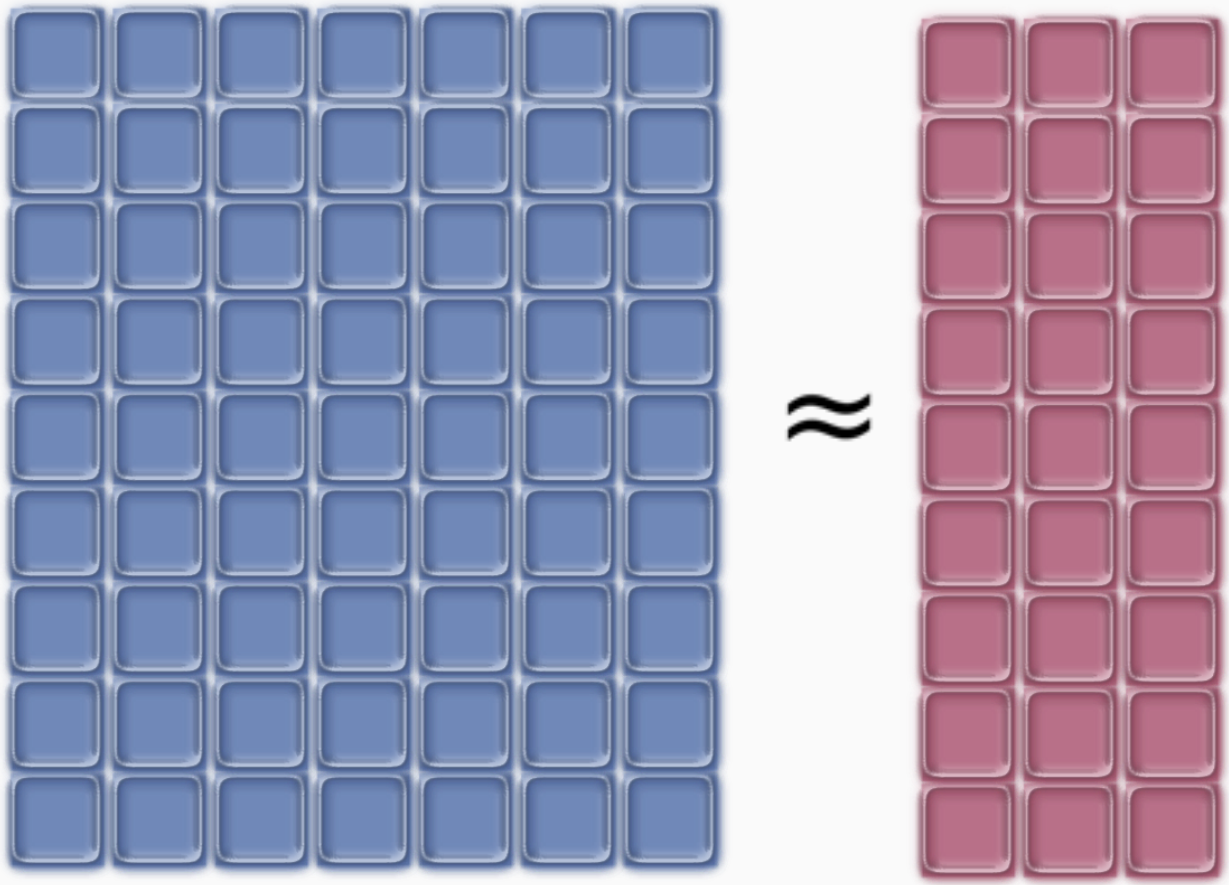
Dimensionality Reduction & Clustering

Paul Brodersen

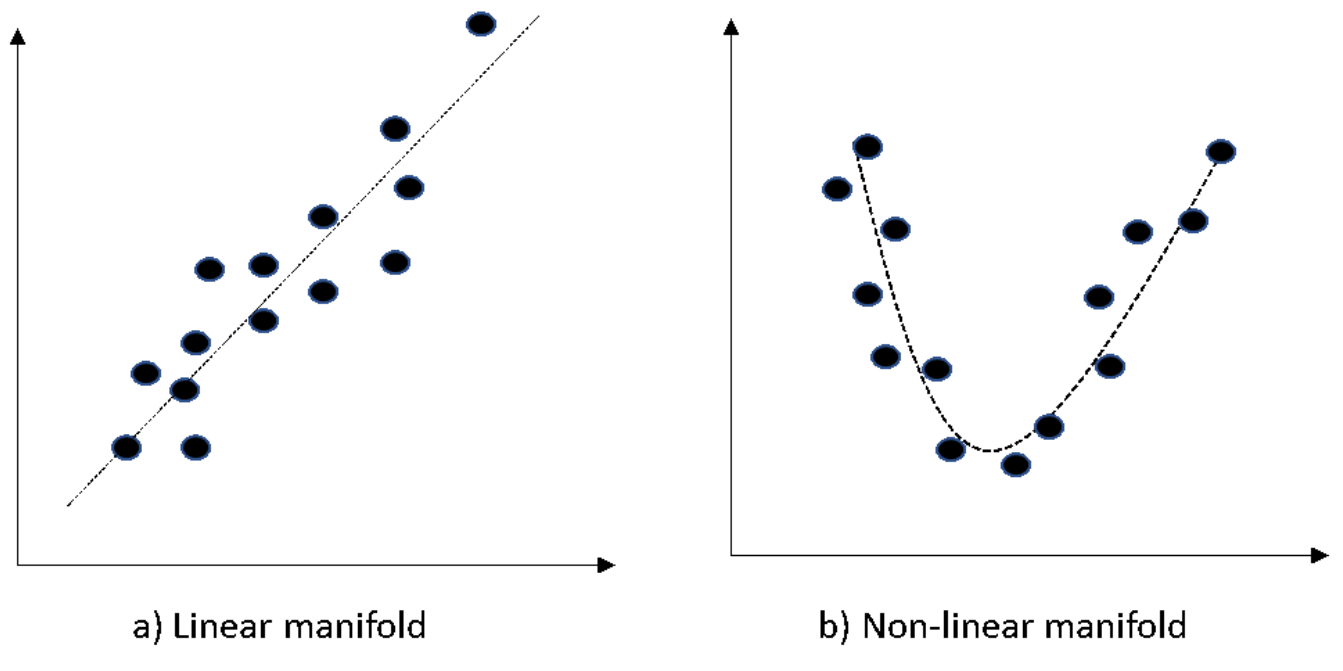
May 20, 2020

Dimensionality reduction

What is dimensionality reduction?

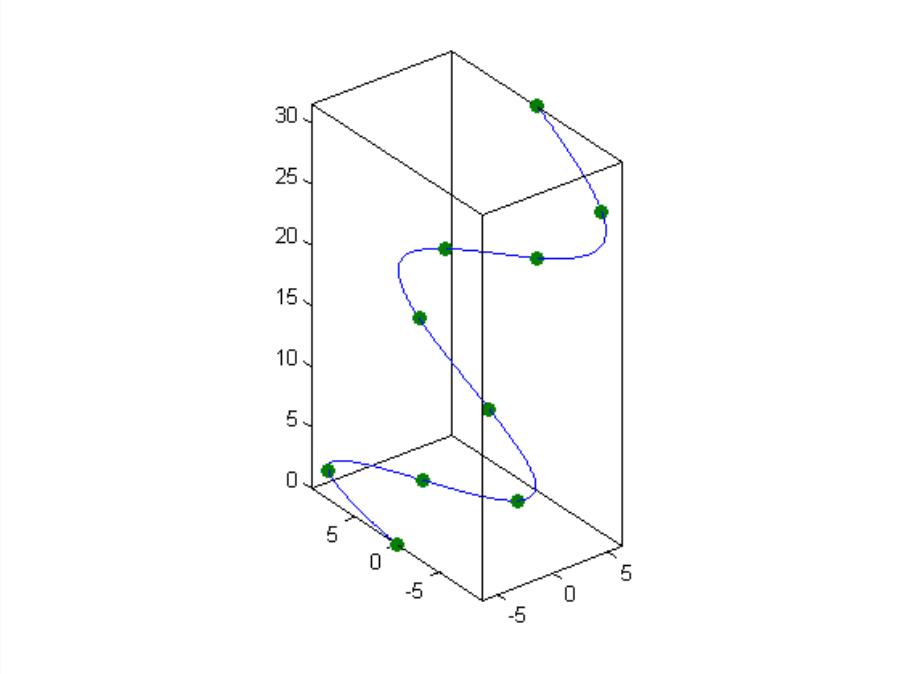


Main assumption: $d < D$



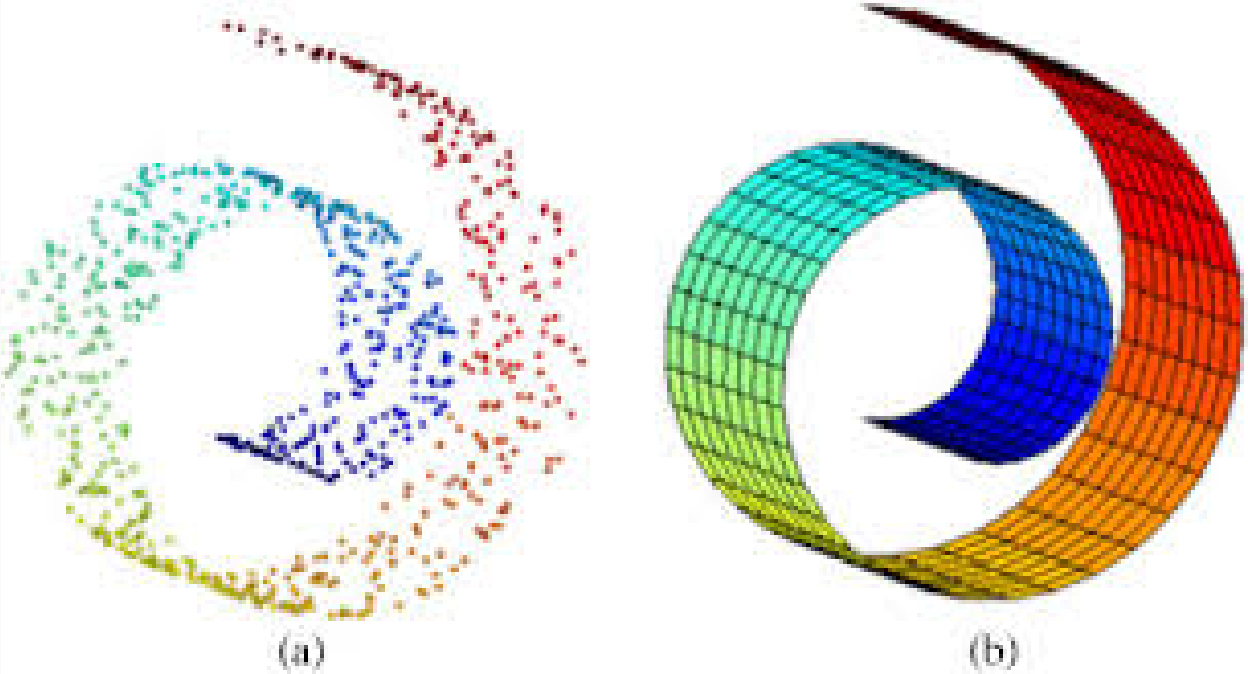
Data has a given dimensionality (D) and a 'native' dimensionality (d). The native dimensionality ***may*** be much smaller than the given dimensionality.

Main assumption: $d < D$



Data has a given dimensionality (D) and a 'native' dimensionality (d). The native dimensionality ***may*** be much smaller than the given dimensionality.

Main assumption: $d < D$

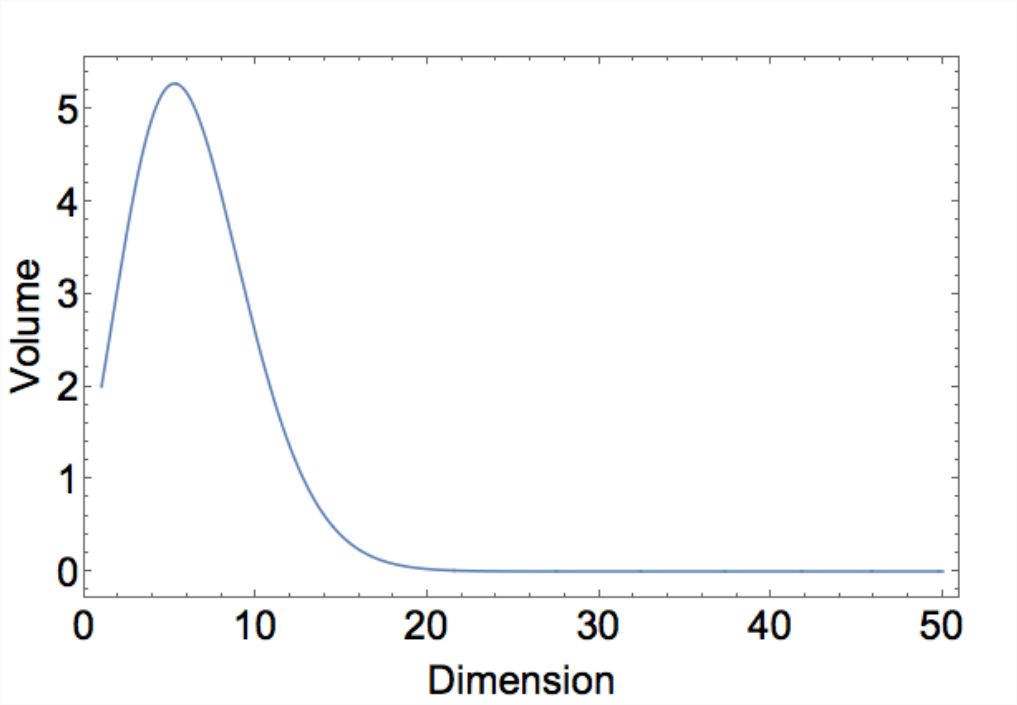


Data has a given dimensionality (D) and a 'native' dimensionality (d). The native dimensionality ***may*** be much smaller than the given dimensionality.

Why reduce the dimensionality of your data?

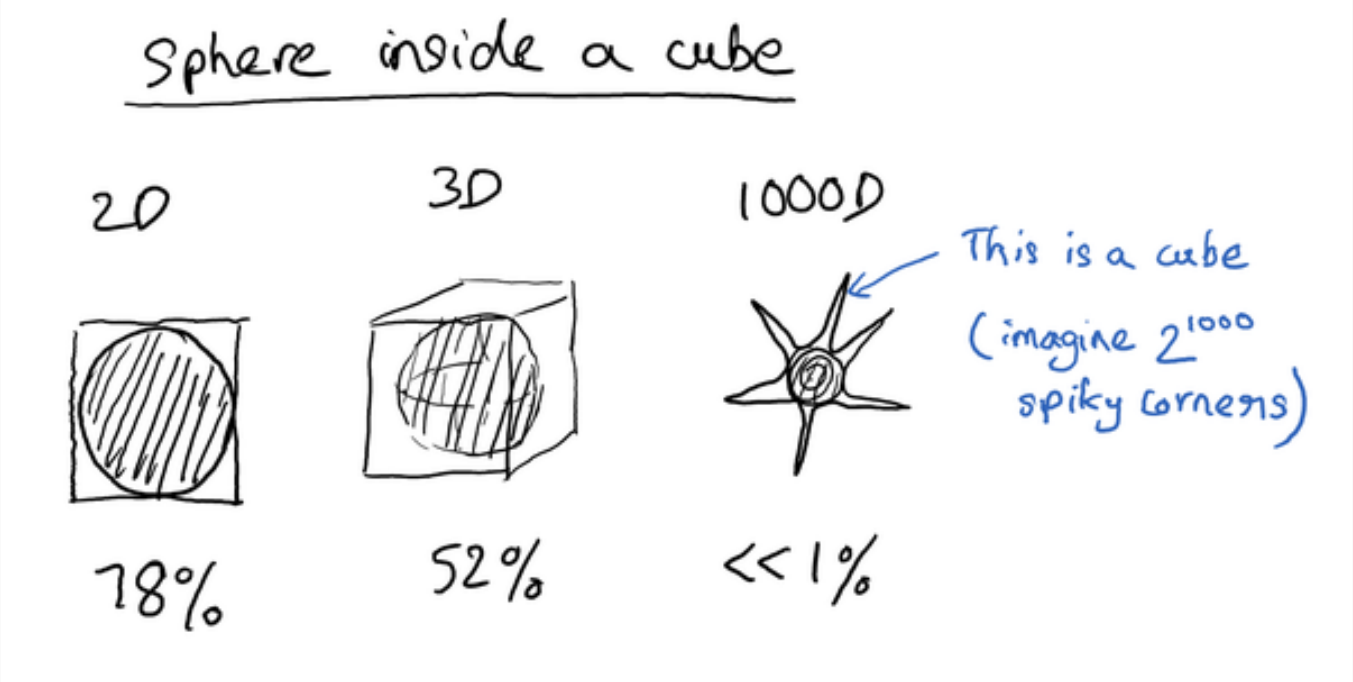
1. Visualisation
2. Interpretability
3. Compression
4. Avoid the surprising behaviour of distance metrics in high dimensional space

The surprising behaviour of distance metrics in high dimensional space (Aggarwal *et al.* (2001))



https://marckhoury.github.io/counterintuitive-properties-of-high-dimensional-space/

The surprising behaviour of distance metrics in high dimensional space (Aggarwal *et al.* (2001))



https://marckhoury.github.io/counterintuitive-properties-of-high-dimensional-space/

Notation

- X : the samples
- y : the labels
- N : the number of samples
- D : the dimensionality of the samples
- d : the desired dimensionality

Notation

- X : the samples
- y : the labels
- N : the number of samples
- D : the dimensionality of the samples
- d : the desired dimensionality
- * : I am lying but in a good way

There are only two approaches to dimensionality reduction*:

There are only two approaches to dimensionality reduction*:

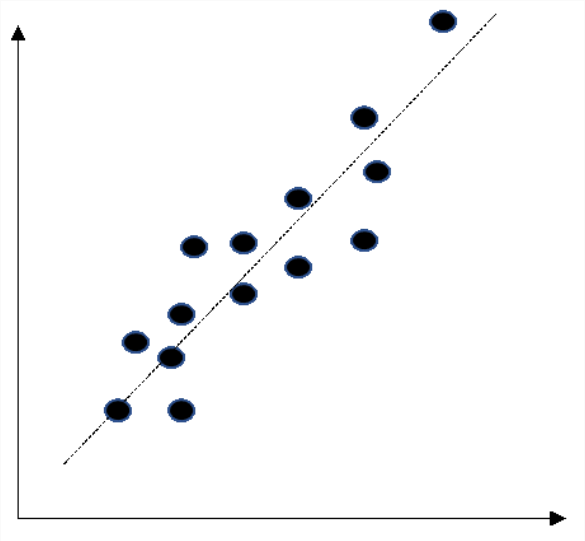
- composition of exemplars / archetypes

There are only two approaches to dimensionality reduction*:

- composition of exemplars / archetypes
- embedding based on nearest neighbour graph

Composition of archetypes

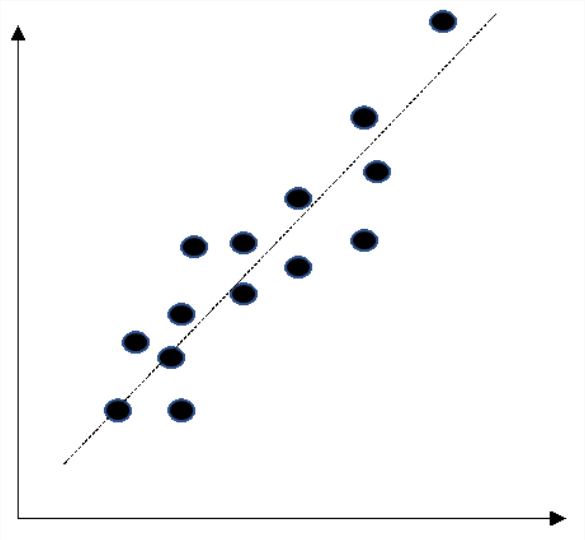
Fitting lines



$$f(x) = ax \tag{1}$$

$$\text{RMSE} : \sqrt{\frac{1}{N}\sum_{i}^{N}(y_i - f(x_i))^2} \tag{2}$$

Fitting lines: an alternative view

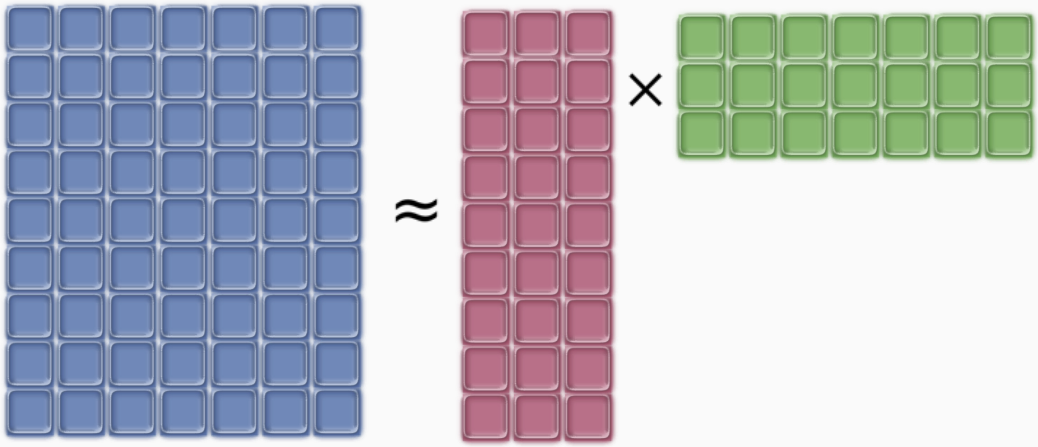


$$X \approx UV^* \tag{3}$$

X is an $(N, 2)$ matrix, i.e. the (x,y) pairs
U is an $(N, 1)$ matrix, i.e. the vector magnitudes
V is a $(1, 2)$ matrix, i.e. the vector (1 a)

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{4}$$

Fitting (hyper-) planes: matrix factorisation

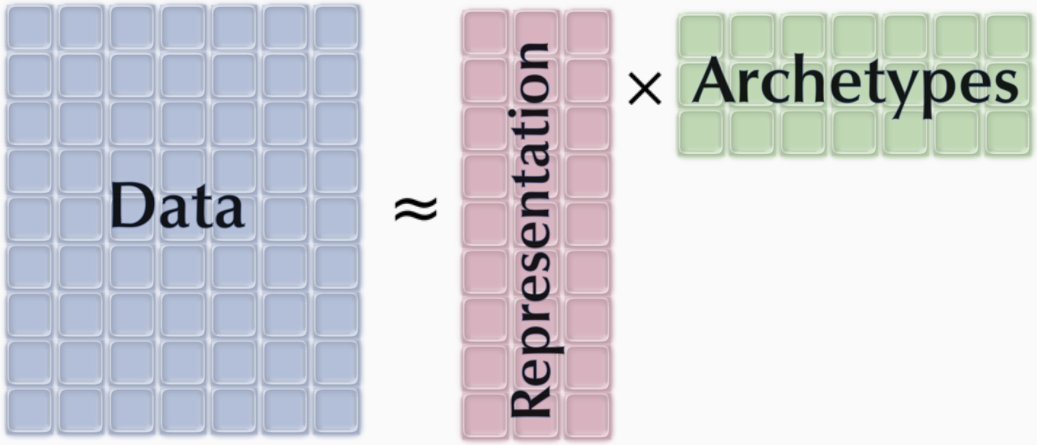


$$X \approx UV^* \tag{5}$$

X is an (N, D) matrix
U is an (N, d) matrix
V is an (d, D) matrix

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{6}$$

Fitting (hyper-) planes: matrix factorisation

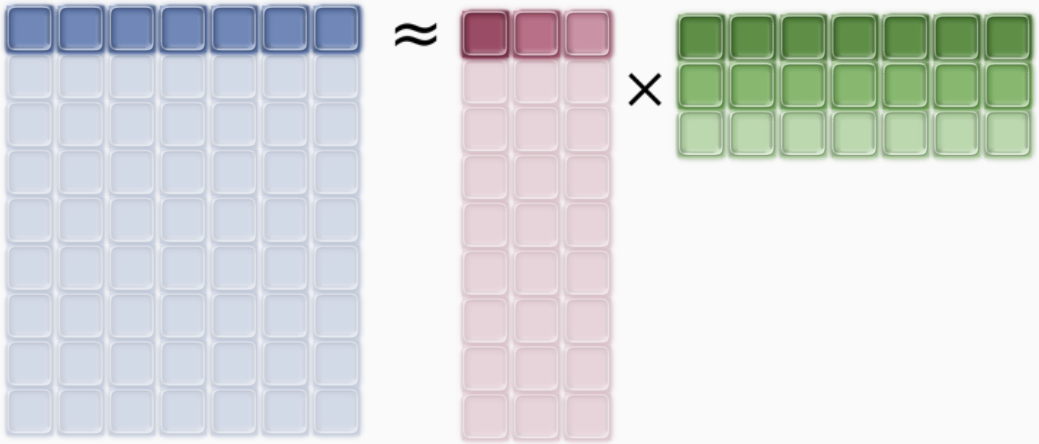


$$X \approx UV^* \tag{5}$$

X is an (N, D) matrix
U is an (N, d) matrix
V is an (d, D) matrix

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{6}$$

Fitting (hyper-) planes: matrix factorisation

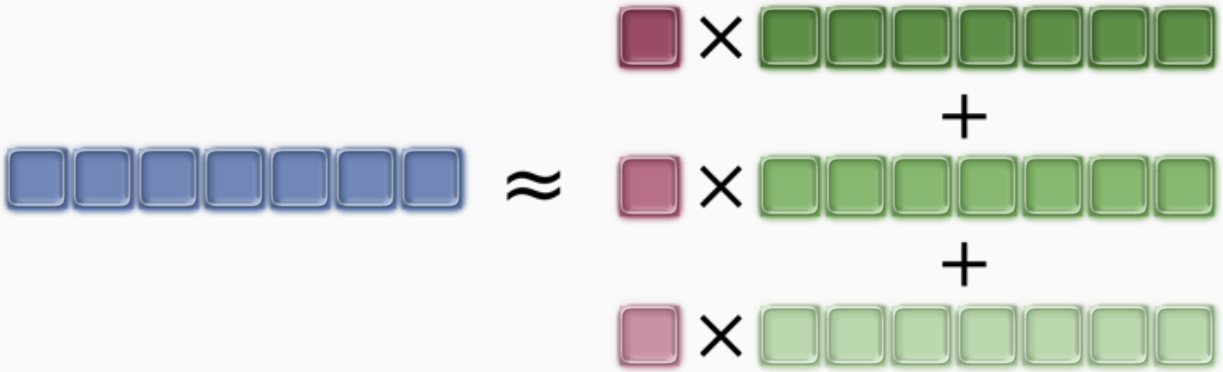


$$X \approx UV^* \tag{5}$$

X is an (N, D) matrix
U is an (N, d) matrix
V is an (d, D) matrix

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{6}$$

Fitting (hyper-) planes: matrix factorisation

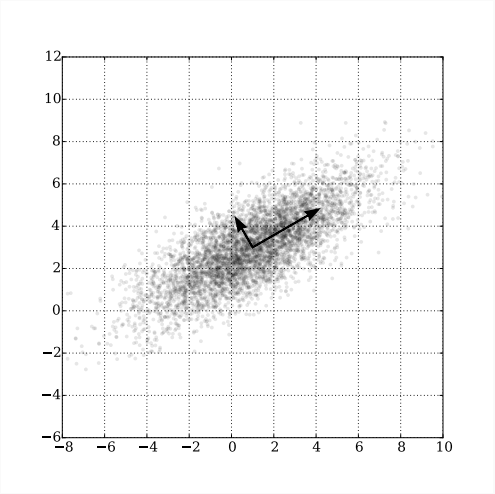


$$X \approx UV^* \tag{5}$$

X is an (N, D) matrix
U is an (N, d) matrix
V is an (d, D) matrix

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{6}$$

Principal component analysis (PCA)



Matrix factorisation, where

1. vectors are orthogonal to each other
2. a vector points into the direction of largest (co-) variance
3. vectors are scaled* such that the representations U_{ij} are normally distributed with unit variance

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{7}$$

Sparse PCA

PCA but for any given representation U_i, most loadings U_{ij} are (near) zero.

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (X_{ij} - (UV)_{ij})^2 \tag{8}$$

With

$$\sum_{j} U_{ij} \leq k \tag{9}$$

$$\sum_{j} U_{ij}^2 = 1 \tag{10}$$

Non-negative matrix factorisation

Non-negative matrix factorisation: PCA but we only allow additive combination of archetypes

$$\text{Cost} : \sum_{i}^{N} \sum_{j}^{D} (UV)_{ij} - X_{ij} \log((UV)_{ij}) \tag{11}$$

- $U_{ij} > 0$,
- $V_{ij} > 0$

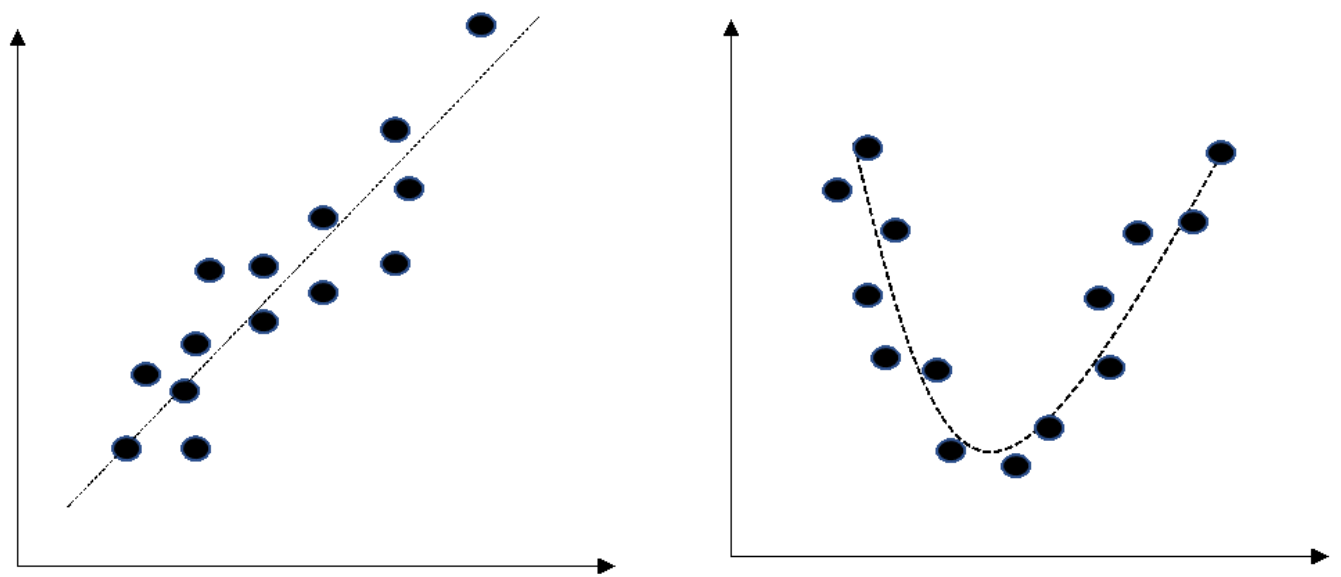
Further listening / reading

Leland McInnes : A bluffer's guide to dimensionality reduction, PyData 2018

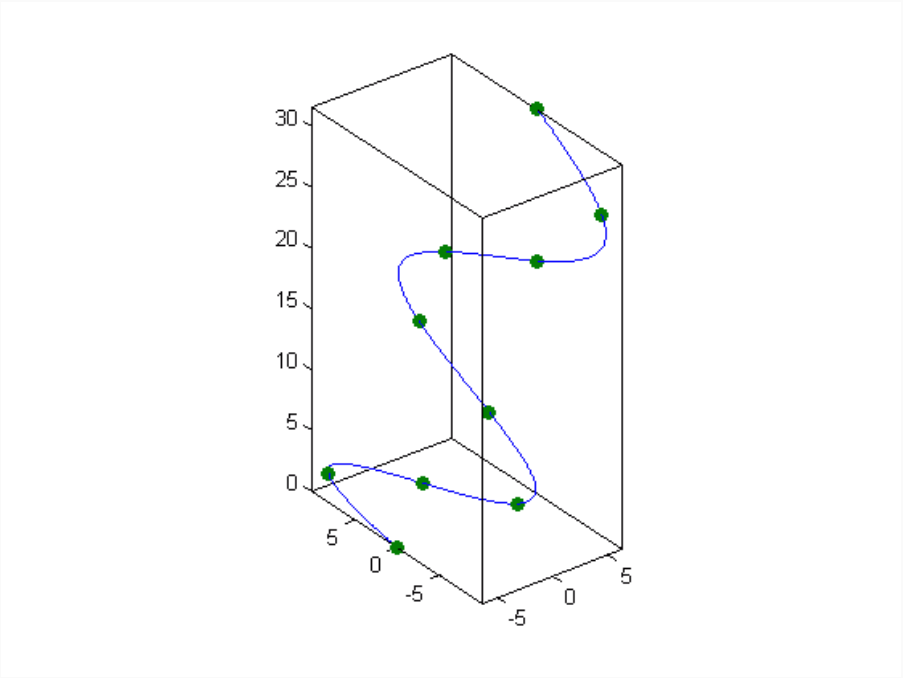
Udell *et al.* (2016) Generalised low rank models

Nearest neighbour graphs

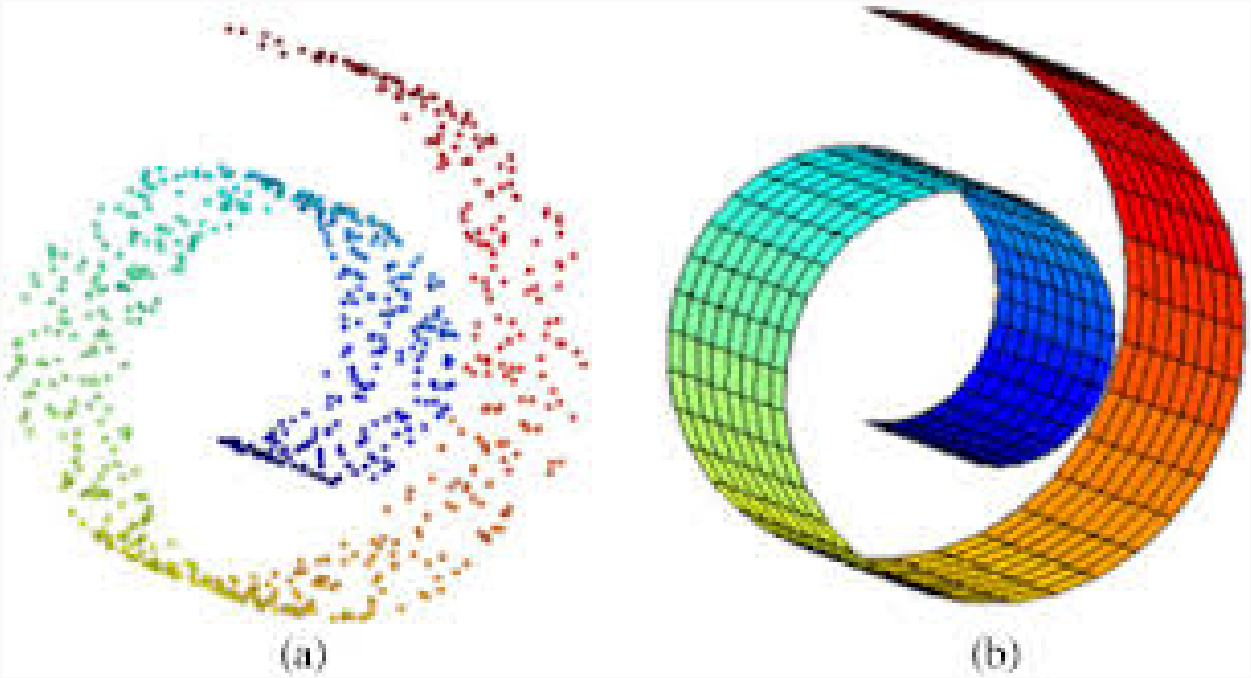
Parameterising non-linear models is hard



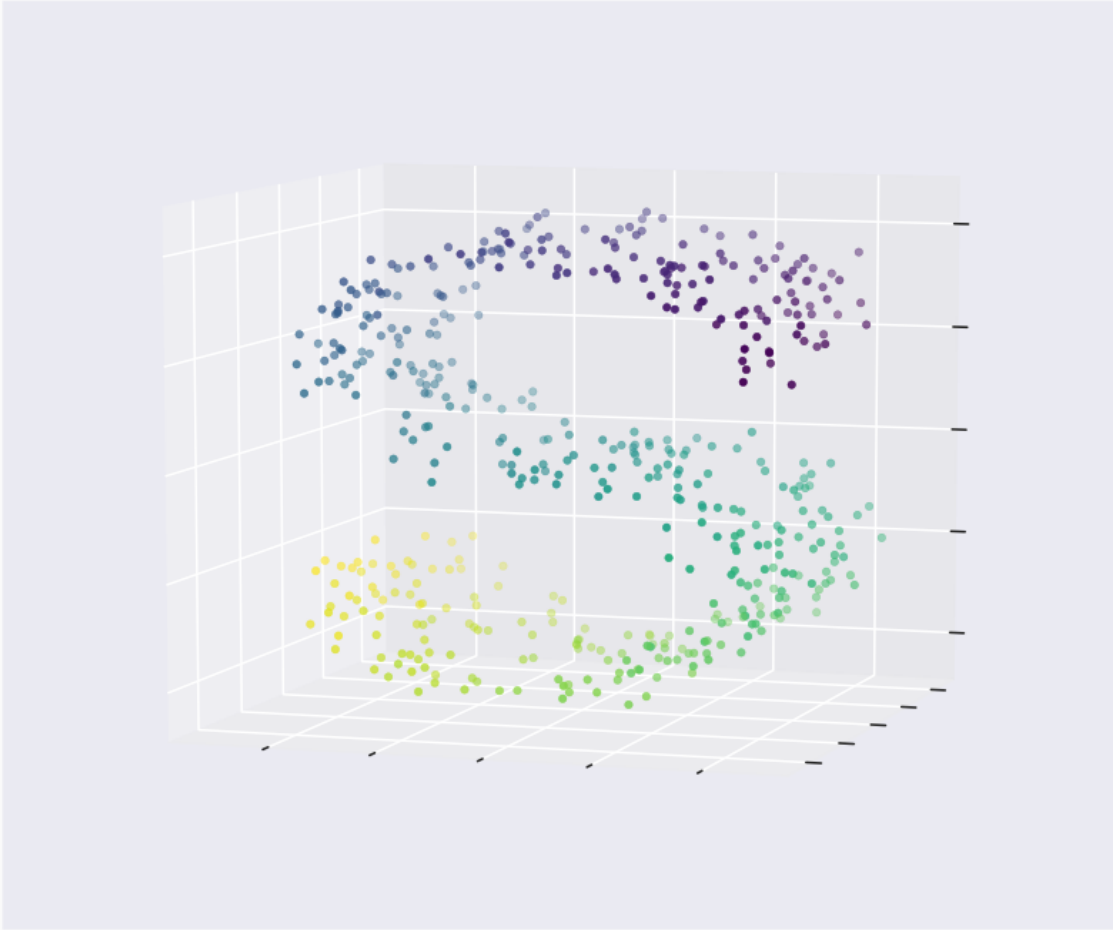
Parameterising non-linear models is hard



Parameterising non-linear models is hard

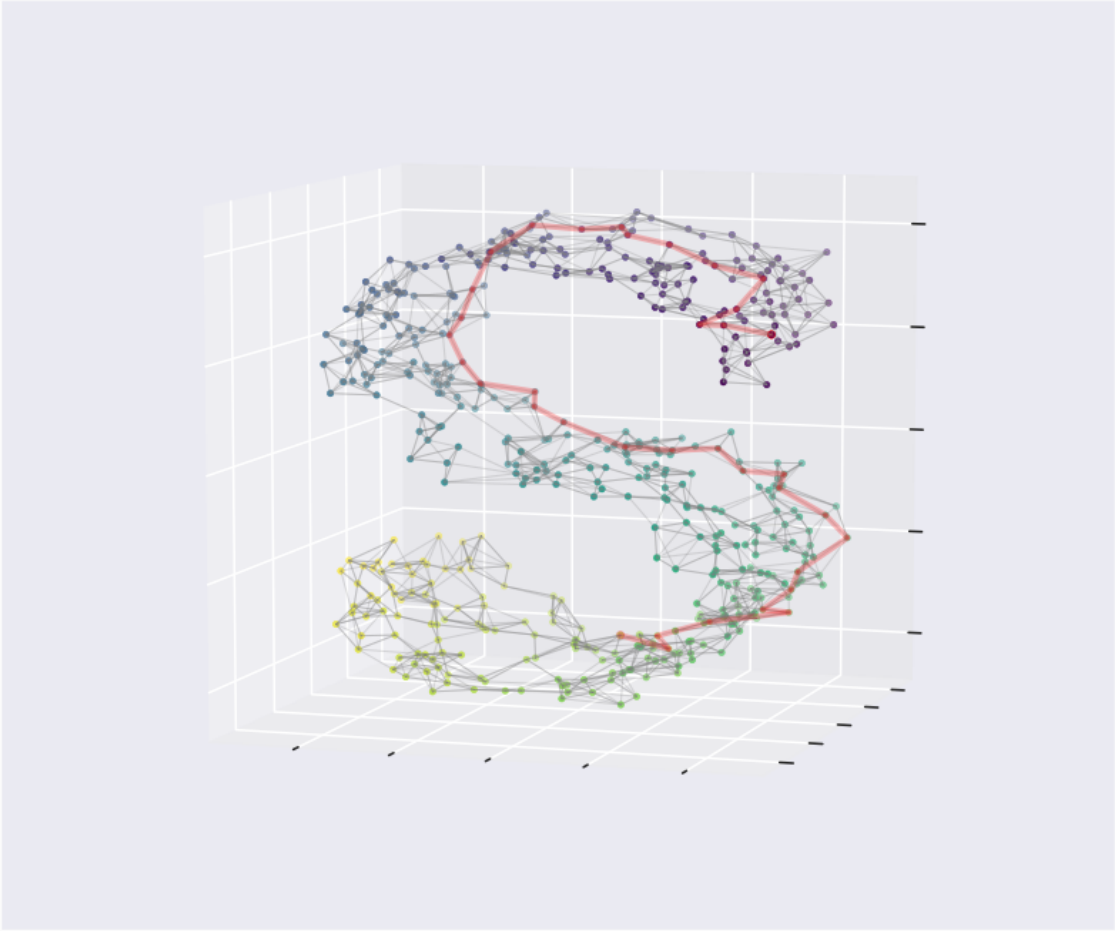


Distance relationships via nearest neighbour graphs



Distance relationships via nearest neighbour graphs

Distance relationships via nearest neighbour graphs



Isomap

1. Connect each sample/node to its k nearest neighbours. Edge weights are proportional to the Euclidean distance between the samples.

Isomap

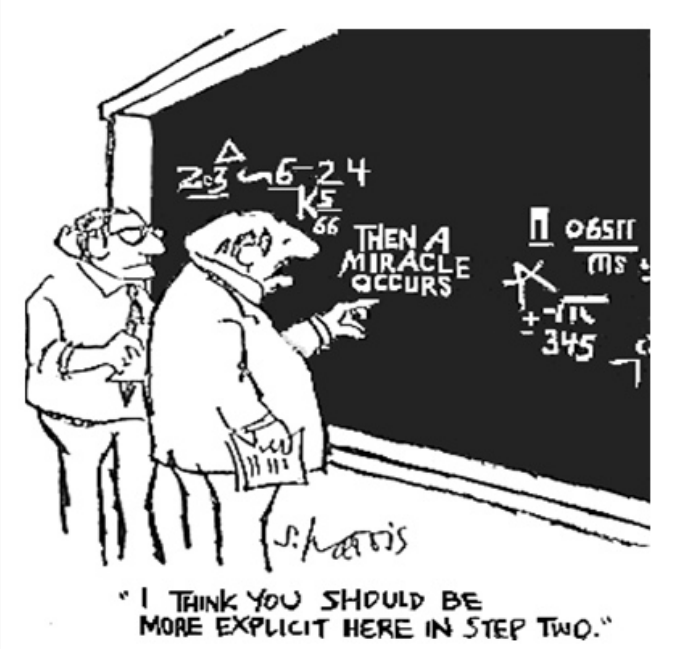
1. Connect each sample/node to its k nearest neighbours. Edge weights are proportional to the Euclidean distance between the samples.
2. Compute all shortest paths between all node pairs. This results in a (N, N) distance matrix.

Isomap

1. Connect each sample/node to its k nearest neighbours. Edge weights are proportional to the Euclidean distance between the samples.
2. Compute all shortest paths between all node pairs. This results in a (N, N) distance matrix.
3. Factor the matrix.

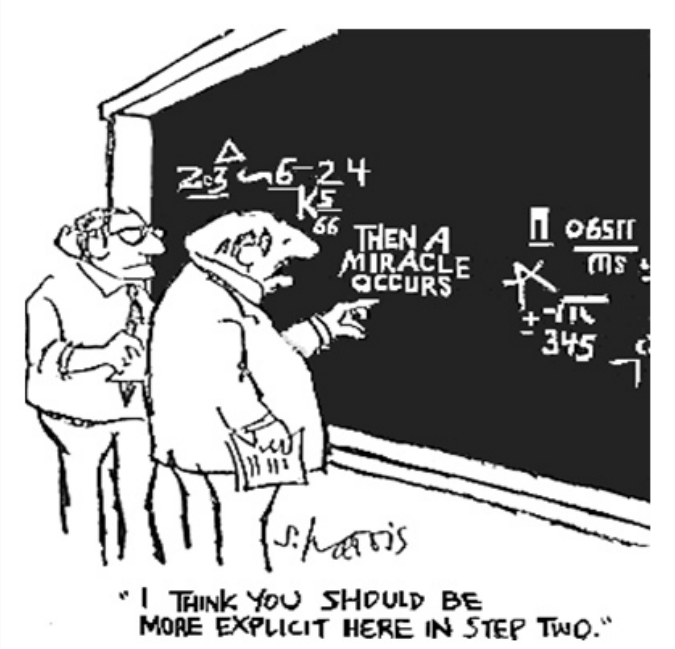
t-SNE & UMAP

1. Connect each sample/node to its *k* nearest neighbours weighted by a kernel adapted to the *k* neighbours.



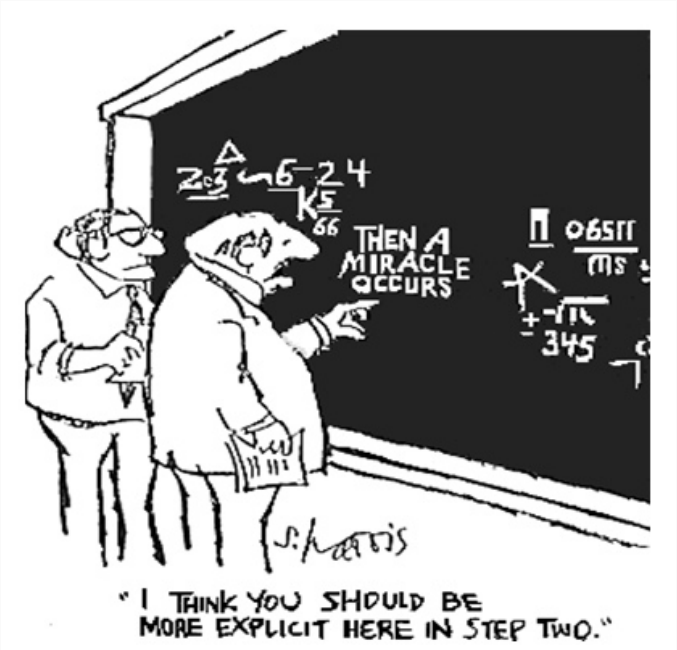
t-SNE & UMAP

1. Connect each sample/node to its *k* nearest neighbours weighted by a kernel adapted to the *k* neighbours.
2. Magic.



t-SNE & UMAP

1. Connect each sample/node to its *k* nearest neighbours weighted by a kernel adapted to the *k* neighbours.
2. Magic.
3. Use a force directed layout to place the nodes in a low-dimensional space.



Caveat: distances are only locally defined (if at all)

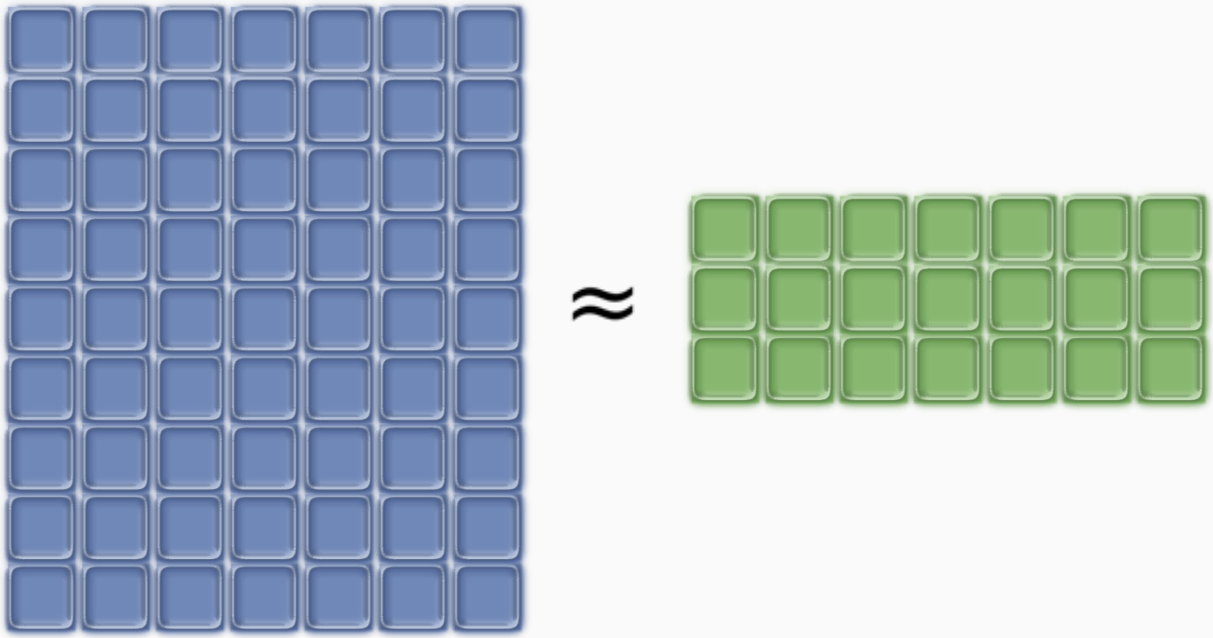
1. Absolute distances mean nothing.
2. Samples that are close in low-dimensional space are close in high-dimensional space.
3. Samples that are far apart in low-dimensional space may also be close in high-dimensional space.

Further listening

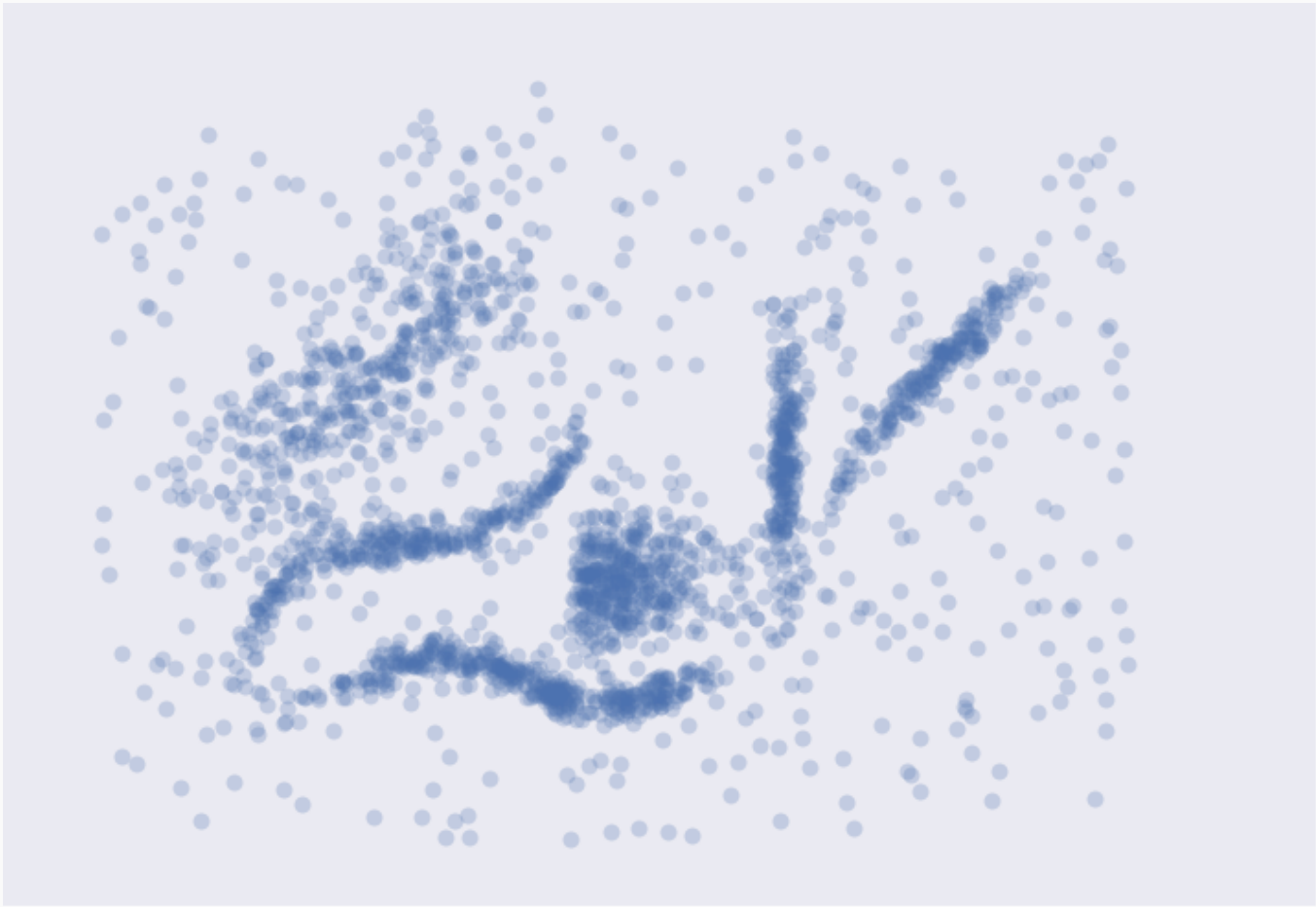
Leland McInnes : UMAP, Scipy 2018

Clustering

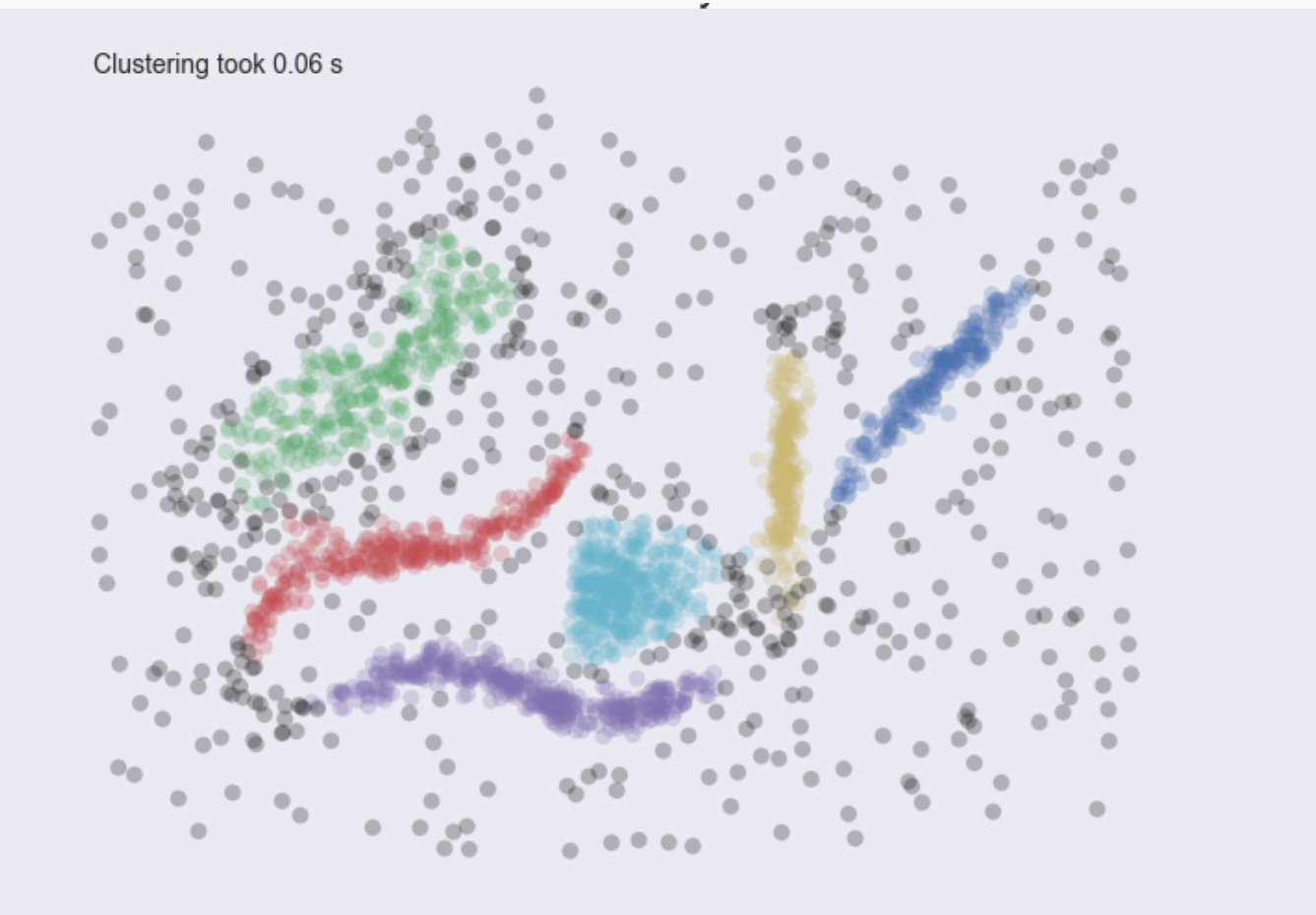
What is clustering?



Data consists of different subcategories and outliers



Data consists of different subcategories and outliers



Why cluster your data?

- (Sub-) classification
- Outlier detection

There are only two approaches to clustering*:

There are only two approaches to clustering*:

- attribution to exemplars / archetypes

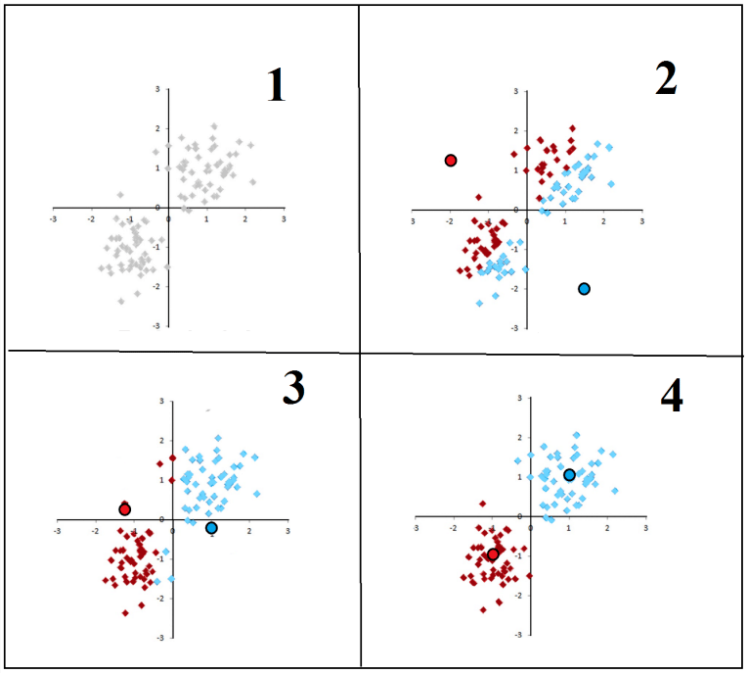
There are only two approaches to clustering*:

- attribution to exemplars / archetypes
- agglomeration based on nearest neighbour distances

NOT A CLUSTERING ALGORITHM

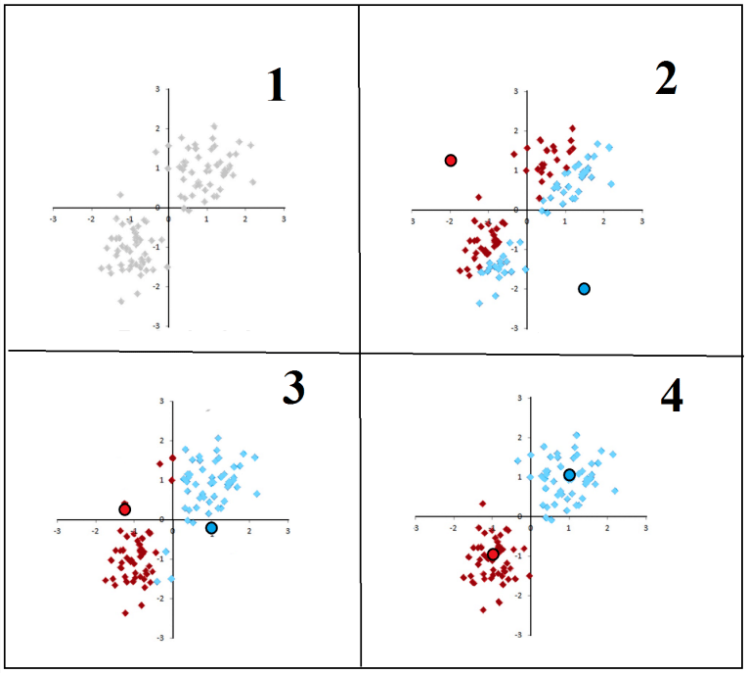
but a *partitioning* algorithm!

k-Means clustering algorithm



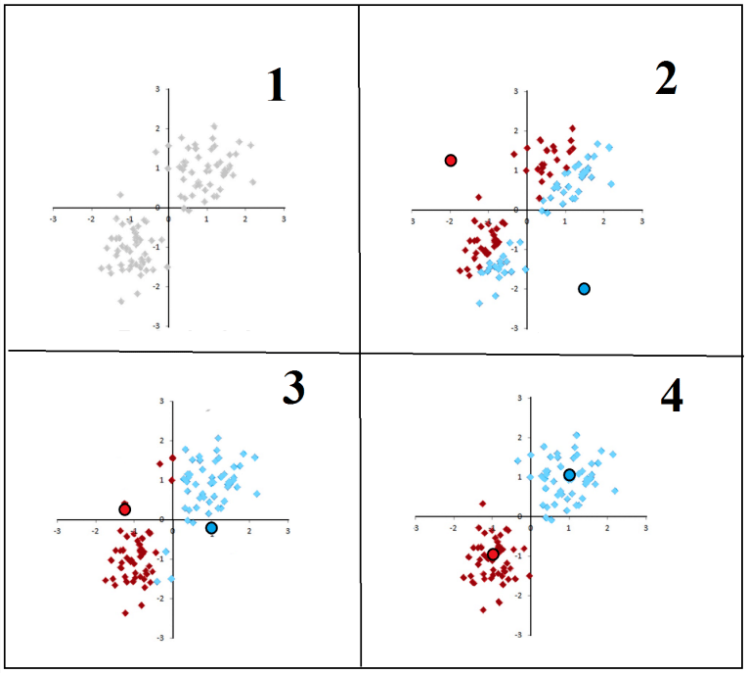
1. Place k **exemplars** randomly within the range of your data.

k-Means clustering algorithm



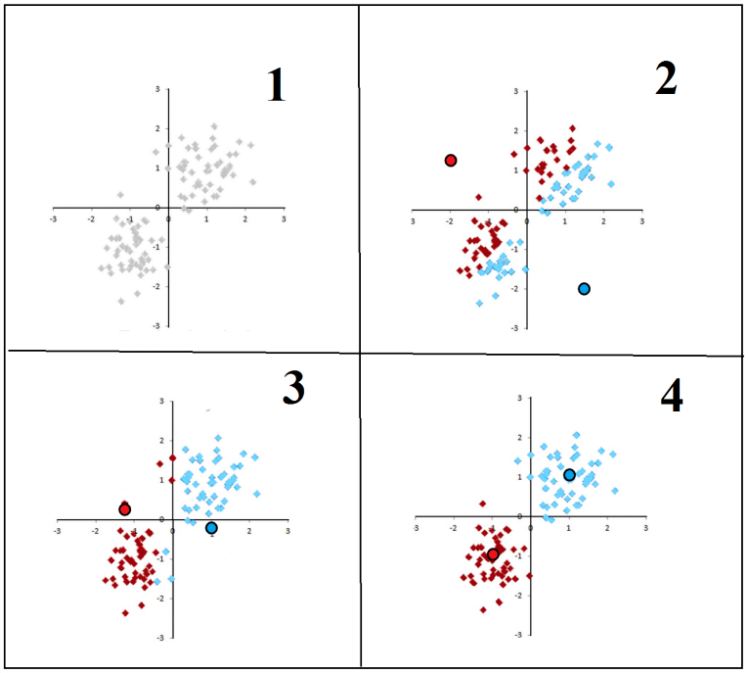
1. Place k **exemplars** randomly within the range of your data.
2. Compute the (euclidean) distances of all points to their nearest exemplars.

k-Means clustering algorithm



1. Place k **exemplars** randomly within the range of your data.
2. Compute the (euclidean) distances of all points to their nearest exemplars.
3. Shift the exemplars towards the center of mass of their assigned points.

k-Means clustering algorithm



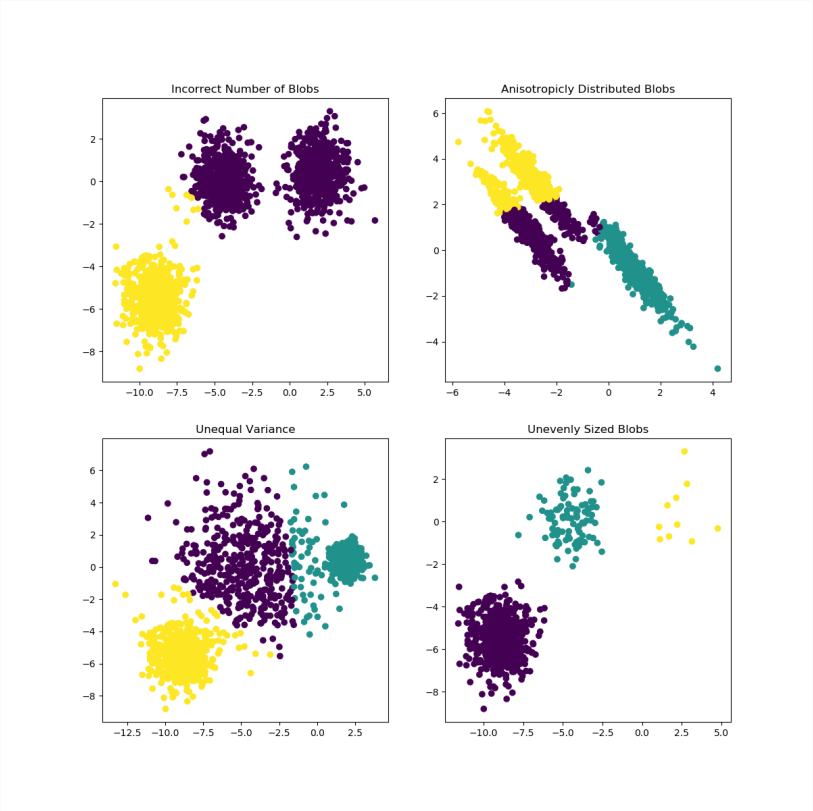
1. Place k **exemplars** randomly within the range of your data.
2. Compute the (euclidean) distances of all points to their nearest exemplars.
3. Shift the exemplars towards the center of mass of their assigned points.
4. Repeat steps 2&3.

Problems with k-means clustering

NOT A CLUSTERING ALGORITHM

but a *partitioning* algorithm!

Problems with k-means clustering



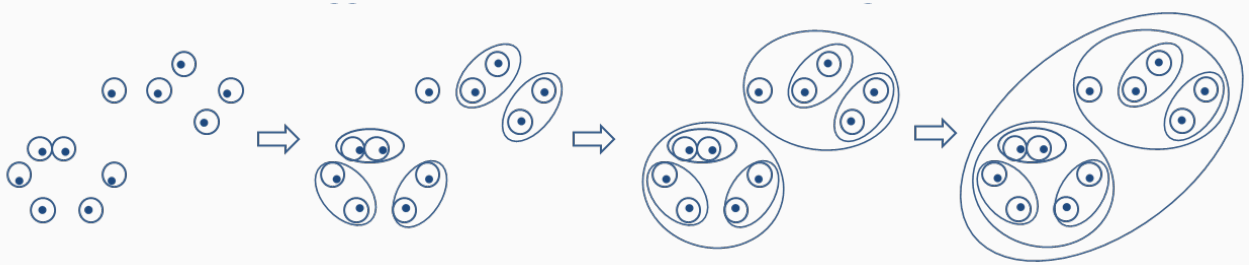
- Need to know *k a priori*.
- Assumes globular clusters.
- Assumes clusters of similar variance.
- No concept of outliers.

Agglomerative/hierarchical clustering

NOT A CLUSTERING ALGORITHM

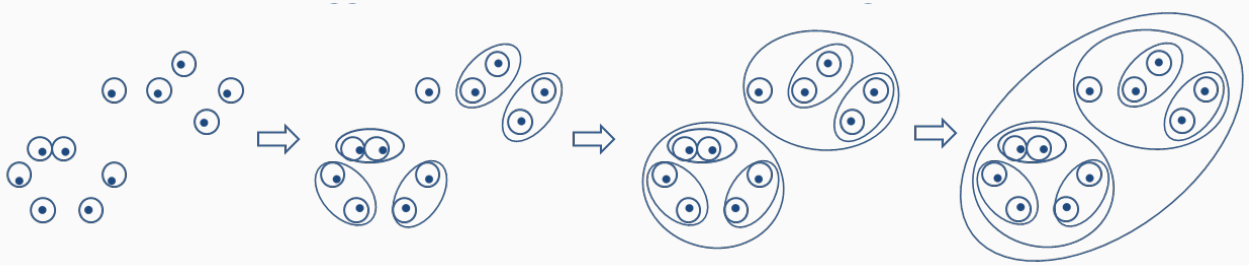
but an *ordering* algorithm!

Agglomerative/hierarchical clustering algorithm



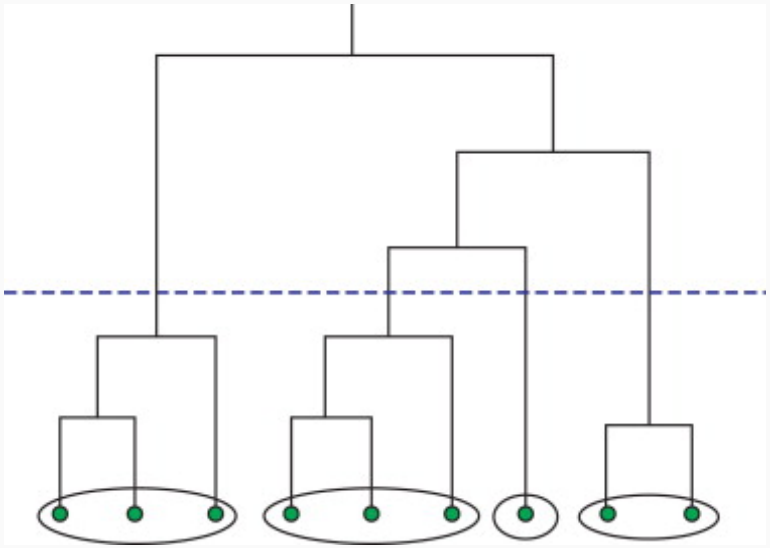
1. Merge nearest neighbours into one cluster (*).

Agglomerative/hierarchical clustering algorithm



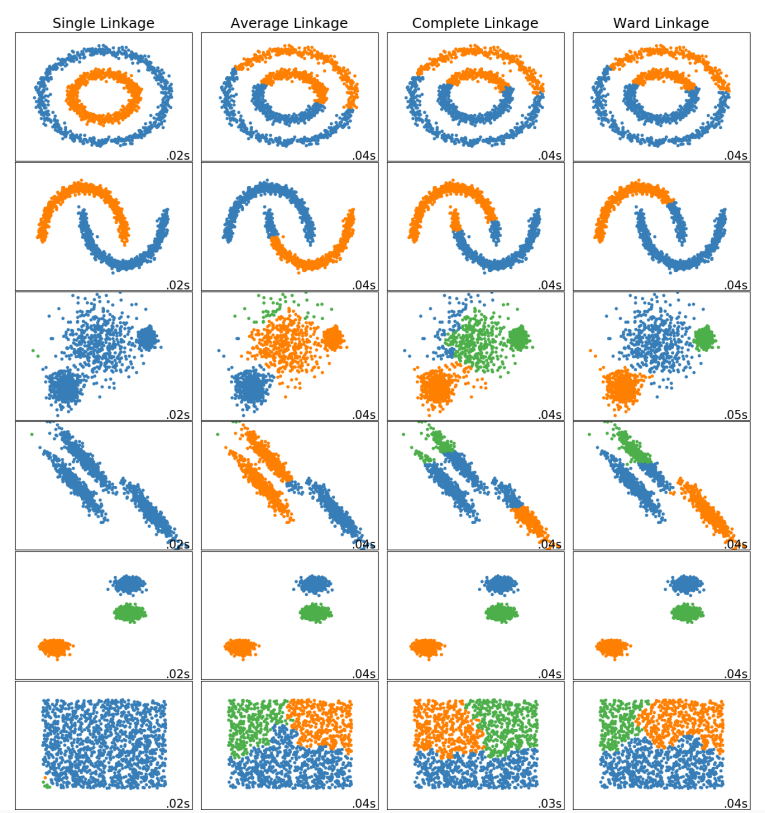
1. Merge nearest neighbours into one cluster (*).
2. Repeat step 1.) until a single cluster remains.

Agglomerative/hierarchical clustering algorithm



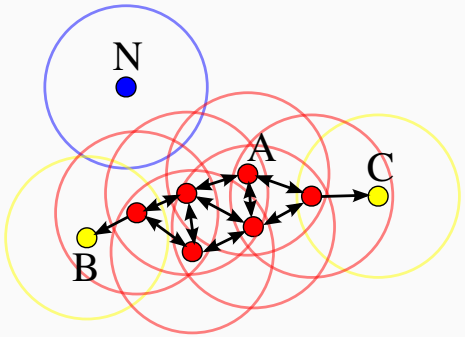
1. Merge nearest neighbours into one cluster (*).
2. Repeat step 1.) until a single cluster remains.
3. The merge operations result in a tree, which is cut at a chosen height.

Problems with agglomerative/hierarchical clustering



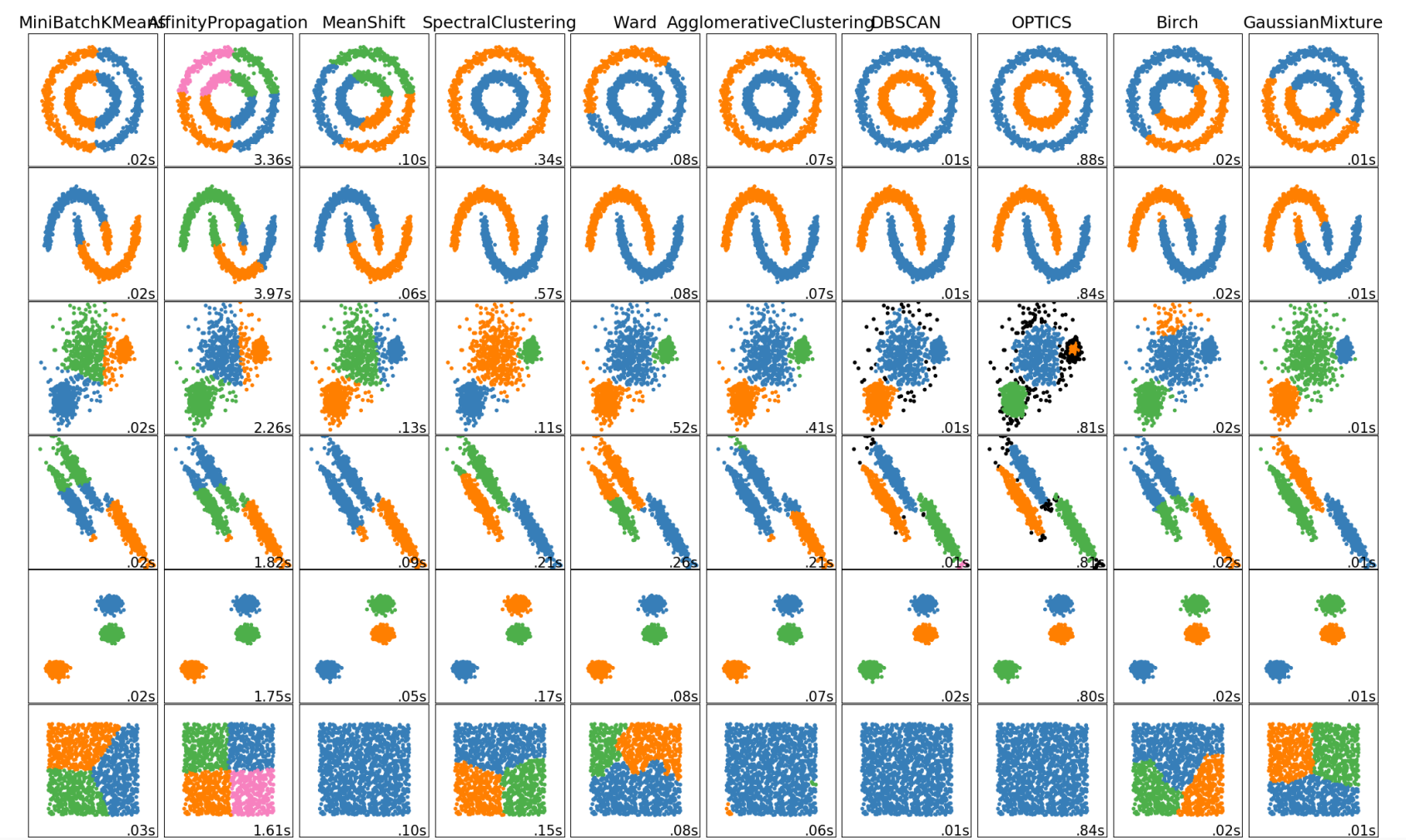
- Need to know the correct cutoff.
- Rich-get-richer dynamics may result in under-clustering.
- No concept of outliers.
- Bridges may connect clusters that do not belong together.

DBSCAN



1. For a given minimum number of points p and a distance ϵ, partition the data into
   - core points (at least p points within ϵ)
   - reachable points (less than p points within ϵ)
   - outliers (no points within ϵ)
2. Merge core points based on Euclidean distance using agglomerative clustering.
3. Add reachable points to their nearest cluster.

The problem with DBSCAN: variable density clusters



HDBSCAN

- Instead of counting points that are ϵ away to estimate local density, determine how far away your k^{th}-nearest neighbour is (core_k).
- Merge points based on reachability using agglomerative clustering.

$$\text{reachability}(a, b) = \textbf{max}(\text{core}_k(a), \text{core}_k(b), d(a, b)) \tag{12}$$

Further reading

https://scikit-learn.org/stable/modules/clustering.html

https://hdbscan.readthedocs.io

Summary: dimensionality reduction

Dimensionality reduction:

- Always reduce the dimensionality of your data if $D > 5$!
- Use PCA or its variants first.
- Use UMAP second. Don't use UMAP without reducing the dimensionality first.

Summary: clustering

- Don't use k-means.
- Don't use hierarchical clustering for clustering (do use it for obtaining hierarchies / ordering).
- Use HDBSCAN for clustering.

Tutorial

scikit-learn API

```
instance = Algorithm(param1, param2)
instance.fit(X, param3)
Xtransformed = instance.transform(X, param4)
```

Your Mission should you choose to accept it

1. load the data
2. get a feel for the data
3. clean the data
4. reduce dimensionality
    4.1 PCA
    4.2 UMAP
5. cluster using HDBSCAN