Data Visualisation and Exploration

COMS4060A/COMS7056A

Lecture 10

Dr Tim Bristow

Outline

- t-SNE
- UMAP
- Clustering vs Dimensionality Reduction

t-distributed Stochastic Neighbour Embedding (t-SNE)

Overview

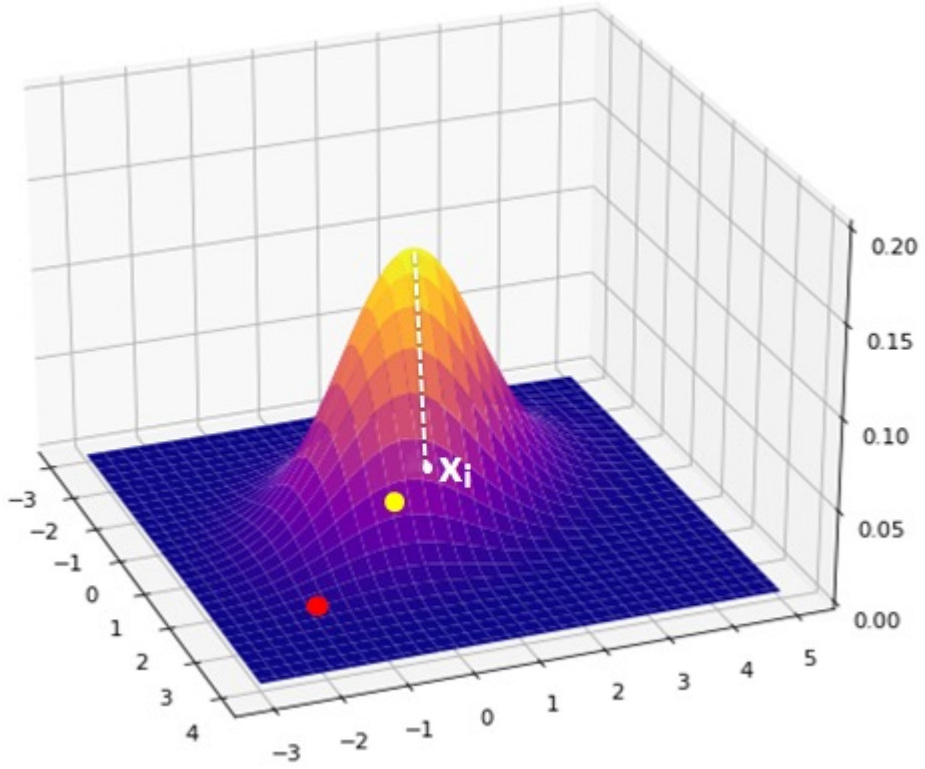
- T-SNE is a method for visualisation of non-linear high-dimensional data in 2 or 3 dimensions
- Compared with PCA, t-SNE preserves only small pairwise distances or local similarities, but PCA is preserves large pairwise distances to maximize variance
- t-SNE calculates a similarity between pairs of instances in both the high-D space and in the low-D space. It then tries to optimise these two similarity measures using a cost function
- It is technically a manifold learning algorithm that focuses on local structure, but not global structure
  - Specifically, it arranges the points in 2/3D so that if they are closely related in high-dim space, they are more likely to be neighbours: data points within a cluster are similar
  - Between cluster distances do not show an actual indication of similarity between the clusters
- T-SNE is an iterative method, unlike PCA, and cannot be used on new data without rebuilding the entire map
- It should only really be used for **visualisation, not learning an embedding** – if it is, it should be used with care
  - Within-cluster show highly related data points, but no information is provided to identify differences between separate cluster
  - The output is highly dependent on the parameters used for initialising the algorithm, and is not deterministic
  - See here for an in-depth discussion https://stats.stackexchange.com/questions/263539/clustering-on-the-output-of-t-sne
- Original paper: https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf
- Decent explanation: https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a

t-SNE algorithm

- Sklearn has a good overview:
  - *It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.*
- The joint probability between two points is defined as:

$$p_{j|i} = \frac{\exp(-|x_i - x_j|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-|x_i - x_k|^2 / 2\sigma_i^2)}$$

- This basically tells us how close x_i is to x_j , considering a Gaussian around x_i with a variance of σ_i^2. Each x_i has a different σ_i , so that there is a smaller variance in denser areas.



https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Perplexity

Intuitive explanation:
The perplexity parameter sets the effective number of neighbours that each point is attracted to. In the optimisation step of t-SNE, *all* pairs of points experience a repulsive force, with a small number of pairs experiencing an attractive force.

Small perplexity: fewer pairs that experience any attraction and the resulting embedding will tend to not be well separated: repulsive forces will dominate and will inflate the whole embedding to a bubble-like round shape.

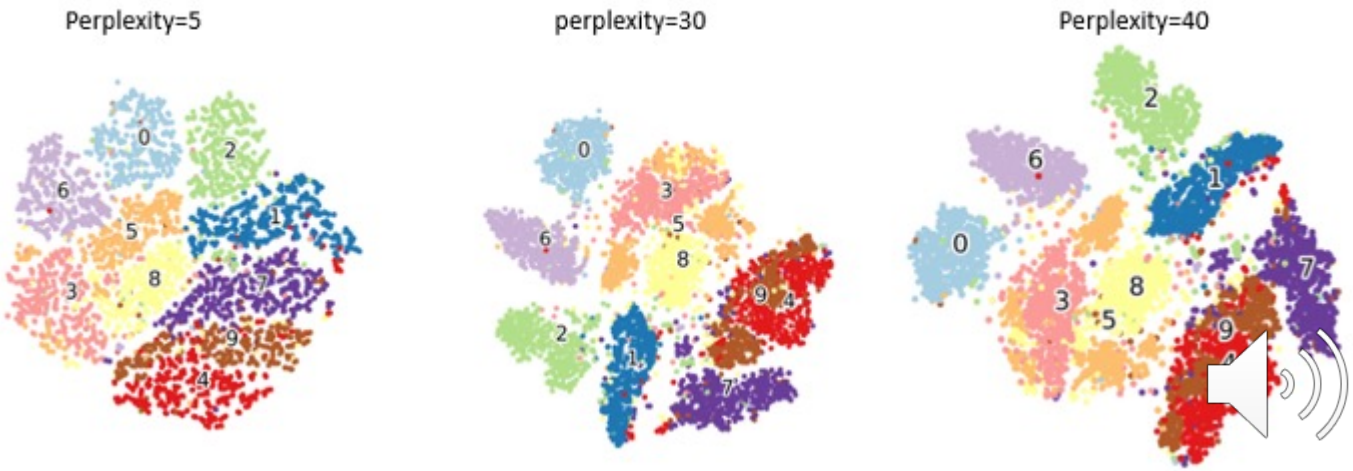
Large perplexity: clusters tend to shrink into denser structures.

The larger the perplexity, the more non-local information will be retained in the dimensionality reduction result.

To find σ_i a *perplexity* value must be chosen by the user:

$$Perp(p_i) = 2^{-\sum p_{j|i} \log_2 p_{j|i}}$$

- We want to find a low perplexity (5-50) that gives a good mapping, which is essentially a way of choosing the number of neighbours for each x_i, should be same for all points so no single point has more influence over the others
- A binary search is used on this to find a good value of σ_i, which is positively correlated with the perplexity



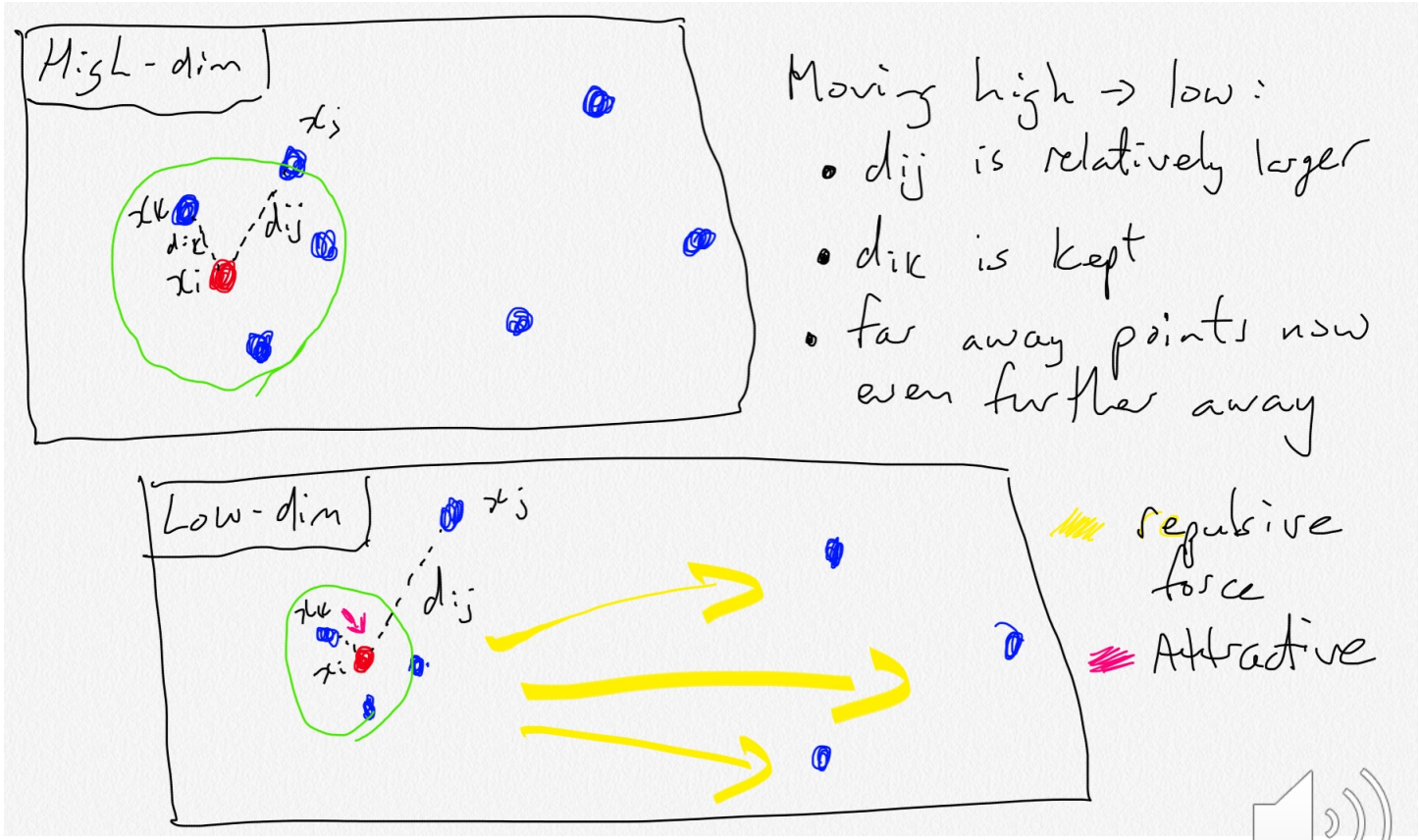
Similarity

- The similarity between x_i and x_j in the original, high-D space is given by: $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$

- The similarity between x_i and x_j in the low-D space is given by:

$$q_{ij} = \frac{f(|x_i - x_j|)}{\sum_{k \neq i} f(|x_i - x_k|)} \quad f(z) = \frac{1}{1 + z^2}$$

- The goal is to find the points in the low-D space such that q_{ij} matches the fixed p_{ij}



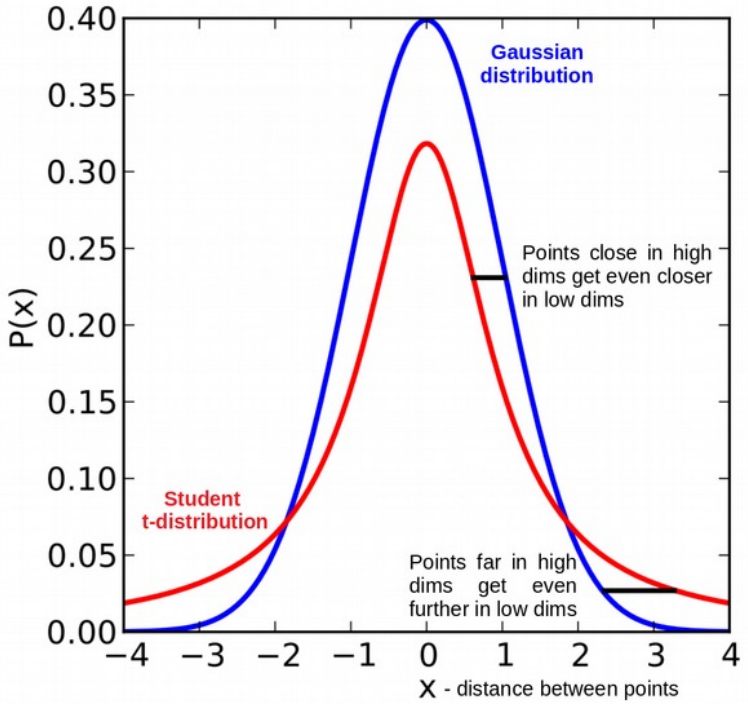
An important observation is that the similarity function in the low-D space is assumed to follow a t-distribution, not a Gaussian distribution, as it is in the high-D space

Also, note that Euclidean distance is used

Overcrowding

- The choice of a t-distribution ($f(z) = \frac{1}{1+z^2}$) for the similarity in low-D space is to avoid overcrowding in the latent space
- In particular, we want to be able to keep similar items close together, but make sure that dissimilar items are kept apart
  - If p_{ij} is large, then q_{ij} should also be large, and if p_{ij} is small, then q_{ij} should also be small
  - What about moderate distances?

Remember: we are moving from a high-D space to a low-D space, and there is less space for the data points to occupy in the lower dimension



- Why does a t-distribution help?
  - Moving from a Gaussian, similar points are forced closer together in a t-distribution, and further apart if dissimilar

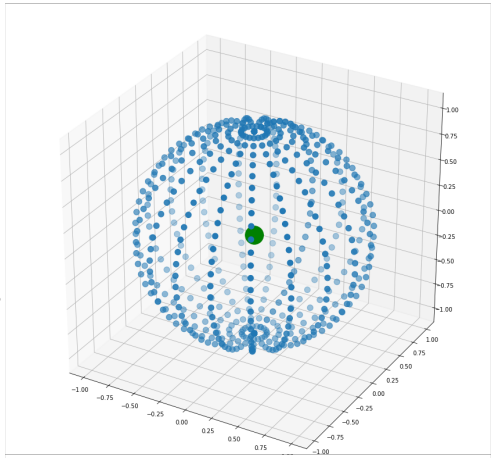
Overcrowding

Imagine the following simplified situation:

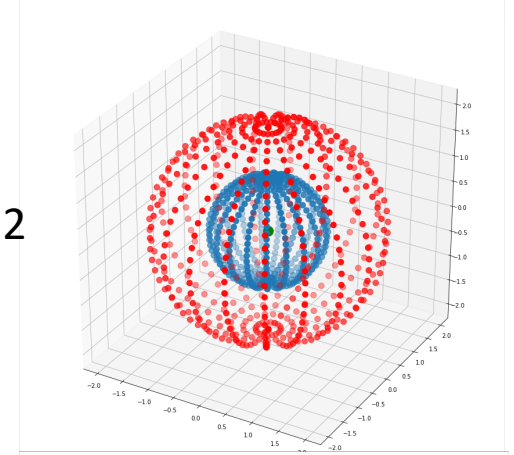
- Imagine a point X (green) in 3D that has a bunch of points all equidistant from it in the shape of a sphere (blue).
- Looking at a sphere around a point x, which has many other data points uniformly distributed around it
- The distance is the same for all of these, so moving to 2D we would have to have points overlapping!

- Now, we add 2 concentric circles around the original one: one distance (yellow), one moderate (red)
- When moving to 2D space, want to keep neighbours (blue) of X close, keep the moderate and far ones apart from each other
- If we demand that the points should follow the same (Gaussian) distribution in 2D, we find that moderately close points are not actually given that much space to be separated
- t-distributions have a longer tail, meaning these moderate data points are now allowed to be placed further away

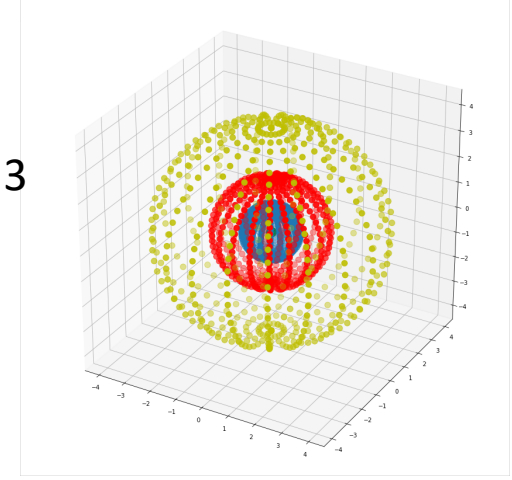
1



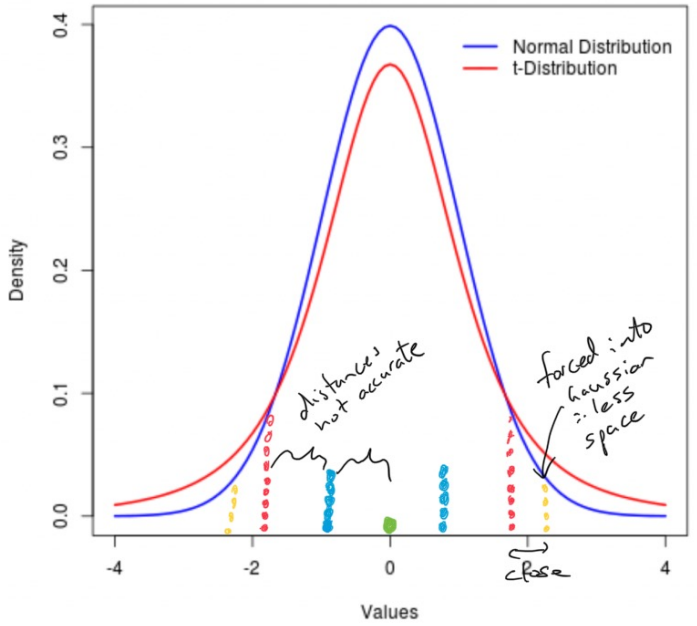
2



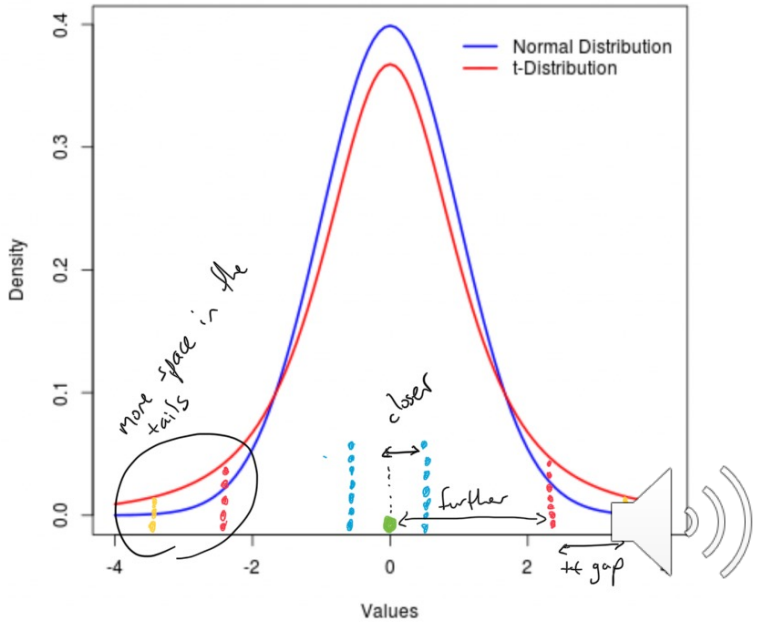
3



Using Gaussian similarity in low-D



Using t-distribution similarity in low-D



Minimisation of objective function

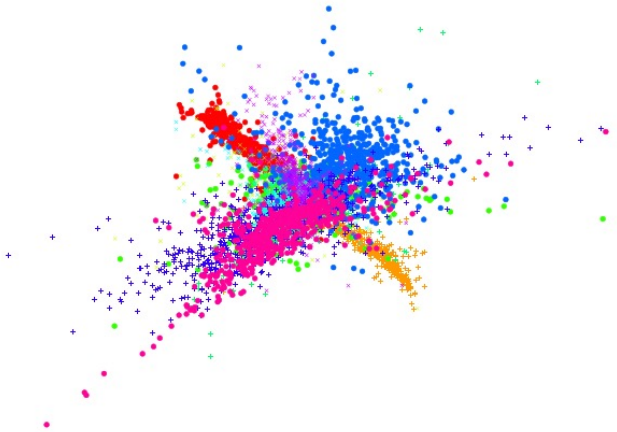
- Minimise *Kullback-Leibler divergence* (KL) using gradient descent
  - $KL(P|Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$
- Note that the objective function not the *sum* of KL divergences between high and low dimensional similarities, but a single KL divergence between p_{ij}, in the high-dimensional space and q_{ij} in the low-dimensional space
- If $p_{i|j}$ is large, and $q_{i|j}$ small, then large penalty for putting points that are nearby in high-D into low-D with a wide separation: good at preserving the local structure!
- If $p_{i|j}$ is small, then *any* $q_{i|j}$ will produce small penalties: points that are far away in high-D can be placed anywhere on the low-D map!

- Objective function involves computing both $p_{i|j}$ and $p_{j|i}$, and similarly in low-D too: optimisation is time consuming and complex

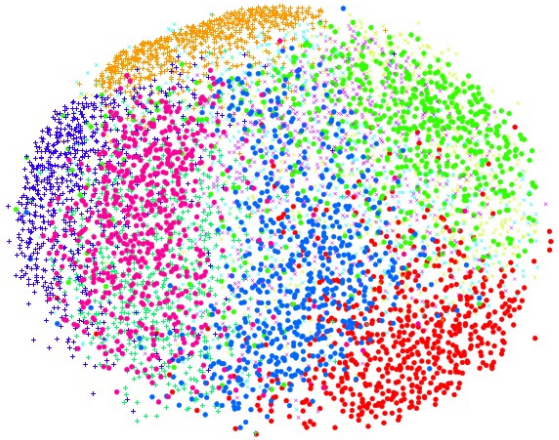
https://www.oreilly.com/content/wp-content/uploads/sites/2/2019/06/animation-94a2c1ff.gif
for an animated example of each step of the optimisation

MNIST comparison

- Here we see the performance of LLE, Sammon mapping, ISOMAP and t-SNE on the MNIST dataset

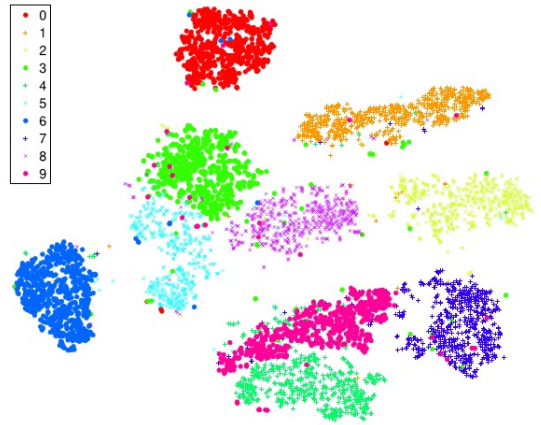


(b) Visualization by LLE.



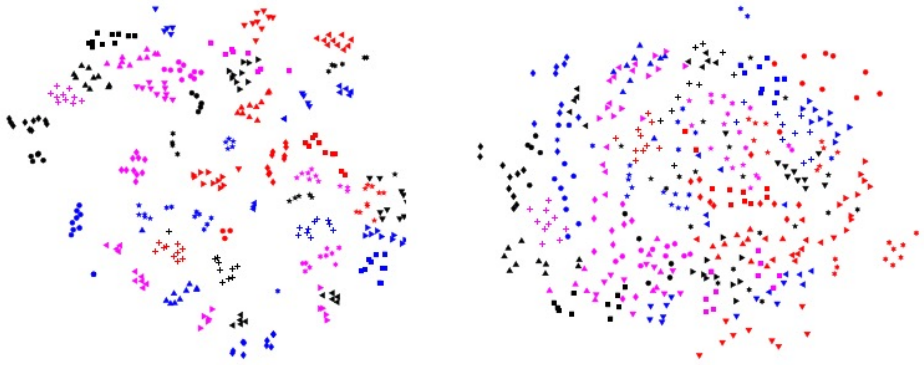
(b) Visualization by Sammon mapping.

(a) Visualization by Isomap.



* From the original paper

Olivetti faces comparison



(a) Visualization by t-SNE.

(b) Visualization by Sammon mapping.

(c) Visualization by Isomap.

(d) Visualization by LLE.

- Visualisation of the dataset using t-SNE, Sammon mapping, ISOMAP, LLE on the Olivetti faces dataset

* From the original paper

t-SNE summary

- Does well at separating clusters
- Works on non-linear data
- Provides good mapping to 2/3D, which is great for visualisations

- How about using t-SNE + PCA?

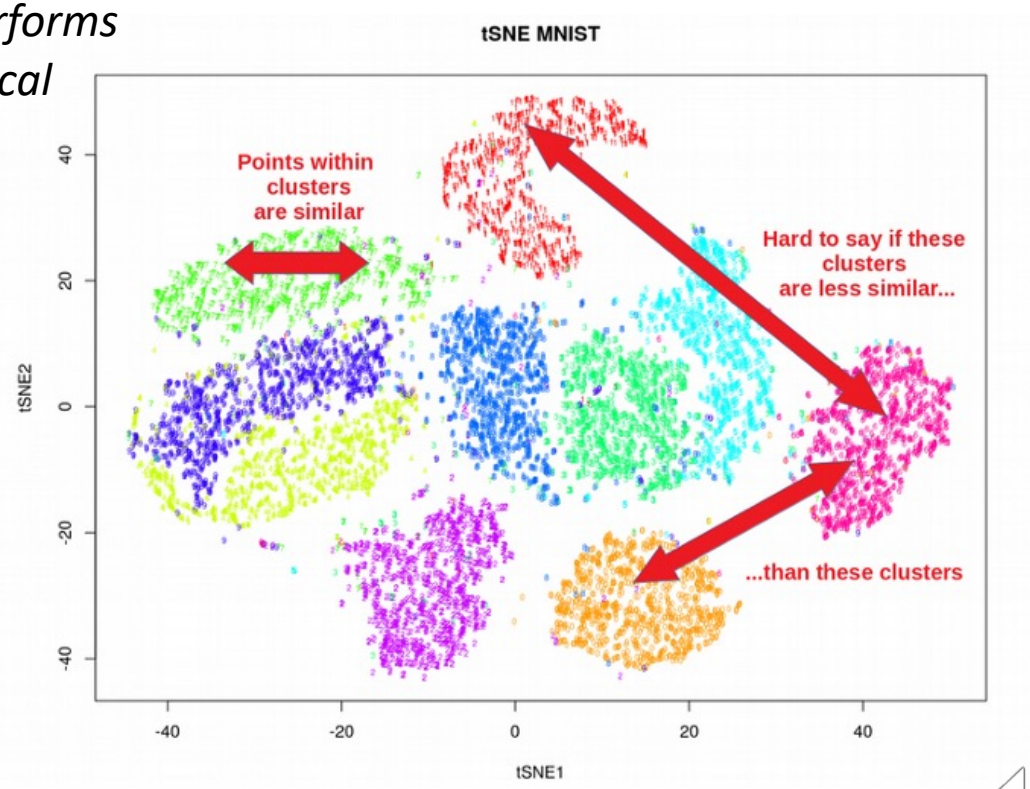
Olivetti: We see that it is able to identify individual clusters within the data, although they weren't separated from other clusters by that much… but in t-SNE, does that mean anything? We could probably obtain better separation by tweaking the perplexity.

MNIST: Much better separation of classes than the other methods tested! Surely this is what we've been looking for?!

t-SNE summary

From the original t-SNE paper: *Although we have shown that t-SNE compares favourably to other techniques for data visualization, t-SNE has three potential weaknesses: (1) it is unclear how t-SNE performs on general dimensionality reduction tasks, (2) the relatively local nature of t-SNE makes it sensitive to the curse of the intrinsic dimensionality of the data, and (3) t-SNE is not guaranteed to converge to a global optimum of its cost function.*

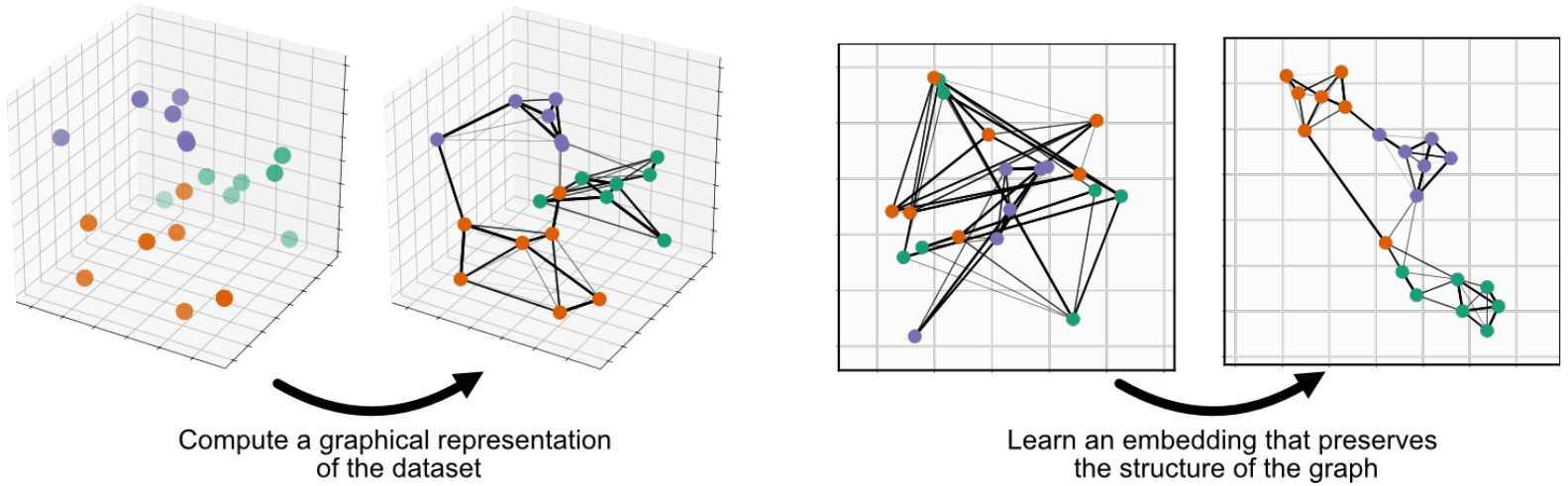
- Probabilistic
- Only focuses on local distances, interpretation of between-cluster distances is not possible
- Slow and does not scale
  - Recommended that PCA (or similar dimensionality reduction technique) is applied beforehand
- Iterative (can't reuse on new data)



Uniform Manifold Approximation and Projection (UMAP)

UMAP

- UMAP has been touted as a solution to the deficits of t-SNE
  - Much faster
  - Theoretically preserves more global structure (this is a contentious issue!)
    - https://www.biorxiv.org/content/10.1101/2019.12.19.877522v1 **UMAP does not preserve global structure any better than t-SNE when using the same initialization**
    - https://towardsdatascience.com/tsne-vs-umap-global-structure-4d8045acba17
  - Can run without additional pre-processing (PCA) on large datasets
  - Possible to add new data to the model without retraining
- At a high level, UMAP essentially has two main differences:
  - Perplexity is replaced with: nearest neighbours (number of expected nearest neighbours, similar to perplexity), minimum distance (how tightly points are packed which are close)
- Can be used for supervised embedding learning
- Good overview (but a bit biased) https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668



UMAP technical details

The join probability function for UMAP is given by $$p_{i|j} = \exp\frac{-d(x_i, x_j) - \rho_i}{\sigma_i}$$

- Where ρ_i is the distance between data point *i* and its nearest neighbour
- (Note that the distance function $d(x_i, x_j)$ is not always Euclidean)
- The joint probability function is not *normalised* like t-SNE (there is no summation in the denominator): faster!

The similarity is given by $p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}\, p_{j|i}$

- The number of neighbours is determined by: $k = 2^{\Sigma p_{ij}}$ (Shannon entropy)

Similarity in low-D is given by $$q_{ij} = \left(1 + a(y_i - y_j)^{2b}\right)^{-1}$$

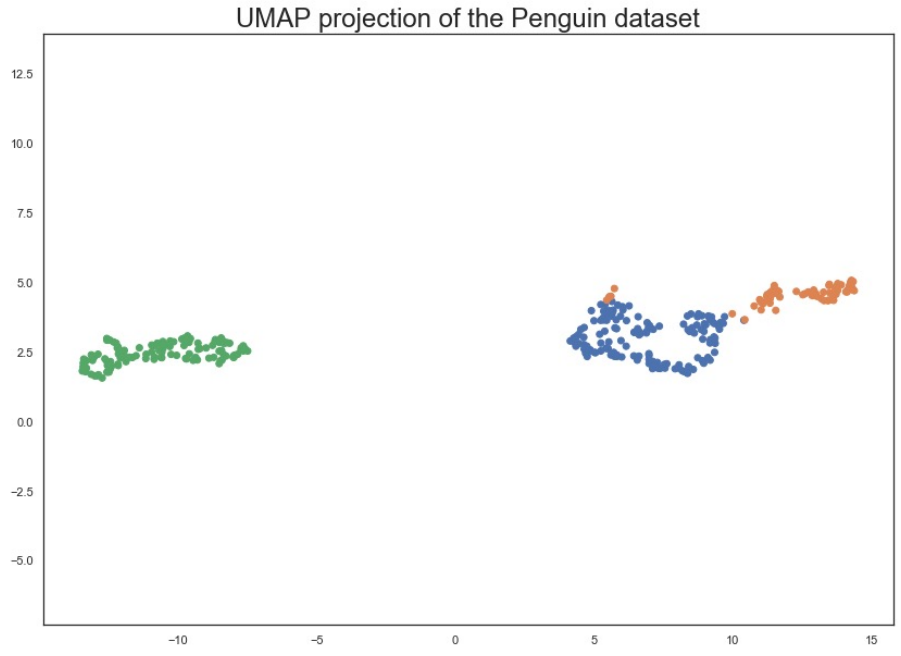
- The values of a and b are configurable, but generally they are determined by the value of min_dist

- Instead of using KL, cross-entropy is used as the objection function

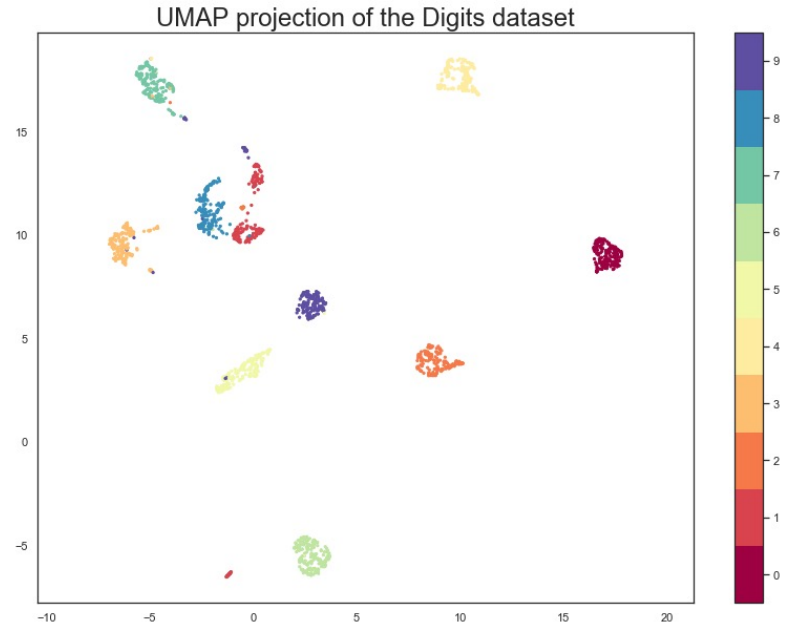
$$CE(\vec{X}, \vec{Y}) = \sum_i \sum_j \left[p_{ij}(\vec{X}) \log \frac{p_{ij}(\vec{X})}{q_{ij}(\vec{Y})} + (1 - p_{ij}(\vec{X})) \log \left(\frac{1 - p_{ij}(\vec{X})}{1 - q_{ij}(\vec{Y})} \right) \right.$$

UMAP examples

Provides a good separation of the different penguin species



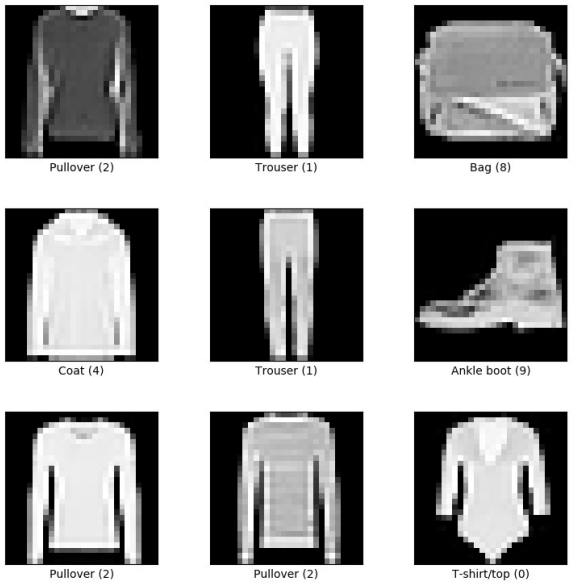
MNIST dataset clusters are tighter, and have a greater separation, than seen in t-SNE, Sammon's, LLE or ISOMAP (as seen a few slides ago)

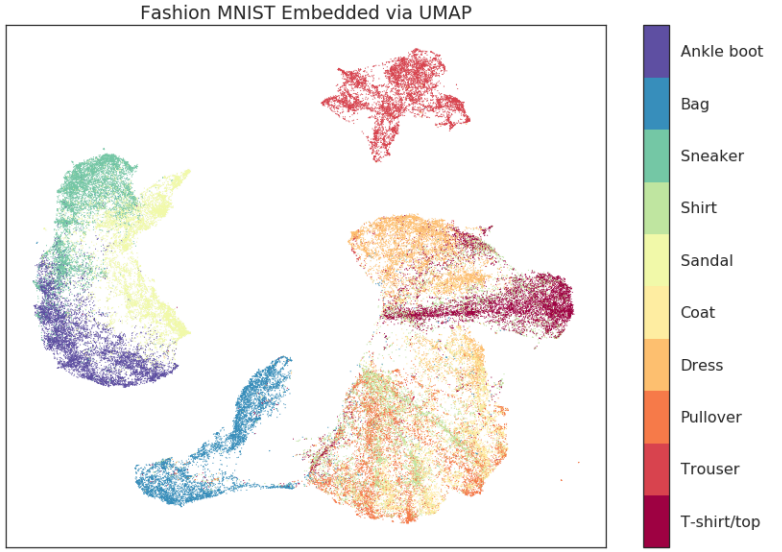


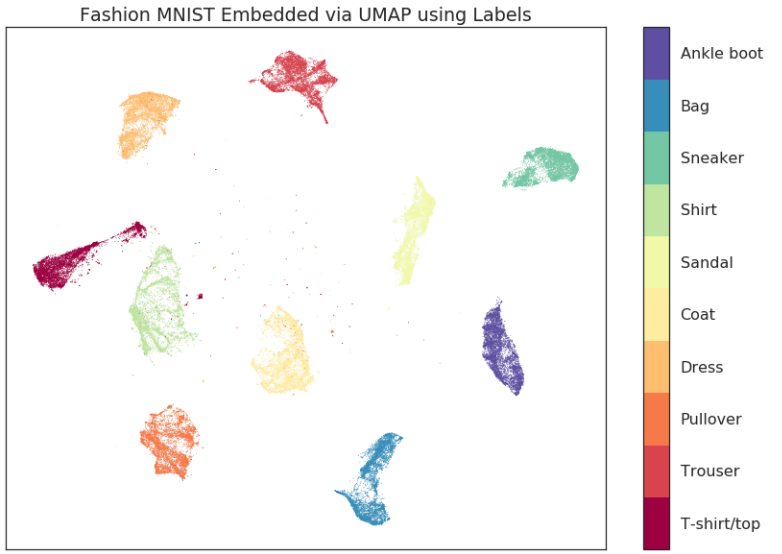
Supervised UMAP: Fashion MNIST

Here we can see the difference in clustering when it is supervised (ie we provide the fashion labels)

The clusters are well separated… but in general, if we're doing EDA on a given dataset we probably don't have labels! Encouraging to see that it still has decent embeddings ☺







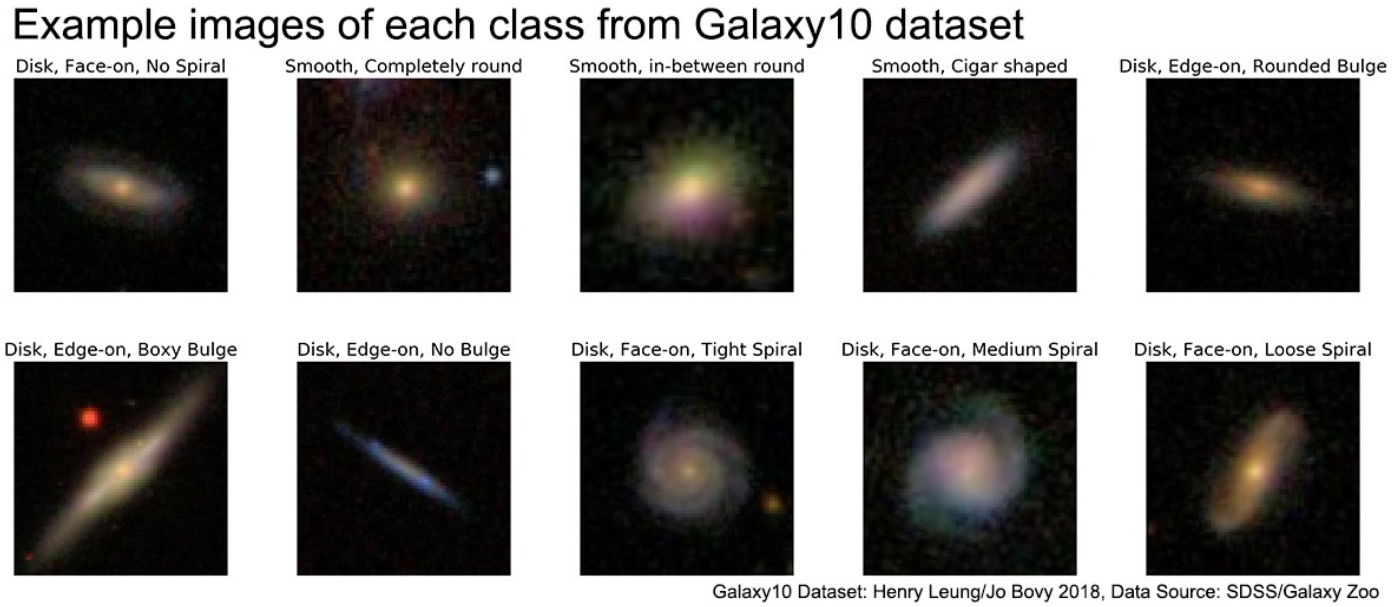
Another example

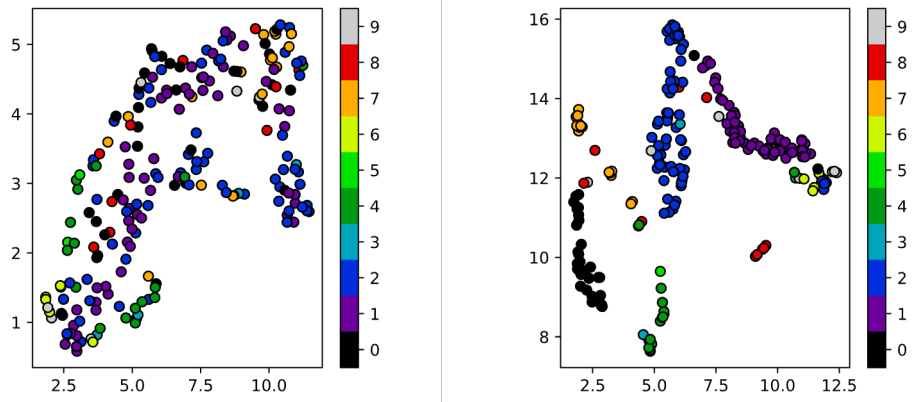
Galaxy10 SDSS images come from Sloan Digital Sky Survey and labels come from Galaxy Zoo. Galaxy Zoo relies on human volunteers to classify galaxy images and the volunteers do not agree on all imagesGalaxy10 only contains images for which more than 55% of the votes agree on the class.

Another example of using UMAP on the Galaxy10 dataset

Performing standard UMAP does not produce great results, although supervised clustering does…

Do we think that this is suitable for this dataset in general? Probably only if we have labelled data. Although we can apply this same embedding on new, unlabelled data – so it depends on your use case!



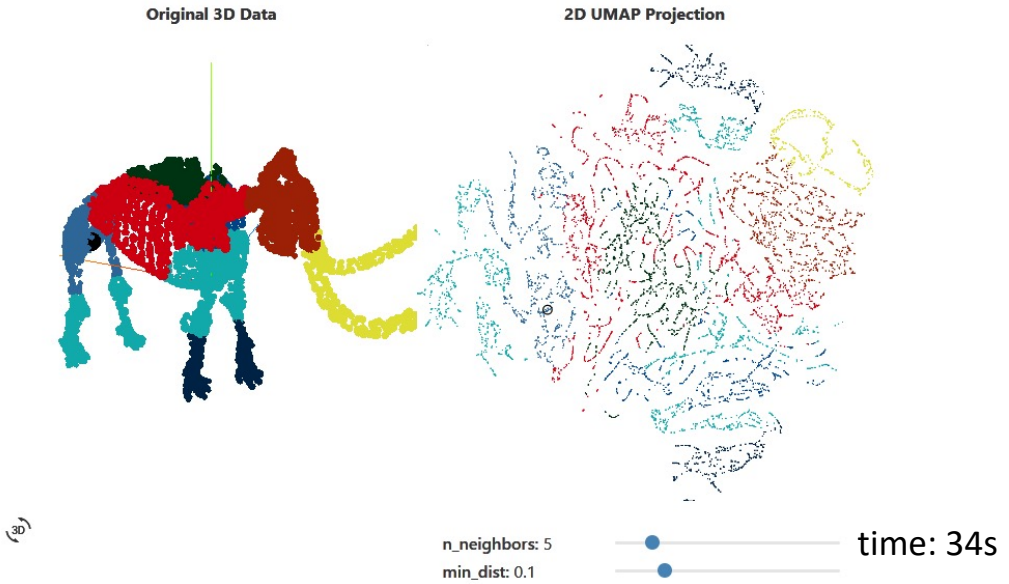


https://astronn.readthedocs.io/en/latest/galaxy10sdss.html
https://umap-learn.readthedocs.io/en/latest/supervised.html

UMAP parameters

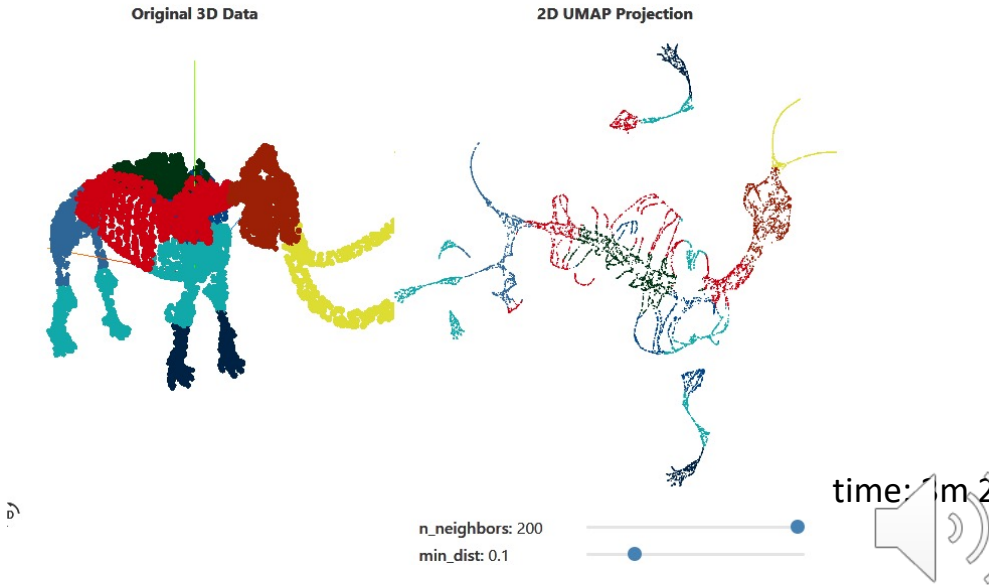
Here we see a good example of how UMAP can find an embedding of a 3D shape.

Notice how the performance changes when the number of neighbours and the min distance parameters are changed: UMAP is highly sensitive to the hyper-parameters



time: 34s

https://pair-code.github.io/understanding-umap/ Play with this example! It has slides that let you change the parameters and see how the results change



time: 1m 22s

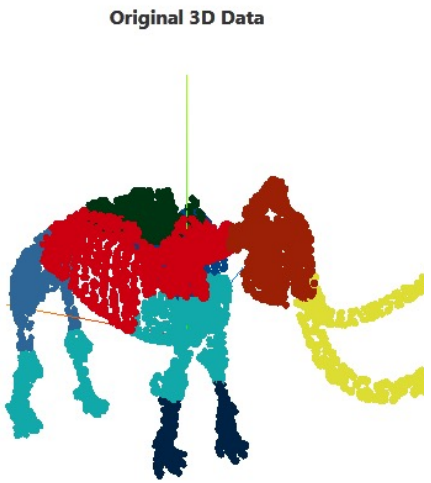
UMAP vs t-SNE

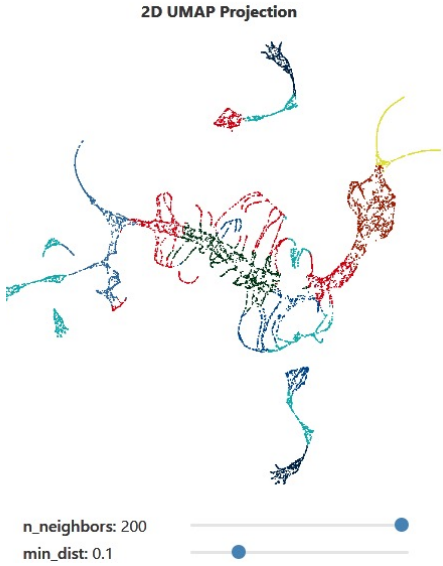
The link on the previous slide also has the option to view the t-SNE projections so you can compare the output

Notice how the t-SNE projection is unable to find an embedding even close to UMAP

Even as the perplexity is increased (ie increasing the number of neighbours and finding denser shapes) it is still not performing as well as UMAP

Another important observation is the time taken for computation: Orders of magnitude



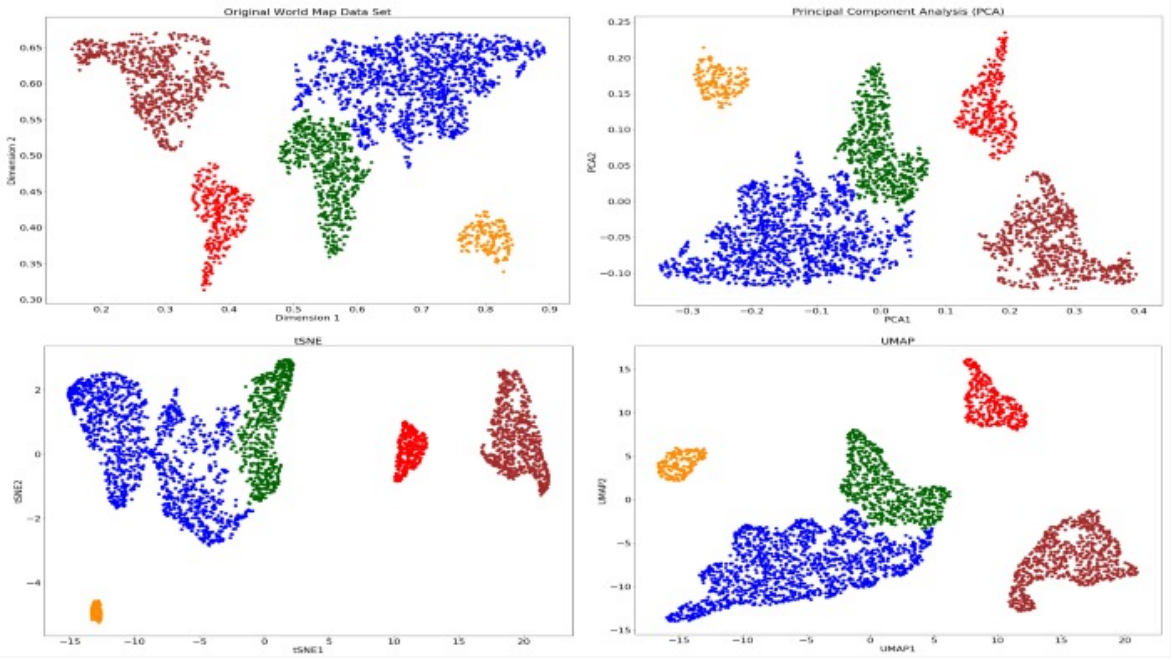


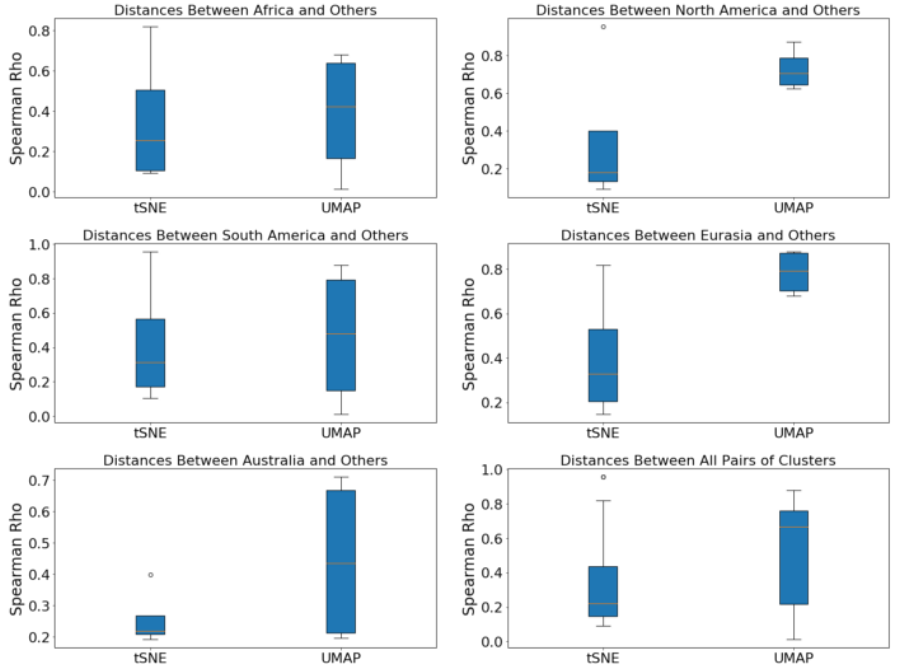
time: 3m 22s

World map example

Here we have an example of how PCA, t-SNE and UMAP project the world map

- PCA has the shapes right, just the image is flipped
- t-SNE … finds clusters, but they have become condensed (notice how small Australia is)
- UMAP better preserves the shapes than t-SNE, and the distances between continents is more accurate





UMAP

- In summary, 2D dimension reduction with t-SNE is, in general, too extreme, and too slow
  - Option to perform PCA (or similar) and then use t-SNE on the output, but need to make sure this captures enough info!
- UMAP provides an improvement: faster, preserves global distances too
  - Could perform dimensionality reduction (to > 3D) using UMAP, and then cluster on this output (using t-SNE or other). This gives a compromise between 2D t-SNE, or clustering on non-optimal dimension reduction (too linear or still too high-dimensional)

- **Successful use-cases**
  - UMAP can be / has been successfully applied to the following domains:
  - Single cell data visualization in biology;
  - Mapping malware based on behavioural data;
  - Pre-processing phrase vectors for clustering;
  - Pre-processing image embeddings (Inception) for clustering;
- Some gotchas:
  - Data points that are maximally far apart from all other points in your data set: UMAP's assumption of all points lying on a connected manifold may not be good.
    - From this point's perspective all other points are equally valid nearest neighbours so its k-nearest neighbour query will return a random selection of neighbours all at this maximal distance. Next we will normalize this distance by applying our UMAP kernel which says that a point should be maximally similar to it's nearest neighbour. Since all k-nearest neighbours are identically far apart they will all be considered maximally similar by our point in question. When we try to embed our data into a low dimensional space our optimization will attempt to pull all these randomly selected points together. Add a sufficiently large number of these points and our entire space gets pulled together destroying any of the structure we had hoped to identify. (https://towardsdatascience.com/tsne-vs-umap-global-structure-4d8045acba17)
- Python implementation https://pypi.org/project/umap-learn/

https://umap-learn.readthedocs.io/en/latest/exploratory_analysis.html Some cool examples here!

UMAP implementation details

Hyperparameters matter

- Choosing good parameters for n_neighbours and min_dist depends on both the data and your intended outcome (eg, how tightly packed the projection should be). UMAP being fast means you are able to do a grid search (or closer to it) over many options, unlike t-SNE.

Cluster sizes in a UMAP plot mean nothing

- Like t-SNE, the size of clusters relative to each other is essentially meaningless. UMAP uses local notions of distance to construct its high-dimensional graph representation.

Distances between clusters might not mean anything

- Distances between clusters is usually meaningless too. The global *positions* of clusters are better preserved, but the distances *between* them are not meaningful.

Random noise doesn't always look random.

- Especially at low values of n_neighbors, spurious clustering can be observed.

You may need more than one plot

- Since UMAP is stochastic (random), different runs with the same hyperparameters can give different results. Run the projection multiple times with the same hyperparameters to ensure stability

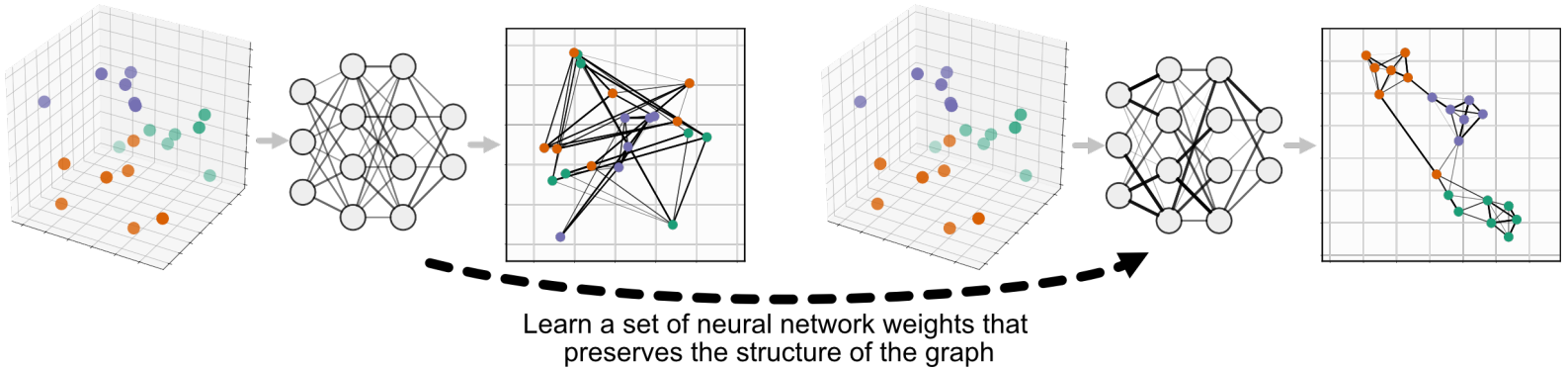
Enhanced UMAP

- The idea here is that one can go to n-dimensions, not just 2 or 3
- UMAP is now being used to reduce the dimensionality of the dataset and essentially find the manifold itself, as we have seen from our other manifold algorithms
- From this step we performing density based clustering

- My advice: do not cluster on it… **or do so with care** https://stats.stackexchange.com/questions/263539/clustering-on-the-output-of-t-sne
  - Depends on what clustering algorithm you are going to use: UMAP does not necessarily produce spherical clusters, so clustering with, for example, K-Means, is a poor choice. Using a density-based option like DBSCAN might work better, *however*, UMAP, with its uniform density assumption, does not preserve density well. It will find keep connected components of the manifold together. With enough data, UMAP can do this, and you can get *useful* clustering results out.

Autoencoder + UMAP

- Autoencoders can be used to find an embedding of the high-D dataset, but autoencoders have no inherent knowledge of the local structure of the data
  - Original paper: https://arxiv.org/abs/2009.12981 Note that on this page there is a link to a Google Colab notebook
  - I have been playing around with some code that does this… See attached
- Idea: find structure/manifold in the autoencoder's embedding
  - Perform UMAP on the output of the bottleneck layer of the autoencoder, but keep enough UMAP components to keep enough of the information (so, typically > 3)
    - Double dimensionality reduction!
  - Now, if we want to visualise the output, we perform clustering on the UMAP space



https://minimatech.org/autoencoder-with-manifold-learning-for-clustering-in-python/ Another example

General comments on clustering and dimensionality reduction

We are given some high-dimensional dataset, which might be linear or non-linear

Dimensionality reduction:

- Reduce the dimensionality of the data by performing PCA, or manifold learning: this is an attempt to move the data to a much lower dim space whilst preserving the distances between data points and faithfully representing the original data structure

Clustering:

- Find clusters directly in the high-D data: this does not actually reduce our dimensions, it just finds similar groupings

Dimensionality reduction + clustering

- Perform dimensionality reduction (using any method that is appropriate)
- Cluster on this manifold and find similar groupings (this is what Spectral Clustering does)
- We could perform this multiple times:
    - perform dimensionality reduction to find some embedding
    - Use another method to reduce the dimensionality of this embedding
    - Cluster on the output of the second embedding
    - This will introduce lots of scope for errors…
    - E.g. See previous page.

Visualisation only:

- T-SNE is a good example of a method that is good at visualisations: by moving to 2 or 3 dimensions, we are likely losing a lot of information about the structure of the data, especially if it is complex. However, it is still able to point out similar points in a lower dimension by using a Euclidean distance metric
- UMAP is similar to t-SNE, but uses a different distance metric, is faster