Principal component analysis (PCA)

Given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, let $\mathbf{X}$ denote the $d \times n$ matrix given by

$$\mathbf{X} = \begin{bmatrix} | & & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ | & & | \end{bmatrix} = \begin{bmatrix} | & & | \\ x_1 - \mu & \cdots & x_n - \mu \\ | & & | \end{bmatrix}$$

The goal in PCA is to find an optimal approximation

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

where $\boldsymbol{\mu} \in \mathbb{R}^d$, $\mathbf{A}$ is a $d \times k$ matrix with orthonormal columns (i.e., $\mathbf{A}^T\mathbf{A} = \mathbf{I}$), and $\boldsymbol{\theta}_i \in \mathbb{R}^k$ $k << d$

For data that is "centered" ($\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$ has been subtracted from each column), this reduces to $\mathbf{X} \approx \mathbf{A}\boldsymbol{\Theta}$

PCA as matrix factorization

The problem of determining $\mathbf{A}$ reduces to the optimization problem

$$\max_{\mathbf{A}} \ \text{trace}(\mathbf{A}^T \mathbf{S} \mathbf{A})$$

$$\text{s.t. } \mathbf{A}^T \mathbf{A} = \mathbf{I}$$

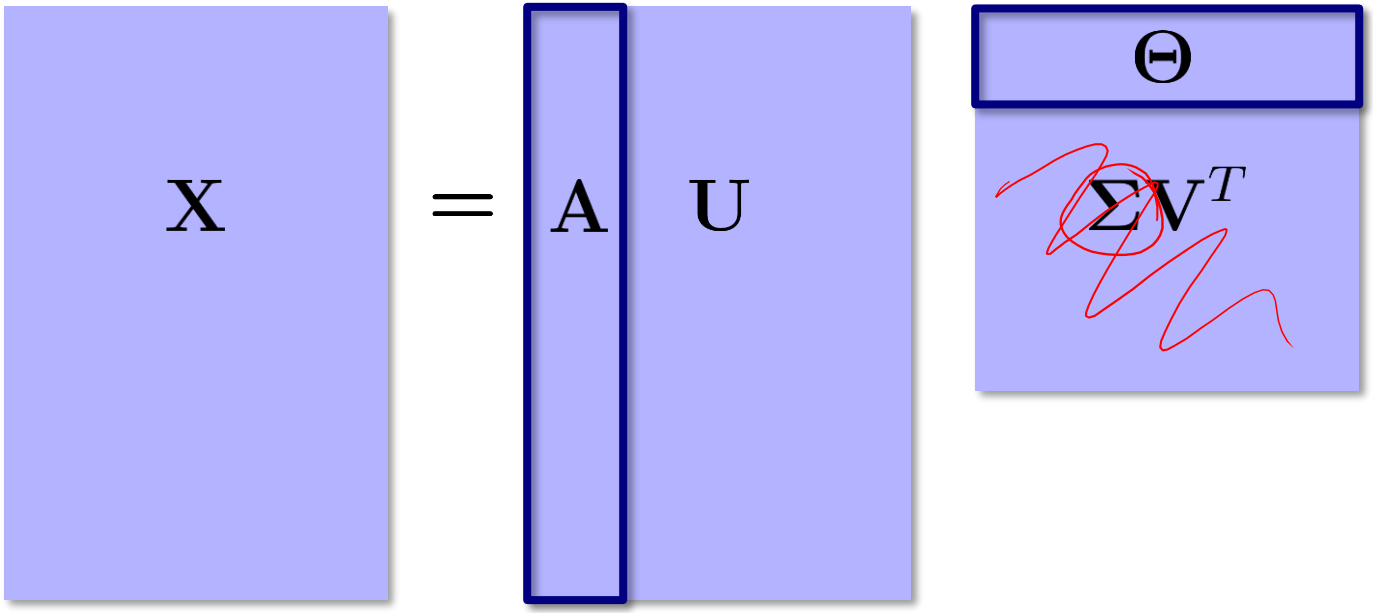
For "centered" data, the solution corresponds to taking $\mathbf{A}$ to be the top k eigenvectors of $\mathbf{S} = \mathbf{X}\mathbf{X}^T$

Note that by using the SVD $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we can also obtain this by taking $\mathbf{A}$ to be the first k columns of $\mathbf{U}$

(This also give us $\mathbf{\Theta}$ by taking the first k rows of $\mathbf{\Sigma}\mathbf{V}^T$)

PCA as matrix factorization

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



Multidimensional scaling (MDS)

We also considered the problem we observe only the "dissimilarity matrix" $\mathbf{D}$

In (classical) MDS, our goal is to find the $k \times n$ matrix $\mathbf{\Theta}$ directly from $\mathbf{D}$

To see how to do this, suppose first that we are given the matrix $\mathbf{D}^2$ whose entries are given by $d_{ij}^2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ for some $\mathbf{x}_1, \ldots, \mathbf{x}_n$

Last time we argued that under this assumption, we have

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

where $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ is the "centering matrix" and $\mathbf{X}\mathbf{H}$ is simply the data set $\mathbf{X}$ with the column mean subtracted off

MDS as matrix factorization

Thus, since

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

we can recover (a potentially rotated version of) $\mathbf{X}\mathbf{H}$ by simply computing an eigendecomposition:

$$\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \qquad \mathbf{X}\mathbf{H} = (\mathbf{V}\mathbf{\Lambda}^{1/2})^T$$

Moreover, $\mathbf{X}\mathbf{H}$ has been rotated so that the optimal embedding into only k dimensions is given by taking $\mathbf{\Theta}$ to be the first k rows of $\mathbf{X}\mathbf{H}$

Equivalence between PCA and MDS

Suppose we have (centered) data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and set $\mathbf{D}$ to be such that $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$

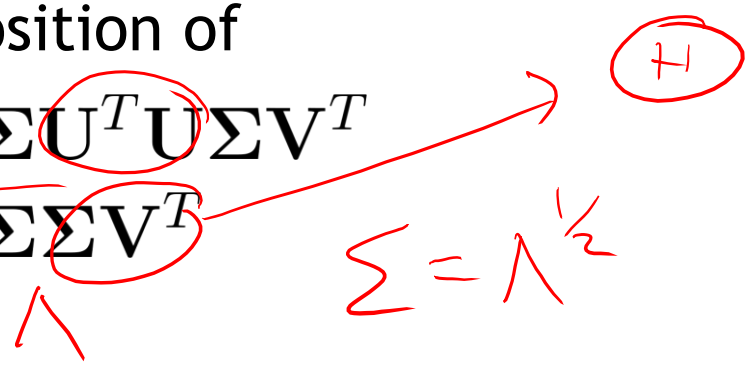
The result of classical MDS applied to $\mathbf{D}$ is the same (up to a rigid transformation) as applying PCA to $\mathbf{X}$

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

$\mathbf{\Theta}$ is the first k rows of $\mathbf{\Sigma}\mathbf{V}^T$

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 Θ

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

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

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

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

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

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

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

$$\Rightarrow \mathbf{A} = \mathbf{XH}\mathbf{\Theta}^\dagger$$

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

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

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

Is this enough?

Putting this all together, we can write

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

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 $\mathbf{d}_\mu - \mathbf{d_x}$:

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

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

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

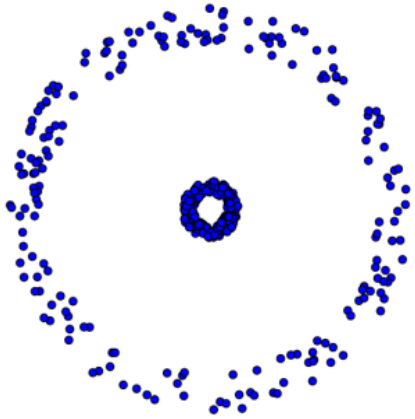
where the last equality follows from our “neat fact” used a few slides before

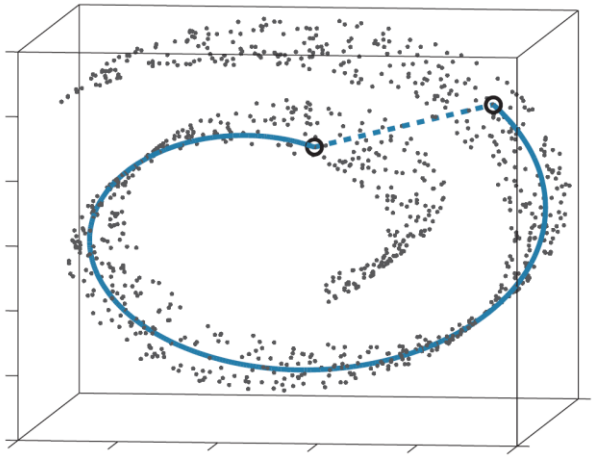
Thus, we can easily add new points to an MDS embedding!

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 in the same way that we did for SVMs

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 (now familiar) centering matrix
2. Compute eigendecomposition $\widetilde{\mathbf{K}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$
3. Set $\mathbf{\Theta}$ 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

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

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

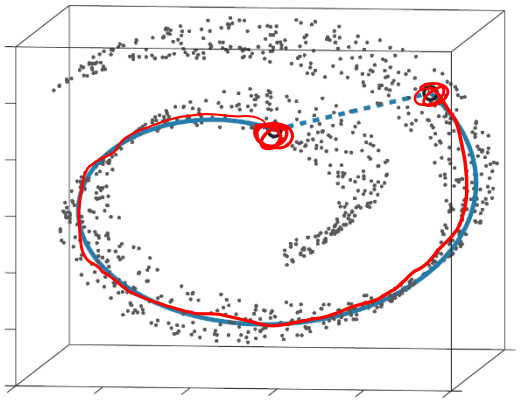
where $\mathbf{k}(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_n)]^T$

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:

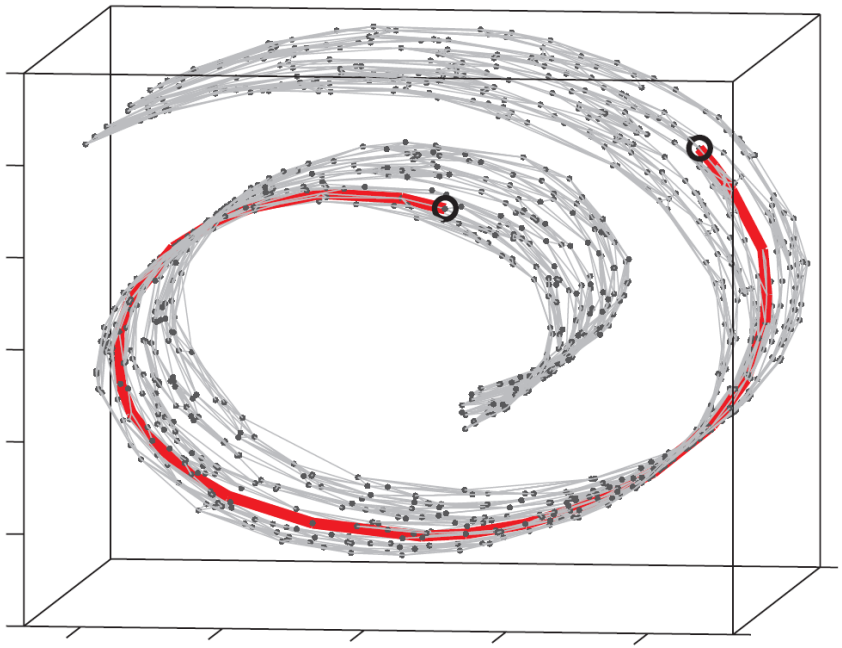
- for each $\mathbf{x}_i$, define a local neighborhood $\mathcal{N}_i$
  - k-nearest neighbors of $\mathbf{x}_i$
  - all $\mathbf{x}_j$ such that $\|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \epsilon$
- for each $\mathbf{x}_i$, set $w_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ for all $\mathbf{x}_j \in \mathcal{N}_i$

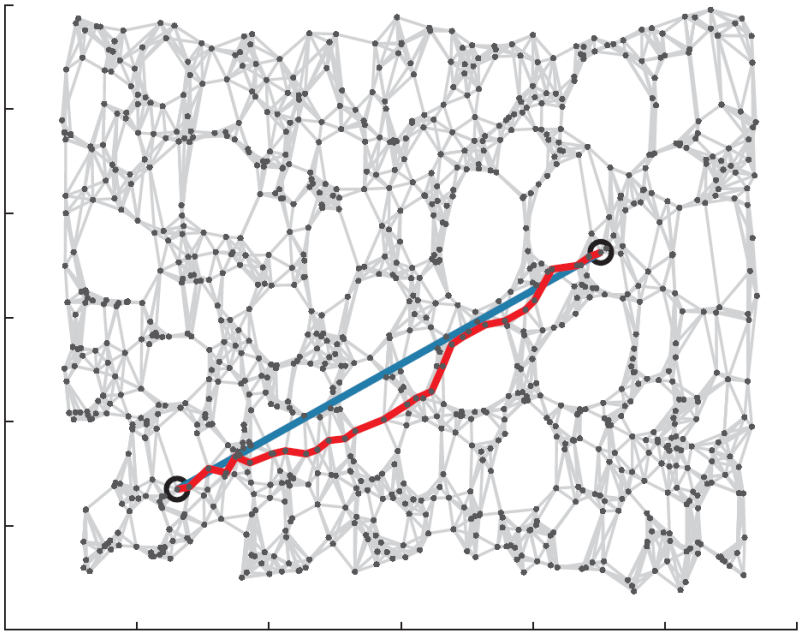
$\mathbf{W}$ represents the weighted adjacency matrix of a graph

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 $\mathbf{W}$

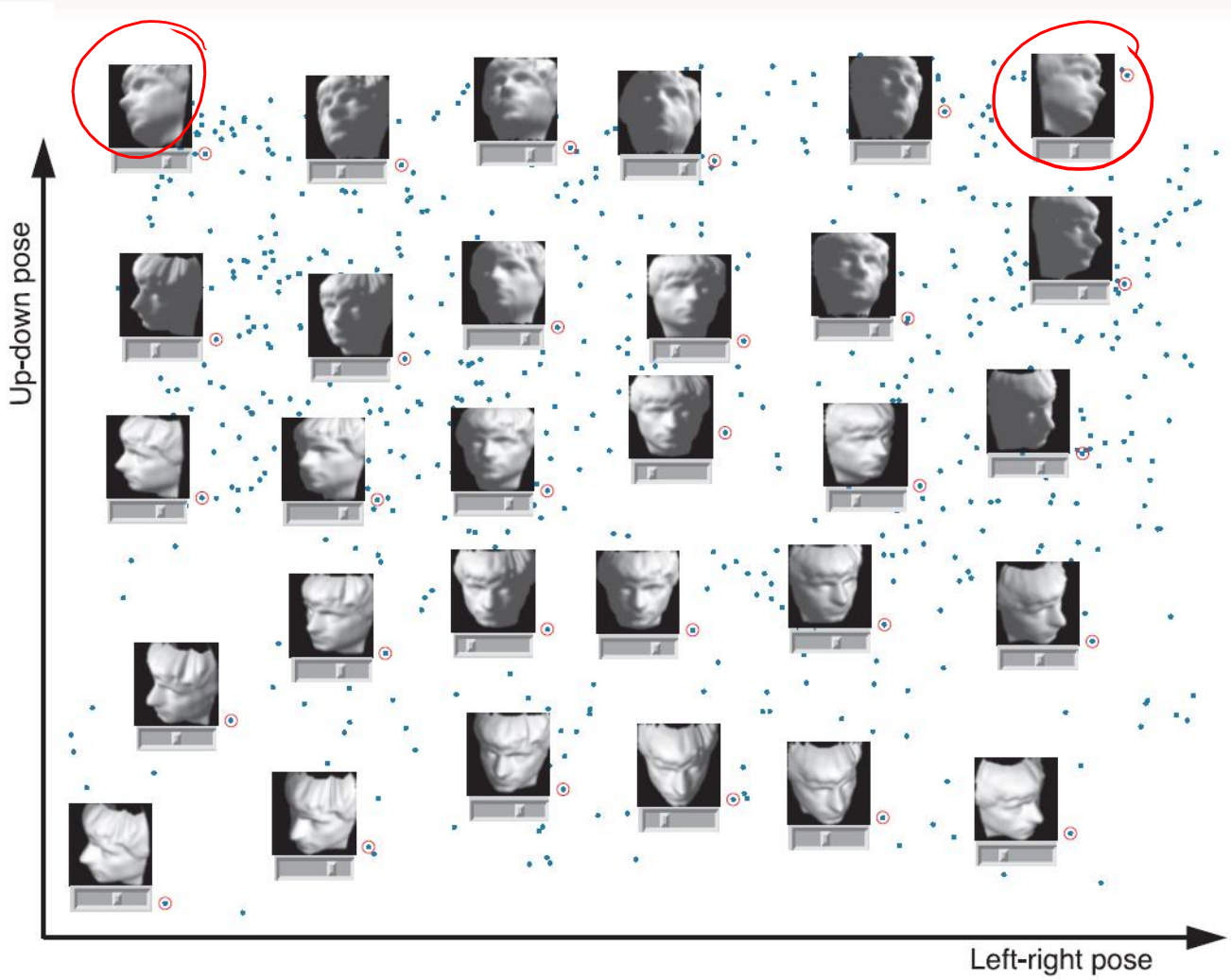
Out-of-sample extension using the same technique as MDS

Example: Swiss roll

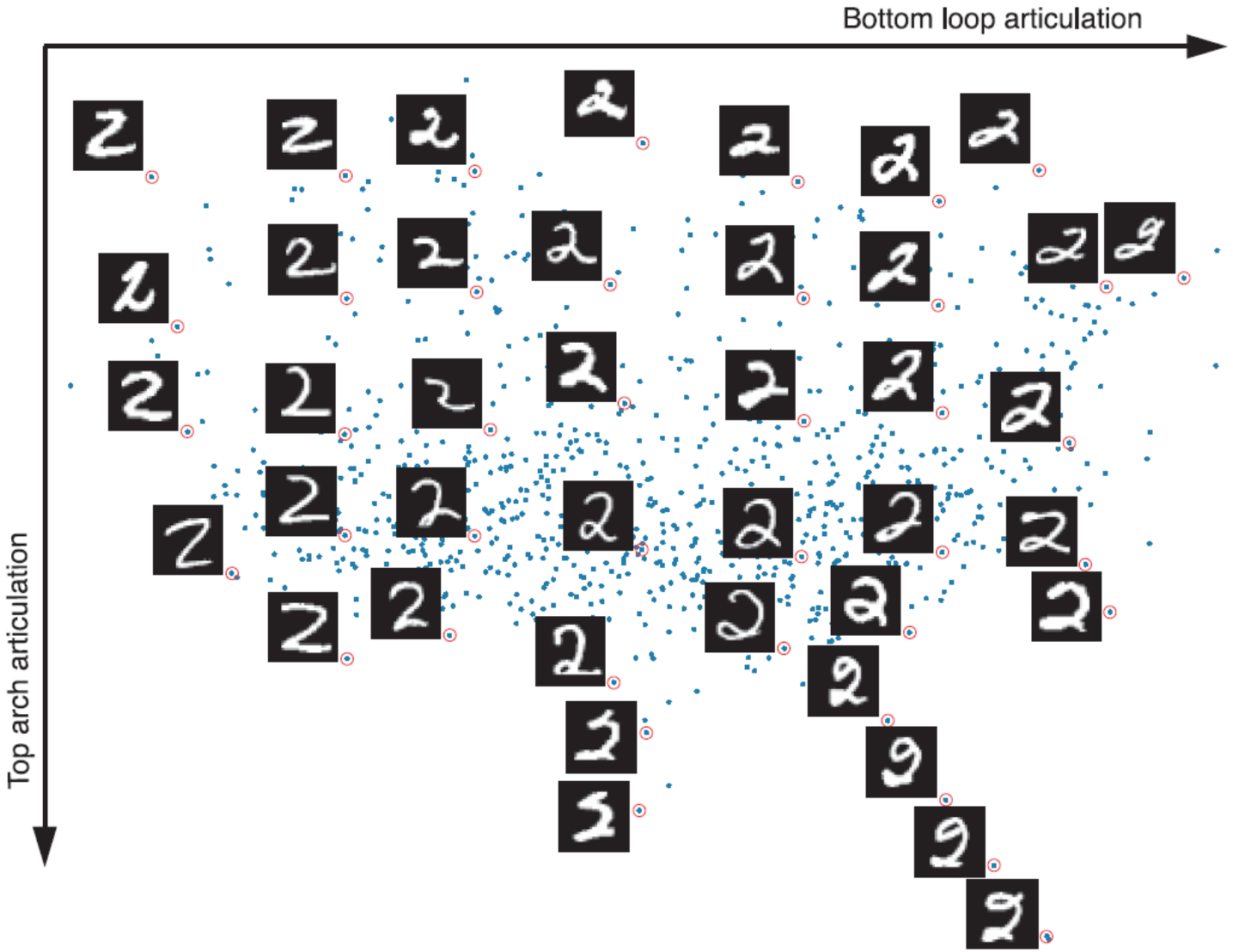




Example: Facial pose



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

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

eigenvalue problem

Another take on LLE

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

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

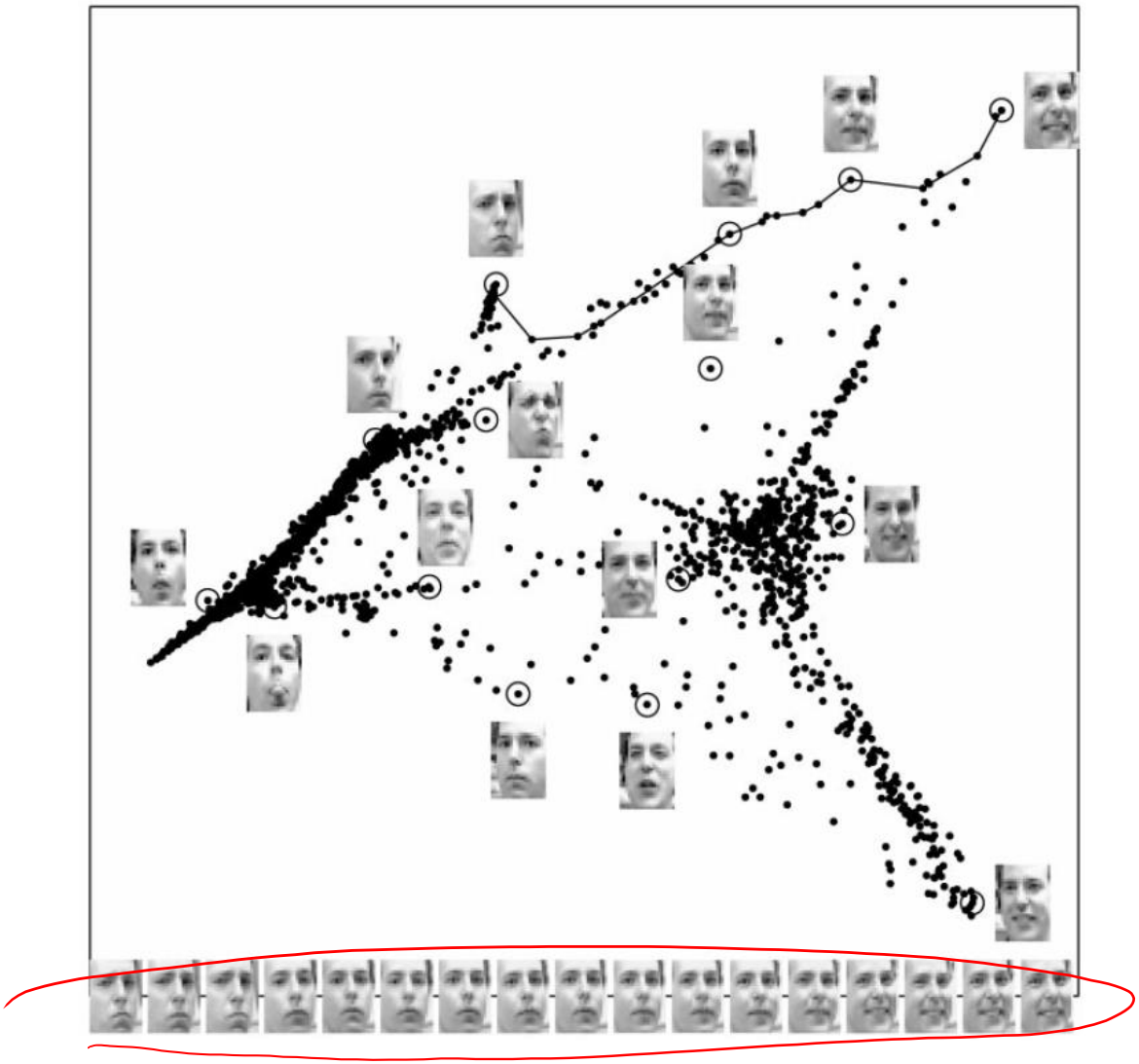
which in turn can also be written as

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

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 $\boldsymbol{\theta}(\mathbf{x}) = \sum_{i=1}^{n} \boldsymbol{\theta}_i w(\mathbf{x}, \mathbf{x}_i)$ 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, Hessian LLE, diffusion maps, …