https://github.com/tedinburgh/ads2023

Non-linear dimensionality reduction

Tom Edinburgh
te269

Example sheets and coursework are coming this week

Today: Non-linear dimensionality reduction

- Recap: power iteration
- Kernel PCA
- Multidimensional scaling
- Isomap
- t-SNE and UMAP

Questions: halfway through, at the end, or by email (te269)

Resources

- Slides adapted from:
  - Prof Stephen Eglen, Cambridge
  - Ethan Fetaya/James Lucas/Emad Andrews, Toronto

Recap: Empirical covariance and eigenvectors

- Q is the empirical covariance matrix of data X: $Q = X^T X/(n-1)$
- Eigenvectors (PCs) $e_1, e_2, \ldots, e_p$ with eigenvalues $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_p$
- Why $1/(n-1)$ rather than $1/n$? In practice, it doesn’t really matter which
- Empirical covariance = sample covariance
- $1/(n-1)$ means that the empirical covariance is **unbiased** (because we have to estimate the sample mean)

Recap: Power iteration

- Power iteration is an eigenvalue algorithm to estimate the largest eigenvalue λ_1 of A, i.e. $Aw_1 = \lambda_1 w_1$

- Start with a vector b_0 and update by the recurrence relation $b_{k+1} = \dfrac{Ab_k}{\|Ab_k\|}$.

  Then b_k converges to w_1 as $k \to \infty$ (assuming $b_0 \cdot w_1 \neq 0$) and $c_k = \dfrac{b_k^T A b_k}{b_k^T b_k}$ converges to λ_1

- Here A is a $n \times n$ matrix (but this process will work for the $p \times p$ matrix Q)

Recap: Power iteration

- $b_{k+1} = \frac{Ab_k}{\|Ab_k\|} = \frac{A^{k+1}b_0}{\|A^{k+1}b_0\|}$ and $b_0 = c_1e_1 + c_2e_2 + \ldots + c_ne_n$ ($c_0 \neq 0$) where $e_1, e_2, \ldots, e_n$ are eigenvectors with eigenvalues $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_n$

$$A_kb_0 = A^k(c_1e_1 + \ldots + c_ne_n) = c_1A^ke_1 + \ldots c_nA^ke_n = c_1\lambda_1^ke_1 + \ldots c_n\lambda_n^ke_n$$

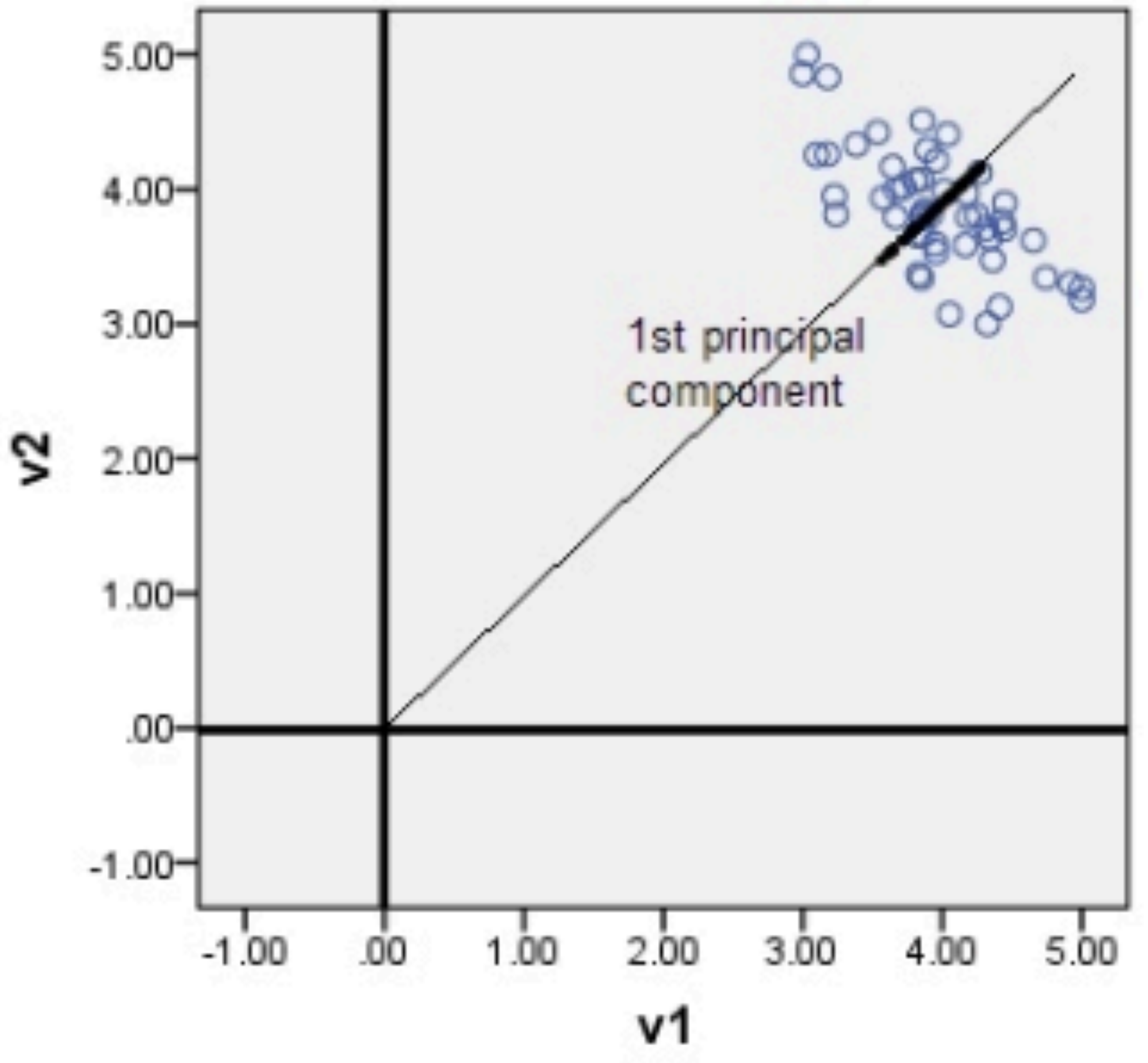
$$A_kb_0 = c_1\lambda_1^k\left(v_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k + \ldots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k\right) \rightarrow c_1\lambda_1^kv_1$$

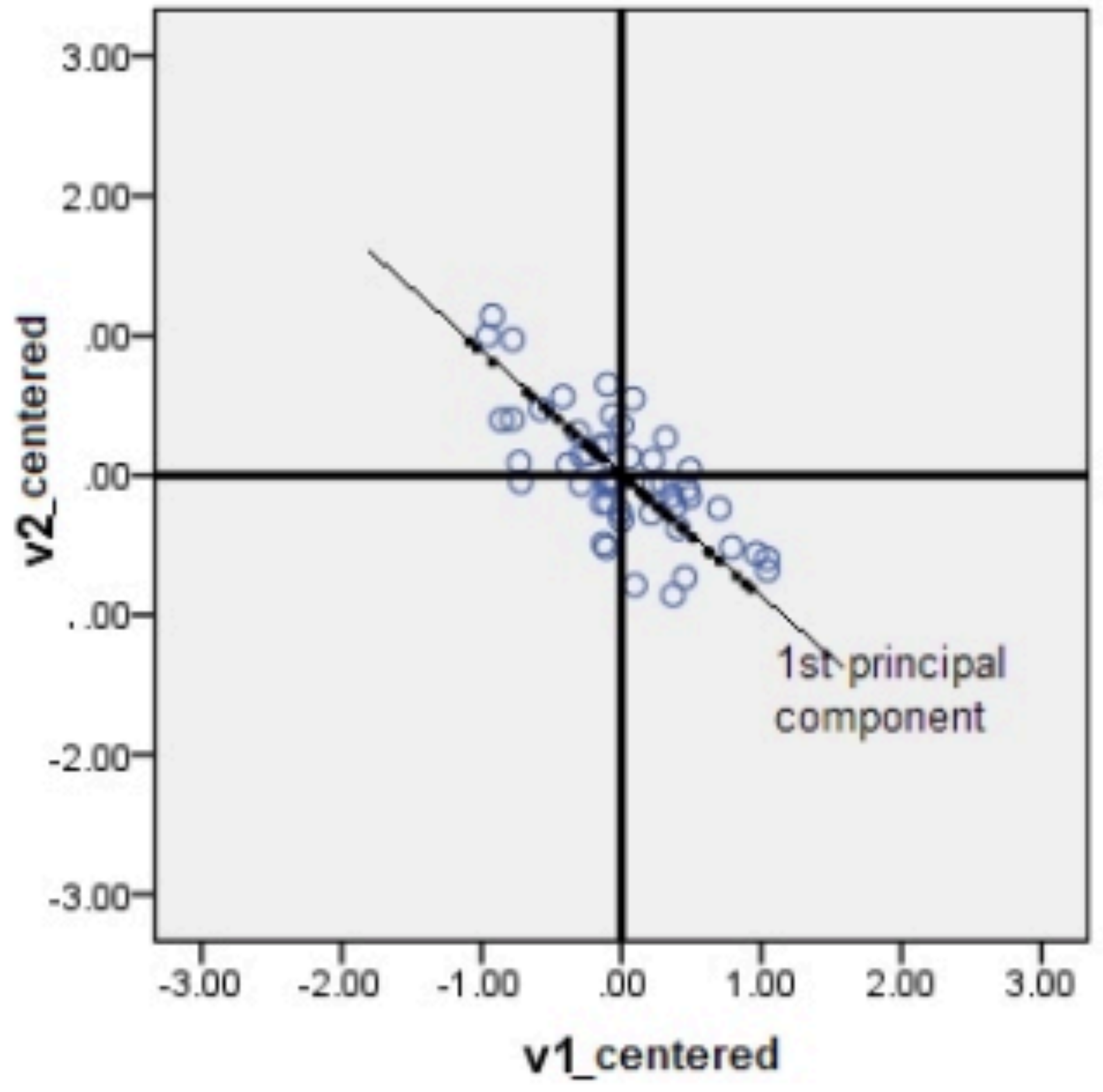
- So $b_k = \frac{A^{k+1}b_0}{\|A^{k+1}b_0\|} \rightarrow \alpha v_1$ for some constant α. But $|b_k| = \frac{\|A^{k+1}b_0\|}{\|A^{k+1}b_0\|} = 1$

Format of the data

$$X = \begin{pmatrix} X_1 & X_2 & \cdots & X_p \end{pmatrix} = \begin{pmatrix} x_1^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

- PCA depends on scaling of each variable
- Standardise each variable (i.e. subtract mean, divide by variance) so that $\sum_{i=1}^{n} x_{ij} = 0$ and $\sum_{i=1}^{n} x_{ij}^2 = 1$ for all $j = 1, \dots, p$.



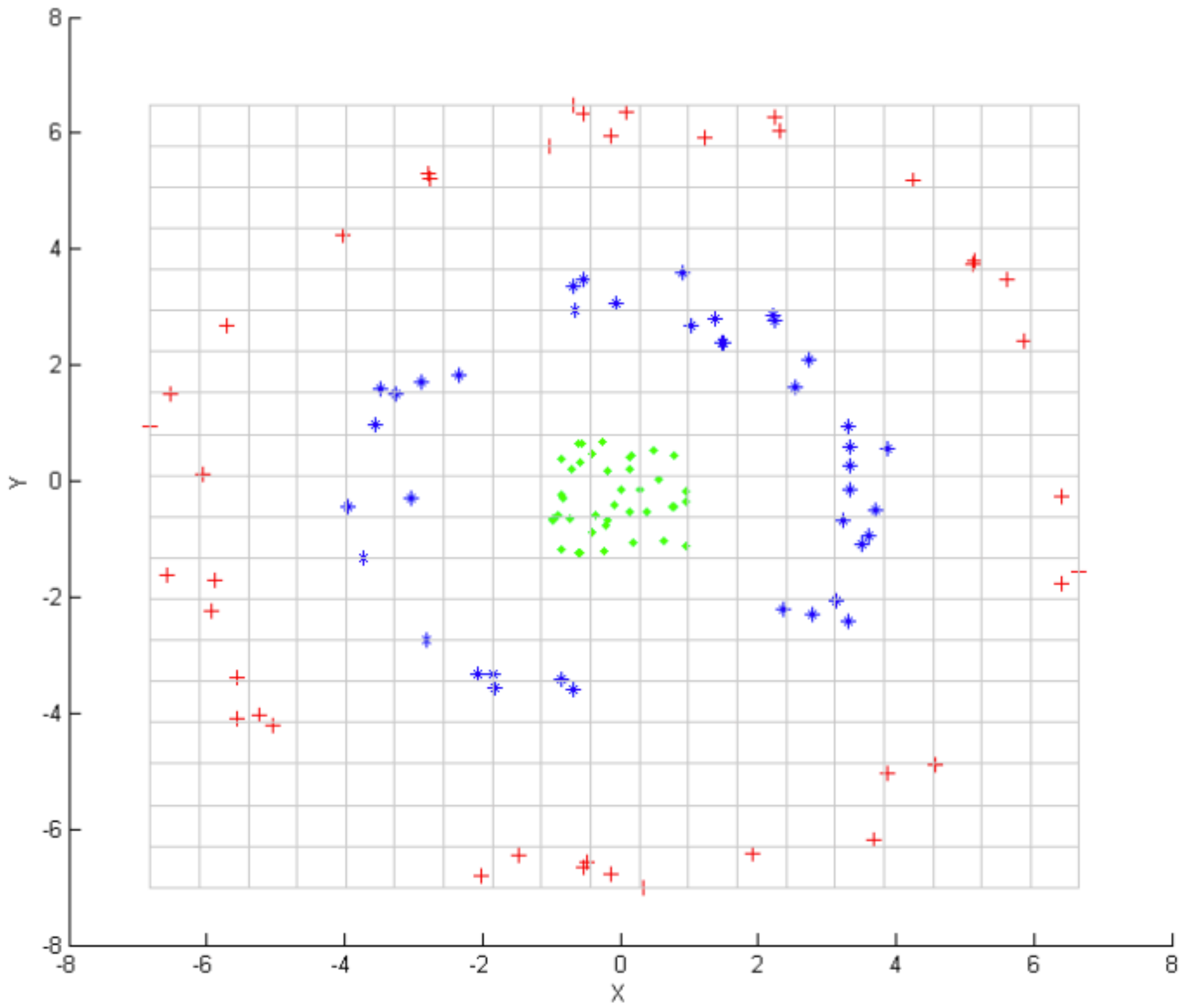


When is PCA not particularly good? (And why?)

When is PCA not particularly good? (And why?)

- PCA captures linear correlations between features but fails when the linearity assumption isn't valid

- We could transform the data first to restore linearity, then apply PCA afterwards



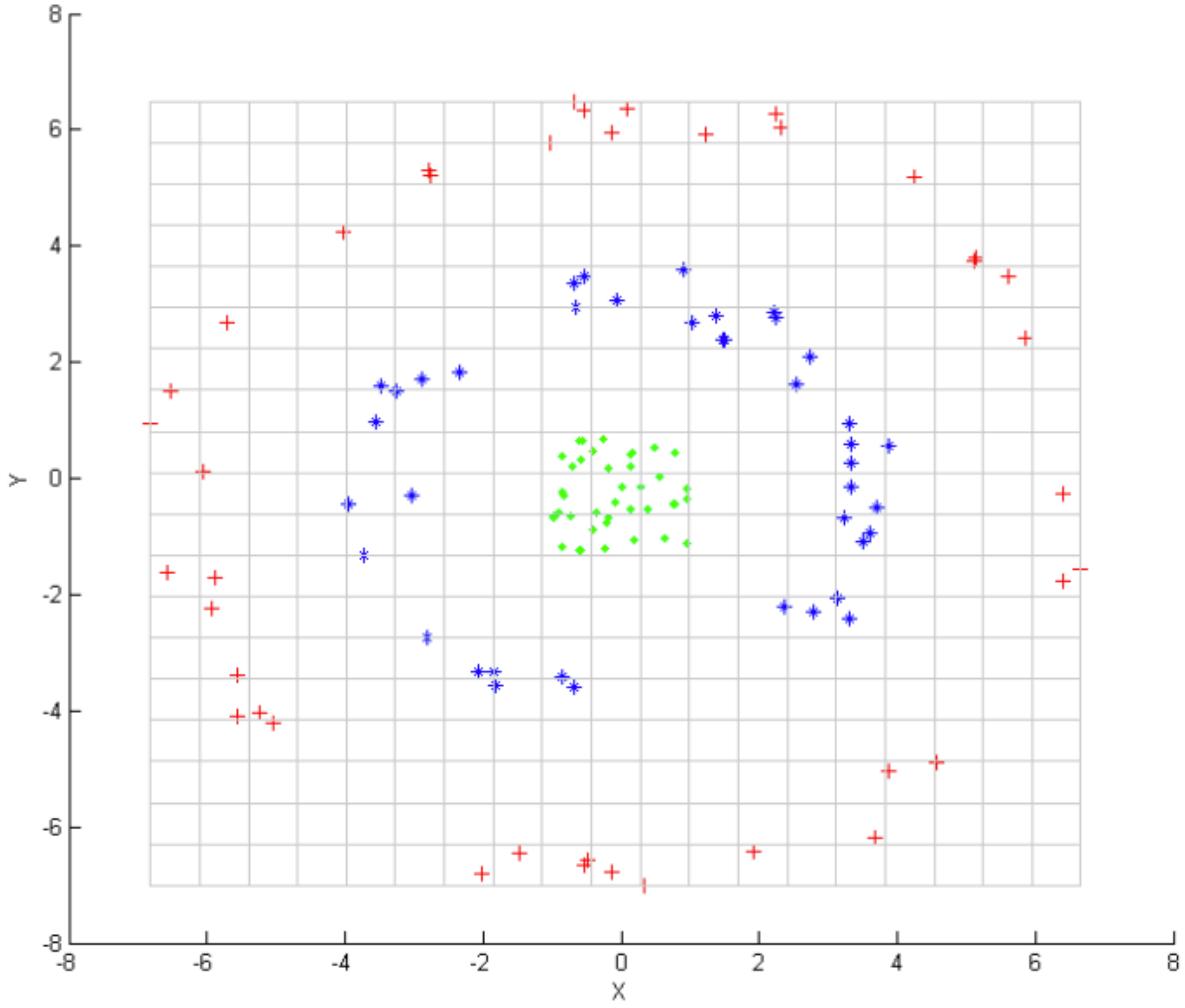
Kernel PCA

Kernel PCA

- PCA captures linear correlations between features but fails when the linearity assumption isn't valid

- We could transform the data first to restore linearity, then apply PCA afterwards

- This is called kernel PCA (applied to kernel K rather than covariance Q)

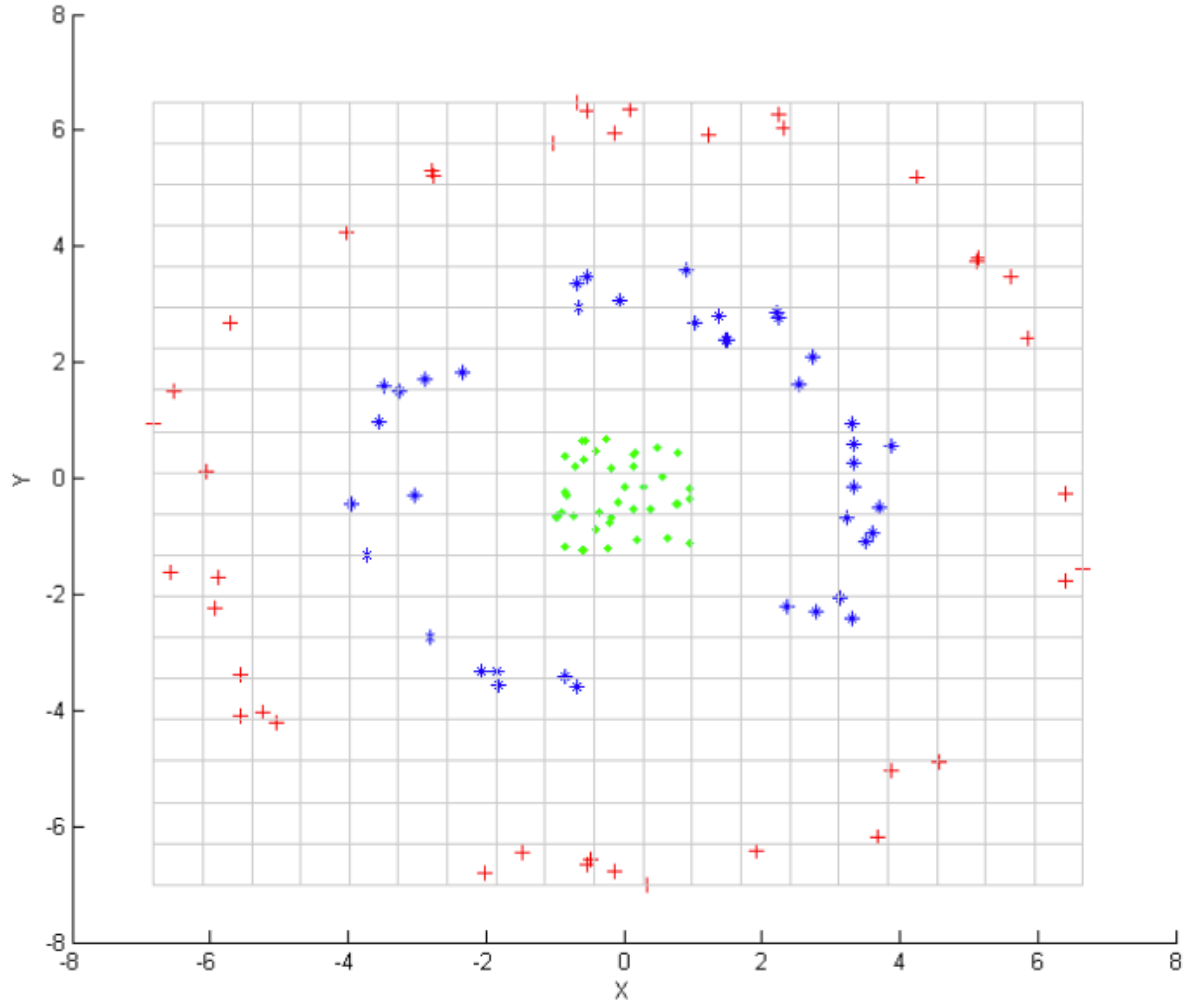


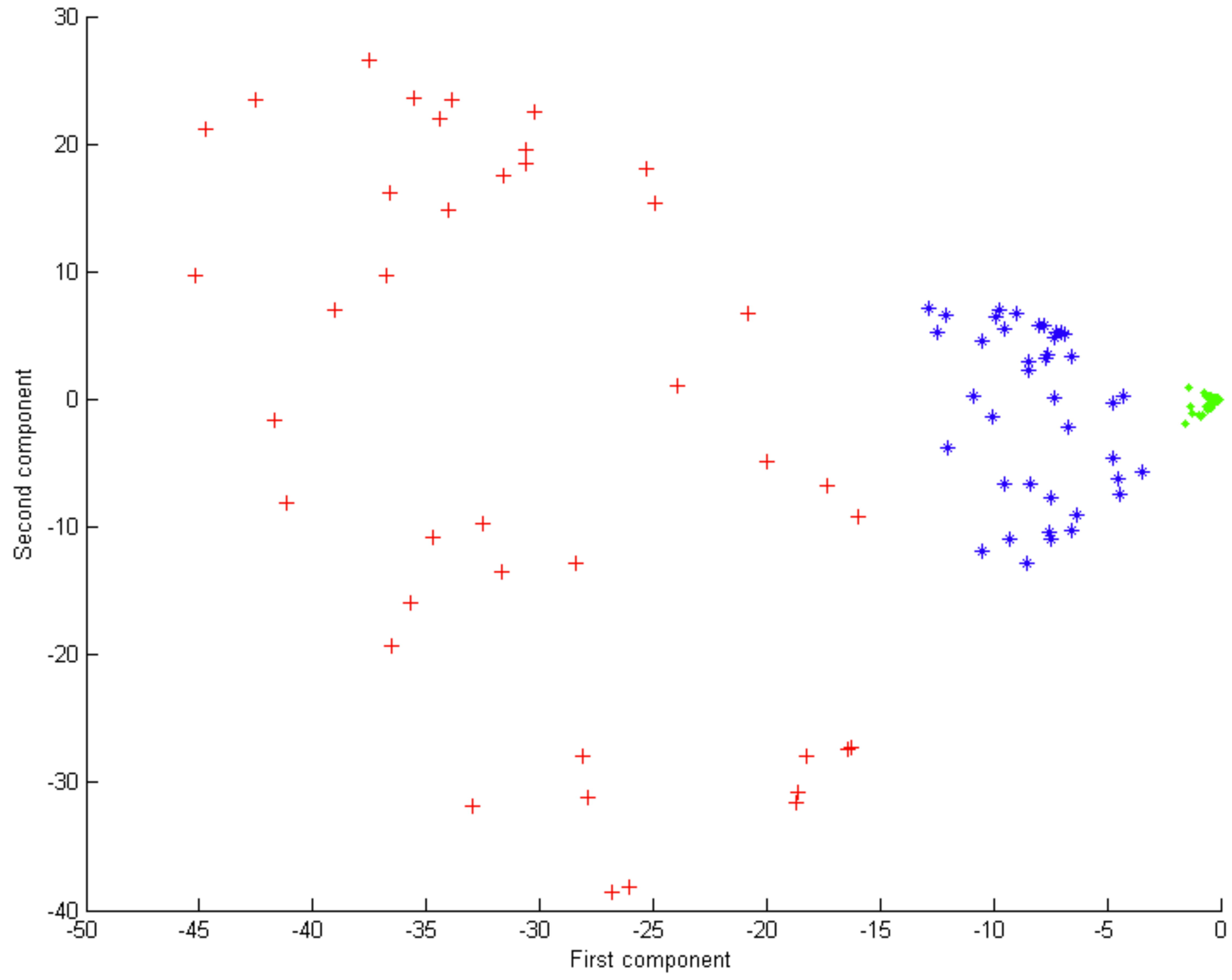
Kernel PCA

- First map the data using a kernel map $\Phi(x_i)$, then do PCA
- Instead of Φ (which is challenging to calculate), we work directly with a kernel K, which has entries $k_{ij} = k(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$
- There are well-established kernel functions $k(x, y)$ e.g. Gaussian, polynomial (you may have something about when learning about SVMs)
- X should be mean-centred but the Φ-transformed data does not necessarily have zero mean, so use a centralised version of K to calculate eigenvectors/eigenvalues
- PCA essentially does this using the canonical map i.e. $\Phi(x_i) = x_i$ i.e. $k_{ij} = x_i^T x_j$

Kernel PCA

- E.g. polynomial kernel, $k_{ij} = k(x_i, x_j) = (x_i^T x_j + 1)^2$





Multidimensional scaling

Multidimensional scaling

- Idea: map high-dimensional data into a low-dimensional space in a way that preserves pairwise distances between data points
- $o_{ij} = d(x_i, x_j) = \|x_i - x_j\|$ is the distance between vectors x_i and x_j (length p)
- d is any metric, $\|\cdot\|$ is the associated norm (e.g. Euclidean distance)
- o_{ij} is fixed
- We want to find y_i for $i = 1, \ldots, n$ in 2d or 3d space with $d_{ij} = d(y_i, y_j) = \|y_i - y_j\|$, such that $o_{ij} \approx d_{ij}$ for all i and j

Multidimensional scaling

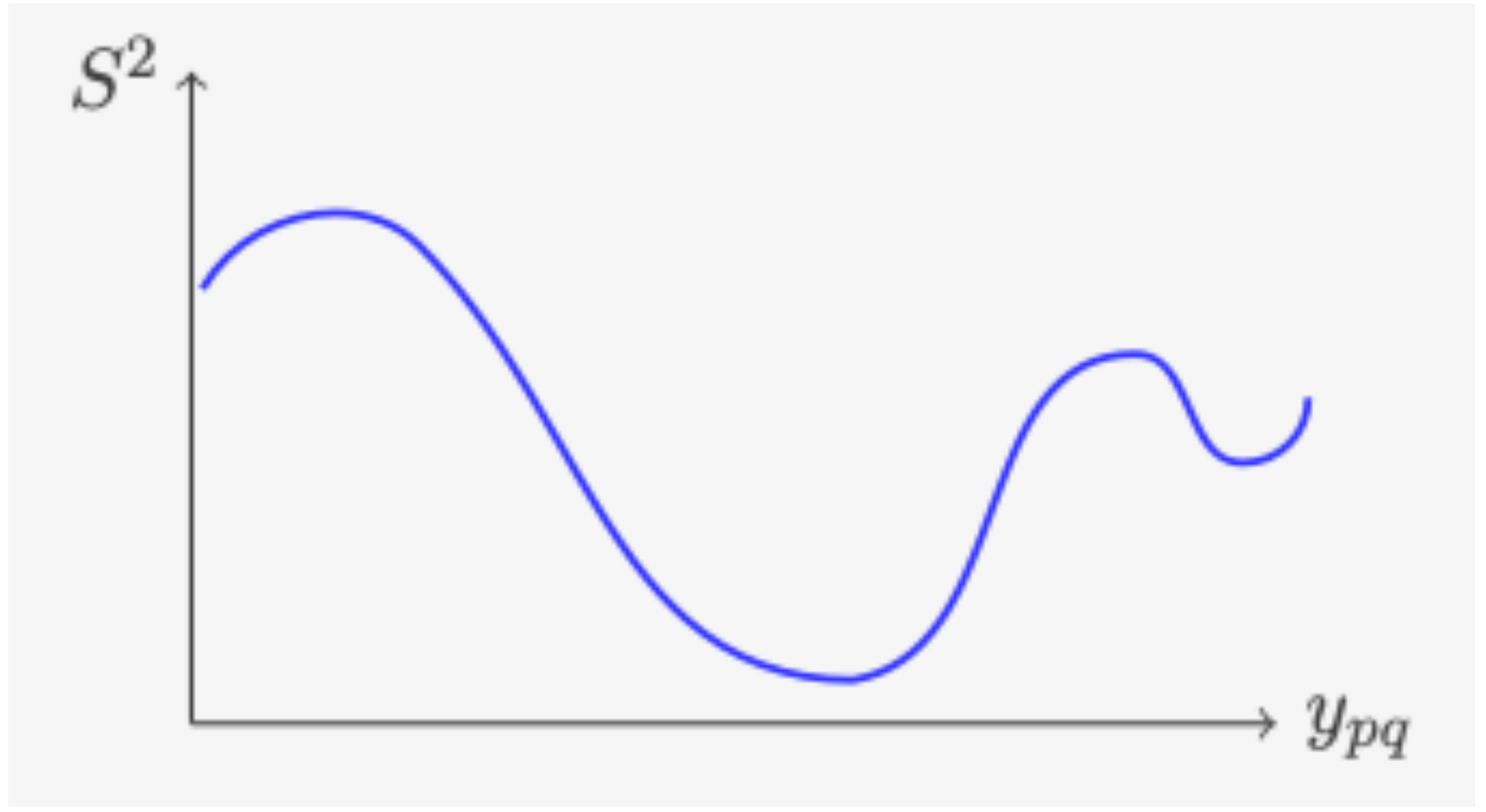
- So minimise a **stress** term, e.g. $S^2 = \dfrac{\sum_{i,j}(d_{ij} - o_{ij})^2}{\sum_{i,j} o_{ij}^2}$
- This is an optimisation problem, so solved using any optimisation technique, e.g. gradient descent
- MDS and PCA are equivalent when using the Euclidean distance plus this stress term
- Different stress terms can emphasise certain aspects of the data

Multidimensional scaling

- Start with random $Y = \begin{pmatrix} y_1^T \\ \vdots \\ y_n^T \end{pmatrix} = \begin{pmatrix} y_{11} & \cdots & y_{1d} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nd} \end{pmatrix}$, $d = 2$ or $d = 3$

- Calculate S^2, gradient at y_{pq}, and $\Delta y_{pq} = -\alpha \frac{\partial S^2}{\partial y_{pq}}$

- Iteratively update y_{pq} until a local minimum



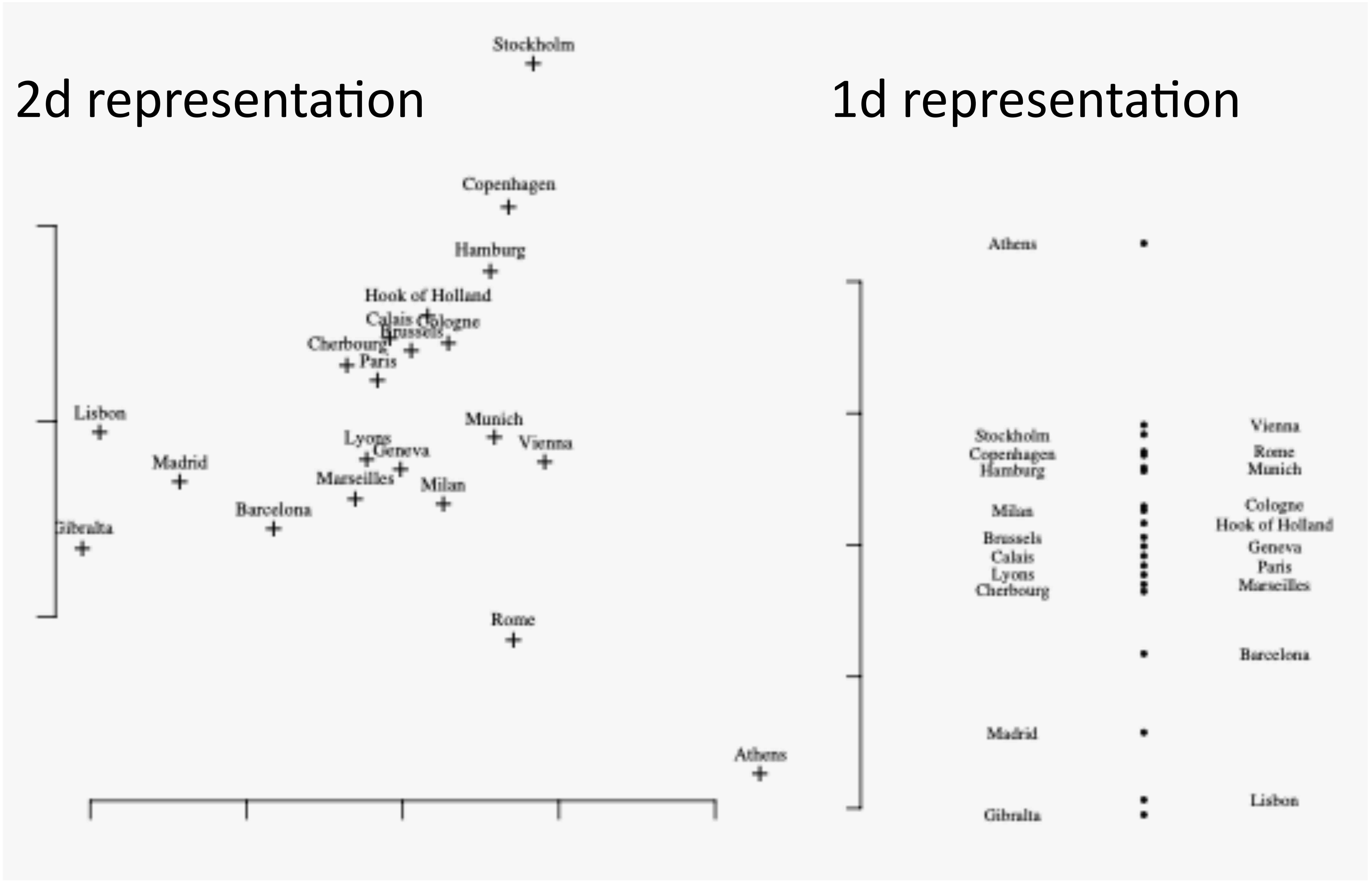
Sammon mapping

- Sammon's stress is $S^2 = \frac{1}{\sum_{i<j} o_{ij}} \sum_{i<j} \frac{(d_{ij} - o_{ij})^2}{o_{ij}}$
- Introduced in 1969, one of the most successful non-linear metric multidimensional scaling methods

MDS example

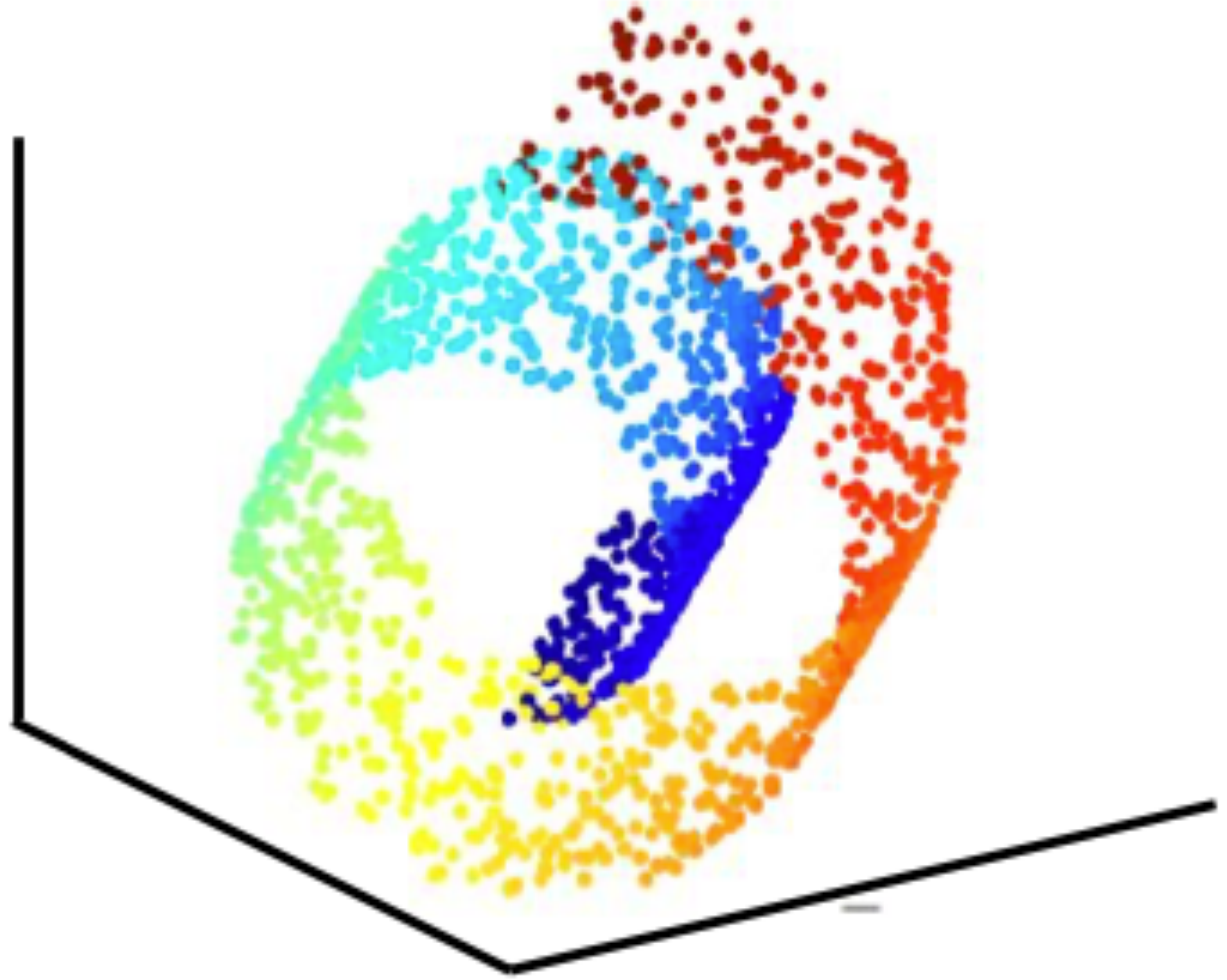
eurodist dataset:
road distance in km
between 21 cities

	Athens	Barcelona	Brussels	Calais	Cherbourg
Barcelona	3313				
Brussels	2963	1318			
Calais	3175	1326	204		
Cherbourg	3339	1294	583	460	
Cologne	2762	1498	206	409	785
Copenhagen	3276	2218	966	1136	1545



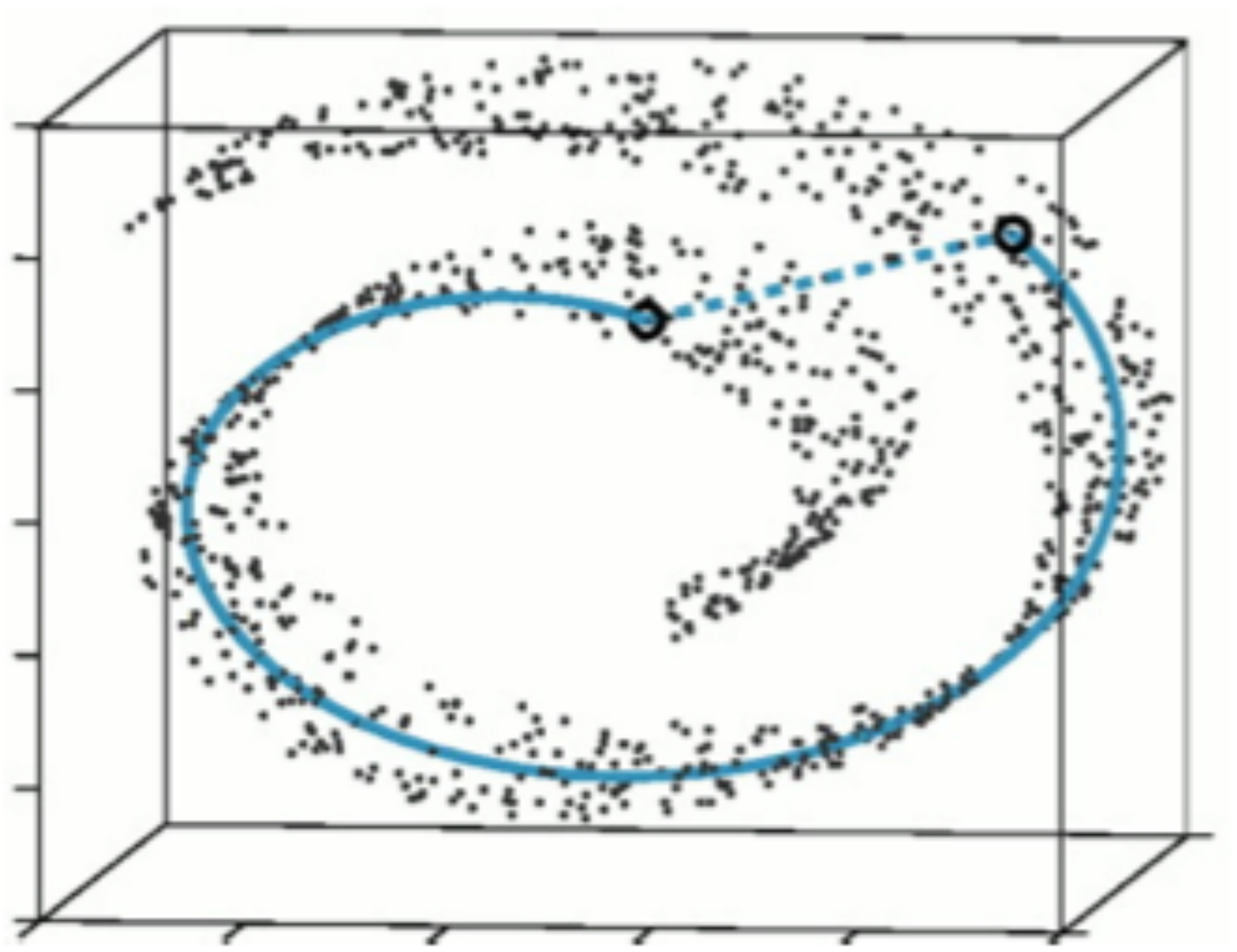
Questions?

Swiss roll



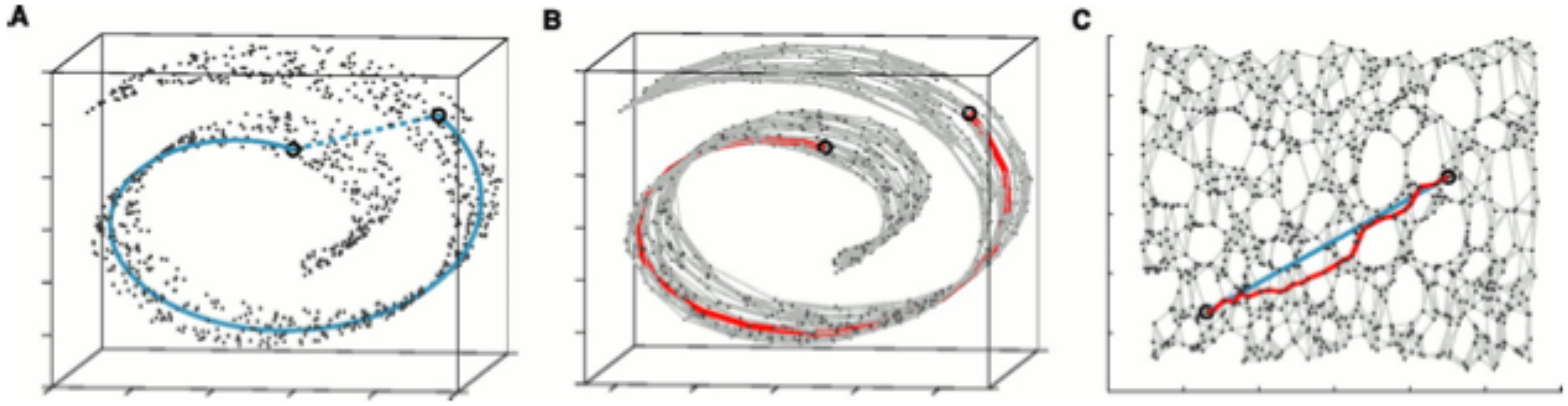
Isomap

- Use geodesic distance (i.e. distance along the manifold) rather than Euclidean distance



Isomap

- Use geodesic distance (i.e. distance along the manifold) rather than Euclidean distance
- For nearby points, the geodesic distance is similar to Euclidean distance
- For faraway points, the geodesic distance is like a sequence of hops along nearby points



Isomap

1. Determine the nearest neighbours of each point (k-nn)
2. Construct neighbourhood graph (each point connected to its neighbours, with edge length equal to Euclidean distance, e.g. if x_i and x_j are neighbours, then $d(x_i, x_j) = ||x_i - x_j||$
3. Compute the shortest path $d_G(x_i, x_j)$ from x_i to x_j for all i and j along edges connecting (e.g. Dijkstra's algorithm), to give matrix $D = d_G(x_i, x_j)$
4. Compute lower-dimensional embedding on the shortest path distance matrix using multidimensional scaling

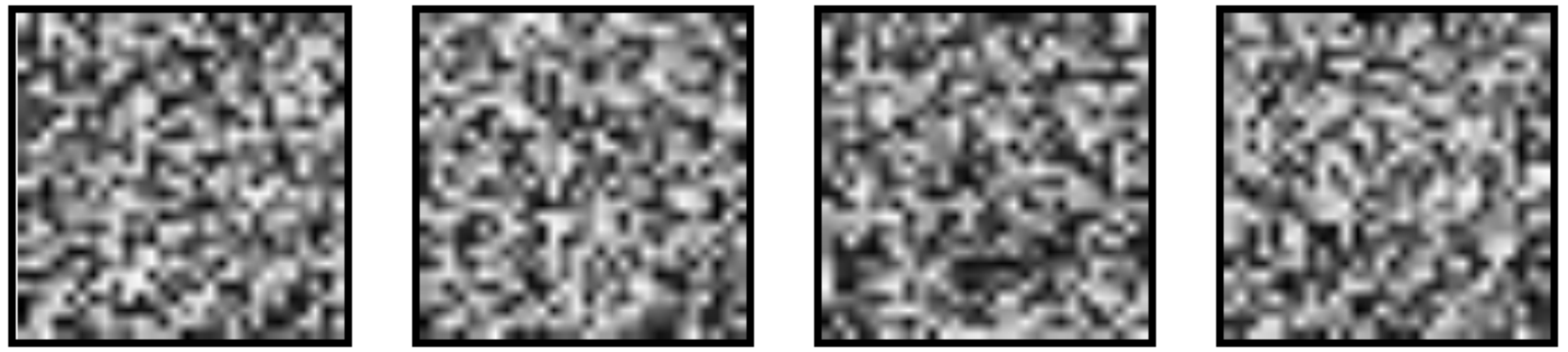
MNIST

- 28x28 pixel images of handwritten digits
- Each pixel has value between 0 and 1 (0=white, 1=black)
- Flatten each image into a vector of length 784
- ‘Hello, world’ task of machine learning
- http://yann.lecun.com/exdb/mnist/

MNIST

- 28x28 pixel images of handwritten digits
- Each pixel has value between 0 and 1 (0=white, 1=black)
- Flatten each image into a vector of length 784
- ‘Hello, world’ task of machine learning
- http://yann.lecun.com/exdb/mnist/
- MNIST digits occupy a lower-dimensional subspace within the full 784-dimensional space
- https://colah.github.io/posts/2014-10-Visualizing-MNIST/

Random 28x28 images look like this



t-SNE

t-SNE

- t-distributed Stochastic Neighbour Embedding
- High-dimensional neighbourhoods encoded as a distribution (e.g. probability that two data points x_i and x_j are related)
- Random walk between all observations, with a higher probability if x_i and x_j are nearby: $p_{j|i} = \dfrac{\exp(-\|x_i - x_j\|^2/(2\sigma_i^2))}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/(2\sigma_i^2))}$, $p_{ij} = \dfrac{p_{i|j} + p_{j|i}}{2n}, p_{ii} = 0$
- σ_i^2 is a 'perplexity' parameter, which is adjusted by density of points near x_i, i.e. the more points near the observation x_i, the smaller σ_i^2 should be

t-SNE: perplexity

- Perplexity is defined as $perp(p_{j|i}) = 2^{H(P_{j|i})}, H(P) = \sum_i p_i \log(p_i)$ is entropy
- Low perplexity = small σ^2 = more probability towards nearest neighbour
- High perplexity = large σ^2 = every other point approx. weighted uniformly

t-SNE: embedding density

- Given the data $X = (x_1^T, \ldots, x_n^T)$, we have the distribution p_{ij}
- We want an embedding $Y = (y_1^T, \ldots, y_n^T)$ in lower-dimensional space that has a similar neighbourhood distribution
- We modelled the density around x_i as a Gaussian, instead model the density around y_i as a t-distribution $q_{ij} = \dfrac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}}$
- Why? In high dimensions, everything is far apart and there are lots of neighbours, but in 1d/2d there's less room to accommodate all neighbours (**crowding problem**)

t-SNE: Kullback-Leibler divergence

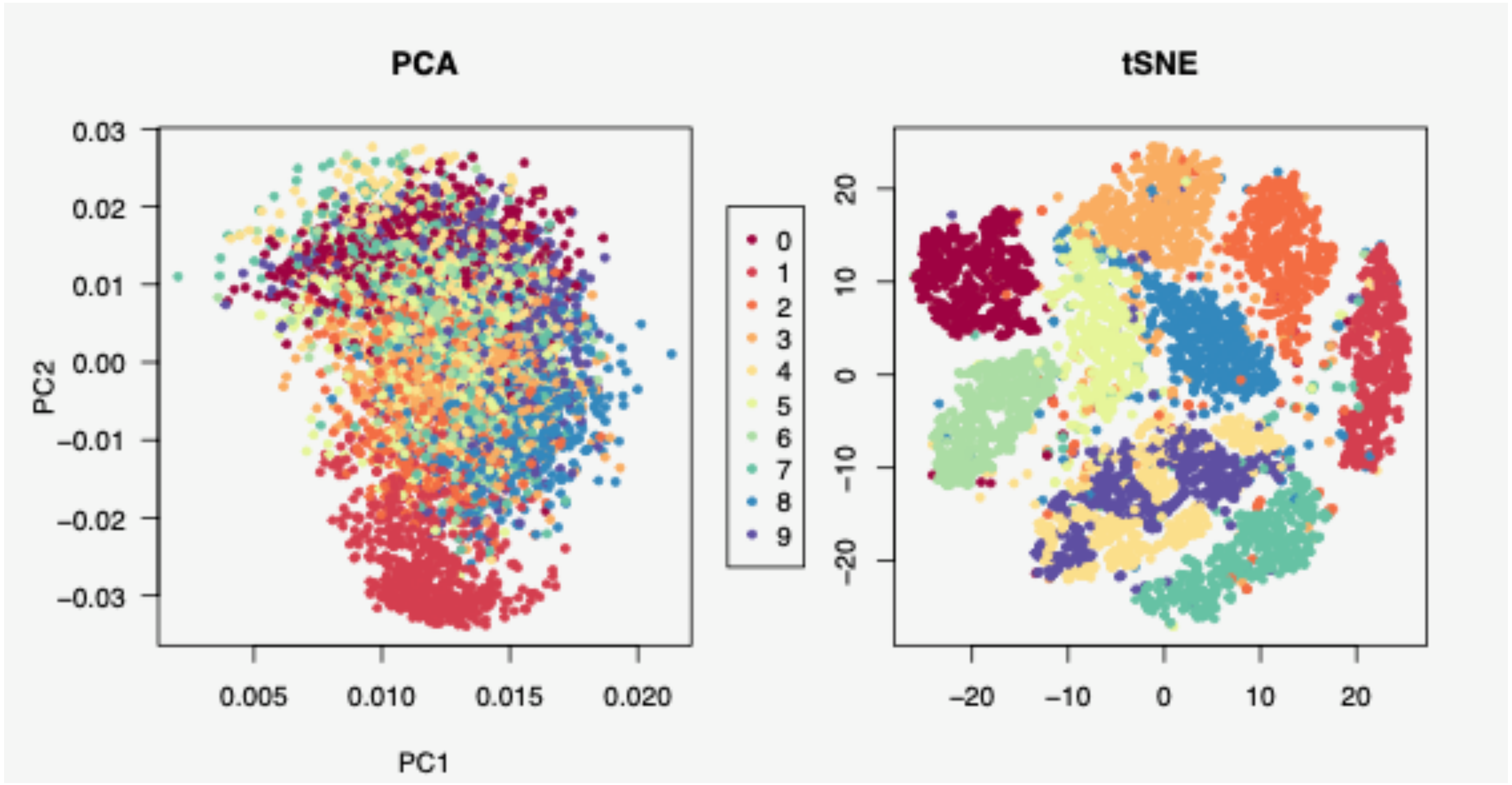
- Adjust the embeddings y_i so that the distribution Q is similar to the (fixed) distribution P, i.e. optimise over $Y = (y_1^T, \ldots, y_n^T)$ so that q_{ij} are close to p_{ij}
- We can measure the 'distance' between two distributions P and Q using the KL-divergence: $KL(Q\|P) = \sum_{ij} q_{ij} \log \frac{q_{ij}}{p_{ij}}$ (this is not symmetric)
- The KL-divergence is a kind of 'penalty' for using the wrong distribution
- Gradient descent: $\frac{\partial L}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$

PCA vs t-SNE

- PCA tries to preserve **global structure**
- t-SNE tries to preserve **local structure**: low dimensional local neighbourhoods should be similar to high-dimensional local neighbourhoods
- We can embed new points into PCA using the original principal components (same for kernel PCA, using the map Φ)
- We can't do this for t-SNE, so this is not so useful for analysis beyond data visualisation

- Similar to t-SNE (generally slightly better) is **UMAP**

PCA vs t-SNE: MNIST



UMAP and t-SNE

- UMAP is uniform manifold approximation and projection
- On the surface, UMAP is similar to t-SNE, underneath it's slightly different
- This involves various complicated concepts (locally connected Riemannian manifolds, topological structure, fuzzy simplicial sets)
- t-SNE and UMAP are stochastic, i.e. you probably won't get exactly the same result if you repeat the embeddings

UMAP vs t-SNE

- See https://pair-code.github.io/understanding-umap and https://jlmelville.github.io/uwot/umap-for-tsne.html for more details

- UMAP initialises the embeddings with **Laplacian eigenmaps**, which may result in capturing more global structure (as well as local structure, like t-SNE)

Overview

- PCA: linear and deterministic
- Kernel PCA: mapping the data before PCA (calculate kernel instead of mapping)
- Multidimensional scaling: preserve pair-wise distances in a low-dim space
- Sammon mapping: a particular form of multidimensional scaling
- Isomap: MDS on matrix of geodesic distances (rather than Euclidean)
- t-SNE: model neighbourhoods as distributions, then preserve random walk probabilities
- UMAP: preserves local neighbourhood structure, similar(-ish) to t-SNE

Overview: can you map out-of-sample points?

- PCA: ✅
- Kernel PCA: ✅
- Multidimensional scaling (and Sammon mapping): ❌
- Isomap: ❌
- t-SNE: ❌
- UMAP: ❌

Overview: does it work for the Swiss roll?

- PCA: ❌
- Kernel PCA: ❌
- Multidimensional scaling (and Sammon): not quite (can flatten but not unroll)
- Isomap: ✅
- t-SNE: ✅
- UMAP: ✅

Code example: mnist.ipynb

- On GitHub/Moodle/GitLab
- Example of each method, for MNIST digits

Questions?

- Feel free to email me at te269@cam.ac.uk

- Example class questions will be from/similar to:

  Introduction to Statistical Learning with Python, Section 8.4 (Exercises)

- You might find it useful to work through Section 8.3 (Lab) beforehand
- Pdf of the book at https://www.statlearning.com/

Next time

- Clustering
  - k-means
  - Fuzzy c-means