Puzzle: Time-series forecasting

Suppose we wish to predict whether the price of a stock is going to go up or down tomorrow

- Take history over a long period of time
- Normalize the time series to zero mean, unit variance
- Form all possible input-output pairs with
  - input = previous 20 days of stock prices
  - output = price movement on the 21st day
- Randomly split data into training and testing data
- Train on training data only, test on testing data only

Based on the test data, it looks like we can consistently predict the price movement direction with accuracy ~52%

Are we going to be rich?

Principal component analysis (PCA)

- Unsupervised
- Linear
- Loss criteria: Sum of squared errors

The idea behind PCA is to find an approximation

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

where

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

Derivation of PCA

Mathematically, we can define $\boldsymbol{\mu}, \mathbf{A}$ and $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ as the solution to

$$\min_{\boldsymbol{\mu}, \mathbf{A}, \{\boldsymbol{\theta}_i\}} \quad \sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\theta}_i\|_2^2$$

The hard part of this problem is finding $\mathbf{A}$

Given $\mathbf{A}$, it is relatively easy to show that

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

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

Determining $\mathbf{A}$

It remains to minimize

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})\|_2^2$$

with respect to $\mathbf{A}$

For convenience, we will assume that $\boldsymbol{\mu} = \mathbf{0}$, since otherwise we could just substitute $\widetilde{\mathbf{x}}_i = \mathbf{x}_i - \boldsymbol{\mu}$

In this case the problem reduces to minimizing

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i\|_2^2$$

Determining $\mathbf{A}$

Expanding this out, we obtain

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i\|_2^2 = \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i)^T(\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i)$$

$$= \sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i + \mathbf{x}_i^T\mathbf{A}\underbrace{\mathbf{A}^T\mathbf{A}}_{\mathbf{A}^T\mathbf{A}=\mathbf{I}}\mathbf{A}^T\mathbf{x}_i$$

$$= \sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{x}_i - \mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i$$

Thus, we can instead focus on maximizing

$$\sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i$$

Determining $\mathbf{A}$

Note that for any vector $\mathbf{v}$, we have $\|\mathbf{v}\|_2^2 = \text{trace}(\mathbf{v}\mathbf{v}^T)$

Thus, we can write

$$\begin{aligned}
\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{A}\mathbf{A}^T \mathbf{x}_i &= \sum_{i=1}^{n} \|\mathbf{A}^T \mathbf{x}_i\|_2^2 \\
&= \sum_{i=1}^{n} \text{trace}(\mathbf{A}^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{A}) \\
&= \text{trace}\left(\mathbf{A}^T \left(\textstyle\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T\right) \mathbf{A}\right) \\
&= \text{trace}\left(\mathbf{A}^T \mathbf{S} \mathbf{A}\right)
\end{aligned}$$

$\mathbf{S} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T$ is a scaled version of the empirical covariance matrix, sometimes called the ***scatter*** matrix

Determining $\mathbf{A}$

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

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

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

Analytically deriving the optimal $\mathbf{A}$ is not too hard, but is a bit more involved than you might initially expect (especially if you already know the answer)

We will provide justification for the solution for the $k = 1$ case – the general case is proven in the supplementary notes

One-dimensional example

Consider the optimization problem

$$\max_{\mathbf{a}} \ \mathbf{a}^T \mathbf{S} \mathbf{a}$$

$$\text{s.t. } \mathbf{a}^T \mathbf{a} = 1$$

Form the Lagrangian $\mathcal{L}(\mathbf{a}) = \mathbf{a}^T \mathbf{S} \mathbf{a} + \lambda(\mathbf{a}^T \mathbf{a} - 1)$

Take the gradient and set it equal to zero

$$\mathbf{S}\mathbf{a} + \lambda \mathbf{a} = 0$$

$\Rightarrow$ $\mathbf{a}$ must be an eigenvector of $\mathbf{S}$

Take $\mathbf{a}$ to be the eigenvector of $\mathbf{S}$ corresponding to the maximal eigenvalue

The general case

For general values of k, the solution is obtained by computing the eigendecomposition of $\mathbf{S}$:

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

where $\mathbf{U}$ is an orthonormal matrix with columns $\mathbf{u}_1, \ldots, \mathbf{u}_d$ and

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_d \end{bmatrix}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$

The general case

The optimal choice of $\mathbf{A}$ in this case is given by

$$\mathbf{A} = [\mathbf{u}_1, \ldots, \mathbf{u}_k]$$

i.e., take the top k eigenvectors of $\mathbf{S}$

Terminology

- principal component transform: $\mathbf{x} \rightarrow \boldsymbol{\theta} = \mathbf{A}^T(\mathbf{x} - \boldsymbol{\mu})$
- j^{th} principal component: $\theta(j) = \mathbf{u}_j^T(\mathbf{x} - \boldsymbol{\mu})$
- j^{th} principal eigenvector: $\mathbf{u}_j$

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 $r = \min(d, n)$ and

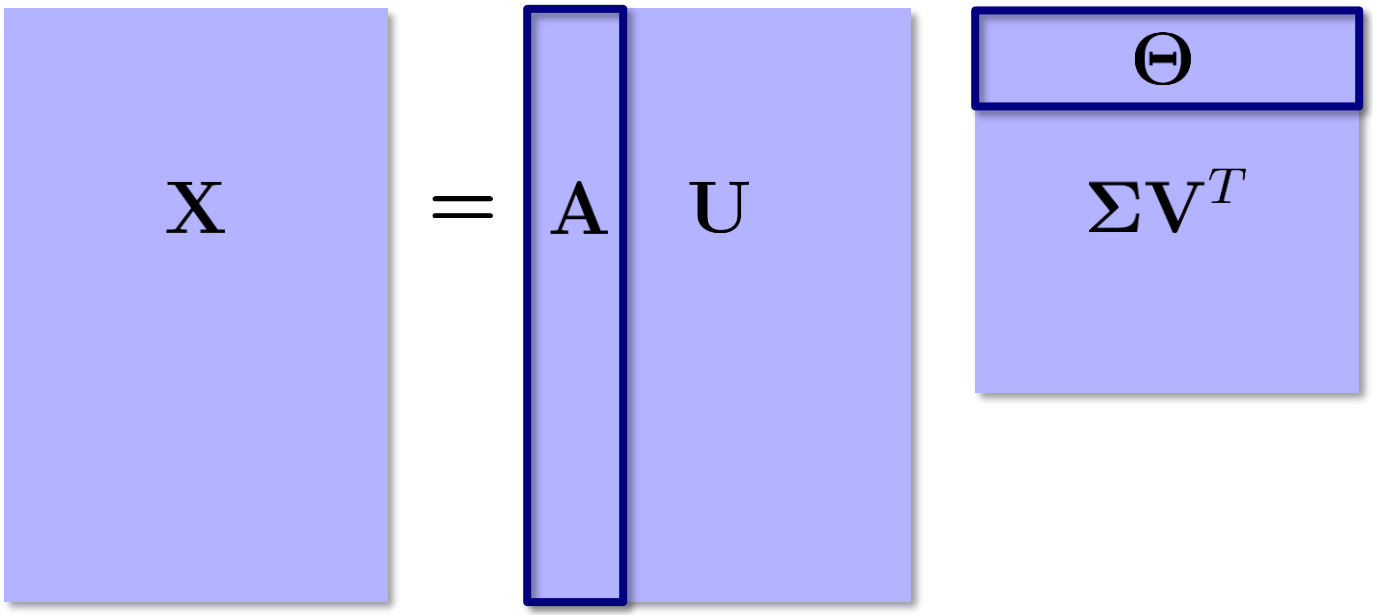
$$\sigma_i = i^{\text{th}} \text{ singular value}$$

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

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

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



Practical matters

It is customary to ***center*** and ***scale*** a data set so that it has zero mean and unit variance along each feature

This puts all features on an "equal playing field"

These steps may be omitted when

- The data are known to be zero mean
- The data are known to have comparable units of measurement

To select k, we typically choose it to be large enough so that

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\theta}_i\|_2^2 = n(\lambda_{k+1} + \ldots + \lambda_d)$$

is sufficiently small

When to use PCA

- When the data form a single “point cloud” in space
- When the data are approximately Gaussian, or some other “elliptical” distribution
- When low-rank subspaces capture the majority of the variation

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)***

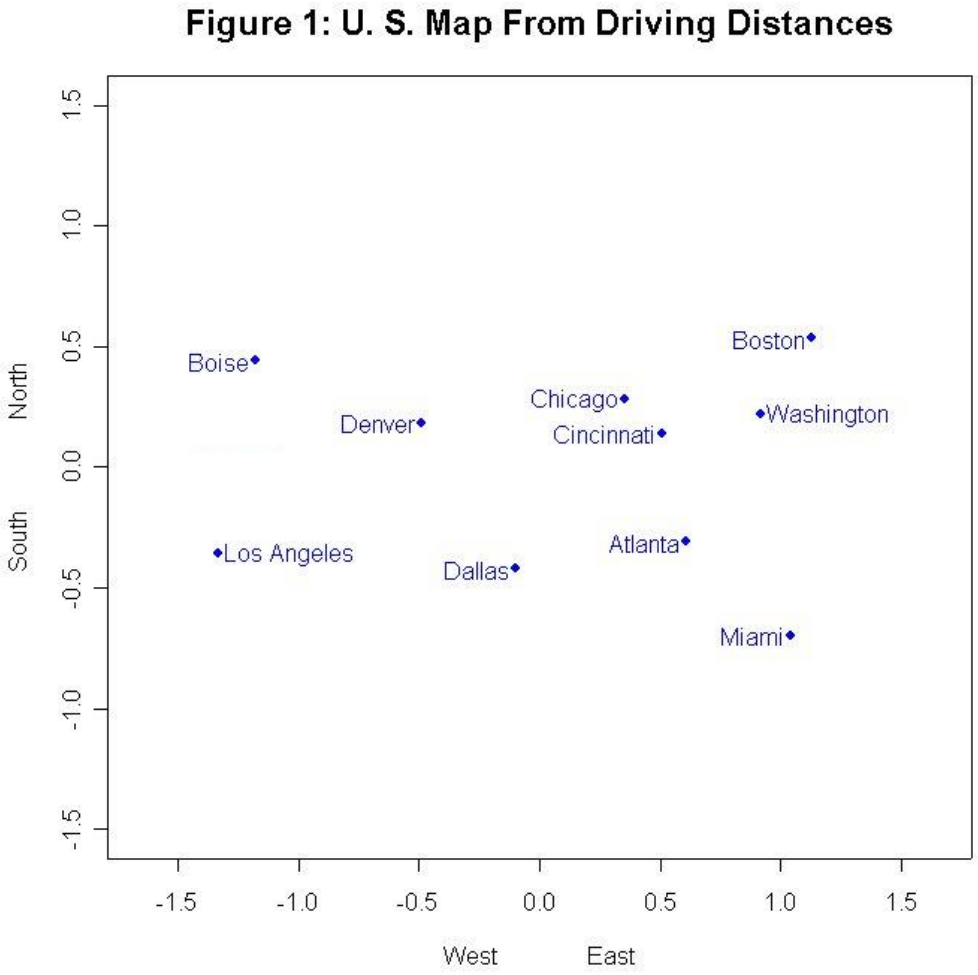
There are a number of variants of MDS based on

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

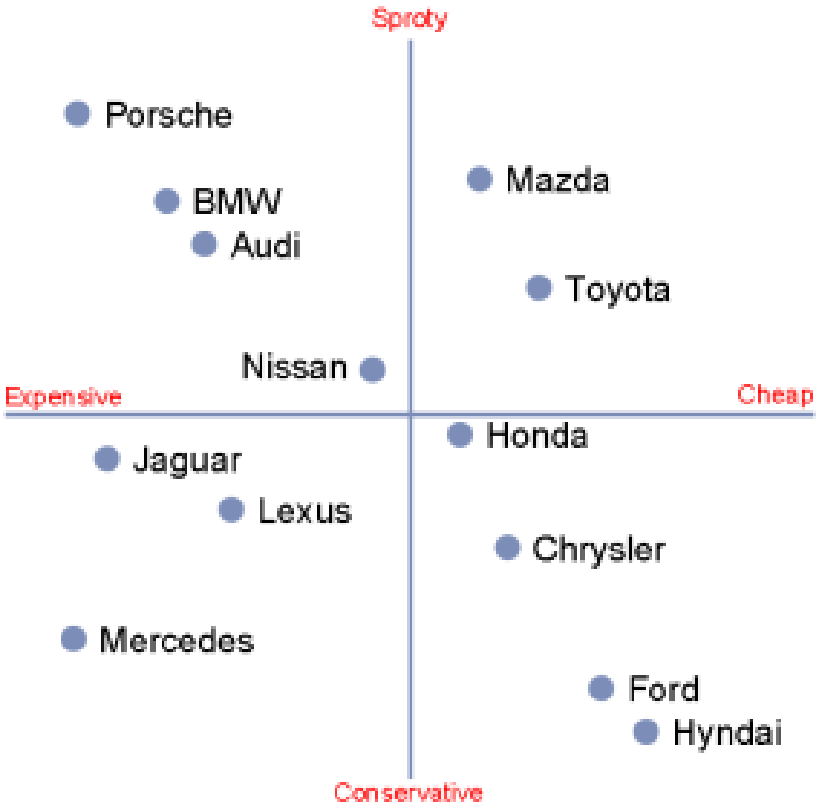
Main distinction

- ***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 $i = 1, \ldots, k$

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

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

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

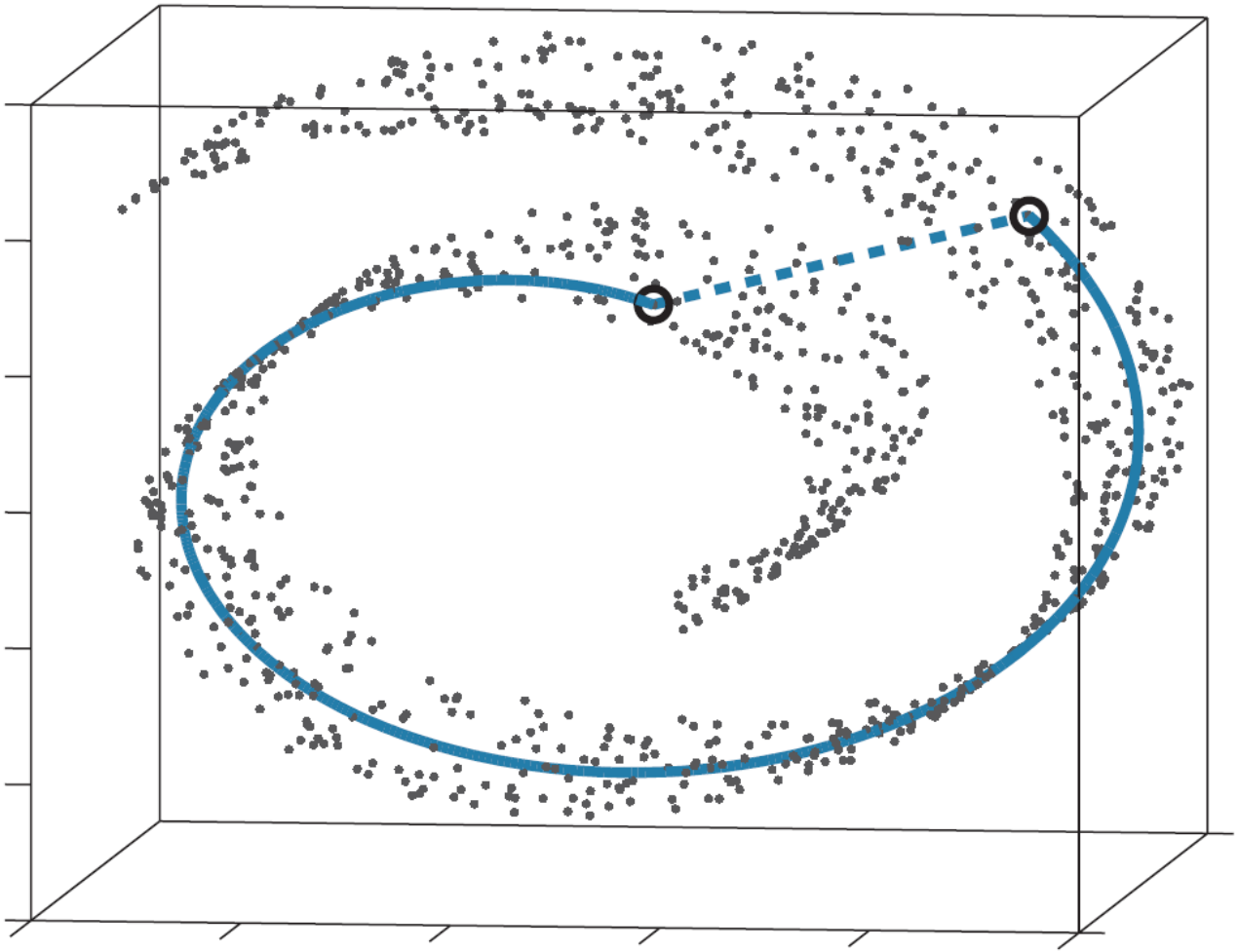
Stress criteria are typically minimized by iterative procedures

Nonlinear embeddings

The goal of embeddings/dimensionality reduction is to map a (potentially) high-dimensional dataset into a low-dimensional one in a way that preserves ***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

Example: Swiss roll



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 $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$

$$\mathbf{x}_i = \begin{bmatrix} x_i(1) \\ \vdots \\ x_i(d) \end{bmatrix} \xrightarrow{\Phi} \Phi(\mathbf{x}_i) = \begin{bmatrix} \Phi^{(1)}(\mathbf{x}_i) \\ \\ \vdots \\ \\ \Phi^{(p)}(\mathbf{x}_i) \end{bmatrix}$$

where Φ is nonlinear and $d \ll p$

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

Isomap

Isometric feature **map**ping is essentially just the application of MDS to dissimilarities which are derived not from the raw data but based on shortest paths in a ***proximity graph***

- e.g., a graph containing edges only between a data point and its nearest-neighbors

This path length is viewed as an approximation to a ***geodesic distance*** on the underlying data ***manifold***

Reference: Tennenbaum, de Silva, and Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction" *Science*, 2000.

The idea behind Isomap is most easily seen by returning to our "Swiss roll" example

Example: Swiss roll