Linear and Nonlinear Data Dimensionality Reduction

David Gering

April 17, 2002

Abstract

This report discusses one paper for linear data dimensionality reduction, Eigenfaces, and two recently developed nonlinear techniques. The first nonlinear method, Locally Linear Embedding (LLE), maps the input data points to a single global coordinate system of lower dimension in a manner that preserves the relationships between neighboring points. The second method, Isomap, computes geodesic distances along a manifold as sequences of hops between neighboring points, and then applies Multidimensional Scaling (MDS) to these geodesic distances instead of Euclidean distances. To provide depth of understanding as well as background for comparison, the classical linear techniques MDS and Principle Component Analysis (PCA) are derived from three different approaches. The algorithmic, applicability, and implementation issues of the three papers are discussed in the common framework of data dimensionality reduction. Simple experimental results and suggestions for improvement are presented.

1 Introduction

Many problems in information processing involve some form of dimensionality reduction. Data dimensionality reduction is a significant problem across a wide variety of fields and thus garners broad interest. It appears in many forms such as characterizing variability, discarding what is unimportant, and discovering compact representations. In this report, we specifically consider the problem of characterizing the variability of images. We regard an image as a set of pixels that specify the Cartesian coordinates of a point with respect to a set of axes – one axis per pixel. In this interpretation, each image can be thought of as a point in an abstract space of images. A set of N images represents a cloud of N points in *image space*.

Data dimensionality reduction refers to the process of deriving a set of degrees of freedom which may be adjusted to reproduce much of the variability observed within a training set. (Informally, imagine creating a small set of knobs which may be turned to generate reconstructions of all the image instances.) For example, faces, being similar in overall configuration, will not be randomly distributed in a huge image space, and thus can be described by a relatively low dimensional subspace. One example presented in [Tenenbaum00] is the rotation of a face through a number of views to produce a set of images. Since only one degree of freedom, the curve's parameterization, is being altered, the images lie along a continuous curve through image space. However, the one-dimensional curve is embedded in a high-dimensional image space, equal to the number of pixels per image. The curve is a simple example of a nonlinear manifold, which is a surface where the linear combinations of two points on the surface produce points off the surface.

The recent papers addressed in this report present unsupervised manifold learning algorithms where the problem is one of inducing a smooth nonlinear constraint manifold from a set of

examples from the manifold. Unlike PCA and MDS which are restricted to learning only linear manifolds, they attempt to discover the underlying structure of nonlinear manifolds in order to map a given data set of high-dimensional points into a surrogate low-dimensional space:

$$\mathbf{X} \in \Re^D \Rightarrow \mathbf{Y} \in \Re^d, \quad d << D$$

This report consists of a section for mathematical background followed by three parts, one for each paper, where the ideas, algorithms, and experimental results are discussed for comparison.

2 Background on PCA and MDS

2.1 Principle Component Analysis

Principle Component Analysis (PCA) replaces the original variables of a data set with a smaller number of uncorrelated variables called the *principle components*. If the original data set of dimension D contains highly correlated variables, then there is an effective dimensionality, d < D, that explains most of the data. The presence of only a few components of d makes it easier to label each dimension with an intuitive meaning. Furthermore, it is more efficient to operate on fewer variables in subsequent analysis. The method is linear in that the new variables are a linear combination of the original. No assumption is made about the probability distribution of the original variables.

The earliest descriptions of the technique were presented in [Pearson1901] and [Hotelling33]. To facilitate comparisons with the three different papers later, this section will present three different approaches to deriving PCA: those of Pearson, Hotelling, and myself.

2.1.1 Approach 1: Least Squares Distance

[Pearson1901] first introduced PCA from the standpoint of finding points and lines which best fit a set of points, and this approach is reviewed in [Duda01]. Suppose we have a set of N D-dimensional sample vectors, $\mathbf{x}_1$, $\mathbf{x}_2$,…, $\mathbf{x}_n$. The question of how to express them as d-dimensional data, where d < D, may be answered by first examining how to approximate them as 0-dimensional and 1-dimensional data. A 0-dimensional data set contains no axes of variation, so we need only compute a single vector, $\mathbf{x}_0$, such that the sum of squared distances between $\mathbf{x}_0$ and each $\mathbf{x}_k$ is as small as possible. We minimize the squared-error criterion:

$$J_0(\mathbf{x}_0) = \sum_{k=1}^{N} \left\| \mathbf{x}_0 - \mathbf{x}_k \right\|^2 \tag{1}$$

The value of $\mathbf{x_0}$ that minimizes J_0 is the sample mean:

$$\mathbf{m} = \frac{1}{N} \sum_{k=1}^{N} \mathbf{x}_k \tag{2}$$

In a similar manner, consider how to reveal something about the variability of the data by computing a 1-dimensional representation. Beginning with the sample mean, **m**, we will add one variable which will assume a value for each data point. We need to determine two items: the value of the variable for each data point, and the coordinate axis along which the value varies. Mathematically, **x** will have a representation:

$$\mathbf{x} = \mathbf{m} + y_k \mathbf{u} \tag{3}$$

The coefficients, y_k, each express the k^{th} point's distance from the sample mean **m** in the direction of unit vector **u**. These coefficients can be found by minimizing the squared-error criterion:

$$J_1(y_k) = \sum_{k=1}^{N} \left\| (\mathbf{m} + y_k \mathbf{u}) - \mathbf{x}_k \right\|^2 = \sum_{k=1}^{N} \left\| y_k \mathbf{u} - (\mathbf{x}_k - \mathbf{m}) \right\|^2 \quad (4)$$

Expand the norm by using the relation that $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$, and $\mathbf{u}^T\mathbf{u} = 1$ since **u** is a unit vector:

$$\begin{aligned} J_1(y_k) &= \sum_{k=1}^{N} (y_k \mathbf{u} - (\mathbf{x}_k - \mathbf{m}))^T (y_k \mathbf{u} - (\mathbf{x}_k - \mathbf{m})) \\ &= \sum_{k=1}^{N} [y_k^2 \mathbf{u}^T \mathbf{u} - 2 y_k \mathbf{u}^T (\mathbf{x}_k - \mathbf{m}) + (\mathbf{x}_k - \mathbf{m})^T (\mathbf{x}_k - \mathbf{m})] \\ &= \sum_{k=1}^{N} y_k^2 - 2 \sum_{k=1}^{N} y_k [\mathbf{u}^T (\mathbf{x}_k - \mathbf{m})] + \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{m} \right\|^2 \end{aligned} \quad (5)$$

Since we are projecting the data onto the space spanned by a single line, the coefficients that minimize J_1 are the projection of $\mathbf{x}_k$ onto the line in the direction of **u** that passes through **m**:

$$y_k = \mathbf{u}^T (\mathbf{x}_k - \mathbf{m}) \quad (6)$$

Substitute (6) for y_k in (5) to find the best direction for **u** to minimize the squared-error.

$$\begin{aligned} J_1(y_k) &= \sum_{k=1}^{N} y_k^2 - 2 \sum_{k=1}^{N} y_k [y_k] + \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{m} \right\|^2 \\ &= -\sum_{k=1}^{N} y_k^2 + \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{m} \right\|^2 \\ &= -\sum_{k=1}^{N} [\mathbf{u}^T (\mathbf{x}_k - \mathbf{m})]^2 + \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{m} \right\|^2 \end{aligned} \quad (7)$$

Expand the quadratric by using the relations that $\mathbf{x}^2 = \mathbf{x}\mathbf{x}^T$, and $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$:

$$\begin{aligned} J_1(\mathbf{u}) &= -\sum_{k=1}^{N} [\mathbf{u}^T (\mathbf{x}_k - \mathbf{m})][\mathbf{u}^T (\mathbf{x}_k - \mathbf{m})]^T + \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{m} \right\|^2 \\ &= -\sum_{k=1}^{N} \mathbf{u}^T (\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^T \mathbf{u} + \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{m} \right\|^2 \end{aligned} \quad (8)$$

Substitute the formula for the scatter matrix, **S**, which is (N-1) times the sample covariance matrix, **Σ**:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \cdots & \sigma_{1N}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & & \sigma_{2N}^2 \\ \vdots & & \ddots & \\ \sigma_{N1}^2 & \sigma_{N2}^2 & & \sigma_{NN}^2 \end{bmatrix} \tag{9}$$

$$\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{k=1}^{N}(\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^T$$

$$\mathbf{S} = \sum_{k=1}^{n}(\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^T$$

$$J_1(\mathbf{u}) = -\mathbf{u}^T\mathbf{S}\mathbf{u} + \sum_{k=1}^{N}\left\|\mathbf{x}_k - \mathbf{m}\right\|^2 \tag{10}$$

The second term on the right-hand side is not a function of **u**, so J_1 will be minimized when $\mathbf{u}^T\mathbf{S}\mathbf{u}$ is maximized. Since this term grows larger with larger lengths of **u**, we cannot maximize it without imposing some constraint to make the problem well posed. In this case, the constraint is obvious because we want **u** to be a unit vector. Therefore, $||\mathbf{u}|| = \mathbf{u}^T\mathbf{u} = 1$.

Maximization problems with constraints are facilitated by the method of Lagrange multipliers. Given a function, f(**x**) and a constraint, g(**x**) = c for some constant c, we can maximize the function L(**x**) containing an arbitrary constant muliplier, λ, attached to the constraint term, (g(**x**) - c), which is 0:

$$\begin{aligned} L(\mathbf{x}) &= f(\mathbf{x}) \\ &= f(\mathbf{x}) + \lambda(g(\mathbf{x}) - c) \\ \frac{\partial L(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial f}{\partial \mathbf{x}} + \lambda\frac{\partial g}{\partial \mathbf{x}} = 0 \end{aligned} \tag{11}$$

In our case:

$$L(\mathbf{u}) = \mathbf{u}^t\mathbf{S}\mathbf{u} - \lambda(\mathbf{u}^T\mathbf{u} - 1) \tag{12}$$

To maximize, set the gradient vector equal to 0, and compute the derivatives using the following identities which can be verified by writing out the components [Duda01]. Given independent vectors **f** and **g**, and symmetric matrix **H**:

$$\begin{aligned} \frac{\partial(\mathbf{f}^T\mathbf{g})}{\partial \mathbf{f}} &= \mathbf{g} \\ \frac{\partial(\mathbf{f}^T\mathbf{H}\mathbf{f})}{\partial \mathbf{f}} &= 2\mathbf{H}\mathbf{f} \end{aligned} \tag{13}$$

Therefore:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{u}} &= 2\mathbf{S}\mathbf{u} - 2\lambda\mathbf{u} = 0 \\ \mathbf{S}\mathbf{u} &= \lambda\mathbf{u} \end{aligned} \tag{14}$$

Inspection of the form of the solution above reveals that **u** must be an eigenvector of **S**. That is, if **u** is an input vector to be transformed by the matrix **S**, then the output is a scaled vector in the same direction as the input. Substitute (14) into the objective function to realize that to maximize the term, we must choose the eigenvector associated with the *largest* eigenvalue:

$$\begin{aligned} \max \mathbf{u}^T \mathbf{S}\mathbf{u} &= \max \mathbf{u}^T (\lambda \mathbf{u}) \\ &= \max \lambda \end{aligned} \tag{15}$$

To summarize this result, we have found that from a standpoint of least squares, the data can best be represented in 1-dimension if the coordinate axis is the eigenvector of the scatter matrix corresponding with the largest eigenvalue. Similarly, representing the data in d-dimensions involves projecting it onto d lines. The derivation follows directly from replacing equations (3) and (4) with those below.

$$\mathbf{x} = \mathbf{m} + \sum_{i=1}^{d} y_d \mathbf{u}_d \tag{16}$$

$$J_d = \sum_{k=1}^{n} \left\| (\mathbf{m} + \sum_{i=1}^{d} y_{ki} \mathbf{u}_i) - \mathbf{x}_k \right\|^2 \tag{17}$$

Therefore, we have shown that D-dimensional data is best represented in d-dimensions when the d coordinate axes are the eigenvectors corresponding to the d largest eigenvalues of the scatter matrix **S**. Since **S** is real and symmetric, $\{\mathbf{u}_i\}$ are orthogonal and comprise a natural set of basis vectors as the principle axes. Their coefficients, y_i in equation 16, are subsequently called the principle components such that $\mathbf{y}_k$ represents the d-dimensional version of the D-dimensional random vector, $\mathbf{x}_k$.

2.1.2 Approach 2: Change of Variables

[Hotelling33] introduced PCA from a different standpoint which is reviewed in several textbooks such as [Chatfield80,Jolliffe86,Johnson92,Harris01]. PCA transforms a set of correlated variables to a new set of uncorrelated variables that are ordered by decreasing variance.

Suppose we are given a D-dimensional random variable **x** with mean **m** and DxD covariance matrix **Σ**. Find a new vector, **y**, of d random variables which are uncorrelated and whose variances are in descending order. Each y_k is a linear combination of the x_k's, where the constant multipliers are denoted by u_{ij}. This produces the linear system of equations:

$$y_k = u_{1k} x_1 + u_{2k} x_2 + \cdots + u_{Dk} x_D = \mathbf{u}_k^T \mathbf{x} \tag{18}$$

The complete problem of finding all d new variables can be written in matrix notation for the dx1 column vector, **y,** the Dx1 column vector, **x**, and the Dxd coefficient matrix, **U,** containing the d $\mathbf{u}_k$ as its columns, as:

$$\mathbf{y} = \mathbf{U}^T \mathbf{x} \tag{19}$$

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix}_{dx1} = \left(\begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \\ | & | & & | \end{bmatrix}_{Dxd} \right)^T \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix}_{Dx1}$$

Since equation 18 exhibits an arbitrary scale factor, impose the constraint that $\mathbf{u}_k$ is a unit vector. This obtains an orthogonal transformation (rotation and reflection only) that preserves distances.

$$\mathbf{u}_k^T \mathbf{u}_k = \sum_{k=1}^{D} u_{kj}^2 = 1 \tag{20}$$

The first *principle component*, y_1, is found by choosing the *principle axis*, $\mathbf{u}_1$, so that y_1 has the largest possible variance. That is, choose $\mathbf{u}_1$ to maximize the variance of $\mathbf{u}_1^T\mathbf{x}$ subject to the constraint $\mathbf{u}_1^T\mathbf{u}_1=1$. Similarly, y_2 is found by choosing $\mathbf{u}_2$ such that y_2 has the largest possible variance while also being uncorrelated with y_1. More generally:

k^{TH} principle component = linear combination $\mathbf{u}_k^T\mathbf{x}$
that maximizes $\mathrm{Var}(\mathbf{u}_k^T\mathbf{x})$
subject to $\mathbf{u}^T\mathbf{u}=1$ and $\mathrm{Cov}(\mathbf{u}_k\mathbf{x}, \mathbf{u}_j\mathbf{x}) = 0$ for $k < j$

Where:

$$\begin{aligned} Var(y_k) &= Var(\mathbf{u}_k^T\mathbf{x}) \\ &= E[(\mathbf{u}^T(\mathbf{x}-\mathbf{m}))^2] \\ &= E[(\mathbf{u}^T(\mathbf{x}-\mathbf{m}))(\mathbf{u}^T(\mathbf{x}-\mathbf{m}))^T] \\ &= E[\mathbf{u}^T(\mathbf{x}-\mathbf{m})(\mathbf{x}-\mathbf{m})^T\mathbf{u}^T] \\ &= \mathbf{u}^T(E[(\mathbf{x}-\mathbf{m})(\mathbf{x}-\mathbf{m})^T])\mathbf{u}^T \\ &= \mathbf{u}_k^T\boldsymbol{\Sigma}\mathbf{u}_k \\ Cov(y_k, y_j) &= \mathbf{u}_k^T\boldsymbol{\Sigma}\mathbf{u}_j \end{aligned} \tag{21}$$

The maximization of variance can be performed using Lagrange multipliers where equation 22 for y_1 has identical form as equation 12 in our first approach. Therefore, y_1 is computed by setting the principle axis, $\mathbf{u}_1$, to be the eigenvector of $\boldsymbol{\Sigma}$ associated with the largest eigenvalue.

For y_2, we have the additional constraint of zero correlation with y_1, which we accommodate with a second Lagrange multiplier, δ:

$$\begin{aligned} L(y_1) &= \mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{u}_1 - \lambda(\mathbf{u}_1^T\mathbf{u}_1 - 1) \\ L(y_2) &= \mathbf{u}_2^T\boldsymbol{\Sigma}\mathbf{u}_2 - \lambda(\mathbf{u}_2^T\mathbf{u}_2 - 1) - \delta(\mathbf{u}_2^T\mathbf{u}_1 - 0) \end{aligned} \tag{22}$$

The solutions are the same as we saw for equation 12 because δ must be 0. To see this requirement, pre-multiply each side of the zero-gradient condition by $\mathbf{u}_1^T$:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{u}_2} &= 2\boldsymbol{\Sigma}\mathbf{u}_2 - 2\lambda\mathbf{u}_2 - \delta\mathbf{u}_1 \\ 0 &= 2(\mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{u}_2) - 2\lambda(\mathbf{u}_1^T\mathbf{u}_2) - \delta(\mathbf{u}_1^T\mathbf{u}_1) \\ 0 &= 2(0) - 2\lambda(0) - \delta(1) \\ 0 &= \delta \end{aligned} \tag{23}$$

2.1.3 Approach 3: Matrix Factorization for Variation Compression

Finally, I present perhaps the most intuitive derivation of PCA to form a clear picture of why the eigenvectors are required. We approach from the perspective of matrix factorization for the purpose of compressing the representation of image variation.

Begin with the definition of an eigenvector, **u**, of a symmetric scatter matrix **S**:

$$\mathbf{Su} = \lambda\mathbf{u} \tag{24}$$

Define **U** to be the matrix of columns of all D orthogonal eigenvectors of **S** scaled to be orthonormal, and **Λ** is the diagonal matrix of eigenvalues. Then it follows that:

$$\mathbf{S}_{NxN}\mathbf{U}_{NxD} = \mathbf{U}_{NxD}\mathbf{\Lambda}_{DxD} \tag{25}$$

$$\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & & s_{1N} \\ \vdots & & \ddots & \\ s_{N1} & s_{N2} & & s_{NN} \end{bmatrix} \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_D \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_D \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & \lambda_D \end{bmatrix}$$

Since **U** is an orthonormal matrix, its inverse is its transpose, which allows to easily complete the factorization of **S** below. This is the known as the *Singular Value Decomposition* for symmetric matrices, or the *diagonalization* of **S** [Strang00]. One way to conceptualize this is to notice that all off-diagonal terms of **Λ** are 0, so there are no correlations between variables. Given that the matrix **U** is orthonormal, and orthonormal transforms perform only rotation, **U** rotated **S** so that its coordinate axes align with the principle component axes. Thus, the new axes represent the directions of maximum variability.

$$\begin{aligned} \mathbf{SU} &= \mathbf{U\Lambda} \\ \mathbf{SUU}^{-1} &= \mathbf{SU\Lambda U}^{-1} \\ \mathbf{S} &= \mathbf{U\Lambda U}^{T} \\ &= \lambda_1\mathbf{u}_1\mathbf{u}_1^T + \lambda_2\mathbf{u}_2\mathbf{u}_2^T + \ldots + \lambda_n\mathbf{u}_D\mathbf{u}_D^T \\ &= \lambda_1\mathbf{P}_1 + \lambda_2\mathbf{P}_2 + \ldots + \lambda_D\mathbf{P}_D \end{aligned} \tag{26}$$

Since the outer products $\mathbf{u}_k\mathbf{u}_k^T$ are projection matrices, **A** is the linear combination of ordered projection matrices, $\mathbf{P}_k$. Consequently, the amount of variance coverage by the first d of D projections can be expressed as:

$$\sum_{i=1}^{d}\lambda_i \Big/ \sum_{i=1}^{D}\lambda_i \tag{27}$$

Now, consider the problem of performing a change of basis for the purpose of image compression. An example zero-mean, D-dimensional image, **x**, (vector of length D) begins in the standard basis which is the identity matrix, **I**. That is, the columns of **I** ($\mathbf{I}_1$, $\mathbf{I}_2$, …$\mathbf{I}_D$) are the basis vectors that span the space of all possible images. Any example image, **x**, is a linear combination of the independent vectors in the standard basis. The D coefficients, **z**, are the image pixel values. Storing a representation of the image in the standard space involves storing the vector **z** of all these coefficients.

$$\begin{aligned}\mathbf{x} &= \mathbf{I}\mathbf{z} \\ &= z_1\mathbf{I}_1 + z_2\mathbf{I}_2 + \ldots + z_D\mathbf{I}_D \\ &= \begin{bmatrix} | & | & & | \\ \mathbf{I}_1 & \mathbf{I}_2 & \ldots & \mathbf{I}_D \\ | & | & & | \end{bmatrix}\mathbf{z}\end{aligned} \tag{28}$$

To store a compressed representation of the image **x**, we could store only d coefficients, where d << D. However, the fraction (D-d)/D of the image would be completely dark. Therefore, we first need to represent the image **x** in a new basis that has better properties for compression. For example, consider the following wavelet basis, **W**, instead of **I**. Like the standard basis, these new basis vectors are orthogonal ($\mathbf{w}_i^T\mathbf{w}_j = 0$). The difference is that this new basis is better for compression because it is possible to represent some images with few basis vectors. For example, a flat image could be represented entirely by using the low-frequency basis vector, $\mathbf{w}_1$.

$$\mathbf{W} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 \\ | & | & | \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \end{bmatrix} \tag{29}$$

The most ideal representation for compression would be for the basis vectors to be the eigenvectors because of the diagonalization explained above. Representing an example 0-mean image, **x**, in this eigenspace, corresponds to storing its vector of coefficients, **y**, or principle components of the principle axes in **U**. Compare equation 30 with 28.

$$\begin{aligned}\mathbf{x} &= \mathbf{U}\mathbf{y} \\ &= y_1\mathbf{u}_1 + y_2\mathbf{u}_2 + \ldots + y_D\mathbf{u}_D \\ &= \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_D \\ | & | & & | \end{bmatrix}\mathbf{y}\end{aligned} \tag{30}$$

Compress the image by storing only its d most principle eigenvectors in a smaller basis matrix, $\mathbf{U}_d$. To perform compression by computing the coefficients, **y**, perform the inverse of the image reconstruction operation:

$$\begin{aligned}\tilde{\mathbf{x}} &= \mathbf{U}_d\mathbf{y} \\ \mathbf{U}_d{}^{-1}\tilde{\mathbf{x}} &= \mathbf{U}_d{}^{-1}\mathbf{U}_d\mathbf{y} \\ \mathbf{y} &= \mathbf{U}_d^T\tilde{\mathbf{x}}\end{aligned} \tag{31}$$

Thus, compression and reconstruction can each be performed with a single matrix multiplication. Compare this result with the standard basis, where the representation and reconstruction coefficients are identical.

Basis	**U** (eigenvectors of scatter)	**I** (standard basis)
Compression	$\mathbf{y} = \mathbf{U}_d^T\mathbf{z}$	$\mathbf{y} = \mathbf{I}^T\mathbf{z}$
Reconstruction	$\tilde{\mathbf{z}} = \mathbf{U}_d\mathbf{y}$	$\mathbf{z} = \mathbf{I}\mathbf{y}$
Compare	$\mathbf{z} \neq \mathbf{y}$	$\mathbf{z} = \mathbf{y}$

2.2 Classical Multidimensional Scaling

Consider starting with a set of data points and computing their squared pairwise distances from their coordinates. MDS addresses the inverse problem of finding a set of coordinate values from a set of squared distances. Preferably, the new coordinates will have a very small number of dimensions. (For example, convert the set of distances of N MIT classrooms (a symmetric N*x*N matrix of distance pairs) to a 2-dimensional campus map.) Note that the output map will have arbitrary location and orientation, so we will constrain its center of gravity to be at the origin, and its orientation to align with principle axes, as derived below.

When the magnitude of the distance matters (classical scaling instead of ordinal scaling) and the distances are Euclidean, MDS is equivalent to PCA [Torgerson52,Mardia79]. While there exists a wide assortment of distance functions in use [Cox01], we will work only with Euclidean distances for the remainder of this discussion.

Given a zero-mean, DxN data matrix, **X**, of D-dimensional vectors of N images, the set of squared distances can be computed from the inner product of the difference between two columns (such as individuals r and s):

$$\begin{aligned} d_{rs}^2 &= \sum_{i=1}^{D} (\mathbf{x}_{ir} - \mathbf{x}_{is})^2 \\ &= (\mathbf{x}_r - \mathbf{x}_s)^T (\mathbf{x}_r - \mathbf{x}_s) \\ &= \mathbf{x}_r^T \mathbf{x}_r - 2\mathbf{x}_s^T \mathbf{x}_r + \mathbf{x}_s^T \mathbf{x}_s \\ &= b_{rr} - 2b_{rs} + b_{ss} \end{aligned} \tag{32}$$

Let the NxN matrix, **B**, be the inner product matrix of all the between-individual inner products:

$$\begin{aligned} \mathbf{B} &= \mathbf{X}^T \mathbf{X} \\ b_{rs} &= \mathbf{x}_r^T \mathbf{x}_s \end{aligned} \tag{33}$$

While with PCA, the input data is always in the form, **X**, with MDS, it may be given as **X** or only the NxN matrix of squared distances, $\mathbf{D} = \{d_{rs}^2\}$. In these cases, we need to invert the bottom line of equation 32 to obtain **B** from **D**. Recall that we are constraining the center of gravity of the set of pairwise distances to lie at the origin. This is referred to as *centering* the matrix **D**, and it is achieved through subtracting the average of each row from every element in its row. Also subtract the average of each column from every element in its column. Then we can solve for b_{rs} in equation 32 to obtain:

$$b_{rs} = -\frac{1}{2} d_{rs}^2 \tag{34}$$

Now that we have obtained **B** from **D**, the next step is to find the eigenvectors and eigenvalues of **B**. The MDS problem differs from PCA in that we are searching for the eigenvectors and eigenvalues of $\mathbf{B} = \mathbf{X}^T\mathbf{X}$ instead of $\mathbf{S} = \mathbf{X}\mathbf{X}^T$. For Euclidean distances, these problems are shown here to be equivalent. Let $\mathbf{u}_i$ and $\mathbf{v}_i$ be the eigenvectors of **S** and **B**, respectively. Then:

$$\begin{aligned} \mathbf{X}\mathbf{X}^T \mathbf{u}_i &= \lambda_i \mathbf{u}_i \\ \mathbf{X}^T \mathbf{X} \mathbf{v}_i &= \lambda_i \mathbf{v}_i \end{aligned} \tag{35}$$

Premultiply each side by **X**:

$$\mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{v}_i) = \lambda_i(\mathbf{X}\mathbf{v}_i) \tag{36}$$
$$\mathbf{X}\mathbf{X}^T(\mathbf{u}_i) = \lambda_i(\mathbf{u}_i)$$
$$\mathbf{u}_i = \mathbf{X}\mathbf{v}_i$$

Thus, the eigenvalues are the same, and the eigenvectors are multiplied by **X**. The difference in MDS is that we are not only solving for the $\{\mathbf{v}_i,\}$ but also **X**. With $\{\mathbf{v}_i\}$ as orthogonal unit vectors, we compute **X** such that the coordinates of the r^{TH} point are the components of $\{\mathbf{f}_i\}$, where:

$$\mathbf{f}_i = \sqrt{\lambda_i}\,\mathbf{v}_i \tag{37}$$
$$\mathbf{X} = \begin{bmatrix} - & \mathbf{f}_1 & - \\ - & \mathbf{f}_2 & - \\ & \vdots & \\ - & \mathbf{f}_d & - \end{bmatrix}_{dxN}$$

Then by orthogonality of $\{\mathbf{v}_i\}$:

$$(\mathbf{X}^T\mathbf{X})\mathbf{v}_i = \mathbf{X}\begin{bmatrix} - & \mathbf{f}_1 & - \\ - & \mathbf{f}_2 & - \\ & \vdots & \\ - & \mathbf{f}_d & - \end{bmatrix}\mathbf{v}_i = \mathbf{X}\begin{bmatrix} - & \sqrt{\lambda_1}\mathbf{v}_{1}^T\mathbf{v}_i & - \\ - & \sqrt{\lambda_2}\mathbf{v}_{2}^T\mathbf{v}_i & - \\ & \vdots & \\ - & \sqrt{\lambda_d}\mathbf{v}_{d}^T\mathbf{v}_i & - \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \mathbf{f}_1 & \mathbf{f}_2 & \cdots & \mathbf{f}_d \\ | & | & & | \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ \sqrt{\lambda_i} \\ 0 \end{bmatrix} \tag{38}$$
$$= \lambda_i\mathbf{v}_i$$

To summarize:

MDS Algorithm	
Step	**Description**
1	Either be given the set of pairwise distances, **D**, or compute it from input data.
2	Obtain inner-products **B** from double-centering **D** and applying $b_{rs} = -1/2\ d_{rs}^2$
3	Compute eigenvalues and eigenvectors of **B**
4	Factor **B** into $\mathbf{X}^1\mathbf{X}$ to obtain coordinates, **X**.

3 Eigenfaces

3.1 Idea

The central idea of the Eigenface system is to project face images onto a feature space that spans the significant variations among known face images. The significant features are known as *eigenfaces* because they are the eigenvectors, or principle axes, of the covariance matrix corresponding to the original face images. These eigenvectors define the subspace of face images which [Turk91a] names *face space*, and each vector is a linear combination of the original face images. The projection operation characterizes an individual face by a weighed sum of the eigenface features. Therefore, to recognize a particular face, the weights are compared to those of known individuals. In essence, the high-dimensional set of pixels becomes reduced to the low-dimensional set of feature weights called the principle components.

3.2 Algorithm

Let a face image be a two-dimensional array of intensity values, or pixels. Given N images, convert each image to a column vector of length D so we have a set of N D-dimensional data points:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N \tag{39}$$

The average face, **m**, is defined as the sample mean of images:

$$\mathbf{m} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i \tag{40}$$

The mean is then subtracted from each training vector to align the centroid of the distribution with the origin (so that each dimension has zero mean). Each face differs from the average by:

$$\mathbf{x}_i - \mathbf{m} \tag{41}$$

Construct the training matrix as the collection of these column vectors:

$$\mathbf{M} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 - \mathbf{m} & \mathbf{x}_2 - \mathbf{m} & \cdots & \mathbf{x}_N - \mathbf{m} \\ | & | & & | \end{bmatrix}_{DxN} \tag{42}$$

As described in section 2, we then to perform PCA to obtain a set of orthogonal vectors that optimally represent the distribution of the data, where optimal is defined in the least squares sense. However, N << D, so we prefer to avoid the computation of a DxD covariance matrix which has complexity $O(D^2)$. We especially prefer to avoid finding its eigenvectors, which as complexity $O(D^3)$. Fortunately, there is an algebraic trick we can exploit. Observe that the covariance matrix can be expressed in terms of the outer product of M with itself.

$$\begin{aligned} \mathbf{\Sigma} &= \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T \\ &= \frac{1}{N-1}\mathbf{M}\mathbf{M}^T \end{aligned} \tag{43}$$

Instead of finding the eigenvectors, $\mathbf{u}_i$, of the DxD outer product:

$$\mathbf{M}\mathbf{M}^T\mathbf{u}_i = \lambda_i\mathbf{u}_i \tag{44}$$

Consider finding the eigenvectors, $\mathbf{v}_i$, of the NxN inner:

$$\mathbf{M}^T\mathbf{M}\mathbf{v}_i = \lambda_i\mathbf{v}_i \tag{45}$$

The result of this was shown in equation 36 to be:

$$\mathbf{u}_i = \mathbf{M}\mathbf{v}_i \tag{46}$$

As explained in Chapter 2.1.3, once the eigenvalues have been computed, a new test face image is transformed into its eigenface coordinates by:

$$\mathbf{y}_i = \mathbf{U}_d^T(\mathbf{x}_i - \mathbf{m}) \tag{47}$$

And an image is reconstructed from its eigen-space representation by:

$$\tilde{\mathbf{x}} = \mathbf{U}_d \mathbf{y} + \mathbf{m} \tag{48}$$

The following table summarizes a complete algorithm for face recognition.

Eigenface Algorithm	
Step	**Description**
1	Calculate the basis from the training set images
2	Project the training images into FaceSpace
3	Project the test images into FaceSpace
4	Determine if the image is a face (by testing if it is sufficiently close to FaceSpace)
5	If it is a face, compare it to the training images

3.3 Experimental Results

I implemented the algorithm in Matlab on a database of N=85 faces of size 300x250 for a total dimensionality of D=75,000. There are three images of each volunteer with various expressions. Figure 1 illustrates the fraction of variance accounted for by the highest subsets of principle components. Figure 2 demonstrates the accuracy of reconstruction for various levels of data dimensionality reduction, where the rightmost image is a single scaled eigenface.

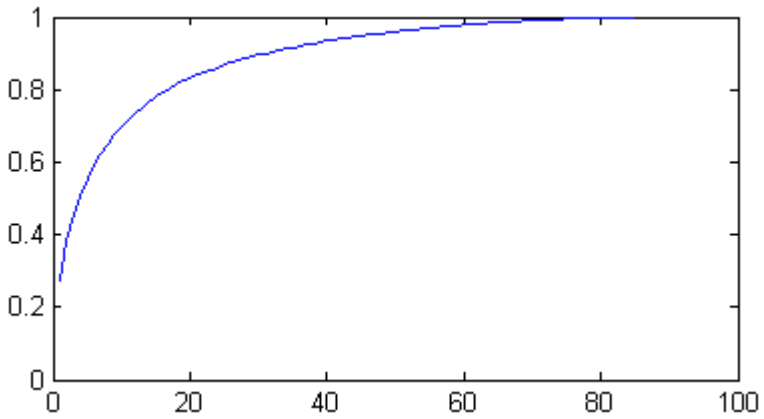

Figure 1: Variance Coverage

Fraction of variance accounted for by the highest subsets of eigenvalues plotted versus the number of eigenvalues used.

Figure 2: Reconstruction Accuracy

From left to right: original, reconstruction from 84, 40, 20, 3, 2, and 1 dimensions.

3.4 Discussion

PCA discovers the structure of data lying on or near a linear subspace of the high-dimensional input space. The method is most applicable when the $\mathbf{x}_i$ form a hyperellipsoid cloud, because then the eigenvectors of **S** form the cloud's principle axes. But the algorithm is not capable of discovering nonlinear degrees of freedom that underlie complex natural observations. Regardless, the reasons for the popularity of PCA became clear from my implementation. The algorithm is

straightforward to implement, has polynomial time complexity, converges to a global solution, and its linear nature is easily understood and visualized.

Training run time is dominated by the solution for eigenvectors, which has complexity $O(N^3)$. However, when a very small number, d, of eigenvectors are required, they can be solved for one at a time using the Power Method [Bai00] with complexity $O(N^2)$ for each eigenvector. A real strength of the algorithm is its run time for projecting novel test images, which requires only that the input vector undergo vector subtraction and matrix multiplication, which has complexity O(D+ND).

One issue with the algorithm is how well a test image is represented within eigen-space. To support the process of selecting what dimension, d, to use, equation 27 can be employed to measure the amount of variance accounted for. An advantage of the method is that it degrades gracefully under small changes.

In addition to the restriction to linear subspaces, PCA also suffers the drawbacks of a high $O(N^3)$ worst-case complexity, inability to handle missing data, and the lack of a probability model in the space of inputs. It may be considered an algorithmic strength in the interest of generality to not require assumptions regarding the statistical distributions of the data. However, it would be desired to be able to answer how well new data are fit by the model in a probabilistic sense. Instead, the only criterion available is the squared distance of the test image from its projection into eigenspace. Approaches [Roweis98] using expectation-maximization (EM) [Dempster77] have addressed all three of these drawbacks with some success.

4 Locally Linear Embedding

4.1 Idea

Consider a set of input data points of dimensionality, D, that lie on or near a smooth underlying nonlinear manifold of lower dimensionality, d. Figure 3 depicts such a situation where the 3D points form the topology of a 2D rectangular manifold bent into the shape of a 3D S-curve. The authors draw the following informal analogy that frames the motivation behind their algorithm. Imagine using a scissors to cut the manifold into small squares that represent locally linear patches of the nonlinear S-curve surface. Then position these squares onto a flat tabletop while preserving the angular relationships between neighboring squares. Note that the transplantation is a linear mapping because it involves only the operations of translation, rotation, and scaling of each patch. Thus, the algorithm identifies the data's nonlinear structure through two linear computational steps: first, compute the locally linear patches, and second, compute the linear mapping to a lower dimensional embedding, which is the coordinate system on the manifold.

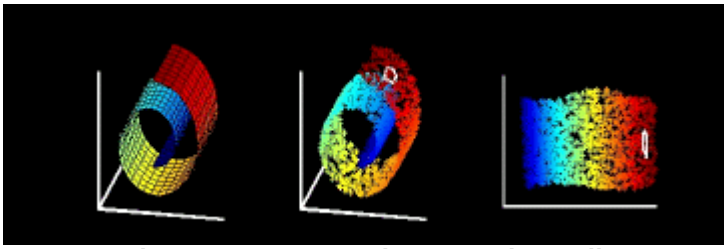

Figure 3: LLE operating on Swiss Roll

From left to right: nonlinear manifold, input data as manifold samples, mapping to low-dimensional space.

More formally, the main idea behind Locally Linear Embedding (LLE) [Roweis00] is to map the input data points to a single global coordinate system of lower dimension in such a way as to preserve the relationships between neighboring points. Each data point and its neighbors are expected to lie on, or close to, a locally-linear patch of a manifold. The intrinsic geometry of a patch can be captured by approximating each point by a linear combination of its neighbors. The coefficients for this combination are chosen to be invariant to the transplantation operations mentioned above (translation, rotation, and scaling). Therefore, the characterization of local geometry in the original high-dimensional data space will be equally valid in the lower-dimensional space. The algorithm then finds a set of low-dimensional points that can be linearly approximated by their neighbors with the same coefficients that were determined from the high-dimensional data points.

4.2 Algorithm

Given N real-valued vectors $\mathbf{x_i}$, each of dimensionality, D, compute the NxN weight matrix, $\mathbf{W}$, of linear coefficients, W_{ij}, that reconstruct each data point from its K neighbors. The weight, W_{ji}, expresses the contribution of the j^{th} data point to the reconstruction of the i^{th} data point. The reconstruction weights for each data point are computed from its local neighborhood independent of the weights for other data points. Choose W_{ij} to minimize a cost function of squared reconstruction errors:

$$J_1(\mathbf{W}) = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \sum_{j=1}^{K} W_{ji} \mathbf{x}_j \right\|^2 \tag{49}$$

LLE then constructs the neighborhood-preserving mapping by mapping each high dimensional observation, $\mathbf{x_i}$, to a low dimensional vector, $\mathbf{y_i}$, representing coordinates on the manifold. This is done by choosing the d-dimensional coordinates, $\mathbf{y_i}$, to minimize the embedding cost function by optimizing $\mathbf{y_i}$ while holding W_{ij} fixed:

$$J_2(\mathbf{Y}) = \sum_{i=1}^{N} \left\| \mathbf{y}_i - \sum_{j=1}^{N} W_{ji} \mathbf{y}_j \right\|^2 \tag{50}$$

The embedding coordinates are computed by a global operation that couples all data points in connected components of the graph defined by the weight matrix. Since the computation is always coupled across data points, the algorithm leverages overlapping local information to discover global structure.

The authors neglected to fully derive the solutions to these two equations in the brief paper in Science [Roweis00], and they added incomplete derivations in a companion paper [Roweis01], so I have taken the opportunity to derive them in full detail below. (Note that my notation uses the transpose of the matrices $\mathbf{W}$ and $\mathbf{M}$ used by the authors because this allows the derivations to follow more directly from the others in this report.)

4.2.1 Solution for Reconstruction Weights

The i^{TH} data point is reconstructed independently from all others, so the i^{TH} column of $\mathbf{W}$ is the only column of $\mathbf{W}$ participating in each term of the summation in equation 49. Therefore, the minimization can be determined one term at a time, allowing us to notate each column by $\mathbf{w}$ for convenience in the following equations. Also denote neighbors by $\mathbf{n}_j$.

$$J_1(\mathbf{w}) = \left\| \mathbf{x}_i - \sum_{j=1}^{K} w_j \mathbf{n}_j \right\|^2 \tag{51}$$

The quality of the reconstruction of a data point from the linear combination of its neighbors depends on how close it lies to the space spanned by its neighbors. Finding this combination poses a problem with D equations and K unknowns.

$$\begin{bmatrix} | & | & & | \\ \mathbf{n}_1 & \mathbf{n}_2 & \cdots & \mathbf{n}_K \\ | & | & & | \end{bmatrix}_{DxK} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}_{Kx1} = \begin{bmatrix} | \\ \widetilde{\mathbf{x}}_i \\ | \end{bmatrix}_{Dx1} \tag{52}$$

For the typical case of D > K, the minimization in equation 52 is an over-constrained, least squares problem. (For the under-constrained case D < K, regularization must be added to make the problem well-posed.) Moreover, since only the K of N coefficients that weight the K neighbors of $\mathbf{x}_i$ are allowed to take on non-zero values, the solution of the entire column of coefficients, **w,** poses a constrained least squares problem. An additional constraint is imposed by the requirement that the reconstruction hold equally well in D-dimensional space as in d-dimensional space. That is, the solution for the weights must be invariant to the transformations of the linear mapping (translation, rotation, and scaling) that maps each high dimensional coordinate of each neighborhood to global coordinates on the manifold. While the authors stated this without proof in the paper, I will prove the invariance to rotation and scaling. Consider multiplying each input data point by the rotation and scaling matrix, **R**. I show that the weights, **w**, applied to the original data points equal the weights, **v**, applied to the rotated and scaled points:

$$\mathbf{R}\mathbf{x}_i = \sum_{j=1}^{K} v_j \mathbf{R}\mathbf{x}_j = v_1 \mathbf{R}\mathbf{x}_1 + v_2 \mathbf{R}\mathbf{x}_2 + \cdots + v_K \mathbf{R}\mathbf{x}_K \tag{59}$$

$$\mathbf{R}^{-1}\mathbf{R}\mathbf{x}_i = \sum_{j=1}^{K} v_j \mathbf{R}^{-1}\mathbf{R}\mathbf{x}_j$$

$$\mathbf{x}_i = \sum_{j=1}^{K} v_j \mathbf{x}_j$$

$$\sum_{j=1}^{K} w_j \mathbf{x}_j = \sum_{j=1}^{K} v_j \mathbf{x}_j$$

$$\mathbf{w} = \mathbf{v}$$

Similarly, consider the translation, **t**, added to each input point:

$$x_i + \mathbf{t} = \sum_{j=1}^{K} v_j (\mathbf{x}_j + \mathbf{t}) \tag{60}$$

$$\sum_{j=1}^{K} w_j \mathbf{x}_j + \mathbf{t} = \sum_{j=1}^{K} v_j \mathbf{x}_j + \sum_{j=1}^{K} v_j \mathbf{t}$$

Therefore, the weights are invariant to translations of the data point and its neighbors (**w** = **v**) if and only if we add the following constraint that all rows of the weight matrix sum to 1:

$$\sum_j w_j = 1 \tag{61}$$

This constraint allows us to massage the minimization cost function:

$$\begin{aligned} J_1(\mathbf{w}) &= \left\| \mathbf{x}_i - \sum_{j=1}^{K} w_j \mathbf{n}_j \right\|^2 \\ &= \left\| \mathbf{x}_i - \mathbf{N}\mathbf{w} \right\|^2 \\ &= \left\| \mathbf{X}\mathbf{w} - \mathbf{N}\mathbf{w} \right\|^2 \\ &= \left\| (\mathbf{X} - \mathbf{N})\mathbf{w} \right\|^2 \\ &= ((\mathbf{X} - \mathbf{N})\mathbf{w})^T ((\mathbf{X} - \mathbf{N})\mathbf{w}) \\ &= \mathbf{w}^T (\mathbf{X} - \mathbf{N})(\mathbf{X} - \mathbf{N})^T \mathbf{w} \\ &= \mathbf{w}^T \mathbf{S}\mathbf{w} \end{aligned} \tag{62}$$

Where I define the **X** matrix to be the matrix of columns $\mathbf{x}_i$ repeated K times. Then **S** represents the scatter matrix of (**X**-**N**):

$$(\mathbf{X} - \mathbf{N}) = \begin{bmatrix} | & | & & | \\ \mathbf{X}_i - \mathbf{N}_1 & \mathbf{X}_i - \mathbf{N}_2 & \cdots & \mathbf{X}_i - \mathbf{N}_K \\ | & | & & | \end{bmatrix} \tag{63}$$

Minimize by finding the zero-gradient condition using Lagrange multipliers to accommodate the constraint that the weights sum to 1:

$$\begin{aligned} L(\mathbf{w}) &= \mathbf{w}^T \mathbf{S}\mathbf{w} + \lambda(\mathbf{w}^T \mathbf{1} - 1) \\ \frac{\partial L}{\partial w} &= 0 = 2\mathbf{S}\mathbf{w} + \lambda \mathbf{1} \\ \mathbf{S}\mathbf{w} &= c\mathbf{1} \end{aligned} \tag{64}$$

Theoretically the above solution for w involves inverting S, but in practice, we equivalently solve with the constant c arbitrarily set to 1, and then scale **w** to sum to 1.

4.2.2 Solution for Lower-Dimensional Coordinates

For the following derivation of a solution to the cost function for the manifold coordinates, define **Y** to be the matrix of columns, $\mathbf{y}_i$, and $\mathbf{w}_i$ to be the i^{TH} column of **W,** and $\mathbf{I}_i$ to be the i^{TH} column of the identity matrix.

$$\begin{aligned} J_2(\mathbf{Y}) &= \sum_{i=1}^{N} \left\| \mathbf{y}_i - \sum_{j=1}^{N} W_{ij} \mathbf{y}_j \right\|^2 \\ &= \sum_{i=1}^{N} \left\| \mathbf{Y}\mathbf{I}_i - \mathbf{Y}\mathbf{w}_i \right\|^2 \\ &= \sum_{i=1}^{N} \left\| \mathbf{Y}(\mathbf{I}_i - \mathbf{w}_i) \right\|^2 \end{aligned} \tag{65}$$

As an aside, note that for column vectors, $\mathbf{a}_i$, of a given matrix **A** [Golub96]:

$$\sum_i \|\mathbf{a}_i\|^2 = \sum_i \mathbf{a}_i^T \mathbf{a}_i = trace(\mathbf{A}^T \mathbf{A}) = \|\mathbf{A}\|^2 \quad \textbf{(66)}$$

Therefore:

$$J_2(\mathbf{Y}) = \|\mathbf{Y}(\mathbf{I} - \mathbf{W})\|^2 \quad \textbf{(67)}$$

Since $|\mathbf{A}|^2 = \Sigma a_{ij}^2 = \Sigma a_{ji}^2 = |\mathbf{A}^T|^2$, we can write:

$$\begin{aligned} J_2(\mathbf{Y}) &= \left\|(\mathbf{I} - \mathbf{W})^T \mathbf{Y}^T\right\|^2 \\ &= trace(\mathbf{Y}(\mathbf{I} - \mathbf{W})(\mathbf{I} - \mathbf{W})^T \mathbf{Y}^T) \\ &= trace(\mathbf{Y}\mathbf{M}\mathbf{Y}^T) \end{aligned} \quad \textbf{(68)}$$

The matrix **M** is the scatter matrix that would be the zero matrix (appropriately yielding a cost of J=0) if **W** were to equal **I**, (which is impossible given the earlier constraint that a point cannot be its own neighbor):

$$\mathbf{M} = (\mathbf{I} - \mathbf{W})(\mathbf{I} - \mathbf{W})^T \quad \textbf{(69)}$$

As before, we can minimize by finding the zero-gradient condition using Lagrange multipliers to include the constraints that make the problem well posed. In this case of determining new coordinates, **Y**, we must remove the degrees of freedom associated with defining **Y** to have an arbitrary origin and orientation. Therefore, constrain **Y** to have the simplest possible mean and covariance:

$$\begin{aligned} \sum_{i=1}^{N} \mathbf{y}_i &= \mathbf{0} \\ \frac{1}{N-1} \sum_{i=1}^{N} \mathbf{y}_i \mathbf{y}_i^T &= \mathbf{I} \end{aligned} \quad \textbf{(70)}$$

Initially, optimize the cost function, J_2, using the second of these two constraints:

$$L(\mathbf{Y}) = \mathbf{Y}\mathbf{M}\mathbf{Y}^T + \lambda(\mathbf{Y}\mathbf{Y}^T - (N-1)\mathbf{I}) \quad \textbf{(71)}$$

To compute the derivative, apply the transposes of the identities from equation 13, so that:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{Y}} &= 0 = 2\mathbf{M}\mathbf{Y}^T + 2\lambda\mathbf{Y}^T \\ \mathbf{M}\mathbf{Y}^T &= \lambda\mathbf{Y}^T \end{aligned} \quad \textbf{(72)}$$

Thus, L is minimized when the columns of the $\mathbf{Y}^T$ (rows of **Y)** are the eigenvectors associated with the lowest eigenvalues of **M**. We can impose the first constraint above (for zero mean) by discarding the eigenvectors associated with eigenvalue 0 (free translation), and keeping the eigenvectors, $\mathbf{u}_i$, associated with the bottom d nonzero eigenvalues. These produce the d rows of the d-by-N output matrix **Y**:

$$\mathbf{Y} = \begin{bmatrix} | & | & & | \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_N \\ | & | & & | \end{bmatrix}_{dxN} = \begin{bmatrix} - & \mathbf{u}_1 & - \\ - & \mathbf{u}_2 & - \\ & \vdots & \\ - & \mathbf{u}_d & - \end{bmatrix}_{dxN} \quad (73)$$

This concludes the complete derivation, and the following table summarizes the three steps of the LLE algorithm.

LLE Algorithm		
Step	**Name**	**Description**
1 $O(DN^2)$	K neighbors	Compute the neighbors of each data point $\mathbf{x_i}$
2 $O(DNK^3)$	W_{ij}	Compute the weights W_{ij} that best reconstruct each data point $\mathbf{x_i}$ from its neighbors, minimizing the cost by constrained linear least squares.
3 $O(dN^2)$	$\mathbf{y}_i$	Compute the vectors $\mathbf{y_i}$ that are best reconstructed by the weights W_{ij}, using the bottom nonzero eigenvectors of the scatter matrix of (**I-W**).

4.3 Discussion

Recall that we derived Eigenfaces (PCA) by finding a new set of data points, Y, that are a linear combination of the original points, X, but of a lower dimension. We also derived PCA from the approach of minimizing the distance of *all* data points to the new coordinate vectors. In contrast, LLE, finds new global coordinate vectors that best fit only the *local* neighborhood geometric relationships between points.

Attempts to extend PCA to nonlinear data sets generally fall into two broad classes which LLE overcomes. Within the first set, local linear techniques [Bregler95, Basri98, Hinton95] attempt to extend PCA to nonlinear data by first clustering the data, and then performing PCA locally within each cluster. However, they are not designed to represent the global structure of a data set within a single coordinate system. Within the second set, greedy optimization procedures [Durby87, Kohonen88, Kramer91, Hecht-Nielson95, Bishop98] attempt to discover global structure, but lack the non-iterative, polynomial time procedure with guarantee of global optimality, absence of many free parameters, and the ability to discover manifolds of arbitrary dimension.

The scissors sketch in section 4.1 is an accurate portrayal of the LLE algorithm if each square represents one data point. Each square does not represent each neighborhood because the squares are disjoint. In the algorithm, there exists a neighborhood of size K surrounding every point. The scissors analogy would have to involve cutting K manifolds of overlapping squares to build one reconstruction on the table. This visualization depicts how global structure emerges from overlapping local neighborhoods. This is the main idea behind LLE (and also Isomap, as described later), that overlapping local structure, collectively analyzed, can provide information about global geometry.

This visualization leads me to suggest ways to improve the algorithm. The quality of the characterization of the nonlinear manifold is heavily dependent on the choice of neighborhoods. But many natural data sets with embedded manifolds feature neither a uniform nor dense

sampling. I suggest allowing a data point's number of neighbors, K, to vary locally according to the properties of the sampling. The author's proposal of choosing neighbors by their proximity to a fixed radius, ε, achieves the effect of a locally varying K, but ignores the ill effects of high curvature. A processing step of clustering can be used to identify local neighborhoods. Although this approach was taken in the aforementioned list of alternative methods, those methods lacked the global optimization step of LLE.

Furthermore, I suggest making the algorithm more robust to noisy outliers from the manifold. Weight less those reconstruction weights computed for points that exhibit poor reconstructions (in a least squares sense) from their linear combination of neighbors.

Finally, I address the issue of mapping a novel test image from the input space to the manifold space. The LLE authors mention neither this problem nor its inverse problem of reconstructing an image from its low-dimensional representation (mapping arbitrary manifold points back to input space). I propose determining the novel point's K neighbors, compute the point's reconstruction via linear combination of neighbors, and *approximating* the point's embedded manifold coordinates by applying its weights to the **Y** obtained in the training stage. For a single test point, this process has time complexity $O(DNK^3)$ verses $O(DN^2)$ for training on N points.

5 Isomap

5.1 Idea

The main idea behind the Isomap (isometric feature mapping) algorithm [Tenenbaum00, Tenenbaum01] is to perform classical MDS to map data points from their high-dimensional input space to low-dimensional coordinates of a nonlinear manifold. The key contribution is to compute the MDS pairwise distances not in the input Euclidean space, but in the geodesic space of the manifold. The geodesic distances represent the shortest paths along the curved surface of the manifold (measured as if the surface were flat). Clues to the shape of the manifold are only provided by the input data as surface samples. The actual geodesic distances are therefore approximated by a sequence of short hops between neighboring sample points. Finally, MDS is applied to the geodesic distances to find a set of low-dimensional points with similar pairwise distances.

Since only the geodesic distances represent the true, low-dimensional geometry of the manifold, the algorithm is capable of discovering nonlinear degrees of freedom that underlie complex natural observations.

5.2 Algorithm

The Isomap algorithm proceeds through three steps: identifying the neighbors of each input point, computing the geodesic pairwise distances between all points, and solving for the manifold coordinates using MDS.

The first step is to determine which points are neighbors on the manifold, **M**, based on the distances $d_X(i,j)$ between pairs of points (i,j) in the input space, **X**. One method involves identifying the K nearest neighbors, while another selects all points within some fixed radius, ε. These neighborhood relations are represented as a weighted graph, **G**, over the data points, with edges of weight $d_X(i,j)$ between neighbors.

The second step involves estimating the geodesic distances $d_M(i,j)$ between all pairs of points on the manifold, **M**, by computing approximations as the shortest path distances $d_G(i,j)$ in the graph

G. Techniques to facilitate this computation include Dijkstra's $O(N^2)$ algorithm [Cormen01] and Floyd's $O(N^3)$ algorithm [Kumar94] described below:

```
Initialize dG(i,j) to dX(i,j) if (i,j) neighbors
for i = 1 to N do
     for j = 1 to N do
          for k = 1 to N do
               dG(i,j) ← min(dG(i,j), dG(i,k)+dG(k,j))
```

Then the final matrix of graph distances $\mathbf{D}_G=\{d_G(i,j)\}$ will contain the shortest path distance between all pairs of points in **G**.

The third step applies classical MDS to $\mathbf{D}_G$ to construct an embedding of the data in a d-dimensional Euclidean space **Y** that best preserves the manifold's estimated intrinsic geometry. As derived in section 2.2, the global minimum of the cost function is achieved by setting the coordinates of $\mathbf{y}_i$ to the top d eigenvectors, of the inner-product matrix **B** obtained from $\mathbf{D}_G$:

$$\mathbf{Y}=\begin{bmatrix} | & | & & | \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_N \\ | & | & & | \end{bmatrix}_{dxN} = \begin{bmatrix} - & \sqrt{\lambda_1}\mathbf{v}_1 & - \\ - & \sqrt{\lambda_2}\mathbf{v}_2 & - \\ & \vdots & \\ - & \sqrt{\lambda_d}\mathbf{v}_d & - \end{bmatrix}_{dxN} \quad (74)$$

The following table summarizes the algorithm and its time complexity:

ISOMAP ALGORITHM		
Step	**Name**	**Description**
1 $O(DN^2)$	Construct neighborhood graph	Define the graph **G** over all data points by connecting neighbors. Set edge lengths to $d_x(i,j)$
2 $O(DN^2)$	Compute shortest paths	Compute matrix $\mathbf{D}_G=\{d_G(i,j)\}$ to contain the shortest path distance between all pairs of points in **G**.
3 $O(dN^2)$	Construct d-dimensional embedding	Use MDS to compute the top d eigenvectors of **B** obtained from $\mathbf{D}_G$ to arrive at d-dimensional coordinate vectors $\mathbf{y}_i$.

5.3 Discussion

The Isomap algorithm shares several similarities with LLE. Both begin with a preprocessing step that decides for each data point which of the other data points should be considered its neighbors. Both seek to preserve the intrinsic geometry of the data by computing a measure of local geometry of the manifold, after which, the original data points may be discarded. For LLE, the measure was the linear combination of neighbors, while Isomap instead captures intrinsic geometry in the geodesic manifold distance between all pairs of data points. Furthermore, like LLE, Isomap overcomes the same attempts to extend PCA as described in section 4.3.

The challenge of nonlinearity can be clearly illustrated by the "Swiss Roll" example in Figure 4. Points on the underlying manifold with distant geodesic distances may have small Euclidean distance. Only the geodesic distances reflect the true low-dimensional geometry of the manifold, but PCA and conventional MDS effectively see the Euclidean structure, and fail to detect the intrinsic two-dimensionality.

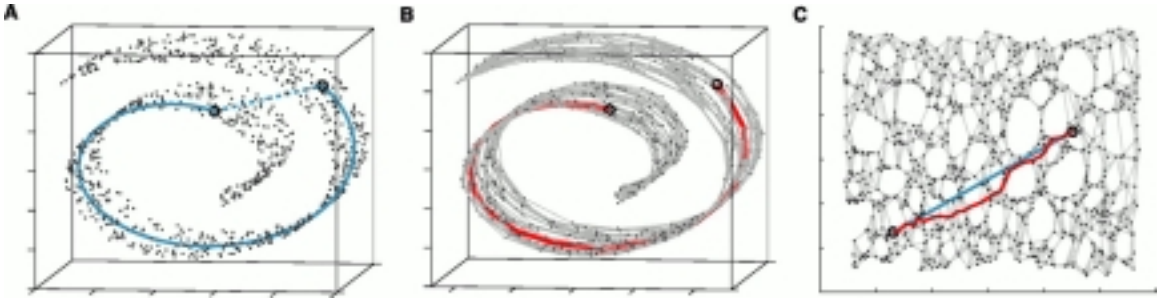

Figure 4: Isomap operating on Swiss Roll
Left: inaccuracy of Euclidean distance (dashed line) compared with geodesic distance (solid line). Center and Right: approximation of geodesic distance (red line) overestimates actual distance.

Isomap also shares similarities with Eigenfaces because Eigenfaces could be computed using MDS where the pairwise distances matrix $\mathbf{D}_G$ would be computed directly from the input coordinates, $\mathbf{X}$ in Euclidean space. The only difference with Isomap is that $\mathbf{D}_G$ is computed in geodesic space, and the algorithm continues on from there exactly as Eigenfaces would. Isomap presents an attractive alternative to MDS for data sets that are known to have a smooth manifold, such as a video sequence or data set that conceptually could be organized as a visually pleasing video sequence. That is, smooth manifolds manifest themselves as slightly noticeable changes between an ordered set of examples. When it is unknown *a priori* if a data set contains nonlinear structure, one merely needs to run both PCA and Isomap to observe if the resulting low-dimensional coordinates differ significantly.

Just as PCA and MDS are guaranteed, given sufficient data, to recover the true structure of linear manifolds, Isomap is guaranteed asymptotically to recover the true dimensionality and geometric structure of a strictly larger class of nonlinear manifolds. This is because as the number of data points increases, the graph distances provide increasingly better approximations to true geodesic distances. The approximation will tend to overestimate $d_M(i,j)$ due to the graph's discreteness. Highly and irregularly curved surfaces may require an impractically large sample size. Also, K needs to be increased to avoid "shortcuts" near regions of high surface curvature.

6 Summary

Three techniques have been discussed for mapping data points to a lower dimension, each of which featured different optimization constraints. Eigenfaces posed the new coordinates as a linear combination of the original that accounts for maximum variance, LLE sought to preserve the intrinsic geometry of neighboring points, and Isomap approximated geodesic distances as a sequence of neighbor-to-neighbor hops. The following table summarizes the major algorithmic issues as well as practical aspects of each method.

	Eigenfaces	**LLE**	**Isomap**
Optimization Constraint	Linear combination of original coordinates that accounts for most variance	Local intrinsic geometry represented as linear combination of neighbors	Geodesic distance approximated as neighbor-to-neighbor hopping distance
Parameters	None	K or ε	K or ε
Global optimality	Yes	Yes	Yes
Handles nonlinear manifolds	No	Yes	Yes
Training Complexity	$O(N^3)$	$O(DN^2)$	$O(DN^2)$

7 References

[Bai00] Z. Bai, J. Demmel, J. Dongarra, A, Ruhe, H. van der Vorst. *Templates for Solution of Algebraic Eigenvalue Problems*. Society for Industrial Applied Mathematics, 2000.

[Basri98] R. Basri, D. Roth, D. Jacobs. "Clustering Appearances of 3D Objects". In: *Computer Vision and Pattern Recognition (CVPR)*. 1998; 414-420.

[Bishop95] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[Bishop98] C.M. Bishop, M. Svensen, C.K.I. Williams. "GTM: The Generative Topographic Mapping". *Neural Computation* 1998; 10:215-234.

[Bregler95] C. Bregler, S.M. Omoundro. "Nonlinear Image Interpolation using Manifold Learning". *Advances in Neural Information Processing Systems* 1995; 7:973-980.

[Chatfield80] C. Chatfield, A.J. Collins. *Introduction to Multivariate Analysis*. Chapman & Hall, 1980.

[Cormen01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C.Stein. *Introduction to Algorithms*. MIT Press, 2001.

[Cox01] T.F. Cox, M.A.A. Cox. *Multidimensional Scaling*. Chapman & Hall, 2000.

[Dempster77] A.P. Dempster, N.M. Laird, D.B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". *Journal Royal Statistical Society* 1977; 39:1-38.

[Duda01] R.O. Duda, P.E. Hart, D.G. Stork. *Pattern Classification*. John Wiley & Sons, 2001.

[Durbin87] R. Durbin, D. Willshaw. "An Analogue Approach to the Travelling Salesman Problem Using an Elastic Surface Net". *Nature* April 1987; 326:689-691.

[Golub96] G.H. Golub, C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[Harris01] R.J. Harris. *A Primer of Multivariate Statistics*. Lawrence Erlbaum Associates, 2001.

[Healy00] M.J.R. Healy. *Matrices for Statistics*. Clarendon Press, 2000.

[Hecht-Nielson95] R. Hecht-Nielsen. "Replicator Neural Networks for Universal Optimal Source Coding". *Science* September 1995; 279:1860-1861.

[Hinton95] G.E. Hinton, M. Revow, P. Dayan. "Recognizing Handwritten Digits Using Mixtures of Linear Models". *Advances in Neural Information Processing Systems* 1995; 7:1015-1022.

[Johnson92] R.A. Johnson, D.W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 1992.

[Jolliffe86] I.T. Jolliffe. *Principle Component Analysis*. Springer-Verlag, 1986.

[Kohonen89] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, 1989.

[Kramer91] M.A Kramer. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks". *AIChE Journal* February 1991; 37:233-243.

[Kumar94] V. Kumar, A. Grama, A. Gupta, G. Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings, 1994.

[Liu98] A.K. Liu, J.W. Belliveau, A.M. Dale. "Visualizing Spatial Resolution of Linear Estimation Techniques of Electromagnetic Brain Activity Localization". In: W.M. Wells III, A. Colchester, S. Delp, eds. *First International Conference on Medical Image Computing and Computer-Assisted Intervention*. Boston: Springer-Verlag, 1998; 670-678.

[Mardia79] K.V. Mardia, J.T. Kent, J.M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[Roweis00] S.T. Roweis, L.K. Saul. "Nonlinear Dimensionality Reduction by Locally Linear Embedding". *Science* December 2000; 290:2323-2326.

[Roweis01] Introduction to Locally Linear Embedding. http://www.cs.toronto.edu/~roweis/lle/publications.html.

[Roweis98] S. Roweis. "EM Algorithms for PCA and SPCA". *Advances in Neural Information Processing Systems* 1998; 10:

[Strang98] G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1998.

[Tenenbaum00] J.B. Tenenbaum, V.d.Silva, J.C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". *Science* December 2000; 290:2319-2323.

[Tenenbaum01] Introduction to Locally Linear Embedding. http://isomap.stanford.edu/.

[Turk91a] M.A. Turk, A.P. Pentland. "Face Recognition Using Eigenfaces". In: *Computer Vision and Pattern Recognition (CVPR)*. Maui, Hawaii: 1991; 586-591.

[Turk91b] M. Turk, A. Pentland. "Eigenfaces for Recognition". *Journal of Cognitive Neuroscience* 1991; 3:71-86.