# Manifold Learning Examples – PCA, LLE and ISOMAP

Dan Ventura

October 14, 2008

**Abstract**

We try to give a helpful concrete example that demonstrates how to use PCA, LLE and Isomap, attempts to provide some intuition as to how and why they work, and compares and contrasts the three techniques.

## 1 Introduction

Suppose we are given the following data points in $\Re^2$ (see Figure 1):

$$\left\{ \begin{pmatrix} -20 \\ -8 \end{pmatrix}, \begin{pmatrix} -10 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 10 \\ 1 \end{pmatrix}, \begin{pmatrix} 20 \\ 8 \end{pmatrix} \right\}$$

These points all live on a nonlinear $1D$ manifold described by the equation $f(x) = (x/10)^3$. We would like to represent this data in $\Re^1$. What mapping is appropriate? PCA, LLE and Isomap all have something slightly different to say, and we will try to see how they compare by using this simple example.

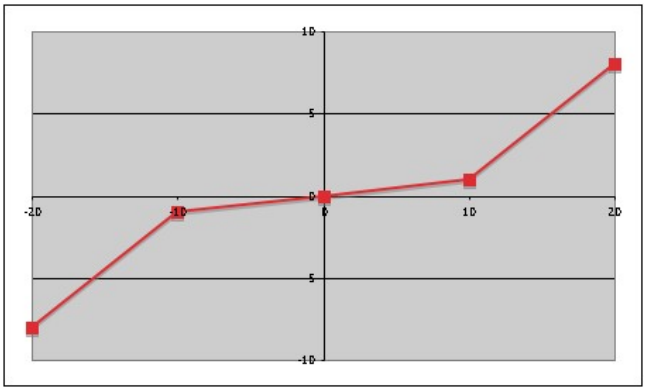


Figure 1: Sample data points on the manifold defined by $f(x) = (x/10)^3$

## 2 PCA

We begin by computing the covariance matrix (note that in general we would have to preprocess the data by subtracting out the mean, but in this clever example the mean is zero),

$$C = \begin{pmatrix} 200 & 68 \\ 68 & 26 \end{pmatrix}$$

Next, we calculate the eigenvectors and eigenvalues of the covariance matrix. How do we do this, you might ask – MATLAB or some other math package will do it nicely (note that I am making liberal use of rounding throughout this example):

$$\text{Eigenvalues:} \quad 223 \quad \text{and} \quad 2.6$$

$$\text{Eigenvectors:} \quad \begin{pmatrix} 0.95 \\ 0.33 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0.33 \\ -0.95 \end{pmatrix}$$

Notice that one eigenvalue is much larger than the other, indicating a significant difference in the variance in the two derived dimensions. Even though this data is non-linear, we can get a pretty decent linear fit in $1D$ and the best single linear axis for the data is represented by the eigenvector associated with the large eigenvalue (see Figure 2).

Remember, if we think of the covariance matrix as an operator, the eigenvectors of that operator define the basis – if we limit ourselves to thinking about unit vectors, they are the axes that are invariant under the application of the operator. So, our new major (and only since we are going to $1D$) axis is defined by the first eigenvector above and we transform our data into the new ($1D$) coordinate system by taking the dot product of this eigenvector with our data vectors:

$$\begin{pmatrix} 0.95 & 0.33 \end{pmatrix} \begin{pmatrix} -20 \\ -8 \end{pmatrix} = (-21.64)$$

$$\begin{pmatrix} 0.95 & 0.33 \end{pmatrix} \begin{pmatrix} -10 \\ -1 \end{pmatrix} = (-9.83)$$

$$\begin{pmatrix} 0.95 & 0.33 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = (0)$$

$$\begin{pmatrix} 0.95 & 0.33 \end{pmatrix} \begin{pmatrix} 10 \\ 1 \end{pmatrix} = (9.83)$$

$$\begin{pmatrix} 0.95 & 0.33 \end{pmatrix} \begin{pmatrix} 20 \\ 8 \end{pmatrix} = (21.64)$$

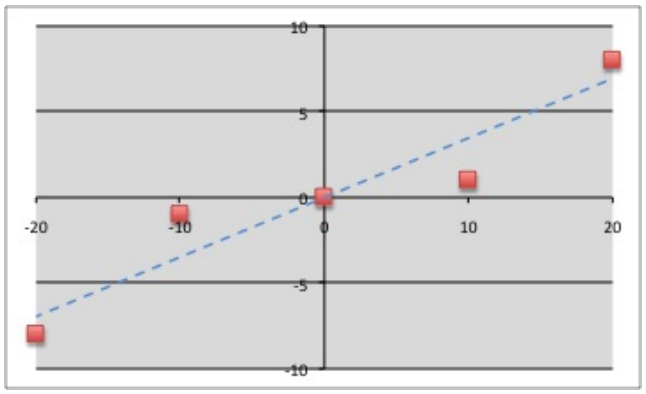


Figure 2: Major axis discovered by PCA

# 3 Isomap

Isomap uses the same basic idea as PCA, the difference being that linearity is only preserved locally (via small neighborhoods). The global geometry of the discovered axes are nonlinear because of the fact that these small neighborhoods are stitched together without trying to maintain linearity. The result is a nonlinear axis or axes that define a manifold. The steps of the algorithm are basically

1. define neighbors for each data point
2. find interpoint distances (graph represented as a matrix $M$)
3. find eigenvectors of $M$

In PCA we try to preserve covariance. Here we try to do the same thing in a nonlinear way and our approach is to try to preserve interpoint distance on the manifold. The matrix $M$ acts just as the matrix $C$ in PCA. In fact, it is really the same thing in a higher dimensional space. $M$ can actually be thought of as the covariance matrix for the space whose dimensions are defined by the data points (if we have $N$ data points, $M$ is an $N \times N$ matrix). Since in this $N$-dimensional space, the dimensions are the datapoints, the covariance for a particular pair of dimensions is simply the distance between the datapoints that define those dimensions. Pretty much everything we do here is linear, just as it is in PCA. The source of the nonlinearity is the way in which we calculate our interpoint distances. We do not, in general use the Euclidean distances between the points. Rather, we use those distances only for points considered neighbors. The rest of interpoint distances are calculated by finding the shortest path through the graph on the manifold.

Starting with the original data points as shown in Figure 1 and setting the number of neighbors $K = 2$, we can begin to calculate the matrix $M$. Here

we have filled in only the actual Euclidean distances of neighbors (we index the matrix with the data in Figure 1 from left to right and rows represent data points and columns represent neighbors):

$$M_{\text{neighbors}} = \begin{pmatrix} 0 & 12.21 & 21.54 & ? & ? \\ 12.21 & 0 & 10.05 & ? & ? \\ ? & 10.05 & 0 & 10.05 & ? \\ ? & ? & 10.05 & 0 & 12.21 \\ ? & ? & 21.54 & 12.21 & 0 \end{pmatrix}$$

Next, we can fill in the remaining entries in $M$ using Floyd's algorithm:

$$M_{\text{Floyd}} = \begin{pmatrix} 0 & 12.21 & 21.54 & 31.59 & 43.8 \\ 12.21 & 0 & 10.05 & 20.1 & 32.31 \\ 22.26 & 10.05 & 0 & 10.05 & 22.26 \\ 32.31 & 20.1 & 10.05 & 0 & 12.21 \\ 43.8 & 31.59 & 21.54 & 12.21 & 0 \end{pmatrix}$$

Notice that $M$ is not quite symmetric. Can you see why? Now, we lied just a little in step 3 above – we don't actually calculate the eigenvectors of $M$. What we do instead is calculate the eigenvectors of $\tau(M)$, where $\tau(M) = -HSH/2$ and $S_{ij} = (M_{ij})^2$ (that is, $S$ is the matrix of squared distances) and $H_{ij} = \delta_{ij} - 1/N$. (Recall that $N$ is the number of data points and $\delta$ is the Krönecker delta function). I believe the point of the $\tau$ operator is analogous to subtracting the mean in the PCA case. So,

$$\tau(M_{\text{Floyd}}) = \begin{pmatrix} 480.6 & 210.7 & 0.987 & -213.8 & -478.5 \\ 223 & 102 & -0.658 & -100 & -224.4 \\ -0.709 & 1 & -0.658 & 1 & -0.709 \\ -224.4 & -100 & -0.658 & 102 & 223 \\ -478.5 & -213.8 & 0.987 & 210.7 & 480.6 \end{pmatrix}$$

Now, remember, this is just like the covariance matrix $C$ in the PCA example. What we have done is taken $2D$ data and "exploded" it into $5D$. This interpoint distance matrix is the covariance matrix for the $5D$ space. So, we now find the eigenvectors and eigenvalues for $\tau(M)$:

| Eigenvalues: | 1157 | 3.5 | 3.3 | 0.5 | −0.02 |
|---|---|---|---|---|---|
| Eigenvectors: | $\begin{pmatrix} -0.64 \\ -0.3 \\ 0 \\ 0.3 \\ 0.64 \end{pmatrix}$ | $\begin{pmatrix} -0.66 \\ 0.28 \\ 0.37 \\ 0.28 \\ -0.66 \end{pmatrix}$ | $\begin{pmatrix} -0.29 \\ 0.65 \\ 0 \\ -0.65 \\ 0.29 \end{pmatrix}$ | $\begin{pmatrix} -0.95 \\ -0.56 \\ 0.17 \\ -0.56 \\ -0.95 \end{pmatrix}$ | $\begin{pmatrix} -0.08 \\ -0.33 \\ -0.88 \\ -0.33 \\ -0.08 \end{pmatrix}$ |

Just as in the PCA example, the first eigenvalue dominates the others, though in this case, the difference in even greater. Why? Our final step is

to transform our original data into the new coordinate system defined by the major eigenvector. Let us re-emphasize that this new axis is linear in the $5D$ space but is nonlinear in the way it preserves interpoint distances in the lower dimensional space that we are really interested in. Because of this high dimesional/low dimensional game we are playing, the data transformation is a little murkier than it was for PCA. We have a $5D$ eigenvector but we need a transform for $2D$ datapoints. The trick is, and I quote from the paper, "Let $\lambda_p$ be the $p$-th eigenvalue (in decreasing order) of the matrix $\tau(D_G)$, and $v_p^i$ be the $i$-th component of the $p$-th eigenvector. Then set the $p$-th component of the $d$-dimensional coordinate vector $\mathbf{y}_i$ equal to $\sqrt{\lambda_p} v_p^i$."

We can rephrase this in terms of our example, for which we are only interested in the largest eigenvalue – Let $\lambda_1$ be the 1-st eigenvalue (in decreasing order) of the matrix $\tau(M)$, and $v_1^i$ be the $i$-th component of the 1-st eigenvector. Then set the 1-st component of the 1-dimensional coordinate vector $\mathbf{y}_i$ equal to $\sqrt{\lambda_1} v_1^i$. Here, $i$ ranges from 1 to 5 because there are 5 components to the eigenvector because there are 5 data points. So, our transformed data points are

$$\begin{pmatrix} -20 \\ -8 \end{pmatrix} \Rightarrow (-21.77)$$

$$\begin{pmatrix} -10 \\ -1 \end{pmatrix} \Rightarrow (-10.2)$$

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow (0)$$

$$\begin{pmatrix} 10 \\ 1 \end{pmatrix} \Rightarrow (10.2)$$

$$\begin{pmatrix} 20 \\ 8 \end{pmatrix} \Rightarrow (21.77)$$

Since the stated goal of Isomap is to preserve geodesic distances rather than Euclidean distances, let's end this section by comparing the original geodesic and Euclidean distances in $2D$ with the interpoint distances of the final $1D$ data. Here are the original Euclidean distances:

$$D_{\text{Euclidean}} = \begin{pmatrix} 0 & 12.21 & 21.54 & 31.32 & 43.08 \\ 12.21 & 0 & 10.05 & 20.1 & 31.32 \\ 21.54 & 10.05 & 0 & 10.05 & 21.54 \\ 31.32 & 20.1 & 10.05 & 0 & 12.21 \\ 43.08 & 31.32 & 21.54 & 12.21 & 0 \end{pmatrix}$$

Here are the original geodesic distances (calculated by moving along the gross approximation of the manifold given the five points and linear interpolation between them):

$$D_{\text{geodesic}} = \begin{pmatrix} 0 & 12.21 & 22.26 & 31.32 & 43.53 \\ 12.21 & 0 & 10.05 & 20.1 & 31.32 \\ 22.26 & 10.05 & 0 & 10.05 & 22.26 \\ 31.32 & 20.1 & 10.05 & 0 & 12.21 \\ 43.53 & 31.32 & 22.26 & 12.21 & 0 \end{pmatrix}$$

Now, here are the interpoint distances for newly transformed data:

$$D_{\text{new}} = \begin{pmatrix} 0 & 11.57 & 21.77 & 31.97 & 43.54 \\ 11.57 & 0 & 10.2 & 20.4 & 31.97 \\ 21.77 & 10.2 & 0 & 10.2 & 21.77 \\ 31.97 & 20.4 & 10.2 & 0 & 11.57 \\ 43.54 & 31.97 & 21.77 & 11.57 & 0 \end{pmatrix}$$

To emphasize the point, let's look at the $L^2$ matrix norm $\sqrt{\Sigma_{i,j} A_{ij}^2}$ of the difference between the Euclidean distance matrix and the new distance matrix

$$E_1 = \|D_{\text{Euclidean}} - D_{\text{new}}\|_{L^2} = 2.05$$

and the $L^2$ matrix norm of the difference between the geodesic distance matrix and the new distance matrix

$$E_2 = \|D_{\text{geodesic}} - D_{\text{new}}\|_{L^2} = 2.14$$

Hmmm. If we are really preserving geodesic distance rather than Euclidean, we should expect $E_1 > E_2$, but that is not what we find. Can you figure out why?

# 4 LLE

Locally Linear Embedding (LLE) does the same basic thing as Isomap – it finds a nonlinear manifold by stitching together small linear neighborhoods. The difference between the two algorithms is in how they do the stitching. As we saw in the previous section, Isomap does this by doing a graph traversal. In contrast, LLE does it by finding a set of weights that perform local linear interpolations that closely approximate the data. This time the steps are

1. define neighbors for each data point
2. find weights that allow neighbors to interpolate original data accurately (represented as a matrix $W$)
3. given those weights, find new data points that minimize interpolation error in lower dimensional space

We again consider the data of Figure 1 and again set the number of neighbors $K = 2$. We must find a weight matrix $W$ that minimizes the cost function

$$\epsilon(W) = \sum_i \left| \vec{X}_i - \Sigma_j W_{ij} \vec{X}_j \right|^2$$

subject to the constraints that $W_{ij} = 0$ if $\vec{X}_j$ is not a neighbor of $\vec{X}_i$ and that the rows of the weight matrix sum to one: $\Sigma_j W_{ij} = 1$. These constraints are reasonable and guarantee some nice invariance properties that make $W$ useful for mapping into a lower dimensional space. In general finding the matrix $W$ involves solving a least-squares problem, and there is not much detail in the paper about how to do that; however, there is more detail available at http://www.cs.toronto.edu/ roweis/lle/algorithm.html. Following the recipe there results in the following weight matrix:

$$W = \begin{pmatrix} 0 & 2.06 & -1.06 & 0 & 0 \\ 0.448 & 0 & 0.552 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.552 & 0 & 0.448 \\ 0 & 0 & -1.06 & 2.06 & 0 \end{pmatrix}$$

If you really try this, you will discover that following the recipe given for calculating the weights results in a singular covariance matrix for row three – there are an infinite number of solutions. Given our particular data and constraints, it is clear that the weights we want are 0.5 and 0.5 (but I would like something better than that). Figure 3 shows the reconstructed points (using $W$) compared with the original data.

So, now we have completed steps 1 and 2 and all we have left to do is step 3. Since our original data is $2D$ and we are interested in moving to $1D$, step 3 involves finding points in $\Re^1$ that minimize the embedding cost function

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \Sigma_j W_{ij} \vec{Y}_j \right|^2$$

Once again this is done by solving an eigenvalue problem. In fact, this is quite intuitive here, if we realize that $W$ supposedly preserves important local geometry independent of what dimensional space we work in. So, what

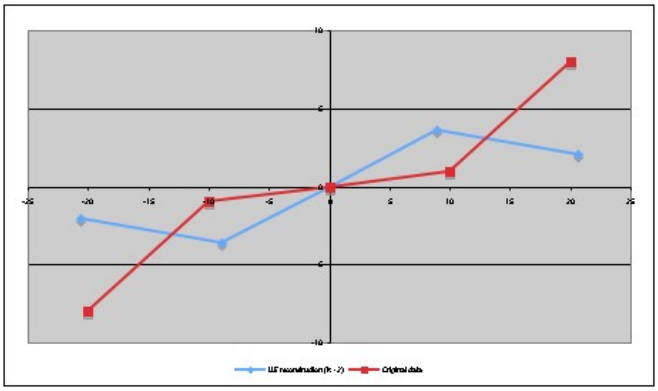

Figure 3: LLE reconstruction for K=2 compared with original data

we are looking for is a vector $\vec{V}$ of data points (each one being one of the $d$-dimensional vectors $\vec{Y_i}$) such that $\vec{V} = W\vec{V}$, or in other words we are looking for a set of points in $d$-dimensional space that don't move when $W$ is applied, or in other words a set of points that are all perfectly reconstructed by their neighbors and $W$, or in other words a set of points that perfectly fit the local geometry described by $W$. In any case, we have to find these points by solving for eigenvectors again, and it would be extra intuitive if we could use $W$. Of course, that would be a little too intuitive, and we have to do some processing to $W$ first, analogous to what we did to $M$ with $\tau()$ in the Isomap example. Digging through the notes of the paper, or visiting the website above will reveal that the way to do this is to compute the matrix $M = (I - W)^T(I - W)$ [where $I$ is the appropriately sized identity matrix] and then find *its* eigenvectors and eigenvalues. So,

$$M = \begin{pmatrix} 1.2007 & -2.508 & 1.3073 & 0 & 0 \\ -2.508 & 5.4936 & -3.2356 & 0.25 & 0 \\ 1.3073 & -3.2356 & 3.85661 & -3.2356 & 1.3073 \\ 0 & 0.25 & -3.2356 & 5.4936 & -2.508 \\ 0 & 0 & 1.3073 & -2.508 & 1.2007 \end{pmatrix}$$

| Eigenvalues: | 0 | 0.00092 | 0.28836 | 6.44338 | 10.5125 |
|---|---|---|---|---|---|
| Eigenvectors: | $\begin{pmatrix} -0.447214 \\ -0.447214 \\ -0.447214 \\ -0.447214 \\ -0.447214 \end{pmatrix}$ | $\begin{pmatrix} 0.637875 \\ 0.305148 \\ 0.0 \\ -0.305148 \\ -0.637875 \end{pmatrix}$ | $\begin{pmatrix} -0.449742 \\ 0.166066 \\ 0.667352 \\ 0.166066 \\ -0.449742 \end{pmatrix}$ | $\begin{pmatrix} 0.305148 \\ -0.637875 \\ 0 \\ 0.637875 \\ -0.305148 \end{pmatrix}$ | $\begin{pmatrix} 0.224182 \\ -0.521941 \\ 0.595517 \\ -0.521941 \\ 0.224182 \end{pmatrix}$ |

This time we are looking for small eigenvalues, rather than large, and we discard the smallest one (which will always be 0 with an accompanying eigenvector whose components are all equal). We take the $d$ next smallest and use them as the $d$ dimensions of our transformed data (the smallest gives the first dimension for each of our points, the next smallest the second dimension for each point, etc.) Since we are moving to $1D$, we will use only the (second) smallest eigenvalue/vector. Both the paper and the website imply that having found the eigenvector(s) you already have the transformed data, but that isn't exactly true. In fact, you have to divide each eigenvector by the square root of its eigenvalue to get the magnitudes correct (it took me a long time to figure this out, and I have sent a note to Sam Roweis seeking confirmation). In this particular case, since the phase of the discarded eigenvector is $-1$, we also have to multiply all our eigenvectors by $-1$ to make the signs all work out correctly. Doing both of these things gives us our final transformation:

$$\begin{pmatrix} -20 \\ -8 \end{pmatrix} \Rightarrow (-21.03)$$

$$\begin{pmatrix} -10 \\ -1 \end{pmatrix} \Rightarrow (-10.06)$$

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow (0)$$

$$\begin{pmatrix} 10 \\ 1 \end{pmatrix} \Rightarrow (10.06)$$

$$\begin{pmatrix} 20 \\ 8 \end{pmatrix} \Rightarrow (21.03)$$

The big win for LLE, supposedly, is that even though it requires solving an $N \times N$ eigenvalue problem just like Isomap does, the matrix for LLE is very sparse (because many weights are 0), whereas the matrix for Isomap has 0s only down the diagonal.

Some annoying unanswered questions for LLE:

1. Is the intuition for eigenvectors here right? In this example $WV \neq V$ so it is not a perfect fit anyway.
2. *Why are we using smallest eigenvectors this time?

# 5 Conclusion

As a final note, in these examples, we assumed a target $d$ for the number of dimensions to which we wanted to reduce. These algorithms can somewhat automatically discover the best $d$. PCA and Isomap do this in an iterative fashion, trying increasing values for $d$ and computing some sort of residual variance. Plotting this residual against $d$ will allow us to find an inflection point that indicates a good value for $d$. LLE estimates $d$ by analyzing a reciprocal cost function in which reconstruction weights derived from the lower dimensional data are applied to the original high dimensional data (again, we would iterate on $d$).