FitchLearning

Unsupervised Machine Learning
Part 2

Panos Parpas

Slides prepared by:
Dr. Claus Huber, CEFA, CFA, FRM
Head of Quantitative Modelling & Analytics
Helvetia Insurance, Basel

May 2023

Applications of Unsupervised Learning in Finance

- This module “Unsupervised Machine Learning 2” discusses more recent methods:
  - t-SNE
  - UMAP
  - Autoencoders
- Applications are in (examples):
  - Macro Forecasting
  - Portfolio Management
  - Asset Allocation
  - Trading (e.g., relative value)
  - Natural Language Processing
  - Visual Risk Analysis (e.g., return-based style classification of hedge funds)

t-SNE: t-Distributed Stochastic Neighbour Embedding

- t-SNE is a visualisation method proposed by van der Maaten / Hinton (2008)
  - Stochastic → not definite but random probability
  - Neighbour → concerned only about retaining the variance of neighbour points
  - Embedding → plotting data in lower dimensions
- For example, PCA works on retaining **global** variance while t-SNE focuses on retaining **local** variance
- Suitable for non-linear (curved manifolds, see next slide), high-dimensional, large data sets
- t-SNE is designed for data visualisation
- t-SNE is extensively applied in image processing, natural language processing, genomic data and speech processing

Non-Linear, Curved Manifolds

- What is a non-linear, curved manifold and how can t-SNE help to better understand its structure?
- The structure of the high dimensional data space in practice will be less clear than in this example

This is what we have: high dimensional (here: 3D) data

This is what we want: a 2-dimensional representation

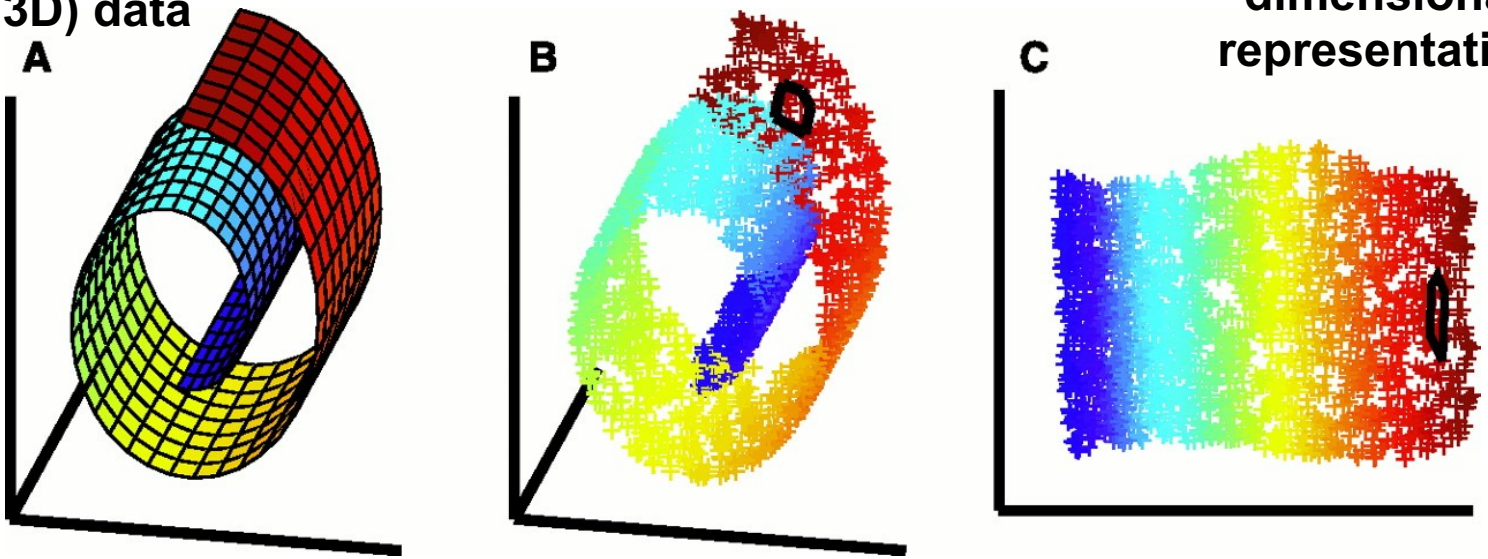
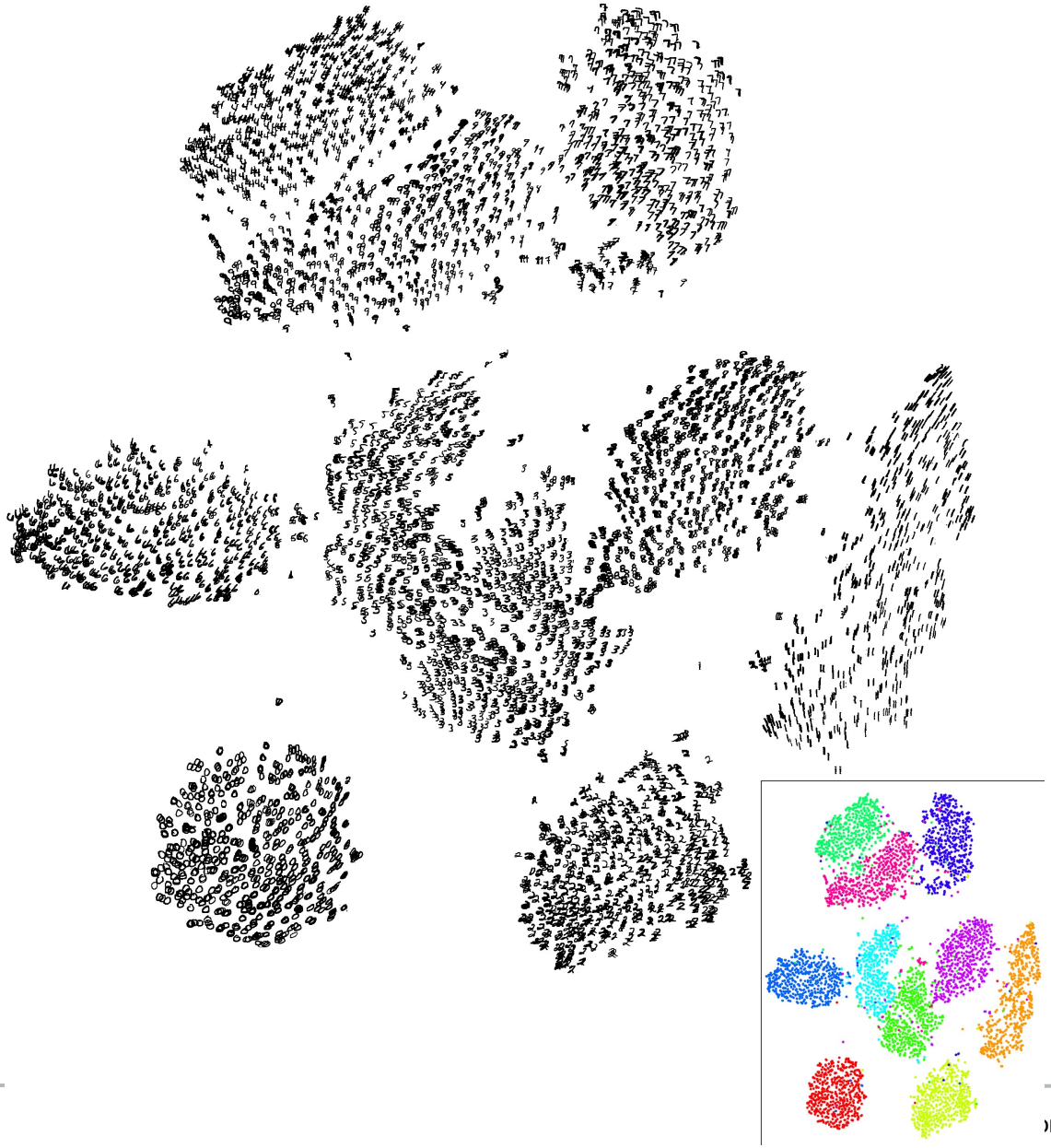


Image from: https://science.sciencemag.org/content/sci/290/5500/2323/F1.large.jpg

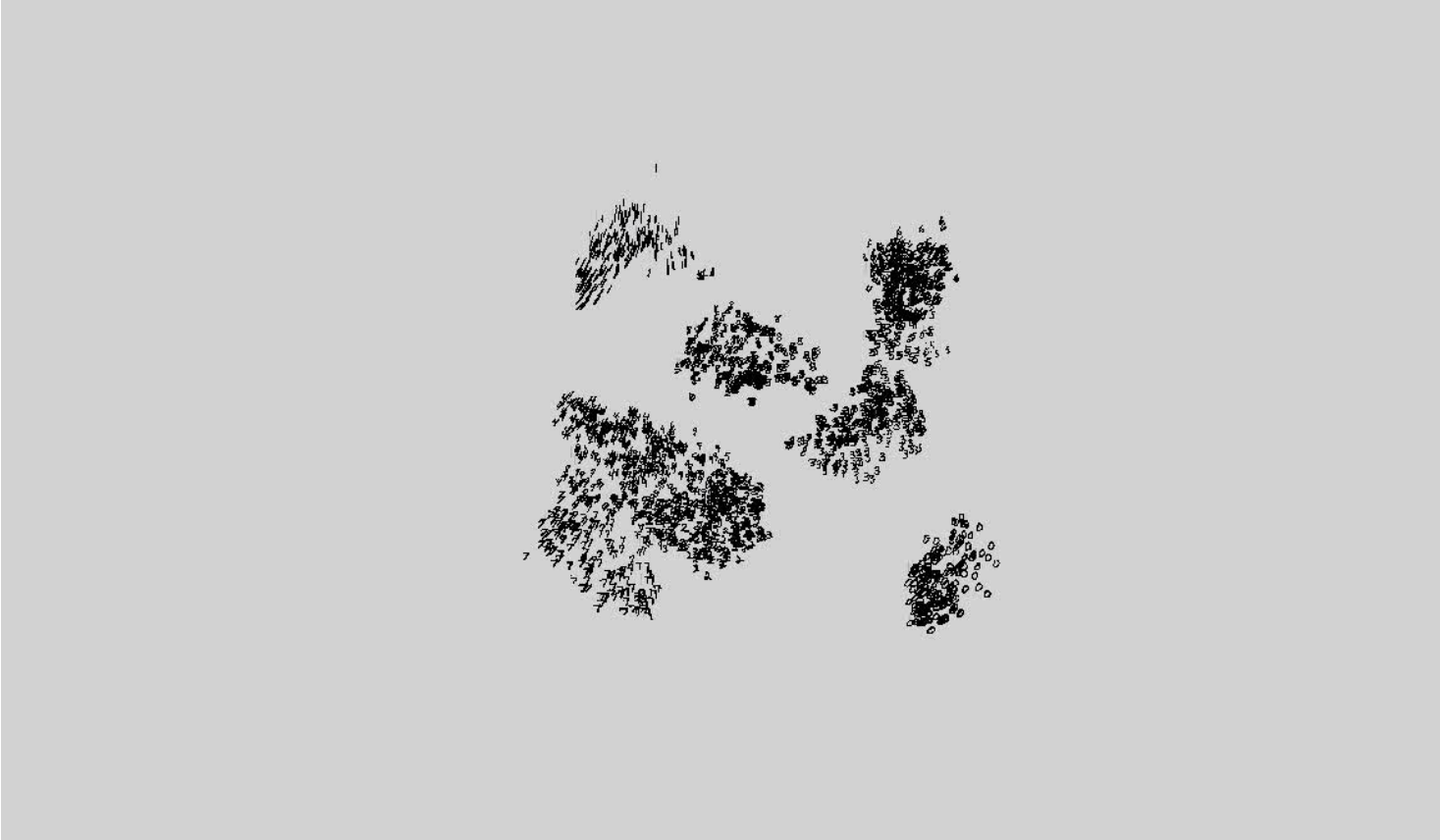
Example MNIST

Image Classification. Each image is in 28x28=784 dimensions

MNIST with t-sne in 2d



MNIST with t-sne in 3d

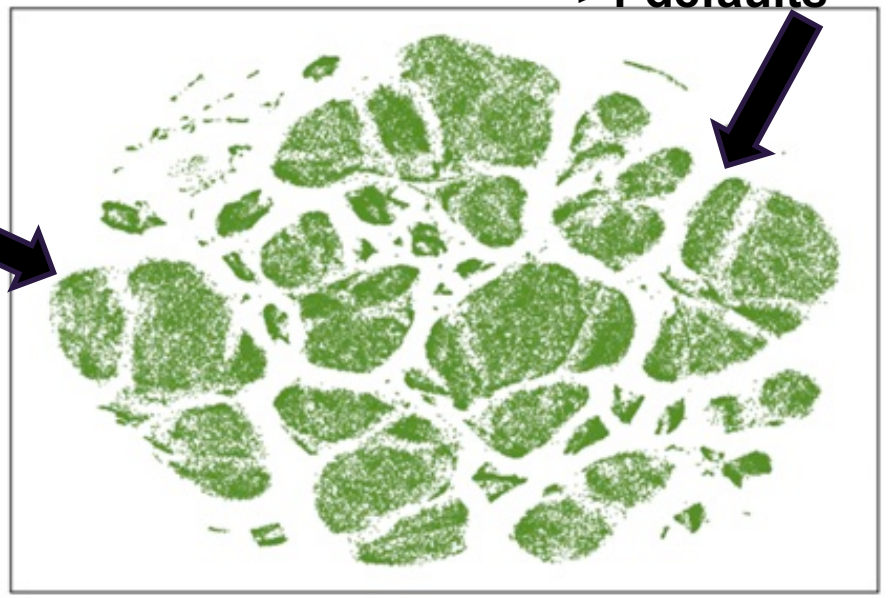


Applications of t-SNE in Finance

- Efimov et al. (2020) apply Generative Adversarial Networks to generate artificial datasets for credit card data
  - t-SNE deployed to visualise the data
- Data includes, for example, net income, number of defaults in the last 10 years, debt level, etc.
- 2 mln data points, 471 features

High Income, low debt level, no defaults

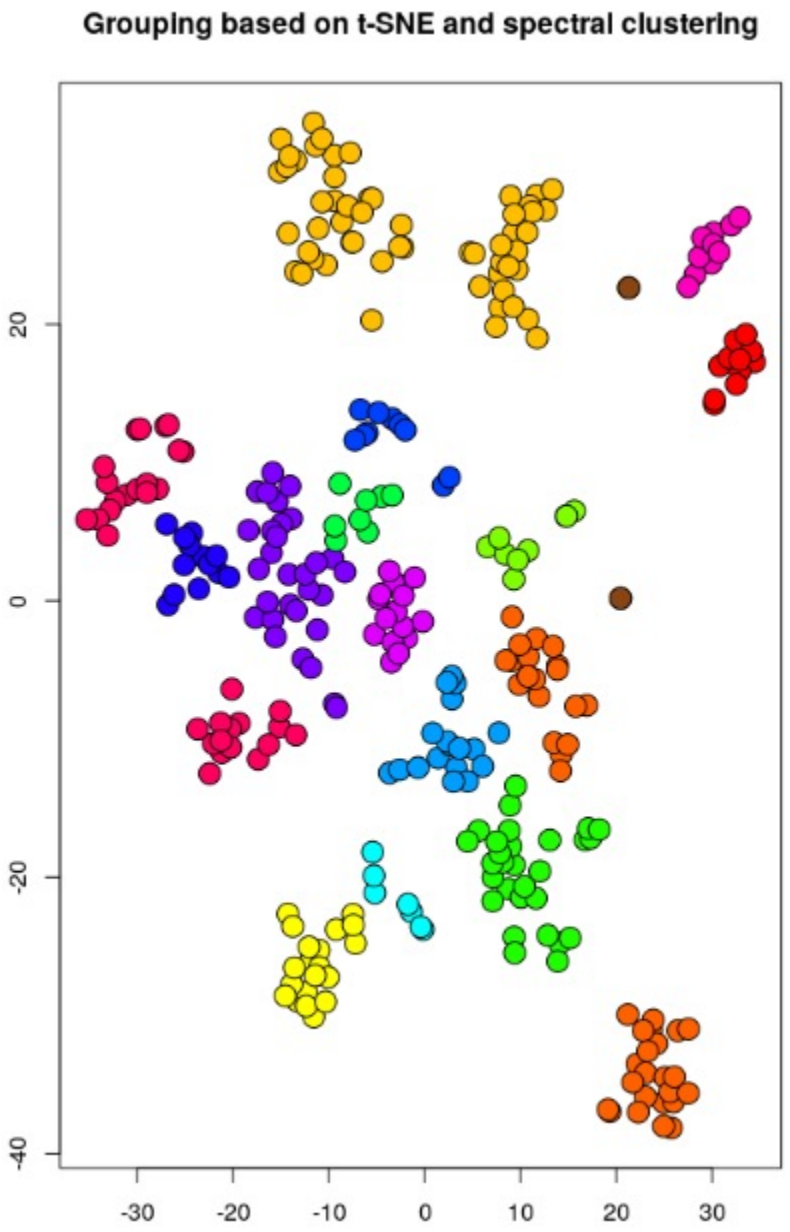
Low Income, high debt level, >1 defaults



(a) Real data

Applications of t-SNE in Finance

- Husmann et al. (2020):
- Stocks are combined in groups with t-SNE in combination with another method, spectral clustering
- Data set comprises return data, balance-sheet data, social responsibility measures, etc.
- The stocks in the individual groups are equal-weighted based on their t-SNE clustering in 2 to 20 groups
- Each group contains similar stocks
- **Markowitz optimisation on the groups is used to determine the optimal portfolio**
- Goal is to reduce noise in the optimisation process by reducing the number of input parameters (→ addressing the curse of dimensionality)



Use Case: Factor Clustering with t-SNE

- Greengard et al. (2020) compose 55 equity long-short portfolios
- Each portfolio represents 1 risk factor, e.g., Fama-French SMB (=Size)
  - Daily returns 03/1975 to 12/2019, ca. 11,000 data points for each risk factor
- Goal is to study similarities between the risk factors

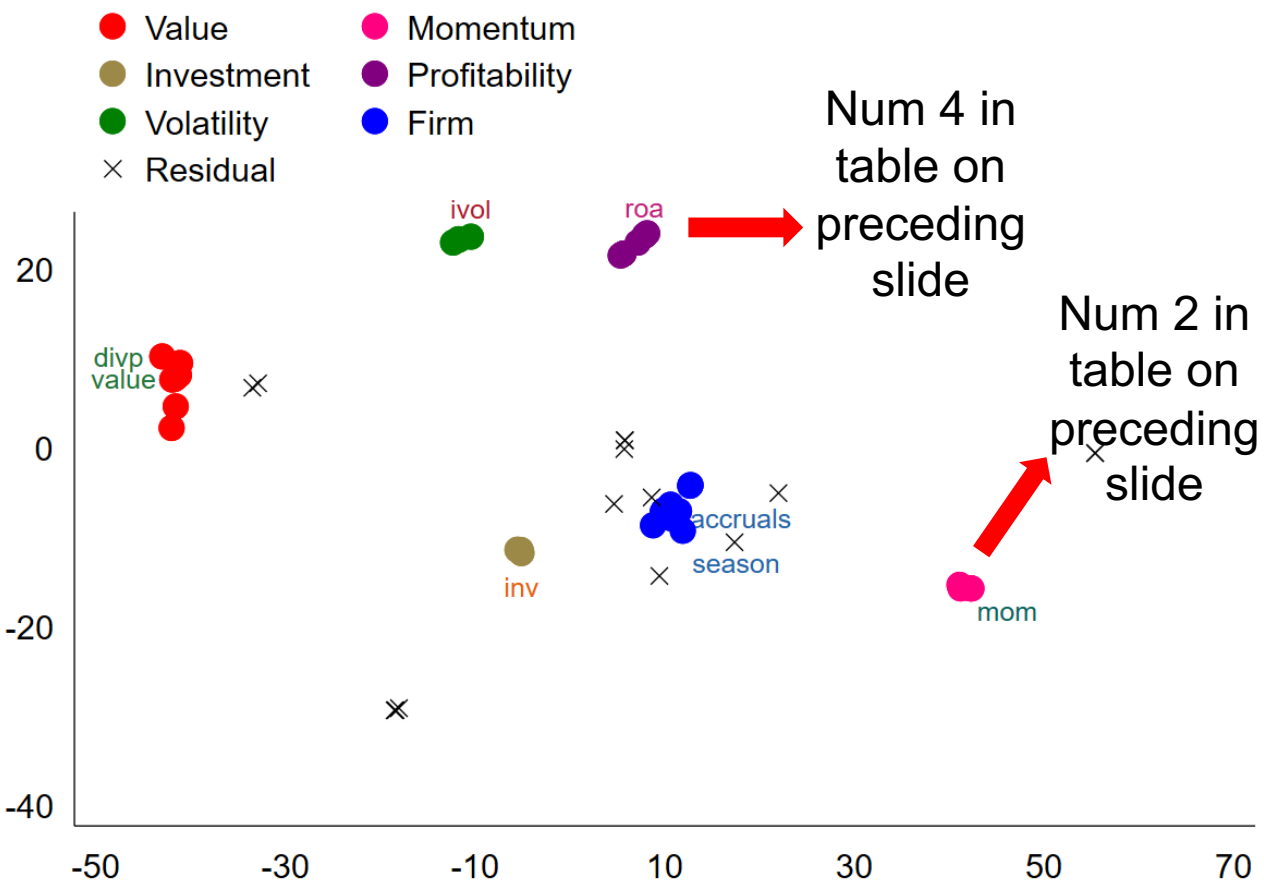
Table 1: Strategy Description

This table reports the summary statistics of the 55 long-short strategies in Kozak, Nagel, and Santosh (2020). The average strategy returns and the CAPM alphas are reported. The numbers correspond to the t-SNE group numbers. *, **, *** denote significance levels at the 10%, 5%, and 1% based on the standard t-statistics.

Num	Strategy		$\overline{Ret}$	α	Num	Strategy		$\overline{Ret}$	α
1	Cfp	Cash Flow-Market Value	0.016**	0.021***	5	Shvol	Share Volume	-0.005	0.021**
1	Divp	Dividend Yield	0.010	0.020**	6	Accruals	Accruals	0.018***	0.016***
1	Dur	Cash Flow Duration	0.019**	0.021**	6	Aturnover	Asset Turnover	0.020***	0.021***
1	Lev	Leverage	0.009	0.005	6	Debtiss	Debt Issuance	0.007**	0.006*
1	Lrrev	Long-term Reversals	0.007	0.010	6	Divg	Dividend Growth	0.001	-0.000
1	Momrev	Momentum-Reversal	0.022**	0.025***	6	Fscore	Piotroski's F-score	0.003	0.005*
1	Sp	Sales-Price	0.021***	0.025***	6	Gltnoa	Growth in LTNOA	0.001	0.002
1	Value	Book-to-Market	0.017**	0.021**	6	Gmargins	Gross Margins	0.006	0.006
1	Valuem	Book-to-Market (monthly)	0.011	0.012	6	Noa	Net Operating Assets	0.024***	0.021***
2	Indmom	Industry Momentum	0.013	0.019	6	Prof	Gross Profitability	0.019***	0.016**
2	Mom	Momentum (6m)	0.009	0.012	6	Repurch	Share Repurchases	0.005	0.007**
2	Mom12	Momentum (1 year)	0.055***	0.059***	6	Season	Seasonality	0.033***	0.030***
2	Valmom	Value-Momentum	0.015*	0.020**	7	Age	Firm Age	-0.000	-0.007
2	Valmomprof	Value-Momentum-Profitability	0.032***	0.035***	7	Ciss	Composite Issuance	0.026***	0.034***
3	Growth	Asset Growth	0.012*	0.017**	7	Exchsw	Exchange Switch	0.007	0.008
3	Igrowth	Investment Growth	0.011*	0.016***	7	Indmomrev	Industry Momentum-Reversal	0.039***	0.040***
3	Inv	Investment	0.017**	0.022***	7	Indrrev	Industry Relative Reversals	0.034***	0.029***
3	Sgrowth	Sales Growth	-0.004	0.002	7	Indrrevlv	Industry Relative Reversals (low vol)	0.047***	0.044***
4	Ep	Earnings-Price	0.028***	0.035***	7	Invaci	Abnormal Corporate Investment	0.004	0.003
4	Price	Price	0.014	0.017*	7	Ipo	Initial Public Offering	-0.005	-0.011
4	Roa	Return to Assets (m)	0.026***	0.032***	7	Nissa	Share Issuance (a)	0.031***	0.038***
4	Roaa	Return to Assets (a)	0.015**	0.022***	7	Nissm	Share Issuance (m)	0.025***	0.033***
4	Roe	Return to Book Equtiy (m)	0.013	0.022***	7	Shortint	Short Interest	-0.006	0.003
4	Roea	Return to Book Equity (a)	0.030***	0.039***	7	Size	Size	0.003	0.014**
4	Rome	Return to Market Equity	0.052***	0.062***	7	Strev	Short-term Reversal	0.014	0.006
5	Betaarb	Beta Arbitrage	0.001	0.038***	7	Sue	PEAD (SUE)	0.023***	0.025***
5	Invcap	Investment-Capital	0.006	0.021**	7	Valprof	Value-Profitability	0.030***	0.036***
5	Ivol	Idiosyncratic Volatility	0.032***	0.052***					

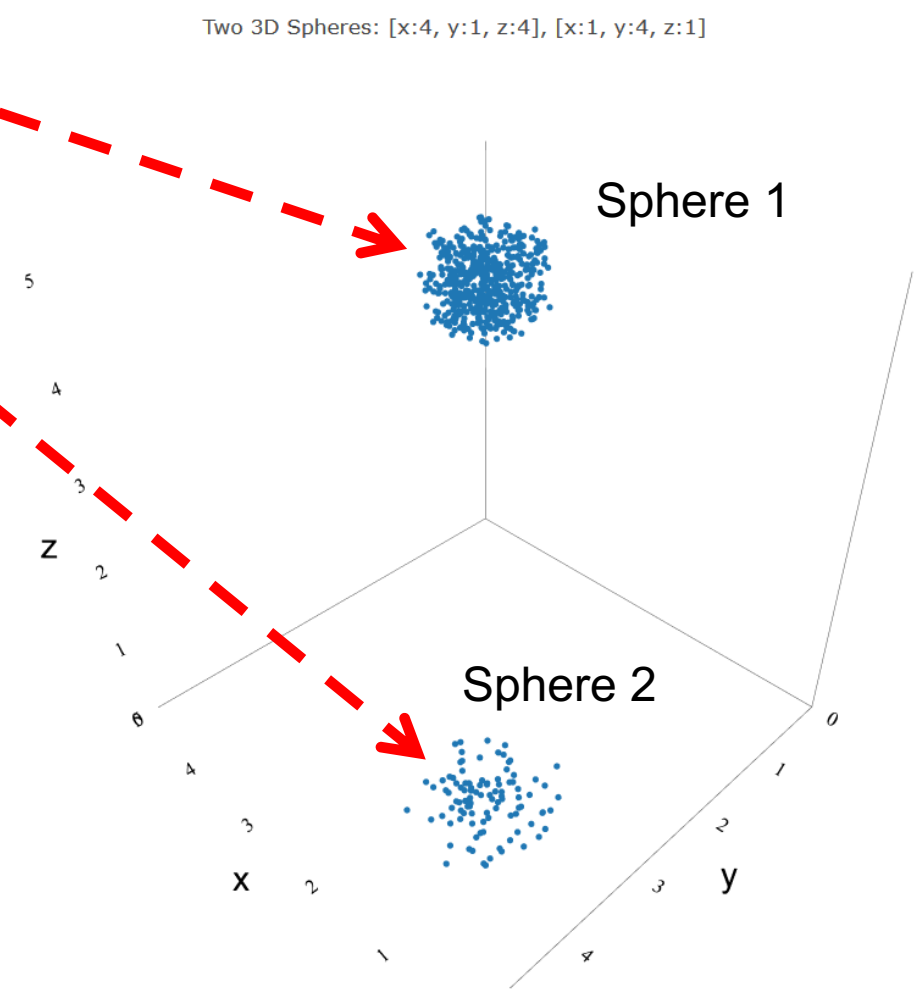
Use Case: Factor Clustering with t-SNE

- t-SNE chart below shows which clusters the risk factors are assigned to
- Greengard et al. (2020) also apply other UML methods
  - PCA
  - k-Means



Toy Examples: t-SNE

- Generate artificial 3D data:
  - Sphere 1: 500 uniform random numbers along [4, 1, 4]
  - Sphere 2: 100 uniform random numbers along [1, 4, 1]
- How do the 2 spheres get mapped with t-SNE?
- We have original data X and want to map them onto a 2-dimensional plane Y
  - See R code “R 3D Plot Toy Example.R”



Kullback–Leibler Divergence: Idea & Formula

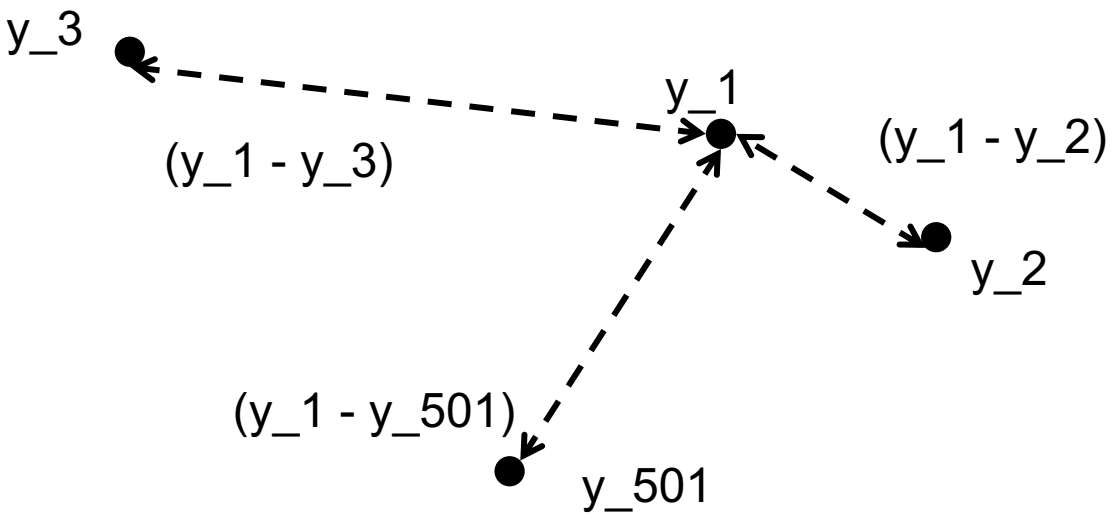
- How can we measure how close the distributions of X and Y are?
- The Kullback–Leibler Divergence is a measure of how one probability distribution (X) is different from a second probability distribution (Y)

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

- p_{ij} measure distances of points i and j in data X ($p_{ii} = 0$)
- $p_{i|j}$ is the probability that i would pick j as its neighbour (we use a gaussian centered at i, more on this later)
- T-sne uses $p_{ij} = p_{ji}$ i.e. its symmetric
- The focus on local probabilities gives the method its **local** flavour
- q_{ij} dto. for data Y
  - p_{ij} and q_{ij} close, e.g., 0.5 and 0.5: : $log\left(\frac{p_{ij}}{q_{ij}}\right)$=0 → KL low
  - p_{ij} and q_{ij} different, e.g., 0.9 and 0.1: $log\left(\frac{p_{ij}}{q_{ij}}\right)$=0.86 → KL high
- KLD can be used to gauge **how close** 2 probability distributions are

Initialising Y

- Goal is to map a 3D dataset X to a 2D map Y: $X=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow Y=\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$
- In our example, Y is initialised with random numbers
  - Typically random numbers with small variance centred around the origin



Mapping Data from X on Y

- How do we arrange the elements of Y so that they reflect X?
  - If 2 points in X are close together, their corresponding parts in Y should also be close together
  - Change the position of the y_i in order to minimise KL Divergence
- First step: calculate the p_ij as in van der Maaten / Hinton (2008), Eq. (1), p. 2581:

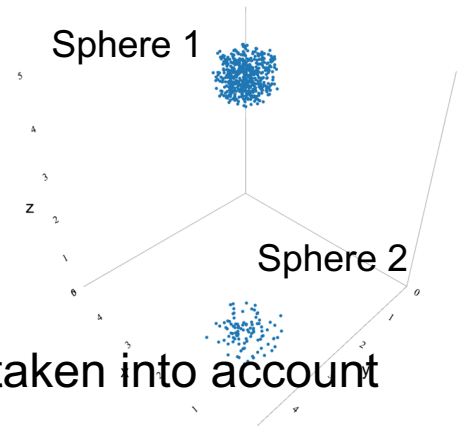
$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)}{\sum_{k \neq l} \exp\left(-\|x_k - x_l\|^2 / 2\sigma^2\right)}$$

- The 2-norm $\|x_i - x_j\|^2$ is the Euclidean Distance
- Goal is to convert the high-dimensional Euclidean Distances between data points to conditional probabilities that represent similarities

Perplexity and Variance

- An important parameter is the Perplexity P that needs to be defined by the user
  - P determines the number of neighbours of each point in X
- The larger P, the larger the variance $\boldsymbol{\sigma\_i^2}$ and the more data points x_ij are included in the neighbourhood
- One feature of t-SNE is that the number of neighbours of x_i is to be the same irrespective of whether the data points are in a densely populated area or in a sparsely populated area
- To achieve this, σ_i varies:
  - In densely populated areas, σ_i is small
  - In sparsely populated areas, σ_i is large
- It is not likely that there is a single value of σ_i that is optimal for all data points in X because the density of the data is likely to vary:
  - In dense regions, a smaller value of σ_i is usually more appropriate than in sparser regions
- SNE performs a binary search for the value of σ_i, hence **σ_i depends on P**

Toy Example: Step 1



- We focus on the 4 points (1, 2, 3, 501) of the 3D chart
- i = 1, j = 2, 3, 501
  - Points 1, 2, 3 are part of sphere 1, point 501 is part of sphere 2
- The larger we choose sigma (1.0, 0.3), the more points are taken into account when calculating probabilities
- Calculating the p_ij as in van der Maaten / Hinton (2018), p. 2584
  - see Excel s/sheet “t-SNE Example CQF.xslx”, tab “t-SNE_sigma only”

		Original Data Set X							
					sigma:	1.0		0.3	
		X_x	X_y	X_z	Euclidean Distance	Numerator	p_ij	Numerator	p_ij
Sphere 1	1	4.4606	0.6049	3.9272	NA	NA	0	NA	0
	2	3.6705	0.5066	3.9192	0.7963	0.6716	0.50	0.0120	0.76
	3	3.6988	1.2193	4.1365	1.0008	0.6063	0.45	0.0038	0.24
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Sphere 2	501	1.3071	4.3273	1.1504	5.6136	0.0604	0.05	0.0000	0.00
	502	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	600	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	Denominator:	NA	NA	NA	NA	1.3382	NA	0.0158	NA
	Sum:	NA	NA	NA	NA	NA	1.00	NA	1.00

Step 2: Calculate the q_ij (p. 2584)

- The map Y contains the same number of points as the data set X
- Y is often initialised with random numbers (or with PCA)
- For example, if Y_1 is randomly initialised with (0.46, 0.68):
  - Then in the first iteration X_1 (4.46, 0.61, 3.93) is mapped onto Y_1 (0.46, 0.68)
- Calculate the q_ij (p. 2585) and plug together with the p_ij into KLD formula:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

- Utilising the t-distribution to measure the area around Y allows us to use the relatively simple formula above
- For the 4 points in our example, this gives a KLD of 0.478 (see Excel s/sheet "t-SNE Example CQF.xslx", tab "t-SNE, KLD", for exact calculation)

Step 3: Building Map Y by Gradient Descent

- How do we make Y reflect X? → Change the points Y until we find an acceptable match
- We want to know in which way we have to change y_1 in order to enhance the match between X and Y
  - Should y_1 be moved up, down, left, right and by how much?
- To this end, gradient descent is useful (eq. 5, p. 2586)

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

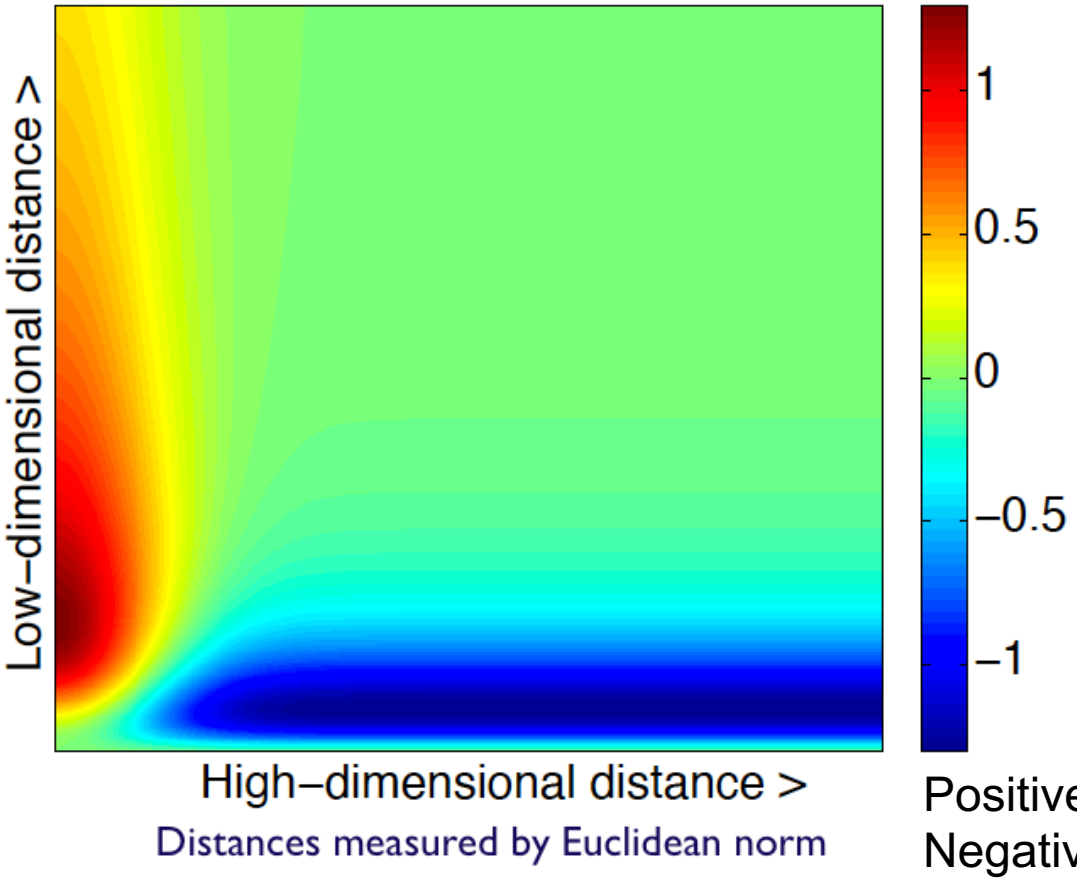
  - δy_i is a vector with #elements = dimension of Y (here: 2)
  - $\frac{\delta C}{\delta y_i}$ is a scalar that gives the degree of change of the cost function

Step 3: Building Map Y by Gradient Descent

- For our example, if we assume $\delta y_i = \begin{pmatrix} -0.14 \\ -0.27 \end{pmatrix}$:
  - The first coordinate (y_11) will be: 0.46 – 0.14 = 0.32
  - 2nd coordinate (y_12): 0.68 – 0.27 = 0.41
- The KL Divergence will change from 0.478 to a smaller value (i.e., $\frac{\delta C}{\delta y_i} < 0$) $\rightarrow$ by changing Y according to the gradient we are moving closer to X
- The other coordinates are treated in the same way

Gradient Descent: Rationale

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



Positive values represent attraction
Negative values represent repulsion

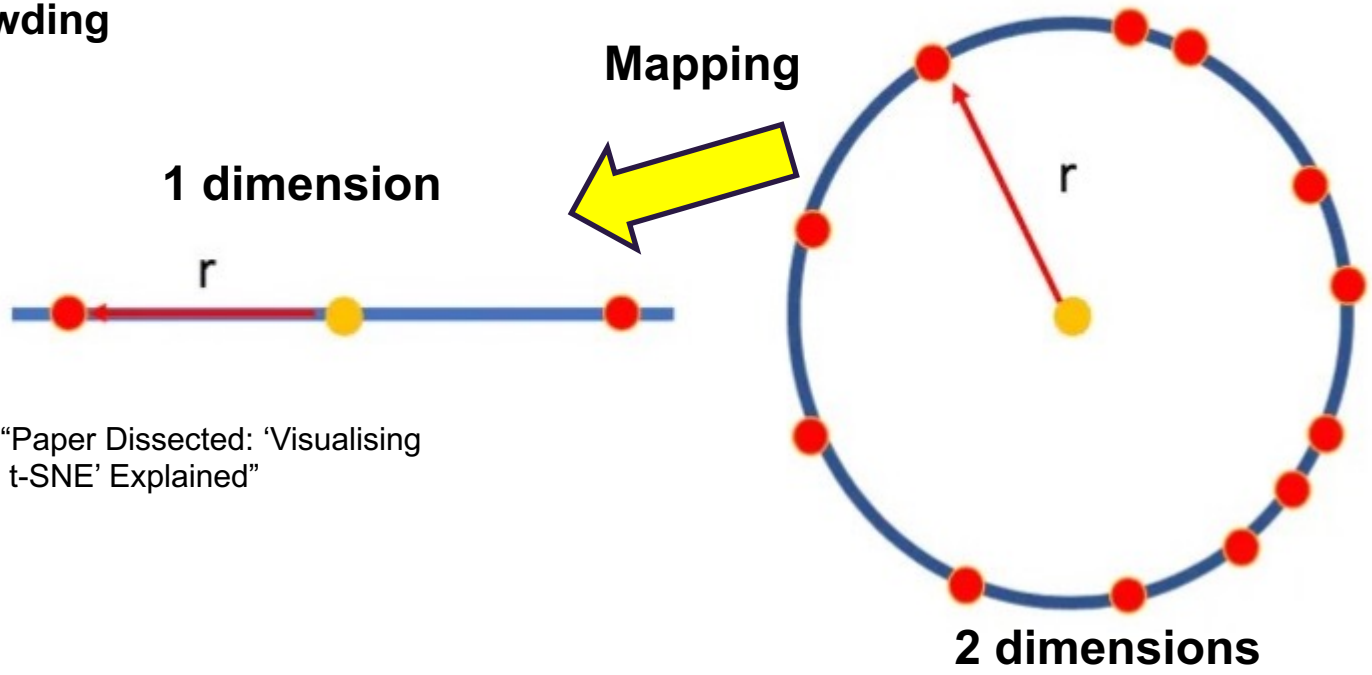
Learning Via Gradient Descent: Elements

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$$

- $\mathcal{Y}^{(t)}$: the solution for Y at iteration t (matrix N x 2)
- η is the learning rate
  - η large: new solution is strongly affected by gradient, i.e., fast learning. η small: gradient has little impact on new solution, i.e., slow learning
  - Rtsne uses a default learning rate of 200 (can be changed)
  - For large data sets (N > 10k), Belkina et al. (2019) recommend η = N/12
- α(t) is the momentum at iteration t
  - α large: prior solutions get more weight in new solution, less pronounced changes induced by gradient descent
  - Works best if the momentum term is small until the map points have become moderately well organised
  - Rtsne starts with α=0.5 and towards the end of the learning process is α=0.8
- In our toy example, η = 1, α = 0

The Curse of Dimensionality: Over-Crowding

- Data points in **high**-dimensional space that have a **similar distance** from the reference point get squashed together in low-dimensional space
- Even though the data points are different from each other, they get mapped onto the same points in **low**-dimensional space: over-crowding
  - There is **not enough space** in the **low**-dimensional space to allow clusters to separate
  - Clusters are forced to collapse on top of each other in low-dim. space → **Over-Crowding**



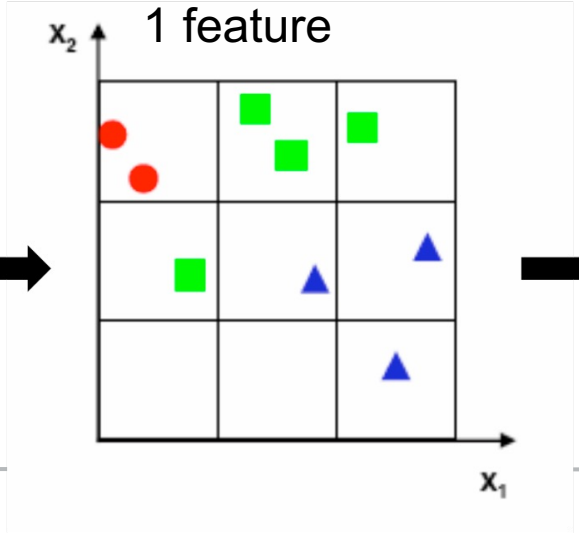
From blog "Paper Dissected: 'Visualising Data using t-SNE' Explained"

The Curse of Dimensionality

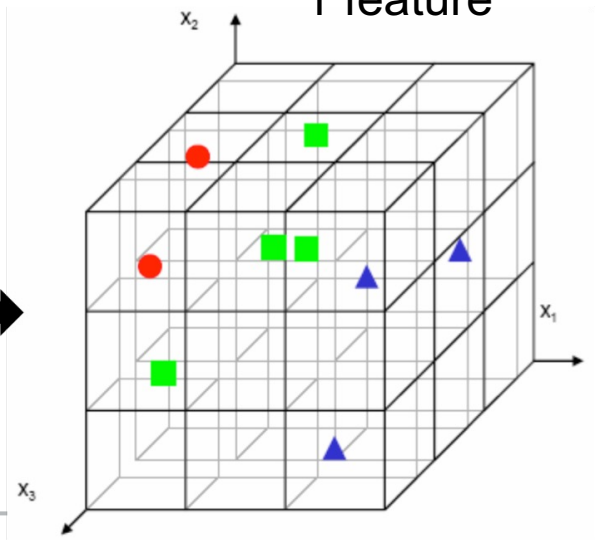
- When dimensions (or variables) are added, the volume of the space increases so fast that the available data becomes sparse
  - This can impede our model to find reliable clusters
  - Value added by additional dimension is small compared to its contribution to find reliable clusters
  - Increases storage space and processing time
  - Typical rule of thumb: min. 5 training examples for each dimension

10 data points,
1 dimension,
1 feature

10 data points,
2 dimensions,
1 feature



10 data points,
3 dimensions,
1 feature



Over-Crowding

- How can we address the crowding problem?
- Increase the dimensionality of the low-dimensional map Y
  - This leaves more space in Y for clusters to separate
- Often the low-dimensional map has only 2 or 3 dimensions
  - Hence increasing dimensionality in most cases not a viable option
- Ideally, attractive and repulsive forces in the optimisation manage to map the high-dimensional space X onto Y

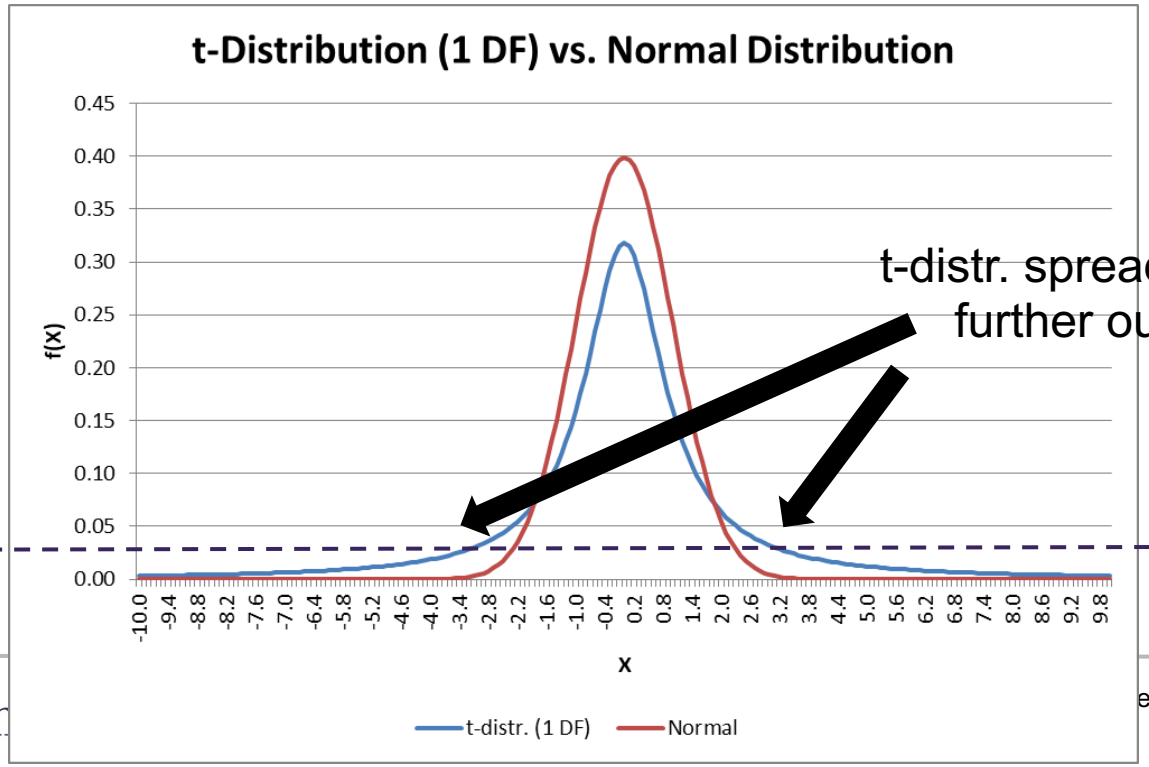
t-Distribution

- Utilising the t-distribution helps t-SNE to deal with over-crowding effectively
- t-SNE employs a heavy-tailed distribution (**t-distribution**) in the **low-dimensional** space Y to alleviate both the crowding problem and the optimisation problems of SNE
- In order to build a lower-dimensional representation of **X**, t-SNE utilises the **normal** distribution to convert distances of data points to probabilities p_ij
- Map **Y** uses probabilities q_ij to reflect the original distances in lower-dimensional space with the **t-distribution**

Gaussian vs. t-distribution with 1 DF

- The Gaussian is tighter around the reference point than the t-distribution
- There is a higher likelihood that the Gaussian leads to over-crowding
- Potentially the t-distribution **allows more space** around the reference point and hence reduces the effect of over-crowding

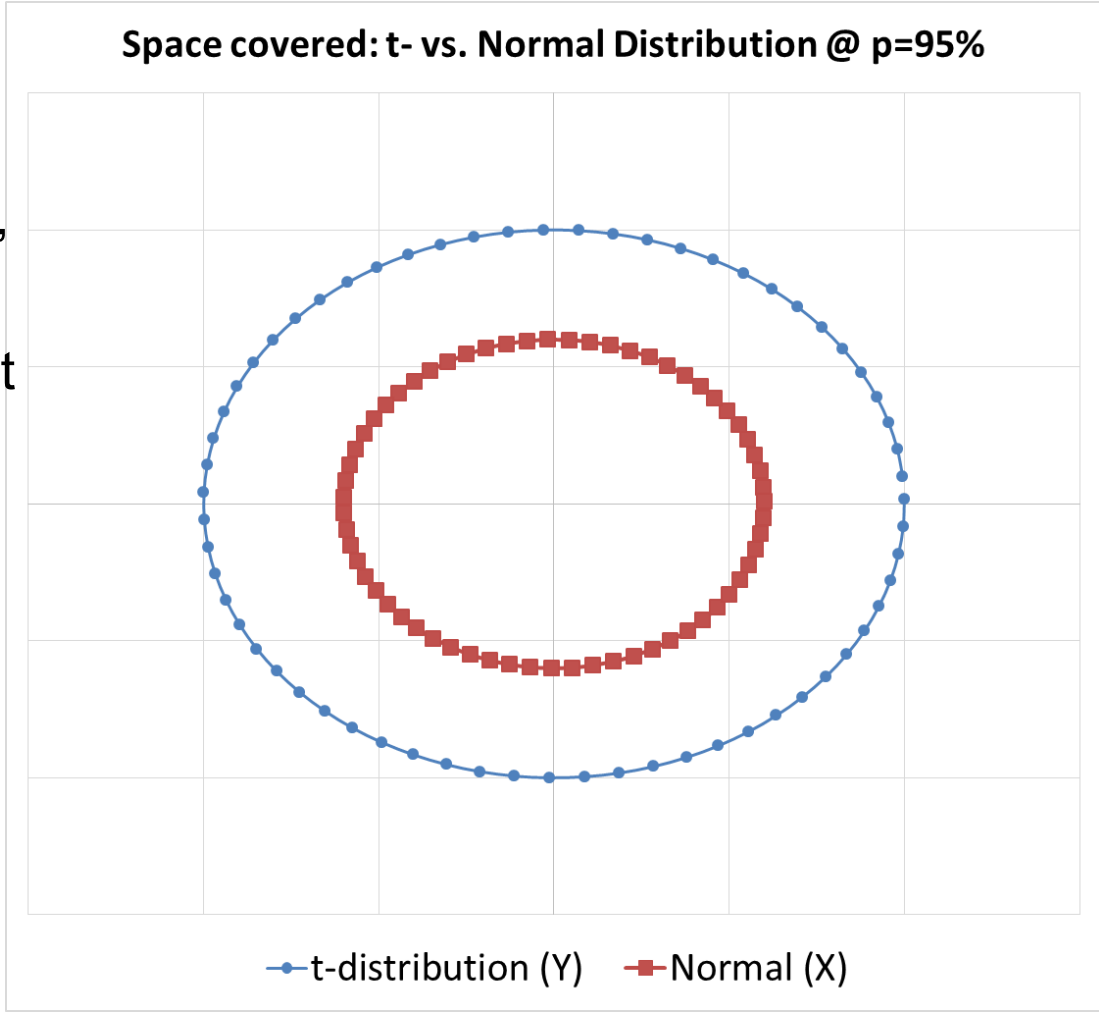
At a probability of 2.5% the t-dist covers a wider area around reference point than the Normal dist



Gaussian vs. t-distribution with 1 DF

- t- vs. Normal Distribution seen from a bird's perspective
- At the same level of probability, t-distribution covers a wider area around the reference point
- **There is more space to reflect similar data points**
- Mismatched tails (bw. Normal and t-distribution) can compensate for mismatched dimensionalities

→ helps to mitigate over-crowding

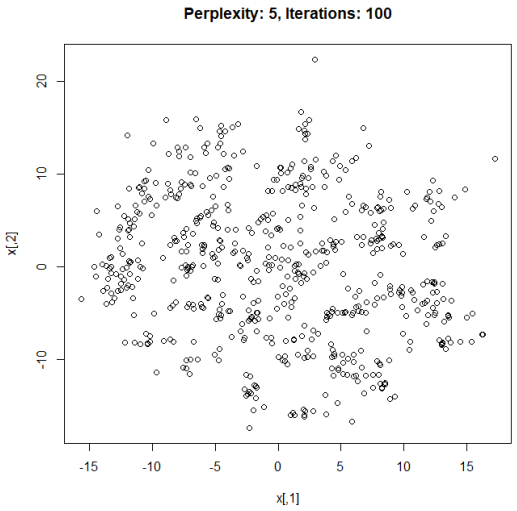


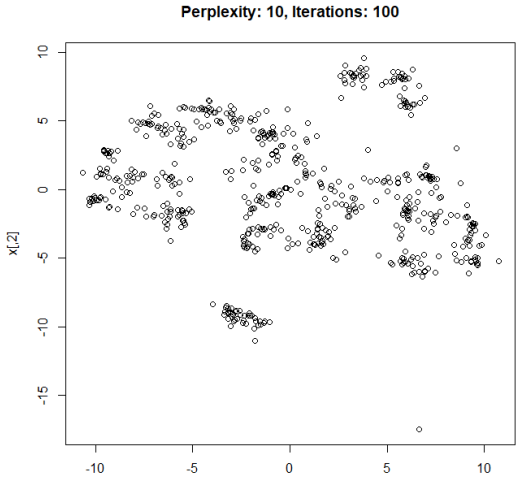
t-SNE: Features

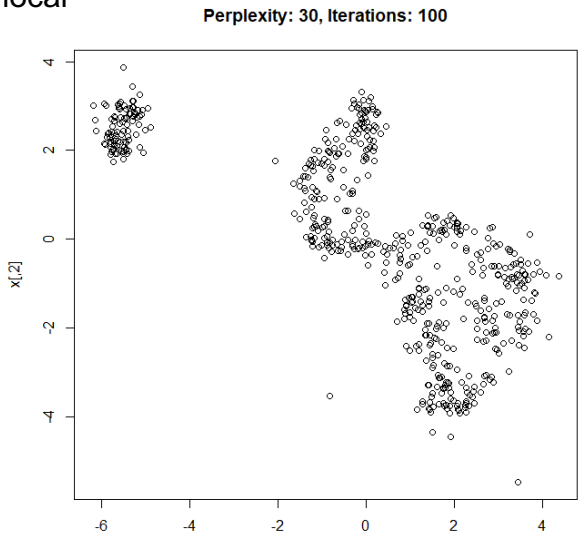
- t-SNE is a nonlinear dimensionality reduction technique for data visualisation
- t-SNE retains **local** variance (PCA → **global** variance)
- **Distances between clusters of the resulting Y map are potentially meaningless and cannot be related to distances of clusters in X**
  - t-SNE shrinks widespread data and expands densely packed data
  - This is due to determining the number of neighbours with perplexity, which varies σ_i
- t-SNE is stochastic (random initialisation): each run may lead to a different output

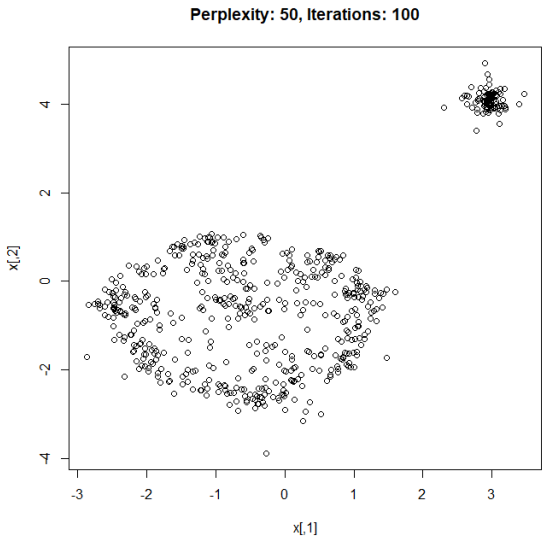
100 Iterations (= Learning Steps) Each, Different Perplexities

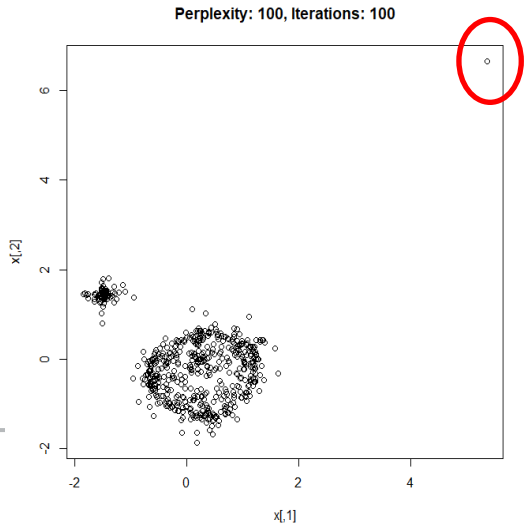
- Data from toy example (600 data points, 2 clusters), **R package tsne**, see R code “R 3D Plot Toy Example.R”
- van der Maaten & Hinton (2008) recommend perplexity values in the range 5 to 50
  - With P = 5 (σ_i small), local variations dominate
  - P large (e.g., 100, σ_i large) means Y map reflects more global structure, less local

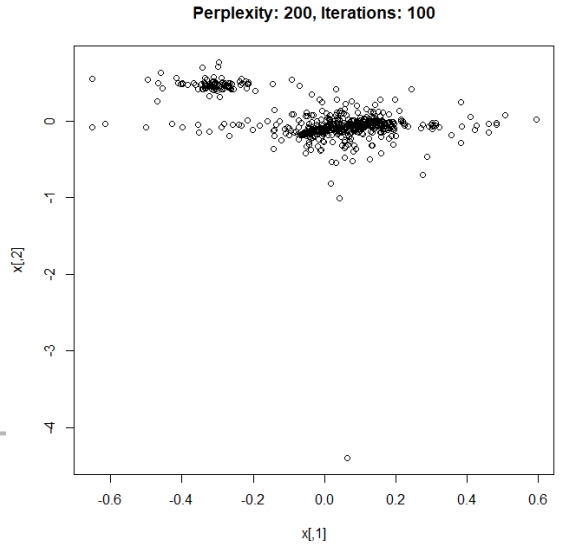






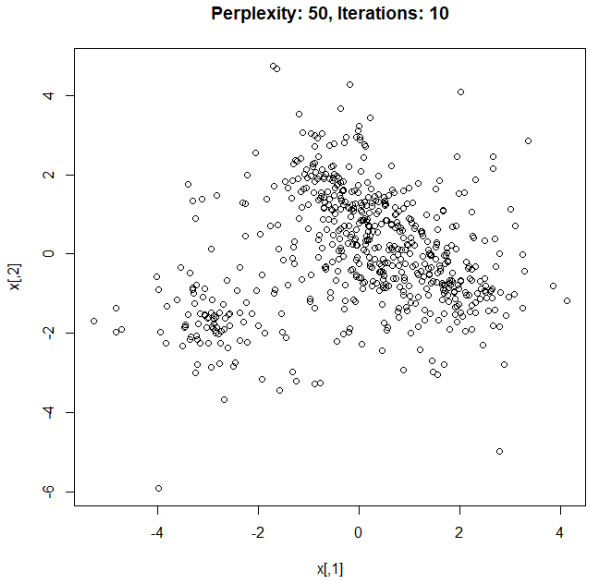


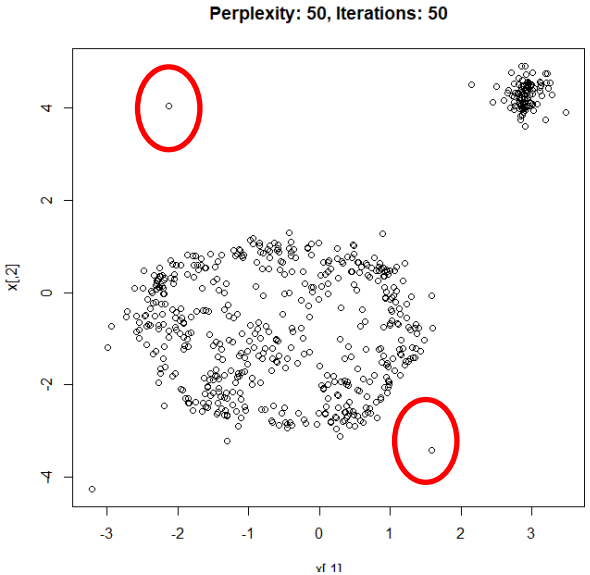


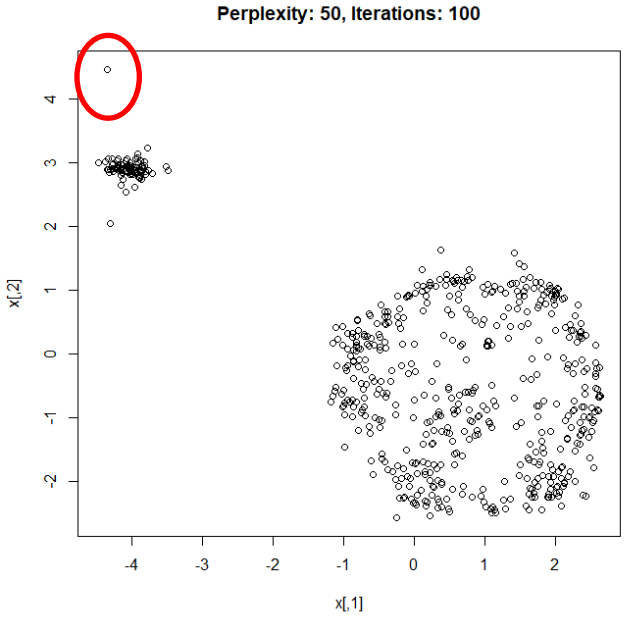


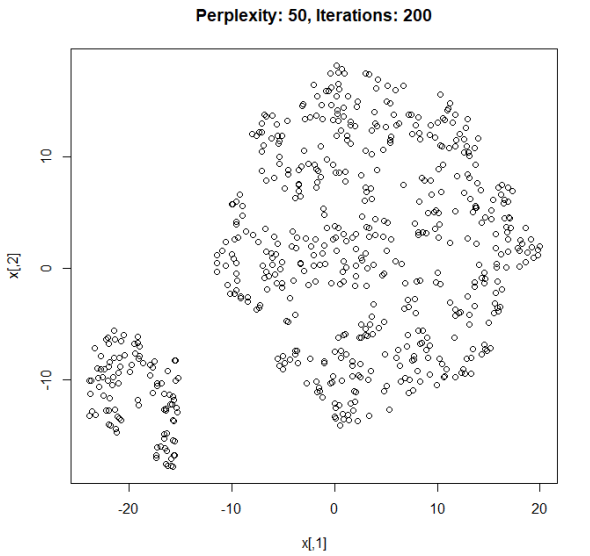
Perplexity Always 50, Different Iterations

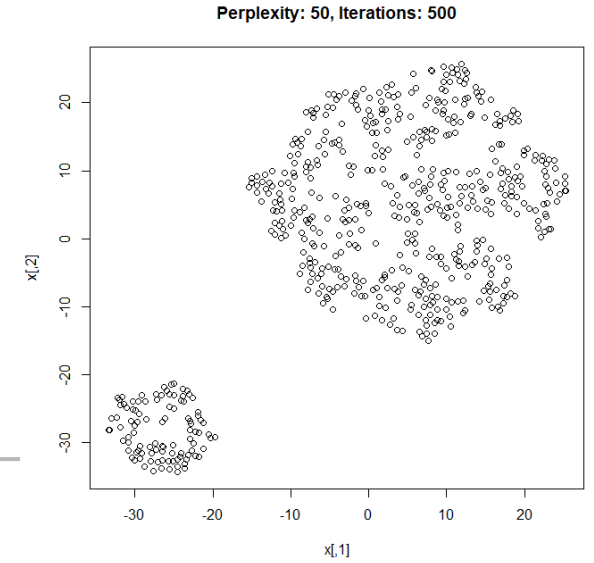
- 50 to 100 iterations sufficient for our toy example, **R package tsne**
- No significant improvement beyond 100 iterations

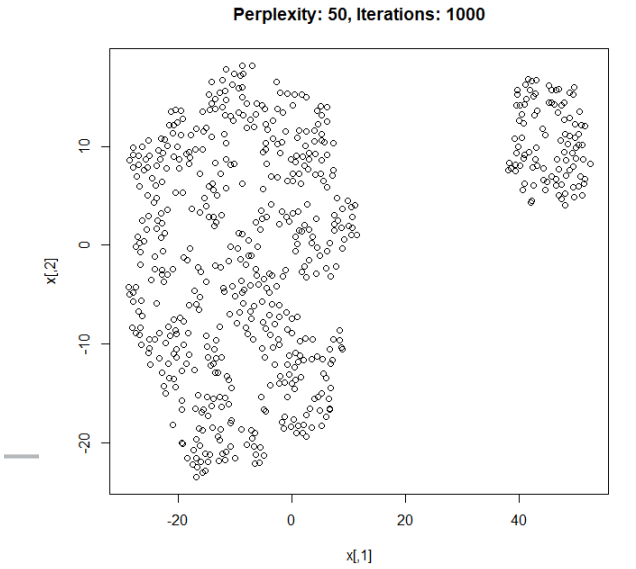






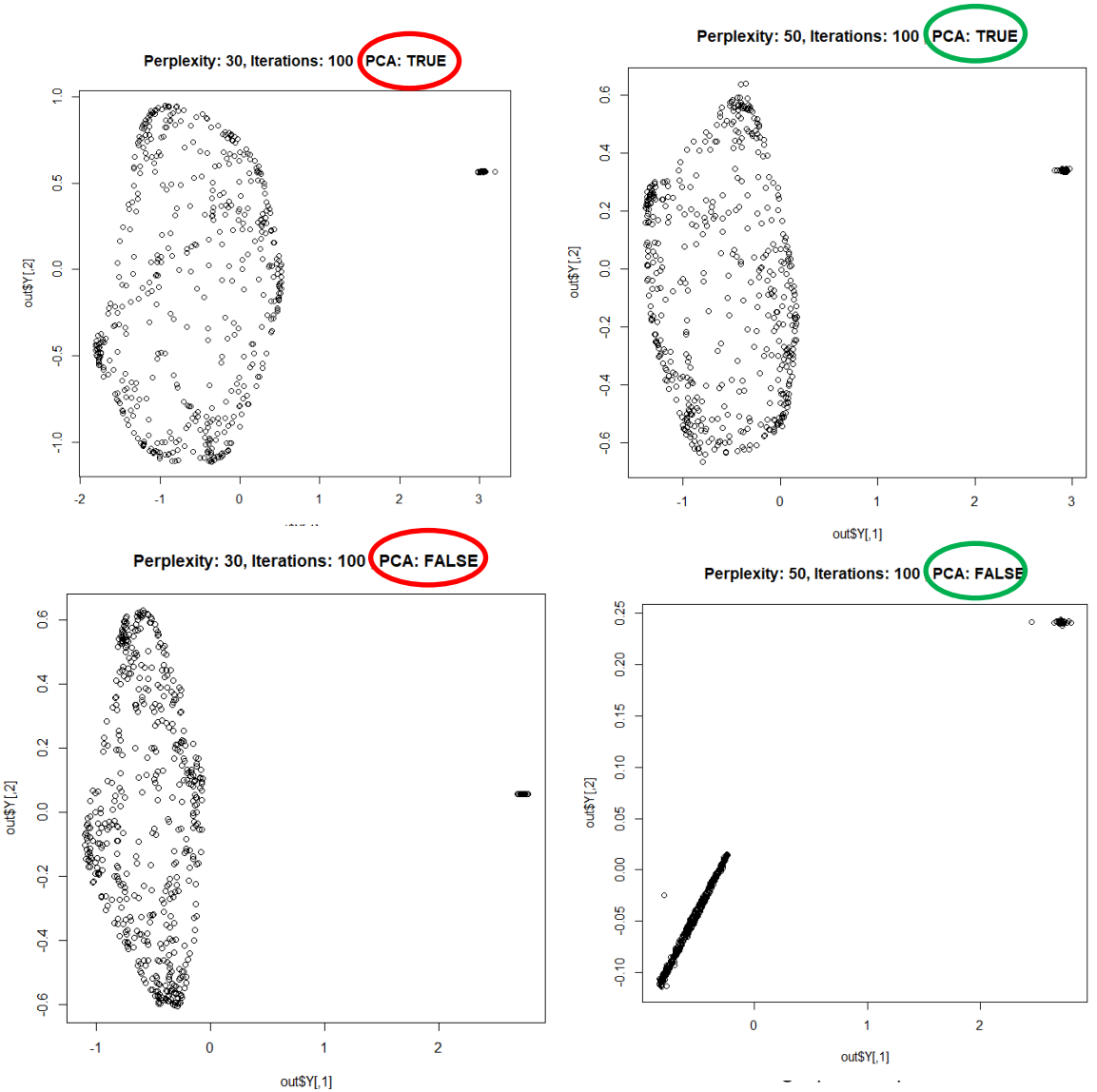






t-SNE: Initialisation PCA vs. Random

- **R package Rtsne**
- PCA initialisation (upper 2 charts) provides stable global structure
- At the bottom are random initialisations without PCA
  - Perplexity 30 provides enough outreach to display global structure w/o PCA (bottom left)
  - Perplexity 50 projects all points in each of the spheres closely together (bottom right)



t-SNE: Perplexity, PCA Initialisation & Learning Rate

- Simple rules of thumb (Kobak / Berens (2019)):
  - Perplexity = 1% of sample (i.e., 1 data point is connected to 1% of all data points)
    - P_min = 30
    - There are suggestions of data-driven variable perplexities (e.g., De Bodt et al. (2018)), but these are not part of standard software yet
  - PCA initialisation sets global structure
    - How many principal components to use? → for example, scree plot, or use the first 2 PCs
    - t-SNE algorithm optimises local structure
    - PCA initialisation is deterministic and reproducible and hence avoids large dependence of results on random initialisation
    - If PCA also gives bad results, then maybe there is not much structure in data
    - It is unusual that PCA works well but t-SNE does not
  - Learning rate $\eta = \max(200, N / 12)$
    - η large: new solution is strongly affected by gradient, large weight changes
    - η small: gradient has little impact on new solution, small weight changes
- These rules particularly suited for larger data samples (N > 10k)

R Software for Running t-SNE

- R implementation:
  - Rtsne is probably closest to original paper by van der Maaten / Hinton (2008)
  - tsne contains fewer parameters and might be the more suitable package to start with
- PCA to initialise t-SNE: Rtsne allows to conveniently run initial PCA
  - Use the first two components from PCA, scaled so their standard deviations are initially 0.0001
  - Automatic pre-processing with PCA is not available in tsne
- See also complementary R code “R 3D Plot Toy Example.R” and “R t-SNE, UMAP, AE, US YC, CQF.R”
- Out of sample mapping: As yet not available in standard software packages

Barnes-Hut t-SNE

- Barnes-Hut t-SNE (Van Der Maaten (2013)) is an algorithm that increases efficiency by treating clusters of faraway objects as single particles
  - Implemented in Rtsne
- Barnes-Hut is an approximation that brings complexity down and speeds up convergence
- An approximation used in Barnes-Hut t-SNE is to only calculate p_ij for n nearest neighbours of i, where n is a multiple of the user-selected perplexity and to assume p_ij = 0 for all other j

UMAP: Uniform Manifold Approximation and Projection

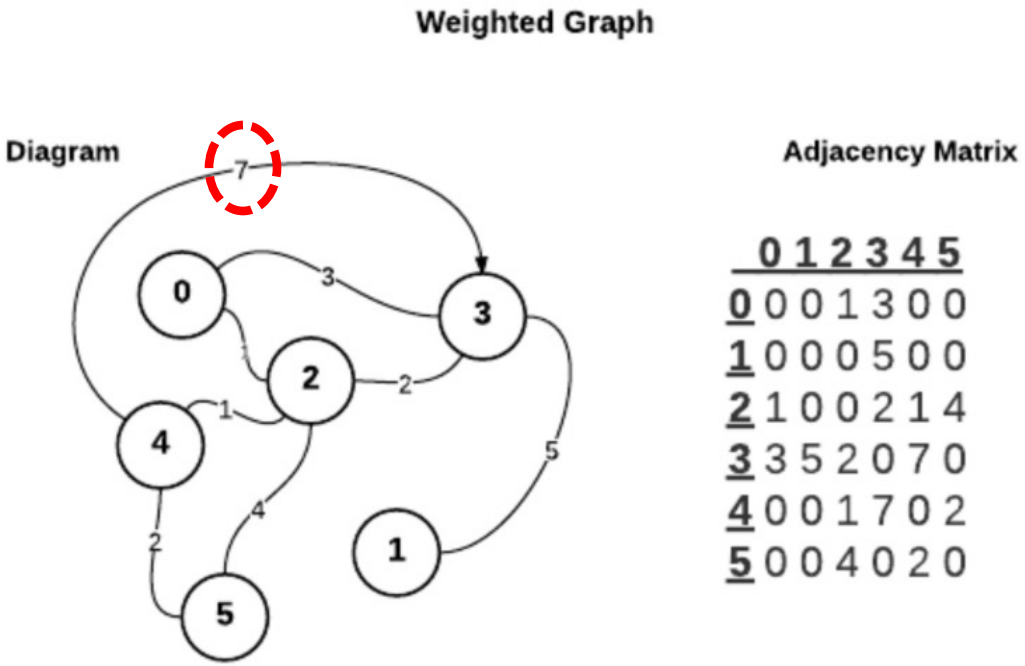
- UMAP proposed by McInnes et al. (2018)
- UMAP can ultimately be described in terms of, construction of, and operations on **weighted graphs**
- UMAP's intuition is similar to t-SNE's:

1) UMAP constructs a **high-dimensional** weighted k-neighbour **graph** representation of the **original data X**
2) UMAP optimises a **low-dimensional** graph to be as structurally similar as possible to map on Y

UMAP: Weighted Graphs

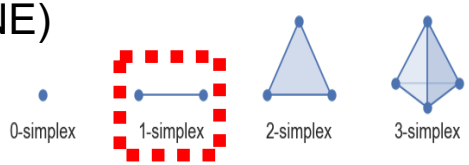
- In a weighted graph each edge has an associated weight or number
- The number can represent many things, such as a distance between 2 locations on a map or between 2 connections in a network
- Typically a weighted graph refers to an edge-weighted graph, i.e., a graph where edges have weights or values

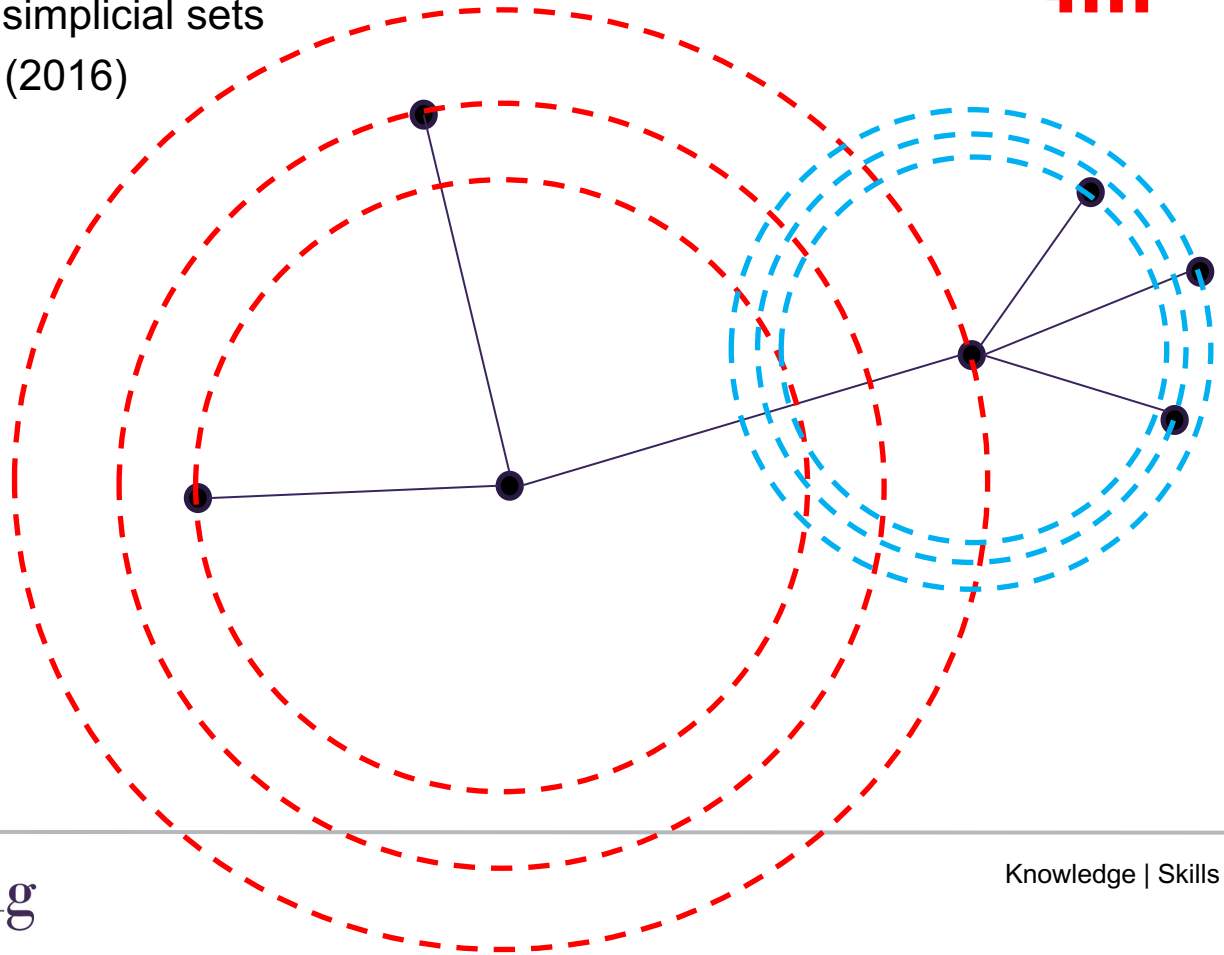
- Edge weights represent the likelihood that two points are connected, e.g., “7” between points 3 and 4 means probability these 2 are connected = 7%
- See also Adjacency Matrix



Simplicial Sets

- UMAP in high-dimensional space X from the reference point x_i expands radius σ_i until k nearest neighbours are covered (here: k = 3)
- Each point in X has exactly 1 representative in Y (similar to t-SNE)
- The nearest neighbours are connected via 1-simplex structures:
- For details on simplicial sets see Friedman (2016)





UMAP: Building a Fuzzy Simplicial Complex

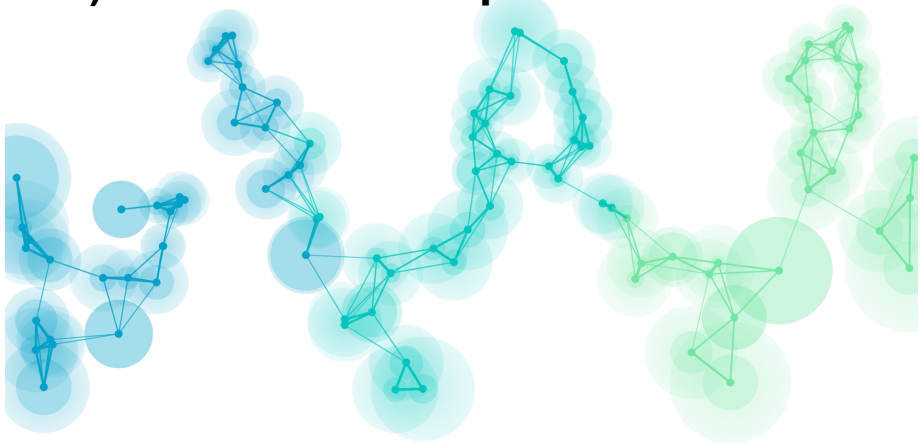
- To construct the initial high-dimensional graph of X, UMAP builds a "fuzzy simplicial complex"
- This is a representation of a weighted graph, with edge weights representing the likelihood that 2 points are connected
- The farther away 2 points are, the lower the likelihood that they belong to the same cluster
  - To determine connectedness, UMAP extends a radius σ_i outwards from each point i, connecting points when those radii overlap
  - Choosing this radius is critical - too small a choice will lead to small, isolated clusters, while too large a choice will connect everything together

UMAP: Learning

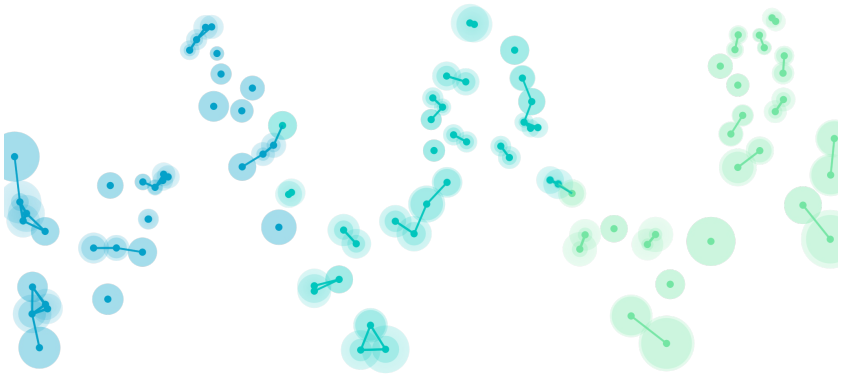
- The weighted graph for X grows in the course of learning

1) Initialisation

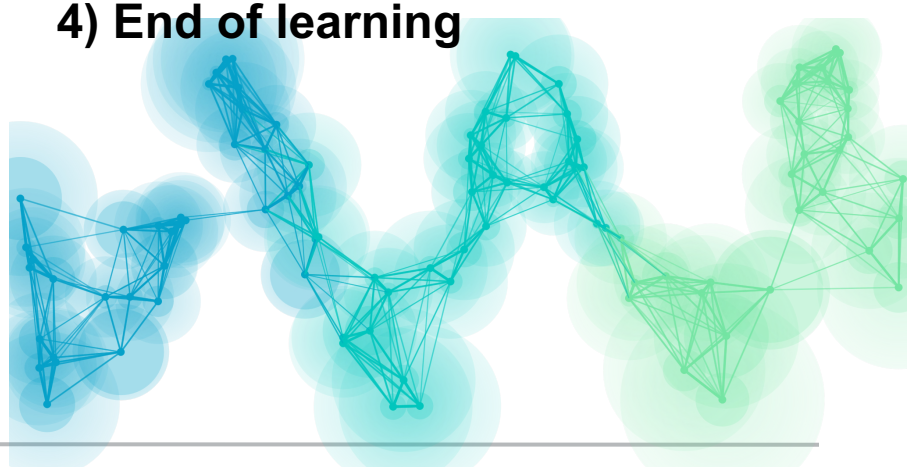
3) After further steps



2) First few steps: extend radii

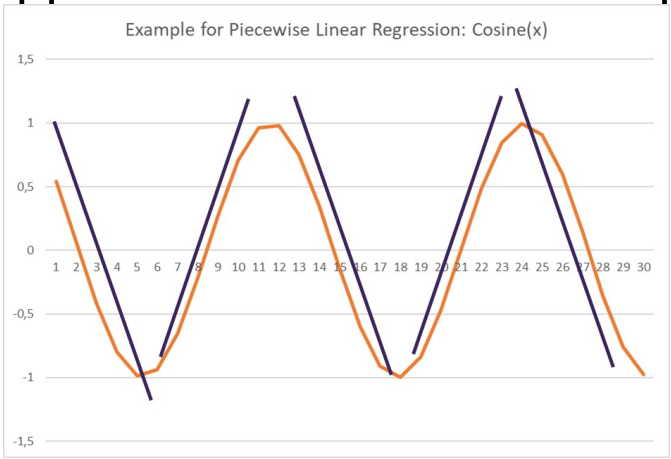


4) End of learning



UMAP: Learning

- UMAP choses σ_i locally, based on the distance to each point's n-th nearest neighbour
  - Each point must be connected to at least its closest neighbour
  - This ensures that local structure is preserved in balance with global structure
- UMAP uses local manifold approximations → similar to piecewise linear regression models

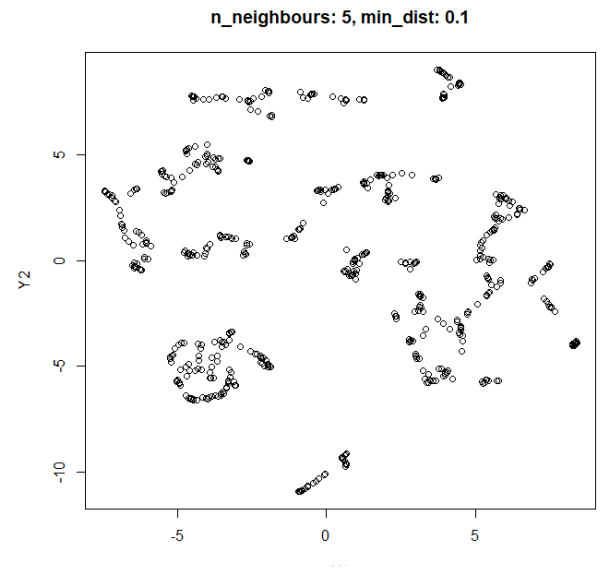


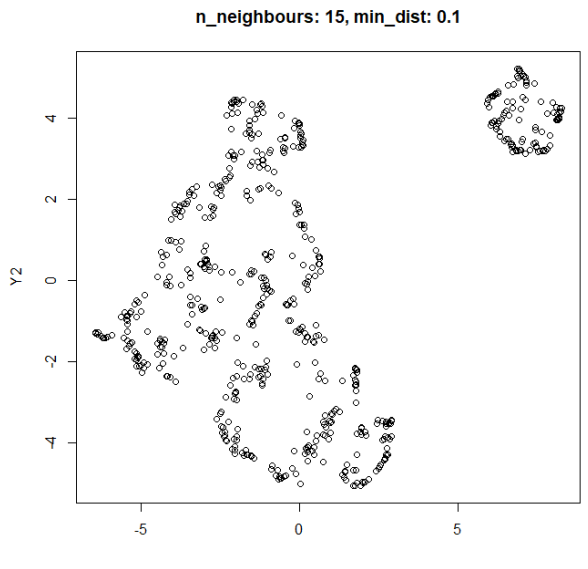
- Idea is to glue together simple building blocks to reflect a more complex topography, i.e., the "fuzzy simplicial complex"
- Fuzzy simplicial set representations are used to construct a topological representation of X

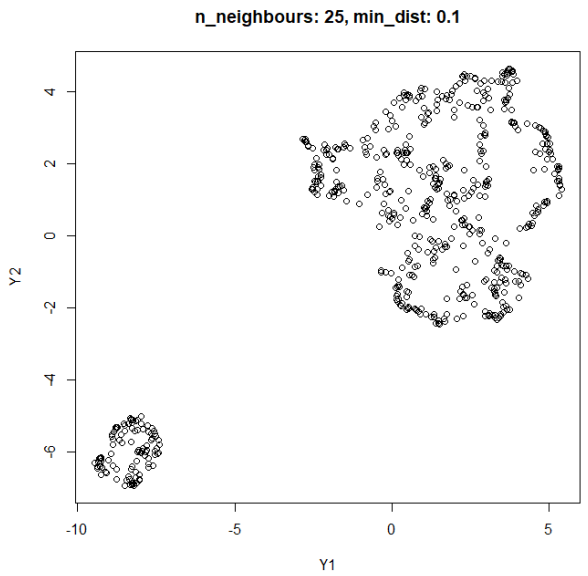
UMAP: Hyper-Parameter n_neighbours

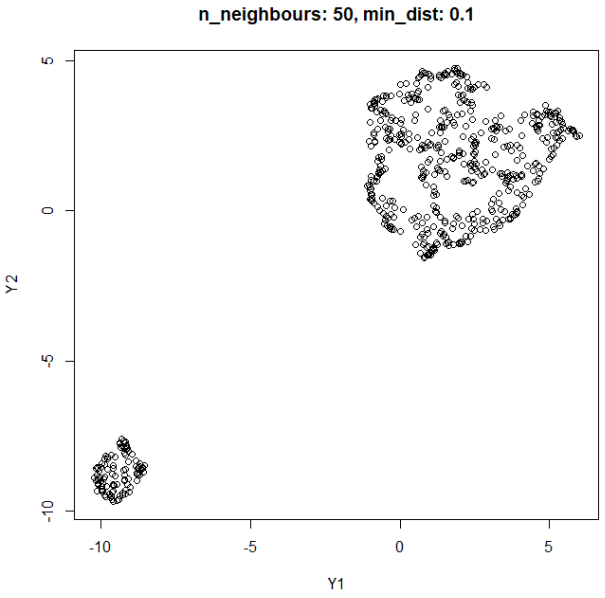
- The most important parameter is n_neighbours - the number of approximate nearest neighbours used to construct the initial high-dimensional graph
- n_neighbours controls how UMAP balances local versus global structure
  - low values will connect only few data points: UMAP to focus more on local structure
  - high values will connect many data points: UMAP represents the global structure while losing detail

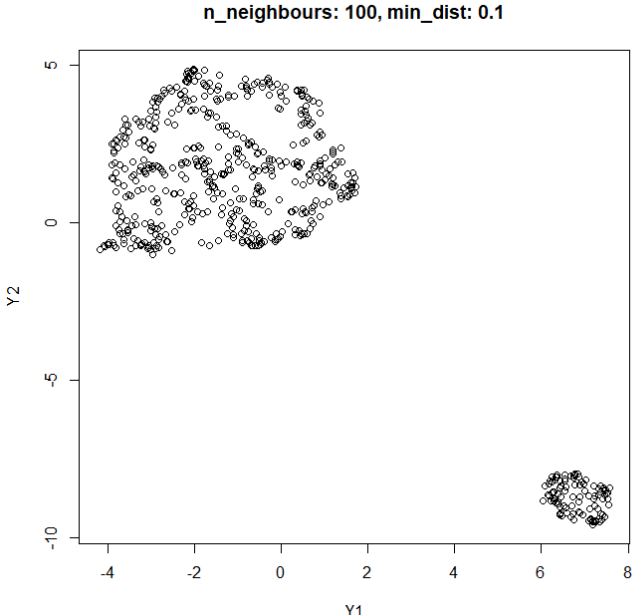
UMAP: Toy Example, Varying n_neighbours

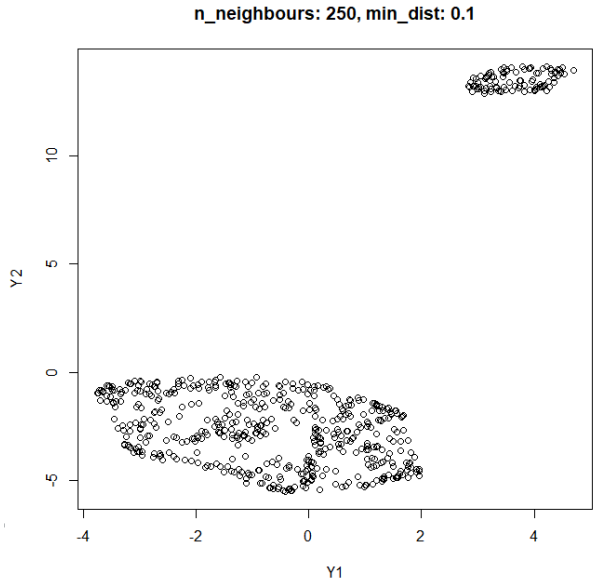








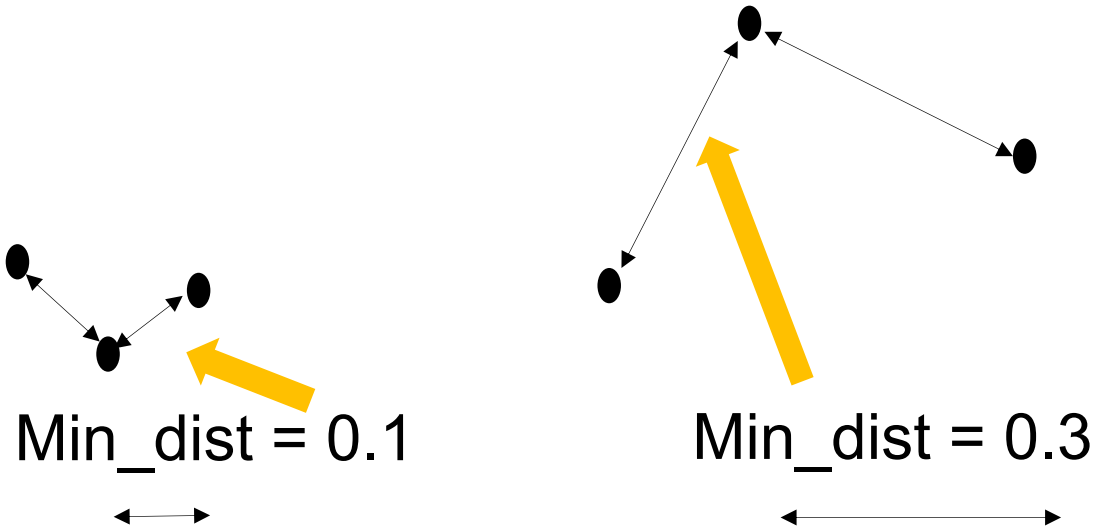




- See R code “R 3D Plot Toy Example.R”

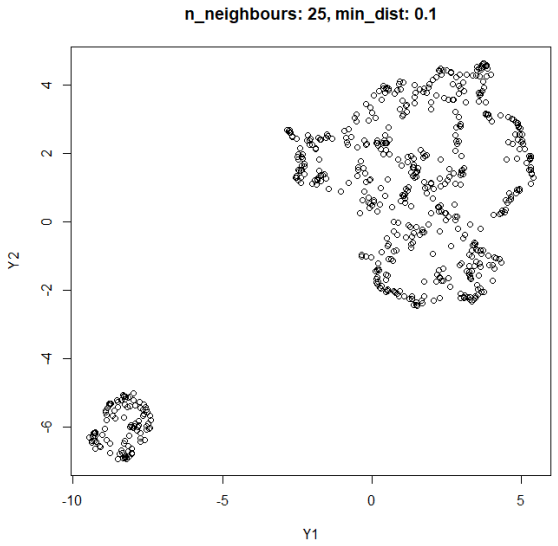
UMAP: Hyper-Parameter min-dist

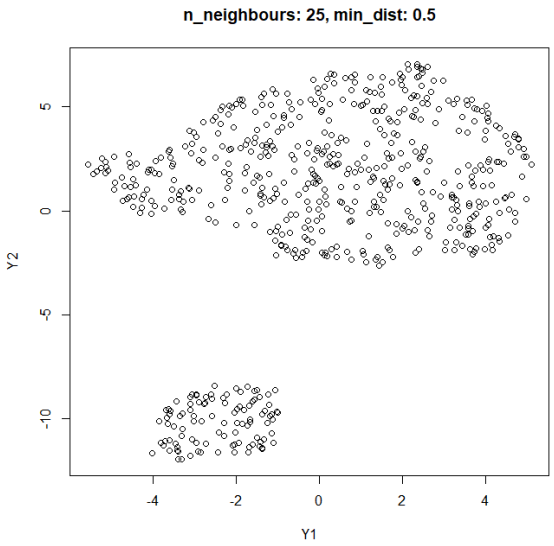
- min-dist is a hyperparameter that affects the output Y
- If 2 points have a distance smaller than min-dist in Y, their distance is set to min-dist
- min-dist determines how closely together the points on Y can be mapped together:
  - min-dist low (for example, 0.1): potentially densely packed regions in Y
  - min-dist high (for example, 0.9): points on Y more spread out
- McInes et al. regard min-dist as an “aesthetic parameter” that influences visualisation

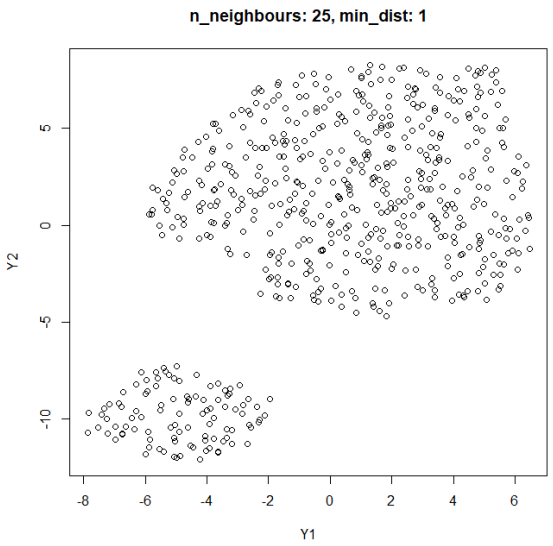


UMAP: Toy Example, Varying min_dist (n_neighbours = 25)

- As the min-dist parameter increases, UMAP tends to "spread out" the projected points, leading to decreased clustering of the data and less emphasis on local structure
- See R code “R 3D Plot Toy Example.R”
  - More examples on: https://pair-code.github.io/understanding-umap/







UMAP vs. t-SNE: Cost Functions

- t-SNE cost function (source: McInnes et al. (2018)):

Same formula as on slide 9, rearranged

$$C_{t-SNE} = \sum_{i \neq j} p_{ij} \log p_{ij} - p_{ij} \log q_{ij} \qquad (11)$$

- The first part of eq. (11) concerns the high-dim. data X and is constant during the optimisation process
- Hyper parameter perplexity affects p_ij
- The second part (via the q_ij) addresses the low-dim. data Y and is changed during the optimisation process
- UMAP's cost function looks similar:

$$C_{UMAP} = \sum_{i \neq j} v_{ij} \log v_{ij} + (1 - v_{ij}) \log (1 - v_{ij}) \\ - v_{ij} \log w_{ij} - (1 - v_{ij}) \log (1 - w_{ij}) \qquad (14)$$

- Eq. (14) also comprises a constant part (the term with the v_ij) and a changing part (w_ij)
- Hyper parameter n_neighbours affects the v_ij, min-dist affects the w_ij

UMAP: Similarities v_ij for X

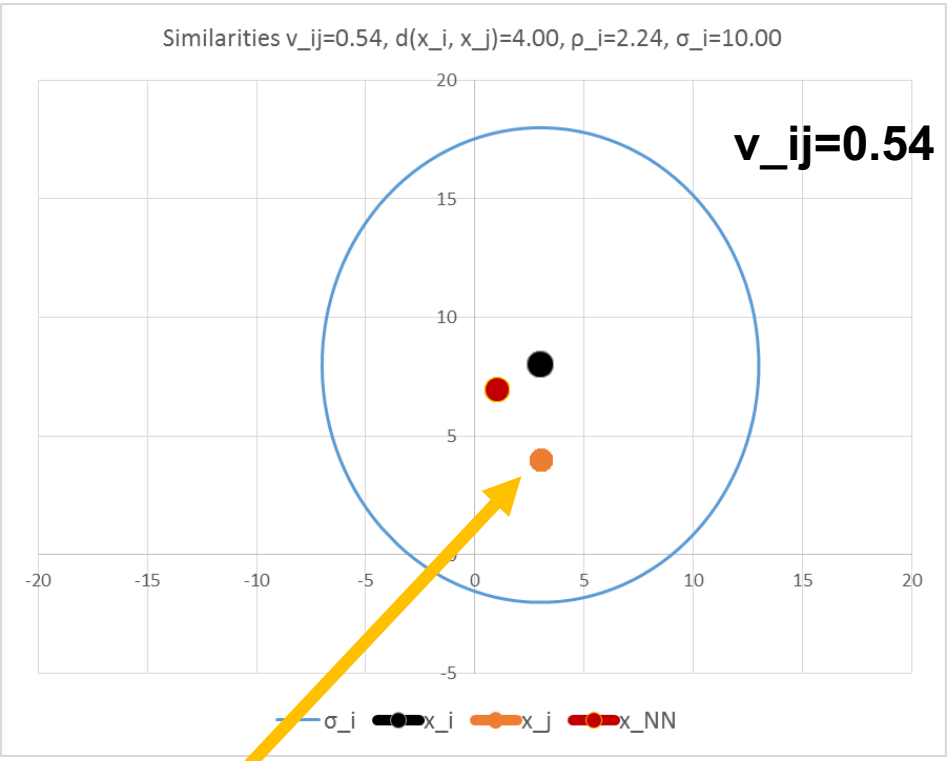
- The similarities v_ij in the **high-dimensional** space are the local fuzzy simplicial set memberships (McInnes et al. (2018), p. 50):

$$v_{j|i} = \exp[(-d(x_i, x_j) - \rho_i)/\sigma_i] \qquad (15)$$

- d(x_i, x_j) is the distance between x_i and x_j (e.g., Euclidean Distance)
- ρ_i is the distance of point i to its nearest neighbour
- σ_i is the normalisation factor, similar to σ_i in t-SNE
- Each point x_i has v_ij to connect to the other k nearest neighbours in X
- Each x_i has an adjacency matrix that contains its v_ij
- The next 2 slides give examples for v_ij, given reference point x_i

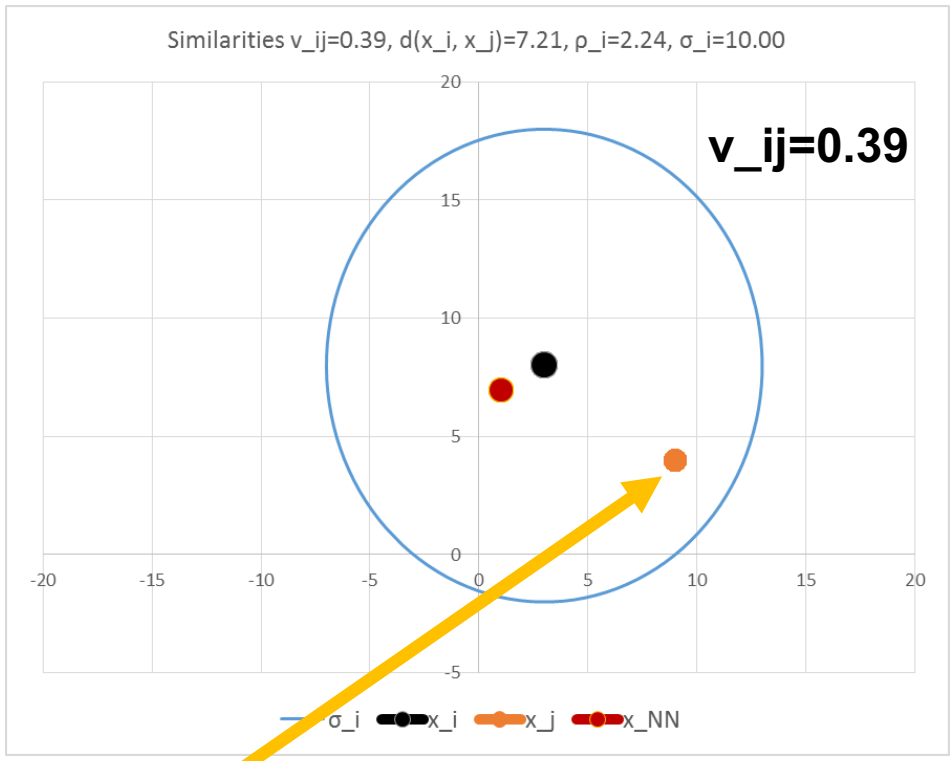
UMAP: Similarities v_ij for X, how does v_ij change when x_j changes?

- A point (orange) farther away from x_i gives a smaller v_ij



	Inputs	
x_i	3.0	8.0
x_j	3.0	4.0
x_NN	1.0	7.0
σ_i	10.0	

Parameters for v_ij	
ρ_i	2.24
d(x_i, x_j)	4.00
v_ij	**0.54**

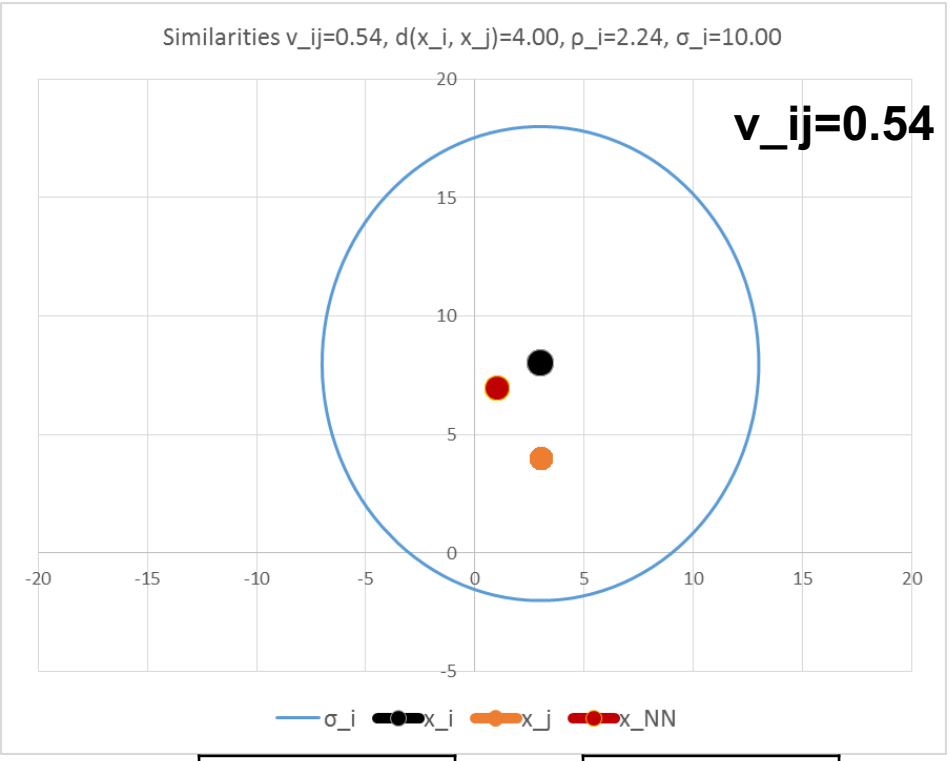


	Inputs	
x_i	3.0	8.0
x_j	9.0	4.0
x_NN	1.0	7.0
σ_i	10.0	

Parameters for v_ij	
ρ_i	2.24
d(x_i, x_j)	7.21
v_ij	**0.39**

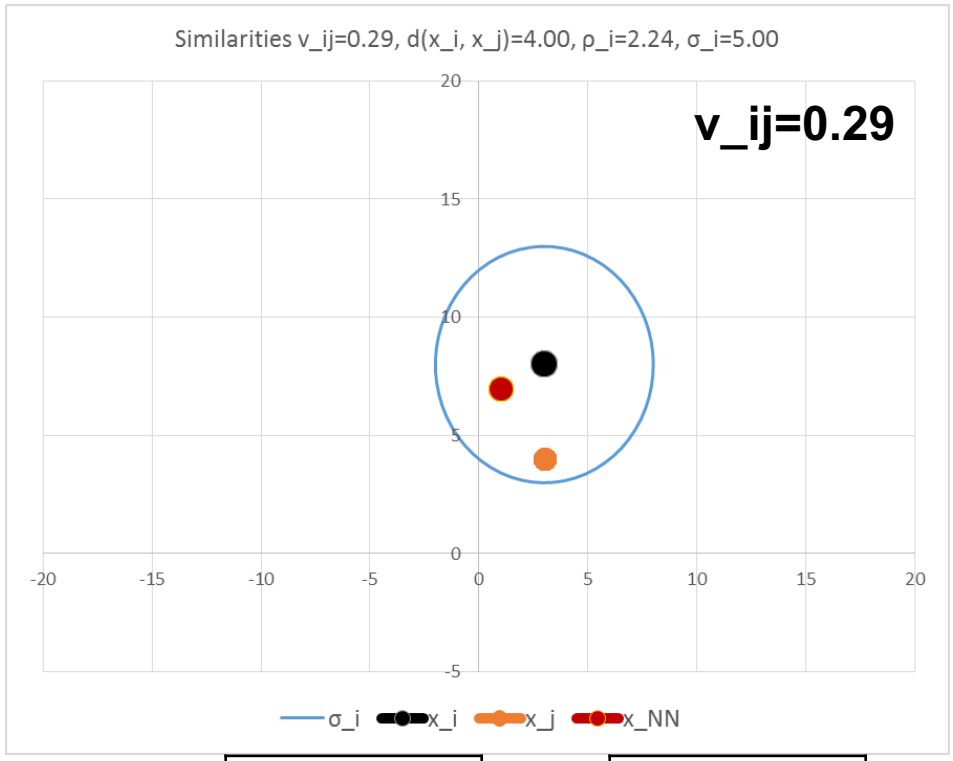
UMAP: Similarities v_ij for X, how does v_ij change when σ_i changes?

- Reducing σ_i also reduces v_ij



	Inputs	
x_i	3.0	8.0
x_j	3.0	4.0
x_NN	1.0	7.0
σ_i	10.0	

Parameters for v_ij	
ρ_i	2.24
d(x_i, x_j)	4.00
v_ij	**0.54**



	Inputs	
x_i	3.0	8.0
x_j	3.0	4.0
x_NN	1.0	7.0
σ_i	5.0	

Parameters for v_ij	
ρ_i	2.24
d(x_i, x_j)	4.00
v_ij	**0.29**

UMAP: Similarities w_ij for Y

$$X=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow Y=\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

- The similarities w_ij in the **low-dimensional** space are calculated as (McInnes et al. (2018)):

$$w_{ij} = \left(1 + a \left\| y_i - y_j \right\|_2^{2b}\right)^{-1} \qquad (17)$$

- a and b are determined with a search algorithm
  - With the UMAP default settings, the 2 parameters are found as a ≈ 1.93 and b ≈ 0.79
  - If we set a = b = 1, the t-distribution results
- By changing the y and hence the w_ij the goal is to minimise the cost function C_{UMAP} (eq (14)) via gradient descent

UMAP vs. t-SNE

$$X=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow Y=\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

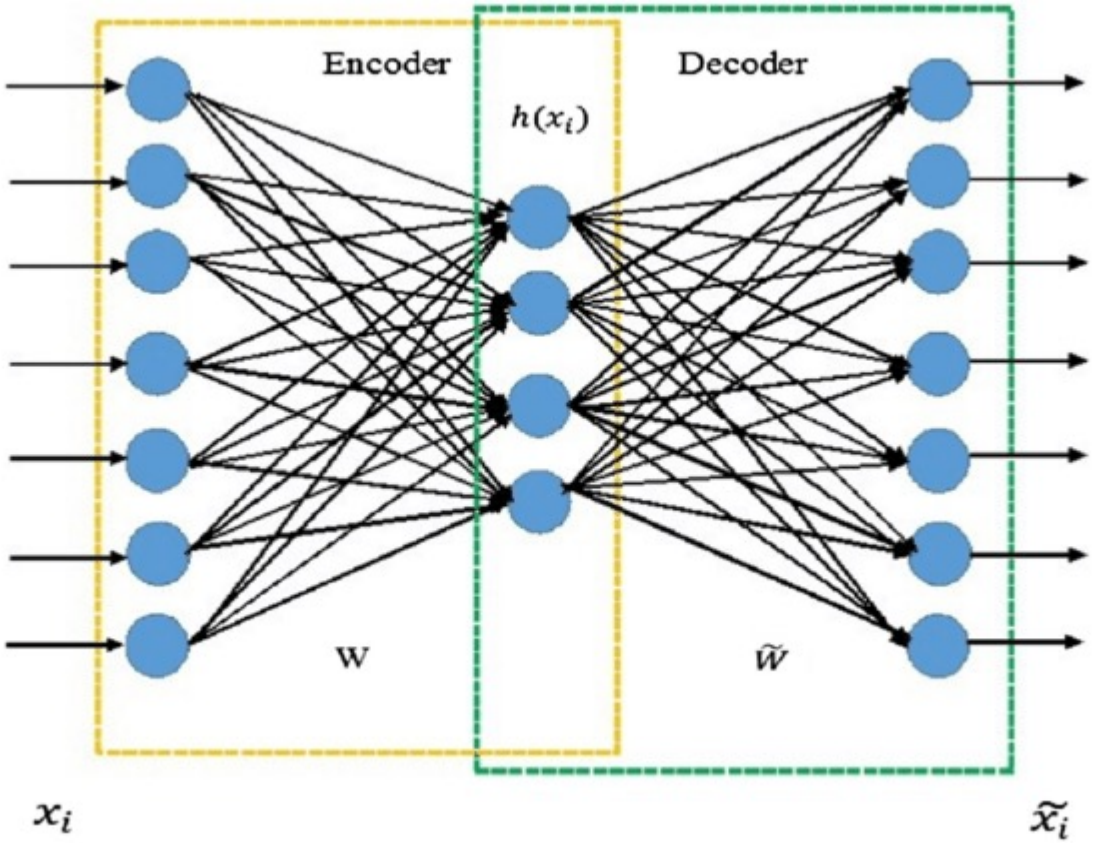
- General idea of UMAP is similar to t-SNE:
  - A high-dimensional data space X is reflected in a low-dimensional map Y
  - UMAP uses **manifold approximation** and patches together local fuzzy simplicial sets (1-simplex) to construct a topological representation of X
  - t-SNE uses **probabilities** to reflect the structure in X
  - Optimise layout of Y to minimise error between the 2 topological representations using gradient descent
- There is one y_i for every x_i
- Y only has 2 dimensions, X has arbitrary dimensions
- Just as in t-SNE, the size of clusters relative to each other is essentially meaningless
  - UMAP uses local simplex structures to construct its high-dimensional graph representation

UMAP vs. t-SNE

- Local vs. global structure: UMAP allegedly preserves as **much of the local** structure as t-SNE and **more of the global** data structure
  - Losing global structure means loss of inter-cluster relationships
- When initialised randomly, UMAP exhibits less variation in the outputs than t-SNE
  - This is due to UMAP's increased emphasis on global structure in comparison to t-SNE
  - But t-SNE can be initialised with PCA (Kobak / Berens (2019))
- UMAP: shorter runtime than t-SNE?

Autoencoders: Unsupervised Deep Learning

- General structure of an Autoencoder



Graphic from:
https://www.researchgate.net/publication/318204554/figure/fig1/AS:512595149770752@1499223615487/Autoencoder-architecture.png

Applications of Autoencoders

- Outlier Detection
  - Typically used for fraud detection
  - How can we apply outlier detection for manager selection?
  - We are interested in managers with unique strategies / return profiles
- AE are used to initialise other machine learning methods, for example, t-SNE or Deep Learning Networks

- Too few data points for the application of AE / Deep Learning in Finance?
- Like other neural networks, AE need large amounts of data for training

Parts of an Autoencoder

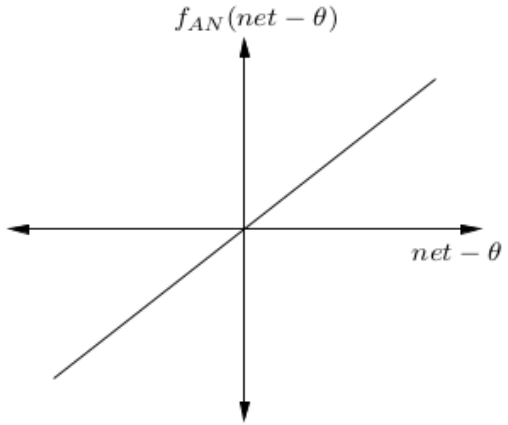
- Parts of an Autoencoder and their tasks:

1. Encoder: generate a compressed version of the input data
   - Encoding creates a more abstract description of the data → describing the latent features
2. Hidden Layer(s): “Bottleneck”, contains the compressed data (i.e., the autoencoder’s “principal components”)
3. Decoder: reconstruct the input as accurately as possible using the compressed data from the hidden layers as inputs

- Errors are minimised via Backpropagation
- Loss function is typically MSE
- Undercomplete Autoencoder: fewer units on hidden layer than on input & output layer
- See also: https://www.jeremyjordan.me/autoencoders/

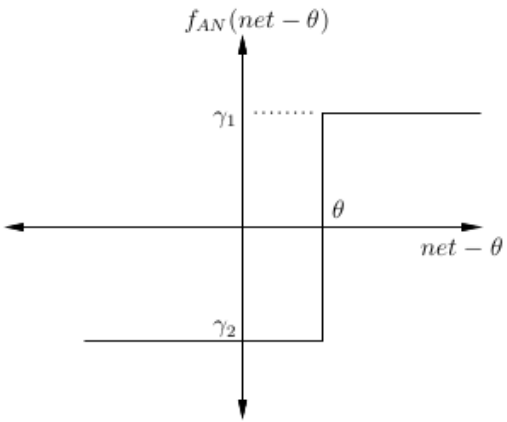
Autoencoders: Activation Functions

- Depending on the activation function in the middle h(x_i) the Autoencoder can process linear data (like PCA) …

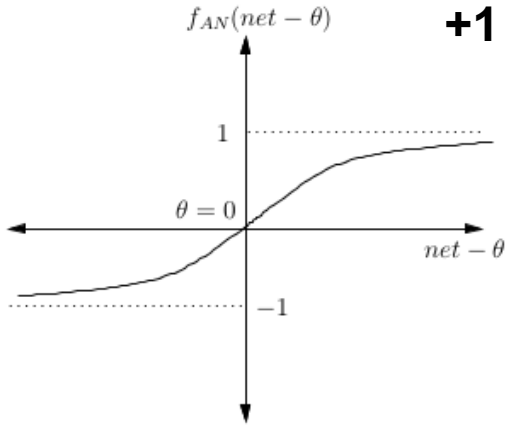


Graphs from: https://cdn-images-1.medium.com/max/1600/1*52pviFr_uDX_JeuBI01e0g.png

- … or non-linear data:



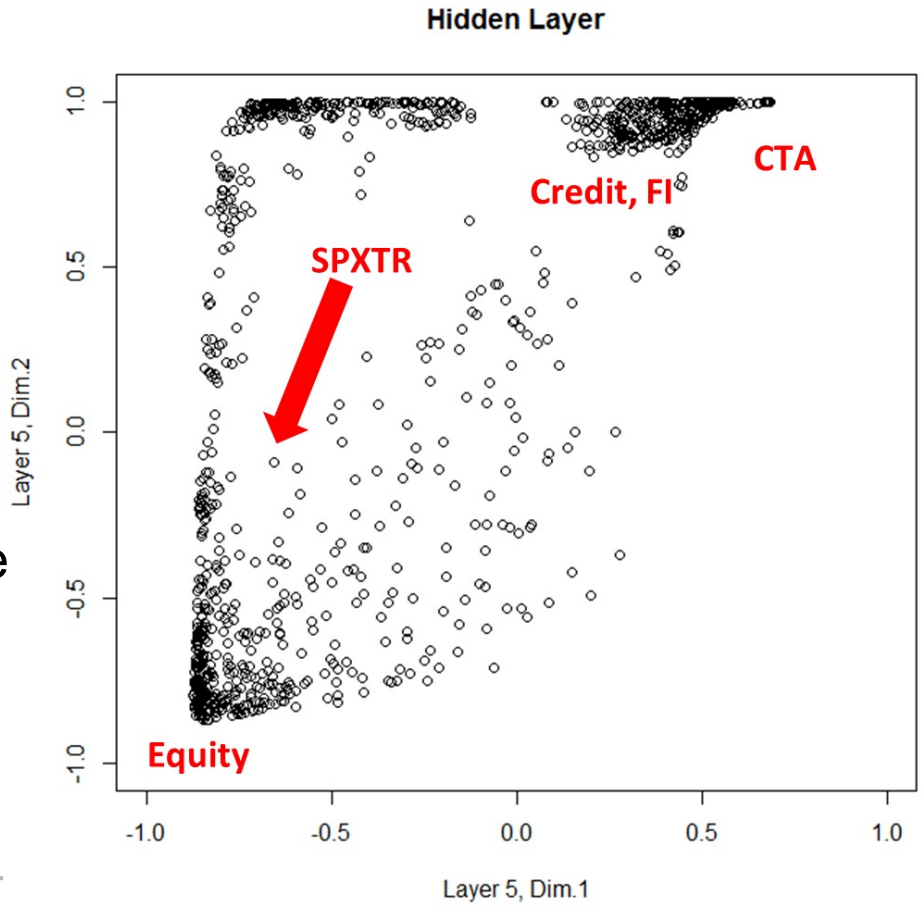
On the following slides, we always use activation function tanh: scales from -1 to +1



(e) Hyperbolic tangent function

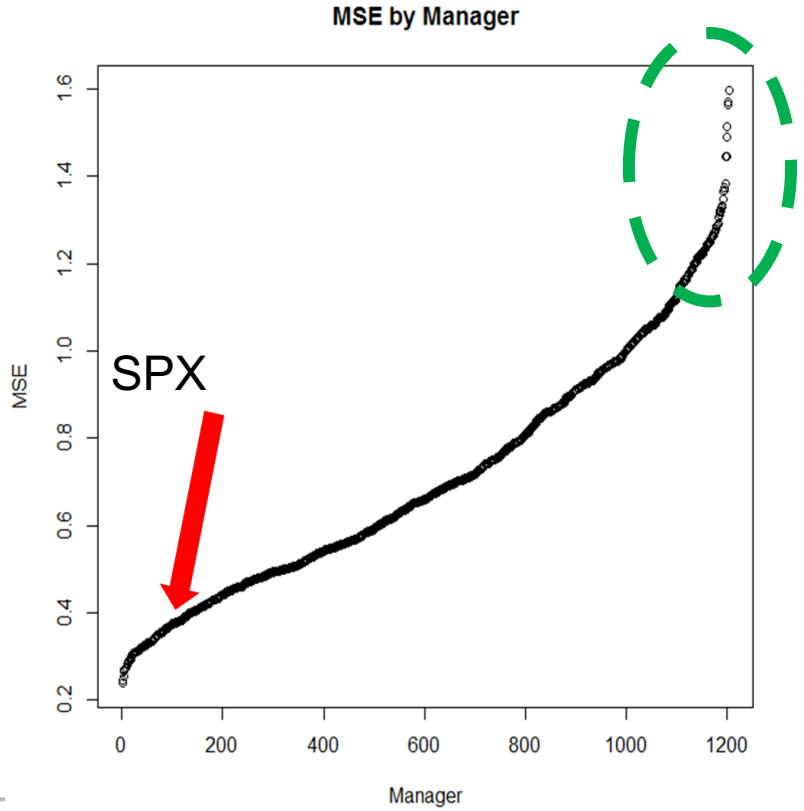
Autoencoders: Hedge Fund Style Analysis

- Input: 1070 hedge funds, 48 monthly returns
- 9 hidden layers: 48, 20, 10, 5, 3, **2**, 3, 5, 10, 20, 48
- Chart shows each hedge fund's scores of the 2-dimensional hidden layer in the middle of the AE (analogy to PCA's factor scores)
- Individual areas reflect certain hedge fund styles, e.g.:
  - Credit: top middle of the chart
  - CTA: top right corner
  - Equity in the bottom left corner



Autoencoders: Portfolio Construction

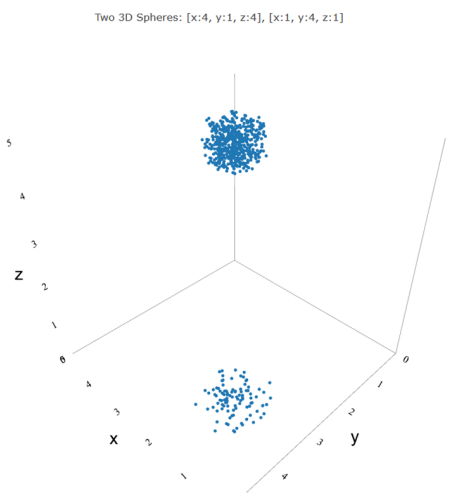
- 1070 Hedge Funds, monthly data 2004 - 2007
- Sort managers by MSE
- Managers with high MSE are unique, those close to SPX are similar
- Managers with low MSE resemble the SPX
- In order to compose a portfolio of **unique managers**, which managers would we pick?
- Portfolios composed of unique managers tend to have very low beta
- Managers with low MSE tend have long equity exposure and to exhibit high correlations with each other
- Those with high MSE tend to have little equity long exposure (however, some are CTAs with directional equity exposure), also Fixed Income and Option



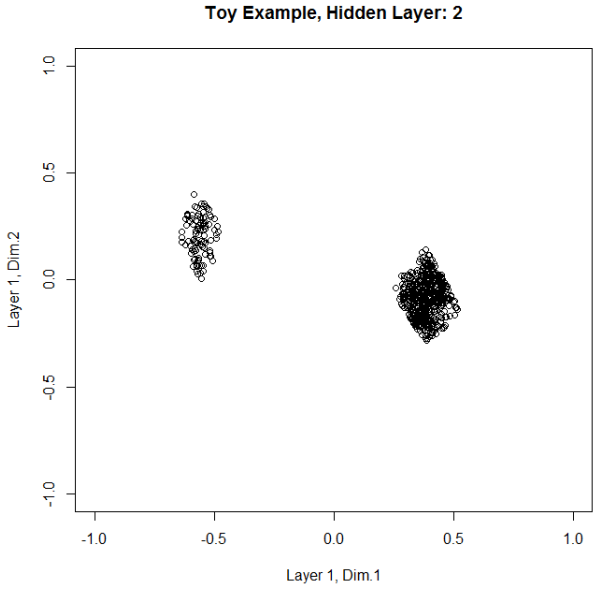
Autoencoders

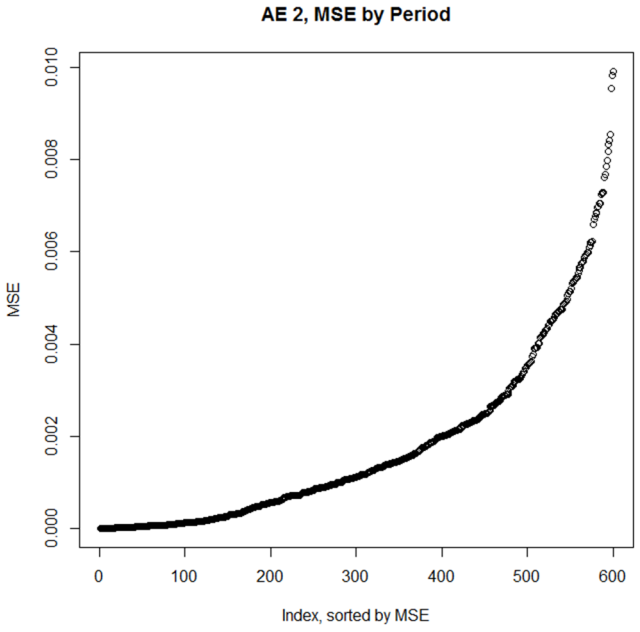
- AE can be seen as some kind of non-linear PCA
- AE with a linear activation function → identical results to PCA
- The encoding part of the AE can be seen as the feature layer which can transform raw data from the input layer to a lower-dimensional representation in the hidden layer
- A feature is defined by the pattern of weights of all connections between a specific hidden layer node and all input nodes (global function approximator)
- AE are often used as step to pre-process data for later input to a supervised learning method
- What are PCA's factor loadings (i.e., elements of the principal components)? → weights of the Neural Network

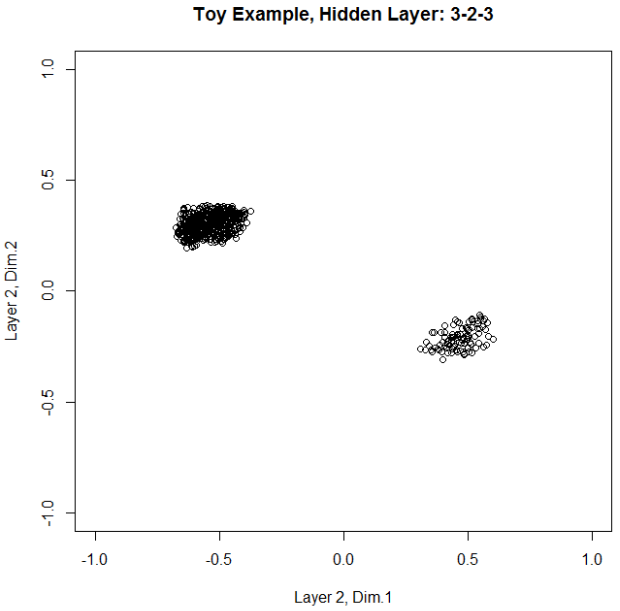
Autoencoder: Toy Example 2 Spheres



- 3 input dimensions, 1000 training epochs
- Run with h2o, see R code “R 3D Plot Toy Example.R”
- Left side: 1 hidden layer with 2 units
  - AE is able to separate the 2 clusters
  - Middle part: 600 data points sorted by MSE. Largest outliers are from both spheres
- Right side: 3 hidden layers (3, 2, 3)
  - No obvious improvement by adding hidden layers



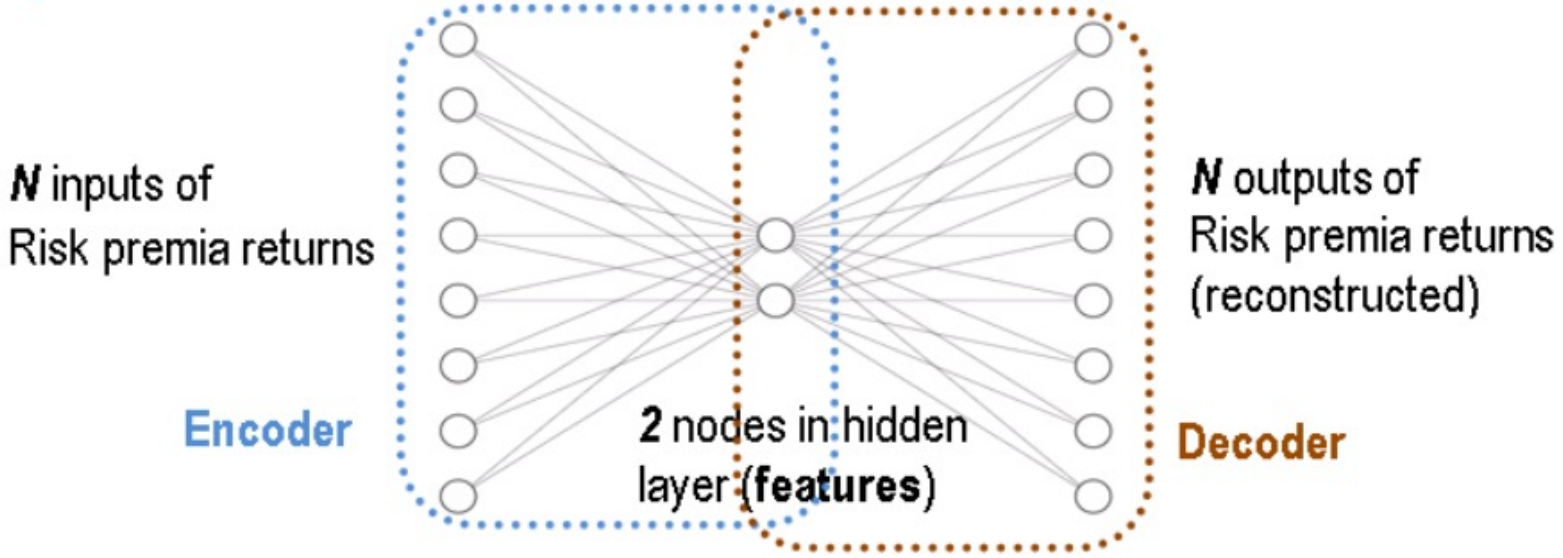




Autoencoders for Portfolio Construction of Risk Premia

- JP Morgan (2020): use AE to divide a set of 23 risk premia into 2 clusters: Aggressive and Defensive
- Risk Premia are investment vehicles that try to harvest the premia from investment strategies like Equity Value, Momentum, Volatility, Low Vol, FX Carry, etc.
- Feed historical data for the risk premia (e.g., T x N = 104 weekly returns x 23) to the AE
- Each risk premium is assigned to 1 of the 2 clusters

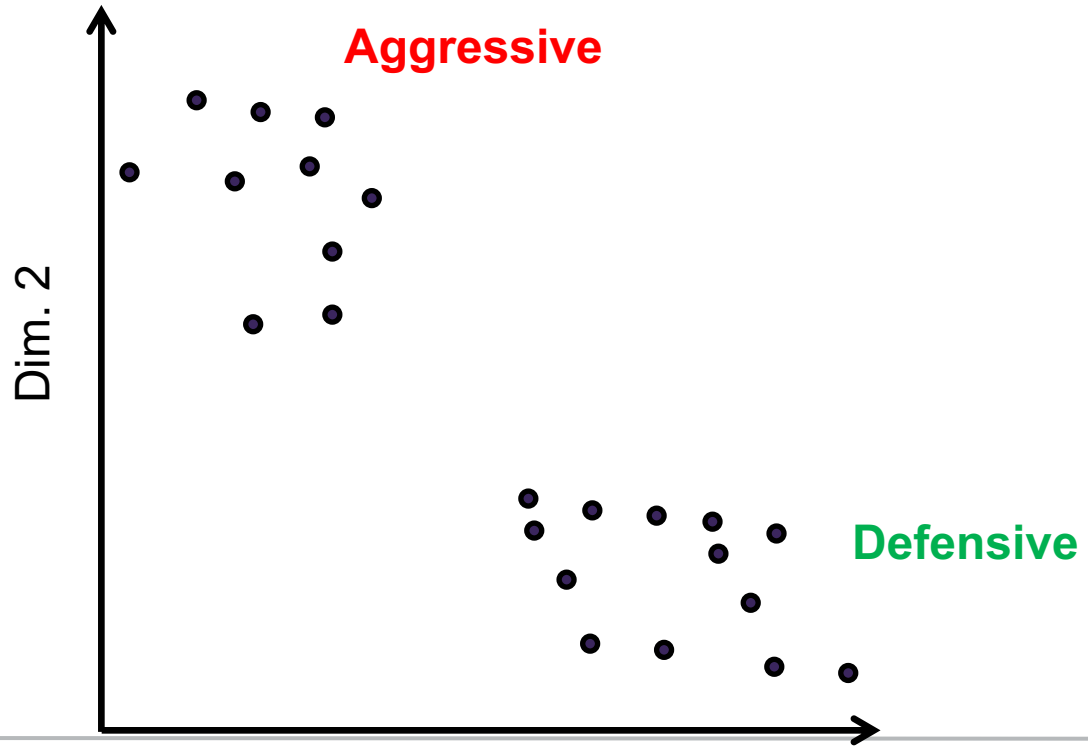
Figure 18: Autoencoders extract features using the hidden layer(s).



Source: J.P. Morgan Quantitative and Derivatives Strategy

Autoencoders for Portfolio Construction of Risk Premia

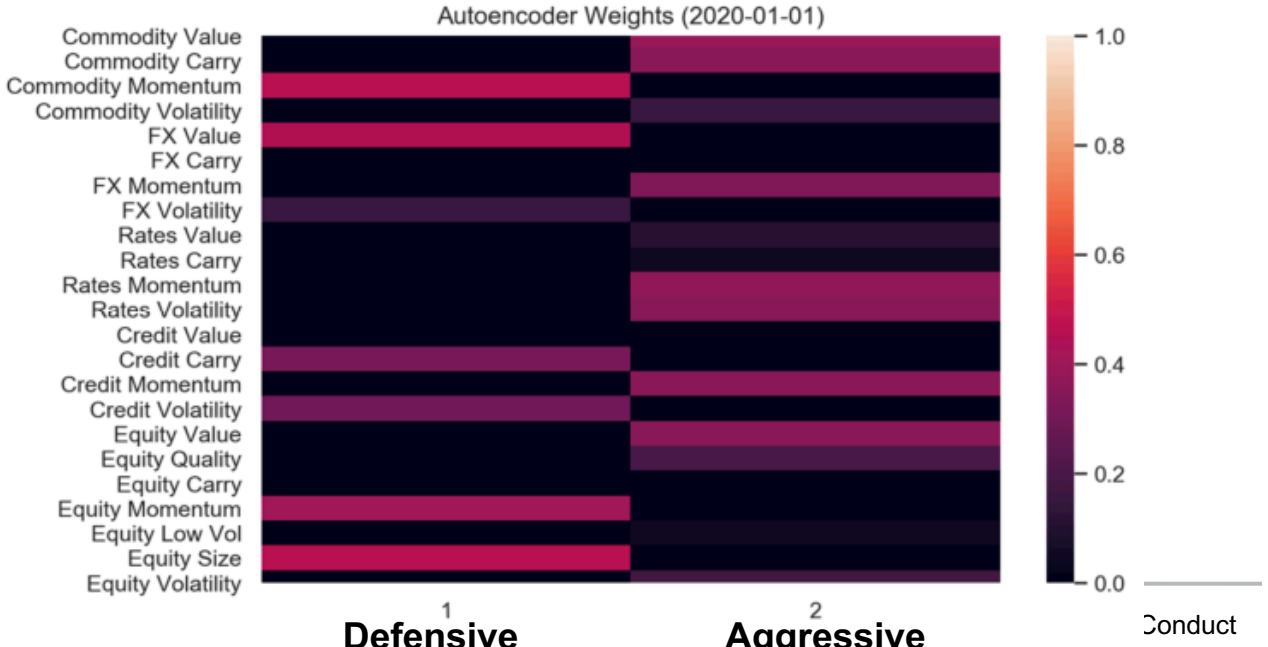
- The 2 dimensions of the hidden layer displayed visually
- Each dot in the chart represents 1 risk premium
- *Stylised* example for 1 point in time, e.g., 2020-01-01
  - Analogy to PCA Factor Scores



Autoencoders for Portfolio Construction of Risk Premia

- Chart shows the 2 clusters and weights assigned by AE to each of the risk premia
  - AE weights are restricted to be long only, i.e., weights $0 \leq w \leq 1$.
- Cluster 1: Defensive (e.g., FX Value), cluster 2: Aggressive (e.g., FX Momentum)
- Build portfolio by equal risk-weighting Aggressive & Defensive: each style gets 50% of the risk budget
  - Risk budget is then distributed across the risk premia in each of the assigned clusters → result is a balanced portfolio

Figure 20: Encoder weights on each risk premia for the 2 clusters



Source: J.P. Morgan Quantitative and Derivatives Strategy

How Do I Find the "Best" Model?

- Global and local measures:
- CPD (**c**orrelation-based **p**airwise **d**istance, **global** measure): Spearman rank correlation between pairwise distances in the high-dimensional space and in the embedding (Kobak / Berens (2019))
  - For each data point, calculate the (Euclidean) distances to all other points for each X and Y → 2 distance matrices with the same dimensions N x N

Euclidean Distance Matrix X					
	1	2	3	...	N
1	0.00	0.10	0.09	...	3.11
2	0.10	0.00	0.15	...	3.04
3	0.09	0.15	0.00	...	3.12
⋮	⋮	⋮	⋮		⋮
N	3.11	3.04	3.12	...	0.00
Sum	637.31	621.24	643.18	...	965.07
rank X	172	196	164	...	5

Euclidean Distance Matrix Y					
	1	2	3	...	N
1	0.00	0.29	0.56	...	24.86
2	0.29	0.00	0.68	...	25.02
3	0.56	0.68	0.00	...	24.34
⋮	⋮	⋮	⋮		⋮
N	24.86	25.02	24.34	...	0.00
Sum	5163.30	5173.47	4985.44	...	7842.60
rank Y	174	171	211	...	1

  - For each column of the 2 distance matrices, calculate the sum of the distances. This gives 1 vector with N elements for each distance matrix, together 2 vectors

How Do I Find the "Best" Model?

- If the global structure is well preserved, the ranks of those distances should be about the same for both X and Y → calculate the Spearman rank correlation between the 2 vectors
  - High rank correlation → good approximation of X by Y
  - The higher CPD, the more global structure is preserved
  - CPD is a **global** measure
  - See supplementary R code "R tSNE test & Goodness of Fit, CQF.R"

How Do I Find the "Best" Model?

- KNN (**local** measure): The fraction of k-nearest neighbours in the original high dimensional data that are preserved as k-nearest neighbours in the model output (Kobak/Berens (2019) and Bibal / Frénay (2016))
  - For each data point, calculate the (Euclidean) distances to its K nearest neighbours for each X and Y
  - KNN is the fraction of k-nearest neighbours in X that are preserved as k-nearest neighbours in Y
  - KNN quantifies preservation of the **local** structure,
  - The higher KNN, the more local structure is preserved
  - Kobak / Berens (2019) suggest K = 10

How Do I Find the "Best" Model?

Example in table (K=4):

- For example, the first 4 rows in column 1 mean that for the 1st element in **X** the 4 nearest neighbours are elements 7, 5, 21, and 8
- The second 4 rows in column 1 mean that for the 1st element in **Y** the 4 nearest neighbours are 8, 6, 5, 23
- As 5 and 8 appear as NN in both X and Y, KNN(1) = 2 / 4 = 0.5
- **KNN = Sum(KNN(j)) / N = (0.5 + 0.25 + … + 0.25) / N**

Example:

		1	2	…	N
X	1	7	7	…	644
	2	5	90	…	135
	3	21	91	…	88
	4	8	3	…	345
Y	1	8	87	…	17
	2	6	7	…	88
	3	5	17	…	131
	4	23	98	…	99
KNN(j)		**0.5**	**0.25**	…	**0.25**

- Contrary to CPD, KNN looks at the **immediate** neighbourhood
- For our 3D toy example, we get:

	t-SNE	UMAP
CPD	0.57	0.35
KNN (K = 10)	0.03	0.11

→ t-SNE seems to preserve the global structure better, while UMAP keeps the neighbourhood more closely together

Criteria for choosing a specific method for UML (Sarlin (2015))

- Form of structure preservation
  - Which methods better assure trustworthy neighbours?
- Computational Cost
  - Method becomes unwieldy if computations take too much time
- Flexibility for problematic data
  - Missing values, non-normal data
- Shape of the Output

Summary

- Unsupervised ML methods useful for:
  - Dimension reduction
  - Noise reduction
  - Understanding the structure of the data
- t-SNE, UMAP were designed for visualisation, AE's hidden layers can be used for that as well
- All 3 are recent methods of UML
- t-SNE, UMAP & AE try to mirror original high-dimensional data in low-dimensional space
  - t-SNE (local approximator) builds on determining probabilities of points that belong together
  - UMAP (local approximator) is based on the mathematical foundation of simplicial sets
  - AE (global approximator) is a deep learning methodology
- ML tools are powerful, but require caution and experience

References

- Belkina, A. C., Ciccolella, C. O., Anno, R., Halpert, R., Spidlen, J., & Snyder-Cappione, J. E. (2019). Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. Nature communications, 10(1), 1-12.
- Bibal, A., & Frénay, B. (2016, April). Interpretability of machine learning models and representations: an introduction. In ESANN.
- De Bodt, C., Mulders, D., Verleysen, M., & Lee, J. A. (2018). Perplexity-free t-SNE and twice Student tt-SNE. In ESANN.
- Efimov, D., Xu, D., Kong, L., Nefedov, A., & Anandakrishnan, A. (2020). Using generative adversarial networks to synthesize artificial financial datasets. arXiv preprint arXiv:2002.02271.
- Friedman, G. (2016). An elementary illustrated introduction to simplicial sets. arXiv preprint arXiv:0809.4221.
- Greengard, P., Liu, Y., Steinerberger, S., & Tsyvinski, A. (2020). Factor Clustering with t-SNE. Available at SSRN.

References

- Husmann, S., Shivarova, A., & Steinert, R. (2020). Company classification using machine learning. preprint arXiv:2004.01496.
- JP Morgan (2020): Quantitative Perspectives on Cross-Asset Risk Premia, Global Quantitative & Derivatives Strategy, 09 June 2020
- Kobak, D., & Berens, P. (2019). The art of using t-SNE for single-cell transcriptomics. Nature communications, 10(1), 1-14.
- McInnes, L., Healy, J., & Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. preprint arXiv:1802.03426.
- Sarlin, P. (2015). Data and dimension reduction for visual financial performance analysis. Information Visualization, 14(2), 148-167.Van Der Maaten, L. (2013). Barnes-hut-sne. arXiv preprint arXiv:1301.3342.
- Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.
- Van Der Maaten, L. (2009, April). Learning a parametric embedding by preserving local structure. In Artificial Intelligence and Statistics (pp. 384-391).