Where else can we use kernels?

Lots of things!

Recall PCA, where our goal is to find an approximation

$$\mathbf{x}_i \approx \boldsymbol{\mu} + \mathbf{A}\mathbf{z}_i$$

where

- $\boldsymbol{\mu} \in \mathbb{R}^d$
- $\mathbf{A} \in \mathbb{R}^{d \times k}$ with orthonormal columns
- $\mathbf{z}_i \in \mathbb{R}^k$

Connection to SVD

Recall the singular value decomposition (SVD)

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

If $\mathbf{X}$ is a real $d \times n$ matrix

- $\mathbf{U}$ is a $d \times d$ orthonormal matrix
- $\mathbf{V}$ is an $n \times n$ orthonormal matrix
- $\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_r)$ is a $d \times n$ diagonal matrix where $r \leq \min(d, n)$ and

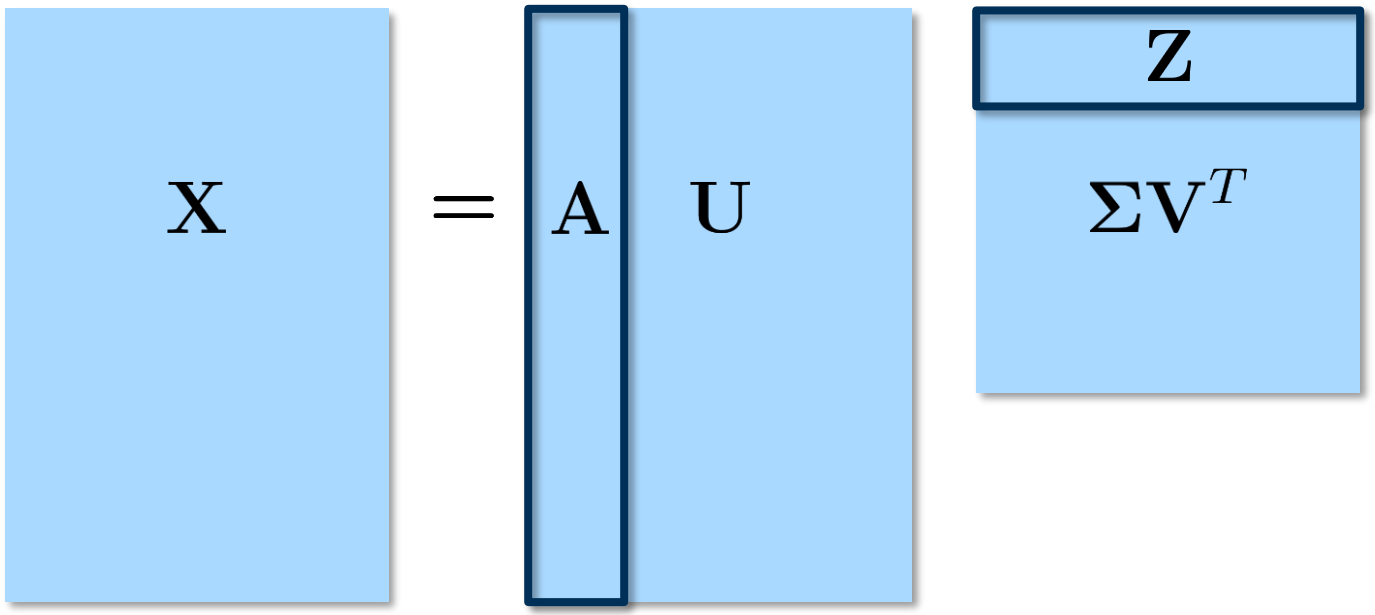
$$\sigma_i = i^{\text{th}} \text{ singular value}$$

$$= \text{square root of } i^{\text{th}} \text{ eigenvalue of } \mathbf{X}\mathbf{X}^T$$

The principal eigenvectors are the first k columns of $\mathbf{U}$ when the columns of $\mathbf{X}$ are filled with $\widetilde{\mathbf{x}}_i = \mathbf{x}_i - \boldsymbol{\mu}$

Visual interpretation

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$



PCA as an embedding

If we don't care about $\mathbf{A}$ and instead only want the $\mathbf{z}_i$'s, we can get this by instead taking the eigendecomposition of

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\ &= \mathbf{V}^T\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\ &= \mathbf{V}^T\mathbf{\Sigma}^2\mathbf{V}^T\end{aligned}$$

Note that $\mathbf{X}^T\mathbf{X}$ depends only on inner products between vectors in the dataset

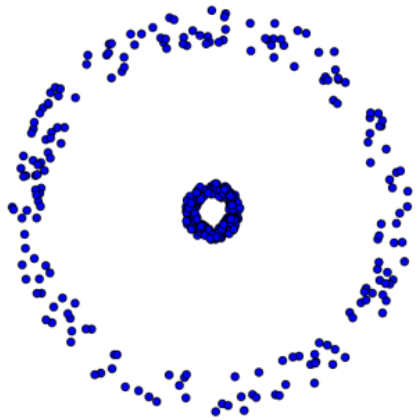
Summary of kernel PCA

Input: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, kernel k, desired dimension r

1. Form $\widetilde{\mathbf{K}} = \mathbf{HKH}$, where $\mathbf{K}$ is our kernel matrix and $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ is the "centering matrix" (Optional)

2. Compute eigendecomposition $\widetilde{\mathbf{K}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$

3. Set $\mathbf{Z}$ to be the first r rows of $(\mathbf{V}\mathbf{\Lambda}^{1/2})^T$



Useful in settings where the data would live in a linear subspace if only we had used the right features

We will return to this later…

Learning from pairwise distances

In PCA, our goal is to take a high-dimensional dataset and generate a ***low-dimensional embedding*** of the data that preserves the (Euclidean) structure of the data

It is not uncommon to encounter situations where the Euclidean distance is not appropriate, or where we do not even begin with direct access to the data

Instead, suppose we are given an $n \times n$ ***dissimilarity matrix*** $\mathbf{D}$ and wish to find a dimension k and points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ such that $\rho(\mathbf{x}_i, \mathbf{x}_j) = d_{ij}$ for some distance function ρ

Applications

- visualization/dimensionality reduction
- extend algorithms to non-Euclidean (or non-metric) data

Dissimilarity matrix

A dissimilarity matrix should satisfy

- $d_{ij} \geq 0$
- $d_{ij} = d_{ji}$
- $d_{ii} = 0$

Note that we do ***not*** require the triangle inequality, since in practice many measures of “dissimilarity” will not have this property

In general, a perfect embedding into the desired dimension will not exist

We will be interested mostly in ***approximate*** embeddings

Multidimensional scaling (MDS)

The problem of finding $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ such that $\rho(\mathbf{x}_i, \mathbf{x}_j)$ approximately agrees with d_{ij} is known as ***multidimensional scaling (MDS)***

There are a number of variants of MDS based on

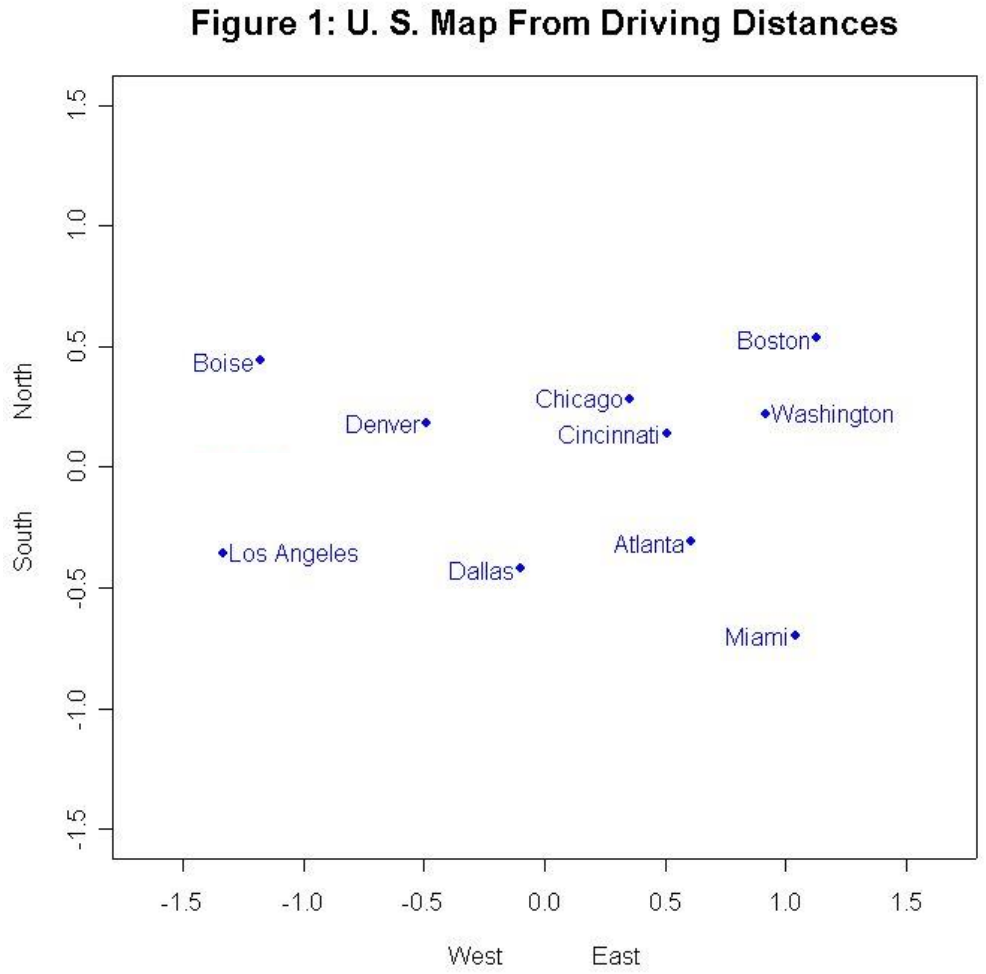
- our choice of distance function ρ
- how we quantify "approximately agrees with"
- whether we have access to all entries of $\mathbf{D}$

Main distinction

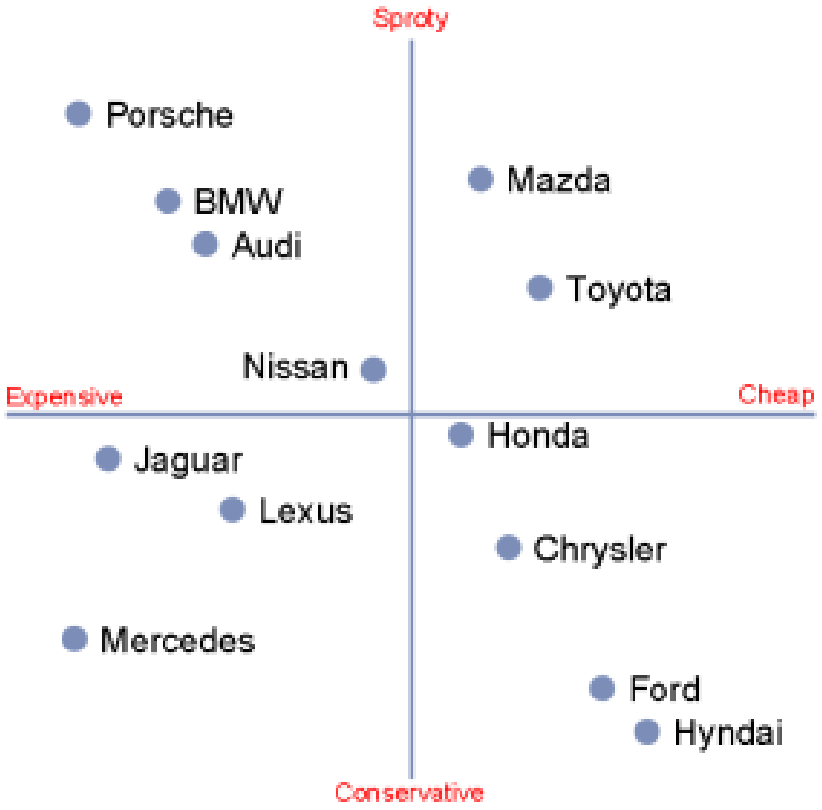
- ***metric*** methods attempt to ensure that $\rho(\mathbf{x}_i, \mathbf{x}_j) \approx d_{ij}$
- ***nonmetric*** methods only attempt to preserve rank ordering, i.e., if $d_{ij} \leq d_{\ell m}$ then nonmetric methods seek an embedding that satisfies

$$\rho(\mathbf{x}_i, \mathbf{x}_j) \leq \rho(\mathbf{x}_\ell, \mathbf{x}_m)$$

Example: Creating a map



Example: Marketing



Example: Whisky

Euclidean embeddings

Today we will focus primarily on the metric case where we observe all of $\mathbf{D}$ and choose $\rho(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$

To see how we might find an embedding, first consider the reverse process…

Given an (exact) embedding $\mathbf{X}$, where the $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ form the columns of $\mathbf{X}$, how can we compute $\mathbf{D}$?

Consider $\mathbf{D}^2$, i.e., the matrix with entries

$$d_{ij}^2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \|\mathbf{x}_i\|_2^2 + \|\mathbf{x}_j\|_2^2 - 2\mathbf{x}_i^T\mathbf{x}_j$$

$$\Rightarrow \mathbf{D}^2 = \mathbf{b}\mathbf{1}^T + \mathbf{1}\mathbf{b}^T - 2\mathbf{X}^T\mathbf{X}$$

where $\mathbf{b} = \left[\|\mathbf{x}_1\|_2^2, \ldots, \|\mathbf{x}_n\|_2^2\right]^T$ and $\mathbf{1} = [1, \ldots, 1]^T$

Finding the embedding

Thus, we know that

$$\mathbf{X}^T\mathbf{X} = \frac{1}{2}(\mathbf{b}\mathbf{1}^T + \mathbf{1}\mathbf{b}^T - \mathbf{D}^2)$$

We are given $\mathbf{D}^2$, but $\mathbf{b}$ is actually part of what we are trying to estimate, right?

Consider the ***"centering matrix"*** $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$

Observe that $\widetilde{\mathbf{X}} = \mathbf{X}\mathbf{H}$ is simply our data set $\mathbf{X}$ with the mean subtracted off

If all we know are distances between pairs of points, we have lost all information about any rigid transformation (e.g., a translation) of our data

Finding the embedding

We are free to enforce that our embedding is centered around the origin, in which case we are interested in

$$\begin{aligned}\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}} &= \mathbf{H}^T\mathbf{X}^T\mathbf{X}\mathbf{H} \\ &= \frac{1}{2}(\mathbf{H}\mathbf{b}\mathbf{1}^T\mathbf{H} + \mathbf{H}\mathbf{1}\mathbf{b}^T\mathbf{H} - \mathbf{H}\mathbf{D}^2\mathbf{H})\end{aligned}$$

Note that $\mathbf{1}^T\mathbf{H} = \mathbf{H}\mathbf{1} = 0$

$$\Rightarrow \quad \widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H}$$

We can compute $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H} = \widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}$, from which we can then find $\widetilde{\mathbf{X}}$ by computing an eigendecomposition

Classical MDS

Even if a dissimilarity matrix $\mathbf{D}$ cannot be perfectly embedded into k dimensions, this suggests an approximate algorithm

1. Form $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H}$
2. Compute the eigendecomposition $\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$
3. Return $\mathbf{X} = (\mathbf{V}_k\mathbf{\Lambda}^{1/2})^T$, i.e., the matrix whose rows are given by $\sqrt{\lambda_i}\mathbf{v}_i^T$ for $i = 1, \ldots, k$

It can be shown that classical MDS finds the embedding that minimizes either $\|\mathbf{X}^T\mathbf{X} - \mathbf{B}\|$ or $\|\mathbf{X}^T\mathbf{X} - \mathbf{B}\|_F$
(Eckart-Young Theorem)

Equivalence between PCA and MDS

Suppose we have $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and set $\mathbf{D}$ to be such that

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

The result of classical MDS applied to $\mathbf{D}$ is the same (up to a rigid transformation) as applying PCA to $\mathbf{X}$

PCA computes an eigendecomposition of $\mathbf{S} = \mathbf{X}\mathbf{X}^T$, or equivalently, the SVD of $\mathbf{X}$ to yield the factorization $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$

MDS computes an eigendecomposition of

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{\Sigma}\mathbf{\Sigma}\mathbf{V}^T\end{aligned}$$

Subtle difference between PCA and MDS

The two approaches give the same embedding $\mathbf{Z}$

PCA also comes with $\mathbf{A}$ and $\boldsymbol{\mu}$

- lets us compute $\mathbf{x}_i \approx \boldsymbol{\mu} + \mathbf{A}\mathbf{z}_i$
- more importantly, lets us compute $\mathbf{z}_i = \mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})$

The latter is ***critical*** in a real-world application if we want to use PCA/MDS as a technique for feature extraction

Is there anything we can do to recover $\mathbf{A}$ and $\boldsymbol{\mu}$ in the MDS setting?

Almost… we ***cannot*** recover $\mathbf{A}$ and $\boldsymbol{\mu}$ without $\mathbf{X}$…

But we ***can*** compute the mapping

$$\mathbf{A}^T(\mathbf{x} - \boldsymbol{\mu})$$

Computing the MDS mapping

Given the SVD of $\mathbf{XH} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$, $\mathbf{A}$ is given by the first k columns of $\mathbf{U}$

Rearranging, we have that $\mathbf{U} = \mathbf{XHV}\boldsymbol{\Sigma}^{-1}$

MDS provides us with $\mathbf{Z}$: the first k rows of $\boldsymbol{\Sigma}\mathbf{V}^T$

This also gives us $\mathbf{Z}^\dagger$: the first k columns of $\mathbf{V}\boldsymbol{\Sigma}^{-1}$

$$\Rightarrow \mathbf{A} = \mathbf{XHZ}^\dagger$$

Remember that we obtained $\mathbf{V}$ via the eigendecomposition

$$\mathbf{B} = -\tfrac{1}{2}\mathbf{HD}^2\mathbf{H} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$$

Neat fact: Since $\mathbf{H1} = \mathbf{0}$, $\mathbf{1}$ is an eigenvector of $\mathbf{B}$ (with eigenvalue $\lambda = 0$), and thus $\mathbf{HZ}^\dagger = \mathbf{Z}^\dagger$

Is this enough?

Putting this all together, we can write

$$\mathbf{A}^T(\mathbf{x} - \boldsymbol{\mu}) = (\mathbf{Z}^\dagger)^T\mathbf{X}^T(\mathbf{x} - \boldsymbol{\mu})$$

OK… but this still looks like we need to have $\mathbf{X}$, right?

In MDS, we only get to observe $\mathbf{D}^2$ and wish to embed $\mathbf{x}$ based on the observations

$$\mathbf{d}_\mathbf{x} = \begin{bmatrix} \|\mathbf{x} - \mathbf{x}_1\|_2^2 \\ \vdots \\ \|\mathbf{x} - \mathbf{x}_n\|_2^2 \end{bmatrix}$$

Using what we actually have…

We are given $\mathbf{d_x}$ and $\mathbf{D}^2$

Let $\mathbf{d_\mu} = \frac{1}{n}\mathbf{D}^2\mathbf{1}$ denote the column mean of $\mathbf{D}^2$

Consider the vector $\mathbf{d_\mu} - \mathbf{d_x}$:

$$
\begin{aligned}
[\mathbf{d_\mu} - \mathbf{d_x}](i) &= \tfrac{1}{n}\textstyle\sum_{j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 - \|\mathbf{x} - \mathbf{x}_i\|_2^2 \\
&= \tfrac{1}{n}\textstyle\sum_{j=1}^{n} \left(\|\mathbf{x}_i\|_2^2 - 2\mathbf{x}_i^T\mathbf{x}_j + \|\mathbf{x}_j\|_2^2\right) \\
&\qquad - \left(\|\mathbf{x}\|_2^2 - 2\mathbf{x}_i^T\mathbf{x} + \|\mathbf{x}_i\|_2^2\right) \\
&= 2\mathbf{x}_i^T(\mathbf{x} - \boldsymbol{\mu}) - \|\mathbf{x}\|_2^2 + \tfrac{1}{n}\textstyle\sum_{j=1}^{n} \|\mathbf{x}_j\|_2^2 \\
&= 2\mathbf{x}_i^T(\mathbf{x} - \boldsymbol{\mu}) + c
\end{aligned}
$$

Out-of-sample extension for MDS

From this, we have that

$$\tfrac{1}{2}(\mathbf{d}_\mu - \mathbf{d}_\mathbf{x}) = \mathbf{X}^T(\mathbf{x} - \boldsymbol{\mu}) + \tfrac{c}{2}\mathbf{1}$$

Combining this with what we had before, we can write

$$\begin{aligned}\mathbf{A}^T(\mathbf{x} - \boldsymbol{\mu}) &= (\mathbf{Z}^\dagger)^T\mathbf{X}^T(\mathbf{x} - \boldsymbol{\mu}) \\ &= (\mathbf{Z}^\dagger)^T\left(\tfrac{1}{2}(\mathbf{d}_\mu - \mathbf{d}_\mathbf{x})\right) - \tfrac{c}{2}(\mathbf{Z}^\dagger)^T\mathbf{1} \\ &= (\mathbf{Z}^\dagger)^T\left(\tfrac{1}{2}(\mathbf{d}_\mu - \mathbf{d}_\mathbf{x})\right)\end{aligned}$$

where the last equality follows from our “neat fact” used a few slides before

Thus, we can easily add new points to an MDS embedding!

Extensions of MDS

Classical MDS minimizes the loss function $\|\mathbf{X}^T\mathbf{X} - \mathbf{B}\|_{(F)}$

Many other choices for loss function exist

Perhaps the most common alternative is the ***stress*** function

$$\sum_{i,j} w_{ij}(d_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|_2)^2$$

where the w_{ij} are fixed weights

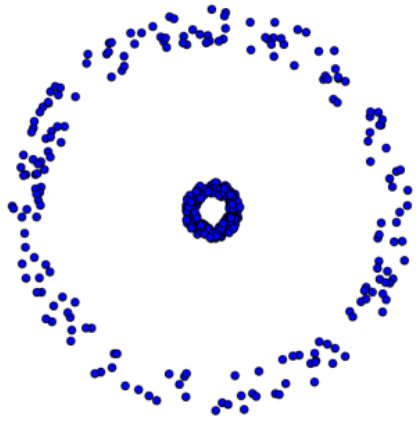
- can use $w_{ij} \in \{0, 1\}$ to handle missing data
- can set $w_{ij} = \frac{1}{d_{ij}^2}$ to more heavily penalize errors on nearby pairs of points

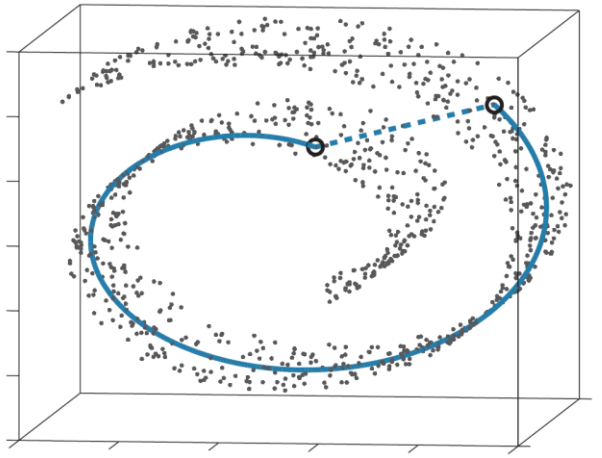
Stress criteria are typically minimized by iterative procedures

Nonlinear embeddings

The goal of embeddings/dimensionality reduction is to map a (potentially) high-dimensional dataset into a low-dimensional one in a way such that ***global*** and/or ***local*** geometric and topological properties are preserved

While PCA/MDS is the most popular method for this in practice, many high-dimensional data sets have ***nonlinear*** structure that is difficult to capture via linear methods





Kernel PCA

One approach to nonlinear dimensionality reduction is to “kernelize” PCA as we previously discussed

Map the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$ where Φ is a nonlinear mapping to a (typically) higher dimensional space

- Apply PCA to $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$ via $\mathbf{S} = \Phi(\mathbf{X})\Phi(\mathbf{X})^T$
  - requires explicitly computing $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$
- Apply MDS via $\mathbf{K} = \Phi(\mathbf{X})^T\Phi(\mathbf{X})$

  i.e., compute eigendecomposition of

$$\mathbf{B} = -\tfrac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H} = (\Phi(\mathbf{X})\mathbf{H})^T\Phi(\mathbf{X})\mathbf{H}$$
$$= \mathbf{H}\mathbf{K}\mathbf{H}$$

Summary of kernel PCA

Input: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, kernel k, desired dimension r

1. Form $\widetilde{\mathbf{K}} = \mathbf{HKH}$, where $\mathbf{K}$ is our kernel matrix and $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ is the "centering matrix" (Optional)
2. Compute eigendecomposition $\widetilde{\mathbf{K}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$
3. Set $\mathbf{Z}$ to be the first r rows of $(\mathbf{V}\mathbf{\Lambda}^{1/2})^T$

Output: A mapping $f : \mathbb{R}^d \to \mathbb{R}^r$ given by

$$f(\mathbf{x}) = (\mathbf{Z}^\dagger)^T \Phi(\mathbf{X})^T (\Phi(\mathbf{x}) - \Phi(\boldsymbol{\mu}))$$

$$= (\mathbf{Z}^\dagger)^T (\mathbf{k}(\mathbf{x}) - \tfrac{1}{n}\mathbf{K}\mathbf{1})$$

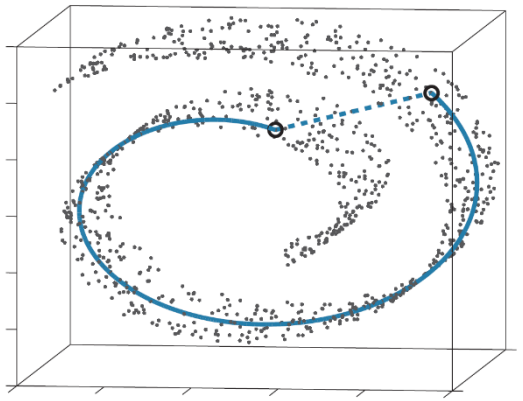
where $\mathbf{k}(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_n)]^T$

Isomap

Isometric feature **map**ping (Isomap) is another nonlinear dimensionality reduction technique that can be viewed as an extension of MDS

Assumes that the data lives on a low-dimensional ***manifold*** (also referred to as a technique for ***manifold learning***)

Given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, rather than computing the matrix $\mathbf{D}$ via $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$, Isomap tries to compute an estimate of the ***geodesic distance*** along the manifold



Estimating the geodesic distance

Geodesic distances are estimated by computing shortest paths in a ***proximity graph***

Form a matrix $\mathbf{W}$ as follows:

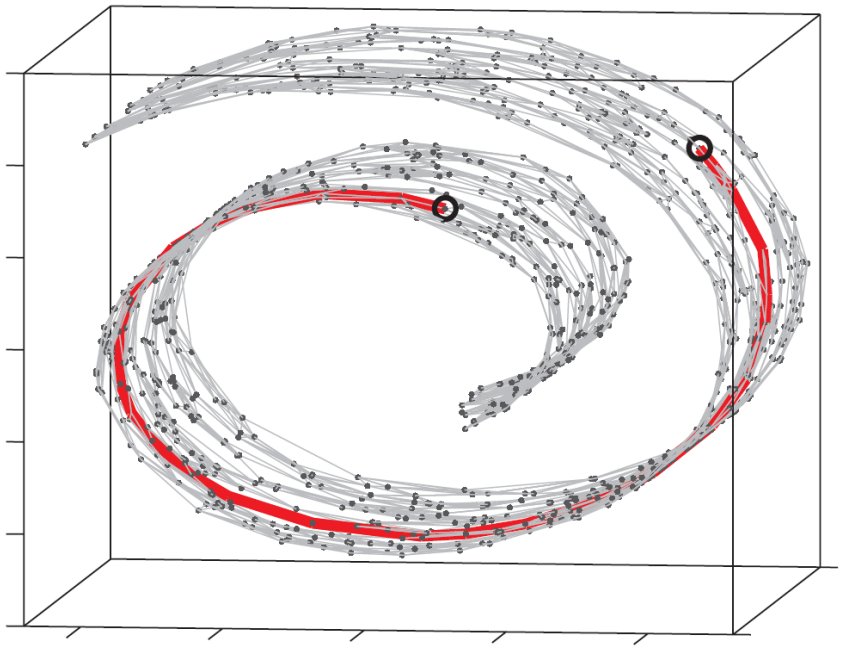
- for each $\mathbf{x}_i$, define a local neighborhood $\mathcal{N}_i$
  - k-nearest neighbors of $\mathbf{x}_i$
  - all $\mathbf{x}_j$ such that $\|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \epsilon$
- for each $\mathbf{x}_i$, set $w_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ for all $\mathbf{x}_j \in \mathcal{N}_i$

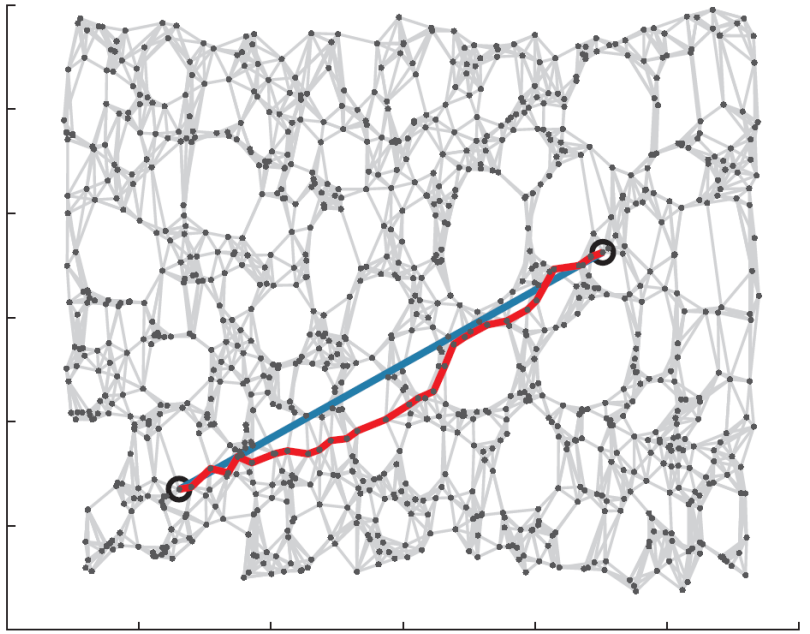
$\mathbf{W}$ represents the weighted adjacency matrix of a graph

Compute $\mathbf{D}$ by setting d_{ij} to be the length of the shortest path from node i to node j in the graph described by $\mathbf{W}$

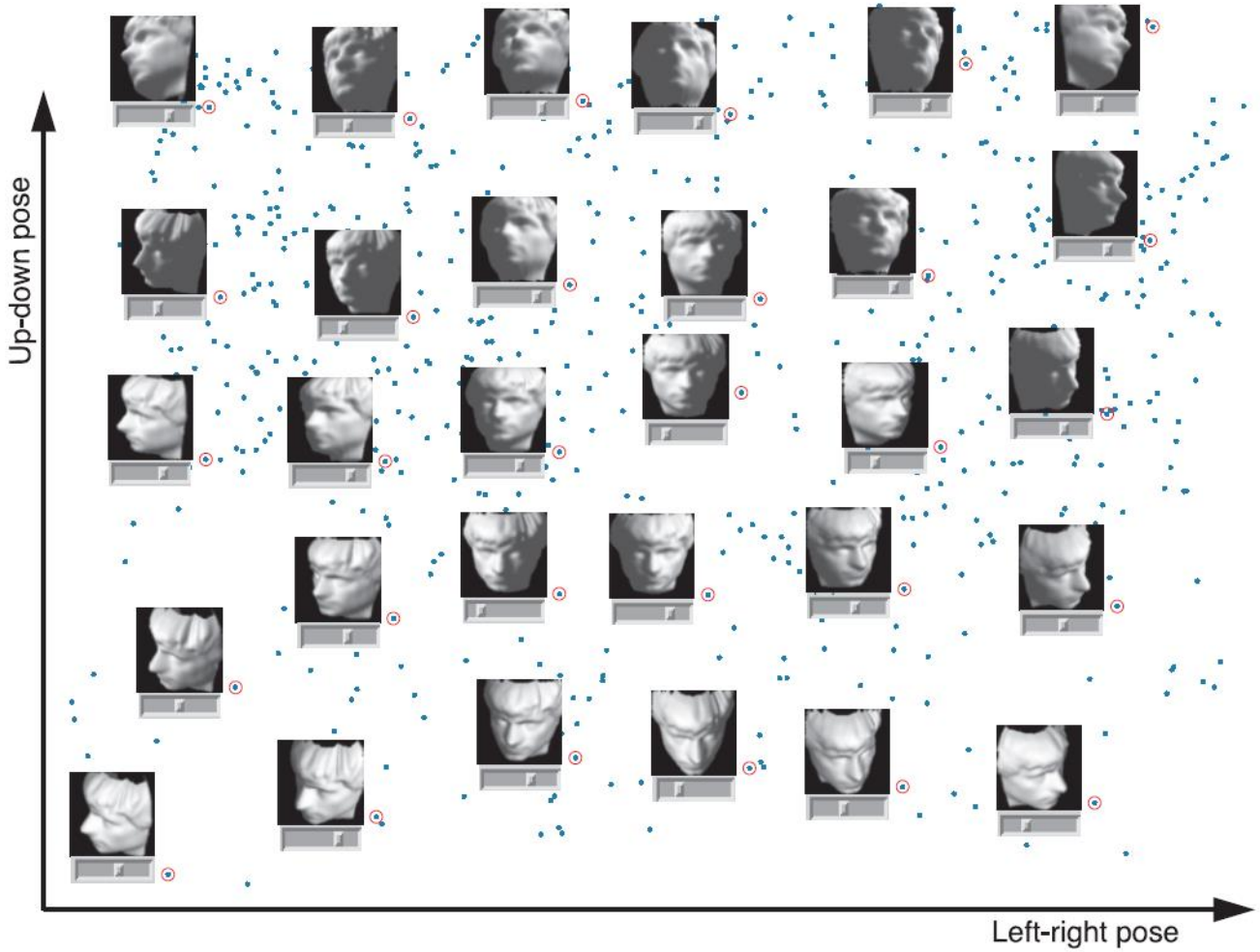
Out-of-sample extension using the same technique as MDS

Example: Swiss roll

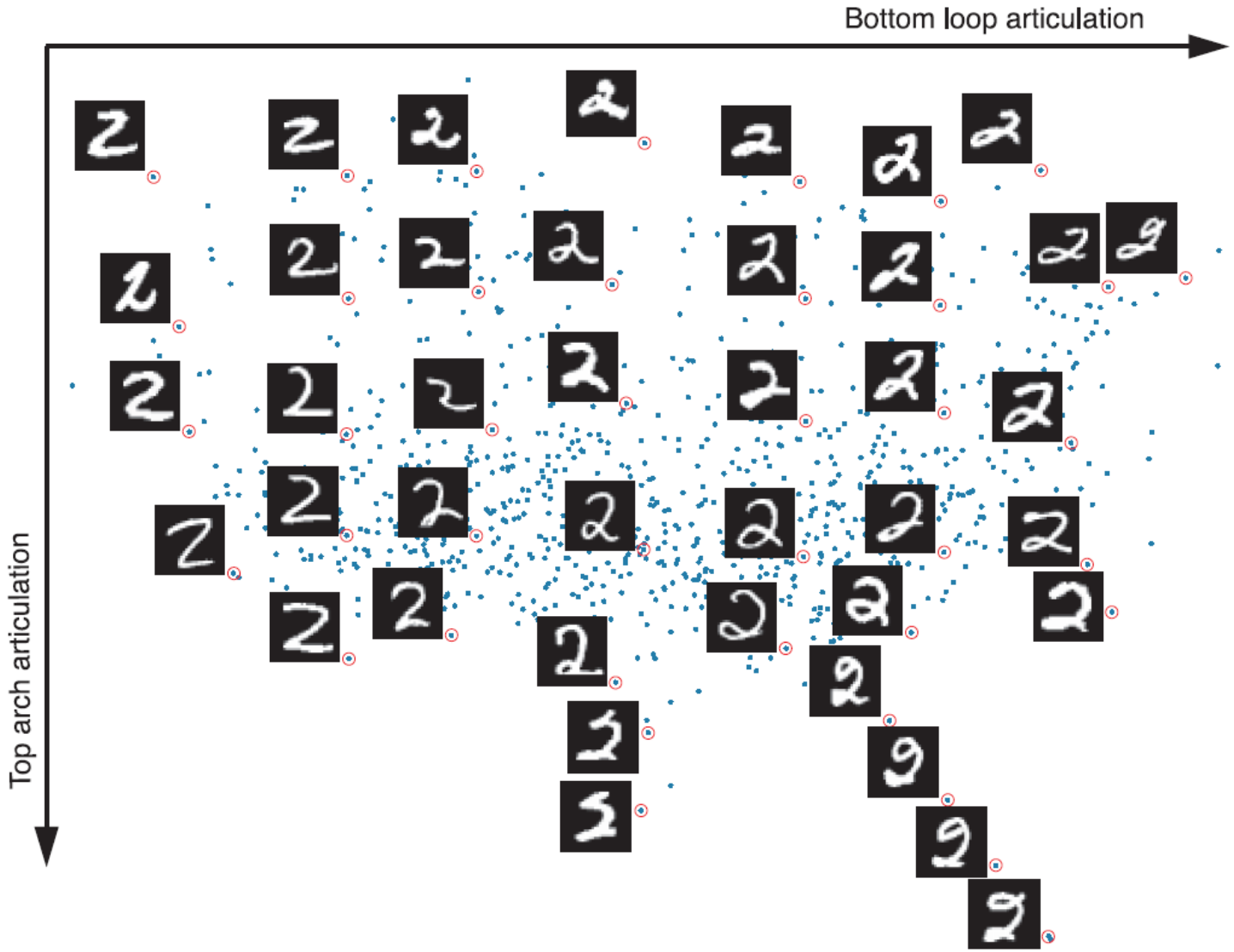




Example: Facial pose



Example: Handwritten digits



Locally linear embedding (LLE)

A potential challenge for Isomap is that estimates of the geodesic distance between points that are very far from each other on the manifold can grow increasingly inaccurate

Locally linear embedding (LLE) capitalizes on the intuition that a data manifold that is globally nonlinear will still appear linear in local pieces

LLE does not try to explicitly model global geodesic distances, but instead tries to preserve the structure in the data by trying to “patch together” local pieces of the manifold

The LLE algorithm

Given a data set $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, LLE consists of

1. For each $\mathbf{x}_i$, define a local neighborhood $\mathcal{N}_i$
2. Solve

$$\begin{aligned} \min_{\mathbf{W}} \quad & \sum_{i=1}^{n} \|\mathbf{x}_i - \textstyle\sum_{j=1}^{n} w_{ij}\mathbf{x}_j\|_2^2 \\ \text{s.t.} \quad & \sum_j w_{ij} = 1 \\ & w_{ij} = 0 \text{ for } j \notin \mathcal{N}_i \end{aligned}$$

constrained least squares problem

3. Fix $\mathbf{W}$ and solve

$$\min_{\{\mathbf{z}_i\}} \quad \sum_{i=1}^{n} \|\mathbf{z}_i - \textstyle\sum_{j=1}^{n} w_{ij}\mathbf{z}_j\|_2^2$$

eigenvalue problem

Another take on LLE

The eigenvalue problem at the heart of LLE can also (more compactly) be written as

$$\min_{\mathbf{Z}} \ \|\mathbf{Z} - \mathbf{Z}\mathbf{W}\|_F^2$$

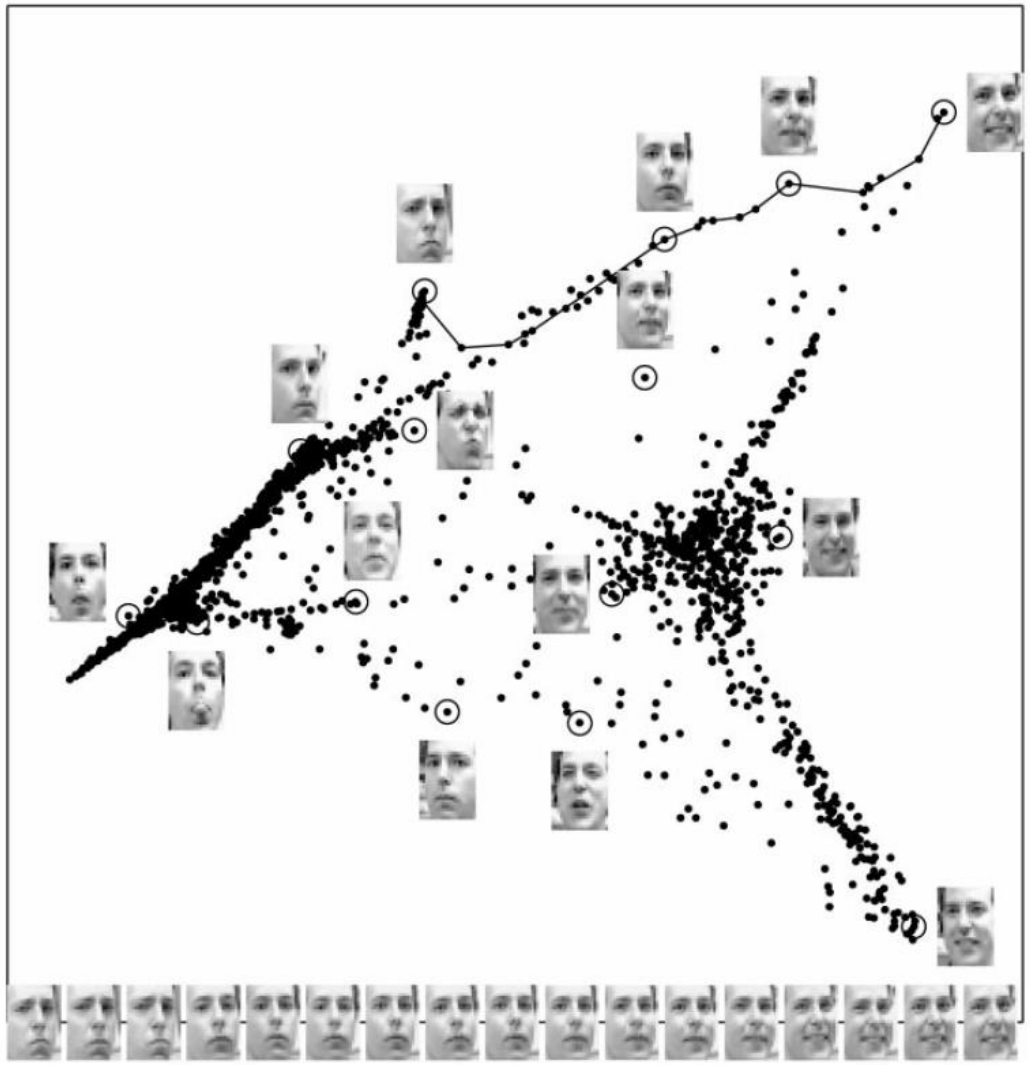
which in turn can also be written as

$$\min_{\mathbf{Z}} \ \text{trace}(\mathbf{Z}(\mathbf{I} - \mathbf{W}^T)(\mathbf{I} - \mathbf{W})\mathbf{Z}^T)$$

This is exactly the same type of problem we encountered in PCA, the solution to which can be obtained via an eigendecomposition of $(\mathbf{I} - \mathbf{W}^T)(\mathbf{I} - \mathbf{W})$

Note: Out-of-sample extension via $\mathbf{z}(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{z}_i w(\mathbf{x}, \mathbf{x}_i)$ where $w(\mathbf{x}, \mathbf{x}_i)$ are computed via the same constrained least squares problem as above

Example: Facial expression



Kernel PCA, Isomap, and LLE

- Kernel PCA
  - assumes linear embedding will work when using suitable features
- Isomap
  - emphasizes global distance preservation
  - can distort local geometry
- LLE
  - emphasizes local geometry preservation
  - can distort global geometry
    - far away points can get mapped close to each other
- Many other variants of nonlinear dimensionality reduction along these same lines have been developed
  - Laplacian eigenmaps, local tangent space alignment, diffusion maps, t-distributed stochastic neighbor embedding…