the dimensionality of the data, and the global and local nature of the data distribution.

The L_p-norm is the most common distance function used for multidimensional data. This distance function does not seem to work well with increasing dimensionality. Higher values of p work particularly poorly with increasing dimensionality. In some cases, it has been shown that fractional metrics are particularly effective when p is chosen in the range $(0, 1)$. Numerous proximity-based measures have also been shown to work effectively with increasing dimensionality.

The data distribution also has an impact on the distance function design. The simplest possible distance function that uses global distributions is the Mahalanobis metric. This metric is a generalization of the Euclidean measure, and stretches the distance values along the principal components according to their variance. A more sophisticated approach, referred to as *ISOMAP*, uses nonlinear embeddings to account for the impact of nonlinear data distributions. Local normalization can often provide more effective measures when the distribution of the data is heterogeneous.

Other data types such as categorical data, text, temporal, and graph data present further challenges. The determination of time-series and discrete-sequence similarity measures is closely related because the latter can be considered the categorical version of the former. The main problem is that two similar time series may exhibit different scaling of their behavioral and contextual attributes. This needs to be accounted for with the use of different normalization functions for the behavioral attribute, and the use of warping functions for the

contextual attribute. For the case of discrete sequence data, many distance and similarity functions, such as the edit distance and the LCSS, are commonly used.

Because distance functions are often intended to model user notions of similarity, feedback should be used, where possible, to improve the distance function design. This feedback can be used within the context of a parameterized model to learn the optimal parameters that are consistent with the user-provided feedback.

3.8 Bibliographic Notes

The problem of similarity computation has been studied extensively by data mining researchers and practitioners in recent years. The issues with high-dimensional data were explored in [17, 88, 266]. In the work of [88], the impact of the distance concentration effects on high-dimensional computation was analyzed. The work in [266] showed the relative advantages of picking distance functions that are locality sensitive. The work also showed the advantages of the Manhattan metric over the Euclidean metric. Fractional metrics were proposed in [17] and generally provide more accurate results than the Manhattan and Euclidean metric. The *ISOMAP* method discussed in this chapter was proposed in [490]. Numerous local methods are also possible for distance function computation. An example of an effective local method is the instance-based method proposed in [543].

Similarity in categorical data was explored extensively in [104]. In this work, a number of similarity measures were analyzed, and how they apply to the outlier detection problem was tested. The Goodall measure is introduced in [232]. The work in [122] uses information theoretic measures for computation of similarity. Most of the measures discussed in this chapter do not distinguish between mismatches on an attribute. However, a number of methods proposed in [74, 363, 473] distinguish between mismatches on an attribute value. The premise is that infrequent attribute values are statistically expected to be more different than frequent attribute values. Thus, in these methods, $S(x_i, y_i)$ is not always set to 0 (or the same value) when x_i and y_i are different. A local similarity measure is presented in [182]. Text similarity measures have been studied extensively in the information retrieval literature [441].

The area of time-series similarity measures is a rich one, and a significant number of algorithms have been designed in this context. An excellent tutorial on the topic may be found in [241]. The use of wavelets for similarity computation in time series is discussed in [130]. While DTW has been used extensively in the context of speech recognition, its use in data mining applications was first proposed by [87]. Subsequently, it has been used extensively [526] for similarity-based applications in data mining. The major challenge in data mining applications is its computationally intensive nature. Numerous methods [307] have been proposed in the time series data mining literature to speed up DTW. A fast method for computing a lower bound on DTW was proposed in [308], and how this can be used for exact indexing was shown. A window-based approach for computing similarity in sequences with noise, scaling, and translation was proposed in [53]. Methods for similarity search in multivariate time series and sequences were proposed in [499, 500]. The edit distance has been used extensively in biological data for computing similarity between sequences [244]. The use of transformation rules for time-series similarity has been studied in [283, 432]. Such rules can be used to create edit distance-like measures for continuous time series. Methods for the string-edit distance are proposed in [438]. It has been shown in [141], how the L_p-norm may be combined with the edit distance. Algorithms for the LCSS problem may be found in [77, 92, 270, 280]. A survey of these algorithms is available

in [92]. A variety of other measures for time series and sequence similarity are discussed in [32].

Numerous methods are available for similarity search in graphs. A variety of efficient shortest-path algorithms for finding distances between nodes may be found in [62]. The page rank algorithm is discussed in the Web mining book [357]. The NP-hardness of the graph isomorphism problem, and other closely related problems to the edit distance are discussed in [221]. The relationship between the maximum common subgraph problem and the graph-edit distance problem has been studied in [119, 120]. The problem of substructure similarity search, and the use of substructures for similarity search have been addressed in [520, 521]. A notion of mutation distance has been proposed in [522] to measure the distances between graphs. A method that uses the frequent substructures of a graph for similarity computation in clustering is proposed in [42]. A survey on graph-matching techniques may be found in [26].

User supervision has been studied extensively in the context of distance function learning. One of the earliest methods that parameterizes the weights of the L_p-norm was proposed in [15]. The problem of distance function learning has been formally related to that of classification and has been studied recently in great detail. A survey that covers the important topics in distance function learning is provided in [33].

3.9 Exercises

1. Compute the L_p-norm between $(1, 2)$ and $(3, 4)$ for $p = 1, 2, \infty$.

2. Show that the Mahalanobis distance between two data points is equivalent to the Euclidean distance on a transformed data set, where the transformation is performed by representing the data along the principal components, and dividing by the standard deviation of each component.

3. Download the *Ionosphere data set* from the *UCI Machine Learning Repository* [213], and compute the L_p distance between all pairs of data points, for $p = 1, 2$, and ∞. Compute the contrast measure on the data set for the different norms. Repeat the exercise after sampling the first r dimensions, where r varies from 1 to the full dimensionality of the data.

4. Compute the match-based similarity, cosine similarity, and the Jaccard coefficient, between the two sets $\{A, B, C\}$ and $\{A, C, D, E\}$.

5. Let $\overline{X}$ and $\overline{Y}$ be two data points. Show that the cosine angle between the vectors $\overline{X}$ and $\overline{Y}$ is given by:

$$cosine(\overline{X}, \overline{Y}) = \frac{||\overline{X}||^2 + ||\overline{Y}||^2 - ||\overline{X} - \overline{Y}||^2}{2||\overline{X}||||\overline{Y}||}. \tag{3.25}$$

6. Download the *KDD Cup Network Intrusion Data Set* for the *UCI Machine Learning Repository* [213]. Create a data set containing only the categorical attributes. Compute the nearest neighbor for each data point using the (a) match measure, and (b) inverse occurrence frequency measure. Compute the number of cases where there is a match on the class label.

7. Repeat Exercise 6 using only the quantitative attributes of the data set, and using the L_p-norm for values of $p = 1, 2, \infty$.

8. Repeat Exercise 6 using all attributes in the data set. Use the mixed-attribute function, and different combinations of the categorical and quantitative distance functions of Exercises 6 and 7.

9. Write a computer program to compute the edit distance.

10. Write a computer program to compute the $LCSS$ distance.

11. Write a computer program to compute the DTW distance.

12. Assume that $Edit(\overline{X}, \overline{Y})$ represents the cost of transforming the string $\overline{X}$ to $\overline{Y}$. Show that $Edit(\overline{X}, \overline{Y})$ and $Edit(\overline{Y}, \overline{X})$ are the same, as long as the insertion and deletion costs are the same.

13. Compute the edit distance, and $LCSS$ similarity between: (a) *ababcabc* and *babcbc* and (b) *cbacbacba* and *acbacbacb*. For the edit distance, assume equal cost of insertion, deletion, or replacement.

14. Show that $Edit(i, j)$, $LCSS(i, j)$, and $DTW(i, j)$ are all monotonic functions in i and j.

15. Compute the cosine measure using the raw frequencies between the following two sentences:

(a) "The sly fox jumped over the lazy dog."

(b) "The dog jumped at the intruder."

16. Suppose that insertion and deletion costs are 1, and replacement costs are 2 units for the edit distance. Show that the optimal edit distance between two strings can be computed only with insertion and deletion operations. Under the aforementioned cost assumptions, show that the optimal edit distance can be expressed as a function of the optimal $LCSS$ distance and the lengths of the two strings.