Two killer applications in manifold learning/dimred

- t-distributed stochastic neighbor embedding (t-SNE)
- Uniform manifold approximation and projection (UMAP)

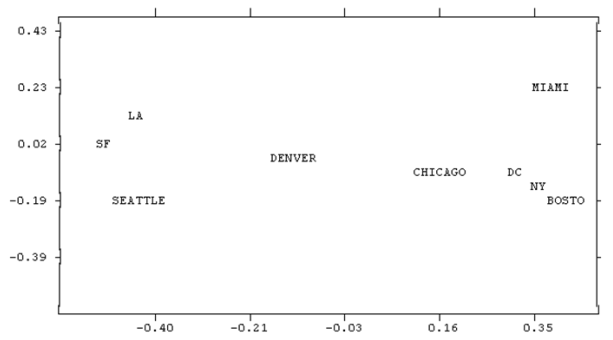
Reminder on Multi-dimensional scaling (MDS)

Input: a distance or dissimilarity matrix

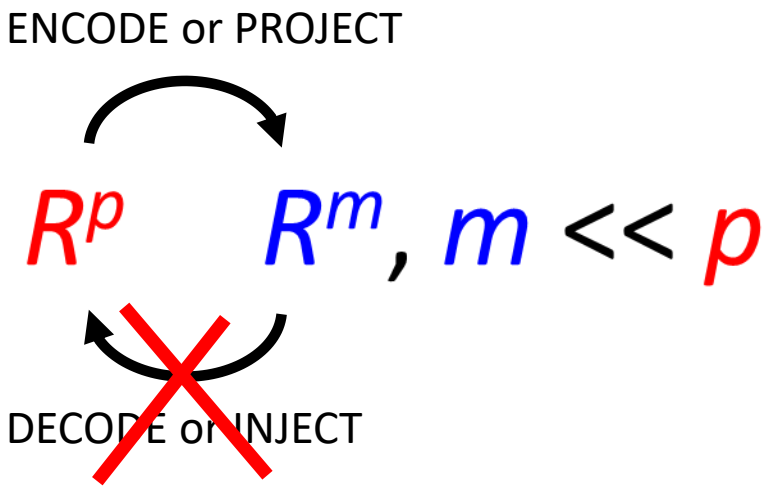
- How to arrange points in 2D (or mD) that the Euclidean distance between them would reproduce the input matrix as much as possible?

From http://www.analytictech.com/networks/mds.htm

		1	2	3	4	5	6	7	8	9
		BOST	NY	DC	MIAM	CHIC	SEAT	SF	LA	DENV
1	BOSTON	0	206	429	1504	963	2976	3095	2979	1949
2	NY	206	0	233	1308	802	2815	2934	2786	1771
3	DC	429	233	0	1075	671	2684	2799	2631	1616
4	MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
5	CHICAGO	963	802	671	1329	0	2013	2142	2054	996
6	SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
7	SF	3095	2934	2799	3053	2142	808	0	379	1235
8	LA	2979	2786	2631	2687	2054	1131	379	0	1059
9	DENVER	1949	1771	1616	2037	996	1307	1235	1059	0



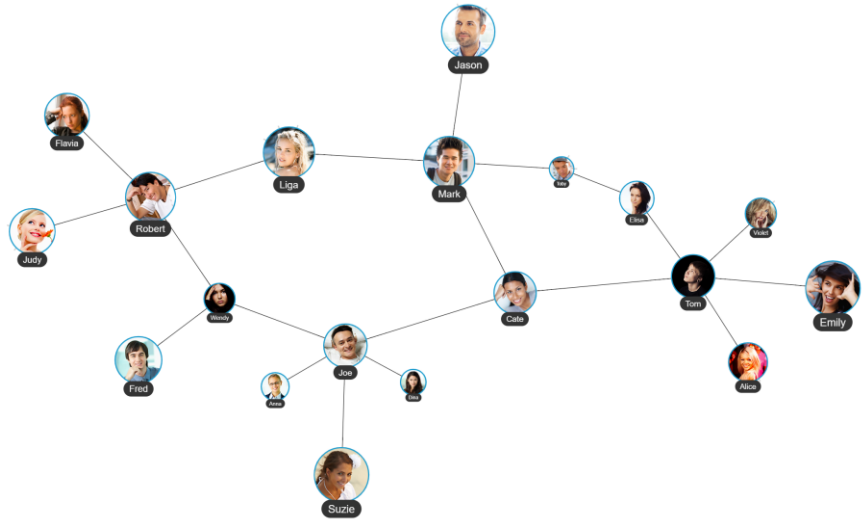
t-SNE and UMAP are projective methods



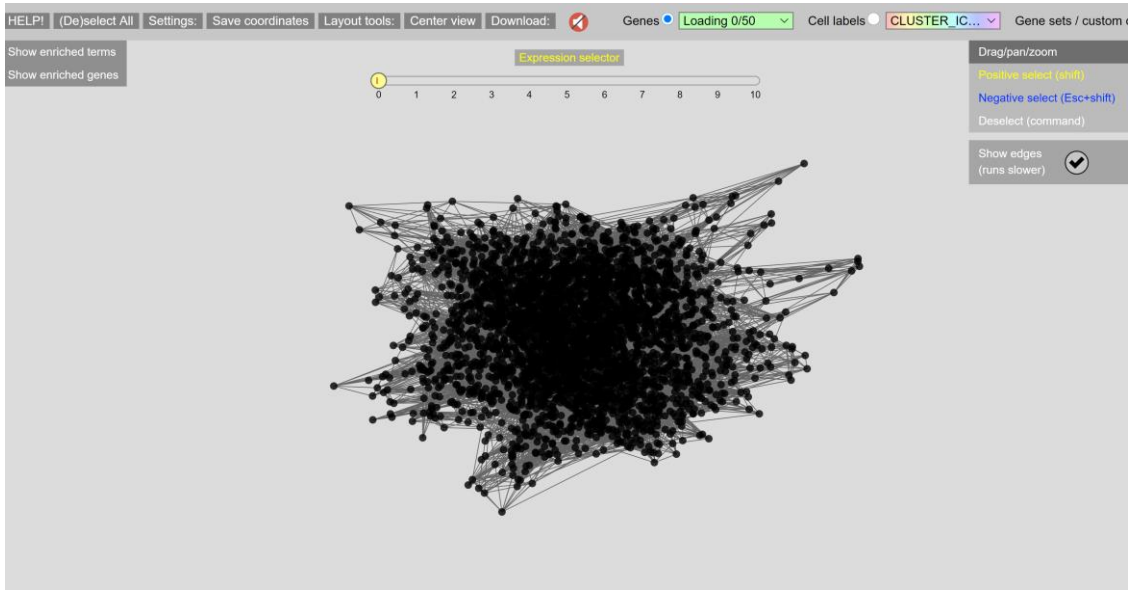
Variant 2: The projector is known only for $\boldsymbol{y} \in \boldsymbol{X}$

Using graph layouts to reduce data dimensionality

- Simple algorithm:
  - Compute the KNN-graph
  - Apply graph drawing (layouting) algorithm to visualize it in 2D (3D)
  - Force-directed layout (e.g., Fruchterman–Reingold's algorithm)



https://demo.zoomcharts.com/net-chart/examples/layout/layout-forced.html



https://www.ihes.fr/~zinovyev/mosaic/SPRING/springViewer.html?datasets/CHLA9_nufp

Problems with drawing knn-graphs

- In many situations, it is already a usefull solution for visualizing the data, but…

- The structure of knn graph heavily depends on the local density
- Point crowding
- The graph is very ‘rigid’

- Solutions:
  - Makes the structure of the neighbourhood graph more adapted to the density variations
  - Introduce a ‘softer’ probabilistic model in constructing the neighborhood graph – instead of a ‘hard’ connection a probability of being connected

- Two methods have become famous for this in recent years: t-SNE and UMAP

Stochastic neighbor embedding (SNE and t-SNE)

t-SNE uses the Kullback-Leibler (KL) divergence

- **Reminder:**

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

- KL measures ‘distance’ from distribution P to Q
- Not a metric function - not symmetric!
- Measures how much more information is contained in P with respect to Q (similar to the mutual information for which Q is the product of marginal distributions)

t-distributed stochastic neighbor embedding (t-SNE)

- SNE by Sam Roweis and Geoffrey Hinton in 2002
- Laurens van der Maaten proposed the t-distributed variant (t-SNE) in 2008

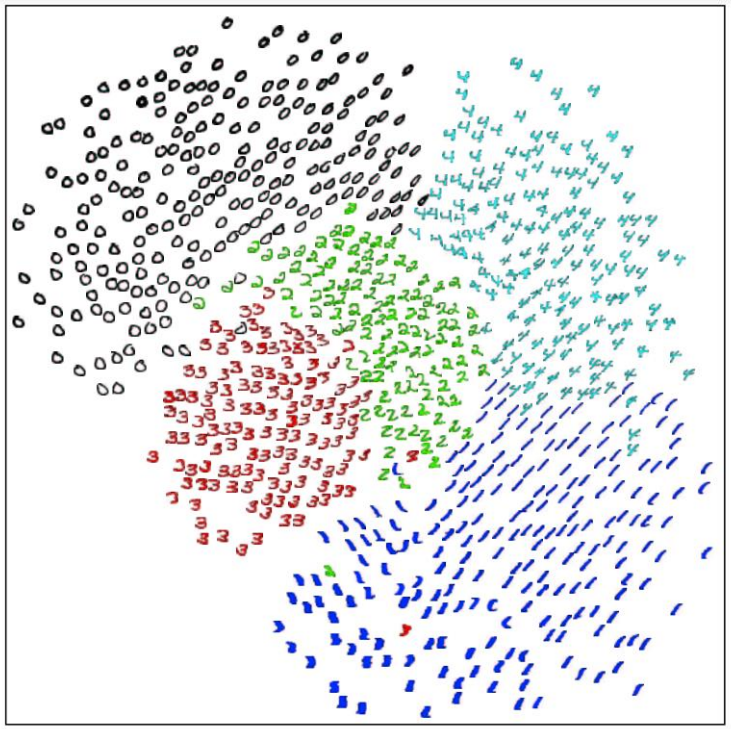
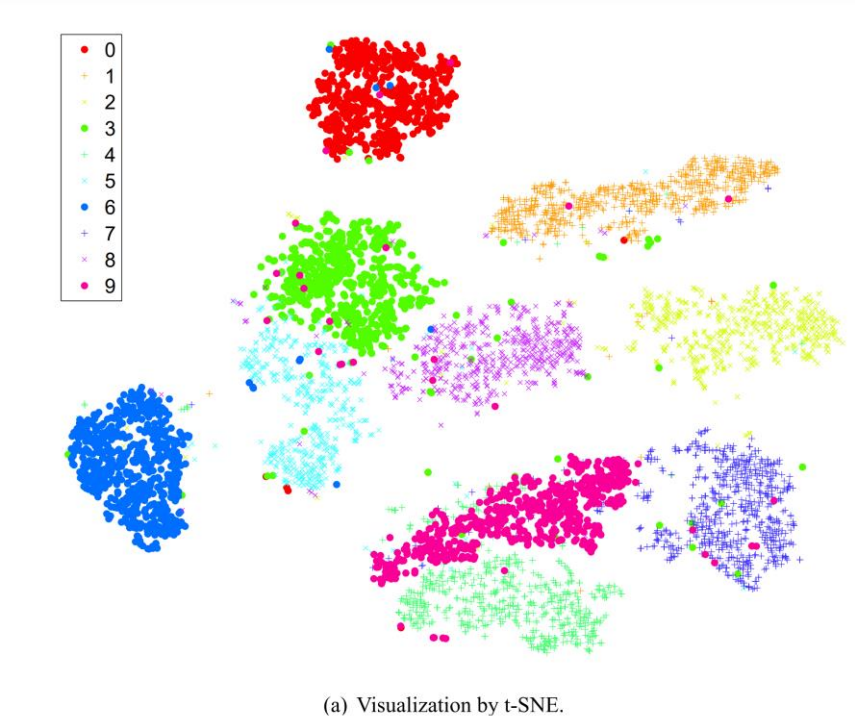


Figure 1: The result of running the SNE algorithm on 3000 256-dimensional grayscale images of handwritten digits. Pictures of the original data vectors $\mathbf{x}_i$ (scans of handwritten digit) are shown at the location corresponding to their low-dimensional images $\mathbf{y}_i$ as found by SNE. The classes are quite well separated even though SNE had no information about class labels. Furthermore, within each class, properties like orientation, skew and stroke-thickness tend to vary smoothly across the space. Not all points are shown: to produce this display, digits are chosen in random order and are only displayed if a 16 x 16 region of the display centered on the 2-D location of the digit in the embedding does not overlap any of the 16 x16 regions for digits that have already been displayed.



(a) Visualization by t-SNE.

Notion of perplexity

- In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample
- It may be used to compare probability models.
- A low perplexity indicates the probability distribution is good at predicting the sample
- The perplexity PP of a discrete probability distribution *p* is defined as

$$PP(p) := 2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

- For Gaussian distribution *PP(p)* = $\frac{1}{2}\ln(\sqrt{2\pi e}\sigma)$

Stochastic neighbor embedding (SNE)

- Conditional probability

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$$

Reminds Kernel Density Estimate of PDF in point *i* , but σ_i depends on *i!* (adaptive KDE)

Conditional probability of that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

Stochastic neighbor embedding (SNE)
evaluating σ_i

- Perplexity of distribution PP($p_{j|i}$) must be the same for each *i*

PP($p_{j|i}$) = PP = const

- Perplexity PP is the main parameter of the method
- Large perplexity -> larger σs
- Maximum perplexity is *N* – 1, corresponding to $\sigma = \infty$ (in this case the similarities are distributed uniformly)

Stochastic neighbor embedding (SNE)

R^p $\qquad\qquad$ R^m, $m << p$

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} \qquad q_{ij} = \frac{\exp(-||\mathbf{y}_i - \mathbf{y}_j||^2)}{\sum_{k\neq i}\exp(-||\mathbf{y}_i - \mathbf{y}_k||^2)}$$

We try to minimize Kullback-Leibler divergence between *p* and *q*

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_i KL(P_i||Q_i)$$

Expression for the gradient of *C*:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 2\sum_j (\mathbf{y}_i - \mathbf{y}_j)(p_{ij} - q_{ij} + p_{ji} - q_{ji})$$

t-distributed Stochastic neighbor embedding (t-SNE)

R^p $\qquad$ R^m, $m << p$

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} \qquad q_{ij} = \frac{\left(1+\|y_i - y_j\|^2\right)^{-1}}{\sum_{k\neq l}\left(1+\|y_k - y_l\|^2\right)^{-1}}$$

We try to minimize Kullback-Leibler divergence between *p* and *q*

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_i KL(P_i||Q_i)$$

Expression for the gradient of *C*:

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1+\|y_i - y_j\|^2\right)^{-1}.$$

Discussion on t-SNE

- Some remarks:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$$

- if $||x_i - x_j|| \gg \sigma_i$ then $p_{i|j} \approx 0$

- if $||x_i - x_j|| \ll \sigma_i$ (large perplexity) then

$p_{i|j} \approx 1 - \frac{\left||x_i - x_j|\right|^2}{2\sigma_i^2} / \sum_k (1 - \frac{\left||x_i - x_j|\right|^2}{2\sigma_i^2})$

and t-SNE becomes very close to Multi-Dimensional Scaling (PCA)

Point ‘crowding problem’

- In high dimension we have more room, points can have a lot of different neighbors
- In 2D a point can have a few neighbors at distance one all far from each other
- This is the ”crowding problem” - we don’t have enough room to accommodate all neighbors
- This is a problem with SNE (and other dimensionality reduction methods such as MDS)
- t-SNE solution: change the Gaussian in q to a heavy tailed distribution (Student distribution)

Gradient for SNE

$$\frac{\partial C}{\partial \mathbf{y}_i} = 2\sum_j (\mathbf{y}_i - \mathbf{y}_j)(p_{ij} - q_{ij} + p_{ji} - q_{ji})$$

Gradient for t-SNE – less attractive

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}.$$

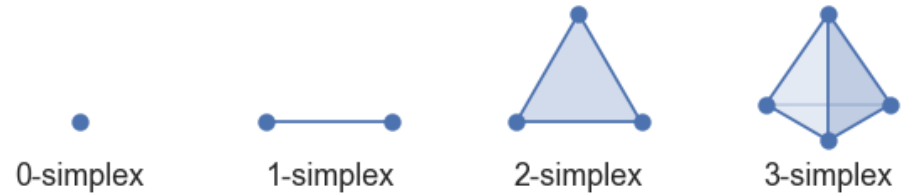
Discussion on t-SNE

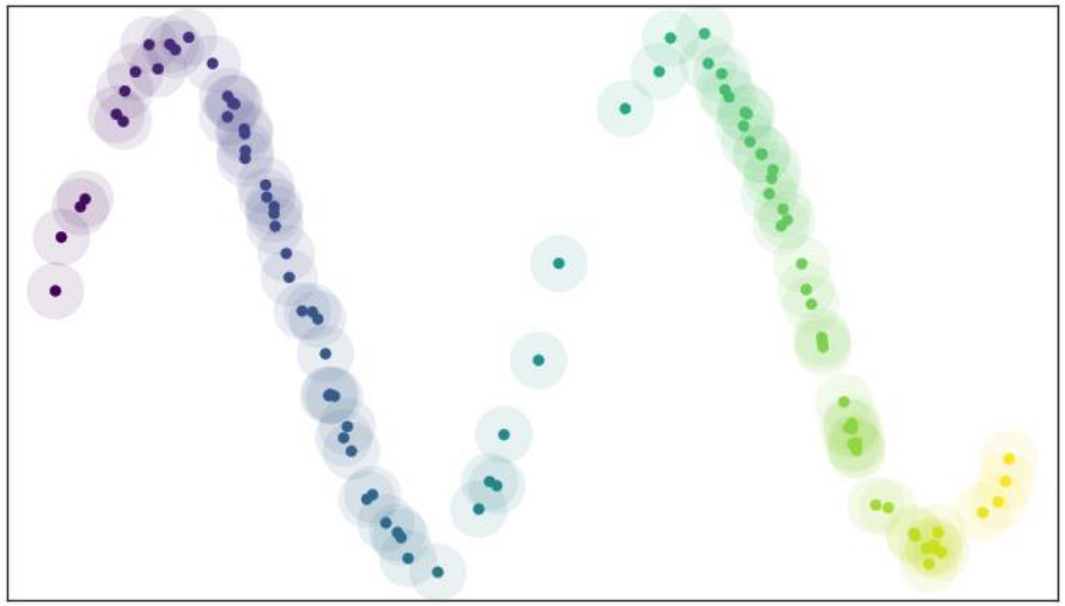
- Low values of perplexity will unveil the local structure in the data, whereas high values of perplexity will enhance the emergence of the global structure at the cost of blurring the local structure
- Short and long distances
- Pro: In linear and non-linear principal manifolds, distant points can become close after projection; in t-SNE it usually does not happen
- Cons: Exaggerated clustering (sometimes)

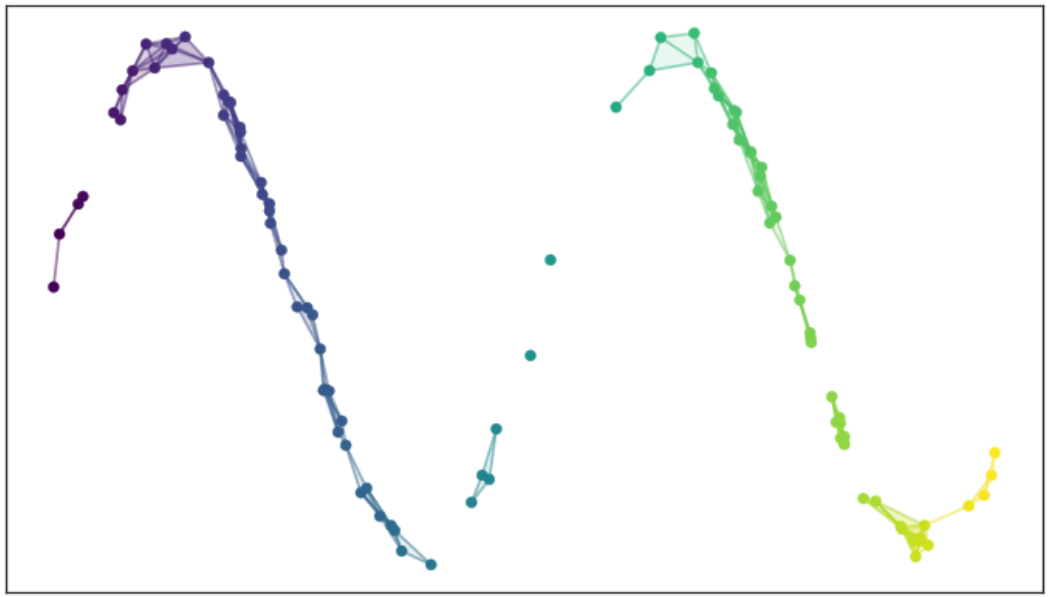
Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP)

- Starts with a ‘fuzzy simplicial complex’ but ends with a simple weighted neighbourhood graph







https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

Constructing weighted neighbourhood graph

For each point *i,* define distance to the nearest neighbour

$$\rho_i = \min\{d(x_i, x_{i_j}) \mid 1 \leq j \leq k, d(x_i, x_{i_j}) > 0\},$$

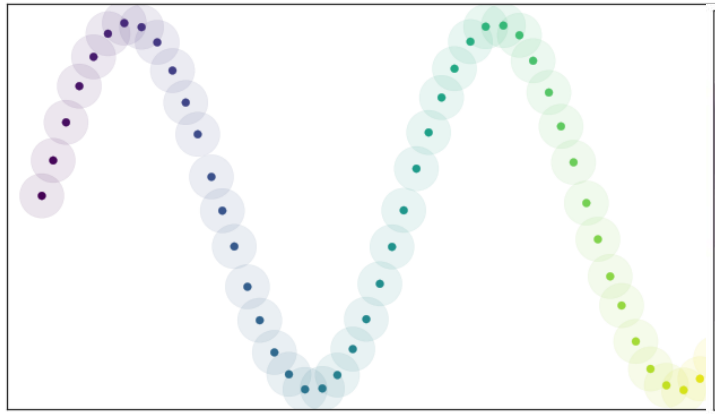
Define σ_i using the following equation (local dispersion)

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) = \log_2(k)$$

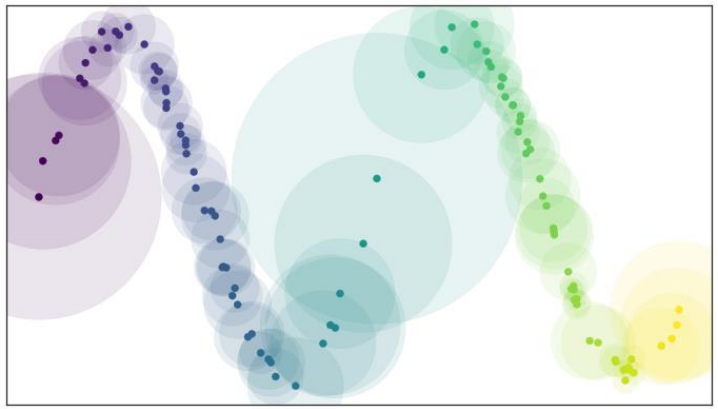
Weight of the neighbourhood graph

$$w((x_i, x_{i_j})) = \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right)$$

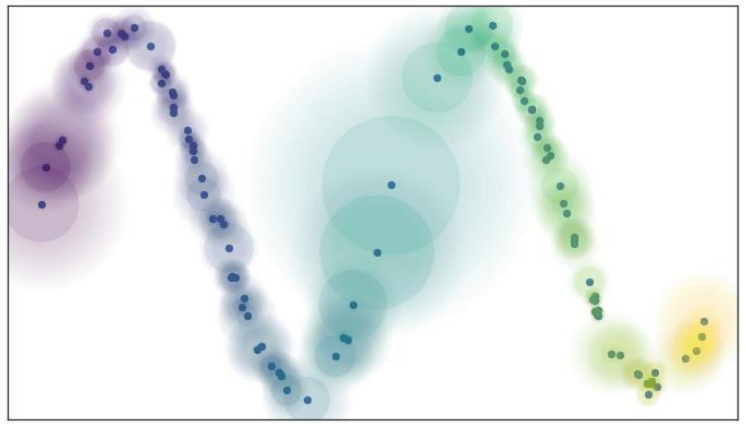
‘Uniform’ case



kNN graph

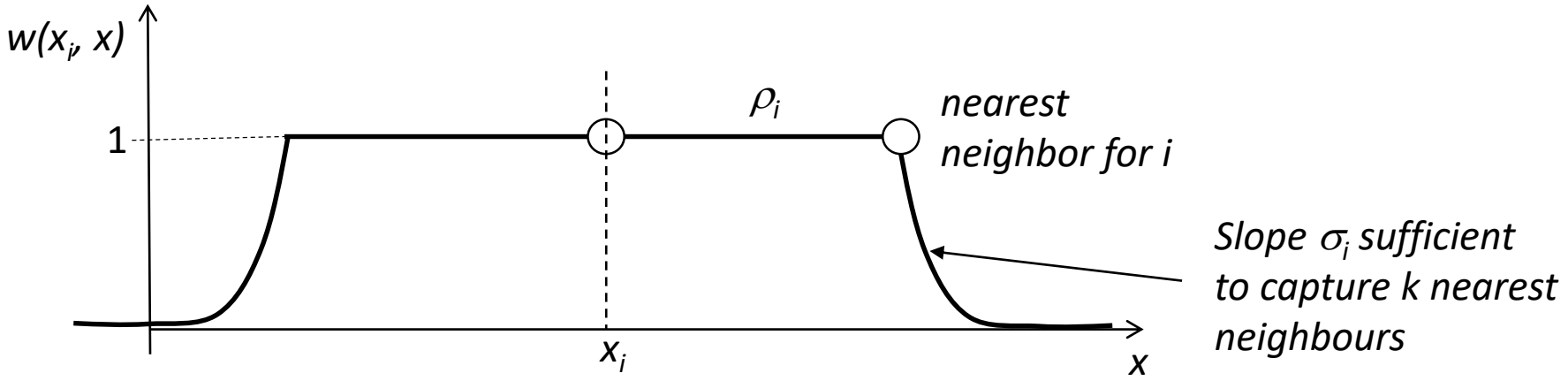


Fuzzy weighted neighbourhood graph (‘UMAP-modified kNN’)

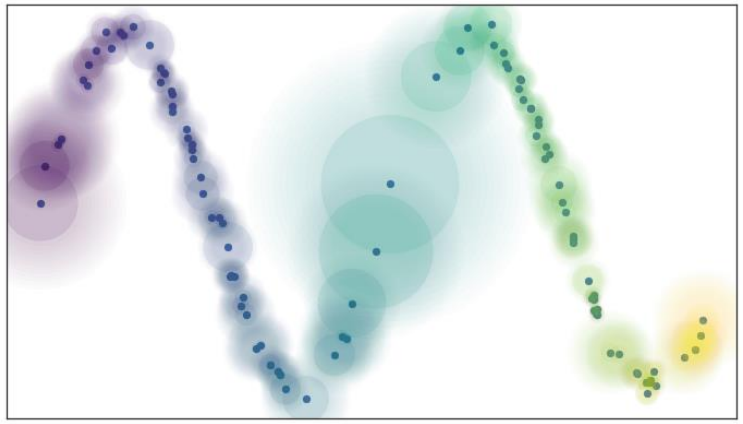


k=4

$$w((x_i, x_{i_j})) = \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right)$$



Fuzzy weighted neighbourhood graph ('UMAP-modified kNN')



Minimizing ‘cross-entropy’

Preservation of small distances

Preservation of large distances

$$\sum_{e \in E} \left[w_h(e) \log \left(\frac{w_h(e)}{w_l(e)} \right) + (1 - w_h(e)) \log \left(\frac{1 - w_h(e)}{1 - w_l(e)} \right) \right]$$

$1 \geq w_h > 0$ – weights in high-dimensional space

$1 \geq w_l > 0$ – weights in low-dimensional space

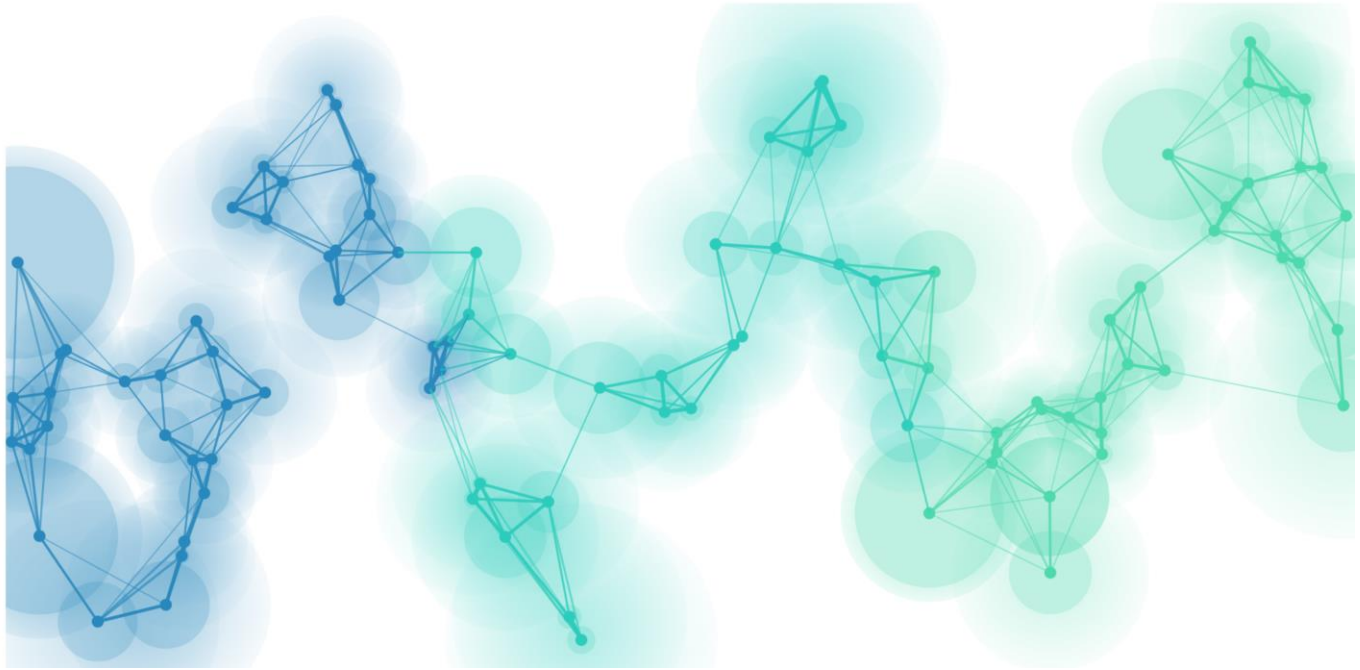
Actual algorithm is very close to force-directed layout algorithm

Hyperparameters of UMAP

- Min-dist:

minimal distance between

points in low-dimensional space

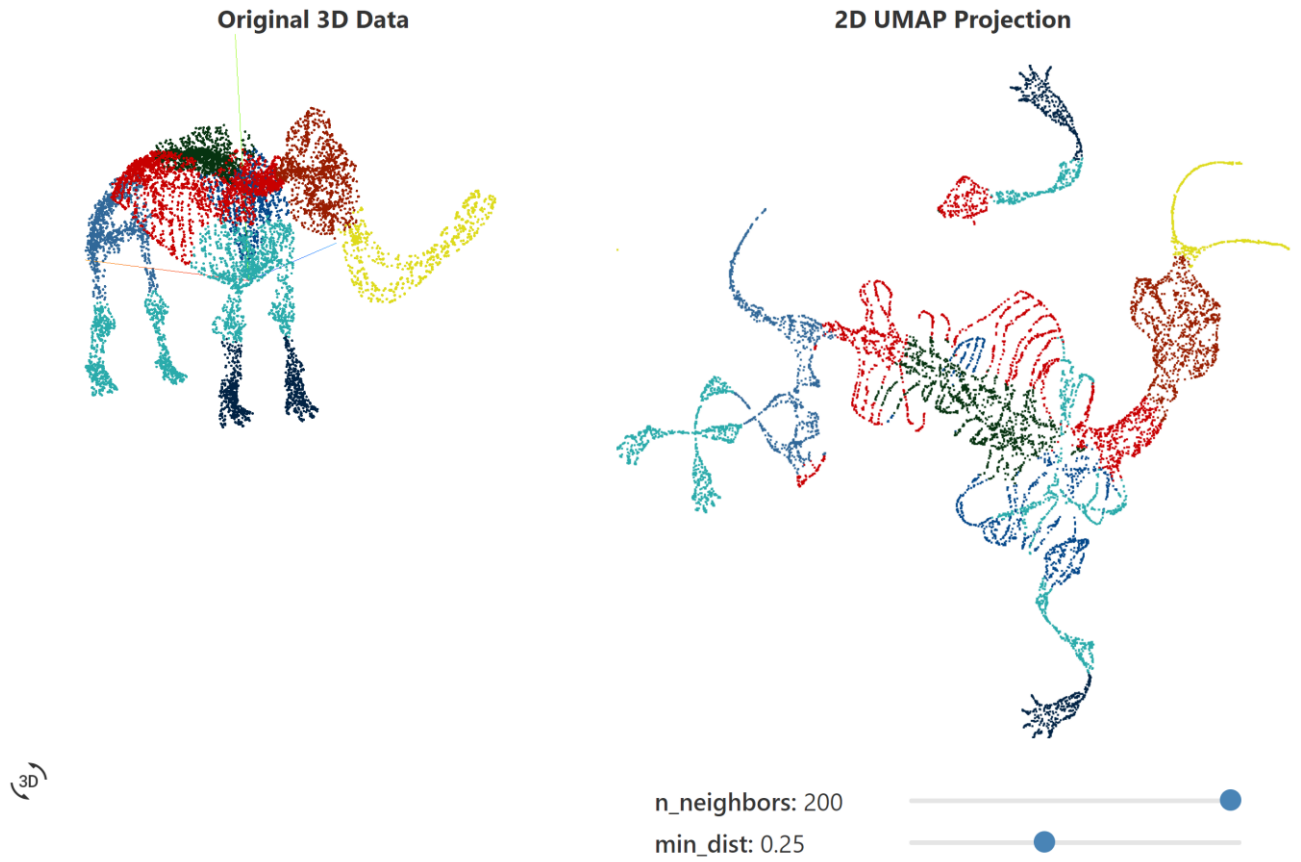


- N_neighbours:

k in the kNN graph

https://pair-code.github.io/understanding-umap/supplement.html

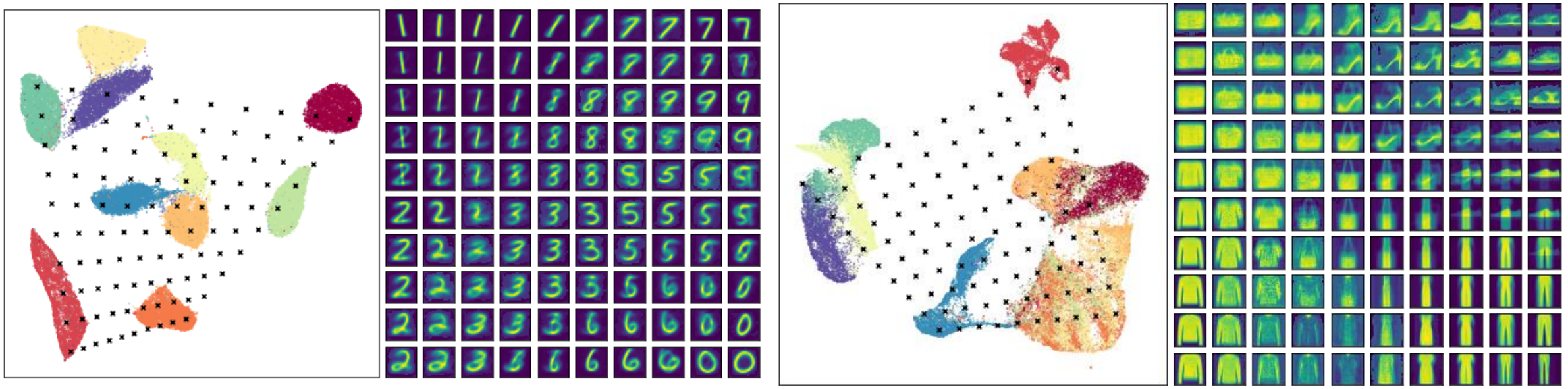
Toy example to play with parameters: *Projecting a mammoth from 3D to 2D*



https://pair-code.github.io/understanding-umap/

https://duhaime.s3.amazonaws.com/apps/umap-zoo/index.html

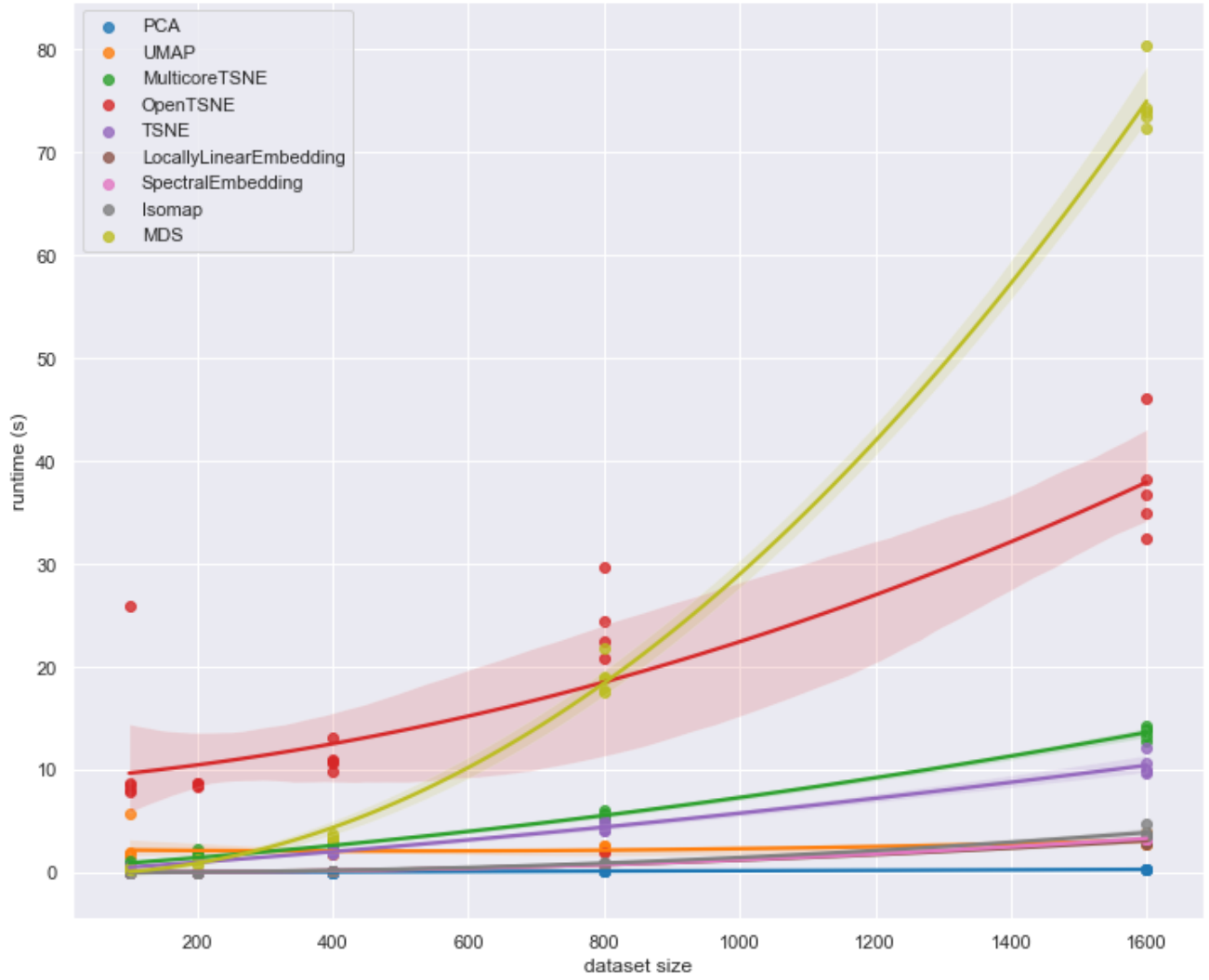
UMAP ‘inverse transform’:
provide a sort of DECODE/INJECT function



https://umap-learn.readthedocs.io/en/latest/inverse_transform.html

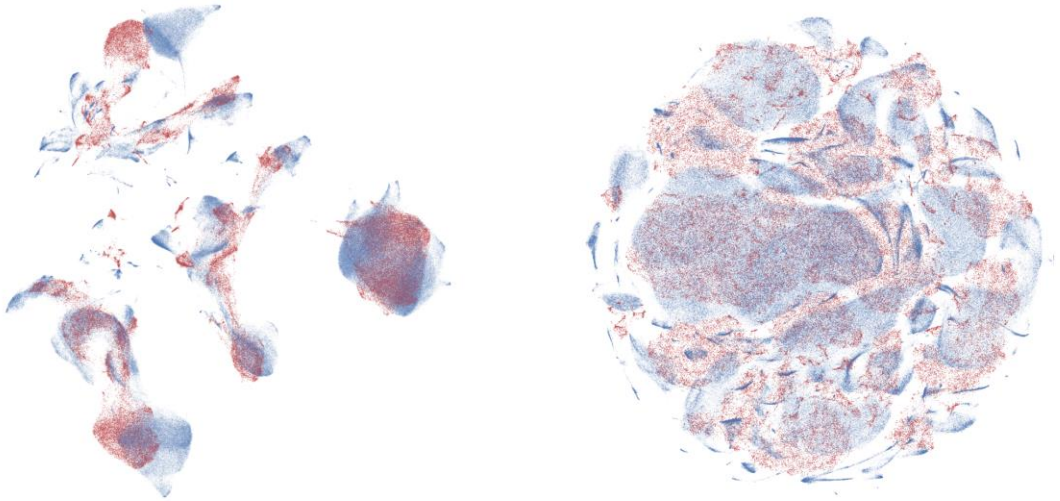
Comparing tSNE and UMAP

- Speed



Comparing tSNE and UMAP

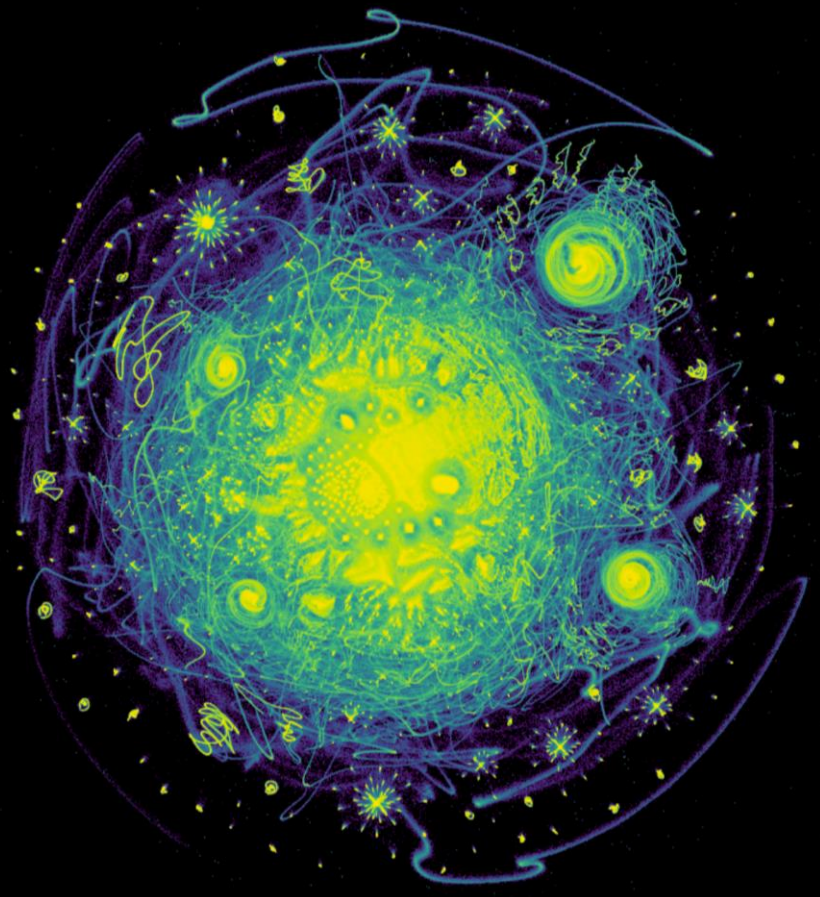
- UMAP better represents the global structure of the dataset
- UMAP is way faster than t-SNE
- UMAP is more stable to subsampling than t-SNE
- UMAP can work directly in very high ambient dimensionalities (>10^6)



(a) UMAP (b) t-SNE

Figure 7: Procrustes based alignment of a 10% subsample (red) against the full dataset (blue) for the flow cytometry dataset for both UMAP and t-SNE.

from https://arxiv.org/pdf/1802.03426.pdf



igure 13: Visualization of 30,000,000 integers as represented by binary vector
f prime divisibility, colored by density of points.

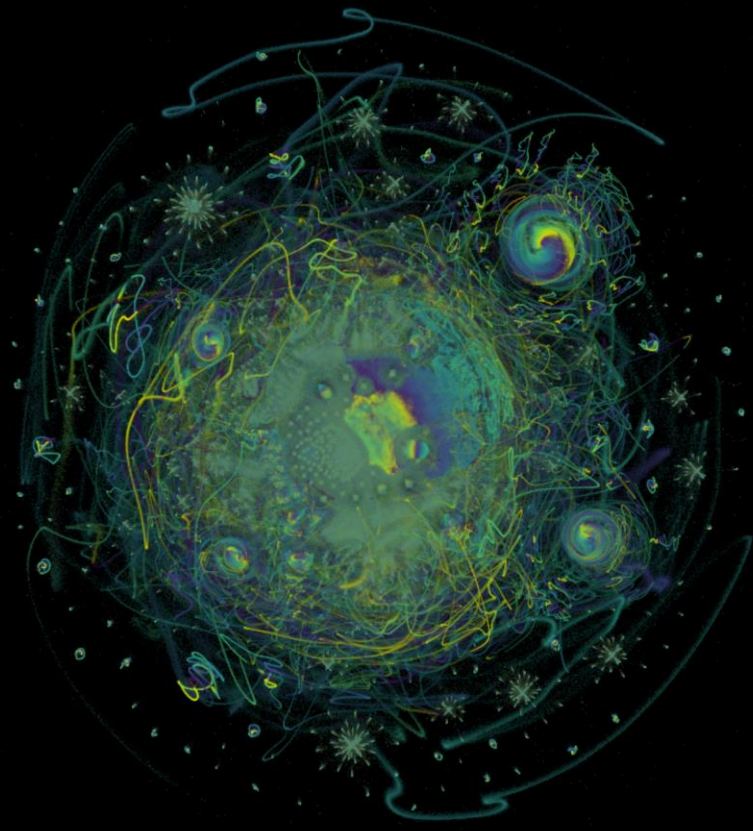


Figure 14: Visualization of 30,000,000 integers as represented by binary vectors of prime divisibility, colored by integer value of the point (larger values are green or yellow, smaller values are blue or purple).

from https://arxiv.org/pdf/1802.03426.pdf

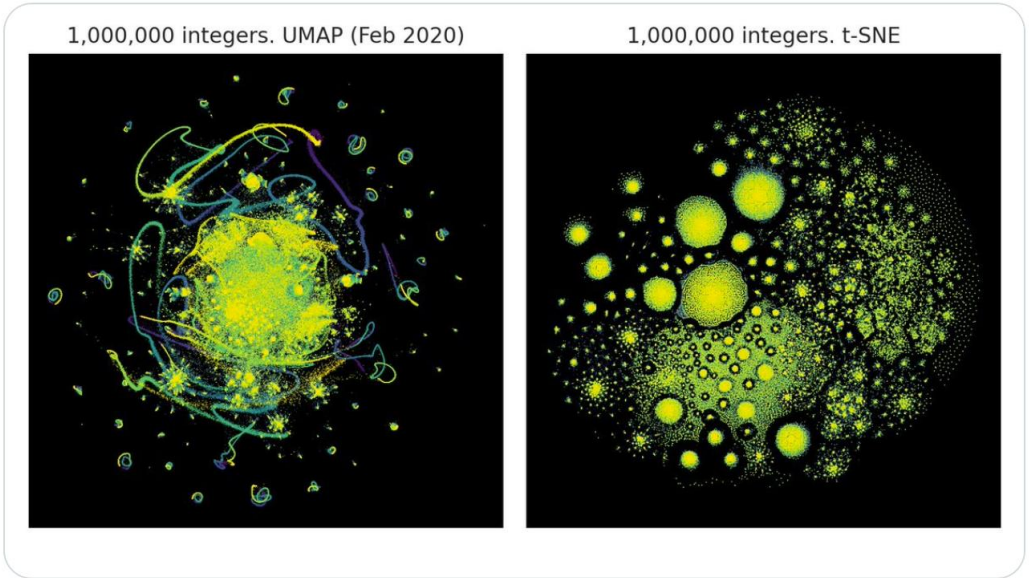
Thread

Dmitry Kobak
@hippopedoid

Remember the galaxy-like UMAP visualization of integers from 1 to 1,000,000 represented as prime factors, made by @jhnhw?

I did t-SNE of the same data, and figured out what the individual blobs are. Turns out, the swirly and spaghetti UMAP structures were artifacts :-(

[1/n]



Comments to both t-SNE and UMAP methods

- Hyperparameters really matter
- Cluster sizes in a UMAP plot mean nothing
- Distances between clusters might not mean anything
- Random noise doesn’t always look random
- You may need more than one plot
- For large ‘neighbourhood’ parameters, both methods give results similar to Multi-dimensional scaling
- Both can work with non-Euclidean metrics in R^p

Minimum Distortion Embedding (MDE)

Agrawal, Ali and Boyd, ***arXiv 2021***

- "Modern" version of metric MDS
- Avoids computing the complete distance matrix
- Quadratic MDE ($f_{ij}(d_{ij}) = w_{ij}d_{ij}^2$, $\boldsymbol{w_{ij}}$ represents similarity between $\boldsymbol{i}$ and $\boldsymbol{j}$) can be reduced to an eigenproblem
- pyMDE Python package

$$\mathbf{E}(X) = \frac{1}{|E|}\sum_{(i,j)\in E} f_{ij}(d_{ij})$$

f_{ij}: Distortion function between i and j.

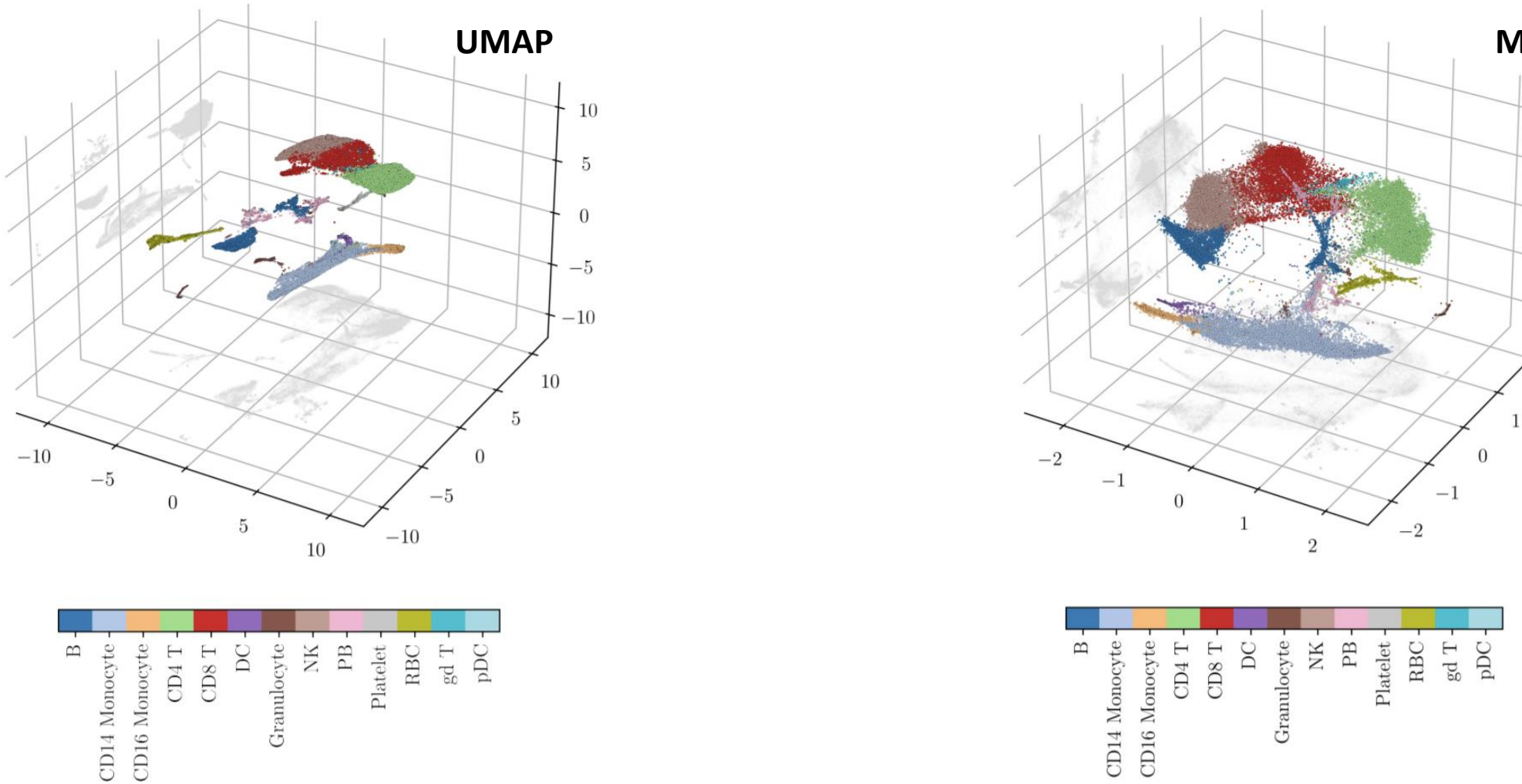
d_{ij}: Embedding distance between X_i and X_j.

X: Objective embedding

E: All sample pairs if tractable, otherwise a set of edges

Of course, for positive f_{ij} the problem is trivial. The authors propose several constraints (e.g. standardized) to avoid $X = 0$.

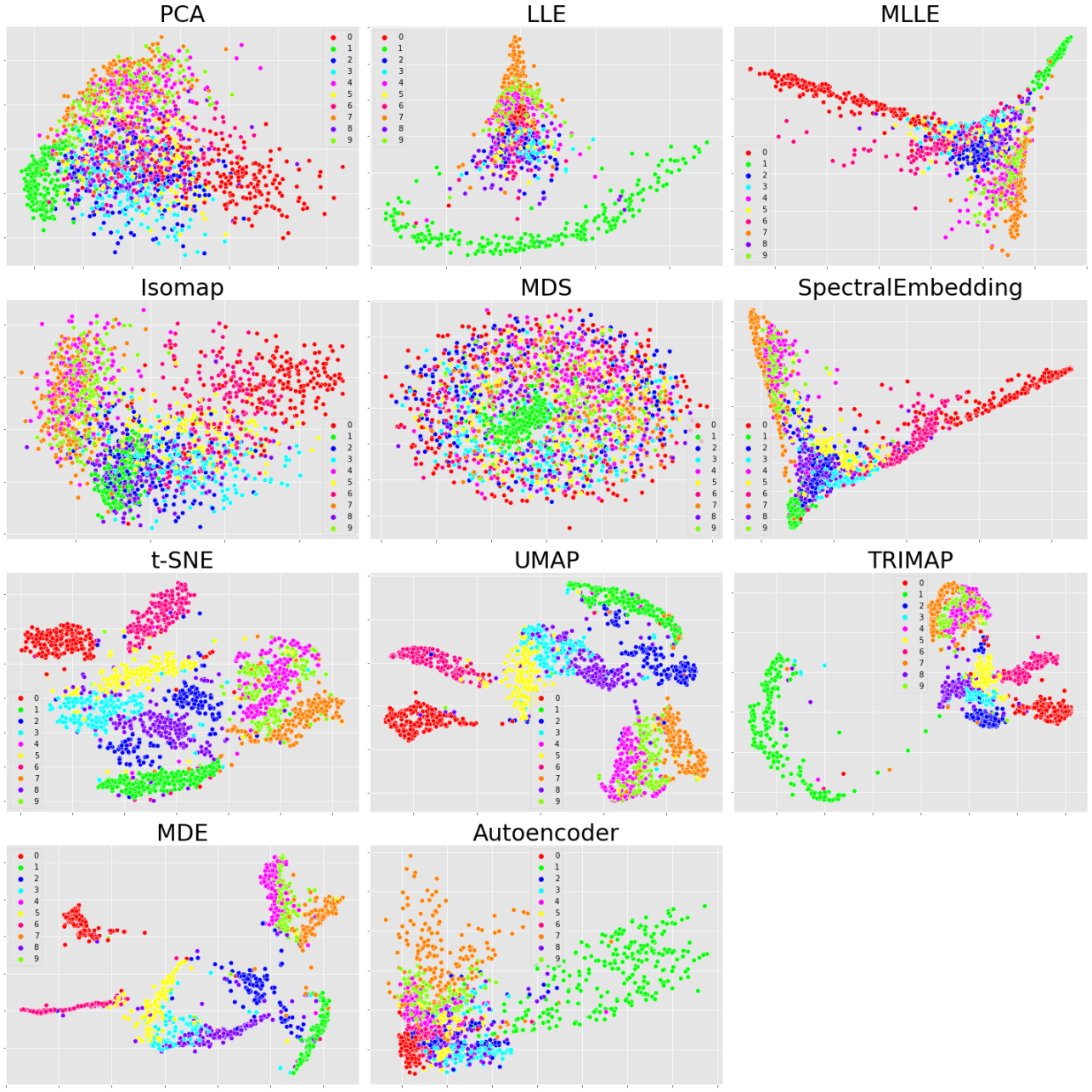
Usage example: RNA-seq dataset, $n > 40{,}000$



Comparison of (many) methods:

https://colab.research.google.com/drive/1miQpnYAa9pZa-YWng1C7V78hM-nPR8Ly?usp=sharing

viz_results = apply_panel_of_manifold_learning_methods(X,color, methods_to_apply=['PCA','UMAP','TRIMAP','MDE','TSNE', 'LLE','MLLE','ISOMAP','MDS','SE', 'AUTOENCODER'])



MNIST dataset (downsampled to 2000 points)

PCA: 0.19 sec
LLE: 9.9 sec
Modified LLE: 11 sec
Isomap: 11 sec
MDS: 11 sec
SpectralEmbedding: 8.1 sec
t-SNE: 21 sec
UMAP: 9.8 sec
TRIMAP: 2.6 sec
MDE: 2.3 sec
Autoencoder: 34 sec

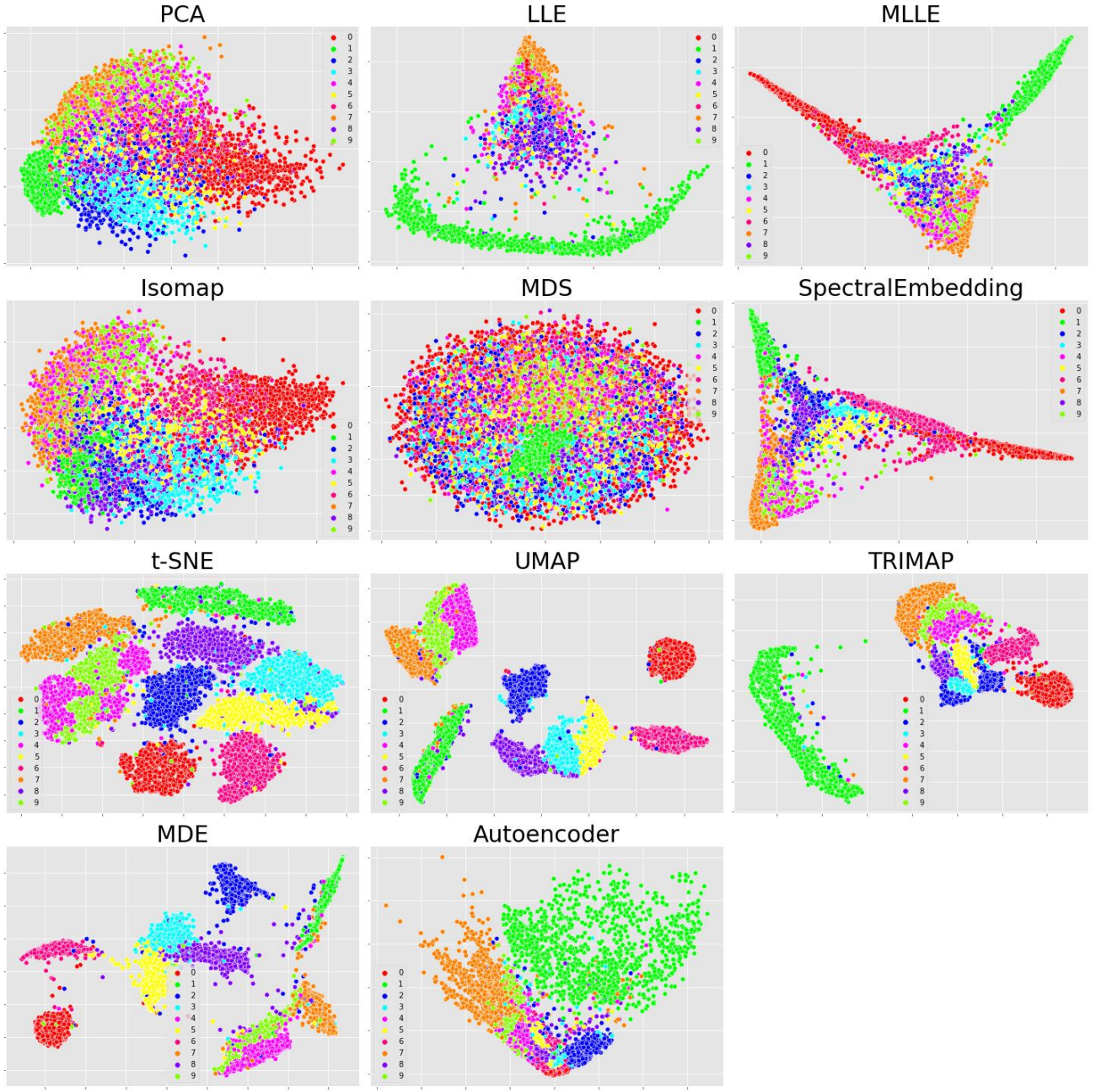
Comparison of (many) methods:

https://colab.research.google.com/drive/1miQpnYAa9pZa-YWng1C7V78hM-nPR8Ly?usp=sharing

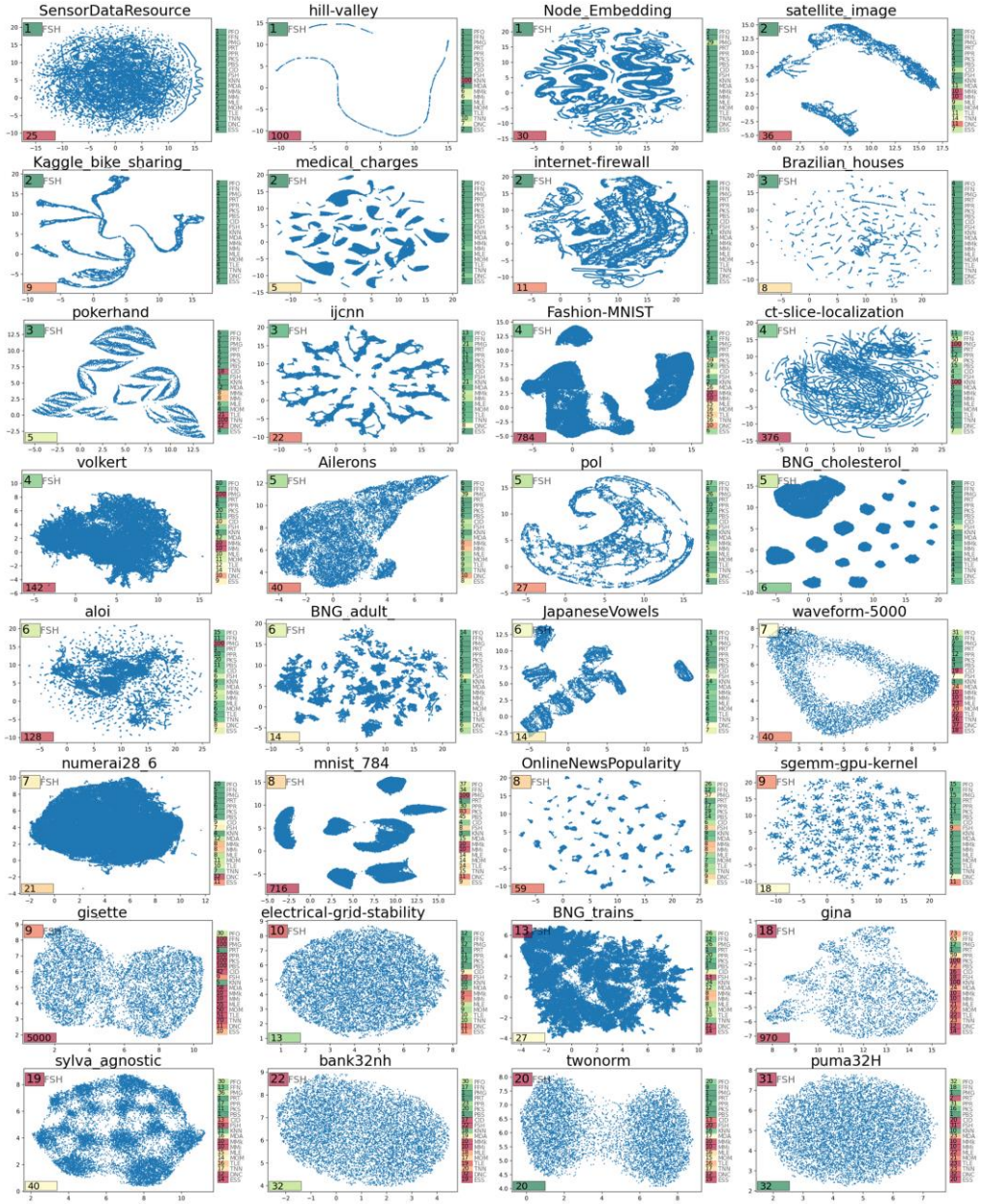
viz_results = apply_panel_of_manifold_learning_methods(X,color, methods_to_apply=['PCA','UMAP','TRIMAP','MDE','TSNE', 'LLE','MLLE','ISOMAP','MDS','SE', 'AUTOENCODER'])

MNIST dataset (downsampled to 2000 points)

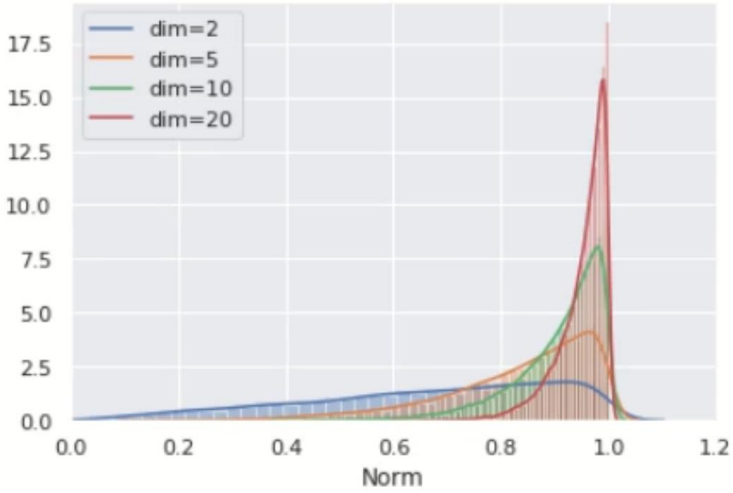
PCA: 0.82 sec
LLE: 260 sec
Modified LLE: 270 sec
Isomap: 280 sec
MDS: 240 sec
SpectralEmbedding: 202 sec
t-SNE: 250 sec
UMAP: 44 sec
TRIMAP: 12 sec
MDE: 15 sec
Autoencoder: 170 sec



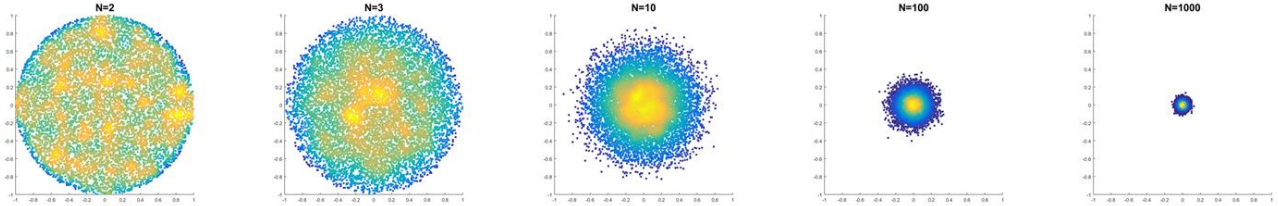
UMAP (or any other methods) become less informative in higher intrinsic dimensions



Some simple intuition for this



Uniformly sampled ball in R^N *observed in* R^2



Does not matter what distribution, it will look normal in any 2D or 3D projection (law of big numbers)