Algorithms for manifold learning

Lawrence Cayton
lcayton@cs.ucsd.edu

June 15, 2005

Abstract

Manifold learning is a popular recent approach to nonlinear dimensionality reduction. Algorithms for this task are based on the idea that the dimensionality of many data sets is only artificially high; though each data point consists of perhaps thousands of features, it may be described as a function of only a few underlying parameters. That is, the data points are actually samples from a low-dimensional manifold that is embedded in a high-dimensional space. Manifold learning algorithms attempt to uncover these parameters in order to find a low-dimensional representation of the data. In this paper, we discuss the motivation, background, and algorithms proposed for manifold learning. Isomap, Locally Linear Embedding, Laplacian Eigenmaps, Semidefinite Embedding, and a host of variants of these algorithms are examined.

1 Introduction

Many recent applications of machine learning – in data mining, computer vision, and elsewhere – require deriving a classifier or function estimate from an extremely large data set. Modern data sets often consist of a large number of examples, each of which is made up of many features. Though access to an abundance of examples is purely beneficial to an algorithm attempting to generalize from the data, managing a large number of features – some of which may be irrelevant or even misleading – is typically a burden to the algorithm. Overwhelmingly complex feature sets will slow the algorithm down and make finding global optima difficult. To lessen this burden on standard machine learning algorithms (*e.g.* classifiers, function estimators), a number of techniques have been developed to vastly reduce the quantity of features in a data set —*i.e.* to reduce the dimensionality of data.

Dimensionality reduction has other, related uses in addition to simplifying data so that it can be efficiently processed. Perhaps the most obvious is visualization; if data lies in a 100-dimensional space, one cannot get an intuitive feel for what the data looks like. However, if a meaningful two- or three-dimensional representation of the data can be found, then it is possible to "eyeball" it. Though this may seem like a trivial point, many statistical and machine learning algorithms have very poor optimality guarantees, so the ability to actually see the data and the output of an algorithm is of great practical interest.

Beyond visualization, a dimensionality reduction procedure may help reveal what the underlying forces governing a data set are. For example, suppose we are to classify an email as spam or not spam. A typical approach to this problem would be to represent an email as a vector of counts of the words appearing in the email. The dimensionality of this data can easily be in the hundreds, yet an effective dimensionality reduction technique may reveal that there are only a few exceptionally telling features, such as the word "Viagra."

There are many approaches to dimensionality reduction based on a variety of assumptions and used in a variety of contexts. We will focus on an approach initiated recently based on the observation that high-dimensional data is often much simpler than the dimensionality would indicate. In particular, a given high-dimensional data set may contain many features that are all measurements of the same underlying cause, so are closely related. This type of phenomenon is common, for example, when taking video footage of a single object from multiple angles simultaneously. The features of such a data set contain much overlapping information; it would be helpful to somehow get a simplified, non-overlapping representation of the data whose features are identifiable with the underlying parameters that govern the data. This intuition is formalized using the notion of a manifold:

the data set lies along a low-dimensional manifold embedded in a high-dimensional space, where the low-dimensional space reflects the underlying parameters and high-dimensional space is the feature space. Attempting to uncover this manifold structure in a data set is referred to as *manifold learning.*

1.1 Organization

We begin by looking at the intuitions, basic mathematics, and assumptions of manifold learning in section 2. In section 3, we detail several algorithms proposed for learning manifolds: Isomap, Locally Linear Embedding, Laplacian Eigenmaps, Semidefinite Embedding, and some variants. These algorithms will be compared and contrasted with one another in section 4. We conclude with future directions for research in section 5.

1.2 Notation and Conventions

Throughout, we will be interested in the problem of reducing the dimensionality of a given data set consisting of high-dimensional points in Euclidean space.

- The high-dimensional input points will be referred to as $x_1, x_2, \dots x_n$. At times it will be convenient to work with these points as a single matrix X, where the ith row of X is x_i.
- The low-dimensional representations that the dimensionality reduction algorithms find will be referred to as $y_1, y_2, \dots, y_n$. Y is the matrix of these points.
- n is the number of points in the input.
- D is the dimensionality of the input (*i.e.* $x_i \in \mathbb{R}^D$).
- d is the dimensionality of the manifold that the input is assumed to lie on and, accordingly, the dimensionality of the output (*i.e.* $y_i \in \mathbb{R}^d$).
- k is the number of nearest neighbors used by a particular algorithm.
- $N(i)$ denotes the set of the k-nearest neighbors of x_i.
- Throughout, we assume that the (eigenvector, eigenvalue) pairs are ordered by the eigenvalues. That is, if the (eigenvector, eigenvalue) pairs are (v_i, λ_i), for $i = 1, \dots, n$, then $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$. We refer to $v_1, \dots v_d$ as the *top d* eigenvectors, and $v_{n-d+1}, \dots v_n$ as the *bottom d* eigenvectors.

2 Preliminaries

In this section, we consider the motivations for manifold learning and introduce the basic concepts at work. We have already discussed the importance of dimensionality reduction for machine learning; we now look at a classical procedure for this task and then turn to more recent efforts.

2.1 Linear Dimensionality Reduction

Perhaps the most popular algorithm for dimensionality reduction is Principal Components Analysis (PCA). Given a data set, PCA finds the directions (vectors) along which the data has maximum variance in addition to the relative importance of these directions. For example, suppose that we feed a set of three-dimensional points that all lie on a two-dimensional plane to PCA. PCA will return two vectors that span the plane along with a third vector that is orthogonal to the plane (so that all of $\mathbb{R}^3$ is spanned by these vectors). The two vectors that span the plane will be given a positive weight, but the third vector will have a weight of zero, since the data does not vary along that direction. PCA is most useful in the case when data lies on or close to a linear subspace of the data set. Given this type of data, PCA will find a basis for the linear subspace and allow one to disregard the irrelevant features.

The manifold learning algorithms may be viewed as non-linear analogs to PCA, so it is worth detailing PCA somewhat. Suppose that $X \in \mathbb{R}^{n \times D}$ is a matrix whose rows are D-dimensional data points. We are looking for the d-dimensional linear subspace of $\mathbb{R}^D$ along which the data has maximum variance. If in fact the data lies perfectly along a subspace of $\mathbb{R}^D$, PCA will reveal that subspace; otherwise, PCA will introduce some error. The objective function it optimizes is

$$\max_V \operatorname{var}(XV),$$

where V is an orthogonal $D \times d$ matrix. If $d = 1$, then V is simply a unit-length vector which gives the direction of maximum variance. In general, the columns of V give the d dimensions that we project the original data onto. This cost function admits an eigenvalue-based solution, which we now detail for $d = 1$. The direction of maximum variation is given by a unit vector v to project the data points onto, which is found as follows:

$$\begin{aligned}
\max_{\|v\|=1} \mathrm{var}(Xv) &= \max_{\|v\|=1} \mathbf{E}(Xv)^2 - (\mathbf{E}Xv)^2 && (1)\\
&= \max_{\|v\|=1} \mathbf{E}(Xv)^2 && (2)\\
&= \max_{\|v\|=1} \sum_i ({x_i}^\top v)^2 && (3)\\
&= \max_{\|v\|=1} \sum_i (v^\top x_i)({x_i}^\top v) && (4)\\
&= \max_{\|v\|=1} v^\top \left(\sum_i (x_i {x_i}^\top) \right) v && (5)\\
&= \max_{\|v\|=1} v^\top X^\top X v. && (6)
\end{aligned}$$

Equality 2 follows from assuming the data is mean-centered, without loss of generality. The last line is a form of Rayleigh's Quotient and so is maximized by setting v to the eigenvector with the largest corresponding eigenvector. In general, the d-dimensional PCA solution is XV, where V is the $D \times d$ matrix whose columns are the top d eigenvectors (*i.e.* (V_i, λ_i) are (eigenvalue, eigenvector) pairs with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$).

Despite PCA's popularity it has a number of limitations. Perhaps the most blatant drawback is the requirement that the data lie on *linear* subspace. Returning the previous example, what if the plane was curled as it is in figure 1? Though the data is still intuitively two-dimensional, PCA will not correctly extract this two-dimensional structure. The problem is that this data set, commonly referred as the "swiss roll," is a two-dimensional manifold, not a two-dimensional subspace. Manifold learning algorithms essentially attempt to duplicate the behavior of PCA, but on manifolds instead of linear subspaces.

We now briefly review the concept of a manifold and formalize the manifold learning problem.

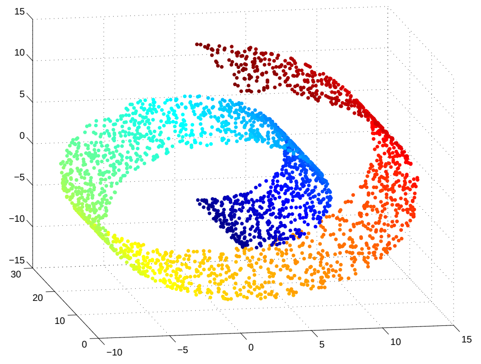

Figure 1: A curled plane: the swiss roll.

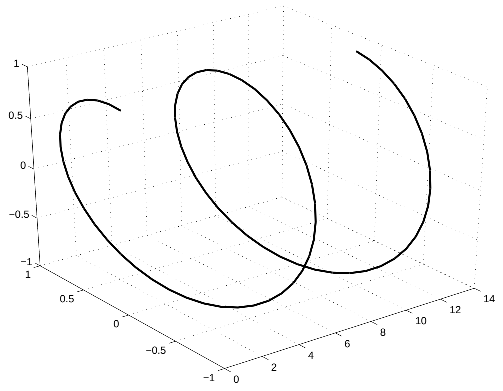

Figure 2: A one-dimensional manifold embedded in three dimensions.

2.2 Manifolds

Consider the curve shown in figure 2. Note that the curve is in $\mathbb{R}^3$, yet it has zero volume, and in fact zero area. The extrinsic dimensionality – three – is somewhat misleading since the curve can be parameterized by a single variable. One way of formalizing this intuition is via the idea of a *manifold*: the curve is a one-dimensional manifold because it locally "looks like" a copy of $\mathbb{R}^1$. Let us quickly review some basic terminology from geometry and topology in order to crystallize this notion of dimensionality.

Definition 1. *A* homeomorphism *is a continuous function whose inverse is also a continuous function.*

Definition 2. *A d-dimensional* manifold *M is set that is locally homeomorphic with $\mathbb{R}^d$. That is, for each $x \in M$, there is an open neighborhood around x, N_x, and a homeomorphism $f : N_x \to \mathbb{R}^d$. These neighborhoods are referred to as* coordinate patches*, and the map is referred to a a* coordinate chart*. The image of the coordinate charts is referred to as the* parameter space*.*

Manifolds are terrifically well-studied in mathematics and the above definition is extremely general. We will be interested only in the case where M is a subset of $\mathbb{R}^D$, where D is typically much larger than d. In other words, the manifold will lie in a high-dimensional space ($\mathbb{R}^D$), but will be homeomorphic with a low-dimensional space ($\mathbb{R}^d$, with $d < D$).

Additionally, all of algorithms we will look at require some smoothness requirements that further constrain the class of manifolds considered.

Definition 3. *A* smooth *(or differentiable) manifold is a manifold such that each coordinate chart is differentiable with a differentiable inverse (i.e., each coordinate chart is a diffeomorphism).*

We will be using the term *embedding* in a number of places, though our usage will be somewhat loose and inconsistent. An embedding of a manifold M into $\mathbb{R}^D$ is a smooth homeomorphism from M to a subset of $\mathbb{R}^D$. The algorithms discussed on this paper find embeddings of discrete point-sets, by which we simply mean a mapping of the point set into another space (typically lower-dimensional Euclidean space). An embedding of a dissimilarity matrix into $\mathbb{R}^d$, as discussed in section 3.1.2, is a configuration of points whose interpoint distances match those given in the matrix.

With this background in mind, we proceed to the main topic of this paper.

2.3 Manifold Learning

We have discussed the importance of dimensionality reduction and provided some intuition concerning the inadequacy of PCA for data sets lying along some non-flat surface; we now turn to the manifold learning approach to dimensionality reduction.

We are given data $x_1, x_2, \ldots, x_n \in \mathbb{R}^D$ and we wish to reduce the dimensionality of this data. PCA is appropriate if the data lies on a low-dimensional subspace of $\mathbb{R}^D$. Instead, we assume only that the data lies on a d-dimensional manifold embedded into $\mathbb{R}^D$, where $d < D$. Moreover, we assume that the manifold is given by a *single* coordinate chart.[1] We can now describe the problem formally.

Problem: Given points $x_1, \ldots, x_n \in \mathbb{R}^D$ that lie on a d-dimensional manifold M that can be described by a single coordinate chart $f : M \to \mathbb{R}^d$, find $y_1, \ldots, y_n \in \mathbb{R}^d$, where $y_i \overset{\text{def}}{=} f(x_i)$.

Solving this problem is referred to as manifold learning, since we are trying to "learn" a manifold from a set of points. A simple and oft-used example in the manifold learning literature is the swiss roll, a two-dimensional manifold embedded in $\mathbb{R}^3$. Figure 3 shows the swiss roll and a "learned" two-dimensional embedding of the manifold found using Isomap, an algorithm discussed later. Notice that points nearby in the original data set neighbor one another in the two-dimensional data set; this effect occurs because the chart f between these two graphs is a homeomorphism (as are all charts).

Though the extent to which "real-world" data sets exhibit low-dimensional manifold structure is still being explored, a number of positive examples have been found. Perhaps the most notable occurrence is in video and image data sets. For example, suppose we have a collection of frames taken from a video of a person rotating his or her head. The dimensionality of the data set is equal to the number of pixels in a frame, which is generally very large. However, the images are actually governed by only a couple of degrees of freedom (such as the angle of rotation). Manifold learning techniques have been successfully applied to a number of similar video and image data sets [Ple03].

3 Algorithms

In this section, we survey a number of algorithms proposed for manifold learning. Our treatment goes roughly in chronological order: Isomap and LLE were the first algorithms developed, followed by Laplacian

[1]All smooth, compact manifolds can be described by a single coordinate chart, so this assumption is not overly restrictive.

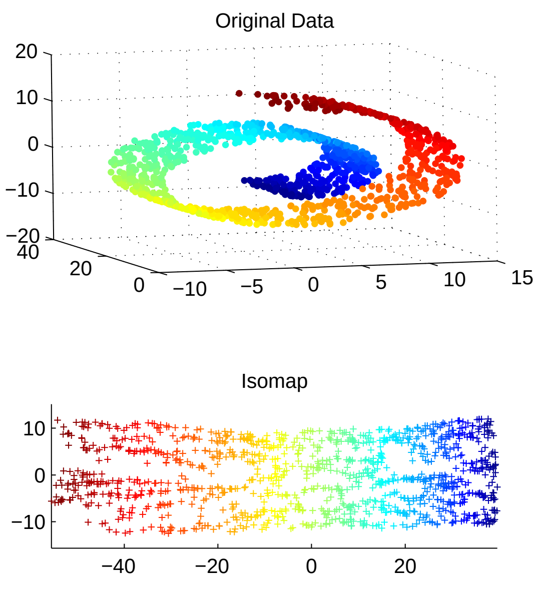

Figure 3: The swiss roll (top) and the learned two-dimensional representation (bottom).

Eigenmaps. Semidefinite Embedding and the variants of LLE were developed most recently.

Before beginning, we note that *every* algorithm that we will discuss requires a neighborhood-size parameter k. Every algorithm assumes that within each neighborhood the manifold is approximately flat. Typically, one simply runs an algorithm over a variety of choices of neighborhood size and compares the outputs to pick the appropriate k. Other neighborhood schemes, such as picking a neighborhood radius ϵ, are sometimes used in place of the k-nearest neighbor scheme.

3.1 Isomap

Isomap [TdL00] – short for isometric feature mapping – was one of the first algorithms introduced for manifold learning. The algorithm is perhaps the best known and most applied among the multitude of procedures now available for the problem at hand. It may be viewed as an extension to Multidimensional Scaling (MDS), a classical method for embedding dissimilarity information into Euclidean space. Isomap consists of two main steps:

1. Estimate the geodesic distances (distances along a manifold) between points in the input using shortest-path distances on the data set's k-nearest neighbor graph.

2. Use MDS to find points in low-dimensional Euclidean space whose interpoint distances match the distances found in step 1.

We consider each of these points in turn.

3.1.1 Estimating geodesic distances

Again, we are assuming that the data is drawn from a d-dimensional manifold embedded in $\mathbb{R}^D$ and we hope to find a chart back into $\mathbb{R}^d$. Isomap assumes further that there is an *isometric* chart —*i.e.* a chart that preserves the distances between points. That is, if x_i, x_j are points on the manifold and $G(x_i, x_j)$ is the geodesic distance between them, then there is a chart $f : M \to \mathbb{R}^d$ such that

$$\|f(x_i) - f(x_j)\| = G(x_i, x_j).$$

Furthermore, it is assumed that the manifold is smooth enough that the geodesic distance between nearby points is approximately linear. Thus, the Euclidean distance between nearby points in the high-dimensional data space is assumed to be a good approximation to the geodesic distances between these points.

For points that are distant in the high-dimensional space, the Euclidean distance between them is *not* a good estimate of the geodesic distance. The problem is that though the manifold is locally linear (at least approximately), this approximation breaks down as the distance between points increases. Estimating distances between distant points, thus, requires a different technique. To perform this estimation, the Isomap algorithm first constructs a k-nearest neighbor graph that is weighted by the Euclidean distances. Then, the algorithm runs a shortest-path algorithm (Dijkstra's or Floyd's) and uses its output as the estimates for the remainder of the geodesic distances.

3.1.2 Multidimensional Scaling

Once these geodesic distances are calculated, the Isomap algorithm finds points whose *Euclidean* distances equal these geodesic distances. Since the manifold is isometrically embedded, such points exist, and in fact, they are unique up to translation and rotation. Multidimensional Scaling [CC01] is a classical technique that may be used to find such points.

The basic problem it solves is this: given a matrix $D \in \mathbb{R}^{n\times n}$ of dissimilarities, construct a set of points whose interpoint Euclidean distances match those in D closely. There are numerous cost functions for this task and a variety of algorithms for minimizing these cost functions. We focus on *classical* MDS (cMDS), since that is what is used by Isomap.

The cMDS algorithm, shown in figure 4, is based on the following theorem which establishes a connection between the space of Euclidean distance matrices and the space of Gram matrices (inner-product matrices).

Theorem 1. *A nonnegative symmetric matrix $D \in \mathbb{R}^{n\times n}$, with zeros on the diagonal, is a Euclidean distance matrix if and only if $B \overset{\text{def}}{=} -\frac{1}{2}HDH$, where $H \overset{\text{def}}{=} I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, is positive semidefinite. Furthermore, this B will be the Gram matrix for a mean-centered configuration with interpoint distances given by D.*

A proof of this theorem is given in the appendix. This theorem gives a simple technique to convert a Euclidean distance matrix D into an appropriate Gram matrix B. Let X be the configuration that we are after. Since B is its Gram matrix, $B = XX^\top$. To get X from B, we spectrally decompose B into $U\Lambda U^\top$. Then, $X = U\Lambda^{1/2}$.

What if D is not Euclidean? In the application of Isomap, for example, D will generally not be perfectly Euclidean since we are only able to approximate the geodesic distances. In this case, B, as described in the above theorem, will not be positive semidefinite and thus will not be a Gram matrix. To handle this case, cMDS projects B onto the cone of positive semidefinite matrices by setting its negative eigenvalues to 0 (step 3).

Finally, users of cMDS often want a low-dimensional embedding. In general, $X := U\Lambda_+^{1/2}$ (step 4) will be an n-dimensional configuration. This dimensionality is reduced to d in step 5 of the algorithm by simply removing the $n-d$ trailing columns of X. Doing so is equivalent to projecting X onto its top d principal components, as we now demonstrate. Let $X \overset{\text{def}}{=} U\Lambda_+^{1/2}$ be the n-dimensional configuration found in step 4. Then, $U\Lambda_+U^\top$ is the spectral decomposition of $XX^\top$. Suppose that we use PCA to project X onto d dimensions. Then, PCA returns XV, where $V \in \mathbb{R}^{n\times d}$ and the columns of V are given by the eigenvectors of $X^\top X$:$v_1, \ldots, v_d$. Let us compute these eigenvectors.

$$\begin{aligned} X^\top X v_i &= \xi_i v_i \\ (U^\top \Lambda_+^{1/2})^\top (U\Lambda_+^{1/2}) v_i &= \xi_i v_i \\ (\Lambda_+^{1/2} U^\top U \Lambda_+^{1/2}) v_i &= \xi_i v_i \\ \Lambda_+ v_i &= \xi_i v_i. \end{aligned}$$

Thus, $v_i = e_i$, where e_i is the ith standard basis vector for $\mathbb{R}^n$, and $\xi_i = 1$. Then, $XV = X[e_1 \cdots e_d] = [X]_{n\times d}$, so the truncation performed in step 5 is indeed equivalent to projecting X onto its first d principal components.

This equivalence is useful because if a set of dissimilarities can be realized in d-dimensions, then all but the first d columns of X will be zeros; moreover, Λ_+ reveals the relative importance of each dimension, since these eigenvalues are the same as those found by PCA. In other words, classical MDS automatically finds the dimensionality of the configuration associated with a set of dissimilarities, where dimensionality refers to the dimensionality of the subspace along which X lies. Within the context of Isomap, classical MDS automatically finds the dimensionality of the parameter space of the manifold, which is the dimensionality of the manifold.

Classical MDS finds the optimal d-dimensional configuration of points for the cost function $\|HDH - HD^*H\|_F^2$, where D^* runs over all Euclidean distance matrices for d-dimensional configurations. Unfortunately, this cost function makes cMDS tremendously sensitive to noise and outliers; moreover, the problem is NP-hard for a wide variety of more robust cost functions [CD05].

```
classical Multidimensional Scaling
input: $D \in \mathbb{R}^{n\times n}(D_{ii} = 0,\ D_{ij} \geq 0),\ d \in \{1, \ldots, n\}$
  1. Set $B := -\frac{1}{2}HDH$, where $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ is the centering matrix.
  2. Compute the spectral decomposition of $B$: $B = U\Lambda U^\top$.
  3. Form $\Lambda_+$ by setting $[\Lambda_+]_{ij} := \max\{\Lambda_{ij}, 0\}$.
  4. Set $X := U\Lambda_+^{1/2}$.
  5. Return $[X]_{n\times d}$.
```

Figure 4: Classical MDS.

3.1.3 Putting it together

The Isomap algorithm consists of estimating geodesic distances using shortest-path distances and then finding an embedding of these distances in Euclidean space using cMDS. Figure 5 summarizes the algorithm.

One particularly helpful feature of Isomap – not found in some of the other algorithms – is that it automatically provides an estimate of the dimensionality of the underlying manifold. In particular, the number of non-zero eigenvalues found by cMDS gives the underlying dimensionality. Generally, however, there will be some noise and uncertainty in the estimation process, so one looks for a tell-tale gap in the spectrum to determine the dimensionality.

Unlike many of the algorithms to be discussed, Isomap has an optimality guarantee [BdSLT00]; roughly, Isomap is guaranteed to recover the parameterization of a manifold under the following assumptions.

1. The manifold is isometrically embedded into $\mathbb{R}^D$.
2. The underlying parameter space is convex —*i.e.* the distance between any two points in the parameter space is given by a geodesic that lies entirely within the parameter space. Intuitively, the parameter space of the manifold from which we are sampling cannot contain any holes.
3. The manifold is well sampled everywhere.
4. The manifold is compact.

The proof of this optimality begins by showing that, with enough samples, the graph distances approach the underlying geodesic distances. In the limit, these distances will be perfectly Euclidean and cMDS will return the correct low-dimensional points. Moreover, cMDS is continuous: small changes in the input will result in small changes in the output. Because of this robustness to perturbations, the solution of cMDS will converge to the correct parameterization in the limit.

Isomap has been extended in two major directions: to handle conformal mappings and to handle very large data sets [dST03]. The former extension is referred to as C-Isomap and addresses the issue that many embeddings preserve angles, but not distances. C-Isomap has a theoretical optimality guarantee similar to that of Isomap, but requires that the manifold be uniformly sampled.

The second extension, Landmark Isomap, is based on using landmark points to speed up the calculation of interpoint distances and cMDS. Both of these steps require $O(n^3)$ timesteps[2], which is prohibitive for large data sets. The basic idea is to select a small set of l landmark points, where $l > d$, but $l \ll n$. Next, the approximate geodesic distances from each point to the landmark points are computed (in contrast to computing *all* $n \times n$ interpoint distances, only $n \times l$ are computed). Then, cMDS is run on the $l \times l$ distance matrix for the landmark points. Finally, the remainder of the points are located using a simple formula based on their distances to the landmarks. This technique brings the time complexity of the two computational bottlenecks of Isomap down considerably: the time complexity of L-Isomap is $O(dln \log n + l^3)$ instead of $O(n^3)$.

[2]The distance calculation can be brought down to $O(kn^2 \log n)$ using Fibonacci heaps.

Isomap
input: $x_1, \ldots, x_n \in \mathbb{R}^D$, k

1. Form the k-nearest neighbor graph with edge weights $W_{ij} := \|x_i - x_j\|$ for neighboring points x_i, x_j.
2. Compute the shortest path distances between all pairs of points using Dijkstra's or Floyd's algorithm. Store the squares of these distances in D.
3. Return $Y :=$cMDS(D).

Figure 5: Isomap.

3.2 Locally Linear Embedding

Locally Linear Embedding (LLE) [SR03] was introduced at about the same time as Isomap, but is based on a substantially different intuition. The idea comes from visualizing a manifold as a collection of overlapping coordinate patches. If the neighborhood sizes are small and the manifold is sufficiently smooth, then these patches will be approximately linear. Moreover, the chart from the manifold to $\mathbb{R}^d$ will be roughly linear on these small patches. The idea, then, is to identify these linear patches, characterize the geometry of them, and find a mapping to $\mathbb{R}^d$ that preserves this local geometry and is nearly linear. It is assumed that these local patches will overlap with one another so that the local reconstructions will combine into a global one.

As in all of these methods, the number of neighbors chosen is a parameter of the algorithm. Let $N(i)$ be the set of the k-nearest neighbors of point x_i. The first step of the algorithm models the manifold as a collection of linear patches and attempts to characterize the geometry of these linear patches. To do so, it attempts to represent x_i as a weighted, convex combination of its nearest-neighbors. The weights are chosen to minimize the following squared error for each i:

$$\left\| x_i - \sum_{j \in N(i)} W_{ij} x_j \right\|^2 .$$

The weight matrix W is used as a surrogate for the local geometry of the patches. Intuitively, W_i reveals the layout of the points around x_i. There are a couple of constraints on the weights: each row must sum to one (equivalently, each point is represented as a convex combination of its neighbors) and $W_{ij} = 0$ if $j \notin N(i)$. The second constraint reflects that LLE is a *local* method; the first makes the weights invariant to global translations: if each x_i is shifted by α then

$$\left\| x_i + \alpha - \sum_{j \in N(i)} W_{ij}(x_j + \alpha) \right\|^2 = \left\| x_i - \sum_{j \in N(i)} W_{ij} x_j \right\|^2 ,$$

so W is unaffected. Moreover, W is invariant to global rotations and scalings.

Fortunately, there is a closed form solution for W, which may be derived using Lagrange multipliers. In particular, the reconstruction weights for each point x_i are given by

$$\tilde{W}_i = \frac{\sum_k C^{-1}_{jk}}{\sum_{lm} C^{-1}_{lm}},$$

where C is the local covariance matrix with entries $C_{jk} \stackrel{\text{def}}{=} (x_i - \eta_j)^\top (x_i - \eta_k)$ and η_j, η_k are neighbors of x_i. $\tilde{W} \in \mathbb{R}^{n \times k}$ is then transformed into the sparse $n \times n$ matrix W; $W_{ij} = \tilde{W}_{il}$ if x_j is the lth neighbor of x_i, and is 0 if $j \notin N(i)$.

W_i is a characterization of the local geometry around x_i of the manifold. Since it is invariant to rotations, scalings, and translations, W_i will also characterize the local geometry around x_i for any linear mapping of the neighborhood patch surround x_i. We expect that the chart from this neighborhood patch to the parameter space should be approximately linear since the manifold is smooth. Thus, W_i should capture the local geometry of the parameter space as well. The second step of the algorithm finds a configuration in d-dimensions (the dimensionality of the parameter space) whose local geometry is characterized well by W. Unlike with Isomap, d must be known *a priori* or estimated. The d-dimensional configuration is found by minimizing

$$\sum_i \left\| y_i - \sum_j W_{ij} y_j \right\|^2 \qquad (7)$$

with respect to $y_1, \dots, y_n \in \mathbb{R}^d$. Note that this expression has the same form as the error minimized in the first step of the algorithm, but that we are now minimizing with respect to $y_1, \dots, y_n$, rather than W. The cost function (7) may be rewritten as

$$Y^\top M Y,$$

where

$$M_{ij} \stackrel{\text{def}}{=} \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj},$$

and $\delta_{ij} \stackrel{\text{def}}{=} 1$ if $i = j$ and 0 otherwise. M may be written more concisely as $(I - W)^\top (I - W)$. There are a couple of constraints on Y. First, $Y^\top Y = I$, which forces the solution to be of rank d. Second, $\sum_i Y_i = \mathbf{0}$; this constraint centers the embedding on the origin. This cost function is a form of Rayleigh's quotient, so is minimized by setting the columns of Y to the bottom d eigenvectors of M. However, the bottom (eigenvector, eigenvalue) pair is $(\mathbf{1}, 0)$, so the final coordinate of each point is identical. To avoid this degenerate dimension, the d-dimensional embedding is given by the bottom *non-constant* d eigenvectors.

Conformal Eigenmaps [SS05] is a recent technique that may be used in conjunction with LLE to provide an automatic estimate of d, the dimensionality of the manifold. LLE is summarized in figure 6.

Locally Linear Embedding
input: $x_1, \dots, x_n \in \mathbb{R}^D$, d, k

1. *Compute reconstruction weights.* For each point x_i, set
$$W_i := \frac{\sum_k C^{-1}_{jk}}{\sum_{lm} C^{-1}_{lm}}.$$
2. *Compute the low-dimensional embedding.*
    - Let U be the matrix whose columns are the eigenvectors of $(I - W)^\top (I - W)$ with nonzero accompanying eigenvalues.
    - Return $Y := [U]_{n \times d}$.

Figure 6: Locally Linear Embedding.

3.3 Laplacian Eigenmaps

The Laplacian Eigenmaps [BN01] algorithm is based on ideas from spectral graph theory. Given a graph and a matrix of edge weights, W, the graph Laplacian is defined as $L \stackrel{\text{def}}{=} D - W$, where D is the diagonal matrix with elements $D_{ii} = \sum_j W_{ij}$.[3] The eigenvalues and eigenvectors of the Laplacian reveal a wealth of information about the graph such as whether it is complete or connected. Here, the Laplacian will be exploited to capture local information about the manifold.

We define a "local similarity" matrix W which reflects the degree to which points are near to one another. There are two choices for W:

1. $W_{ij} = 1$ if x_j is one of the k-nearest neighbors of x_i and equals 0 otherwise.
2. $W_{ij} = e^{-\|x_i - x_j\|^2/2\sigma^2}$ for neighboring nodes; 0 otherwise. This is the Gaussian heat kernel, which has an interesting theoretical justification given in [BN03]. On the other hand, using the Heat kernel requires manually setting σ, so is considerably less convenient than the simple-minded weight scheme.

We use this similarity matrix W to find points $y_1, \dots, y_n \in \mathbb{R}^d$ that are the low-dimensional analogs of $x_1, \dots, x_n$. Like with LLE, d is a parameter of the algorithm, but it can be automatically estimated by using Conformal Eigenmaps as a post-processor. For simplicity, we first derive the algorithm for $d = 1$ —*i.e.* each y_i is a scalar.

Intuitively, if x_i and x_k have a high degree of similarity, as measured by W, then they lie very near to one another on the manifold. Thus, y_i and y_j should be near to one another. This intuition leads to the following cost function which we wish to minimize:

$$\sum_{ij} W_{ij}(y_i - y_j)^2. \tag{8}$$

This function is minimized by the setting $y_1 = \dots = y_n = 0$; we avoid this trivial solution by enforcing the constraint

$$y^\top D y = 1. \tag{9}$$

This constraint effectively fixes a scale at which the y_i's are placed. Without this constraint, setting $y_i' = \alpha y_i$, with any constant $\alpha < 1$, would give a configuration with lower cost than the configuration $y_1, \dots, y_n$. Note that we could achieve the same result with the constraint $y^\top y = 1$, however, using (9) leads to a simple eigenvalue solution.

[3] L is referred to as the *unnormalized* Laplacian in many places. The normalized version is $D^{-1/2}LD^{-1/2}$.

Laplacian Eigenmaps
input: $x_1 \dots x_n \in \mathbb{R}^D$, d, k, σ.

1. Set $W_{ij} = \begin{cases} e^{-\|x_i - x_j\|^2/2\sigma^2} & \text{if } x_j \in N(i) \\ 0 & \text{otherwise.} \end{cases}$
2. Let U be the matrix whose columns are the eigenvectors of $L\mathbf{y} = \lambda D\mathbf{y}$ with nonzero accompanying eigenvalues.
3. Return $Y := [U]_{n \times d}$.

Figure 7: Laplacian Eigenmaps.

Using Lagrange multipliers, it can be shown that the bottom eigenvector of the generalized eigenvalue problem $L\mathbf{y} = \lambda D\mathbf{y}$ minimizes 8. However, the bottom (eigenvalue,eigenvector) pair is $(0, \mathbf{1})$, which is yet another trivial solution (it maps each point to 1). Avoiding this solution, the embedding is given by the eigenvector with smallest non-zero eigenvalue.

We now turn to the more general case, where we are looking for d-dimensional representatives $y_1, \dots, y_n$ of the data points. We will find a $n \times d$ matrix Y, whose rows are the embedded points. As before, we wish to minimize the distance between similar points; our cost function is thus

$$\sum_{ij} W_{ij}\|y_i - y_j\|^2 = \operatorname{tr}(Y^\top L Y). \tag{10}$$

In the one-dimensional case, we added a constraint to avoid the trivial solution of all zeros. Analogously, we must constrain Y so that we do not end up with a solution with dimensionality less than d. The constraint $Y^\top D Y = I$ forces Y to be of full dimensionality; we could use $Y^\top Y = I$ instead, but $Y^\top D Y = I$ is preferable because it leads to a closed-form eigenvalue solution. Again, we may solve this problem via the generalized eigenvalue problem $L\mathbf{y} = \lambda D\mathbf{y}$. This time, however, we need the bottom d non-zero eigenvalues and eigenvectors. Y is then given by the matrix whose columns are these d eigenvectors.

The algorithm is summarized in figure 7.

3.3.1 Connection to LLE

Though LLE is based on a different intuition than Laplacian Eigenmaps, Belkin and Niyogi [BN03] were able to show that that the algorithms were equivalent in some situations. The equivalence rests on another motivation for Laplacian Eigenmaps, not discussed here, that frames the algorithm as a discretized version of finding the eigenfunctions[4] of the Laplace-Beltrami operator, $\mathcal{L}$, on a manifold. The authors demonstrate that LLE, under certain assumptions, is finding the eigenfunctions of $\frac{1}{2}\mathcal{L}^2$, which are the same as the eigenfunctions of $\mathcal{L}$. In the proof, they assume that the neighborhood patches selected by LLE are perfectly locally linear (*i.e.*, the neighbors of a point x actually lie on the tangent plane at x). Furthermore, the equivalence is only shown in expectation over locally uniform measures on the manifold.

The connection is an intriguing one, but has not been exploited or even demonstrated on any real data sets. It is not clear how reasonable the assumptions are. These two algorithms are still considered distinct, with different characteristics.

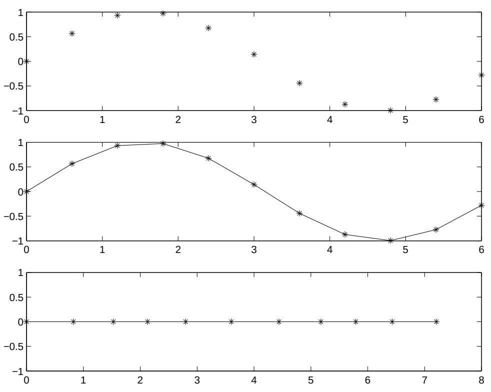
Figure 8: An illustration of the idea behind SDE. The top figure is a few points sampled from a sine curve. The middle figure shows the result of connecting each point to its two neighbors. The bottom shows the result of the "pulling" step.

3.4 Semidefinite Embedding

Semidefinite embedding (SDE) [WSS04], yet another algorithm for manifold learning, is based on a physical intuition. Imagine that each point is connected to its nearest neighbors with a rigid rod. Now, we take this structure and pull it as far apart as possible –*i.e.* we attempt to maximize the distance between points that are *not* neighbors. If the manifold looks like a curved version of the parameter space, then hopefully this procedure will unravel it properly.

To make this intuition a little more concrete, let us look at a simple example. Suppose we sample some points from a sine curve in $\mathbb{R}^2$. The sine curve is a one-dimensional manifold embedded into $\mathbb{R}^2$, so let us apply the SDE idea to it to try to find a one-dimensional representation. We first attach each point to its two nearest neighbors. We then pull apart the resulting structure as far as possible. Figure 8 shows the steps of the procedure. As the figure shows, we have successfully found a one-dimensional representation of the data set. On the other hand, if the neighbors were chosen a bit differently, the procedure would have failed.

We this simple example in mind, we proceed to describe the algorithm more precisely. The SDE algorithm uses a *semidefinite program* [VB96], which is essentially a linear program with additional constraints that force the variables to form a positive semidefinite matrix. Recall that a matrix is a Gram matrix (an inner product matrix) if and only if it is positive semidefinite. The SDE algorithm uses a semidefinite program to find an appropriate Gram matrix, then extracts a configuration from the Gram matrix using the same trick employed in classical MDS.

The primary constraint of the program is that distances between neighboring points remain the same –*i.e* if x_i and x_j are neighbors then $\|y_i - y_j\| = \|x_i - x_j\|$, where y_i and y_j are the low-dimensional representatives of x_i and x_j. However, we must express this constraint in terms of the Gram matrix, B, since this is the variable the semidefinite program works with (and not Y). Translating the constraint is simple enough, though, since

$$\begin{aligned}\|y_i - y_j\|^2 &= \|y_i\|^2 + \|y_j\|^2 - 2\langle y_i, y_j\rangle \\ &= \langle y_i, y_i\rangle + \langle y_j, y_j\rangle - 2\langle y_i, y_j\rangle \\ &= B_{ii} + B_{jj} - 2B_{ij}.\end{aligned}$$

The constraint, then is

$$B_{ii} + B_{jj} - 2B_{ij} = \|x_i - x_j\|^2$$

for all neighboring points x_i, x_j.

We wish to "pull" the remainder of the points as far apart as possible, so we maximize the interpoint distances of our configuration subject to the above constraint. The objective function is:

$$\max \sum_{i,j} \|y_i - y_j\|^2 \quad = \quad \max \sum_{i,j} B_{ii} + B_{jj} - 2B_{ij}$$

[4]In the algorithm presented, we are looking for eigenvectors. The eigenvectors are the eigenfunctions evaluated at the data points.

Semidefinite Embedding

```
input: $x_1, \ldots, x_n \in \mathbb{R}^D$, $k$
  1. Solve the following semidefinite program:
       maximize      $\mathrm{tr}(B)$
       subject to:   $\sum_{ij} B_{ij} = 0$;
                     $B_{ii} + B_{jj} - 2B_{ij} = \|x_i - x_j\|^2$
                         (for all neighboring $x_i, x_j$);
                     $B \succeq 0$.
  2. Compute the spectral decomposition of $B$: $B = U\Lambda U^\top$.
  3. Form $\Lambda_+$ by setting $[\Lambda_+]_{ij} := \max\{\Lambda_{ij}, 0\}$.
  4. Return $Y := U\Lambda_+^{1/2}$.
```

Figure 9: Semidefinite Embedding.

$$
\begin{aligned}
&= \max \sum_{i,j} B_{ii} + B_{jj} \\
&= \max \mathrm{tr}(B)
\end{aligned}
$$

The second equality follows from a regularization constraint: since the interpoint distances are invariant to translation, there is an extra degree of freedom in the problem which we remove by enforcing

$$\sum_{ij} B_{ij} = 0.$$

The semidefinite program may be solved by using any of a wide variety of software packages. The second part of the algorithm is identical to the second part of classical MDS: the eigenvalue decomposition $U\Lambda U^\top$ is computed for B and the new coordinates are given by $U\Lambda^{1/2}$. Additionally, the eigenvalues reveal the underlying dimensionality of the manifold. The algorithm is summarized in figure 9.

One major gripe with SDE is that solving a semidefinite program requires tremendous computational resources. State-of-the-art semidefinite programming packages cannot handle an instance with $n \geq 2000$ or so on a powerful desktop machine. This is a a rather severe restriction since real data sets often contain many more than 2000 points. One possible approach to solving larger instances would be to develop a *projected subgradient* algorithm [Ber99]. This technique resembles gradient descent, but can be used to handle non-differentiable convex functions. This approach has been exploited for a similar problem: finding robust embeddings of dissimilarity information [CD05]. This avenue has not been pursued for SDE.

Instead, there is an approximation known as ℓSDE [WPS05], which uses matrix factorization to keep the size of the semidefinite program down. The idea is to factor the Gram matrix B into $QLQ^\top$, where L is a $l \times l$ Gram matrix of *landmark* points and $l \ll n$. The semidefinite program only finds L and then the rest of the points are located with respect to the landmarks. This method was a full two orders of magnitude faster than SDE in some experiments [WPS05]. This approximation to SDE appears to produce high-quality results, though more investigation is needed.

3.5 Other algorithms

We have looked at four algorithms for manifold learning, but there are certainly more. In this section, we briefly discuss two other algorithms.

As discussed above, both the LLE and Laplacian Eigenmaps algorithms are Laplacian-based algorithms. Hessian LLE (hLLE) [DG03] is closely related to these algorithms, but replaces the graph Laplacian with a Hessian[5] estimator. Like the LLE algorithm, hLLE looks at locally linear patches of the manifold and attempts to map them to a low-dimensional space. LLE models these patches by representing each x_i as a weighted combination of its neighbors. In place of this technique, hLLE runs principal components analysis on the neighborhood set to get a estimate of the tangent space at x_i (*i.e.* the linear patch surrounding x_i). The second step of LLE maps these linear patches to linear patches in $\mathbb{R}^d$ using the weights found in the first step to guide the process. Similarly, hLLE maps the tangent space estimates to linear patches in $\mathbb{R}^d$, warping the patches as little as possible. The "warping" factor is the Frobenius norm of the Hessian matrix. The mapping with minimum norm is found by solving an eigenvalue problem.

The hLLE algorithm has a fairly strong optimality guarantee – stronger, in some sense, then that of Isomap. Whereas Isomap requires that the parameter space be convex and the embedded manifold be globally isometric to the parameter space, hLLE requires only that the parameter space be connected and *locally* isometric to the manifold. In these cases hLLE is guaranteed to asymptotically recover the parameterization. Additionally, some empirical evidence supporting this theoretical claim has been given: Donoho and Grimes [DG03] demonstrate that

[5] Recall that the Hessian of a function is the matrix of its partial second derivatives.

hLLE outperforms Isomap on a variant of the swiss roll data set where a hole has been punched out of the surface. Though its theoretical guarantee and some limited empirical evidence make the algorithm a strong contender, hLLE has yet to be widely used, perhaps partly because it is a bit more sophisticated and difficult to implement than Isomap or LLE. It is also worth pointing out the theoretical guarantees are of a slightly different nature than those of Isomap: whereas Isomap has some finite sample error bounds, the theorems pertaining to hLLE hold only in the continuum limit. Even so, the hLLE algorithm is a viable alternative to many of the algorithms discussed.

Another recent algorithm proposed for manifold learning is Local Tangent Space Alignment (LTSA) [ZZ04]. Similar to hLLE, the LTSA algorithm estimates the tangent space at each point by performing PCA on its neighborhood set. Since the data is assumed to lie on a d-dimensional manifold, and be locally linear, these tangent spaces will admit a d-dimensional basis. These tangent spaces are aligned to find global coordinates for the points. The alignment is computed by minimizing a cost function that allows any linear transformation of each local coordinate space. LTSA is a relatively recent algorithm that has not find widespread popularity yet; as such, its relative strengths and weaknesses are still under investigation.

There are a number of other algorithms in addition the ones described here. We have touched on those algorithms that have had the greatest impact on developments in the field thus far.

4 Comparisons

We have discussed a myriad of different algorithms, all of which are for roughly the same task. Choosing between these algorithms is a difficult task; there is no clear "best" one. In this section, we discuss the trade-offs associated with each algorithm.

4.1 Embedding type

One major distinction between these algorithms is the type of embedding that they compute. Isomap, Hessian LLE, and Semidefinite Embedding all look for isometric embeddings —*i.e.* they all assume that there is a coordinate chart from the parameter space to the high-dimensional space that preserves interpoint distances and attempt to uncover this chart. In contrast, LLE (and the C-Isomap variant of Isomap) look for conformal mappings – mappings which preserve local angles, but not necessarily distances. It remains open whether the LLE algorithm actually can recover such mappings correctly. It is unclear exactly what type of embedding the Laplacian Eigenmaps algorithm computes [BN03]; it has been dubbed "proximity preserving" in places [WS04]. This breakdown is summarized in the following table.

Algorithm	**Mapping**
C-Isomap	conformal
Isomap	isometric
Hessian LLE	isometric
Laplacian Eigenmaps	unknown
Locally Linear Embedding	conformal
Semidefinite Embedding	isometric

This comparison is not particularly practical, since the form of embedding for a particular data set is usually unknown; however, if, say, an isometric method fails, then perhaps a conformal method would be a reasonable alternative.

4.2 Local versus global

Manifold learning algorithms are commonly split into two camps: local methods and global methods. Isomap is considered a global method because it constructs an embedding derived from the geodesic distance between *all* pairs of points. LLE is considered a local method because the cost function that it uses to construct an embedding only considers the placement of each point with respect to its neighbors. Similarly, Laplacian Eigenmaps and the derivatives of LLE are local methods. Semidefinite embedding, on the other hand, straddles the local/global division: it is a local method in that intra-neighborhood distances are equated with geodesic distances, but a global method in that its objective function considers distances between all pairs of points. Figure 10 summarizes the breakdown of these algorithms.

The local/global split reveals some distinguishing characteristics between algorithms in the two categories. Local methods tend to characterize the local geometry of manifolds accurately, but break down at the global level. For instance, the points in the embedding found by local methods tend be well placed with respect to their neighbors, but non-neighbors may be much nearer to one another than they are on the manifold. The reason for these aberrations is simple: the constraints on the local methods do

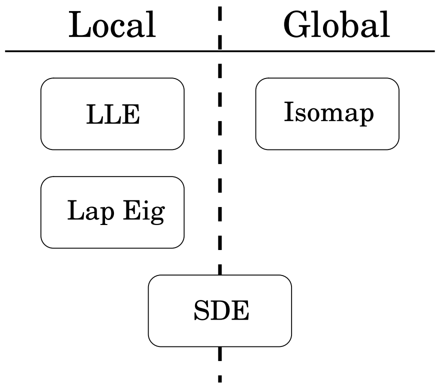

Figure 10: The local/global split.

not pertain to non-local points. The behavior for Isomap is just the opposite: the intra-neighborhood distances for points in the embedding tend to be inaccurate whereas the large interpoint distances tend to be correct. Classical MDS creates this behavior; its cost function contains fourth-order terms of the distances, which exacerbates the differences in scale between small and large distances. As a result, the emphasis of the cost function is on the large interpoint distances and the local topology of points is often corrupted in the embedding process. SDE behaves more like the local methods in that the distances between neighboring points are guaranteed to be the same before and after the embedding.

The local methods also seem to handle sharp curvature in manifolds better than Isomap. Additionally, shortcut edges – edges between points that should not actually be classified as neighbors – can have a potentially disastrous effect in Isomap. A few shortcut edges can badly damage all of geodesic distances approximations. In contrast, this type of error does not seem to propagate through the entire embedding with the local methods.

4.3 Time complexity

The primary computational bottleneck for all of these algorithms – except for semidefinite embedding – is a spectral decomposition. Computing this decomposition for a dense $n \times n$ matrix requires $O(n^3)$ time steps. For sparse matrices, however, a spectral decomposition can be performed much more quickly. The global/local split discussed above turns out to be a dense/sparse split as well.

First, we look at the local methods. LLE builds a matrix of weights $W \in \mathbb{R}^{n \times n}$ such that $W_{ij} \neq 0$ only if j is one of i's nearest neighbors. Thus, W is very sparse: each row contains only k nonzero elements and k is usually small compared to the number of points. LLE performs a sparse spectral decomposition of $(I-W)^\top(I-W)$. Moreover, this matrix contains some additional structure that may be used to speed up the spectral decomposition [SR03]. Similarly, the Laplacian Eigenmaps algorithm finds the eigenvectors of the generalized eigenvalue problem $Ly = \lambda Dy$. Recall that $L \overset{\text{def}}{=} D - W$, where W is a weight matrix and D is the diagonal matrix of row-sums of W. All three of these matrices are sparse: W only contains k nonzero entries per row, D is diagonal, and L is their difference.

The two LLE variants described, Hessian LLE and Local Tangent Space Alignment, both require one spectral decomposition per point; luckily, the matrices to be decomposed only contain $n \times k$ entries, so the decompositions requires only $O(k^3)$ time steps. Both algorithms also require a spectral decomposition of an $n \times n$ matrix, but these matrices are sparse.

For all of these local methods, the spectral decompositions scale, pessimistically, with $O((d+k)n^2)$, which is a substantial improvement over $O(n^3)$.

The spectral decomposition employed by Isomap, on the other hand, is not sparse. The actual decomposition, which occurs during the cMDS phase of the algorithm, is of $B \overset{\text{def}}{=} -\frac{1}{2}HDH$, where D is the matrix of approximate interpoint square geodesic distances, and H is a constant matrix. The matrix $B \in \mathbb{R}^{n \times n}$ only has rank approximately equal to the dimensionality of the manifold, but is nevertheless dense since D has zeros only on the diagonal. Thus the decomposition requires a full $O(n^3)$ timesteps. This complexity is prohibitive for large data sets. To manage large data sets, Landmark Isomap was created. The overall time complexity of Landmark Isomap is the much more manageable $O(dln \log n + l^3)$, but the solution is only an approximate one.

Semidefinite embedding is by far the most computationally demanding of these methods. It also requires a spectral decomposition of a dense $n \times n$ matrix, but the real bottleneck is the semidefinite program. Semidefinite programming methods are still relatively new and, perhaps as a result, quite slow. The theoretical time complexity of these procedures varies, but standard contemporary packages cannot handle an instance of the SDE program if the number of points is greater than 2000 or so. Smaller instances can be handled, but often require an enormous amount of time. The ℓSDE approximation is a fast alternative to SDE that has been shown to yield a speedup of two orders of magnitude in some experiments [WPS05]. There has been no work relating the time requirements of ℓSDE back to other manifold learning algorithms yet.

To summarize: the local methods – LLE, Laplacian Eigenmaps, and variants – scale well up to very large data sets; Isomap can handle medium size data sets and the Landmark approximation may be used for large data sets; SDE can handle only fairly small instances, though its approximation can tackle medium to large instances.

4.4 Other issues

One shortcoming of Isomap, first pointed out as a theoretical issue in [BdSLT00] and later examined empirically, is that it requires that the underlying parameter space be convex. Without convexity, the geodesic estimates will be flawed. Many manifolds do not have convex parameter spaces; one simple example is given by taking the swiss roll and punching a hole out of it. More importantly, [DG02] found that many image manifolds do not have this property. None of the other algorithms seem to have this shortcoming.

The only algorithms with theoretical optimality guarantees are Hessian LLE and Isomap. Both of these guarantees are asymptotic ones, though, so it is questionable how useful they are. On the other hand, these guarantees at least demonstrate that hLLE and Isomap are conceptually correct.

Isomap and SDE are the only two algorithms that automatically provide an estimate of the dimensionality of the manifold. The other algorithms requires the target dimensionality to be know *a priori* and often produce substantially different results for different choices of dimensionality. However, the recent Conformal Eigenmaps algorithm provides a patch with which to automatically estimate the dimensionality when using Laplacian Eigenmaps, LLE, or its variants.

5 What's left to do?

Though manifold learning has become an increasingly popular area of study within machine learning and related fields, much is still unknown. One issue is that all of these algorithms require a neighborhood size parameter k. The solutions found are often very different depending on the choice of k, yet there is little work on actually choosing k. One notable exception is [WZZ04], but these techniques are discussed only for LTSA.

Perhaps the most curious aspect of the field is that there are already so many algorithms for the same task, yet new ones continue to be developed. In this paper, we have discussed a total of nine algorithms, but our coverage was certainly not exhaustive. Why are there so many?

The primary reason for the abundance of algorithms is that evaluating these algorithms is difficult. Though there are a couple of theoretical performance results for these algorithms, the primary evaluation methodology has been to run the algorithm on some (often artificial) data sets and see whether the results are intuitively pleasing. More rigorous evaluation methodologies remain elusive. Performing an experimental evaluation requires knowing the underlying manifolds behind the data and then determining whether the manifold learning algorithm is preserving useful information. But, determining whether a real-world data set actually lies on a manifold is probably as hard as learning the manifold; and, using an artificial data set may not give realistic results. Additionally, there is little agreement on what constitutes "useful information" – it is probably heavily problem-dependent. This uncertainty stems from a certain slack in the problem: though we want to find a chart into low-dimensional space, there may be many such charts, some of which are more useful than others. Another problem with experimentation is that one must somehow select a realistic cross-section of manifolds.

Instead of an experimental evaluation, one might establish the strength of an algorithm theoretically. In many cases, however, algorithms used in machine learning do not have strong theoretical guarantees associated with them. For example, Expectation-Maximization can perform arbitrarily poorly on a data set, yet it is a tremendously useful and popular algorithm. A good manifold learning algorithm may well have similar behavior. Additionally, the type of theoretical results that are of greatest utility are finite-sample guarantees. In contrast, the algorithmic guarantees for hLLE and Isomap[6], as well as a number of other algorithms from machine learning, are asymptotic results.

[6]The theoretical guarantees for Isomap are a bit more useful as they are able to bound the error in the finite sample case.

Devising strong evaluation techniques looks to be a difficult task, yet is absolutely necessary for progress in this area. Perhaps even more fundamental is determining whether real-world data sets actually exhibit a manifold structure. Some video and image data sets have been found to feature this structure, but much more investigation needs to be done. It may well turn out that most data sets do not contain embedded manifolds, or that the points lie along a manifold only very roughly. If the fundamental assumption of manifold learning turns out to be unrealistic for data mining, then it needs to be determined if these techniques can be adapted so that they are still useful for dimensionality reduction.

The manifold leaning approach to dimensionality reduction shows alot of promise and has had some successes. But, the most fundamental questions are still basically open:

Is the assumption of a manifold structure reasonable?

If it is, then:

How successful are these algorithms at uncovering this structure?

Acknowledgments

Thanks to Sanjoy Dasgupta for valuable suggestions. Thanks also to Henrik Wann Jensen and Virginia de Sa for being on my committee. Figures 1 and 3 were generated using modified versions of code downloaded from the Isomap website (`http://isomap.stanford.edu`) and the Hessian LLE website (`http://basis.stanford.edu/HLLE`).

References

[BdSLT00] Mira Bernstein, Vin de Silva, John Langford, and Joshua Tenenbaum. Graph approximations to geodesics on embedded manifolds. Manuscript, Dec 2000.

[Ber99] Dimitri P. Bertsekas. *Nonlinear Progamming.* Athena Scientific, 2nd edition, 1999.

[BN01] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, 2001.

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.

[Bur05] Christopher J.C. Burges. Geometric methods for feature extraction and dimensional reduction. In L. Rokach and O. Maimon, editors, *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers.* Kluwer Academic Publishers, 2005.

[CC01] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling.* Chapman and Hall/CRC, 2nd edition, 2001.

[CD05] Lawrence Cayton and Sanjoy Dasgupta. Cost functions for euclidean embedding. Unpublished manuscript, 2005.

[DG02] David L. Donoho and Carrie Grimes. When does isomap recover the natural parametrization of families of articulated images? Technical report, Department of Statistics, Stanford University, 2002.

[DG03] David L. Donoho and Carrie Grimes. Hessian eigenmaps: new locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100:5591–5596, 2003.

[dST03] Vin de Silva and Joshua Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, 2003.

[dST04] Vin de Silva and Joshua B. Tenenbaum. Sparse multidimensional scaling using landmark points. Manuscript, June 2004.

[Mil65] John W Milnor. *Topology from the differentiable viewpoint.* The University Press of Virginia, 1965.

[Ple03] Robert Pless. Using isomap to explore video sequences. In *Proc. International Conference on Computer Vision*, 2003.

[SR03] L. Saul and S. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

[SS05] F. Sha and L.K. Saul. Analysis and extension of spectral methods for nonlinear dimensionality reduction. Submitted to ICML, 2005.

[TdL00] J. B. Tenenbaum, V. deSilva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[VB96] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 1996.

[WPS05] K. Weinberger, B. Packer, and L. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.

[WS04] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[WSS04] K.Q. Weinberger, F. Sha, and L.K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the International Conference on Machine Learning*, 2004.

[WZZ04] Jing Wang, Zhenyue Zhang, and Hongyuan Zha. Adaptive manifold learning. In *Advances in Neural Information Processing Systems*, 2004.

[ZZ03] Hongyuan Zha and Zhenyue Zhang. Isometric embedding and continuum isomap. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[ZZ04] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *SIAM Journal of Scientific Computing*, 26(1):313–338, 2004.

A Proof of theorem 1

First, we restate the theorem.

Theorem. *A nonnegative symmetric matrix $D \in \mathbb{R}^{n\times n}$, with zeros on the diagonal, is a Euclidean distance matrix if and only if $B \overset{\text{def}}{=} -\frac{1}{2}HDH$, where $H \overset{\text{def}}{=} I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, is positive semidefinite. Furthermore, this B will be the Gram matrix for a mean-centered configuration with interpoint distances given by D.*

Proof: Suppose that D is a Euclidean distance matrix. Then, there exist $x_1, \ldots, x_n$ such that $\|x_i - x_j\|^2 = D_{ij}$ for all i, j. We assume without loss that these points are mean-centered —*i.e.* $\sum_i x_{ij} = 0$ for all j. We expand the norm:

$$D_{ij} = \|x_i - x_j\|^2 = \langle x_i, x_i\rangle + \langle x_j, x_j\rangle - 2\langle x_i, x_j\rangle, \quad (11)$$

so

$$\langle x_i, x_j\rangle = -\frac{1}{2}(D_{ij} - \langle x_i, x_i\rangle - \langle x_j, x_j\rangle). \quad (12)$$

Let $B_{ij} \overset{\text{def}}{=} \langle x_i, x_j\rangle$. We proceed to rewrite B_{ij} in terms of D. From (11),

$$\begin{aligned}\sum_i D_{ij} &= \left(\sum_i B_{ii} + \sum_i B_{jj} - 2\sum_i B_{ij}\right)\\ &= \sum_i (B_{ii} + B_{jj}).\end{aligned}$$

The second equality follows because the configuration is mean-centered. It follows that

$$\frac{1}{n}\sum_i D_{ij} = \frac{1}{n}\sum_i B_{ii} + B_{jj}. \quad (13)$$

Similarly,

$$\begin{aligned}\frac{1}{n}\sum_j D_{ij} &= \frac{1}{n}\sum_j B_{jj} + B_{ii}, \text{ and} \quad (14)\\ \frac{1}{n^2}\sum_{ij} D_{ij} &= \frac{2}{n}\sum_i B_{ii}. \quad (15)\end{aligned}$$

Subtracting (15) from the sum of (14) and (13) leaves $B_{jj} + B_{ii}$ on the right-hand side, so we may rewrite (12) as

$$\begin{aligned}B_{ij} &= -\frac{1}{2}\left(D_{ij} - \frac{1}{n}\sum_i D_{ij} - \frac{1}{n}\sum_j D_{ij} + \frac{1}{n^2}\sum_{ij} D_{ij}\right)\\ &= -\frac{1}{2}[HDH]_{ij}.\end{aligned}$$

Thus, $-\frac{1}{2}HDH$ is a Gram matrix for $x_1, \ldots, x_n$.

Conversely, suppose that $B \overset{\text{def}}{=} -\frac{1}{2}HDH$ is positive semidefinite. Then, there is a matrix X such that $XX^\top = B$. We show that the points given by the rows of X have square interpoint distances given by D.

First, we expand B using its definition:

$$\begin{aligned} B_{ij} &= -\frac{1}{2}[HDH]_{ij} \\ &= -\frac{1}{2}\left(D_{ij} - \frac{1}{n}\sum_i D_{ij} - \frac{1}{n}\sum_j D_{ij} + \frac{1}{n^2}\sum_{ij} D_{ij}\right) \\ &= -\frac{1}{2}\left(d_{ij}^2 - \overline{d_i^2} - \overline{d_j^2} + \overline{d^2}\right). \end{aligned}$$

Let x_i be the ith row of X. Then,

$$\begin{aligned} \|x_i - x_j\|^2 &= \|x_i\|^2 + \|x_j\|^2 - 2\langle x_i, x_j\rangle \\ &= B_{ii} + B_{jj} - 2B_{ij} \\ &= D_{ij}. \end{aligned}$$

The final equality follows from expanding B_{ij}, B_{ii}, and B_{jj} as written above. Thus, we have shown that the configuration $x_1, \ldots, x_n$ have square interpoint distances given by D, so D is a Euclidean distance matrix. $\square$