Dimensionality Reduction
A Short Tutorial

Ali Ghodsi

Department of Statistics and Actuarial Science
University of Waterloo

Waterloo, Ontario, Canada, 2006

Contents

List of Tables

List of Figures

Chapter 1

An Introduction to Spectral Dimensionality Reduction Methods

Manifold learning is a significant problem across a wide variety of information processing fields including pattern recognition, data compression, machine learning, and database navigation. In many problems, the measured data vectors are high-dimensional but we may have reason to believe that the data lie near a lower-dimensional manifold. In other words, we may believe that high-dimensional data are multiple, indirect measurements of an underlying source, which typically cannot be directly measured. Learning a suitable low-dimensional manifold from high-dimensional data is essentially the same as learning this underlying source.

Dimensionality reduction[1] can also be seen as the process of deriving a set of degrees of freedom which can be used to reproduce most of the variability of a data set. Consider a set of images produced by the rotation of a face through different angles. Clearly only one degree of freedom is being altered, and thus the images lie along a continuous one-dimensional curve through image space. Figure 1.1 shows an example of image data that exhibits one intrinsic dimension.

Manifold learning techniques can be used in different ways including:

- Data dimensionality reduction: Produce a compact low-dimensional encoding of a given high-dimensional data set.
- Data visualization: Provide an interpretation of a given data set in terms of intrinsic degree of freedom, usually as a by-product of data dimensionality reduction.

[1] In this tutorial 'manifold learning' and 'dimensionality reduction' are used interchangeably.

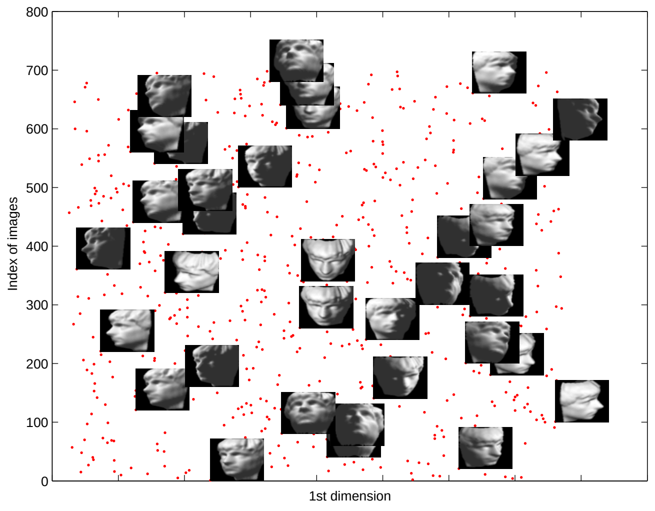

Figure 1.1: *A canonical dimensionality reduction problem from visual perception. The input consists of a sequence of 4096-dimensional vectors, representing the brightness values of 64 pixel by 64 pixel images of a face. Applied to* $N = 698$ *raw images. The first coordinate axis of the embedding correlates highly with one of the degrees of freedom underlying the original data: left-right pose.*

- Preprocessing for supervised learning: Simplify, reduce, and clean the data for subsequent supervised training.

Many algorithms for dimensionality reduction have been developed to accomplish these tasks. However, since the need for such analysis arises in many areas of study, contributions to the field have come from many disciplines. While all of these methods have a similar goal, approaches to the problem are different.

Principal components analysis (PCA) [8] is a classical method that provides a sequence of best linear approximations to a given high-dimensional observation. It is one of the most popular techniques for dimensionality reduction. However, its effectiveness is limited by its global linearity. Multidimensional scaling (MDS) [3], which is closely related to PCA, suffers from the same drawback. Factor analysis [4, 17] and independent component analysis (ICA) [7] also assume that the underling manifold is a linear subspace. However, they differ from PCA in the way they identify and model the subspace. The subspace modeled by PCA captures the maximum variability in the data, and can be viewed as modeling the covariance structure of the data, whereas factor analysis models the correlation structure. ICA starts from a factor analysis solution and searches for rotations that lead to independent components [17, 2].The main drawback with all these classical dimensionality reduction approaches is that they only characterize *linear* subspaces (manifolds) in the data. In order to resolve the problem of dimensionality reduction in *nonlinear* cases, many recent techniques, including kernel PCA [10, 15], locally linear embedding (LLE) [12, 13], Laplacian eigenmaps (LEM) [1], Isomap [18, 19], and semidefinite embedding (SDE) [21, 20] have been proposed.

1.1 Principal Components Analysis

Principal components analysis (PCA) is a very popular technique for dimensionality reduction. Given a set of data on n dimensions, PCA aims to find a linear subspace of dimension d lower than n such that the data points lie mainly on this linear subspace (See Figure 1.2 as an example of a two-dimensional projection found by PCA). Such a reduced subspace attempts to maintain most of the variability of the data.

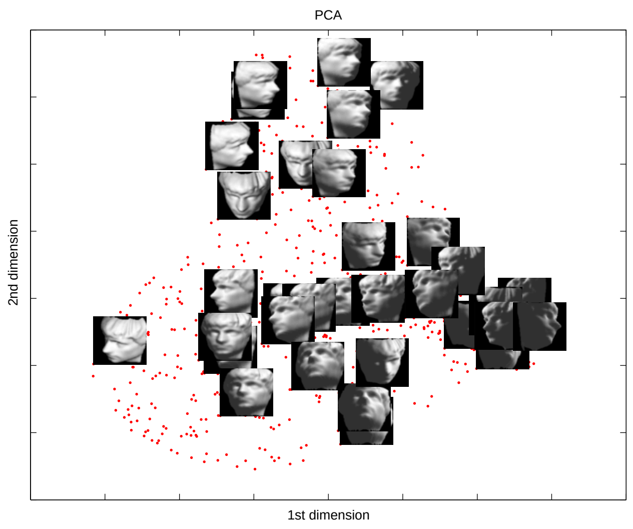

Figure 1.2: *PCA applied to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

The linear subspace can be specified by d orthogonal vectors that form a new coordinate system, called the 'principal components'. The principal components are orthogonal, linear transformations of the original data points, so there can be no more than n of them.

However, the hope is that only $d < n$ principal components are needed to approximate the space spanned by the n original axes.

The most common definition of PCA, due to Hotelling [6], is that, for a given set of data vectors x_i, $i \in 1...t$, the d principal axes are those orthonormal axes onto which the variance retained under projection is maximal.

In order to capture as much of the variability as possible, let us choose the first principal component, denoted by U_1, to have maximum variance. Suppose that all centered observations are stacked into the columns of an $n \times t$ matrix X, where each column corresponds to an n-dimensional observation and there are t observations. Let the first principal component be a linear combination of X defined by coefficients (or weights) $w = [w_1...w_n]$. In matrix form:

$$U_1 = w^T X$$

$$var(U_1) = var(w^T X) = w^T S w$$

where S is the $n \times n$ sample covariance matrix of X.

Clearly $var(U_1)$ can be made arbitrarily large by increasing the magnitude of w. Therefore, we choose w to maximize $w^T S w$ while constraining w to have unit length.

$$\max \ w^T S w$$
$$subject\ to\ w^T w = 1$$

To solve this optimization problem a Lagrange multiplier α_1 is introduced:

$$L(w, \alpha) = w^T S w - \alpha_1 (w^T w - 1) \tag{1.1}$$

Differentiating with respect to w gives n equations,

$$S w = \alpha_1 w$$

Premultiplying both sides by w^T we have:

$$w^T S w = \alpha_1 w^T w = \alpha_1$$

$var(U_1)$ is maximized if α_1 is the largest eigenvalue of S.

Clearly α_1 and w are an eigenvalue and an eigenvector of S. Differentiating (1.1) with respect to the Lagrange multiplier α_1 gives us back the constraint:

$$w^T w = 1$$

This shows that the first principal component is given by the normalized eigenvector with the largest associated eigenvalue of the sample covariance matrix S. A similar argument can show that the d dominant eigenvectors of covariance matrix S determine the first d principal components.

Another nice property of PCA, closely related to the original discussion by Pearson [11], is that the projection onto the principal subspace minimizes the squared reconstruction error, $\sum_{i=1}^{t} ||x_i - \hat{x}_i||^2$. In other words, the principal components of a set of data in $\Re^n$ provide a sequence of best linear approximations to that data, for all ranks $d \leq n$.

Consider the rank-d linear approximation model as :

$$f(y) = \bar{x} + U_d y$$

This is the parametric representation of a hyperplane of rank d.

For convenience, suppose $\bar{x} = 0$ (otherwise the observations can be simply replaced by their centered versions $\tilde{x} = x_i - \bar{x}$). Under this assumption the rank d linear model would be $f(y) = U_d y$, where U_d is a $n \times d$ matrix with d orthogonal unit vectors as columns and y is a vector of parameters. Fitting this model to the data by least squares leaves us to minimize the reconstruction error:

$$\min_{U_d, y_i} \sum_{i}^{t} ||x_i - U_d y_i||^2$$

By partial optimization for y_i we obtain:

$$\frac{d}{dy_i} = 0 \Rightarrow y_i = U_d^T x_i$$

Now we need to find the orthogonal matrix U_d:

$$\min_{U_d} \sum_{i}^{t} ||x_i - U_d U_d^T x_i||^2$$

Define $H_d = U_d U_d^T$. H_d is a $n \times n$ matrix which acts as a projection matrix and projects each data point x_i onto its rank d reconstruction. In other words, $H_d x_i$ is the orthogonal projection of x_i onto the subspace spanned by the columns of U_d. A unique solution U can be obtained by finding the singular value decomposition of X [17]. For each rank d, U_d consists of the first d columns of U.

Clearly the solution for U can be expressed as singular value decomposition (SVD) of X.

$$X = U \Sigma V^T$$

since the columns of U in the SVD contain the eigenvectors of XX^T. The PCA procedure is summarized in Algorithm 1 (see Table 1.1).

Algorithm 1

Recover basis: Calculate $XX^\top = \sum_{i=1}^{t} x_i x_i^\top$ and let U = eigenvectors of $XX^\top$ corresponding to the top d eigenvalues.

Encode training data: $Y = U^\top X$ where Y is a $d \times t$ matrix of encodings of the original data.

Reconstruct training data: $\hat{X} = UY = UU^\top X$.

Encode test example: $y = U^\top x$ where y is a d-dimensional encoding of x.

Reconstruct test example: $\hat{x} = Uy = UU^\top x$.

Table 1.1: *Direct PCA Algorithm*

1.1.1 Dual PCA

It turns out that the singular value decomposition also allows us to formulate the principle components algorithm entirely in terms of dot products between data points and limit the direct dependence on the original dimensionality n. This fact will become important below.

Assume that the dimensionality n of the $n \times t$ matrix of data X is large (i.e., $n >> t$). In this case, Algorithm 1 (Table 1.1) is impractical. We would prefer a run time that depends only on the number of training examples t, or that at least has a reduced dependence on n.

Note that in the SVD factorization $X = U\Sigma V^T$, the eigenvectors in U corresponding to nonzero singular values in Σ (square roots of eigenvalues) are in a one-to-one correspondence with the eigenvectors in V.

Now assume that we perform dimensionality reduction on U and keep only the first d eigenvectors, corresponding to the top d nonzero singular values in Σ. These eigenvectors will still be in a one-to-one correspondence with the first d eigenvectors in V:

$$X\,V \;=\; U\,\Sigma$$

where the dimensions of these matrices are:

$$\begin{array}{cccc} X & U & \Sigma & V \\ n \times t & n \times d & d \times d & t \times d \\ & & \text{diagonal} & \end{array}$$

Crucially, Σ is now square and invertible, because its diagonal has nonzero entries. Thus, the following conversion between the top d eigenvectors can be derived:

$$U \quad = \quad X\, V\, \Sigma^{-1} \tag{1.2}$$

Replacing all uses of U in Algorithm 1 with $XV\Sigma^{-1}$ gives us the dual form of PCA, Algorithm 2 (see Table 1.2). Note that in Algorithm 2 (Table 1.2), the steps of *"Reconstruct training data"* and *"Reconstruction test example"* still depend on n, and therefore still will be impractical in the case that the original dimensionality n is very large. However all other steps can be done conveniently in the run time that depends only on the number of training examples t.

Algorithm 2

Recover basis: Calculate $X^\top X$ and let $V =$ eigenvectors of $X^\top X$ corresponding to the top d eigenvalues. Let $\Sigma =$ diagonal matrix of *square roots* of the top d eigenvalues.

Encode training data: $Y = U^\top X = \Sigma V^\top$ where Y is a $d \times t$ matrix of encodings of the original data.

Reconstruct training data: $\hat{X} = UY = U\Sigma V^\top = XV\Sigma^{-1}\Sigma V^\top = XVV^\top$.

Encode test example: $y = U^\top x = \Sigma^{-1}V^\top X^\top x = \Sigma^{-1}V^\top X^\top x$ where y is a d dimensional encoding of x.

Reconstruct test example: $\hat{x} = Uy = UU^\top x = XV\Sigma^{-2}V^\top X^\top x = XV\Sigma^{-2}V^\top X^\top x$.

Table 1.2: *Dual PCA Algorithm*

1.2 Kernel PCA

PCA is designed to model linear variabilities in high-dimensional data. However, many high dimensional data sets have a nonlinear nature. In these cases the high-dimensional data lie on or near a nonlinear manifold (not a linear subspace) and therefore PCA can not model the variability of the data correctly. One of the algorithms designed to address the problem of nonlinear dimensionality reduction is Kernel PCA (See Figure 1.3 for an example). In Kernel PCA, through the use of kernels, principle components can be computed efficiently in high-dimensional feature spaces that are related to the input space by some nonlinear mapping.

Kernel PCA finds principal components which are nonlinearly related to the input space by performing PCA in the space produced by the nonlinear mapping, where the low-dimensional latent structure is, hopefully, easier to discover.

Consider a feature space $\mathcal{H}$ such that:

$$\Phi : x \to \mathcal{H}$$

$$x \mapsto \Phi(x)$$

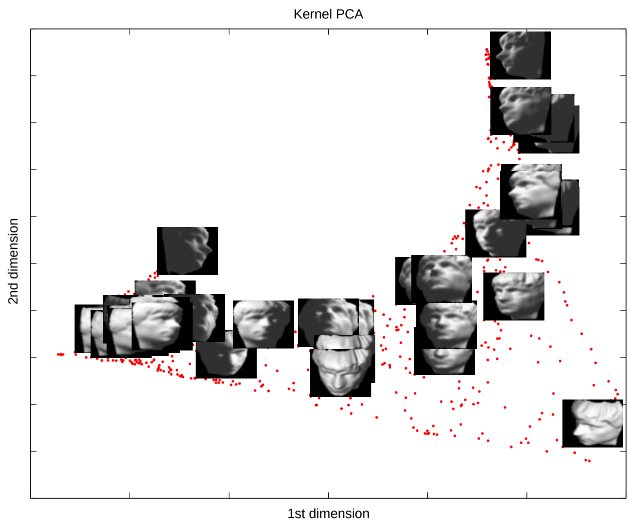

Figure 1.3: *Kernel PCA with Gaussian kernel applied to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

Suppose $\sum_i^t \Phi(x_i) = 0$ (we will return to this point below, and show how this condition can be satisfied in a Hilbert space). This allows us to formulate the kernel PCA objective as follows:

$$\min \sum_i^t ||\Phi(x_i) - U_q U_q^T \Phi(x_i)||$$

By the same argument used for PCA, the solution can be found by SVD:

$$\Phi(X) = U \Sigma V^T$$

where U contains the eigenvectors of $\Phi(X)\Phi(X)^T$. Note that if $\Phi(X)$ is $\mathbf{n} \times t$ and the dimensionality of the feature space $\mathbf{n}$ is large, then U is $\mathbf{n} \times \mathbf{n}$ which will make PCA impractical.

To reduce the dependence on $\mathbf{n}$, first assume that we have a kernel $K(\cdot, \cdot)$ that allows us to compute $K(x, y) = \Phi(x)^\top \Phi(y)$. Given such a function, we can then compute the matrix $\Phi(X)^\top \Phi(X) = K$ efficiently, without computing $\Phi(X)$ explicitly. Crucially, K is $t \times t$ here and does not depend on $\mathbf{n}$. Therefore it can be computed in a run time that depends only on t. Also, note that PCA can be formulated entirely in terms of dot products between data points (Algorithm 2 represented in Table 1.2). Replacing dot products in Algorithm 2 (1.2) by kernel function K, which is in fact equivalent to the inner product of a Hilbert space yields to the Kernel PCA algorithm.

1.2.1 Centering

In the derivation of the kernel PCA we assumed that $\Phi(X)$ has zero mean. The following normalization of the kernel satisfies this condition.

$$\tilde{K}(x,y) = K(x,y) - E_x[K(x,y)] - E_y[K(x,y)] + E_x[E_y[K(x,y)]]$$

In order to prove that, define:

$$\tilde{\Phi}(X) = \Phi(X) - E_x[\Phi(X)]$$

Finally, the corresponding kernel is:

$$\tilde{K}(x,y) = \tilde{\Phi}(x)\Phi\tilde{(y)}$$

This expands as follows:

$$\tilde{K}(x,y) = (\Phi(x) - E_x[\Phi(x)]).(\Phi(y) - E_y[\Phi(y)])$$

$$= K(x,y) - E_x[K(x,y)] - E_y[K(x,y)] + E_x[E_y[K(x,y)]]$$

To perform Kernel PCA, one needs to replace all dot products $x^T y$ by $\tilde{K}(x,y)$ in Algorithm 2 (Table 1.2). Note that V is the eigenvectors of $K(X,X)$ corresponding to the top d eigenvalues, and Σ is diagonal matrix of square roots of the top d eigenvalues.

Unfortunately Kernel PCA does not inherit all the strength of PCA. More specifically reconstruction of training and test data points is not a trivial practice in Kernel PCA. Algorithm 2 (Table 1.2) shows that data can be reconstructed in feature space $\hat{\Phi}(x)$. However finding the corresponding pattern x is difficult and sometimes even impossible [14].

1.3 Locally Linear Embedding

Locally linear embedding (LLE) is another approach which address the problem of nonlinear dimensionality reduction (See Figure 1.4 for an example) by computing low-dimensional, neighbourhood preserving embedding of high-dimensional data. A data set of dimensionality n, which is assumed to lie on or near a smooth nonlinear manifold of dimensionality $d < n$, is mapped into a single global coordinate system of lower dimensionality, d. The global nonlinear structure is recovered by locally linear fits.

Consider t n-dimensional real-valued vectors x_i sampled from some underlying manifold. We can assume each data point and its neighbours lie on, or are close to, a locally linear

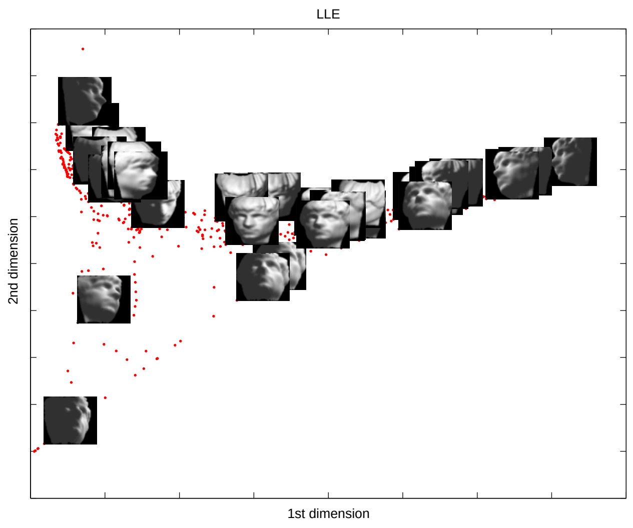

Figure 1.4: *LLE applied ($k = 5$) to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

patch of the manifold. By a linear mapping, consisting of a translation, rotation, and rescaling, the high-dimensional coordinates of each neighbourhood can be mapped to global internal coordinates on the manifold. Thus, the nonlinear structure of the data can be identified through two linear steps: first, compute the locally linear patches, and second, compute the linear mapping to the coordinate system on the manifold.

The main goal here is to map the high-dimensional data points to the single global coordinate system of the manifold such that the relationships between neighbouring points are preserved. This proceeds in three steps:

1. Identify the neighbours of each data point x_i. This can be done by finding the k nearest neighbours, or by choosing all points within some fixed radius, ϵ.

2. Compute the weights that best linearly reconstruct x_i from its neighbours.

3. Find the low-dimensional embedding vector y_i which is best reconstructed by the weights determined in the previous step.

After finding the nearest neighbours in the first step, the second step must compute a local geometry for each locally linear patch. This geometry is characterized by linear coefficients that reconstruct each data point from its neighbours.

$$\min_{w} \sum_{i=1}^{t} ||\mathbf{x}_i - \sum_{j=1}^{k} w_{ij}\mathbf{x}_{N_i(j)}||^2$$

where $N_i(j)$ is the index of the jth neighbour of the ith point. It then selects code vectors so as to preserve the reconstruction weights by solving

$$\min_{Y} \sum_{i=1}^{t} ||\mathbf{y}_i - \sum_{j=1}^{k} w_{ij}\mathbf{y}_{N_i(j)}||^2$$

This objective can be reformulated as

$$\min_{Y} \operatorname{Tr}(Y^T Y L) \tag{1.3}$$

where $L = (I - W)^T(I - W)$.

The solution for Y can have an arbitrary origin and orientation. In order to make the problem well-posed, these two degrees of freedom must be removed. Requiring the coordinates to be centered on the origin ($\sum_i y_i = 0$), and constraining the embedding vectors to have unit covariance ($Y^T Y = I$), removes the first and second degrees of freedom respectively.

The cost function can be optimized initially by the second of these two constraints. Under this constraint, the cost is minimized when the columns of Y^T (rows of Y) are the eigenvectors associated with the lowest eigenvalues of L.

Discarding the eigenvector associated with eigenvalue 0 satisfies the first constraint.

1.4 Laplacian Eigenmaps

A closely related approach to locally linear embedding is Laplacian eigenmaps (See Figure 1.5 for an example). Given t points in n-dimensional space, the Laplacian eigenmaps Method (LEM) [1] starts by constructing a weighted graph with t nodes and a set of edges connecting neighbouring points. Similar to LLE, the neighbourhood graph can be constructed by finding the k nearest neighbours, or by choosing all points within some fixed radius ϵ. For weighting the edges, there are two variations: either each edge is weighted by $W_{ij} = e^{-\frac{||x_i - x_j||^2}{s}}$, where s is a free parameter which should be chosen a priori, or simply all W_{ij} is set to 1 if vertices i and j are connected. The embedding map is then provided

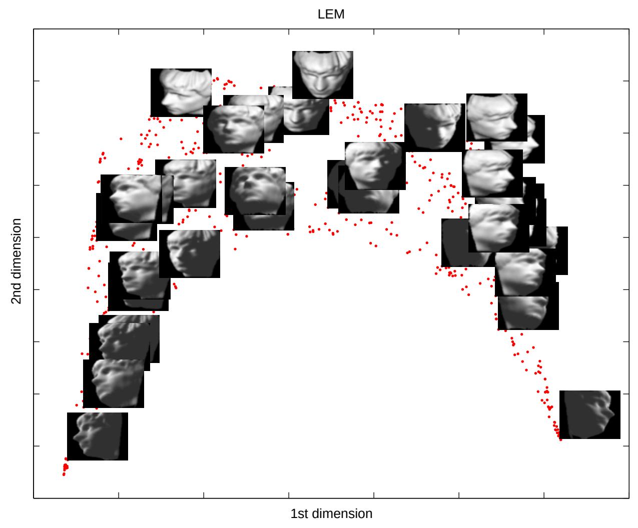

Figure 1.5: *LEM applied ($k = 7$) to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

by the following objective

$$\min_Y \sum_{i=1}^{t} \sum_{j=1}^{t} (\mathbf{y}_i - \mathbf{y}_j)^2 W_{ij}$$

subject to appropriate constraints. This objective can be reformulated as

$$\min_Y \operatorname{Tr}(YLY^T)$$

where $L = R - W$, R is diagonal, and $R_{ii} = \sum_{j=1}^{t} W_{ij}$. This L is called the *Laplacian function*. Similar to (1.3), after adding orthogonality and centering constraint, a solution to this problem can be found by making Y to be the eigenvectors of L (non-normalized solution). As an alternative, (1.3) can be constrained to $Y^TLY = I$. In this case, the solution is provided by the eigenvectors of the generalized eigenvalue problem $My = \lambda Dy$ (normalized solution). Note that the final objectives for both LEM and LLE have the same form and differ only in how the matrix L is constructed. Therefore, same closed form solution (taking Y to be the eigenvectors of L) works.

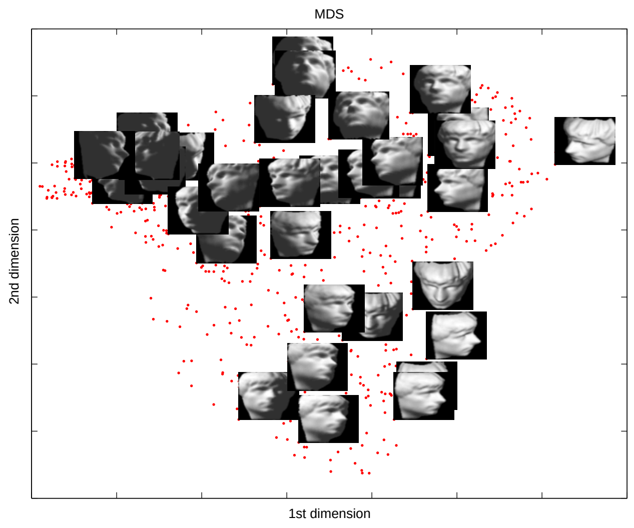

Figure 1.6: *MDS applied to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

1.5 Metric Multidimensional Scaling (MDS)

An alternative perspective on dimensionality reduction is offered by Multidimensional scaling (MDS). MDS is another classical approach that maps the original high dimensional space to a lower dimensional space, but does so in an attempt to preserve pairwise distances (See Figure 1.6 for an example). That is MDS addresses the problem of constructing a configuration of t points in Euclidean space by using information about the distances between the t patterns. Although it has a very different mathematics from PCA, it winds up being closely related, and in fact yields a linear embedding, as we will see.

A $t \times t$ matrix D is called a distance or affinity matrix if it is symmetric, $\mathtt{d}_{ii} = 0$, and $\mathtt{d}_{ij} > 0, \quad i \neq j$.

Given a distance matrix D, MDS attempts to find t data points $y_1, ..., y_t$ in d dimensions, such that if $\hat{d}_{ij}$ denotes the Euclidean distance between y_i and y_j, then $\hat{D}$ is similar to D. In particular, we consider metric MDS [3], which minimizes

$$\min_Y \sum_{i=1}^{t} \sum_{i=1}^{t} (\mathtt{d}_{ij}^{(X)} - \mathtt{d}_{ij}^{(Y)})^2 \tag{1.4}$$

where $\mathtt{d}_{ij}^{(X)} = ||x_i - x_j||^2$ and $\mathtt{d}_{ij}^{(Y)} = ||y_i - y_j||^2$. The distance matrix $D^{(X)}$ can be converted to a kernel matrix of inner products X^TX by

$$X^TX = -\frac{1}{2}HD^{(X)}H$$

where $H = I - \frac{1}{t}ee^T$ and e is a column vector of all 1's. Now (1.4) can be reduced to

$$\min_Y \sum_{i=1}^{t}\sum_{i=1}^{t}(x_i^Tx_j - y_i^Ty_i)^2$$

It can be shown [3] that the solution is $Y = \Lambda^{1/2}V^T$ where V is the eigenvectors of X^TX corresponding to the top d eigenvalues, and Λ is the top d eigenvalues of X^TX. Clearly the solution for MDS is identical to dual PCA (see Table 1.2), and as far as Euclidean distance is concerned, MDS and PCA produce the same results. However, the distances need not be based on Euclidean distances and can represent many types of dissimilarities between objects.

1.6 Isomap

Similar to PCA, MDS has been recently extended to perform nonlinear dimensionality reduction. A recent approach to nonlinear dimensionality reduction based on MDS is the Isomap algorithm (See Figure 1.7 for an example). Unlike the linear case, nonlinear forms of MDS are different from nonlinear forms of PCA—a fact I exploit in Chapter 2 below.

Isomap is a nonlinear generalization of classical MDS. The main idea is to perform MDS, not in the input space, but in the geodesic space of the nonlinear data manifold. The geodesic distances represent the shortest paths along the curved surface of the manifold measured as if the surface were flat. This can be approximated by a sequence of short steps between neighbouring sample points. Isomap then applies MDS to the geodesic rather than straight line distances to find a low-dimensional mapping that preserves these pairwise distances.

Like LLE, the Isomap algorithm proceeds in three steps:

1. Find the neighbours of each data point in high-dimensional data space.

2. Compute the geodesic pairwise distances between all points.

3. Embed the data via MDS so as to preserve these distances.

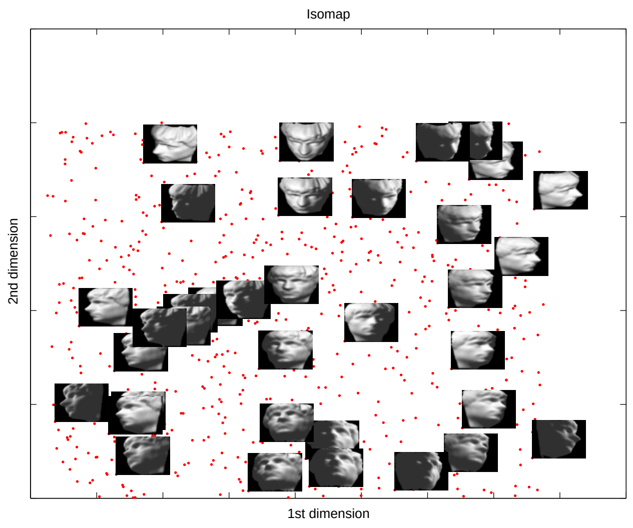

Figure 1.7: *Isomap applied* ($k = 6$) *to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

Again like LLE, the first step can be performed by identifying the k nearest neighbours, or by choosing all points within some fixed radius, ϵ. These neighbourhood relations are represented by a graph G in which each data point is connected to its nearest neighbours, with edges of weight $d_X(i, j)$ between neighbours.

The geodesic distances $d_M(i, j)$ between all pairs of points on the manifold M are then estimated in the second step. Isomap approximates $d_M(i, j)$ as the shortest path distance $d_G(i, j)$ in the graph G. This can be done in different ways including Dijkstra's algorithm [16] and Floyd's algorithm [9].

These algorithms find matrix of graph distances $D^{(\mathcal{G})}$ contains the shortest path distance between all pairs of points in G. In its final step, Isomap applies classical MDS to $D^{(\mathcal{G})}$ to generate an embedding of the data in a d-dimensional Euclidean space Y. The global minimum of the cost function is obtained by setting the coordinates of y_i to the top d eigenvectors of the inner-product matrix B obtained from $D^{(\mathcal{G})}$

1.7 Semidefinite Embedding (SDE)

In 2004, Weinberger and Saul introduced semidefinite embedding (SDE) [21, 20] (See Figure 1.8 for an example). SDE can be seen as a variation on kernel PCA, in which the kernel matrix is also learned from the data. This is in contrast with classical kernel PCA which chooses a kernel function a priori. To derive SDE, Weinberger and Saul formulated the problem of learning the kernel matrix as an instance of semidefinite programming. Since the kernel matrix K represents inner products of vectors in a Hilbert space it must be positive semidefinite. Also the kernel should be centered, *i.e.*, $\sum_{ij} K_{ij} = 0$. Finally, SDE imposes constraints on the kernel matrix to ensure that the distances and angles between points and their neighbours are preserved under the neighbourhood graph η. That is, if both x_i and x_j are neighbours (*i.e.*,$\eta_{ij} = 1$) or are common neighbours of another input (*i.e.*, $[\eta^T \eta]_{ij} > 0$), then the distance should be preserved

$$||\Phi(x_i) - \Phi(x_j)||^2 = ||x_i - x_j||^2.$$

In terms of the kernel matrix, this constraint can be written as:

$$K_{ij} - 2K_{ij} + K_{jj} = ||x_i - x_j||^2.$$

By adding an objective function to maximize $\mathrm{Tr}(K)$ which represents the variance of the data points in the learned feature space, SDE constructs a semidefinite program for learning the kernel matrix K. The last detail of SDE is the construction of the neighbourhood graph η_{ij}. This graph is constructed by connecting the k nearest neighbours using a similarity function over the data, $||x_i - x_j||$. In its last step, SDE runs kernel PCA on learned kernel K. The algorithm is summarized in Algorithm SDE (Table 1.3).

```
Algorithm: SDE
Construct neighbours, $\eta$, using $k$-nearest neighbours.
Maximize $\mathrm{Tr}(K)$ subject to $K \succeq 0$, $\sum_{ij} K_{ij} = 0$, and
 $\forall ij$   $\eta_{ij} > 0 \vee [\eta^T \eta]_{ij} > 0 \Rightarrow$
        $K_{ii} - 2K_{ij} + K_{jj} = ||x_i - x_j||^2$
Run Kernel PCA with learned kernel, $K$.
```

Table 1.3: *SDE Algorithm*

1.8 Unified Framework

All of the algorithms presented above can be cast as kernel PCA, which I now show. Although this is obvious in some cases, it is less obvious for MDS, Isomap, LLE and Laplacian eigenmaps.

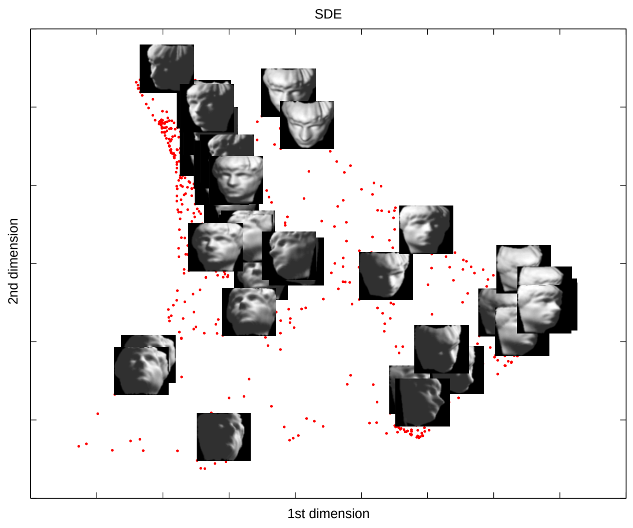

Figure 1.8: *SDE applied* ($k = 5$) *to the same data set. A two-dimensional projection is shown, with a sample of the original input images.*

A straightforward connection between LLE and Kernel PCA has been shown in [15] and [23]. Let λ_{max} be the largest eigenvalue of $L = (I - W)^T(I - W)$. Then define the LLE kernel to be:

$$K_{LLE} = \lambda_{max} I - L \tag{1.5}$$

This kernel is, in fact, a similarity measure based on the similarity of the weights required to reconstruct two patterns in terms of k neighbouring patterns. The leading eigenvector of K_{LLE} is e, and the eigenvectors $2, \ldots, d + 1$ provide the LLE embedding.

An alternative interpretation of LLE as a specific form of Kernel PCA has been discussed in [5] in details. Based on this discussion, performing Kernel PCA on pseudo-inverse $L^{\dagger}$ is equivalent to LLE up to scaling factors.

$$K_{LLE} = L^{\dagger} \tag{1.6}$$

Similarly Laplacian eigenmaps can be cast as Kernel PCA [5] by defining K_{LEM} as:

$$K_{LEM} = L^{\dagger} \tag{1.7}$$

where $L = R - W$, R is diagonal, and $R_{ii} = \sum_{j=1}^{t} W_{ij}$, as discussed in Section 1.4. K_{LEM} here is related to commute times of diffusion on the underlying graph.

It has been also shown [22] that metric MDS can be interpreted as kernel PCA. Given a distance matrix D, one can define K_{MDS} as:

$$K_{MDS} = -\frac{1}{2}(I - ee^T)D(I - ee^T) \tag{1.8}$$

where e is a column vector of all ones.

In the same fashion, given the geodesic distance $D^{(\mathcal{G})}$ used in Isomap, K_{Isomap} can be defined as [5]:

$$K_{Isomap} = -\frac{1}{2}(I - ee^T)D^{(\mathcal{G})}(I - ee^T) \tag{1.9}$$

The eigenvectors of (1.8) and (1.9) yield solutions identical to MDS and Isomap, up to scaling factor $\sqrt{\lambda_p}$, where λ_p is the p-th eigenvector.

The connection between kernel PCA and SDE is even more obvious. In fact, SDE is an instance of kernel PCA and the only difference is that SDE learns a kernel from data which is suitable for manifold discovery, while classical kernel PCA chose a kernel function a priori.

Bibliography

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[2] V. Cherkassky and F. Mulier. *Learning from data.* Wiley, New York, 1998.

[3] T. Cox and M. Cox. *Multidimensional Scaling.* Chapman Hall, Boca Raton, 2nd edition, 2001.

[4] B. Frey. *Graphical models for machine learning and digital communication.* MIT Press, Cambridge, Mass, 1998.

[5] J. Ham, D. Lee, S. Mika, and Schölkopf B. A kernel view of the dimensionality reduction of manifolds. In *International Conference on Machine Learning*, 2004.

[6] H. Hotelling. Analysis of a complex of statistical variables into components. *J. of Educational Psychology*, 24:417–441, 1933.

[7] A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128, 1999.

[8] I. Jolliffe. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[9] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing.* Benjamin, Cummings, 1994.

[10] S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Proceedings NIPS 11.* MIT Press, 1999.

[11] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, Sixth Series 2:559–572, 1901.

[12] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[13] L. Saul and S. Roweis. Think globally, fit locally: Unsupervised learning of nonlinear manifolds. *JMLR*, 2003.

[14] B. Schölkopf, S. Mika, A. Smola, G. Rätsch, and K.-R. Müller. Kernel PCA pattern reconstruction *via* approximate pre-images. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, pages 147–152, 1998.

[15] B. Schölkopf and A. Smola. *Learning with Kernels.* MIT Press, Cambridge, Massachusetts, 2002.

[16] R. Rivest T. Cormen, C. Leiserson and C. Stein. *Introduction to algorithms.* MIT Press, Cambridge, Massachusetts, 2001.

[17] J. Friedman T. Hastie, R. Tibshirani. *The elements of statistical learning.* Springer, New York, 2002.

[18] J. Tenenbaum. Mapping a manifold of perceptual observations. In *Advances in Neural Information Processing Systems 10*, pages 682–687, 1998.

[19] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[20] K. Weinberger and L. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the International Conference on Machine Learning*, pages 839–846, 2004.

[21] K. Weinberger and L. Saul. Unsupervised learning of image manifolds by semidefinite programing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 988–995, 2004.

[22] C. K. I. Williams. On a connection between kernel PCA and metric multidimensional scaling. *Machine Learning*, 46(1-3):11–19, 2002.

[23] P. Vincent Y. Bengio and J.-F. Paiement. Learning eigenfunctions of similarity: Linking spectral clustering and kernel pca. Technical Report 1232, Universite de Montreal, 2003.