Exercises

1. Prove that the DFT is a unitary transformation.
2. Show how the DFT can be computed in $O(n \log n)$ time for an n-dimensional feature vector.
3. In [588], curves (termed *trails*) formed by a sequence of points are decomposed into subcurves (termed *subtrails*) formed by subsequences of points. Note that the subcurves are not connected. How would you decompose the curve into subcurves? What is a good criterion for the decomposition process? Recall that the subcurves are indexed by an R*-tree. Thus, are the conditions that make for a good R-tree decomposition (i.e., minimum overlap and minimum total area) also valid here? Alternatively, you may want to look at monotonicity of the subcurves.
4. Show that the DFT is a linear transformation.
5. Show that a shift in the time domain is equivalent to a rotation of the coefficients of the DFT.

4.6.5 Summary

Dimension reduction techniques have a number of drawbacks with respect to the ability to use them in conjunction with indexing methods so as to facilitate similarity searching:

1. As we have seen, search accuracy can suffer if the pruning property is not satisfied.
2. Most dimension reduction methods are tied to a specific distance metric (typically L_2, as in SVD/KLT/PCA discussed in Section 4.6.4.1 and DFT discussed in Section 4.6.4.2) and do not guarantee that the pruning property is satisfied for other distance metrics.
3. In order to be effective, they require that the data be strongly correlated, which means that, in the high-dimensional space, there are only a few independent dimensions, while the rest of the dimensions depend on them. This results in a loss of precision as many distinct points will be potentially mapped to the same representative point in the lower-dimension transformed space, which will result in degraded search performance.
4. Even if we reduce the number of dimensions, the resulting number may still be quite high with respect to the number of dimensions for which conventional indexing methods work well.
5. There are challenges in making them work in a dynamic environment, as the quality of the transformation may degrade (i.e., newly added data may not follow the same distribution as existing data), and thus it may be preferable to compute a new transformation based on the entire database. However, some work has been done to address this challenge (e.g., [988] for SVD).

4.7 Embedding Methods

Embedding methods are designed to facilitate similarity queries in an environment where we are given a finite set of N objects and a distance metric d indicating the distance values between them (collectively termed a *finite metric space*). As we discuss in Section 4.5, there are many applications where the cost of evaluating the distance between two objects is very high. Thus, the number of distance evaluations should be kept to a minimum, while (ideally) maintaining the quality of the result. At times, this distance function is represented by an $N \times N$ similarity matrix containing the distance d between every pair of objects. One way to approach this goal is to *embed* the data objects in a vector space so that the distances of the embedded objects as measured by a distance metric d' approximate the actual distances. Thus, queries can be performed (for the most part) on

the embedded objects. In this section, we are especially interested in examining the issue of whether the embedding method returns the same query result as would be returned if the actual distances of the objects were used, thus ensuring that no relevant objects are left out (i.e., there are no false dismissals). In this context, particular attention is paid to SparseMap [886], a variant of Lipschitz embeddings [1181], and FastMap [587], which is inspired by the linear transformations discussed in Section 4.6.4.1 (e.g., KLT and the equivalent PCA and SVD). We also discuss locality sensitive hashing (LSH) [924] (Section 4.7.4), which is a method that combines an embedding into a Hamming space with a set of random projections into a subspace of the Hamming space. The key to its utility for similarity searching is its deployment in randomized approximate nearest neighbor algorithms. All three of these methods can be applied to arbitrary metric spaces.

This section is organized as follows. Section 4.7.1 contains an introduction to embedding methods, including a brief overview of multidimensional scaling. Section 4.7.2 describes Lipschitz embeddings, where the coordinate axes correspond to reference sets consisting of subsets of the objects. This includes a discussion of SparseMap. Section 4.7.3 presents the FastMap method. Section 4.7.4 discusses locality sensitive hashing in the context of randomized algorithms. As we point out earlier, embedding methods such as, locality sensitive hashing could have also been discussed in the context of dimension reduction techniques. We have placed them here as our emphasis is on how the distances in the embedding space approximate the actual distances.

Although it is not explicitly mentioned in the original presentations of some of the embedding methods (e.g., FastMap [587] and SparseMap [886]), we shall see below that it is desirable that the mapping F be such that distance functions d and d' obey the pruning property. Recall from Section 4.6.1 that the pruning property is satisfied if and only if $d'(F(a),F(b)) \leq d(a,b)$ for all pairs of objects a and b. Satisfaction of the pruning property is useful for responding to similarity queries (such as nearest neighbors) when using a spatial index that has been built on the result of the mapping. This is because the pruning property ensures 100% recall, which guarantees that no correct responses are missed. The bulk of the presentation in this section is drawn from the analysis of Hjaltason and Samet [850, 855] of the extent to which various embedding methods satisfy the pruning property.

Nevertheless, despite our focus on the extent to which F satisfies the pruning property (i.e., F is contractive), we also briefly discuss the alternative or additional notion of constructing an embedding that is proximity preserving. This turns out to be harder than merely requiring that the embedding be contractive. As we saw in Section 4.6.1.2, the proximity-preserving property is not commonly satisfied. Therefore, there has been some interest in devising embeddings that are approximately proximity preserving. One example of such a mapping is known as *BoostMap* and was developed by Athitsos, Alon, Sclaroff, and Kollios [106] (see also extensions [105, 107, 108]). BoostMap uses machine learning techniques to construct the mapping from a collection of one-dimensional embeddings that are classified as weakly proximity preserving. A mapping is said to be *weakly proximity preserving* if we know that, given triples of elements s, t, and u in the underlying data domain S, the following properties are both either true or false for more than half of all of the triples of elements:

1. $d(s,u) \leq d(t,u)$
2. $|f(s) - f(u)| \leq |f(t) - f(u)|$

In other words, whenever u is closer to s than to t, we want $f(u)$ to be closer to $f(s)$ than to $f(t)$ and vice versa (i.e., when $f(u)$ is closer to $f(s)$ than to $f(t)$, we want u to be closer to s than to t). The notion of being weakly proximity preserving is used to capture the fact that this property is necessarily satisfied for all triples in S.

Some possible weakly proximity preserving one-dimensional mappings include $f_r(s) = d(r,s)$, where r and s are both in S, and r serves as a parameter reference object, termed a *pivot*, which means that in effect we are computing the distance of the objects

with respect to r. Another possibility, among many others, is to take the analog of a projection of an object s onto the line joining two parameter reference objects (i.e., two pivots) p and q—that is,

$$f_{p,q}(s) = \frac{d(p,s)^2 + d(p,q)^2 - d(q,s)^2}{2 \cdot d(p,q)}$$

The interpretation of this mapping as a projection only makes sense if the underlying domain S is a vector space; of course, the mapping is still valid regardless of the nature of S.

The AdaBoost machine learning technique (e.g., [648, 1682]) is used by BoostMap to construct an optimal linear combination of k of these weakly proximity preserving one-dimensional mappings, where "optimal" is with respect to the proximity-preserving property. In particular, the k one-dimensional mappings that make up the linear combination are used as the coordinates of an embedding into $\mathbb{R}^k$, and the coefficients a_j $(1 \leq j \leq k)$ of the linear combination are used as the coefficients for a generalized L_1 distance, $D(X,Y) = \sum_{j=1}^{k} a_j \cdot |Y_j - X_j|$, where X and Y are in $\mathbb{R}^k$ (an image of S).

With this embedding and easily computable distance function, we can use an appropriate indexing technique, coupled with a filter-and-refine strategy, to perform nearest neighbor queries, although we may not achieve 100% recall as there is no guarantee that the embedding is completely proximity preserving. Note that the construction process of the embedding that we have described does not make any provisions for contractiveness or for a limited amount of distortion (i.e., the ratio of the distance in the embedding space to that in the original space). Of course, it is desirable for an embedding to be both contractive and proximity preserving, although neither is necessary for the other—that is, an embedding can be contractive while not being proximity preserving. Similarly, an embedding can be proximity preserving while not being contractive.

4.7.1 Introduction

The motivation behind the development of embedding methods is that, given a sufficiently high value of k, we can find a function F that maps the N objects into a vector space of dimensionality k, such that the distances between the points are approximately preserved when using a distance function d' in the k-dimensional space. In other words, for any pair of objects a and b, we have $d(a,b) \approx d'(F(a),F(b))$. Formally, the mapping F embeds a finite metric space (S,d) into a vector space $\mathbb{R}^k$ possessing a distance metric d', where $S \subset U$ is a dataset of N objects drawn from a universe U, and $d : U \to \mathbb{R}^+$ is the original distance function on U. The mapping $F : S \to \mathbb{R}^k$ is often said to be an *embedding*, and, if d' is the Euclidean distance metric, then F is a *Euclidean embedding*. Note that this problem setting also includes the situation where the N objects are described by an n-dimensional feature vector (this is an example of the case that the original distance function d is computed from the objects rather than existing in a distance matrix). In this case, the mapping F reduces the dimensionality of the problem setting from n to k.

In practice, our goal is to use a relatively low value for k in the mapping (i.e., $k \ll N$), while still achieving reasonable distance preservation. Observe that distance computation can be expensive (e.g., the Euclidean distance in very high-dimensional spaces), so the mapping F should ideally be fast to compute (i.e., take $O(N)$ time or $O(N \log N)$ time for the N objects rather than requiring the computation of the distances between all $O(N^2)$ pairs of objects), should preserve distance to a reasonable extent, and should provide a fast way of obtaining the k-dimensional point corresponding to the object.

At times, the distances between the objects are preserved exactly by the mapping F—that is, $d(a,b) = d'(F(a),F(b))$ for all objects $a,b \in S$. Of course, in this case, F is also proximity preserving. When this occurs, we say that (S,d) and $(\mathbb{R}^k,d')$ are *isometric* (strictly speaking, (S,d) is isometric to $(F(S),d')$, where $F(S) \subset \mathbb{R}^k$ is the image of S

under F). For example, this is possible when the data objects are originally drawn from a vector space, and d and d' are both Euclidean distance metrics. In this particular case, distance preservation among the N objects is ensured when $k = N - 1$ (but a much lower value suffices in most cases). However, usually, the distances cannot be preserved exactly for an arbitrary combination of d and d', regardless of the value of k (i.e., there is no guarantee that a distance-preserving mapping F exists).

As an example of a situation where distances cannot be preserved exactly, suppose that we are given four objects a, b, c, and e with a distance function d such that the distance between each pair in $\{a,b,c\}$ is 2, and the distance from e to each of a, b, and c is 1.1. This distance function d satisfies the triangle inequality. However, these four objects cannot be embedded into a three-dimensional Euclidean space (i.e., where d' is the Euclidean distance metric). In other words, we cannot position the objects in a three-dimensional space so that the Euclidean distance d' between the positions corresponds to the distance between the objects given by d. On the other hand, if the distance between e and the three remaining objects is at least $2/\sqrt{3}$, then such a positioning is possible by placing a, b, and c in a plane p and placing e on the line perpendicular to p that passes through the centroid of the triangle in p formed by a, b, and c.

Interestingly, the above embedding can be achieved if we use the City Block distance metric. In particular, we position objects a, b, and c at locations (0,0,0), (2,0,0), and (1,1,0), respectively, and e at (1,0,0.1). Thus, we see that we can often obtain better distance correspondence by being flexible in choosing the distance function d'. In fact, it is always possible to achieve exact distance preservation when d' is the Chessboard metric (L_∞). In one such embedding, there is one dimension for each object o_i, where $o_1, o_2, \ldots, o_N$ is an enumeration of the objects. Each object o is mapped by F into the vector $\{d(o,o_1), d(o,o_2), \ldots, d(o,o_N)\}$. For any pair of objects o_i and o_j, their distance in the embedding is $d'(F(o_i), F(o_j)) = d_M(F(o_i), F(o_j)) = \max\{|F(o_i) - F(o_j)|\} = \max_l\{|d(o_i,o_l) - d(o_j,o_l)|\}$. Observe that for any l, $|d(o_i,o_l) - d(o_j,o_l)| \leq d(o_i,o_j)$ by the triangle inequality, while equality is achieved for $l = i$ and $l = j$, and thus distances are indeed preserved by F when using the Chessboard metric.

Notice that the number of dimensions in this distance-preserving embedding is rather high (i.e., $k = N$). Thus, we may want to define F in terms of a subset of the objects as done by Faragó, Linder, and Lugosi [595], although they do not indicate how to choose the k objects that form the subset. However, for lower values of k, it is possible that a better approximation of the original distances is obtained by a choice of d' other than L_∞. The same type of embedding is also used by Vleugels and Veltkamp [1934], who employ an application-dependent method to choose the k reference objects (e.g., in an application involving hue, the dominant colors of each of the objects differ), which are called *vantage objects*.

In most applications, it is not possible to achieve distance preservation given a particular choice for d' and k. In fact, we are often satisfied when the distance d' in the k-dimensional embedding space between the points that correspond to any pair of individual objects a and b is less than the true distance d between the objects in the original space—that is, $d'(F(a), F(b)) \leq d(a,b)$. This property is known as the *pruning property* and is discussed in Section 4.6.1. As pointed out earlier, satisfaction of the pruning property is useful for responding to similarity queries (such as nearest neighbors) when using a spatial index that has been built on the result of the mapping. This is because the pruning property ensures 100% recall, which guarantees that no correct responses are missed.

The concept of *distortion* (e.g., [1181]) is frequently used for measuring the quality of an embedding procedure (i.e., a method that constructs a mapping F) or of a particular embedding F produced by such a procedure. Distortion measures how much larger or smaller the distances in the embedding space $d'(F(o_1), F(o_2))$ are than the corresponding distances $d(o_1,o_2)$ in the original space. In particular, the distortion is defined as c_1c_2 when we are guaranteed that

$$\frac{1}{c_1} \cdot d(o_1, o_2) \leq d'(F(o_1), F(o_2)) \leq c_2 \cdot d(o_1, o_2) \tag{4.17}$$

for all pairs of objects $o_1, o_2 \in S$, where $c_1, c_2 \geq 1$. In other words, the distance values $d'(F(o_1), F(o_2))$ in the embedding space may be as much as a factor of c_1 smaller and a factor of c_2 larger than the actual distances $d(o_1, o_2)$. Note that for a given embedding procedure, there may be no upper or lower bound on the distance ratio for the embeddings that it constructs, so c_1, c_2, or both may be infinite in this case. Of course, the distortion is always bounded for any given embedding F and finite metric space (S,d). A number of general results are known about embeddings—for example, that any finite metric space can be embedded in Euclidean space with $O(\log N)$ distortion [1181].

Another common measure of a particular embedding F with respect to a dataset S is *stress* [1090]. Stress measures the overall deviation in the distances (i.e., the extent to which they differ) and is typically defined in terms of variance:

$$\frac{\sum_{o_1,o_2}(d'(F(o_1), F(o_2)) - d(o_1, o_2))^2}{\sum_{o_1,o_2} d(o_1, o_2)^2}$$

Alternative definitions of stress may be more appropriate for certain applications. For example, the sum in the denominator may be on $d'(F(o_1), F(o_2))^2$, or the division by $d(o_1, o_2)^2$ may occur inside the sum (instead of in a separate sum). Multidimensional scaling (MDS) [1090, 1879, 2056] is a method of constructing F that is based on minimizing stress. It has been widely used for many decades, in both the social and physical sciences, for purposes such as visualizing and clustering the data resulting from experiments. MDS is defined in many ways, some of which even allow nonmetric distances (i.e., the triangle inequality is not required).

Minimizing stress is essentially a nonlinear optimization problem, where the variables are the $N \cdot k$ coordinate values corresponding to the embedding (i.e., k coordinate values for each of the N objects). Typically, solving such a problem involves starting with an arbitrary assignment of the variables and then trying to improve on it in an iterative manner using the method of steepest descent (e.g., [1090]). The result of the optimization is not always the embedding that achieves the absolute minimum stress, but instead it is one that achieves a local minimum (i.e., the minimization can be pictured as finding the deepest valley in a landscape by always walking in a direction that leads downhill; the process can thus get stuck in a deep valley that is not necessarily the deepest).

In principle, it is possible to make the result of MDS satisfy the pruning property by constraining the minimization of the stress with $O(N^2)$ pruning property conditions, one for each pair of objects—that is, to minimize

$$\sum_{a,b \in S} (d'(F(a), F(b)) - d(a,b))^2 \Big/ \sum_{a,b \in S} d(a,b)^2$$

subject to the $O(N^2)$ conditions $d'(F(a), F(b)) - d(a,b) \leq 0$ (for all pairs $a, b \in S$). Note that simply minimizing the stress does not necessarily mean that the pruning property is satisfied. In particular, the stress could be minimized when the distance difference in the k-dimensional space (relative to the distance in the original space) increases by a large amount for one of the object pairs, whereas it decreases by just a small amount for all remaining pairs.

Unfortunately, MDS has limited applicability in similarity search, regardless of whether the resulting embedding satisfies the pruning property. This is partly due to the high cost of constructing the embedding, both in terms of the number of distance computations (i.e., $O(N^2)$, one for each pair of objects) and partly due to the inherent complexity of the optimization process. More seriously, when performing similarity queries, we must compute the embedding of the query object q, again, by minimizing stress subject only to varying the coordinate values of $F(q)$. Although the minimization process is itself expensive, the most serious drawback is that the distances of all objects in S from q must be computed in order to evaluate the stress—that is, $O(N)$ distance

computations are required. This completely defeats the goal of performing similarity queries in terms of the embedding space, namely, to avoid as many distance computations as possible. In fact, after computing the distances of all objects in S from q, we can immediately tell what the result of the query should be, thereby making the embedding of q unnecessary.

The Karhunen-Loève Transform (KLT), as well as the equivalent Principal Component Analysis (PCA) and the Singular Value Decomposition (SVD) methods discussed in Section 4.6.4.1, are the optimal methods of linearly transforming n-dimensional points to k-dimensional points ($k \leq n$). In particular, they minimize the *mean square error*—that is, the sum of the squares of the Euclidean distances between each n-dimensional point and its corresponding k-dimensional point (notice the similarity to minimizing the stress function in MDS described above). The goal of these methods is to find the most important features (more precisely, a linear combination of features) for a given set of feature vectors. The rationale behind them is that they can be interpreted as providing a new set of features ranked by their importance (i.e., their variation) so that if we ignore the less important (i.e., varying) ones by projecting onto the most important (i.e., varying) ones, then we preserve as much as possible of the variation between the original set of feature vectors as measured by the Euclidean distance between them.

These methods have several drawbacks. First, they cannot be applied when we are given only the distances between the objects (i.e., the distance function) and do not have the m-dimensional feature vectors. Second, they are very slow to compute, taking $O(N \cdot m^2)$ time, especially when there are many feature vectors (N), and the dimension m of the original space is very high.

Moreover, these methods are really meaningful only if d is the Euclidean distance metric and not one of the other Minkowski metrics. The reason is twofold. First, the other Minkowski metrics are not invariant under rotation. In other words, the distance between some pairs of points may increase or decrease, depending on the direction of the rotation. Second, variance is defined in terms of second powers, just like the Euclidean distance metric. Thus, the variance criteria that determine the rotation and, in turn, what axes to drop when reducing the dimensionality are inherently related to the Euclidean distance metric. In particular, the sum of the variances over all coordinate axes corresponds to the sum of the squared Euclidean distances from the origin to each point (recall that we are assuming that the mean along each dimension is at the origin). Therefore, dropping the axes having the least variance corresponds to reducing as little as possible the sum of the squared Euclidean distances from the origin to each point.

In contrast, as we will see, the embedding methods that we describe can be used with other distance metrics than the Euclidean distance metric. Moreover, they can be used with data from an arbitrary metric space (i.e., the only information about the data objects consists of the interobject distances). However, for some of the methods, such as FastMap, such usage is usually of limited interest as it does not necessarily satisfy the pruning property, and it may work only for very small values of k (even as small as just 1), instead of an arbitrary value of k when d is the Euclidean distance metric. However, the general class of Lipschitz embeddings does satisfy the pruning property, and, as we shall see, even the SparseMap method can be made to satisfy the pruning property.

Exercise

1. Devise an implementation of MDS where the variance in the difference of the distance values is minimized and the distance in the k-dimensional space satisfies the pruning property. If this is not possible, then give a counterexample.

4.7.2 Lipschitz Embeddings

A powerful class of embedding methods is known as *Lipschitz embeddings* [237, 962]. They are based on defining a coordinate space where each axis corresponds to a reference set that is a subset of the objects. This is the subject of this section, which is organized as follows. Section 4.7.2.1 defines a Lipschitz embedding. Section 4.7.2.2 describes how to select the reference sets. Section 4.7.2.3 presents an example of a Lipschitz embedding. Section 4.7.2.4 explains SparseMap, which is an instance of a Lipschitz embedding that attempts to reduce the computational cost of the embedding.

4.7.2.1 Definition

A Lipschitz embedding is defined in terms of a set R of subsets of S, $R = \{A_1, A_2, \ldots, A_k\}$.[61] The subsets A_i are termed the *reference sets* of the embedding. Let $d(o, A)$ be an extension of the distance function d to a subset $A \subset S$ such that $d(o, A) = \min_{x \in A}\{d(o, x)\}$. An embedding with respect to R is defined as a mapping F such that $F(o) = (d(o, A_1), d(o, A_2), \ldots, d(o, A_k))$. In other words, we are defining a coordinate space where each axis corresponds to a subset $A_i \subset S$ of the objects, and the coordinate values of object o are the distances from o to the closest element in each of A_i.

Notice that the distance-preserving L_∞ embedding that we describe in Section 4.7.1 is a special case of a Lipschitz embedding, where R consists of all singleton subsets of S (i.e., $R = \{\{o_1\}, \{o_2\}, \ldots, \{o_N\}\}$). Recall also the embedding of Faragó et al. [595], which made use of a subset of singleton subsets of S for a nearest neighbor search application (although they do not specify how to choose the objects that make up R). Another variation on a Lipschitz embedding is presented by Cowen and Priebe [426], where the number and sizes of the reference sets A_i are chosen based on an objective function that is meant to capture the quality of clustering that results. The distance-based indexing methods that make use of distance matrices described in Section 4.5.7 (e.g., see [130, 1292, 1743, 1930, 1951]) are also somewhat related to Lipschitz embeddings. These methods typically (e.g., see [1292]) make use of a matrix $D = (D_{ij})$ of distances, where $D_{ij} = d(o_i, p_j)$ and $T = \{p_1, p_2, \ldots, p_k\}$ is a set of k reference objects; for some methods (e.g., AESA [1930]), $T = S$ and $k = N$. The row vectors of D are equivalent to the result of a Lipschitz embedding using the singleton reference sets of objects in T. However, the search algorithms proposed for distance matrix methods do not explicitly treat the row vectors as if they represent points in geometric space (i.e., by using the Euclidean metric or some other Minkowski metric).

The intuition behind the Lipschitz embedding is that if x is an arbitrary object in the dataset S, some information about the distance between two arbitrary objects o_1 and o_2 is obtained by comparing $d(o_1, x)$ and $d(o_2, x)$ (i.e., the value $|d(o_1, x) - d(o_2, x)|$). This is especially true if one of the distances $d(o_1, x)$ or $d(o_2, x)$ is small. Observe that, due to the triangle inequality, we have $|d(o_1, x) - d(o_2, x)| \leq d(o_1, o_2)$, as illustrated in Figure 4.87. This argument can be extended to a subset A. In other words, the value $|d(o_1, A) - d(o_2, A)|$ is a lower bound on $d(o_1, o_2)$. This can be seen as follows. Let $x_1, x_2 \in A$ be such that $d(o_1, A) = d(o_1, x_1)$ and $d(o_2, A) = d(o_2, x_2)$. Since $d(o_1, x_1) \leq d(o_1, x_2)$ and $d(o_2, x_2) \leq d(o_2, x_1)$, we have $|d(o_1, A) - d(o_2, A)| = |d(o_1, x_1) - d(o_2, x_2)|$. Accounting for the fact that $d(o_1, x_1) - d(o_2, x_2)$ can be positive or negative, we have $|d(o_1, x_1) - d(o_2, x_2)| \leq \max\{|d(o_1, x_1) - d(o_2, x_1)|, |d(o_1, x_2) - d(o_2, x_2)|\}$. Finally, from the triangle inequality, we have $\max\{|d(o_1, x_2) - d(o_2, x_2)|, |d(o_1, x_1) - d(o_2, x_1)|\} \leq d(o_1, o_2)$. Thus, $|d(o_1, A) - d(o_2, A)|$ is a lower bound on $d(o_1, o_2)$. By using a set R of subsets, we increase the likelihood that the distance $d(o_1, o_2)$ between two objects o_1 and o_2 (as measured relative to other distances) is captured adequately by the distance in the embedding space between $F(o_1)$ and $F(o_2)$ (i.e., $d'(F(o_1), F(o_2))$).

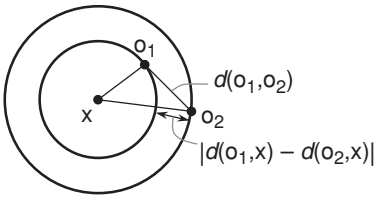

Figure 4.87
Demonstration of the distance bound $|d(o_1, x) - d(o_2, x)| \leq d(o_1, o_2)$. The objects o_1, o_2, and x are represented as points, and the distances between them by the lengths of the line segments between them.

[61] Note that the union of the set of elements comprising the subsets that make up R is not necessarily equivalent to S.

4.7.2.2 Selecting Reference Sets

With a suitable definition of R, the set of reference sets, we can establish bounds on the distance $d'(F(o_1), F(o_2))$ for all pairs of objects $o_1, o_2 \in S$, where d' is one of the L_p metrics. Such a definition was provided by Linial, London, and Rabinovich [1181], based in part on previous work by Bourgain [237]. In particular, in their definition [1181], R consists of $O(\log^2 N)$ randomly selected subsets of S, where each group of $O(\log N)$ subsets is of size 2^i, where $i = 1, \ldots, O(\log N)$. More concretely, the value $O(\log N)$ is typically approximately $\lfloor \log_2 N \rfloor$ (or perhaps $\lfloor \log_2 (N-1) \rfloor$). Thus, $R = \{A_1, A_2, \ldots, A_k\}$, where $k = \lfloor \log_2 N \rfloor^2$, and A_i is of size 2^j with $j = \lfloor (i-1)/(\log_2 N) + 1 \rfloor$. The embedding proposed by Linial et al. [1181] is a variant of the basic Lipschitz embedding, where each coordinate value is divided by a factor that depends on k. In particular, if d' is the L_p metric, F is defined such that $F(o) = (d(o, A_1)/q, d(o, A_2)/q, \ldots, d(o, A_k)/q)$, where $q = k^{1/p}$. Given this definition, Linial et al. [1181] prove that F satisfies

$$\frac{c}{\lfloor \log_2 N \rfloor} \cdot d(o_1, o_2) \leq d'(F(o_1), F(o_2)) \leq d(o_1, o_2) \tag{4.18}$$

for any pair of objects $o_1, o_2 \in S$, where $c > 0$ is a constant.[62] Thus, the distortion in distance values (i.e., the relative amount of deviation of the distance values in the embedding space with respect to the original distance values) is guaranteed to be $O(\log N)$ (with high probability). The proof for the bound $c/\lfloor \log_2 N \rfloor d(o_1, o_2) \leq d'(F(o_1), F(o_2))$ is rather sophisticated [237, 1181] and is beyond the scope of this discussion. However, the bound $d'(F(o_1), F(o_2)) \leq d(o_1, o_2)$ is easy to show. In particular, for each $A_i \in R$, we have $|d(o_1, A_i) - d(o_2, A_i)| \leq d(o_1, o_2)$, as shown in Section 4.7.2.1. Thus, when d' is an arbitrary L_p distance metric,

$$d'(F(o_1), F(o_2)) = \left(\sum_{i=1}^{k} \left(\frac{d(o_1, A_i) - d(o_2, A_i)}{k^{1/p}} \right)^p \right)^{1/p} \tag{4.19}$$

$$\leq \left(k \cdot \frac{d(o_1, o_2)^p}{k} \right)^{1/p} = d(o_1, o_2) \tag{4.20}$$

A distortion of $O(\log N)$ may seem rather large and may render the embedding ineffective at preserving relative distances. For example, if the range of distance values is less than the distortion, then the relative order of the neighbors of a given object may be completely scrambled. However, note that $O(\log N)$ is a worst-case (probabilistic) bound. In many cases, the actual behavior is much better. For example, in a computational biology application [886, 1180], the embedding defined above was found to lead to good preservation of clusters, as defined by biological functions of proteins.

Notice that the mapping F as defined by Linial et al. [1181] satisfies the pruning property (i.e., it is contractive), which is advantageous in similarity search. In many other situations, only relative differences in distance are important in the embedding space, while the satisfaction of the pruning property is immaterial. In other words, the crucial information that we wish to retain is which objects are close to each other and which are far (i.e., a weak form of proximity preservation). An example of an application of this sort is cluster analysis [886, 1180]. In such situations, it may be more convenient to use the regular Lipschitz embedding definition of F with respect to the set of reference sets R defined in [1181] (i.e., without dividing by $k^{1/p}$). Recall that $k = \lfloor \log_2 N \rfloor^2$, thereby implying that $\sqrt{k} = \lfloor \log_2 N \rfloor$. Therefore, when the Euclidean distance metric is being used, this embedding guarantees distance bounds of

$$c \cdot d(o_1, o_2) \leq d_E(F(o_1), F(o_2)) \leq \lfloor \log_2 N \rfloor \cdot d(o_1, o_2)$$

for any pair of objects $o_1, o_2 \in S$, where $c > 0$ is a constant (with high probability).

[62] More accurately, since the sets A_i are chosen at random, the proof is probabilistic, and c is a constant with high probability.

Object	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
o_1	0	2	13	7	3	8	11	4	9	10
o_2	2	0	11	9	3	10	9	2	11	8
o_3	13	11	0	6	10	9	4	9	6	3
o_4	7	9	6	0	6	3	8	9	2	5
o_5	3	3	10	6	0	7	8	3	8	7
o_6	8	10	9	3	7	0	9	10	3	6
o_7	11	9	4	8	8	9	0	7	10	3
o_8	4	2	9	9	3	10	7	0	11	6
o_9	9	11	6	2	8	3	10	11	0	7
o_{10}	10	8	3	5	7	6	3	6	7	0

Figure 4.88
Distance matrix for 10 sample objects.

Reference Set	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
A_1	2	0	9	9	3	10	7	0	11	6
A_2	0	2	10	6	0	7	8	3	8	7
A_3	4	2	3	2	3	0	3	0	0	0
A_4	0	2	4	0	3	3	0	0	2	3

Figure 4.89
Four-dimensional coordinate values for the 10 objects based on the distances in Figure 4.88, as determined by a Lipschitz embedding with the four reference sets $A_1 = \{o_2, o_8\}$, $A_2 = \{o_1, o_5\}$, $A_3 = \{o_6, o_8, o_9, o_{10}\}$, and $A_4 = \{o_1, o_4, o_7, o_8\}$.

Unfortunately, the embedding of [1181] described above is rather impractical for similarity searching for two reasons. First, due to the number and sizes of the subsets in R, there is a high probability that all N objects appear in some set in R. Thus, when computing the embedding $F(q)$ for a query object q (which generally is not in S), we will need to compute the distances between q and practically all objects in S, which is exactly what we wish to avoid. Second, the number of subsets in R, and thus the number of coordinate values (i.e., dimensions) in the embedding, is relatively large—that is, $\lfloor \log_2 N \rfloor^2$. Even with as few as 100 objects, the number of dimensions is 36, which is much too high to index on efficiently with multidimensional indexing methods. These drawbacks are acknowledged in [1181], but addressing them is left for future work (the only suggestion made is to drop the sets A_i of largest sizes).

4.7.2.3 Example of a Lipschitz Embedding

We now give an example of Lipschitz embedding. Figure 4.88 shows the interobject distances between 10 objects (these distance values were constructed by positioning 10 two-dimensional points and measuring the L_1 distance between them). In this case, $\lfloor \log_2 N \rfloor = 3$, so we can have three reference sets of three different sizes (2, 4, and 8) for a total of nine dimensions. Since a set of size 8 contains nearly all of the objects, we instead choose to use only reference sets of sizes 2 and 4, and two sets of each size. Choosing objects at random, we arrive at the sets $A_1 = \{o_2, o_8\}$, $A_2 = \{o_1, o_5\}$, $A_3 = \{o_6, o_8, o_9, o_{10}\}$, and $A_4 = \{o_1, o_4, o_7, o_8\}$. The resulting four-dimensional coordinates are given in Figure 4.89. Here, we use the regular Lipschitz embedding, where coordinate value j for object o_i is $d(o_i, A_j)$, rather than $d(o_i, A_j)/k^{1/p}$ as specified by Linial et al. [1181]. For example, $d(o_4, A_2) = \min\{d(o_4, o_1), d(o_4, o_5)\} = \min\{7, 6\} = 6$.

We now give an example of how to compute the distance between two objects in the embedding space, when d' is the Euclidean distance metric. In particular, for objects o_3 and o_8, we have from Figure 4.89 that $\mathsf{F}(\mathsf{o}_3) = (9,10,3,4)$ and $\mathsf{F}(\mathsf{o}_8) = (0,0,3,0)$. Therefore,

$$d'(\mathsf{F}(\mathsf{o}_3), \mathsf{F}(\mathsf{o}_8)) = \sqrt{(9-0)^2 + (10-3)^2 + (3-0)^2 + (4-0)^2}$$
$$= \sqrt{81+49+9+16} = \sqrt{155} \approx 12.4$$

In comparison, their actual distance is $d(\mathsf{o}_3, \mathsf{o}_8) = 9$. Notice that in this case the distance in the embedding space is greater than the actual distance. In contrast, when we use the embedding of Linial et al. [1181], the distance in the embedding space will be about $12.4/\sqrt{4} = 6.2$. Also, if d' is the Chessboard distance metric (L_∞), the distance in the embedding space will be $\max\{9,7,3,4\} = 9$, which happens to equal $d(\mathsf{o}_3, \mathsf{o}_8)$.

4.7.2.4 SparseMap

SparseMap [886] is an embedding method originally proposed for mapping a database of proteins into Euclidean space. It is built on the work of Linial et al. [1181] in that the same set of reference sets R is used. The SparseMap method [886] comprises two heuristics, each aimed at alleviating one of the drawbacks discussed in Section 4.7.2.2—that is, the potentially high cost of computing the embedding in terms of the number of distance computations that are needed and the large number of coordinate values. The first heuristic reduces the number of distance computations by calculating an upper bound $\hat{d}(o, A_i)$ instead of the exact value $d(o, A_i)$. The second heuristic reduces the number of dimensions by using a "high-quality" subset of R instead of the entire set, as defined in Section 4.7.2.2. Both heuristics have the potential to reduce the quality of the embedding in terms of the correspondence of distances in the original metric space and in the embedding space, but their goal [886] is to maintain the quality to the greatest extent possible. Note that the embedding used in SparseMap employs the regular Lipschitz embedding with respect to R, rather than the embedding proposed in [1181] (which divides the distances $d(o, A_i)$ by $k^{1/p}$), and uses the Euclidean distance metric.

In SparseMap, the coordinate values of the vectors are computed one by one. In other words, if $R = \{A_1, A_2, \ldots, A_k\}$ is the sequence of reference sets in order of size, we first compute $d(o, A_1)$ for all objects $o \in S$, next $d(o, A_2)$ for all objects o, and so on. Since evaluating $d(o, A_i)$ for any given object o can be very expensive in terms of distance computations, SparseMap adopts a heuristic that instead computes $\hat{d}(o, A_i)$, which is an upper bound on $d(o, A_i)$. This heuristic exploits the partial vector that has already been computed for each object, and only calculates a fixed number of distance values for each object (as opposed to $|A_i|$ distance values). In particular, for each object $x \in A_i$, it computes $d_E(F_{i-1}(o), F_{i-1}(x))$, where F_{i-1} is the embedding based on $A_1, \ldots, A_{i-1}$. On the basis of this approximate distance value, a fixed number l of objects in A_i having the smallest approximate distance value from o is picked, and the actual distance value $d(o,x)$ for each such object x is computed. The smallest distance value among those serves as the upper-bound distance value $\hat{d}(o, A_i)$, which becomes the ith coordinate value of the vector corresponding to o in the embedding.

The second heuristic involved in SparseMap reduces the dimensionality of the result and is termed *greedy resampling* in [886]. Greedy resampling is applied after the k coordinate axes have all been determined, and its goal is to reduce the number of coordinate axes to $k' < k$. Essentially, this means eliminating some of the reference sets A_i. A natural question is whether we cannot eliminate a poor reference set A_i before computing all the approximate distances $\hat{d}(o, A_i)$. However, the problem is that we cannot know whether a set A_i is good before evaluating $\hat{d}(o, A_i)$ (or $d(o, A_i)$) for each object o. The basic idea of greedy resampling is to start with a single "good" coordinate axis and then incrementally to add coordinate axes that maintain "goodness." In particular, initially, the coordinate axis whose sole use leads to the least stress [1090]

is determined (this is somewhat analogous in spirit to basing the first coordinate axis on a pair of objects that are far apart in the FastMap method as described in Section 4.7.3). Unfortunately, calculating the stress requires computing distances for all pairs of objects, which is prohibitively expensive. Instead, the heuristic computes the stress based on some fixed number of object pairs (e.g., 4,000 in experiments in [886], which constituted 10% of the total number of pairs). Next, the coordinate axis that leads to the least stress when used in conjunction with the first axis is determined. This procedure is continued until the desired number of coordinate axes has been obtained.

A drawback of the embedding that forms the basis of SparseMap (i.e., the regular Lipschitz embedding on the reference sets, without taking the heuristics into account) is that it does not satisfy the pruning property (i.e., it is not contractive). In particular, the distance value in the embedding may be as much as a factor of $\log_2 N$ larger than the actual distance value. Two methods can be applied to obtain an embedding that satisfies the pruning property. First, the embedding proposed in [1181] (where the coordinate values are divided by $k^{1/p}$) can be employed, which does satisfy the pruning property. Second, the distance function d' can be modified to yield the same effect. In particular, if $d_p(F(o_1), F(o_2))$ is one of the Minkowski metrics, we can define $d'(F(o_1), F(o_1)) = d_p(F(o_1), F(o_2))/(k^{1/p})$. The advantage of modifying the distance function d' rather than the embedding itself is that it allows modifying the number of coordinate axes (for example, during the construction of the embedding and in the second SparseMap heuristic) without changing existing coordinate values. With either method, the embedding will satisfy Equation 4.18 for any distance metric L_p (i.e., subject to modification when using the second method).

Unfortunately, the heuristics applied in SparseMap do not allow deriving any practical bounds (in particular, bounds that rely on N, or k, or both) on the distortion resulting from the embedding. In particular, the first heuristic can lead to larger distances in the embedding space, thus possibly causing the pruning property to be violated (in contrast, the second heuristic can reduce only distances in the embedding space). This is because the value of $|\hat{d}(o_1, A_i) - \hat{d}(o_2, A_i)|$ may not necessarily be a lower bound on $d(o_1, o_2)$. To see why, note that the upper bound distances $\hat{d}(o_1, A_i)$ and $\hat{d}(o_2, A_i)$ can be larger than the actual distances $d(o_1, A_i)$ and $d(o_2, A_i)$ (which are the minimum distances from o_1 and o_2 to an object in A_i) by an arbitrary amount. In particular, we cannot rule out a situation where $\hat{d}(o_1, A_i) > \hat{d}(o_2, A_i) + d(o_1, o_2)$, in which case $|\hat{d}(o_1, A_i) - \hat{d}(o_2, A_i)| > d(o_1, o_2)$.

Thus, we see that in order to satisfy the pruning property, we must use the actual values $d(o, A_i)$ in the embedding, rather than an upper bound on these distances, as is done in SparseMap. Fortunately, there is a way to modify the first heuristic of SparseMap so that it computes the actual value $d(o, A_i)$ while still (at least potentially) reducing the number of distance computations [850, 855]. We illustrate this for the case when the Chessboard distance metric, d_M, is used as d' in the embedding space. Note that $d_M(F(o_1), F(o_2)) \leq d(o_1, o_2)$, as shown in Section 4.7.2.2 (the key observation is that $|d(o_1, A_i) - d(o_2, A_i)| \leq d(o_1, o_2)$ for all A_i). Furthermore, if F_i is the partial embedding for the first i coordinate values, we also have $d_M(F_i(o_1), F_i(o_2)) \leq d(o_1, o_2)$. In this modified heuristic for computing $d(o, A_i)$, instead of computing the actual distance value $d(o, x)$ for only a fixed number of objects $x \in A_i$, we must do so for a variable number of objects in A_i. In particular, we first compute the approximate distances $d_M(F_{i-1}(o), F_{i-1}(x))$ for all objects $x \in A_i$, which are lower bounds on the actual distance value $d(o, x)$. Observe that in SparseMap the approximate distances $d_E(F_{i-1}(o), F_{i-1}(x))$ are computed for each $x \in A_i$, which has the same cost complexity as evaluating $d_M(F_{i-1}(o), F_{i-1}(x))$, although the constants of proportionality are lower for d_M than for d_E. Next, we compute the actual distances of the objects $x \in A_i$ in increasing order of their lower-bound distances, $d_M(F_{i-1}(o), F_{i-1}(x))$. Let $y \in A_i$ be the object whose actual distance value $d(o, y)$ is the smallest distance value so far computed following this procedure. Once all lower-bound distances $d_M(F_{i-1}(o), F_{i-1}(x))$ of the remaining elements $x \in A_i$ are greater than $d(o, y)$, we are assured that $d(o, A_i) = d(o, y)$.

Even though we described our modified heuristic in terms of the Chessboard distance metric, by using a suitable definition of the distance function d', the heuristic can be applied to any Minkowski metric L_p. In particular, if k' is the current number of coordinate axes (at the completion of the process, $k' = k$), the distance function d' based on L_p is defined as

$$d'(F_{k'}(o_1), F_{k'}(o_2)) = \frac{\left(\sum_i |d(o_1, A_i) - d(o_2, A_i)|^p\right)^{1/p}}{(k')^{1/p}} \tag{4.21}$$

For any choice of p, this distance metric makes F satisfy the pruning property; note the similarity with Equation 4.20.

Moreover, observe that for fixed values of o_1, o_2, and $F_{k'}$, the function d' defined by Equation 4.21 increases with increasing values of p. For example, for $p = 1$, $d'(F_{k'}(o_1), F_{k'}(o_2))$ is the average among the coordinate value differences. On the other hand, for $p = \infty$, it is the maximum difference. Thus, the use of the Chessboard metric L_∞ would lead to the largest values of $d'(F_{k'}(o_1), F_{k'}(o_2))$ for any given choice of the sets A_i. For similarity queries, as well as for the modified heuristic described above, given a fixed set of reference sets A_i, this would therefore lead to the best possible pruning during search. To see why this is the case, suppose that we are performing a range query with query object q and query radius r, and we wish to report all objects o such that $d(q,o) \leq r$. Let o' be an object that is too far from q (i.e., $d(q,o') > r$). However, if $d'(F(q), F(o')) \leq r$, o' will be a part of the result set when performing a query in the embedding space. Thus, the situation can easily arise that $d'(F(q), F(o')) \leq r$ when basing d' on the City Block or Euclidean distance metrics (i.e., L_1 or L_2), but $d'(F(q), F(o')) > r$ when d' is based on the Chessboard distance metric L_∞. Such a hypothetical example is illustrated in Figure 4.90 with distance values r_1, r_2, and r_∞ corresponding to the use of L_1, L_2, and L_infty distance metrics, respectively.

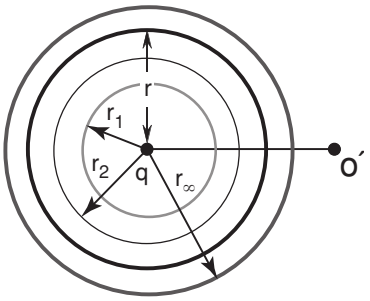

Figure 4.90
A hypothetical range query example where an object o′ is outside the distance range r from the query object q. The distance d'(F(q), F(o′)) from q to o′ in the embedding space will lie somewhere on the line between q and o′. Thus, this distance may lie inside the query range if d' is based on L_1 or L_2 but outside it if d' is based on L_∞.

Although the modified heuristic presented above will likely lead to a higher number of distance computations than the SparseMap heuristic, the higher cost of the embedding (which mainly affects preprocessing) may be justified, as the resulting embedding satisfies the pruning property. This allows effective pruning in similarity queries, while obtaining accurate results, as we get 100% recall and thus do not miss any relevant answers.

One way to measure the quality of embeddings such as those produced by SparseMap is to use the concept of a cluster preservation ratio (CPR) [886]. It can be applied to a dataset when a known clustering exists for the objects. In particular, for each object o whose cluster is of size s, we find the s nearest neighbors in the embedding space and compute the fraction of cluster objects that are among these s neighbors. The CPR indicates the average ratio of cluster preservation over all objects in the dataset.

In order to study the validity of the SparseMap method, the results of various experiments are described in [886] in which the datasets consist of proteins (or more accurately, the amino acid sequences that make up each protein). This data is useful because these proteins have been studied extensively in terms of their biochemical functions, so proteins having similar functions can be grouped together. Therefore, we can test whether amino acid sequences representing these known proteins follow this grouping when an embedding is performed. The focus of the experiments is mainly on comparing the performance of SparseMap with that of FastMap [587], another embedding method proposed for similarity searching and described in greater detail in Section 4.7.3. Both methods are based on heuristics where some parameter controls the number of distance computations that are performed. In SparseMap, this is the number of actual distance computations performed in evaluating $\hat{d}(o, A_i)$, while in FastMap, it is the number of iterations performed in the pivot-finding process (see Section 4.7.3.2). Thus, the two methods can be made to perform approximately the same number of distance computations when obtaining a given number of coordinate axes. Using this technique, the embedding produced by SparseMap was of a higher quality than that produced by

FastMap in terms of stress and how well clusters are retained by the embedding using CPR [886], especially for a small number of coordinate axes. In addition, SparseMap was found to scale up better than FastMap, in terms of the time to perform the mapping, as the pattern in which the database is accessed leads to fewer disk I/Os.

4.7.3 FastMap

In this section, we explain how the FastMap method works in some detail. Recall from the introduction to Section 4.7, that FastMap was inspired by dimension reduction methods for Euclidean space that are based on linear transformations such as KLT (and the equivalent PCA and SVD) while also being applicable to arbitrary metric spaces. As we will see, FastMap is most useful for query processing when both d and d' are the Euclidean distance metric (i.e., the original objects are points in a multidimensional space) since these are the only cases in which satisfaction of the pruning property is guaranteed. This is primarily a result of the fact that many of the derivations used in the development of FastMap make an implicit assumption that d is the Euclidean distance metric in a vector space of varying dimensionality. Nevertheless, FastMap can be applied, with varying success, with other distance metrics as well as with data from an arbitrary metric space (i.e., the only information about the data objects consists of the interobject distances). However, in such cases, often some key aspects such as the pruning property will not necessarily hold; nor will we always be able to obtain as many as k coordinate axes. Similarly, due to the nature of the FastMap method, the best result is obtained when d', the distance function in the embedding space, is the Euclidean distance metric. Thus, unless otherwise stated, we assume d' to be the Euclidean distance metric.

This section is organized as follows. Section 4.7.3.1 gives an overview of the basic mechanics of the FastMap method. Section 4.7.3.2 describes how to obtain the pivot objects. Section 4.7.3.3 shows how to derive the first coordinate value. Section 4.7.3.4 indicates how to calculate the projected distance. Section 4.7.3.5 shows how to derive the remaining coordinate values. Section 4.7.3.6 discusses in great detail when the pruning property is satisfied. Section 4.7.3.7 points out the extent of the deviation of the distance in the embedding space from that in the original space (termed *expansion*). Section 4.7.3.8 discusses a heuristic that has been proposed to deal with the case that the projected distance is complex valued. Section 4.7.3.9 summarizes the properties of FastMap and the Lipschitz embeddings described in Section 4.7.2, while also briefly comparing FastMap with some other embeddings such as MetricMap [1950].

4.7.3.1 Mechanics of FastMap

The FastMap method works by imagining that the objects are points in a hypothetical high-dimensional Euclidean space of unknown dimension—that is, a vector space with the Euclidean distance metric. However, the various implications of this Euclidean space assumption are not explored by Faloutsos and Lin [587]. In the sequel, the terminology reflects the assumption that the objects are points (e.g., a line can be defined by two objects). The coordinate values corresponding to these points are obtained by projecting them onto k mutually orthogonal directions, thereby forming the coordinate axes of the space in which the points are embedded. The projections are computed using the given distance function d. The coordinate axes are constructed one by one, where at each iteration two objects (termed *pivot objects*) are chosen, a line is drawn between them that serves as the coordinate axis, and the coordinate value along this axis for each object o is determined by mapping (i.e., projecting) o onto this line.

Assume, without loss of generality, that the objects are actually points (this makes it easier to draw the examples that we use in our explanation) and that they lie in an m-dimensional space. We prepare for the next iteration by determining the $(m-1)$-dimensional hyperplane H perpendicular to the line that forms the previous coordinate axis and projecting all of the objects onto H. The projection is performed by defining

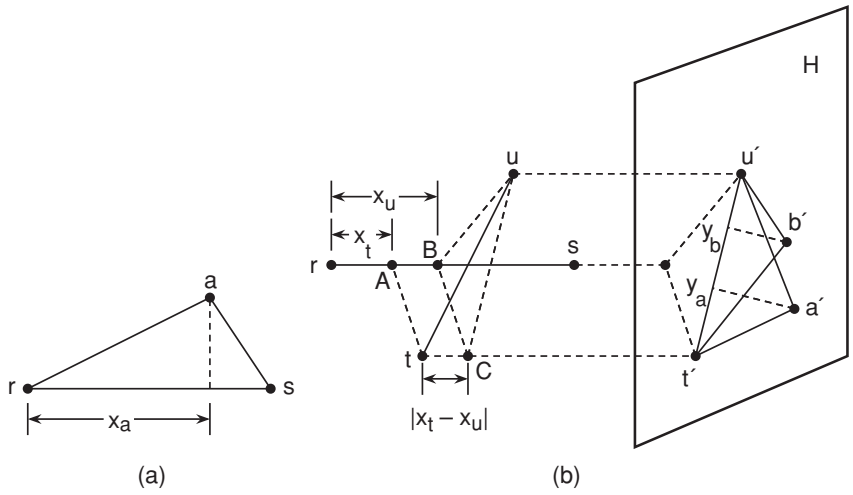

Figure 4.91
Examples of projections of objects with the FastMap method on (a) the first coordinate axis and (b) the second coordinate axis.

a new distance function d_H that measures the distance between the projections of the objects on H. In particular, we will see that d_H is derived from the original distance function d and the coordinate axes determined so far. At this point, the problem has been replaced by a recursive variant of the original problem with m and k reduced by one, and a new distance function d_H. This process is continued until the necessary number of coordinate axes has been determined.

Figure 4.91 illustrates how the first coordinate axis is determined, how the objects are mapped onto the hyperplane H, and how the projected objects are used to determine the second coordinate axis. Again, we assume, without loss of generality, that the objects are actually points. Figure 4.91(a) shows the result of the projection that yields the first coordinate axis where the pivot objects are r and s. In particular, the first coordinate value x_a for object a (i.e., the first coordinate value in the vector F(a)) is the distance from r to the projection of a onto the line through r and s. We postpone for now the discussion of Figure 4.91(b), which illustrates how d_H and the next set of coordinate values are determined.

4.7.3.2 Choosing Pivot Objects

As we saw, the pivot objects that are chosen at each step serve to anchor the line that forms the newly formed coordinate axis. Ideally, there should be a large spread of the projected values on the line between the pivot objects, where *spread* is defined as $\max_{a,b \in S} |x_a - x_b|$, and a and b are arbitrary objects. The reason is that a greater spread generally means that more distance information can be extracted from the projected values—that is, for any pair of objects a and b, it is more likely that $|x_a - x_b|$ is large, thereby providing more information. This principle is similar to that used in the KLT (as well as the equivalent SVD and PCA) method, described in Section 4.6.4.1. The difference here is that spread along an axis is a weaker notion than variance, which is used by KLT (as well as the equivalent SVD and PCA). The reason why spread is a weaker notion is that a large spread may be caused by a few outliers, while most of the other values may be clustered within a small range. On the other hand, a large variance means that the values are genuinely scattered over a large range. Nevertheless, spread usually provides a reasonably good estimate of the variance. In order to maximize the likelihood of obtaining a large spread, the pivot objects should be as far as possible from each other. Unfortunately, determining the farthest pair of objects in a given set of N

objects is computationally expensive. In particular, it takes $O(N^2)$ distance computations as we need to examine the distance between each pair of objects, although we need not sort them (which is actually not very costly in comparison to the cost of the distance computation) as we want only the maximum value.

Faloutsos and Lin [587] propose a heuristic for computing an approximation of the farthest pair of objects. This heuristic first arbitrarily chooses one of the objects a. Next, it finds the object r that is farthest from a. Finally, it finds the object s that is farthest from r. The last step can be iterated a number of times (e.g., 5 [587]) in order to obtain a better estimate of the pair that is farthest apart. In fact, it can be shown that for a given set of N objects, the procedure for finding the farthest pair of objects can be iterated a maximum of $N-1$ steps (see Exercise 1) for a total of $O(N^2)$ distance computations. The process of finding the farthest pair of points in Euclidean space is also known as the *diameter* of the point set (see [1535] for some background on the problem and [799] for a recent practical approximate algorithm). The above heuristic process of finding the pivot objects requires $O(N)$ distance computations as long as the number of iterations is fixed. Unfortunately, the $O(N)$ cost bound for the heuristic may not always hold (see Exercise 2 in Section 4.7.3.7). Note that the original distance function d is used only when determining the first coordinate axis. However, the modified distance functions (resulting from successive projections on hyperplanes) used for subsequent coordinate axes are based on d, and thus an evaluation of d is also required for any distance computations in later steps.

Choosing pivot objects r and s in this way guarantees that for any objects a and b, we have $d(a,b) \leq 2d(r,s)$, or $d(r,s) \geq \frac{1}{2}d(a,b)$ (see Exercise 3). In other words, $d(r,s)$ is at least half of the distance between the most distant pair of objects. This follows directly from the triangle inequality and the assumption that s is the object farthest from r. However, this bound is guaranteed to hold only for the first pair of pivot objects, as shown in Section 4.7.3.6, since it is possible that the distance functions used to determine the second and subsequent coordinate values do not satisfy the triangle inequality. Notice that a tighter lower bound for $d(r,s)$ (i.e., larger than one-half of the distance between the objects that are the farthest apart) cannot be guaranteed for any fixed number of iterations in the pivot-choosing heuristic since $N-1$ iterations may be needed to find pivots whose distance is any greater than this lower bound. As an example of such a situation, suppose that one pair of objects has distance 2 while all other pairs have distance 1. Determining the pair having distance 2, which is the only one greater than 1, requires $O(N^2)$ distance computations, on the average, instead of the $O(N)$ used by the heuristic, since the number of distinct pairs is $O(N^2)$.

Exercises

1. Prove that the procedure for finding the farthest pair of objects to serve as pivots in the FastMap algorithm can be iterated a maximum of $N-1$ steps.
2. Can you prove that the maximum number of iterations in the algorithm for finding the pivot objects in Exercise 1 can indeed by attained? If so, construct an example where this happens.
3. Show that choosing the pivot objects r and s by using the heuristic described in the text guarantees that $d(r,s)$ is at least half the distance between the most distant pair of objects.

4.7.3.3 Deriving the First Coordinate Value

In order to understand better how and why the FastMap mapping process works, let us examine its mechanics in greater detail as we compute the first coordinate value. Initially, we project the objects onto a line between the pivot objects, say r and s, as shown in Figure 4.92 for an object a. Note that the projection of object a may actually lie beyond the line segment between r and s, as shown in Figure 4.92(b). This does not pose problems but may cause x_a to be negative. The actual value of x_a is obtained by

solving the following equation for x_a:

$$d(r,a)^2 - x_a^2 = d(s,a)^2 - (d(r,s) - x_a)^2 \tag{4.22}$$

Expanding terms in Equation 4.22 and rearranging yields

$$x_a = \frac{d(r,a)^2 + d(r,s)^2 - d(s,a)^2}{2d(r,s)} \tag{4.23}$$

Observe that Equation 4.22 is obtained by applying the Pythagorean theorem to each half of the triangle in Figure 4.92(a) (a similar interpretation applies to the case in Figure 4.92(b)). Since the Pythagorean theorem is specific to Euclidean space, here we have an instance where Faloutsos and Lin [587], in their development of the method, make the implicit assumption that d is the Euclidean distance metric. Thus, the equation is only a heuristic when used for general metric spaces. The implications of the use of this heuristic are explored in Section 4.7.3.6. In particular, we find that in this case the embedding produced by FastMap might not satisfy the pruning property, and this may cause the mapping process to terminate prematurely.

A number of observations can be made about x_a, based on Equation 4.23 and the selection of pivot objects. First, $x_r = 0$ and $x_s = d(r,s)$, as would be expected. Second, note that $|x_a| \leq d(r,s)$ (see Exercise 1). This implies that the maximum difference between two values x_a and x_b (i.e., the spread for the first coordinate axis as defined earlier) is $2d(r,s)$, which is equal to the maximum possible distance between any pair of objects, as shown in Section 4.7.3.2. In fact, it can be shown that the spread is never larger than the distance between the farthest pair of objects (see Exercise 2). Since the spread is at least $d(r,s)$ when r and s serve as the pivot objects (as $x_r = 0$ and $x_s = d(r,s)$), this implies that the spread obtained from pivots r and s is at least half of the maximum obtainable spread, which is $2d(r,s)$. Unfortunately, as we show in Section 4.7.3.6, it is possible that the distance functions used in subsequent iterations of FastMap do not satisfy the triangle inequality if d is not the Euclidean distance metric. Thus, the above bounds may not hold when determining the subsequent coordinate values. Note that Equation 4.23 is also used in BoostMap as one of two possible weakly proximity preserving mappings (see the preamble to Section 4.7).

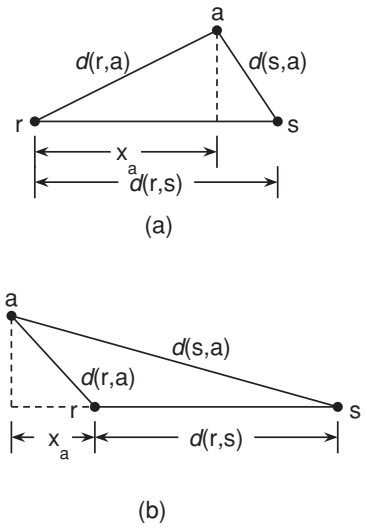

Figure 4.92
Examples of two possible positions for the projection of an object on the line joining the points corresponding to the pivot objects.

Exercises

1. Prove that $|x_a| \leq d(r,s)$ for any distance metric d, where x_a is the first coordinate value of a as defined in Equation 4.23 for pivot objects r and s (see Figure 4.92).
2. Let t and u be the two objects that are farthest apart of all the objects in S. Prove that for any distance metric d the spread (i.e., $\max_{a,b} |x_a - x_b|$ for any objects a and b) is never more than $d(t,u)$ for any choice of pivot objects.

4.7.3.4 Projected Distance

Before we can determine the second coordinate value for each object, we must derive d_H, the distance function for the distances between objects when projected onto the hyperplane H, as mentioned in Section 4.7.3.1. Figure 4.91(b) illustrates how the objects are projected onto the hyperplane H, and how the projected objects are used to determine the second coordinate axis. For expository purposes, assume that the underlying space is three-dimensional. In this case, points A and B are the projections of objects t and u, respectively, on the first coordinate axis (formed by the line joining the pivot objects r and s) with a separation of $|x_t - x_u|$. Points t′ and u′ are the projections of objects t and u, respectively, on the plane H that is perpendicular to the line between r and s that forms the first coordinate axis. Point C is the projection of u onto the line through t and t′ parallel to the line through r and s. Thus, the distance between t′ and u′ equals the distance between C and u. The latter can be determined by applying the Pythagorean theorem since the

angle at C in the triangle tuC is 90°. Therefore, we have

$$d(t,u)^2 = d(t,C)^2 + d(C,u)^2 = (x_t - x_u)^2 + d(t',u')^2 \tag{4.24}$$

Thus, defining $d_H(t,u) = d(t',u')$ and changing the order of the terms, we obtain

$$d_H(t,u)^2 = d(t,u)^2 - (x_t - x_u)^2 \tag{4.25}$$

Note that Equation 4.25 applies to any pair of objects t and u and not just to the ones that serve as pivots in the next iteration, as is the case in Figure 4.91(b). Also, note that Faloutsos and Lin [587] use the notation $d_H(t',u')$ rather than $d_H(t,u)$.

Observe that this is another occasion where Faloutsos and Lin [587], in their development of the method, make the implicit assumption that d is the Euclidean distance metric (or that (S,d) is isometric to a subset of some Euclidean space). This assumption has some undesirable side effects. For example, as we show in Section 4.7.3.6, if d is not a Euclidean distance metric, then it is possible for d_H to fail to satisfy the triangle inequality, which in turn may cause Equation 4.23 to produce coordinate values that violate the pruning property. Furthermore, violation of the pruning property in earlier iterations of FastMap may cause negative values of $d_H(a,b)^2$. This complicates the search for pivot objects as the square root of a negative value is a complex number, which in this case means that a and b (or, more precisely, their projections) cannot serve as pivot objects.

4.7.3.5 Subsequent Iterations

Each time we recursively invoke the FastMap coordinate determination method, we must determine the distance function d_H for the current set of projections in terms of the current distance function (i.e., the one that was created in the previous recursive invocation). Thus, the original distance function d is used only when obtaining the first coordinate axis. In subsequent iterations, d is the distance function d_H from the previous iteration. At this point, it is instructive to generalize Equations 4.23 and 4.25 to yield a recursive definition of the distance functions and the resulting coordinate values for each object. Before we do so, we must define a number of symbols, each representing the ith iteration of FastMap. In particular, x_o^i is the ith coordinate value obtained for object o, $F_i(o) = \{x_o^1, x_o^2, \ldots, x_o^i\}$ denotes the first i coordinate values of $F(o)$, d_i is the distance function used in the ith iteration, and p_1^i and p_2^i denote the two pivot objects chosen in iteration i (with the understanding that p_2^i is the farthest object from p_1^i). Now, the general form of Equation 4.23 for iteration i is

$$x_o^i = \frac{d_i(p_1^i,o)^2 + d_i(p_1^i,p_2^i)^2 - d_i(p_2^i,o)^2}{2d_i(p_1^i,p_2^i)} \tag{4.26}$$

given the recursive distance function definition

$$\begin{aligned} d_1(a,b) &= d(a,b) \\ d_i(a,b)^2 &= d_{i-1}(a,b)^2 - (x_a^{i-1} - x_b^{i-1})^2 \\ &= d(a,b)^2 - d_E(F_{i-1}(a), F_{i-1}(b))^2 \end{aligned} \tag{4.27}$$

As an example consider again Figure 4.88, which shows the interobject distances between 10 objects that were constructed by positioning 10 two-dimensional points and measuring the L_1 distance between them. We now show how FastMap obtains two-dimensional coordinate values for the objects. Seeing that the largest distance in the table is that between o_1 and o_3, we choose these objects as pivots. The result is shown as the first dimension in Figure 4.93 (i.e., in the first row), where the values are given to a precision of one fractional digit. In order to see how these values are determined, we derive the first coordinate value of o_5:

Coordinate	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
First	0.0	2.0	13.0	7.0	3.0	5.8	10.5	4.0	8.2	10.0
Second	4.4	1.0	4.4	8.5	3.7	9.7	0.7	0.0	10.2	2.8

Figure 4.93
Two-dimensional coordinate values for the 10 objects based on the distances in Figure 4.88, as determined by FastMap using o_1 and o_3 as pivot objects for the first coordinate axis and o_8 and o_9 for the second coordinate axis.

Object	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
o_1	0.0	0.0	0.0	0.0	0.0	29.8	9.9	0.0	13.3	0.0
o_2	0.0	0.0	0.0	56.0	8.0	85.2	8.1	0.0	82.2	0.0
o_3	0.0	0.0	0.0	0.0	0.0	29.8	9.9	0.0	13.3	0.0
o_4	0.0	56.0	0.0	0.0	20.0	7.7	51.5	72.0	2.5	16.0
o_5	0.0	8.0	0.0	20.0	0.0	40.9	7.2	8.0	36.6	0.0
o_6	29.8	85.2	29.8	7.7	40.9	0.0	59.0	96.6	3.3	18.7
o_7	9.9	8.1	9.9	51.5	7.2	59.0	0.0	6.2	94.7	8.7
o_8	0.0	0.0	0.0	72.0	8.0	96.6	6.2	0.0	103.1	0.0
o_9	13.3	82.2	13.3	2.5	36.6	3.3	94.7	103.1	0.0	45.9
o_{10}	0.0	0.0	0.0	16.0	0.0	18.7	8.7	0.0	45.9	0.0

Figure 4.94
Distances of the 10 sample objects as determined by the first projected distance function, d_H. The values in the table are actually the squared distances, $d_H(o_i, o_j)^2$.

$$\frac{d(o_1,o_5)^2 + d(o_1,o_3)^2 - d(o_3,o_5)^2}{2d o_1,o_3)} = \frac{3^2 + 13^2 - 10^2}{2 \cdot 13} = 78/26 = 3$$

Figure 4.94 shows the (squared) projected distances $d_H(o_i, o_j)^2$ for the 10 objects obtained by projecting them on the hyperplane H perpendicular to the line through the pivot objects o_1 and o_3. Again, the distance values are given to a precision of only one fractional digit. As an example of how these distance values are computed, we derive $d_H(o_5, o_6)^2$ with the help of Figure 4.93:

$$d_H(o_5,o_6)^2 = d(o_5,o_6)^2 - |x_5 - x_6|^2 \approx 7^2 - |3 - 5.8|^2 = 49 - 2.8^2 \approx 41.2$$

This value does not exactly match the value $d_H(o_5, o_6)^2 \approx 40.9$ found in Figure 4.94 due to roundoff error.

The largest distance value is $d_H(o_8, o_9)^2 \approx 103.1$, so the objects o_8 and o_9 (or, more precisely, their projected versions) get chosen as the second pair of pivot objects used to determine the values along the second dimension, as given in Figure 4.93. Again, let us show how the coordinate value for o_5 is determined, this time along the second coordinate axis:

$$\frac{d_H(o_8,o_5)^2 + d_H(o_8,o_9)^2 - d_H(o_9,o_5)^2}{2d_H(o_8,o_9)} \approx \frac{8 + 103.1 - 36.6}{2\sqrt{103.1}} \approx 74.5/20.3 \approx 3.7$$

The process of mapping the N objects to points in a k-dimensional space makes $O(k \cdot N)$ distance computations as there are $O(N)$ distance calculations at each of k iterations. It requires $O(k \cdot N)$ space to record the k coordinate values of each of the points corresponding to the N objects. It also requires a $2 \times k$ array to record the identities of the k pairs of pivot objects as this information is needed to process queries. Note that query objects are transformed to k-dimensional points by applying the same algorithm that was used to construct the points corresponding to the original objects, except that we use the

existing pivot objects. In other words, given query object q, we obtain its k-dimensional coordinate values by projecting q onto the lines formed by the corresponding pivot objects using the appropriate distance function. This process is facilitated by recording the distance between the points corresponding to the pivot objects so that it need not be recomputed for each query, although this can be done if we do not want to store these distance values. The entire process of obtaining the k-dimensional point corresponding to the query object takes $O(k)$ distance computations (which is actually $O(1)$ if we assume that k is a constant), in contrast to the size of the database, which is $O(N)$.

Exercises

1. Show that the total number of distance computations performed by FastMap is $kP(N-1)$, assuming P iterations in the pivot-finding heuristic.
2. Prove that at least $k+1$ of the pivot objects in the process of constructing a point in k-dimensional space in the FastMap method are different, assuming that it is indeed possible to apply k iterations of FastMap.
3. Given N objects and a distance function that indicates the interobject distances, implement the FastMap algorithm to find the corresponding set of N points in a k-dimensional space.
4. Given N objects, a distance function that indicates the interobject distances query object q, and a set of N points in a k-dimensional space that have been obtained via the use of the FastMap algorithm, give an algorithm to find the point in the k-dimensional space corresponding to q.

4.7.3.6 Pruning Property

The embedding F produced by FastMap satisfies the pruning property when (S,d) is a subset of a Euclidean space (or isometric to such a subset). This is not surprising since key aspects of FastMap are based on a property unique to Euclidean spaces, namely, the Pythagorean theorem. In particular, both Equation 4.23, which computes a single coordinate value, and Equation 4.25, which computes the projected distance d_H used in the next iteration, are based on the Pythagorean theorem. Equivalently, FastMap can be seen to be based on the Euclidean distance metric's property of being invariant to rotation and translation. Below, we present an intuitive argument for the satisfaction of the pruning property by FastMap and show that total distance preservation can be achieved.

For (S,d_E), where $S \subset \mathbb{R}^m$ and d_E is the Euclidean distance metric, an iteration of FastMap essentially amounts to performing a rotation and a translation, followed by extracting the value of a coordinate axis. This is depicted in Figure 4.95 for pivot objects r and s. Notice that x_a for object $\mathsf{a} \in S$ is simply the coordinate value along the new axis x' since the Euclidean distance metric is invariant to rotation and translation. Furthermore, $d_H(\mathsf{a},\mathsf{b})$ as defined by Equation 4.25 can be shown to be equivalent to the Euclidean distance between the projections of a and b onto the hyperplane H. In general, we can show that $d_i(a,b) = d_E(a_i,b_i)$ for $a,b \in S$, where d_i is defined by Equation 4.27, and a_i and b_i are the projections of objects a and b, respectively, onto the intersection of the $i-1$ hyperplanes obtained in iterations 1 through $i-1$. Thus, we can obtain coordinate axis i by rotating and translating along the $(m-i+1)$-dimensional hyperplane formed by the intersection of these $i-1$ hyperplanes. After k iterations, Equation 4.27 yields $d_{k+1}(a,b)^2 = d_E(a_{k+1},b_{k+1})^2 = d_E(a,b)^2 - d'_E(F(a),F(b))^2$, where d'_E is the Euclidean distance metric for $\mathbb{R}^k$. Thus, since $d_E(a_{k+1},b_{k+1})^2 \geq 0$, we have $d_E(a,b)^2 - d'_E(F(a),F(b))^2 \geq 0$, which directly implies $d_E(a,b) \geq d'_E(F(a),F(b))$. In other words, F satisfies the pruning property.

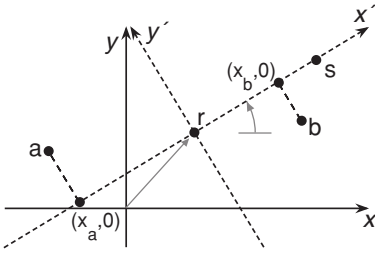

Figure 4.95
An example of how to rotate and translate a coordinate system such that the first axis is coincident to the line passing through r and s when the data reside in a Euclidean space. In the new coordinate system, the coordinate values of the projected images of data objects a and b become $(\mathsf{x}_\mathsf{a},0)$ and $(\mathsf{x}_\mathsf{b},0)$, respectively.

We can also show that with enough iterations, F will preserve the distances between all objects. In particular, observe that projecting on an $(m-1)$-dimensional hyperplane in each iteration reduces the number of points by at least one since the two pivot points get projected to the same point. In other words, if S_i is the set of points obtained by projecting on the hyperplanes in iterations 1 through $i-1$, then $|S_i| \leq |S| - (i-1)$

(as each iteration reduces the number of points by one). Thus, since $|S_{k+1}| \geq 1$, this implies that $1 \leq N - k$ or $k \leq N - 1$, where $N = |S|$. Furthermore, when $k = m$, S_{k+1} is the result of projecting S onto a 0-dimensional hyperplane (i.e., a point), so $k \leq m$. In summary, after at most $\min\{m, N-1\}$ iterations, all the points get projected to the same point. Therefore, $d_E(a_{k+1}, b_{k+1})$ is then zero for all objects $a, b \in S$, so $d_E(a,b) = d'_E(F(a), F(b))$, implying that total distance preservation is obtained.

Above, we saw that applying FastMap is essentially equivalent to applying a linear transformation involving translation and rotation and then retaining only some of the coordinate axes. Thus, FastMap works in a manner analogous to applying KLT (and the equivalent PCA and SVD) for dimension reduction as described in Section 4.6.4.1. However, we can expect that applying KLT (and the equivalent PCA and SVD) will yield a better approximation to the original distances as the coordinate axes with the greatest variance are the ones that are retained.

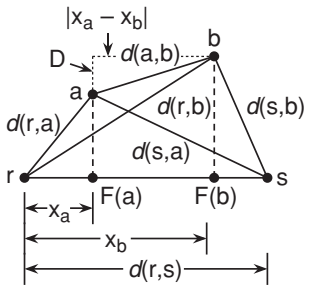

Figure 4.96
Example projection of two objects on the line joining the points corresponding to the pivot objects.

We also saw that the distances between all objects are preserved if $|S_{k+1}| = 1$ since all objects in S are then projected onto a single point in S_{k+1}. An interesting perspective is that even if F does not preserve all distances, F will still preserve the distances between some of the objects. In particular, the distances between any objects a and b that are projected onto the same point in S_{k+1} are preserved by the embedding since $d_{k+1}(a,b)$ is then zero. In fact, we can form equivalence classes for the objects based on the points in S_i such that objects mapped to the same point in S_i belong to the same equivalence class. Thus, the set of objects is partitioned into $|S_{k+1}| \leq N - k$ equivalence classes after k iterations of FastMap.

Figure 4.96 illustrates how FastMap maps objects a and b after having determined just one coordinate value, assuming that r and s are the pivot objects. Notice how it seems intuitively obvious that the distance between F(a) and F(b) is smaller than that between a and b. Unfortunately, intuition is misleading here as it is colored by the fact that we perceive the three-dimensional world around us as obeying Euclidean geometry. In particular, the relative lengths of the line segments between the points in Figure 4.96 can arise only if d is the Euclidean distance metric (or if (S,d) is isometric to a subset of a Euclidean vector space). Thus, we see that in the figure, $d(\mathsf{a},\mathsf{b})^2 = (\mathsf{x_a} - \mathsf{x_b})^2 + \mathsf{D}^2$ (according to the Pythagorean theorem), so we clearly have $d(\mathsf{a},\mathsf{b}) \geq |\mathsf{x_a} - \mathsf{x_b}|$ (and thus the pruning property is satisfied). In general, we cannot assume that this relationship holds (i.e., D^2 may be negative!).

The pruning property is guaranteed to hold only when (S,d) is a subset of a Euclidean space, and d' is the Euclidean distance metric. There are two general scenarios where the pruning property is not guaranteed to hold. The first is when d is the Euclidean distance metric, and d' is a non-Euclidean distance metric. The second is when d is not the Euclidean distance metric (and (S,d) is not isometric to a subset of a Euclidean vector space), regardless of the nature of d', and is due to the implicit assumption in Equations 4.23 and 4.25 that d is the Euclidean distance metric.

An example of the first scenario occurs when the City Block (L_1) distance metric d_A is used for d'. To illustrate this, note that the City Block distance $d_A(F(1), F(2))$ between two points $F(1)$ and $F(2)$ may be greater than the Euclidean (L_2) distance $d_E(F(1), F(2))$ between the same points. For example, $d_A((0,0),(3,4)) = 7$ while $d_E((0,0),(3,4)) = 5$. Thus, we may have $d_A(F(a), F(b)) > d(a,b)$, even though $d_E(F(a), F(b)) \leq d(a,b)$, where $F(a)$ and $F(b)$ are the result of mapping a and b with FastMap. In fact, the pruning property is guaranteed to hold only if d' is the Euclidean distance metric (i.e., L_2) or some other Minkowski distance metric L_p, where $p \geq 2$ (see Exercise 1).

The second, and more serious, scenario where the pruning property does not hold occurs when d is not a Euclidean metric, regardless of the choice of d'. This is serious because the pruning property may be violated after determining any number of coordinate axes. In particular, this means that the value of $d_i(a,b)^2$ in Equation 4.27 can be negative (or zero) for any $i \geq 2$, which precludes using a and b as the pivot pair in iteration i of

Object	a	b	c	e
a	0	10	4	5
b	10	0	8	7
c	4	8	0	1
e	5	7	1	0

Figure 4.97
Distance matrix for an example where Equation 4.23 causes the pruning property to be violated.

FastMap. Thus, the situation can arise that $d_i(a,b)^2 \leq 0$ for all pairs of objects a and b, in which case no more coordinate values can be determined (but see Section 4.7.3.8).

There are two possible causes for the violation of the pruning property: Equation 4.23 and Equation 4.25. Both are due to the implicit assumption made in deriving these equations that the Pythagorean theorem applies to the distance metric d. However, the Pythagorean theorem is unique to Euclidean spaces. As an example where Equation 4.23 causes the pruning property to be violated, consider objects a, b, c, and e, with the distance matrix given in Figure 4.97, for which satisfaction of the triangle inequality can be easily verified.

Since a and b are the objects that are the farthest apart, they are selected as the pivot objects in FastMap when determining the first coordinate axis. Following Equation 4.23, the values of the first coordinate of the points derived from objects c and e are as follows:

$$x_c = (4^2 + 10^2 - 8^2)/(2 \cdot 10) = 52/20 = 13/5$$
$$x_e = (5^2 + 10^2 - 7^2)/(2 \cdot 10) = 76/20 = 19/5$$

Thus, the distance between these one-dimensional points corresponding to objects c and e used in the first iteration of FastMap is $x_e - x_c = 6/5 = 1.2$, which is higher than the original distance between them, obtained from the distance matrix, which is 1. Thus the pruning property does not hold for c and e.

The pruning property is not directly, but indirectly, violated by Equation 4.25. In particular, if d is not a Euclidean distance metric, then Equation 4.25 may lead to a definition of d_H such that d_H does not satisfy the triangle inequality (thereby also failing to be a distance metric), and this, in turn, causes the pruning property to be violated in the next iteration of FastMap. We first show how the violation of the triangle inequality causes the pruning property to be violated.

Assuming that r and s are the pivot objects, the pruning property is violated if $d(r,a) < |x_r - x_a| = x_a$ (since $x_r = 0$). Now, let us explore what conditions are equivalent to $d(r,a) < x_a$, and thus give rise to a pruning property violation.

$$\begin{aligned}
d(r,a) < x_a &\Leftrightarrow d(r,a) < \frac{d(r,a)^2 + d(r,s)^2 - d(s,a)^2}{2d(r,s)} \\
&\Leftrightarrow 2d(r,a)d(r,s) < d(r,a)^2 + d(r,s)^2 - d(s,a)^2 \\
&\Leftrightarrow d(s,a)^2 < d(r,a)^2 - 2d(r,a)d(r,s) + d(r,s)^2 = (d(r,a) - d(r,s))^2 \\
&\Leftrightarrow d(s,a) < |(d(r,a) - d(r,s))| \\
&\Leftrightarrow d(s,a) + d(r,a) < d(r,s) \vee d(s,a) + d(r,s) < d(r,a)
\end{aligned}$$

Similarly, it can be shown that $d(s,a) < |x_s - x_a|$ if and only if $d(r,a) + d(s,a) < d(r,s) \vee d(r,s) + d(r,a) < d(s,a)$. Thus, we see that the pruning property is violated (for $d(r,a)$, $d(s,a)$, or both) if and only if the triangle inequality is violated for any of the distances between the two pivot objects r and s and an arbitrary object a.

Object	a	b	c	e
a	0	6	5	4
b	6	0	3	4
c	5	3	0	6
e	4	4	6	0

Figure 4.98
Distance matrix for an example where Equation 4.25 indirectly causes the pruning property to be violated.

Next, we give an example that actually results in d_H failing to satisfy the triangle inequality (thereby, also failing to be a distance metric). Consider objects a, b, c, and e, with the distance matrix given in Figure 4.98, for which satisfaction of the triangle inequality can be easily verified.

Following Equation 4.23, the values of the first coordinate of the points derived from c and e using a and b as the pivot objects are obtained as follows:

$$x_c = (5^2 + 6^2 - 3^2)/(2 \cdot 6) = 13/3$$
$$x_e = (4^2 + 6^2 - 4^2)/(2 \cdot 6) = 3$$

The distances between the projections of a, c, and e onto the hyperplane H that is orthogonal to the first coordinate axis follow from Equation 4.25:

$$d_H(a,c) = \sqrt{5^2 - (0 - 13/3)^2} \approx 2.494$$
$$d_H(a,e) = \sqrt{4^2 - (0 - 3)^2} \approx 2.647$$
$$d_H(c,e) = \sqrt{6^2 - (13/3 - 3)^2} \approx 5.850$$

Thus, we see that $d_H(a,c) + d_H(a,e) \approx 5.141$, which is less than $d_H(c,e)$, thereby violating the triangle inequality.

Thus far, we have examined the source of violation of the pruning property in FastMap and given examples where FastMap results in an embedding that does not satisfy the pruning property. This enables us to conclude that the embedding produced by FastMap may fail to satisfy the pruning property when d is not a Euclidean distance metric. It is possible to make an even stronger statement: if d is not a Euclidean distance metric, then FastMap will eventually result in an embedding that does not satisfy the pruning property if enough iterations are performed. Toward proving this, we first show that only a finite number of iterations can be applied in FastMap. At the start of this section, we used geometric arguments to show that when d is the Euclidean distance metric, the pivot objects used in an iteration of FastMap are effectively merged and are thereby indistinguishable in subsequent iterations. This is also true for arbitrary metric spaces. In particular, if r and s are pivot objects in iteration $i-1$ of FastMap, it can be shown that $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for an arbitrary object t, regardless of whether d_{i-1} satisfies the triangle inequality (see Exercise 2). Thus, r and s will have identical coordinate values for coordinate axes i through k. Furthermore, we can show that when the distances of any two objects r and s satisfy the distance property above (i.e., $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$), then the distance between r and s is preserved by FastMap (see Exercise 3).

Based on this, we argue below that if (S,d) is a finite metric space that is not isometric to a subset of any Euclidean space, then repeated iterations of FastMap (i.e., k) will eventually result in a mapping F that violates the pruning property. We will show that $k < N - 1$. In particular, since (S,d) is not isometric to a subset of any Euclidean space, we know that no distance-preserving Euclidean embedding exists for (S,d). Thus, such

an embedding cannot be produced by FastMap either. Also, note that $d_E(F_i(a), F_i(b))$ is nondecreasing in i for any objects $a, b \in S$. Therefore, the only way in which FastMap can be applied repeatedly without ever resulting in $d_E(F_i(a), F_i(b)) > d(a,b)$, for some objects a and b, is if an infinite series of iterations can be performed. Below, we show that at most $N-2$ iterations can be performed by FastMap and that the final iteration necessarily will result in an embedding that does not satisfy the pruning property. In the process, we characterize the level of distance preservation among the objects obtained by the embedding.

The above observations can be used to define equivalence classes for data objects whose distances are preserved (in a similar way as we mentioned for Euclidean vector spaces). In particular, we say that two objects r and s (not necessarily pivot objects) belong to the same equivalence class in iteration i if $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for all objects $t \in S$. Initially, each object belongs to a distinct equivalence class (except for objects having a distance of zero from each other). Furthermore, the fact that pivot objects become indistinguishable in the remaining iterations ensures that each iteration of FastMap causes the merger of at least two equivalence classes, namely, the ones containing the two pivot objects. Thus, the number of equivalence classes after i iterations is clearly at most $N-i$, as we initially have at most N equivalence classes and each iteration reduces the number of equivalence classes by at least one.

Since a distance-preserving embedding does not exist for (S,d), at least two equivalence classes must remain after the performance of the last iteration of FastMap (as the existence of just one equivalence class means that the embedding is distance preserving). Hence, the number of iterations k must satisfy $N-k \geq 2$, or $k \leq N-2$. Furthermore, if no more iterations can be performed, then we must have $d_{k+1}(a,b)^2 < 0$ for any objects a and b from distinct equivalence classes since a and b could otherwise be chosen as the next pair of pivot objects, thereby allowing yet another iteration. However, Equation 4.27 then implies that $d(a,b)^2 - d_E(F_k(a), F_k(b))^2 < 0$, or, in other words, $d(a,b) < d_E(F(a), F(b))$, since $F = F_k$. Therefore, we have shown that F does not satisfy the pruning property (as $d' = d_E$). Moreover, since F was chosen in an arbitrary way (i.e., we made no assumption on the choices of pivot objects), we have shown that such a noncontractive embedding is always produced by FastMap, given enough iterations.

Exercises

1. Letting (S,d) be a subset of a Euclidean vector space and d' be some Minkowski distance metric L_p, prove that the embedding F produced by FastMap is guaranteed to satisfy the pruning property only if $p \geq 2$.
2. Letting r and s be the pivot objects in iteration $i-1$ of FastMap, show that $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for an arbitrary object t, regardless of whether d_{i-1} satisfies the triangle inequality.
3. Let r and s be data objects, not necessarily pivot objects. Show that if $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for all objects t after $i-1$ iterations of FastMap, then $d'(F(r), F(s)) = d(r,s)$ where F is the final embedding produced after k iterations of FastMap. In other words, show that the distance between r and s is preserved by F.

4.7.3.7 Expansion

Satisfaction of the pruning property means that $d'(F(o_1), F(o_2)) \leq d(o_1, o_2)$. We are also interested in how much larger the distances in the embedding space can be in comparison to the original distances. This is termed the *expansion* of F and is defined as

$$\max_{o_1, o_2} \left\{ \frac{d'(F(o_1), F(o_2))}{d(o_1, o_2)} \right\}$$

Note that if the expansion is not greater than 1, then F satisfies the pruning property. Furthermore, if we can derive an upper bound c on the expansion, then any embedding F

produced by FastMap can be made to satisfy the pruning property by defining $d'(o_1, o_2) = d_E(F(o_1), F(o_2))/c$ such that the expansion with respect to this d' is no more than 1. Unfortunately, the expansion of embeddings produced by FastMap when determining even just one coordinate is already relatively large (i.e., 3; see Exercise 1) [850], and for additional coordinates the expansion can be very large (see Exercise 2) [850].

Exercises

1. Prove that the expansion after determining one coordinate with FastMap is at most 3 and that this bound is tight.

2. As we saw in Section 4.7.3.6, the triangle inequality does not necessarily hold for the distance functions used in the second and subsequent iterations of FastMap. This means that the upper bound on the expansion derived in Exercise 1 holds only for the first coordinate value obtained by FastMap. The basic problem is that for iteration $i > 1$, the value of $d_i(r,s)^2$, as defined in Equation 4.27, may be less than or equal to zero for all objects s, even after several iterations of the pivot-finding heuristic (see Section 4.7.3.2). Moreover, even when $d_i(r,s)^2$ is found to be strictly positive, it can be arbitrarily close to zero, thereby yielding an arbitrarily large expansion. One solution is to set a strict lower bound on the distance between the pivot objects, based on the distance between the first pair of pivot objects. In other words, if r and s are the first two pivot objects, then any pivot objects t and u chosen in any subsequent iteration i must obey $d_i(t,u) \geq d(r,s)/\beta$, where $\beta > 1$ is some constant. Unfortunately, this requirement means that the pivot-finding heuristic may no longer succeed in finding a legal pair of pivots in a constant number of iterations, and the number of distance computations may become $O(N^2)$. An alternative solution is to terminate FastMap if a legal pivot pair is not found in $O(1)$ iterations of the pivot-finding process (see Section 4.7.3.2). This means that we may obtain fewer coordinate axes than desired. However, this is usually not a problem as a low number of coordinate axes is preferable in most applications. Nevertheless, it is still not clear that the original distances are adequately approximated in the embedding space in cases when legal pivots cannot be found in $O(N)$ distance computations. In any case, letting $d(r,s)/\beta$ ($\beta > 1$ is some constant) be the minimum distance value $d_2(t,u)$ for the pivot objects t and u used in the second iteration of FastMap (i.e., when finding the second coordinate value for each object), as well as for the distances between any two pairs of objects (using the original distance function), show that the expansion after determining two coordinates with FastMap is no more than $36\beta^2$.

4.7.3.8 Heuristic for Non-Euclidean Metrics

As we pointed out in Section 4.7.3.4, it is possible for the value $d_H(t,u)^2$ in Equation 4.25 to be negative when d is not a Euclidean distance metric. More generally, this implies that $d_i(a,b)^2$ may be negative for $i \geq 2$ in Equation 4.27. Such a situation is undesirable since it means that $d_i(a,b)$ becomes complex valued, which precludes the choice of a and b as the pair of pivot objects in iteration i of FastMap. Furthermore, since $d_i(a,b)^2$ can have a large negative value, such values can cause a large expansion in the distances between coordinate values determined by Equation 4.26, as discussed in Section 4.7.3.7.

Wang, Wang, Lin, Shasha, Shapiro, and Zhang [1950] introduce a heuristic to alleviate the situation that $d_i(a,b)^2$ may be negative for $i \geq 2$ in Equation 4.27. The heuristic defines $d_i(a,b)$ ($i \geq 2$) in such a way that it is always real valued but possibly negative:

$$d_i(a,b) = \begin{cases} \sqrt{d_{i-1}(a,b)^2 - (x_a^{i-1} - x_b^{i-1})^2} & \text{if } d_{i-1}(a,b)^2 \geq (x_a^{i-1} - x_b^{i-1})^2 \\ -\sqrt{(x_a^{i-1} - x_b^{i-1})^2 - d_{i-1}(a,b)^2} & \text{otherwise} \end{cases} \quad (4.28)$$

Equivalently, we can use the definition $d_i(a,b) = \text{sign}(d_i(a,b)^2) \cdot \sqrt{|d_i(a,b)^2|}$, where $d_i(a,b)^2$ is defined as in Equation 4.27. Although this heuristic apparently resolves the drawbacks of negative $d_i(a,b)^2$ values, it does not correct the fundamental problem with Equation 4.27, namely, that d_i may violate the triangle inequality if d is not the Euclidean distance metric (see the discussion in Section 4.7.3.6). Furthermore, notice

that this formulation also means that if $d_i(a,b)$ is negative for some i, then $d_j(a,b)^2$ is not equal to $d(a,b)^2 - d_E(F_{j-1}(a), F_{j-1}(b))^2$ for all $j \geq i$.

Notice that when $d_i(a,b)$ is defined according to Equation 4.28, the value of $d_i(a,b)^2$ is always nonnegative, regardless of whether $d_i(a,b)$ is negative. Thus, in situations where Equation 4.27 leads to negative values, the coordinate value x_o^i for an object o as determined by Equation 4.26 can be different, depending on which definitions of d_i is used—that is, Equation 4.27 or 4.28. If we focus on the result of just one iteration of FastMap, neither definition of d_i is always better than the other in terms of how well distances are preserved. In particular, sometimes it is better to use Equation 4.27, and sometimes it is better to use Equation 4.28. However, the advantage of the definition in Equation 4.28 is that the value of $d_i(a,b)^2$ tends to decrease as i increases (i.e., as more iterations are performed). In particular, Equation 4.28 implies that $d_i(a,b)^2 = |d_{i-1}(a,b)^2 - (x_a^{i-1} - x_b^{i-1})^2|$, so $d_i(a,b)^2$ is only larger than $d_{i-1}(a,b)^2$ if $(x_a^{i-1} - x_b^{i-1})^2 > 2d_{i-1}(a,b)^2$. In contrast, according to Equation 4.27, the value of $d_i(a,b)^2$ is monotonically nonincreasing in i, so it can become a large negative value, which makes the mapping not very attractive in similarity searching applications as it results in a large expansion with concomitant adverse effects on search performance (see Section 4.7.3.7). Recall that any amount of expansion means that the distance in the embedding space becomes larger than the distance in the original space, thereby violating the pruning property (see Section 4.7.3.6).

The heuristic of Equation 4.28 enables the use of two objects a and b as a pivot pair even when $d_i(a,b)$ is negative. In contrast, such pairs cannot be utilized with Equation 4.27. However, it is not clear how appropriate such a choice of pivot objects is in terms of attempting to reduce the distance in the embedding space. In particular, the fact that $d_i(a,b)$ is negative implies that the distance between $F_{i-1}(a)$ and $F_{i-1}(b)$ is greater than $d(a,b)$, so using a and b as pivot objects further increases the amount by which the distance in the embedding space exceeds the distance in the original space (i.e., expansion).

In Section 4.7.3.6, we show that FastMap eventually results in an embedding that does not satisfy the pruning property if enough iterations are performed. This result still holds when the heuristic of Equation 4.28 is used. In particular, Equation 4.28, which defines the computation of the intermediate distance functions, is equivalent to Equation 4.27 unless the intermediate mapping F_{i-1} obtained in iteration $i-1$ already violates the pruning property. To see why they are otherwise equivalent, note that if i is the lowest value for which $d_{i-1}(a,b)^2 < (x_a^{i-1} - x_b^{i-1})^2$ for some objects a and b (thereby causing the invocation of the second case in Equation 4.28), then we can apply Equation 4.27 to show that $d(a,b)^2 < d_E(F_{i-1}(a), F_{i-1}(b))^2$ (i.e., F_{i-1} violates the pruning property). Furthermore, when F_{i-1} violates the pruning property, the final mapping F also violates the pruning property as $d_E(F_i(a), F_i(b))$ is monotonically nondecreasing in i (since, as we add coordinate axes, the distance can only get larger).

It is interesting to observe that the largest possible expansion when determining just one coordinate value is just as large with the heuristic as without (i.e., 3, as mentioned in Section 4.7.3.7). Nevertheless, for a larger number of coordinates, use of the heuristic may reduce the worst-case expansion somewhat, but it can still be large (e.g., 7 when determining two coordinate values) [850].

4.7.3.9 Summary and Comparison with Other Embeddings

In this section and in Section 4.7.2, we examine the embeddings of finite metric spaces and evaluate them in the context of their usage for similarity searching in multimedia databases with 100% recall. In similarity search, achieving 100% recall is important as it ensures that no relevant object is dropped from the query response. Particular attention has been paid to Lipschitz embeddings (as exemplified by SparseMap) and FastMap, which was inspired by dimension reduction methods for Euclidean space that are based

on linear transformations such as KLT (and the equivalent PCA and SVD), while also being applicable to arbitrary metric spaces. When the resulting embedding satisfies the pruning property, 100% recall is achieved. Although Linial et al. [1181] have shown how to make the Lipschitz embeddings satisfy the pruning property, we prove that the speedup heuristics that make up the SparseMap adaptation result in an embedding that does not satisfy the pruning property. Moreover, we demonstrate how to modify the SparseMap heuristics so that the resulting embedding does indeed satisfy the pruning property.

In the case of FastMap, we first prove that it satisfies the pruning property when the data is drawn from a Euclidean vector space, and the distance metric in the embedding space is a Minkowski metric L_p ($p \geq 2$, which also includes the Euclidean distance metric). In their development of FastMap, Faloutsos and Lin [587] claim two advantages for FastMap over SVD:

1. SVD takes $O(N \cdot n^2)$ time while FastMap requires $O(N \cdot k)$ distance computations, each of which is $O(n)$, assuming that the data is drawn from an n-dimensional Euclidean vector space. Thus, a more accurate assessment of the execution time complexity of FastMap is $O(Nnk)$ in this setting.
2. FastMap can work for data drawn from an arbitrary metric space (i.e., the only information about the data objects consists of the interobject distances, which are required to satisfy the triangle inequality). SVD, on the other hand, only works for data drawn from a Euclidean vector space (i.e., a vector space with the Euclidean distance metric). Of course, d' for both SVD and FastMap can be any Minkowski distance metric L_p provided $p \geq 2$, although $p = 2$ is usually the most appropriate.

However, we show that the second advantage is somewhat mixed since, in the arbitrary metric space case, it is possible for FastMap not to satisfy the pruning property, which reduces the accuracy of similarity search queries (i.e., we can no longer guarantee 100% recall). We show that this is a direct result of the implicit assumption by Faloutsos and Lin [587] of the applicability of the Pythagorean theorem, which, in the case of a general metric space, can be used only as a heuristic in computing the projected distance values. In fact, this leads to definitions of distance functions at intermediate iterations that do not satisfy the triangle inequality and, thereby, fail to be distance metrics. Failure to satisfy the pruning property enables us to prove the following properties of FastMap for this situation:

1. Given a value k, application of FastMap may not always be possible in the sense that we are not guaranteed to be able to determine k coordinate axes.
2. The distance expansion (as well as the more general concept of distortion) of the embedding can be very large, as evidenced by the bounds that we gave, some of which were attainable, on how much larger the distances in the embedding space can be.
3. The fact that we may not be able to determine k coordinate axes limits the extent of achievable distance preservation. However, more importantly, failure to determine more coordinate axes does not necessarily imply that relative distances among the objects are effectively preserved.
4. The presence of many nonpositive, or very small positive, distance values (which can cause large expansion) in the intermediate distance functions (i.e., those used to determine the second and subsequent coordinate axes) may cause FastMap no longer to satisfy the claimed $O(N)$ bound on the number of distance computations in each iteration. In particular, finding a legal pivot pair may, in the worst case, require examining the distances between a significant fraction of all possible pairs of objects, or $\Omega(N^2)$ distance computations.

In Section 4.7.3.8, we describe a heuristic for non-Euclidean distance metrics in FastMap proposed by Wang et al. [1950]. This heuristic alleviates some of the drawbacks listed above. In particular, it should reduce the amount of distance expansion (as well

as distortion) in the embedding, and the number of object pairs that do not qualify as pivots should be lower, thus reducing the likelihood of not satisfying the $O(N)$ bound on the number of distance computations in each iteration of FastMap. However, a detailed empirical study of the effect of the heuristic on actual datasets remains to be performed.

Wang et al. [1950, 1955, 2045] propose an interesting embedding, termed *MetricMap*, that has some similarity to SVD, FastMap, and a special class of Lipschitz embeddings. In essence, given a finite metric space (S,d), $2k$ of the objects (termed *reference objects*) are selected at random and used to form a coordinate space in $\mathbb{R}^{2k-1}$. In particular, one of the reference objects o_0 is mapped to the origin, while the remaining objects O_i $(1 \leq i \leq 2k-1)$ are mapped to unit vectors e_i in $\mathbb{R}^{2k-1}$. The interobject distances of the reference objects are used to form a matrix M that serves as the basis of a distance measure that preserves the distances between the reference objects. Through a process that resembles SVD, the k "most important" coordinate axes determined by M are established, from which a suitable linear transformation is computed. The k coordinate values for each object in S are then obtained by using the pairs (o_0, o_i) as pivots in a manner similar to FastMap and then applying the linear transformation. The result of MetricMap, like that of FastMap, may violate the pruning property if d is not the Euclidean distance metric. Furthermore, the way in which the original $2k$ reference objects are reduced down to $k+1$ means that, even if d is Euclidean, the result may still violate the pruning property.

The distance function d' used on the coordinate values produced by MetricMap is *pseudo-Euclidean* in that it is defined in terms of the squares of the differences of the individual coordinate values just like the Euclidean distance metric. However, some of the coordinate axes may make a negative contribution to the overall distance between two points. Thus, in extreme cases, the squared distance between points can be negative. In order to produce the square root of such a negative value, MetricMap takes the square root of the absolute value and makes it negative (i.e., $-\sqrt{-D}$). This is similar to what was done in the FastMap heuristic to deal with non-Euclidean distance metrics. Thus, due to the possibility of negative distance values, the values produced by d' should really be termed "pseudo-distances" or "quasi-distances." In contrast, in a Euclidean space, all coordinate axes have a positive contribution to the overall distance so that the more coordinates that we have, the greater is the distance.

At a first glance, MetricMap appears to resemble the Lipschitz embedding of Faragó et al. [595] (mentioned in Sections 4.7.1 and 4.7.2.1) in that each of the reference objects except o_0 are, in some sense, used to construct a dimension in $\mathbb{R}^{2k-1}$. However, this resemblance is superficial as the values of each of the k coordinate axes in the final embedding are potentially influenced by all of the reference objects.

4.7.4 Locality Sensitive Hashing

Locality sensitive hashing (LSH) is a randomized algorithm for similarity searching developed by Indyk and Motwani [924]. A *randomized algorithm* is any algorithm that makes some random (or pseudorandom) choices (e.g., [398, 1318, 1325]. Randomized algorithms are usually characterized as being either Monte Carlo or Las Vegas. A *Monte Carlo randomized algorithm* [1318] is a randomized algorithm that may produce incorrect results, but with bounded error probability. In particular, a Monte Carlo algorithm gives more accurate results the more often it is invoked. On the other hand, a *Las Vegas randomized algorithm* (e.g., [397, 398, 1318]) is a randomized algorithm that always produces correct results, with the only variation from one run to another being its execution time.

One way to understand the difference between these two classes of randomized algorithms is to observe that a Monte Carlo algorithm yields a probabilistic result, while the Las Vegas algorithm yields a deterministic result. For example, the algorithm for finding the nearest neighbor in a kNN graph [1704] (see Section 4.5.6) is a Monte Carlo

algorithm due to the use of pseudorandom seeds to identify the object at which the search starts. Thus, the more times we invoke the algorithm, using different seeds, the greater the likelihood of a correct answer. A bound on the error probability of this method should be derivable, hence justifying its characterization as a Monte Carlo algorithm (see Exercise 2). Therefore, the result of the search process is probabilistic, with some likelihood of an error. On the other hand, the nearest neighbor finding algorithm for a Delaunay graph [1424] (see Section 4.5.5) is a Las Vegas algorithm as the result is always correct, but the execution time varies depending on the starting object.

The approximate k-nearest neighbor algorithm for the SASH (Spatial Approximation Sample Hierarchy) method [883, 884] (see Section 4.5.8) is an example of an algorithm that is neither a Monte Carlo nor a Las Vegas algorithm. Recall that SASH uses a probabilistic algorithm to distribute the objects at the various levels. However, once the SASH has been constructed, all instantiations of an algorithm such as the one for finding the k nearest neighbors of a query object will have the same execution time and will return the same result, which may not be accurate; moreover, there is no known bound on the extent to which the answer deviates from the correct one. Thus, the SASH approximate k-nearest neighbor algorithm fails to be a Las Vegas algorithm on two counts. First, it does not necessarily yield the correct result, and, second, there is no variation in its execution time from one instantiation to another. The algorithm also fails to be a Monte Carlo algorithm in that repeated invocations with the same query object yield the same result—that is, neither does the accuracy of the answer change, nor is there a known bound on it.

The goal in LSH is to find a hashing function that is approximately distance preserving. LSH differs from other approaches by using a two-stage process that employs the embed-project-hash paradigm [510]. The key idea is for the hashing functions used in the two stages to be distance preserving, or at least to be approximate distance preserving within some tolerance.

Indyk and Motwani implement the first stage by embedding objects (usually point objects) drawn from $\mathbb{R}^d$ under a Minkowski L_p norm (i.e., distance metric) into the Hamming space $\{0,1\}^i$ for some i, which is a vector space whose elements are strings of zeros and ones (i.e., bitstrings), and then use the L_1 distance metric (i.e., City Block). In essence, the value c_i of each feature f_i that makes up the multidimensional object at hand is mapped into a vector v_i in a Hamming space. In particular, letting feature value c_i be in range $[0,t]$ so that t is the integer greater than or equal to ($\geq$) the maximum among all feature values [709], v_i is represented by a string of c_i ones followed by $t - c_i$ zeros (e.g., the feature value 5 in the domain $[0,7]$ is mapped into the bitstring 1111100, corresponding to five ones followed by two zeros). The vectors v_i are concatenated to form a single bitstring V (see also Exercise 3), which is an element of the Hamming space $\{0,1\}^{td}$. It should be clear that this embedding, coupled with the L_1 distance metric, is distance preserving.

Indyk and Motwani implement the second stage by projecting the result V of the first stage onto an m-dimensional subspace of the Hamming space, which we term the *projection space*. This projection is achieved by drawing m bit positions at random with replacement. As we point out below, Indyk and Motwani show that coupling this embedding with the L_1 distance metric is approximately distance preserving within some tolerance.

Notice that this two-stage process involves two "hashing" functions. The first hashing function, h_1, corresponds to mapping the original data to the Hamming space. The second hashing function, h_2, corresponds to projecting from the Hamming space to the projection space. Queries are performed in the projection space. In order to make the approach practical, we need to be able to map a query object o in the original metric space to an object in the projection space, and this is achieved by making use of a third hashing function h_3 corresponding to the composition of h_2 and h_1 (i.e., $h_3 \equiv h_2 \circ h_1$). Note that the unary hashing function h_1 based on a Hamming space proposed by Indyk

and Motwani [924] is just one of several possible hashing functions that could be used for the LSH function. In particular, Indyk [923] points out that similar hashing functions were used by Karp, Waarts, and Zweig [995] for the bit vector intersection problem and by Broder, Glassman, Manasse, and Zweig [265] in information-retrieval and pattern recognition applications.

As a more concrete example of LSH using the Hamming space, let us consider the set of records consisting of eight cities and their locations specified by their x and y coordinate values, as shown in Figure 1.1 in Chapter 1 and given in Figure 4.99. In order to make our example manageable, we have quantized the x and y coordinate values so that they lie in the range $[0, 7]$ by applying the mapping $f(x) = x \div 12.5$ and $f(y) = y \div 12.5$, respectively. The result of mapping the quantized coordinate values (i.e., $f(x)$ and $f(y)$) to the Hamming space is given by the function unary, and the result of the concatenation of the mappings is given by the function HS.

Figure 4.99 contains four examples of possible instances of projections involving three bit positions (i.e., {2,9,13}, {7,10,14}, {1,5,11}, and {8,12,14}). The physical interpretations of the mapping of the original data from the Hamming space to the projection space for these examples are given in Figure 4.100. In particular, the fact that $m = 3$ means that we are effectively mapping the objects into vertices of the unit cube. From this figure, we can quickly see that collisions exist whenever more than one object is mapped into the same vertex. Thus, by examining the locations in the original metric space of the objects associated with the same vertex of a cube (i.e., at a projected Hamming distance of 0), we find that some collisions, such as Chicago and Omaha for the {2,9,13} projection instance (Figure 4.100(a)), can be characterized as "good" collisions as they are close to each other in both the original metric space and in the projected Hamming space. On the other hand, collisions such as Atlanta and Denver for the {8,12,14} projection (Figure 4.100(d)) can be characterized as "bad' collisions as they are far from each other in the original metric space, while they are close to each other in the projected Hamming space.

It should be clear that the motivation for the formulation of h_3 is to reduce the probability of bad collisions to less than the probability of good collisions. Any hashing function that has this effect can be characterized as *locality preserving*, and such functions are termed *locality sensitive hashing functions* by Indyk and Motwani [924]. Indyk and Motwani [924] reduce the probability of bad collisions by increasing the number of different projections (i.e., increasing the number of hashing functions h_2) that are taken into account when computing the approximate nearest or k nearest neighbors to j. Notice that using this approach makes LSH a Monte Carlo randomized algorithm as the effect is to increase the accuracy of the query results by additional invocations. In this respect, the effect is reminiscent of (and analogous to) that which results from increasing the

NAME	ATTRIBUTE i		$f(i) = i \div 12.5$		HS = UNARY(X) & UNARY(Y)		PROJECTION INSTANCES			
	X	Y	f(X)	f(Y)	UNARY(X)	UNARY(Y)	{2,9,13}	{7,10,14}	{1,5,11}	{8,12,14}
Chicago	35	42	2	3	1100000	1110000	110	010	100	100
Mobile	52	10	4	0	1111000	0000000	100	000	100	000
Toronto	62	77	4	6	1111000	1111110	111	010	101	110
Buffalo	82	65	6	5	1111110	1111100	110	010	111	110
Denver	5	45	0	3	0000000	1110000	010	010	000	100
Omaha	27	35	2	2	1100000	1100000	110	000	100	100
Atlanta	85	15	6	1	1111110	1000000	100	000	110	100
Miami	90	5	7	0	1111111	0000000	100	100	110	000
q = Reno	5	55	0	4	0000000	1111000	010	010	001	100

Figure 4.99
Example of LSH using the data in Figure 1.1 in Chapter 1.

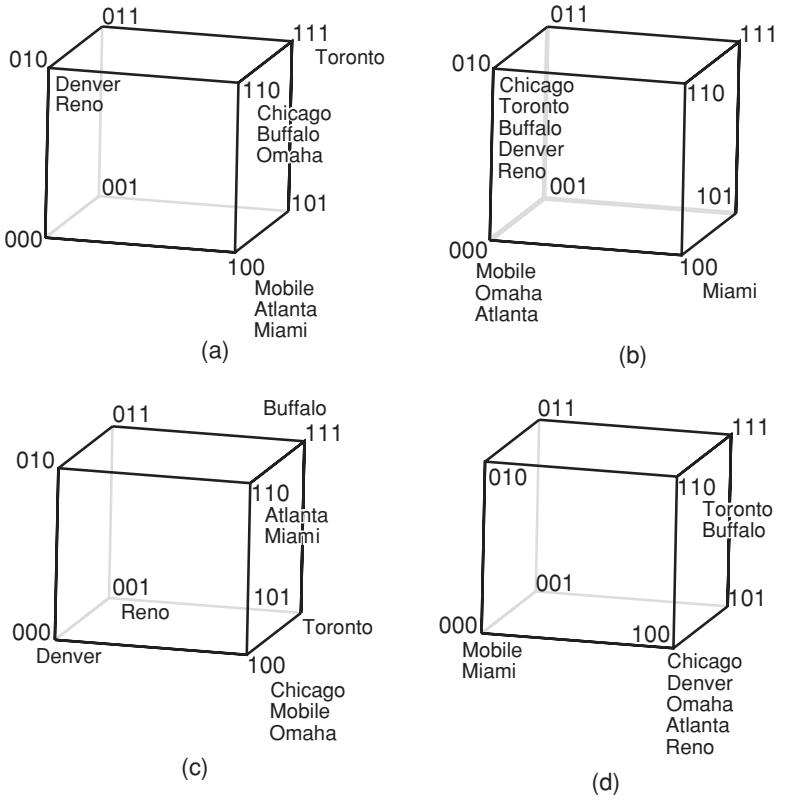

Figure 4.100
The physical interpretation of the mapping of the quantized representation of the original data in Figure 1.1 in Chapter 1 from the Hamming space to the projection space given by considering the three-bit position sequences (a) {2,9,13}, (b) {7,10,14}, (c) {1,5,11}, and (d) {8,12,14}.

number of seeds when finding the k nearest neighbors using the kNN graph. In addition, we can also attempt to increase the dimension m of the projected space to s as a means of reducing the number of additional projections a needed while maintaining the same number of bit positions drawn from the Hamming space (i.e., a total of $j \cdot m = a \cdot s$).

Executing a k-nearest neighbor query for q in LSH is accomplished by applying the hashing function h_3 to q, thereby obtaining q_p, and collecting all of the objects associated with q_p (i.e., these objects collide with q as they are associated with the same elements of the projection space and make up the *collision set* of q_p). This process is repeated j times, once for the j different projections, obtaining q_{pi} for each projection i $(1 \leq i \leq j)$. At this point, we must decide what to report as the response to the query. One possible response to the k-nearest neighbor query is to report the k most frequently occurring objects in the j collision sets of q_{pi} for each projection i $(1 \leq i \leq j)$. Of course, for some query objects, the situation could arise that the corresponding collision sets are empty. In this case, a reasonable solution is to examine the objects associated with elements of the projection space that are at an increasing distance from the elements q_{pi} for each projection i $(1 \leq i \leq j)$.

As an example of the use of LSH, let us examine how it is used to determine the approximate nearest neighbor of $q = \mathsf{Reno}$ at (5,55), whose corresponding embedding is shown in the last row and column of the table in Figure 4.99. We make use of the same four instances of projections involving three bit positions shown in Figure 4.99, whose physical interpretation of the mapping of the original data from the Hamming space to the projection space is given in Figure 4.100. Applying these four projections to $q = \mathsf{Reno}$, we find that q collides with Denver at object 010 for projection {2,9,13};

Object	Chicago	Mobile	Toronto	Buffalo	Denver	Omaha	Atlanta	Miami	q = Reno
Chicago		1	1	11	11	111	1		1
Mobile	3.6					1	11	111	1
Toronto	3.6	6.0		1					1
Buffalo	4.5	5.4	2.1						1
Denver	2.0	5.0	5.0	6.3					111
Omaha	1.0	2.8	4.5	5.0	2.2				1
Atlanta	4.5	2.2	5.4	4.0	6.3	4.1		1	1
Miami	5.8	3.0	6.7	5.1	7.6	5.4	1.4		
q = Reno	2.2	5.7	4.5	6.1	1.0	2.8	6.7	8.1	

Figure 4.101
The upper triangular half of the matrix, above the empty diagonal, indicates the unary representation of the number of times objects (a,b) are members of the collision list of the same element of the projection space for the projections given in Figure 4.99. The lower triangular half of the matrix, below the empty diagonal, indicates the interobject distances in the quantized original metric space.

with Chicago, Toronto, Buffalo, and Denver at object 010 for projection {7,10,14}; with an empty set at object 001 for projection {1,5,11}; and with Chicago, Omaha, Atlanta, and Denver at object 100 for projection {8,12,14}. Clearly, Reno collides most often with Denver, and hence Denver is returned as the approximate nearest neighbor of Reno. The upper triangular half of the matrix, above the empty diagonal, given in Figure 4.101 summarizes the various collision lists by indicating, using a unary representation, how many times each pair of objects, including Reno, collides, where, for example, the entry (Mobile, Miami) has a value of 111, denoting that Mobile and Miami occur three times as members of the collision lists of the same element of the projection space for the four instances of projections involving the three bit positions. The lower triangular half of the matrix, below the empty diagonal, gives the interobject distances in the quantized original metric space.

Assuming a d-dimensional metric space, N objects, and j projection instances, the approximate nearest neighbor query using LSH has an execution time of $O(d \cdot j)$ and requires $O(N \cdot (d + j))$ space. The fact that we repeat the process serves to reduce the likelihood that there are no collisions. If, in fact, we have no collisions for a particular query, then we could examine the objects associated with elements of the projection space that are at increasing distance from the elements q_{pi} for each projection i $(1 \leq i \leq j)$, as mentioned earlier. Alternatively, we can increase the number of projections, but this is not very attractive as it is equivalent to rebuilding a hash table upon overflow, which is not always practical. Thus, such an occurrence serves as a reminder that LSH is an approximate nearest neighbor algorithm and has a probability of not returning the proper response.

Indyk and Motwani [924] prove that h_2 is approximately distance preserving within a tolerance, thereby justifying its use and thereby rendering h_3 also to be approximate distance preserving as h_3 is the result of composing h_2 with a distance-preserving hashing function h_1. This is done by appealing to the result of Frankl and Maehara [631] that improves upon the Johnson-Lindenstrauss Lemma [962], which states that given an ε and N points in a metric space, a mapping that projects onto some subspace defined by approximately $9\varepsilon^{-2}\ln N$ random lines is a distance-preserving mapping within an error of ε. Indyk [923] notes that this appeal to the improvement of the Johnson-Lindenstrauss Lemma by Frankl and Maehara is similar to that deployed by Kleinberg [1032] and Kushilevitz, Ostrovsky, and Rabani [1097] in their use of normal projections onto randomly chosen lines that pass through the origin in approximate nearest neighbor searching.

Note that, although we earlier characterized LSH as a Monte Carlo algorithm, Indyk [922] has also subsequently shown how the LSH method can be converted into a Las Vegas algorithm using a greedy set cover algorithm (e.g., [385]). The LSH method has been found to perform well in large databases where the data resides on disk [709]. It has found use in a number of applications in similarity searching of images in a number of computer vision applications (e.g., [700, 1523, 1737]).

Loosely speaking, LSH can be said to be similar in spirit to Lipschitz embeddings (see Section 4.7.2). Recall that in Lipschitz embeddings objects are embedded in a vector space where the coordinates consist of reference sets, each of which makes up a subset of the objects in the underlying domain S. The embedding is constructed by computing the distance of the embedded objects from the different reference sets, which is a form of a projection onto the reference sets. By using a sufficiently large number of randomly formed reference sets, the distortion of the values of distances in the embedding space is guaranteed to be bounded with a high probability. This is a form of approximate distance preservation. LSH replaces the "large" number of randomly formed reference sets by repeated application of the projection step using different elements of the embedding space that are somewhat analogous to reference sets in the Lipschitz embedding. The analogy can be made more concrete by appealing to the connection that we mentioned earlier between LSH and the solutions of Kleinberg [1032] and Kushilevitz, Ostrovsky, and Rabani [1097] in their use of random projections onto lines in approximate nearest neighbor searching. In particular, in this case, the random lines are analogous to the reference sets (see Exercise 4).

Exercises

1. Compare LSH and SASH for finding the approximate k nearest neighbors.
2. Derive a bound on the error probability of the algorithm for finding the nearest neighbor in a kNN graph, thereby demonstrating that it can be characterized as a Monte Carlo randomized algorithm.
3. At an initial glance, the mapping into a Hamming space that we described appears to resemble bit concatenation as described in Section 1.6 of Chapter 1 in the sense that the unary representations of the quantized values of the individual features that describe the multidimensional object are concatenated. Would using the analog of bit interleaving applied to the unary representation lead to improved performance?
4. (Sydney D'Silva) In the text, we argue that LSH is similar in spirit to Lipschitz embeddings (see Section 4.7.2). One way to rationalize this connection is to consider the related predecessor methods of Kleinberg [1032] and Kushilevitz, Ostrovsky, and Rabani [1097] that make use of random projections onto lines. In particular, we claim that the random choice of a subset of objects A_i and the mapping of each object o to $(d_1, d_2, .., d_k)$, where d_i is the distance from o to the nearest object in A_i, is akin to the random projections used in LSH. D'Silva [510] has observed that one of the difficulties with such an analogy is the absence of a common reference object with respect to which all of the distances that form the "coordinate" values are measured (e.g., an "origin" object) in the case of the Lipschitz embedding. In order to overcome this difficulty, D'Silva proposes to modify the Lipschitz embedding by imposing a point of origin O so that for each object o its mapping to tuple $(d_1, d_2, .., d_k)$ is such that d_i is the distance from the origin O to the closest object to o in A_i, instead of the distance from o to the closest object in A_i. The result is that this modified Lipschitz embedding is analogous to the method of Kleinberg [1032]. In order to complete the comparison, can you prove that the modified Lipschitz embedding is approximately distance preserving within a tolerance, thereby validating the relationship between Lipschitz embeddings and LSH?