Learning an Affine Transformation for Non-linear Dimensionality Reduction

Pooyan Khajehpour Tadavani[1] and Ali Ghodsi[1,2]

[1] David R. Cheriton School of Computer Science
[2] Department of Statistics and Actuarial Science
University of Waterloo
Waterloo, Ontario, Canada

Abstract. The foremost nonlinear dimensionality reduction algorithms provide an embedding only for the given training data, with no straightforward extension for test points. This shortcoming makes them unsuitable for problems such as classification and regression. We propose a novel dimensionality reduction algorithm which learns a parametric mapping between the high-dimensional space and the embedded space. The key observation is that when the dimensionality of the data exceeds its quantity, it is always possible to find a linear transformation that preserves a given subset of distances, while changing the distances of another subset. Our method first maps the points into a high-dimensional feature space, and then explicitly searches for an affine transformation that preserves local distances while pulling non-neighbor points as far apart as possible. This search is formulated as an instance of semi-definite programming, and the resulting transformation can be used to map out-of-sample points into the embedded space.

Keywords: Machine Learning, Dimensionality Reduction, Data Mining

1 Introduction

Manifold discovery is an important form of data analysis in a wide variety of fields, including pattern recognition, data compression, machine learning, and database navigation. In many problems, input data consists of high-dimensional observations, where there is reason to believe that the data lies on or near a low-dimensional manifold. In other words, multiple measurements forming a high-dimensional data vector are typically indirect measurements of a single underlying source. Learning a suitable low-dimensional manifold from high-dimensional data is essentially the task of learning a model to represent the underlying source. This type of dimensionality reduction[3] can also be seen as the process of deriving a set of degrees of freedom which can be used to reproduce most of the variability of the data set.

[3] In this paper the terms ‘manifold learning’ and ‘dimensionality reduction’ are used interchangeably.

Several algorithms for dimensionality reduction have been developed based on eigen-decomposition. Principal components analysis (PCA) [4] is a classical method that provides a sequence of the best linear approximations to the given high-dimensional observations. Another classical method is multidimensional scaling (MDS) [2], which is closely related to PCA. Both of these methods estimate a linear transformation from the training data that projects the high-dimensional data points to a low-dimensional subspace. This transformation can then be used to embed a new test point into the same subspace, and consequently, PCA and MDS can easily handle out-of-sample examples.

The effectiveness of PCA and MDS is limited by the linearity of the subspace they reveal. In order to resolve the problem of dimensionality reduction in nonlinear cases, many nonlinear techniques have been proposed, including kernel PCA (KPCA) [5], locally linear embedding (LLE) [6], Laplacian Eigenmaps [1], and Isomap [12]. It has been shown that all of these algorithms can be formulated as KPCA [3]. The difference lies mainly in the choice of kernel. Common kernels such as RBF and polynomial kernels generally perform poorly in manifold learning, which is perhaps what motivated the development of algorithms such as LLE and Isomap.

The problem of choosing an appropriate kernel remained crucial until more recently, when a number of authors [9–11, 13, 14] have cast the manifold learning problem as an instance of semi-definite programming (SDP). These algorithms usually provide a faithful embedding for a given training data; however, they have no straightforward extension for test points[4]. This shortcoming makes them unsuitable for supervised problems such as classification and regression.

In this paper we propose a novel nonlinear dimensionality reduction algorithm: *Embedding by Affine Transformation (EAT).* The proposed method learns a parametric mapping between the high-dimensional space and the embedding space, which unfolds the manifold of data while preserving its local structure. An intuitive explanation of the method is outlined in Section 2. Section 3 presents the details of the algorithm followed by experimental results in Section 4.

2 The Key Intuition

Kernel PCA first implicitly projects its input data into a high-dimensional feature space, and then performs PCA in that feature space. PCA provides a linear and distance preserving transformation - i.e., all of the pairwise distances between the points in the feature space will be preserved in the embedded space. In this way, KPCA relies on the strength of kernel to unfold a given manifold and reveal its underlying structure in the feature space.

We present a method that, similar to KPCA, maps the input points into a high-dimensional feature space. The similarity ends here, however, as we explicitly search for an affine transformation in the feature space that preserves only the local distances while pulling the non-neighbor points as far apart as possible.

[4] An exception is kernel PCA with closed-form kernels; however, closed-form kernels generally have poor performance even on the training data.

In KPCA, the choice of kernel is crucial, as it is assumed that mapping the data into a high-dimensional feature space can flatten the manifold; if this assumption is not true, its low-dimensional mapping will not be a faithful representation of the manifold. On the contrary, our proposed method does not expect the kernel to reveal the underlying structure of the data. The kernel simply helps us make use of "the blessing of dimensionality". That is, when the dimensionality of the data exceeds its quantity, a linear transformation can span the whole space. This means we can find a linear transformation that preserves the distances between the neighbors and also pulls the non-neighbors apart; thus flattening the manifold. This intuition is depicted in Fig. 1.

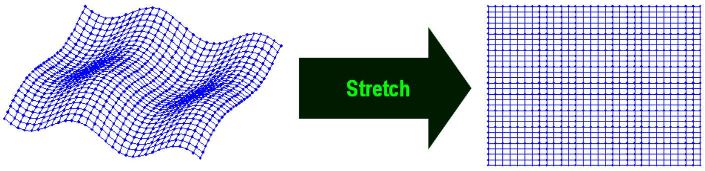


Fig. 1. A simple 2D manifold is represented in a higher-dimensional space. Stretching out the manifold allows it to be correctly embedded in a 2D space.

3 Embedding by Affine Transformation (EAT)

We would like to learn an Affine transformation that preserves the distances between neighboring points, while pulling the non-neighbor points as far apart as possible. In a high dimensional space, this transformation looks locally like a rotation plus a translation which leads to a local isometry; however, for non-neighbor points, it acts as a scaling.

Consider a training data set of n d-dimensional points $\{\mathbf{x}_i\}_{i=1}^n \subset \mathbb{R}^d$. We wish to learn a $d \times d$ transformation matrix $\mathbf{W}$ which will unfold the underlying manifold of the original data points, and embed them into $\{\mathbf{y}_i\}_{i=1}^n$ by:

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} \tag{1}$$

This mapping will not change the dimensionality of the data; $\mathbf{y}$ has the same dimensionality as $\mathbf{x}$. Rather, the goal is to learn $\mathbf{W}$ such that it preserves the local structure but stretches the distances between the non-local pairs. After this projection, the projected data points $\{\mathbf{y}_i\}_{i=1}^n$ will hopefully lie on or close to a linear subspace. Therefore, in order to reduce the dimensionality of $\{\mathbf{y}_i\}_{i=1}^n$, one may simply apply PCA to obtain the ortho-normal axes of the linear subspace. In order to learn $\mathbf{W}$, we must first define two disjoint sets of pairs:

$$\mathcal{S} = \{(i,j)|\ \ \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are neighbors}\}$$
$$\mathcal{O} = \{(i,j)|\ \ \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are non-neighbors}\}$$

The first set consists of pairs of neighbor points in the original space, for which their pairwise distances should be preserved. These pairs can be identified, for example, by computing a neighborhood graph using the K-Nearest Neighbor (KNN) algorithm. The second set is the set of pairs of non-neighbor points, which we would like to pull as far apart as possible. This set can simply include all of the pairs that are not in the first set.

3.1 Preserving Local Distances

Assume for all (i, j) in a given set $\mathcal{S}$, the target distances are known as τ_{ij}. we specify the following cost function, which attempts to preserve the known squared distances:

$$\sum_{(i,j)\in\mathcal{S}} \left(\|\mathbf{y}_i - \mathbf{y}_j\|^2 - \tau_{ij}^2\right)^2 \tag{2}$$

Then we normalize it to obtain[5]:

$$Err = \sum_{(i,j)\in\mathcal{S}} \left(\|\frac{\mathbf{y}_i - \mathbf{y}_j}{\tau_{ij}}\|^2 - 1\right)^2 \tag{3}$$

By substituting (1) into (3), we have:

$$\sum_{(i,j)\in\mathcal{S}} \left(\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{\tau_{ij}}\right)^\mathsf{T} \mathbf{W}\mathbf{W}^\mathsf{T} \left(\frac{\mathbf{x}_i - \mathbf{x}_j}{\tau_{ij}}\right) - 1\right)^2 = \sum_{(i,j)\in\mathcal{S}} \left(\boldsymbol{\delta}_{ij}^\mathsf{T}\mathbf{A}\boldsymbol{\delta}_{ij} - 1\right)^2 \tag{4}$$

where $\boldsymbol{\delta}_{ij} = \frac{\mathbf{x}_i - \mathbf{x}_j}{\tau_{ij}}$ and $\mathbf{A} = \mathbf{W}\mathbf{W}^\mathsf{T}$ is a positive semidefinite (PSD) matrix. It can be verified that:

$$\boldsymbol{\delta}_{ij}^\mathsf{T}\mathbf{A}\boldsymbol{\delta}_{ij} = vec(\mathbf{A})^\mathsf{T} vec(\boldsymbol{\delta}_{ij}\boldsymbol{\delta}_{ij}^\mathsf{T}) = vec(\mathbf{A})^\mathsf{T} vec(\boldsymbol{\Delta}_{ij}) \tag{5}$$

where $vec()$ simply rearranges a matrix into a vector by concatenating its columns, and $\boldsymbol{\Delta}_{ij} = \boldsymbol{\delta}_{ij}\boldsymbol{\delta}_{ij}^\mathsf{T}$. For a symmetric matrix $\mathbf{A}$ we know:

$$vec(\mathbf{A}) = \mathbf{D}_d vech(\mathbf{A}) \tag{6}$$

where $vech(\mathbf{A})$ is the half-vectorization operator, and $\mathbf{D}_d$ is the unique $d^2 \times \frac{d(d+1)}{2}$ duplication matrix. Similar to the $vec()$ operator, the half-vectorization operator rearranges a matrix into a vector by concatenating its columns; however, it stacks

[5] (2) corresponds to the assumption that noise is additive while (3) captures a multiplicative error, i.e. if $\|\mathbf{y}_i - \mathbf{y}_j\|^2 = \tau_{ij}^2 + \varepsilon_{ij}$, where ε_{ij} is additive noise, then clearly $\varepsilon_{ij} = \left(\|\mathbf{y}_i - \mathbf{y}_j\|^2 - \tau_{ij}^2\right)^2$; however, if $\|\mathbf{y}_i - \mathbf{y}_j\|^2 = \tau_{ij}^2 + \varepsilon_{ij} \times \tau_{ij}^2$, then $\varepsilon_{ij} = \left(\|\frac{\mathbf{y}_i - \mathbf{y}_j}{\tau_{ij}}\|^2 - 1\right)^2$. The latter one makes the summation terms comparable.

the columns from the principal diagonal downwards in a column vector. In other words, a symmetric matrix of size d will be rearranged to a column vector of size d^2 by the *vec*() operator, whereas *vech*() will stack it into a column vector of size $\frac{d(d+1)}{2}$. This can significantly reduce the number of unknown variables, especially when d is large. D_d is a unique constant matrix. For example, for a 2×2 symmetric matrix $\mathbf{A}$ we have:

$$\mathit{vec}(\mathbf{A}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathit{vech}(\mathbf{A})$$

Since both $\mathbf{A}$ and $\boldsymbol{\Delta}_{ij}$ are symmetric matrices, we can rewrite (5) using *vech*() and reduce the size of the problem:

$$\boldsymbol{\delta}_{ij}^{\mathsf{T}} \mathbf{A} \boldsymbol{\delta}_{ij} = \mathit{vech}(\mathbf{A})^{\mathsf{T}} \mathbf{D}_d^{\mathsf{T}} \mathbf{D}_d \mathit{vech}(\boldsymbol{\Delta}_{ij}) = \mathit{vech}(\mathbf{A})^{\mathsf{T}} \boldsymbol{\xi}_{ij} \tag{7}$$

where $\boldsymbol{\xi}_{ij} = \mathbf{D}_d^{\mathsf{T}} \mathbf{D}_d \mathit{vech}(\boldsymbol{\Delta}_{ij})$. Using (7), we can reformulate (4) as:

$$\mathit{Err} = \sum_{(i,j)\in\mathcal{S}} (\mathit{vech}(\mathbf{A})^{\mathsf{T}} \boldsymbol{\xi}_{ij} - 1)^2 = \mathit{vech}(\mathbf{A})^{\mathsf{T}} \mathbf{Q} \mathit{vech}(\mathbf{A}) - 2\mathit{vech}(\mathbf{A})^{\mathsf{T}} \mathbf{p} + |\mathcal{S}| \tag{8}$$

where $\mathbf{Q} = \sum_{(i,j)\in\mathcal{S}} \boldsymbol{\xi}_{ij} \boldsymbol{\xi}_{ij}^{\mathsf{T}}$ and $\mathbf{p} = \sum_{(i,j)\in\mathcal{S}} \boldsymbol{\xi}_{ij}$, and $|\mathcal{S}|$ in (8) denotes the number of elements in $\mathcal{S}$, which is constant and can be dropped from the optimization. Now, we can decompose the matrix $\mathbf{Q}$ using the singular value decomposition technique to obtain:

$$\mathbf{Q} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^{\mathsf{T}} \tag{9}$$

If $\mathit{rank}(\mathbf{Q}) = r$, then $\mathbf{U}$ is a $\frac{d(d+1)}{2} \times r$ matrix with r orthonormal basis vectors. We denote the null space of $\mathbf{Q}$ by $\overline{\mathbf{U}}$. Any vector of size $\frac{d(d+1)}{2}$, including vector $\mathit{vech}(\mathbf{A})$, can be represented using the space and the null space of $\mathbf{Q}$:

$$\mathit{vech}(\mathbf{A}) = \mathbf{U}\boldsymbol{\alpha} + \overline{\mathbf{U}}\boldsymbol{\beta} \tag{10}$$

$\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are vectors of size r and $\left(\frac{d(d+1)}{2} - r\right)$ respectively. Since $\mathbf{Q}$ is the summation of $\boldsymbol{\xi}_{ij}\boldsymbol{\xi}_{ij}^{\mathsf{T}}$, and $\mathbf{p}$ is the summation of $\boldsymbol{\xi}_{ij}$, it is easy to verify that $\mathbf{p}$ is in the space of $\mathbf{Q}$ and therefore $\overline{\mathbf{U}}^{\mathsf{T}}\mathbf{p} = 0$. Substituting (9) and (10) in (8), the objective function can be expressed as:

$$\mathit{vech}(\mathbf{A})^{\mathsf{T}} \left(\mathbf{Q}\mathit{vech}(\mathbf{A}) - 2\mathbf{p}\right) = \boldsymbol{\alpha}^{\mathsf{T}} \left(\boldsymbol{\Lambda}\boldsymbol{\alpha} - 2\mathbf{U}^{\mathsf{T}}\mathbf{p}\right) \tag{11}$$

The only unknown variable in this equation is $\boldsymbol{\alpha}$. Hence, (11) can be solved in closed form to obtain:

$$\boldsymbol{\alpha} = \boldsymbol{\Lambda}^{-1}\mathbf{U}^{\mathsf{T}}\mathbf{p} \tag{12}$$

Interestingly, (11) does not depend on $\boldsymbol{\beta}$. This means that the transformation $\mathbf{A}$ which preserves the distances in $\mathcal{S}$ (local distances) is not unique. In fact, there is a family of transformations in the form of (10) that preserve local distances for any value of $\boldsymbol{\beta}$. In this family, we can search for the one that is both positive semi-definite, and increases the distances of the pairs in the set $\mathcal{O}$ as much as possible. The next section shows how the freedom of vector $\boldsymbol{\beta}$ can be exploited to search for a transformation that satisfies these conditions.

3.2 Stretching the Non-Local Distances

We define the following objective function which, when optimized, attempts to maximize the squared distances between the non-neighbor points. That is, it attempts to maximize the squared distances between $\mathbf{x}_i$ and $\mathbf{x}_j$ if $(i,j) \in \mathcal{O}$.

$$Str = \sum_{(i,j)\in\mathcal{O}} \frac{||\mathbf{y}_i - \mathbf{y}_j||^2}{\tau_{ij}^2} \tag{13}$$

Similar to the cost function *Err* in the previous section, we have:

$$\begin{aligned} Str &= \sum_{(i,j)\in\mathcal{O}} \left(\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{\tau_{ij}} \right)^{\mathsf{T}} \mathbf{W}\mathbf{W}^{\mathsf{T}} \left(\frac{\mathbf{x}_i - \mathbf{x}_j}{\tau_{ij}} \right) \right) \\ &= \sum_{(i,j)\in\mathcal{O}} \boldsymbol{\delta}_{ij}^{\mathsf{T}} \mathbf{A} \boldsymbol{\delta}_{ij} = \sum_{(i,j)\in\mathcal{O}} vech(\mathbf{A})^{\mathsf{T}} \boldsymbol{\xi}_{ij} = vech(\mathbf{A})^{\mathsf{T}} \mathbf{s} \end{aligned} \tag{14}$$

where $\mathbf{s} = \sum_{(i,j)\in\mathcal{O}} \boldsymbol{\xi}_{ij}$. Then, the optimization problem is:

$$\max_{\mathbf{A}\succeq 0} \quad vech(\mathbf{A})^{\mathsf{T}} \mathbf{s} \tag{15}$$

Recall that $vech(\mathbf{A}) = \mathbf{U}\boldsymbol{\alpha} + \overline{\mathbf{U}}\boldsymbol{\beta}$, and $\boldsymbol{\alpha}$ is already determined from (12). So the problem can be simplified as:

$$\max_{\mathbf{A}\succeq 0} \quad \boldsymbol{\beta}^{\mathsf{T}} \overline{\mathbf{U}}^{\mathsf{T}} \mathbf{s} \tag{16}$$

Clearly if $\mathbf{Q}$ is full rank, then the matrix $\overline{\mathbf{U}}$ (i.e. the null space of $\mathbf{Q}$) does not exist and therefore, it is not possible to stretch the non-local distances. However, it can be shown that if the dimensionality of the data is more than its quantity, $\mathbf{Q}$ is always rank deficient, and $\overline{\mathbf{U}}$ exists. The rank of $\mathbf{Q}$ is at most $|\mathcal{S}|$, which is due to the fact that $\mathbf{Q}$ is defined in (8) as a summation of $|\mathcal{S}|$ rank-one matrices. Clearly, the maximum of $|\mathcal{S}|$ is the maximum possible number of pairs i.e. $\frac{n\times(n-1)}{2}$; however the size of $\mathbf{Q}$ is $\frac{d\times(d+1)}{2}$.

$\mathbf{Q}$ is rank deficient when $d \geq n$. To make sure that $\mathbf{Q}$ is rank deficient, one can project the points into a high-dimensional space, by some mapping $\phi()$; however, performing the mapping is typically undesirable (e.g. the features may have infinite dimension), so we employ the well-known *kernel trick* [8], using some kernel $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ function that computes the inner products between the feature vectors without explicitly constructing them.

3.3 Kernelizing the Method

In this section, we show how to extend our method to non-linear mappings of data. Conceptually, the points are mapped into a feature space by some non-linear mapping $\phi()$, and then the desired transformation is learned in that space. This can be done implicitly through the use of kernels.

The columns of the linear transformation $\mathbf{W}$ can always be re-expressed as linear combinations of the data points in the feature space, $\mathbf{W} = \mathbf{X}\boldsymbol{\Omega}$. Therefore, we can rewrite the squared distance as:

$$\begin{aligned} |\mathbf{y}_i - \mathbf{y}_j|^2 &= (\mathbf{x}_i - \mathbf{x}_j)^\mathsf{T}\mathbf{W}\mathbf{W}^\mathsf{T}(\mathbf{x}_i - \mathbf{x}_j) \\ &= (\mathbf{x}_i - \mathbf{x}_j)^\mathsf{T}\mathbf{X}\boldsymbol{\Omega}\boldsymbol{\Omega}^\mathsf{T}\mathbf{X}^\mathsf{T}(\mathbf{x}_i - \mathbf{x}_j) \\ &= (\mathbf{x}_i^\mathsf{T}\mathbf{X} - \mathbf{x}_j^\mathsf{T}\mathbf{X})\boldsymbol{\Omega}\boldsymbol{\Omega}^\mathsf{T}(\mathbf{X}^\mathsf{T}\mathbf{x}_i - \mathbf{X}^T\mathbf{x}_j) \\ &= (\mathbf{X}^\mathsf{T}\ \mathbf{x}_i - \mathbf{X}^\mathsf{T}\mathbf{x}_j)^\mathsf{T}\boldsymbol{\mathcal{A}}(\mathbf{X}^\mathsf{T}\mathbf{x}_i - \mathbf{X}^\mathsf{T}\mathbf{x}_j) \end{aligned} \tag{17}$$

where $\boldsymbol{\mathcal{A}} = \boldsymbol{\Omega}\boldsymbol{\Omega}^\mathsf{T}$. We have now expressed the distance in terms of a matrix to be learned, $\boldsymbol{\mathcal{A}}$, and the inner products between the data points which can be computed via the kernel, $\mathbf{K}$.

$$\begin{aligned} |\mathbf{y}_i - \mathbf{y}_j|^2 &= (\mathcal{K}(\mathbf{X}, \mathbf{x}_i) - \mathcal{K}(\mathbf{X}, \mathbf{x}_j))^\mathsf{T}\boldsymbol{\mathcal{A}}(\mathcal{K}(\mathbf{X}, \mathbf{x}_i) - \mathcal{K}(\mathbf{X}, \mathbf{x}_j)) \\ &= (\mathbf{K}_i - \mathbf{K}_j)^\mathsf{T}\boldsymbol{\mathcal{A}}(\mathbf{K}_i - \mathbf{K}_j) \end{aligned} \tag{18}$$

where $\mathbf{K}_i = \mathcal{K}(\mathbf{X}, \mathbf{x}_i)$ is the i_{th} column of the kernel matrix $\mathbf{K}$. The optimization of $\boldsymbol{\mathcal{A}}$ then proceeds just as in the non-kernelized version presented earlier, by substituting $\mathbf{X}$ and $\mathbf{W}$ by $\mathbf{K}$ and $\boldsymbol{\Omega}$ respectively.

3.4 The Algorithm

The training procedure of Embedding by Affine Transformation (EAT) is summarized in Alg.1. Following it, Alg.2 explains how out-of-sample points can be mapped into the embedded space. In these algorithms, we suppose that all training data points are stacked into the columns of a $d \times n$ matrix $\mathbf{X}$. Likewise, all projected data points $\{\mathbf{y}_i\}_{i=1}^n$ are stacked into the columns of a matrix $\mathbf{Y}$ and the $d' \times n$ matrix $\mathbf{Z}$ denotes the low-dimensional representation of the data. In the last line of Alg.1, the columns of $\mathbf{C}$ are the eigenvectors of $\mathbf{Y}\mathbf{Y}^\mathsf{T}$ corresponding to the top d' eigenvalues which are calculated by PCA.

After the training phase of EAT, we have the desired transformation $\mathbf{W}$ for unfolding the latent structure of the data. We also have $\mathbf{C}$ from PCA, which is used to reduce the dimensionality of the unfolded data. As a result, we can embed any new point $\mathbf{x}$ by using the algorithm shown in Alg.2.

4 Experimental Results

In order to evaluate the performance of the proposed method, we have conducted several experiments on synthetic and real data sets.

Alg. 1 EAT - Training

```
Input: $\mathbf{X}$, and $d'$
Output: $\mathbf{Z}$, and linear transformations $\mathbf{W}$ (or $\boldsymbol{\Omega}$) and $\mathbf{C}$

Compute a neighborhood graph and form the sets $\mathcal{S}$ and $\mathcal{O}$
Choose a kernel function and compute the kernel matrix $\mathbf{K}$
Calculate the matrix $\mathbf{Q}$, and the vectors $\mathbf{p}$ and $\mathbf{s}$, based on $\mathbf{K}$, $\mathcal{S}$ and $\mathcal{O}$
Compute $\mathbf{U}$ and $\boldsymbol{\Lambda}$ by performing SVD on $\mathbf{Q}$ such that $\mathbf{Q} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathsf{T}}$
Let $\boldsymbol{\alpha} = \boldsymbol{\Lambda}^{-1}\mathbf{U}^{\mathsf{T}}\mathbf{p}$
Solve the SPD problem $\max_{\mathbf{A}\succeq 0}(\boldsymbol{\beta}^{\mathsf{T}}\overline{\mathbf{U}}^{\mathsf{T}}\mathbf{s})$, where $vech(\mathbf{A}) = \mathbf{U}\boldsymbol{\alpha} + \overline{\mathbf{U}}\boldsymbol{\beta}$
Decompose $\mathbf{A} = \mathbf{W}\mathbf{W}^{\mathsf{T}}$ (or in the kernelized version $\mathcal{A} = \boldsymbol{\Omega}\boldsymbol{\Omega}^{\mathsf{T}}$)
Compute $\mathbf{Y} = \mathbf{W}^{\mathsf{T}}\mathbf{X} = \boldsymbol{\Omega}^{\mathsf{T}}\mathbf{K}$
Apply PCA to $\mathbf{Y}$ and obtain the final embedding $\mathbf{Z} = \mathbf{C}^{\mathsf{T}}\mathbf{Y}$
```

Alg. 2 EAT - Embedding

```
Input: out-of-sample example $\mathbf{x}_{d\times 1}$, and the transformations $\mathbf{W}$ (or $\boldsymbol{\Omega}$) and $\mathbf{C}$
Output: vector $\mathbf{z}_{d'\times 1}$ which is a low-dimensional representation of $\mathbf{x}$

Compute $\mathbf{K}_{\mathbf{x}} = \mathcal{K}(.,\mathbf{x})$
Let $\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} = \boldsymbol{\Omega}^{\mathsf{T}}\mathbf{K}_{\mathbf{x}}$
Compute $z = \mathbf{C}^{\mathsf{T}}\mathbf{y}$
```

To emphasize the difference between the transformation computed by EAT and the one that PCA provides, we designed a simple experiment on a synthetic data set. In this experiment we consider a three-dimensional V-shape manifold illustrated in the top-left panel of Fig. 2. We represent this manifold by 1000 uniformly distributed sample points, and divide it into two subsets: a training set of 28 well-sampled points, and a test set of 972 points. EAT is applied to the training set, and then the learned transformation is used to project the test set. The result is depicted in the top-right panel of Fig. 2. This image illustrates $\mathbf{Y} = \mathbf{W}^{\mathsf{T}}\mathbf{X}$, which is the result of EAT in 3D before applying PCA. It shows that the third dimension carries no information, and the unfolding happens before PCA is applied to reduce the dimensionality to 2D.

The bottom-left and bottom-right panels of Fig. 2 show the results of PCA and KPCA, when applied to the whole data set. PCA computes a global distance preserving transformation, and captures the directions of maximum variation in the data. Clearly, in this example, the direction with the maximum variation is not the one that unfolds the V-shape. This is the key difference between the functionality of PCA and EAT. Kernel PCA does not provide a satisfactory embedding either. Fig. 2 shows the result that is generated by an RBF kernel; we experimented KPCA with a variety of popular kernels, but none were able to reveal a faithful embedding of the V-shape.

Unlike kernel PCA, EAT does not expect the kernel to reveal the underlying structure of data. When the dimensionality of data is higher than its quantity, a linear transformation can span the whole space. This means we can always find $\mathbf{W}$ to flatten the manifold.

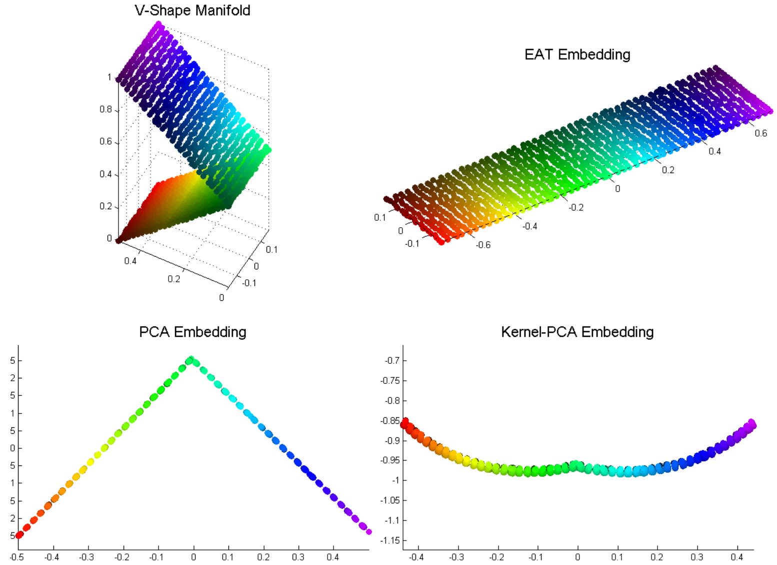


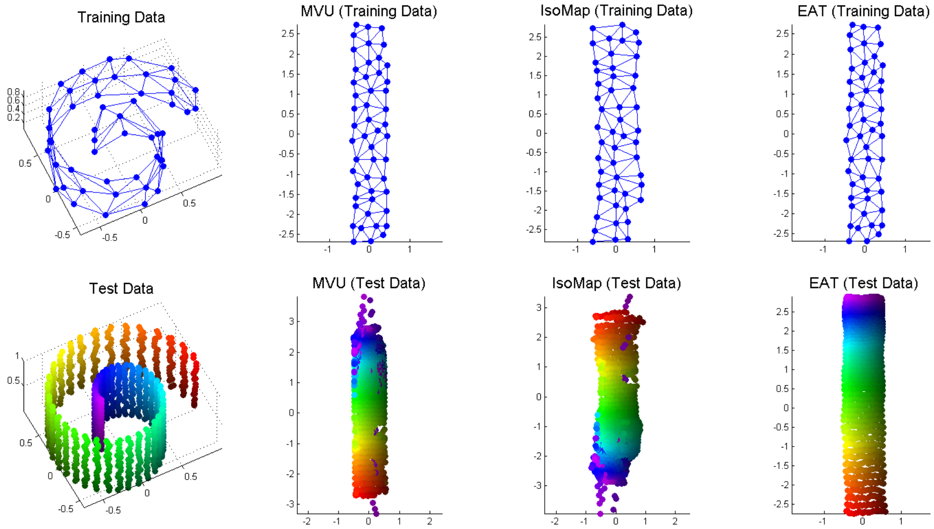
Fig. 2. A V-shape manifold, and the results of EAT, PCA and kernel PCA.

When the original dimensionality of data is high ($d > n$, e.g. for images), EAT does not need a kernel in principal; however, using a linear kernel reduces the computational complexity of the method [6]. In all of the following experiments, we use a linear kernel when the original dimensionality of data is high (e.g. for images), and RBF in all other cases. In general EAT is not that sensitive to the type of kernel. We will discuss the effect of kernel type and its parameter(s) later in this section.

The next experiment is on a Swiss roll manifold, depicted in the bottom-left panel of Fig. 3. Although Swiss roll is a three-dimensional data set, it tends to be one of the most challenging data sets due to its complex global structure. We sample 50 points for our training set, and 950 points as an out-of-sample test set. The results of Maximum Variance Unfolding (MVU), Isomap, and EAT [7] are presented in the first row of Fig. 3. The second row shows the projection of the out-of-sample points into a two-dimensional embedded space. EAT computes a transformation that maps the new data points into the low-dimensional space. MVU and Isomap, however, do not provide any direct way to handle out-of-sample examples. A common approach to resolve this problem is to learn a non-parametric model between the low and high dimensional spaces.

[6] In the kernelized version, $\mathbf{W}$ is $n \times n$ but in the original version it is $d \times d$. Thus, Computing $\mathbf{W}$ in the kernelized form is less complex when $d > n$.

[7] In general, Kernel PCA fails to unfold the Swiss roll data set. LLE generally produces a good embedding, but not on small data sets (e.g. the training set in this experiment). For this reason we do not demonstrate their results.


Training Data

MVU (Training Data)

IsoMap (Training Data)

EAT (Training Data)

Test Data

MVU (Test Data)

IsoMap (Test Data)

EAT (Test Data)

Fig. 3. A Swiss roll manifold, and the results of different dimensionality reduction methods: MVU, Isomap, and EAT. The top row demonstrates the results on the training set, and the bottom row shows the results of the out-of-sample test set.

In this approach, a high-dimensional test data point $\mathbf{x}$ is mapped to the low dimensional space in three steps: (i) the k nearest neighbors of $\mathbf{x}$ among the training inputs (in the original space) are identified; (ii) the linear weights that best reconstruct $\mathbf{x}$ from its neighbors, subject to a sum-to-one constraint, are computed; (iii) the low-dimensional representation of $\mathbf{x}$ is computed as the weighted sum (with weights computed in the previous step) of the embedded points corresponding to those k neighbors of $\mathbf{x}$ in the original space. In all of the examples in this paper, the out-of-sample embedding is conducted using this non-parametric model except for EAT, PCA, and Kernel PCA which provide parametric models. It is clear that the out-of-sample estimates of MVU and Isomap are not faithful to the Swiss roll shape, especially along its border.

Now we illustrate the performance of the proposed method on some real data sets. Fig. 4 shows the result of EAT when applied to a data set of face images. This data set consists of 698 images, from which we randomly selected 35 as the training set and the rest are used as the test data. Training points are indicated with a solid blue border.

The images in this experiment have three degrees of freedom: pan, tilt, and brightness. In Fig. 4, the horizontal and vertical axes appear to represent the pan and tilt, respectively. Interestingly, while there are no low-intensity images among the training samples, darker out-of-sample points appear to have been organized together in the embedding. These darker images still maintain the correct trends in the variation of pan and tilt across the embedding. In this example, EAT was used with a linear kernel.

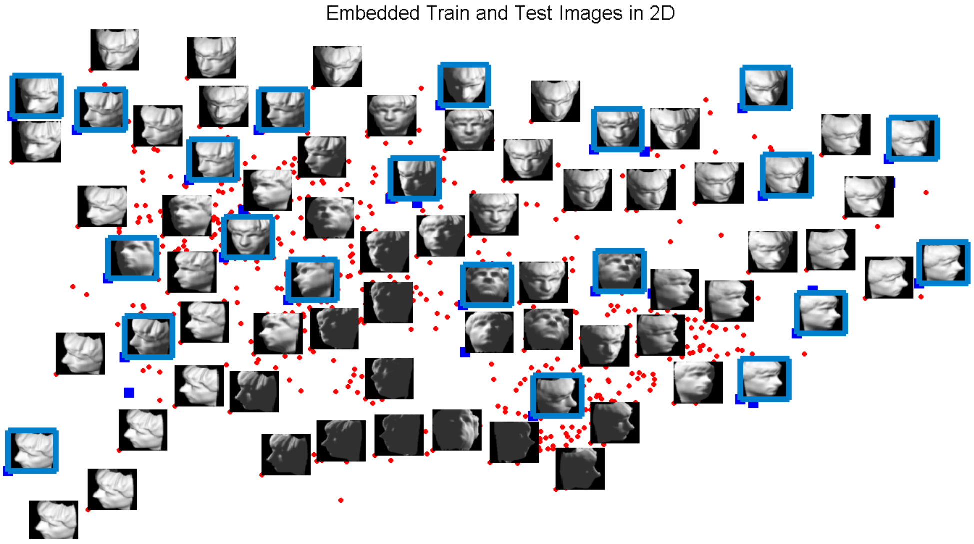


Fig. 4. The result of manifold learning with EAT (using a linear kernel) on a data set of face images.

In another experiment , we conducted an experiment on a subset of the Olivetti image data set [7]. Face images of three different persons are used as the training set, and images of a fourth person are used as the out-of-sample test examples. The results of MVU, LLE, Isomap, PCA, KPCA, and EAT are illustrated in Fig. 5. Different persons in the training data are indicated by red squares, green triangles and purple diamonds. PCA and Kernel PCA do not provide interpretable results even for the training set. The other methods, however, separate the different people along different chains. Each chain shows a smooth change between the side view and the frontal view of an individual.

The key difference between the algorithms is the way they embed the images of the new person (represented by blue circles). MVU, LLE, Isomap, PCA, and Kernel PCA all superimpose these images onto the images of the most similar individual in the training set, and by this, they clearly lose a part of information. This is due to the fact, that they learn a non-parametric model for embedding the out-of-samples. EAT, however, embeds the images of the new person as a separate cluster (chain), and maintains a smooth gradient between the frontal and side views.

Finally, we attempt to unfold a globe map (top-left of Fig. 6) into a faithful 2D representation. Since a complete globe is a closed surface and thus cannot be unfolded, our experiment is on a half-globe. A regular mesh is drawn over the half-globe (top-right of Fig. 6), and 181 samples are taken for the training set. EAT is used to unfold the sampled mesh and find its transformation (bottom-right of Fig. 6).

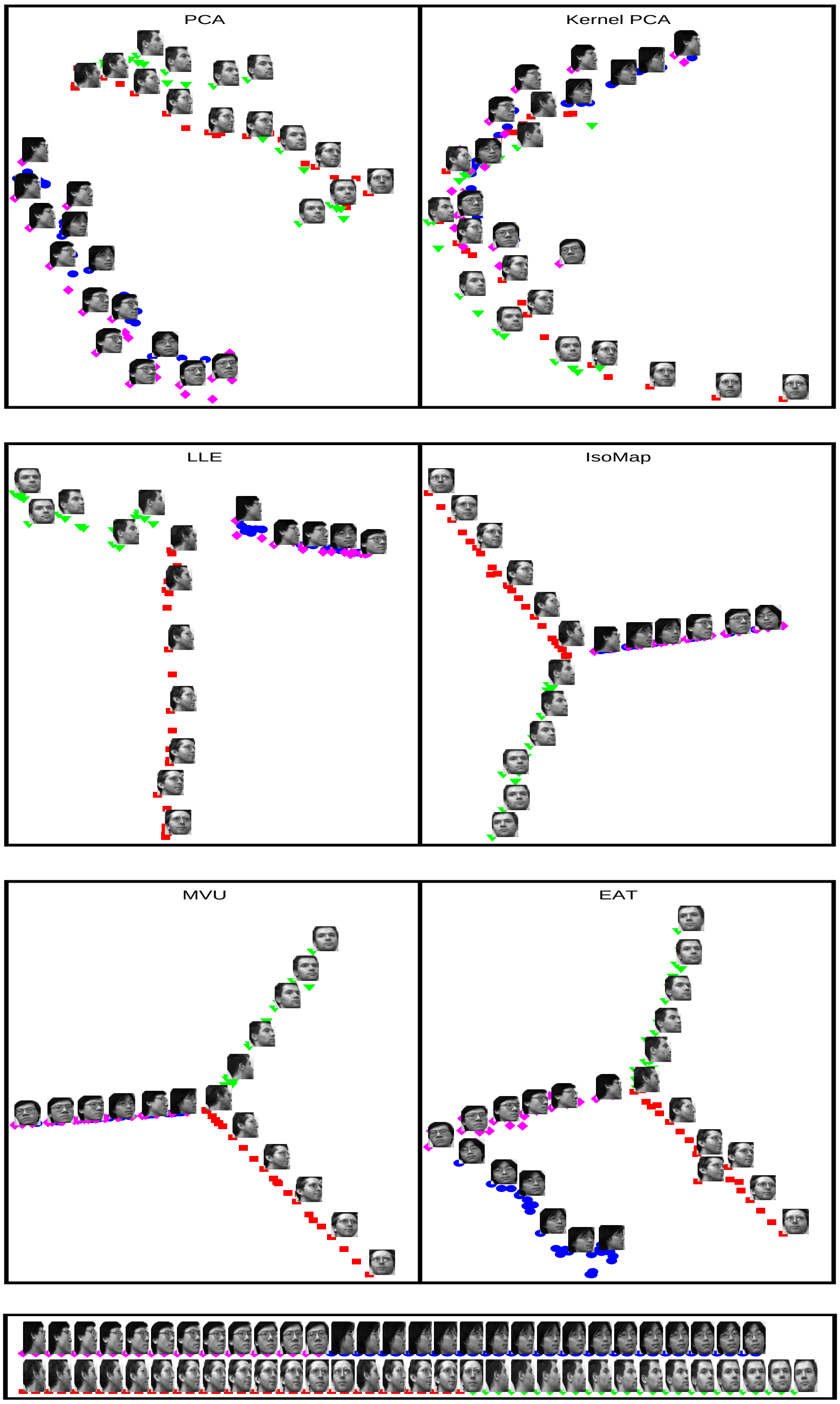


Fig. 5. The results of different dimensionality reduction techniques on a data set of face photos. Each color represents the pictures of one of four individuals. The blue circles show the test data (pictures of the fourth individual).

Note that it is not possible to unfold a globe into a 2D space while preserving the original local distances; in fact, the transformation with the minimum preservation error is the identity function. So rather than preserving the local distances, we define Euclidean distances based on the latitude and longitude of the training points along the surface of the globe; then the 2D embedding becomes feasible. This is an interesting aspect of EAT: it does not need to operate on the original distances of the data, but can instead be supplied with arbitrary distance values (as long as they are compliant with the desired dimensionality reduction of the data).

Our out-of-sample test set consists of 30,000 points as specified by their 3D position with respect to the center of the globe. For this experiment we used an RBF kernel with $\sigma = 0.3$. Applying the output transformation of EAT results in the 2D embedding shown in the bottom-left of Fig. 6; color is used to denote elevation in these images. Note that the pattern of the globe does not change during the embedding process, which demonstrates that the representation of EAT is faithful. However, the 2D embedding of the test points is distorted at the sides, which is due to the lack of information from the training samples in these areas.

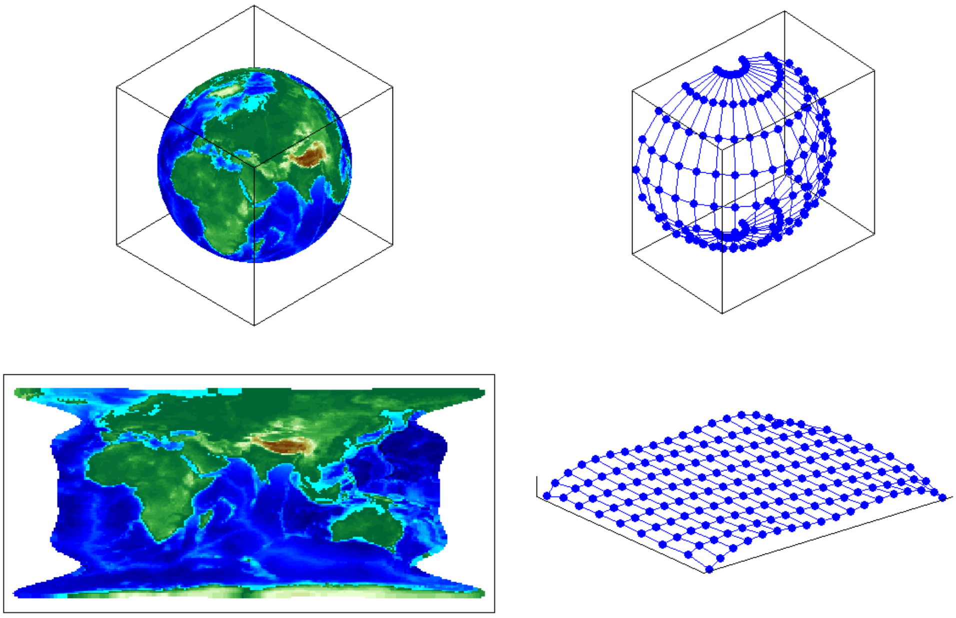


Fig. 6. Unfolding a half-globe into a 2D map by EAT. A half-sphere mesh is used for training. Color is used to denote elevation. The out-of-sample test set comprises 30,000 points from the surface of Earth.

4.1 The Effect of Type and Parameters of the Kernels

The number of bases corresponding to a particular kernel matrix is equal to the rank of that matrix. If we use a full rank kernel matrix (i.e. $rank(\mathbf{K}) = n$), then the number of bases is equal to the number of data points and a linear transformation can span the whole space. That is, it is always possible to find a transformation $\mathbf{W}$ that perfectly unfolds the data as far as the training data points are concerned. For example, an identity kernel matrix can perfectly unfold any training data set; but it will fail to map out-of-sample points correctly, because it cannot measure the similarity between the out-of-sample points and the training examples. In other words, using a full rank kernel is a sufficient condition in order to faithfully embed the training points. But if the correlation between the kernel and the data is weak (an extreme case is using the identity matrix as a kernel), EAT will not perform well for the out-of-sample points.

We define $r = \frac{rank(\mathbf{K})}{n}$. Clearly $r = 1$ indicates a full rank matrix and $r < 1$ shows a rank deficient kernel matrix $\mathbf{K}$. The effect of using different kernels on the Swiss roll manifold (bottom-left of Fig. 3) are illustrated in Fig. 7.

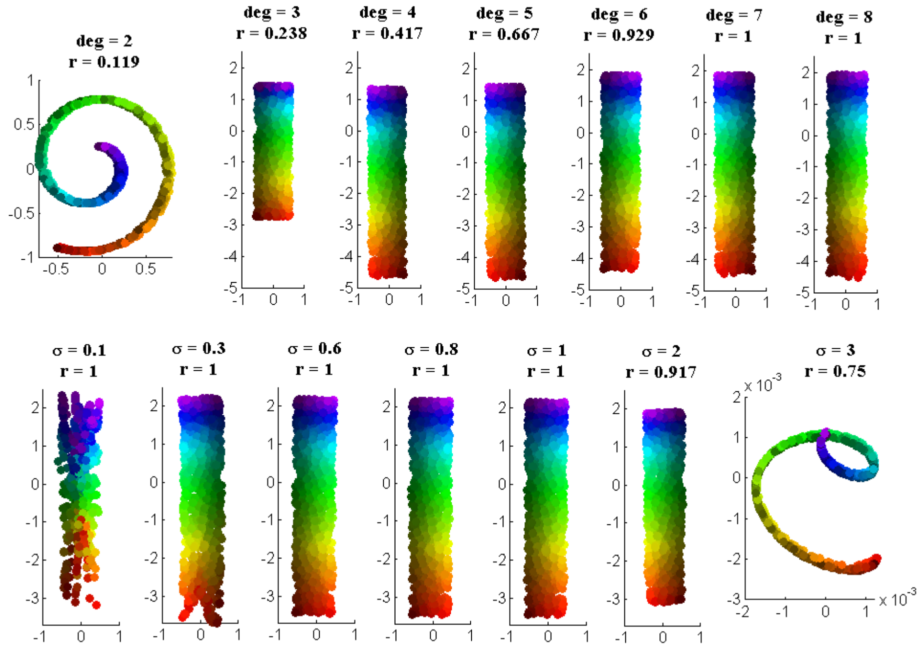


Fig. 7. The effect of using different kernels for embedding a Swiss-roll manifold. Polynomials of different degrees are used in the first row, and in the second row RBF kernels with different σ values map the original data to the feature space.

Two different kernels are demonstrated. In the first row polynomial kernels of different degrees are used, and the second row shows the result of RBF kernels which have different values for their variance parameter σ. The dimensionality of the feature spaces of the low degree polynomial kernels ($deg = 2, 3$) is not high enough; thus they do not produce satisfactory results. Similarly, in the experiment with RBF kernels, when σ is high, EAT is not able to find the desired affine transformation in the feature space to unfold the data (e.g. the rightmost-bottom result).

The leftmost-bottom result is generated by an RBF kernel with a very small value assigned to σ. In this case, the kernel is full rank and consequently $r = 1$. The training data points are mapped perfectly as expected but EAT fails to embed the out-of-sample points correctly. Note that with such a small σ the resulted RBF kernel matrix is very close to the identity matrix, so over-fitting will happen in this case. Experiments with a wide variety of other kernels on different data sets show similar results. Based on these experiments, we suggest that an RBF kernel can be used for any data set. The parameter σ should be selected such that (i) the kernel matrix is full rank or close to full rank ($r \approx 1$), and (ii) the resulting kernel is able to measure the similarity between non-identical data points (σ is not too small). This method is not sensitive to type of kernel. For an RBF kernel a wild range of values for σ can be safely used, as long as the conditions (i) and (ii) are satisfied. When the dimensionality of the original data is more than or equal to the number of the data points, there is no need for a kernel, but one may use a simple linear kernel to reduce the computational complexity[8].

5 Conclusion

We presented a novel dimensionality reduction method which, unlike other prominent methods, can easily embed out-of-sample examples. Our method learns a parametric mapping between the high and low dimensional spaces, and is performed in two steps. First, the input data is projected into a high-dimensional feature space, and then an affine transformation is learned that maps the data points from the feature space into the low dimensional embedding space. The search for this transformation is cast as an instance of semi-definite programming (SDP), which is convex and always converges to a global optimum. However, SDP is computationally intensive, which can make it inefficient to train EAT on large data sets.

Our experimental results on real and synthetic data sets demonstrate that EAT produces a robust and faithful embedding even for very small data sets. It also shows that it is successful at projecting out-of-sample examples. Thus, one approach for handling large data sets with EAT would be to downsample the data by selecting a small subset as the training input and embedding the rest of the data as test examples.

[8] An RBF kernel can be used for this case as well.

Another feature of EAT is that it treats the distances between the data points in three different ways. One can preserve a subset of the distances (set $\mathcal{S}$), stretch another subset (set $\mathcal{O}$) and leave the third set (pairs that are not in $\mathcal{S}$ and $\mathcal{O}$) unspecified. This is in contrast with methods like MVU that preserve local distances but stretch any non-local pairs. This property means that EAT could be useful for semi-supervised tasks where only partial information about similarity and dissimilarity of points is known.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Proceedings NIPS (2001)
2. Cox, T., Cox, M.: Multidimensional Scaling. Chapman Hall, Boca Raton, 2nd edn. (2001)
3. Ham, J., Lee, D., Mika, S., B., S.: A kernel view of the dimensionality reduction of manifolds. In: International Conference on Machine Learning (2004)
4. Jolliffe, I.: Principal Component Analysis. Springer-Verlag, New York (1986)
5. Mika, S., Schölkopf, B., Smola, A., Müller, K.R., Scholz, M., Rätsch, G.: Kernel PCA and de-noising in feature spaces. In: Kearns, M.S., Solla, S.A., Cohn, D.A. (eds.) Proceedings NIPS 11. MIT Press (1999)
6. Roweis, S., Saul, L.: Nonlinear dimensionality reduction by locally linear embedding. Science 290, 2323–2326 (2000)
7. Samaria, F., Harter, A.: Parameterisation of a Stochastic Model for Human Face Identification. In: Proceedings of 2nd IEEE Workshop on Applications of Computer Vision (1994)
8. Schölkopf, B., Smola, A.: Learning with Kernels. MIT Press, Cambridge, Massachusetts (2002)
9. Shaw, B., Jebara, T.: Minimum volume embedding. In: Meila, M., Shen, X. (eds.) Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics March 21-24, 2007, San Juan, Puerto Rico. vol. 2 of JMLR: W&CP, pp. 460–467 (March 2007)
10. Shaw, B., Jebara, T.: Structure preserving embedding. In: Bottou, L., Littman, M. (eds.) Proceedings of the 26th International Conference on Machine Learning. pp. 937–944. Omnipress, Montreal (June 2009)
11. Song, L., Smola, A.J., Borgwardt, K.M., Gretton, A.: Colored maximum variance unfolding. In: NIPS (2007)
12. Tenenbaum, J.: Mapping a manifold of perceptual observations. In: Advances in Neural Information Processing Systems 10. pp. 682–687 (1998)
13. Weinberger, K.Q., Saul, L.K.: Unsupervised learning of image manifolds by semidefinite programming. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR-04). vol. II, pp. 988–995 (2004)
14. Weinberger, K., Sha, F., Zhu, Q., Saul, L.: Graph Laplacian regularization for large-scale semidefinite programming. Advances in neural information processing systems 19, 1489 (2007)