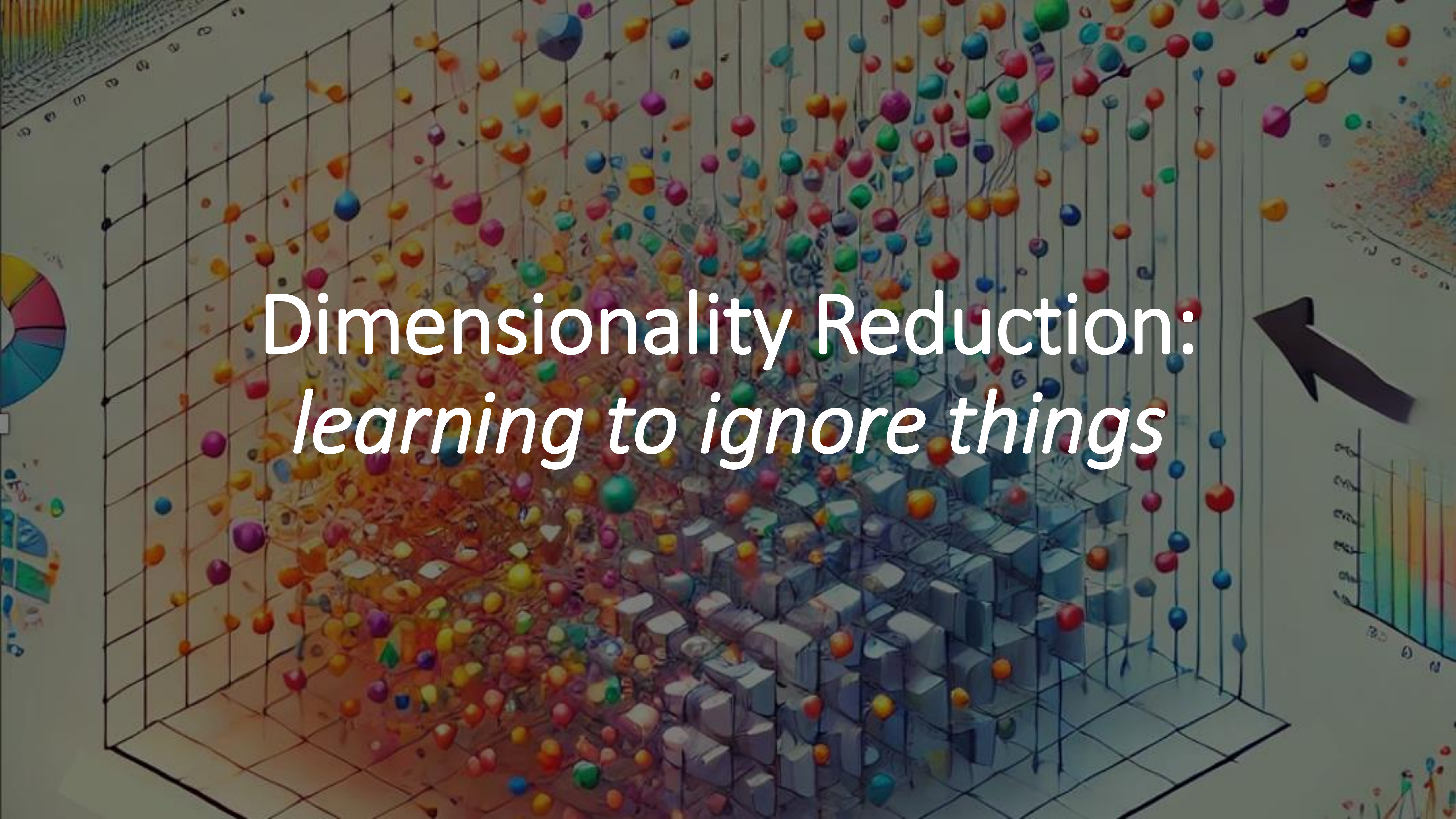


Dimensionality Reduction: *learning to ignore things*

Why should we use dimensionality reduction?

Features/Input ***Output***

$$[x_1, x_2, \ldots, x_{10,000}] \longrightarrow [y_1]$$

Do we need all of these features to accurately predict the output?
Can we get away with only using a subset of the feature space?

Considerations

- Model trains faster since it has fewer dimensions.
- Make the model simpler for researchers to interpret/visualize.
- Improve model accuracy due to less misleading or noisy data.

Types of Dimensionality Reduction Techniques

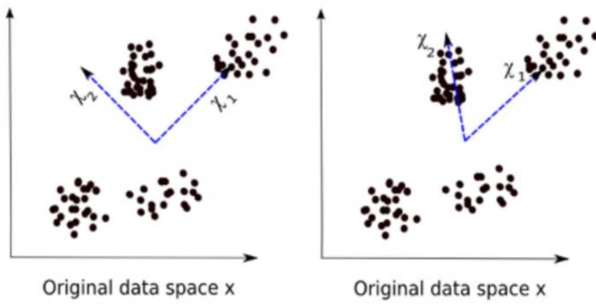
Most DR techniques fall into a few categories

- ***Feature Elimination and Extraction***

  we have been working with this already a bit
  in general, the goal is to systematically remove uninformative/redundant variables

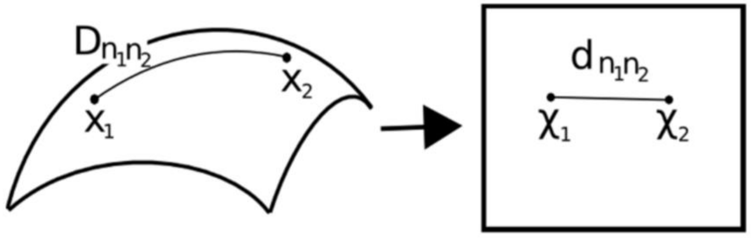
- ***Linear Transformations/Components***

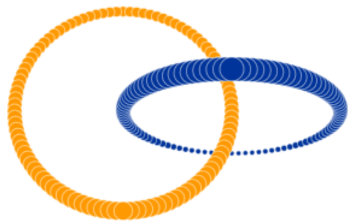
  here the strategy is to find a linear transformation of your feature space that results in more "useful" set of coordinates



- ***Non-linear Transformations & manifolds***

  in this case, we usually perform some non-linear transformation and project the features onto a lower-dimensional manifold





Not linearly separable!

Feature Elimination and Extraction

From a feature set, we are looking for a subset of features to use:

- *Remove features with too many missing values* ***(Missing Value Ratio)***
- *Remove features that exhibit small variance* ***(Low-variance Filter)***
- *Remove highly correlated features (e.g., using Pearson's r)* ***(High Correlation filter)***
- *Assess* ***feature importance*** *to model predictions*
  - *by using e.g.,* ***Random Forest, SHAP***
  - *by systematically removing features (****Backward Feature Elimination)***
  - *by systematically adding features (****Forward Feature Selection)***

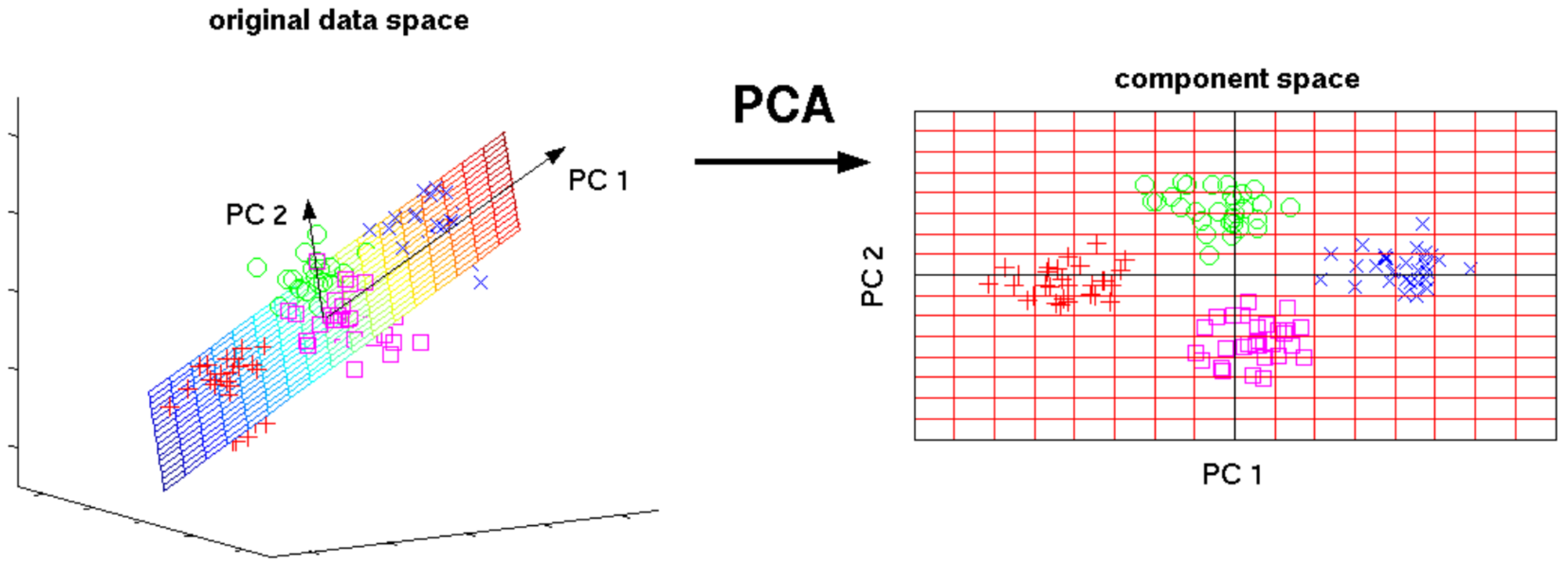
These are all very simple, powerful strategies that may be complementary to other methods → always worth considering!

Principal Component Analysis (PCA)

PCA is the “canonical” and most common DR technique

Important features of PCA

- *PCA assumes linear relationships between variables.*
- *PCA is scale dependent (features with larger values look more important)*
- *PCA looks at variance in the feature data (it can be important to preprocess the data via normalization, mean-centering, or scaling).*
- *PCA is one of the central applications of SVD*
- *PCA can be used for preprocessing for other DR techniques*



The basic goal:

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_p; \boldsymbol{x}_i \in \mathbb{R}^d$$

$$\Big\downarrow \boldsymbol{W}$$

$$\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_p; \boldsymbol{y}_i \in \mathbb{R}^k$$

$$k \ll d$$

Principal Component Analysis (PCA)

The big idea:

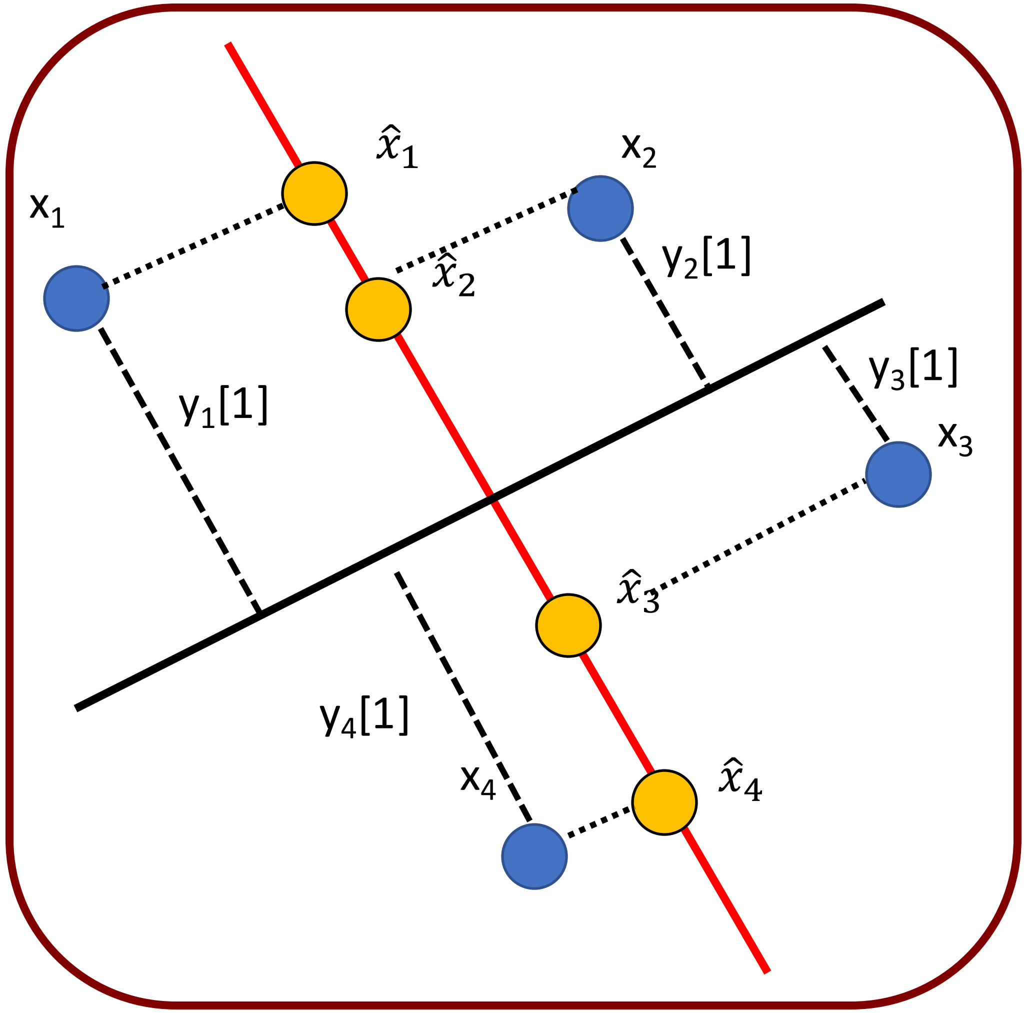
1. We want to identify an orthonormal basis to represent $\boldsymbol{x}_i - \boldsymbol{\mu}$ → d-dimensional basis
2. We will only pick *k* of the *d* basis vectors
3. All of the points will then be represented in a k-dimensional subspace spanned by $w_1, \ldots, w_k$

$$\boldsymbol{x}_i - \boldsymbol{\mu} = \sum_{j=1}^{d} \boldsymbol{y}_i[j]\boldsymbol{w}_j$$

exact reconstruction

$$\hat{\boldsymbol{x}}_i - \boldsymbol{\mu} = \sum_{j=1}^{k} \boldsymbol{y}_i[j]\boldsymbol{w}_j$$

approximate reconstruction

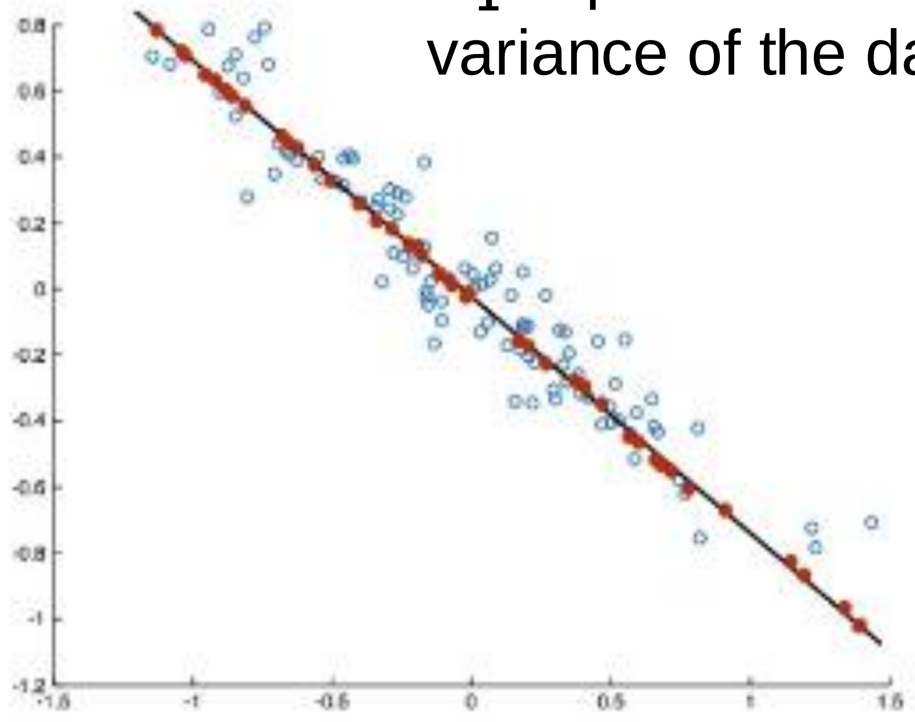


*Algorithm → how to choose **W** and which to keep*

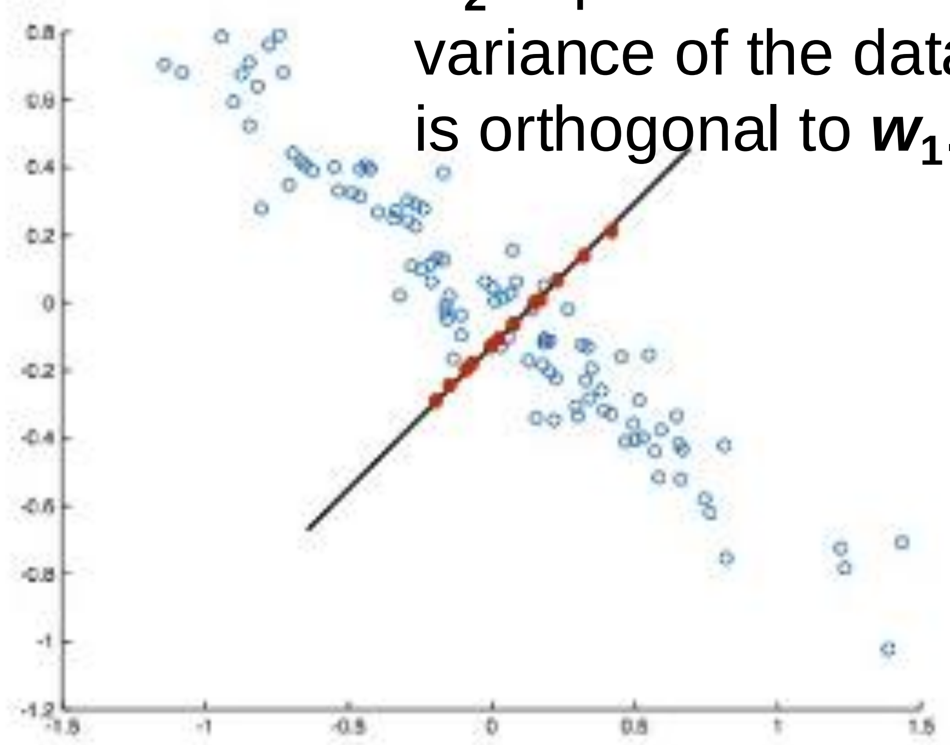
PCA: How to pick basis vectors

Idea: Find the directions along which the data is maximally spread (highest variance).

$\boldsymbol{w_1}$ captures the most variance of the data.



$\boldsymbol{w_2}$ captures the 2nd most variance of the data and is orthogonal to $\boldsymbol{w_1}$.



https://www.cs.cornell.edu/courses/cs4786/2020sp/

PCA: How to pick basis vectors

Idea: Find the directions along which the data is maximally spread (highest variance).

$$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_p]$$

$$\boldsymbol{\mu} = \left[\frac{1}{p}\sum_{i=1}^{p} \boldsymbol{x}_i[1], \cdots, \frac{1}{p}\sum_{i=1}^{p} \boldsymbol{x}_i[d]\right]^T$$

$$\boldsymbol{B} = \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{1}_d^T$$

- mean-subtracted data

$$\boldsymbol{C} = \frac{1}{n-1}\boldsymbol{B}^T\boldsymbol{B}$$

- covariance matrix

$$\boldsymbol{w}_1 = \underset{||\boldsymbol{w}_1||=1}{\arg\max}\, \boldsymbol{w}_1^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{w}_1$$

- first principal component

subsequent components can be determined by orthogonalization and repetition

PCA: How to pick basis vectors

Idea: Find the directions along which the data is maximally spread (highest variance).

$$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_p]$$

$$\boldsymbol{\mu} = \left[\frac{1}{p}\sum_{i=1}^{p}\boldsymbol{x}_i[1], \cdots, \frac{1}{p}\sum_{i=1}^{p}\boldsymbol{x}_i[d]\right]^T$$

$$\boldsymbol{B} = \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{1}_d^T$$

- mean-subtracted data

$$\boldsymbol{C} = \frac{1}{n-1}\boldsymbol{B}^T\boldsymbol{B}$$

- covariance matrix

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^T$$

Or you can get them all in one go by doing the SVD or some eigendecomposition

PCA: additional details/realizations

- We want to maximize this function subject to the constraint that the norm of ***w*** is 1. We can do this with Lagrange multipliers

$$w_1 = \arg max \, w^T \Sigma w - \lambda \|w\|_2^2$$

- We now take the derivative of this with respect to w equate to 0, and arrive at

$$\Sigma w_1 - \lambda \, w_1 = 0$$

- This is an eigenvalue equation! The direction w_1 that we obtain by maximizing the variance in the direction is some unit vector that satisfies this eigenvalue equation. We want to maximize $w^T \Sigma w$, so if we plug in the eigenvalue solution we get that

$$w^T \Sigma w = \lambda \|w_1\|_2^2 = \lambda$$

So the eigenvector w_1 that maximizes the variance is the one with the largest eigenvalue!

- We can proceed in similar fashion to pick eigenvectors with the second, third, etc largest eigenvalues to form this basis.

Principal components analysis then amounts to simply finding the eigenvectors of the covariance matrix Σ!

PCA: basic Python implementation

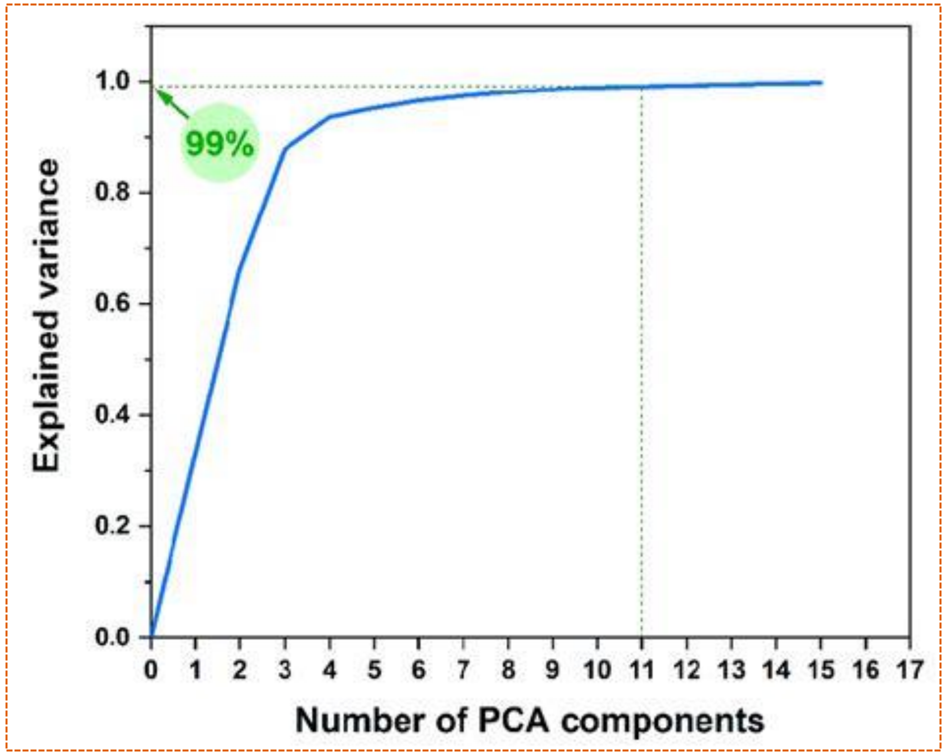
```python
def pca(x,n_components):
    #Vanilla PCA
    #subtract off the mean of the data
    x_mean = x - np.mean(x, axis=0)

    #Calculate the covariance matrix
    #Numpy as this calculation automated
    cov_mat = np.cov(x_mean, rowvar = False)

    #Solve for eigensystem of covariance matrix
    eval,evec = eigh(cov_mat)

    #Sort eigenvalues and eigenvectors
    sort_i = np.argsort(eval)[::-1]
    sort_eval = eval[sort_i]
    sort_evec = evec[:,sort_i]

    #Now take the first "n_components" principal components
    eval_subset = sort_eval[0:n_components]
    evec_subset = sort_evec[:,0:n_components]

    #Project dataset onto principal components
    x_transform = np.dot(evec_subset.transpose(), x_mean.transpose()).transpose()

    return x_transform, evec_subset, eval_subset
```

PCA: how much of the data is explained?

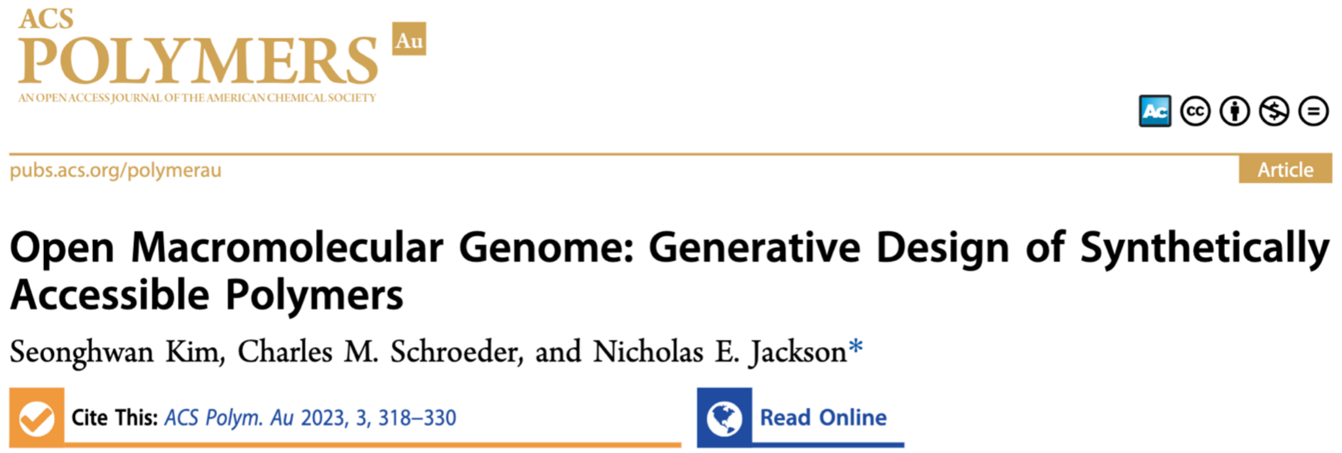
It is natural to ask how much of the information in a data set is lost by projecting the observations onto a small subset of principal components.

The amount of variance explained can be analytically related to the eigenvalues of the covariance matrix via:

$$\text{Explained Variance Ratio} = \frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$$



PCA: Application to Visualizing Chemical Space



ACS POLYMERS Au

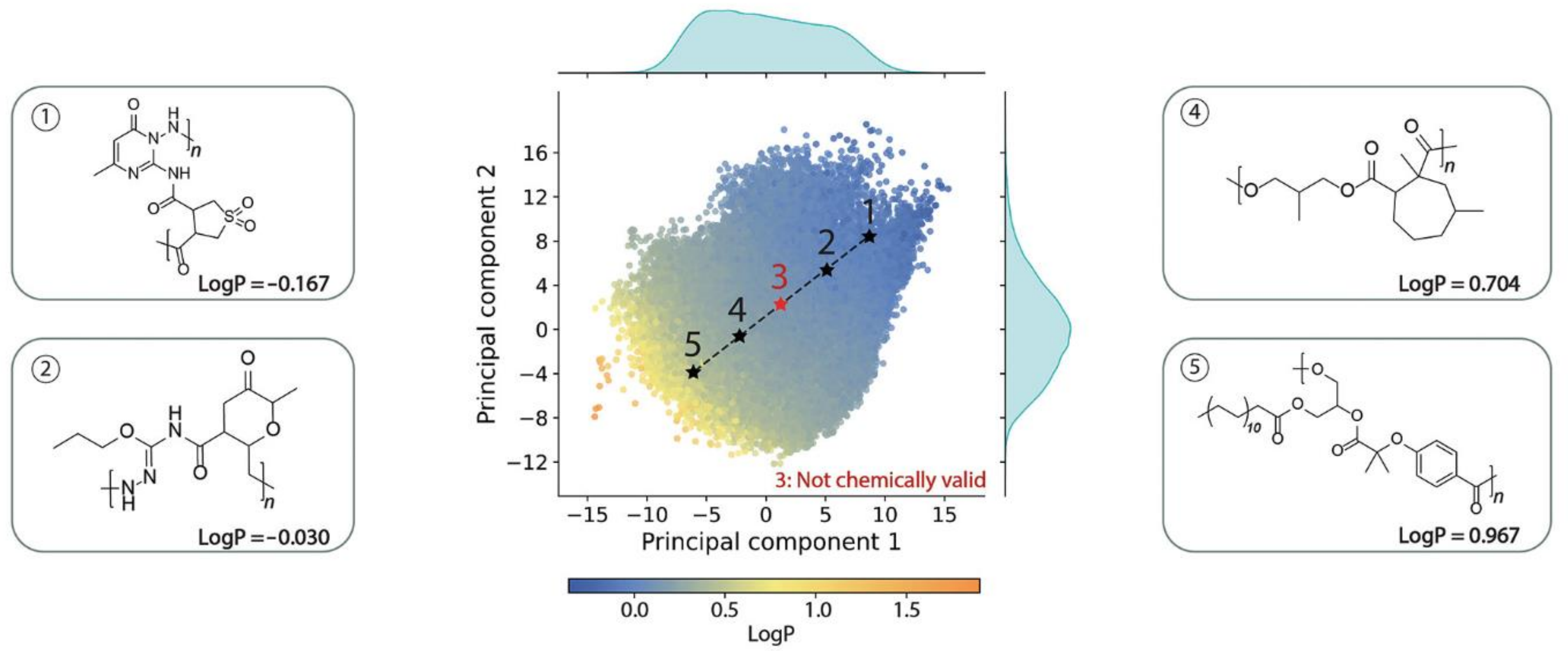
AN OPEN ACCESS JOURNAL OF THE AMERICAN CHEMICAL SOCIETY

pubs.acs.org/polymerau Article

Open Macromolecular Genome: Generative Design of Synthetically Accessible Polymers

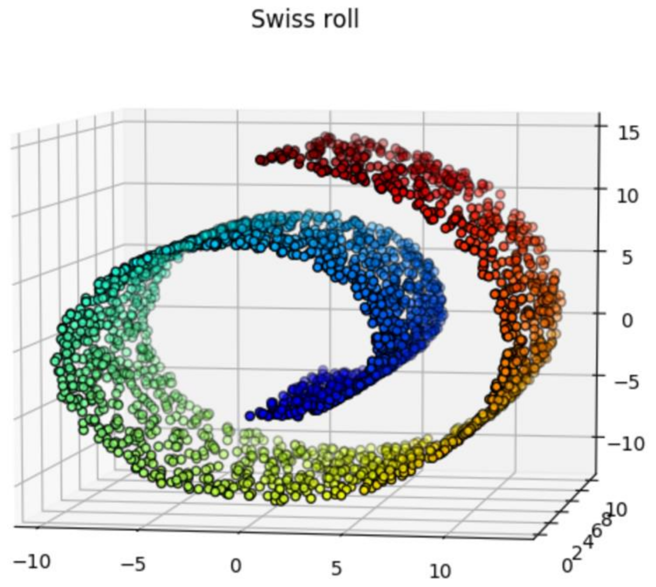
Seonghwan Kim, Charles M. Schroeder, and Nicholas E. Jackson*

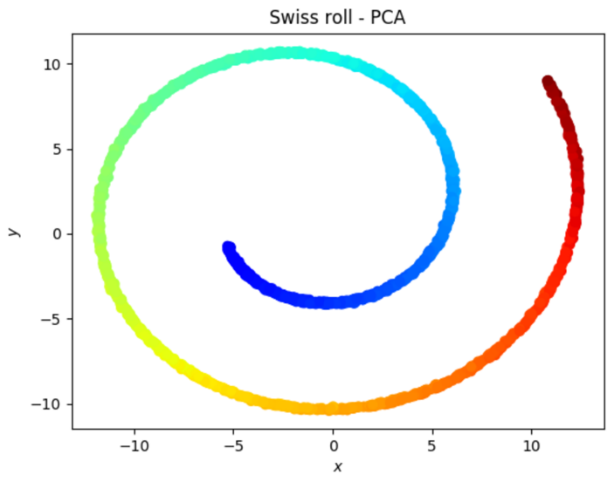
Cite This: *ACS Polym. Au* 2023, 3, 318–330 **Read Online**

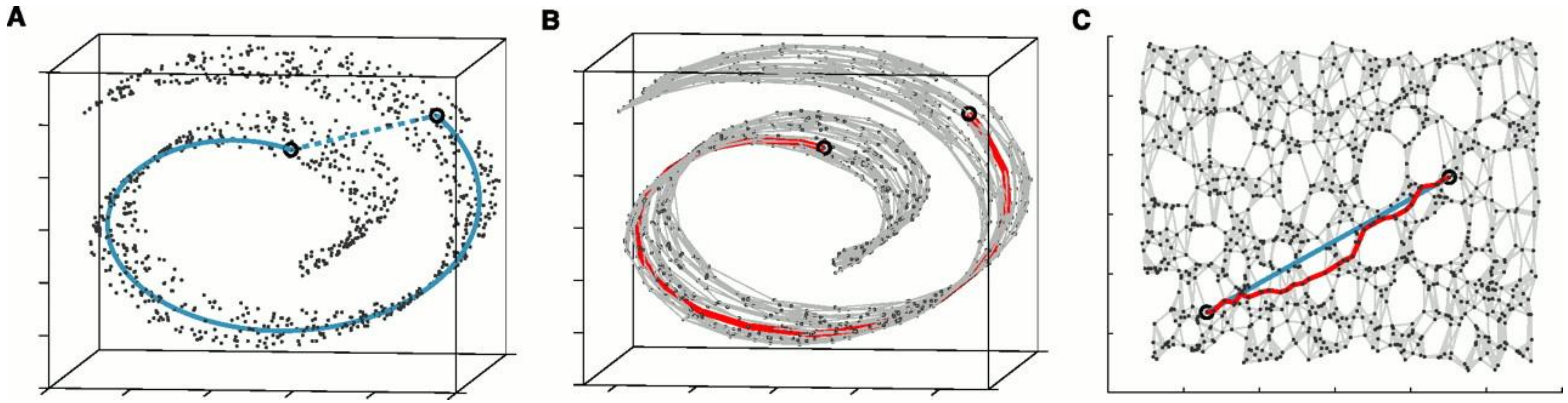


Correcting for failures of PCA

PCA generally fails to detect low-dimensional manifolds



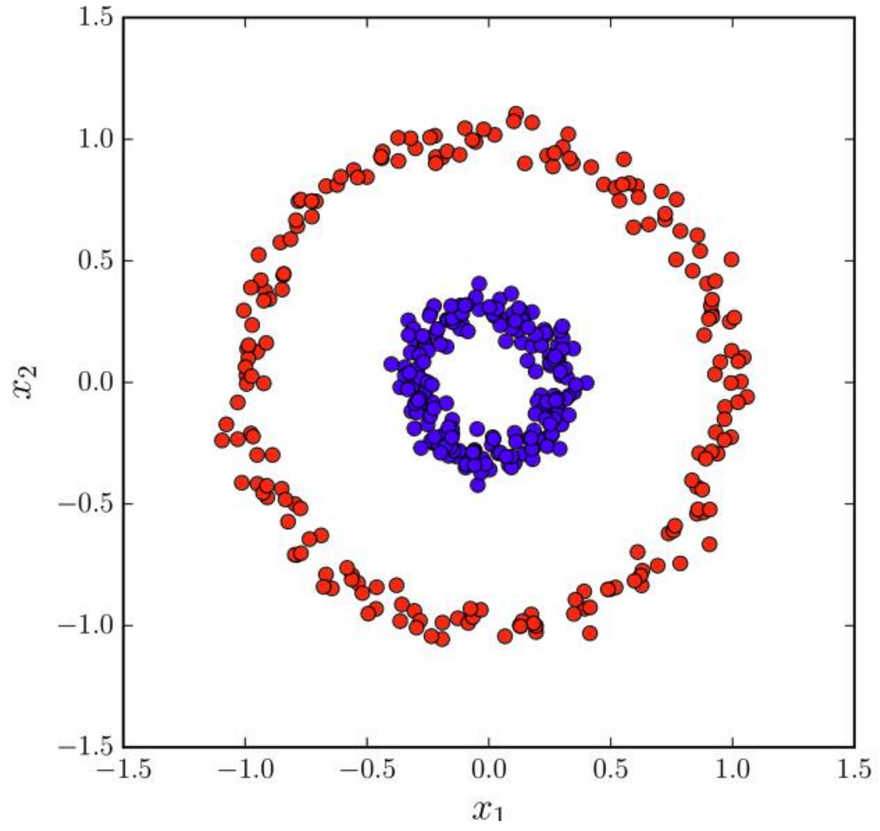




Tenenbaum, J. B.; de Silva, V. & Langford, J. C. A Global Geometric Framework for Nonlinear Dimensionality Reduction *Science*, **2000**, *290*, 2319-2323

A (relatively) Simple Modification: Kernel PCA

Consider the following…



What happens if we do PCA here?

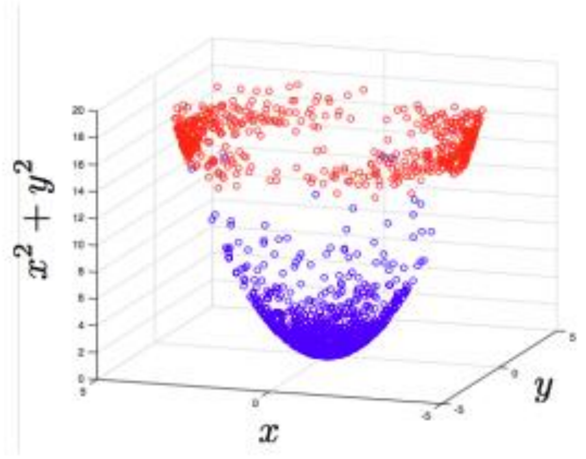
The Kernel Trick

- effectively project the data into a *higher*-dimensional space
- this usually enables some further resolution of data
- the kernel trick provides an efficient approach to the data transformation
- specifically, it allows for computations of distances in the higher-dimensional feature space but never actually performs the coordinate transformation

Let $\phi : \mathcal{V}_{\text{old}} \to \mathcal{V}_{\text{new}}$

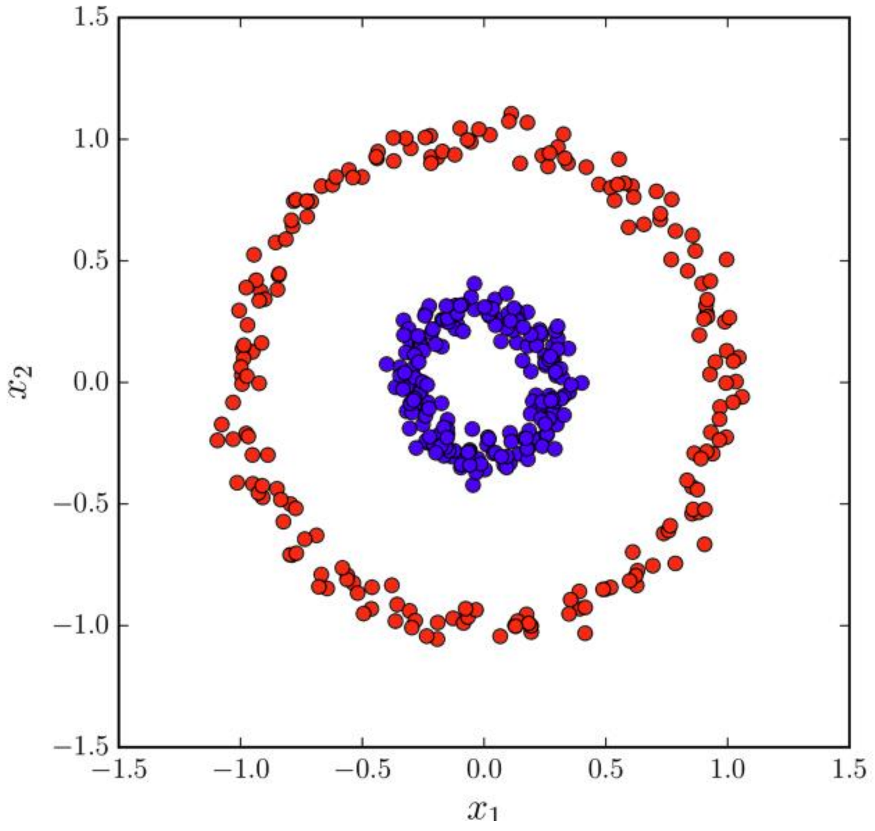
Then, $k(\boldsymbol{x}, \boldsymbol{y}) := \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{V}_{\text{new}}}$

this is a non-trivial but somewhat arbitrary function



A (relatively) Simple Modification: Kernel PCA

Consider the following…



What happens if we do PCA here?

The Kernel Trick

- effectively project the data into a *higher*-dimensional space
- this usually enables some further resolution of data
- the kernel trick provides an efficient approach to the data transformation
- specifically, it allows for computations of distances in the higher-dimensional feature space but never actually performs the coordinate transformation

Common kernels:

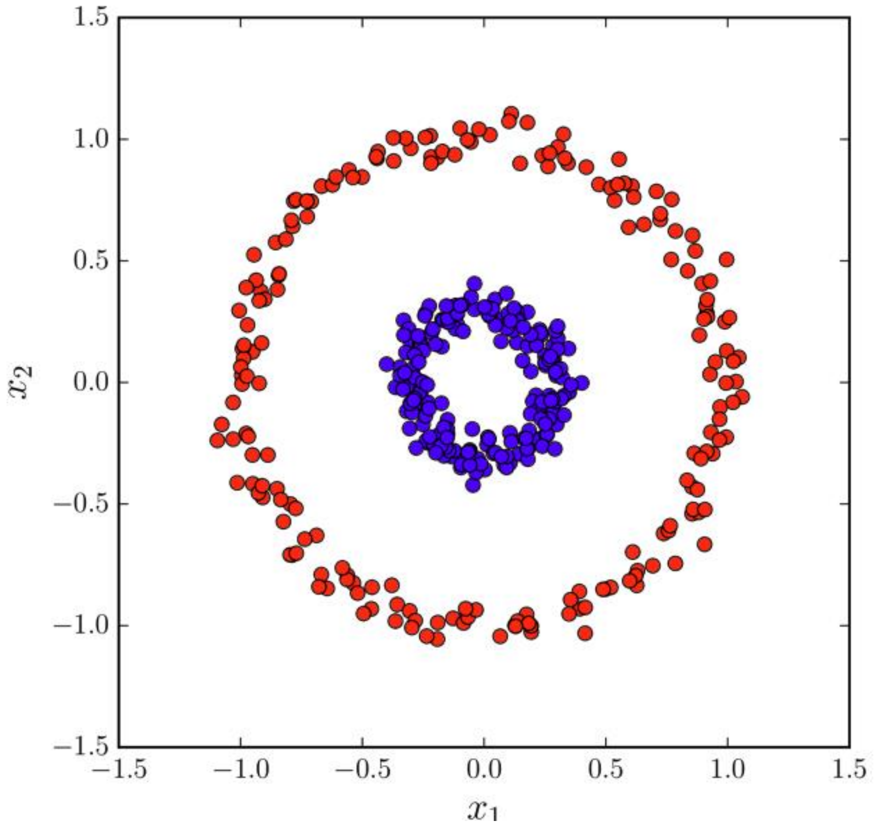
$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + 1)^d$$

$$k(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma ||\boldsymbol{x}-\boldsymbol{y}||^2}$$

Note: Care must be taken to use zero-mean data in KPCA

A (relatively) Simple Modification: Kernel PCA

Consider the following…



What happens if we do PCA here?

Algorithm to implement Kernel PCA:

1. Compute the Kernel matrix, K.

$$K = \begin{bmatrix} \kappa(x^{(1)},\, x^{(1)}) & \cdots & \kappa(x^{(1)},\, x^{(n)}) \\ \vdots & \ddots & \vdots \\ \kappa(x^{(n)},\, x^{(1)}) & \cdots & \kappa(x^{(n)},\, x^{(n)}) \end{bmatrix}$$

2. Center the kernel matrix (like normal feature scaling, but in kernel space) using

$$K' = K - 1_n K - K 1_n + 1_n K 1_n$$

1_n= *n*x*n* matrix with all elements equal to 1/*n*.

3. Solve *K'v* = λ*v* and extract the top k eigenvectors.

Kernel principal component analysis, B. Scholkopf, A. Smola, and K.R. Muller, 583-588, 1997.

Other Greatest Hits of Dimensionality Reduction

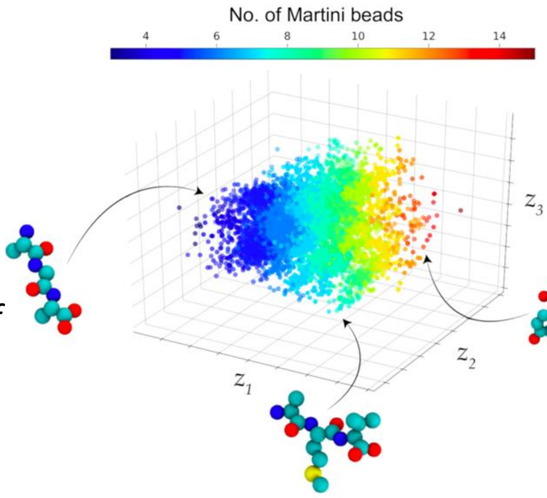
- Linear Discriminant Analysis
- Generalized Discrimination Analysis
- **UMAP**
- Non-negative Matrix Factorization
- Classical Scaling
- Maximum Variance Unfolding
- **Diffusion Maps**
- Locally Linear Embedding
- **t-distributed stochastic neighbor embedding (t-SNE)**
- Laplacian Eigenmaps
- Hessian LLE
- Local Tangent Space Analysis
- Sammon Mapping
- Multilayer Autoencoders
- Locally Linear Coordination
- Manifold Charting
- **ISOMAP**
- Kernel PCA

There are many more beyond this!

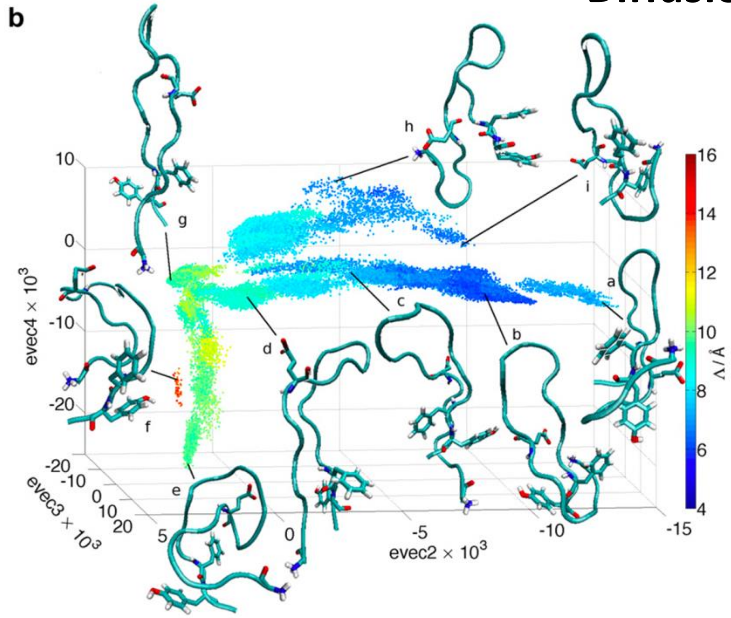
Other Greatest Hits of Dimensionality Reduction

Variational Autoencoders

Shmilovich, K.; Mansbach, R. A.; Sidky, H.; Dunne, O. E.; Panda, S. S.; Tovar, J. D. & Ferguson, A. L. Discovery of Self-Assembling pi-Conjugated Peptides by Active Learning-Directed Coarse-Grained Molecular Simulation. *The Journal of Physical Chemistry B* **2020**, *124*, 3873-3891

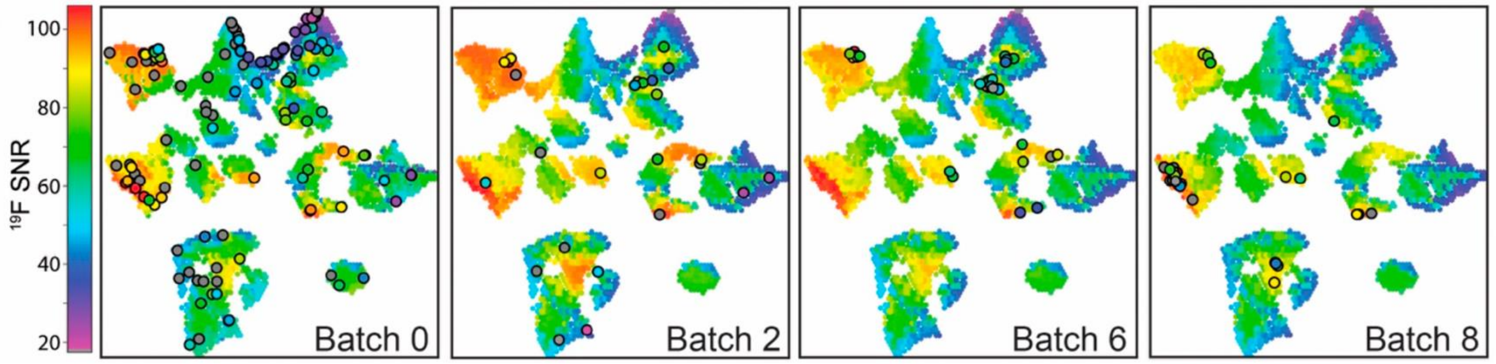


Diffusion Maps



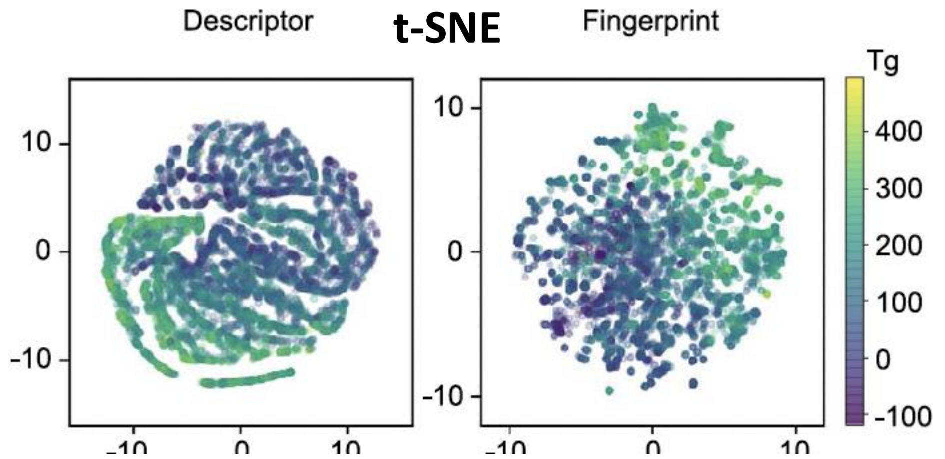
Ferguson, A. L.; Panagiotopoulos, A. Z.; Kevrekidis, I. G. & Debenedetti, P. G. Nonlinear dimensionality reduction in molecular simulation: The diffusion map approach *Chemical Physics Letters*, **2011**, *509*, 1-11

UMAP



Reis, M.; Gusev, F.; Taylor, N. G.; Chung, S. H.; Verber, M. D.; Lee, Y. Z.; Isayev, O. & Leibfarth, F. A. Machine-Learning-Guided Discovery of 19F MRI Agents Enabled by Automated Copolymer Synthesis *Journal of the American Chemical Society*, **2021**, *143*, 17677-17689

t-SNE

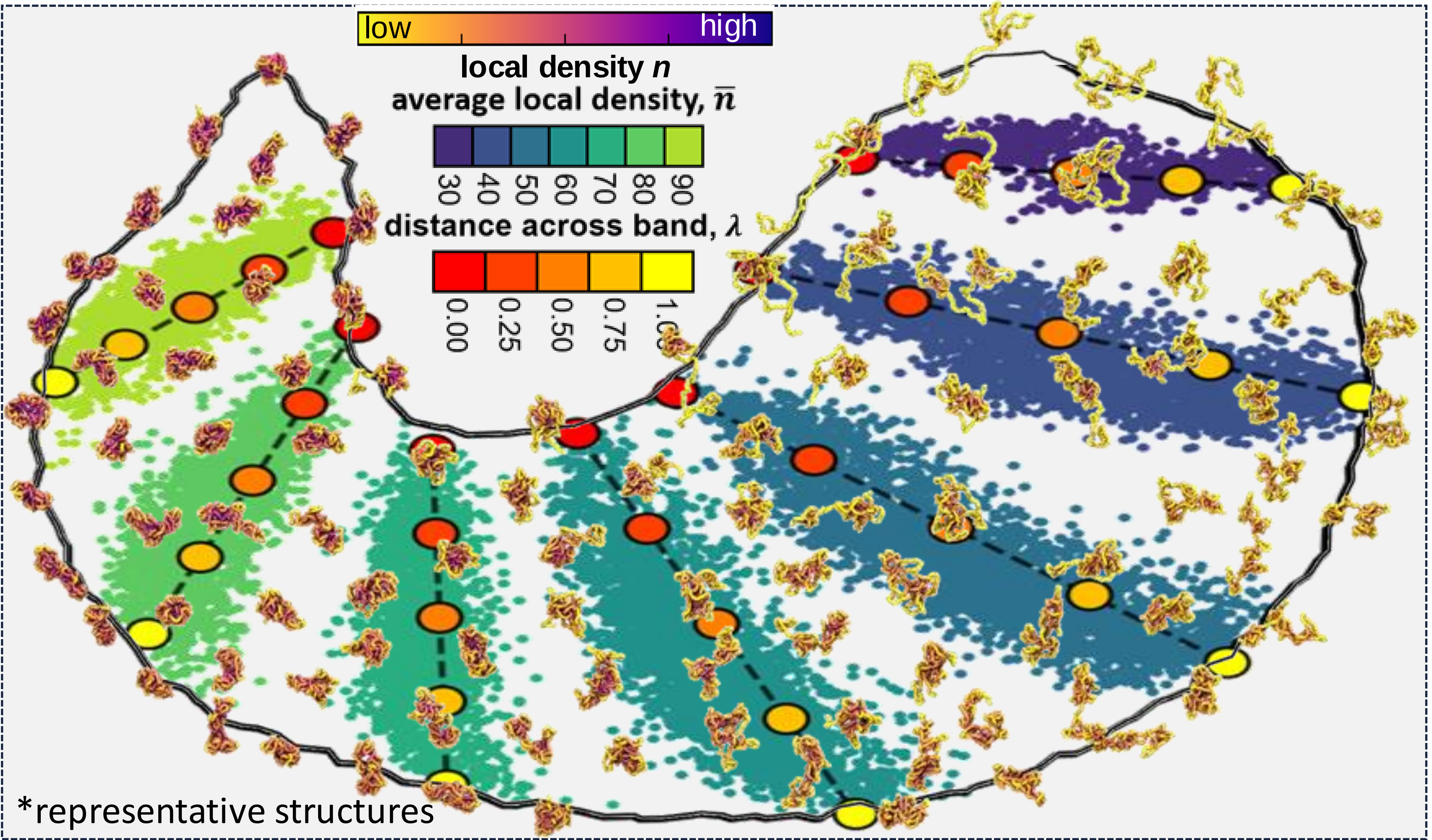


Tao, L.; Chen, G. & Li, Y. Machine learning discovery of high-temperature polymers *Patterns*, **2021**, *2*, 100225

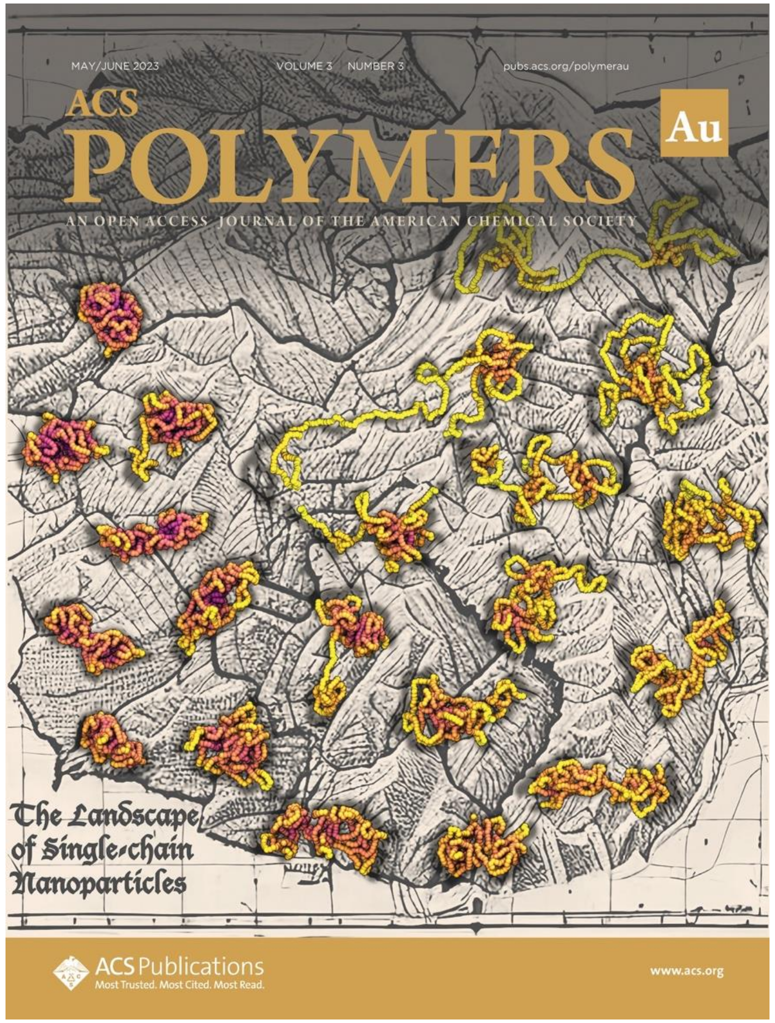
Other Greatest Hits of Dimensionality Reduction

1) Structures are **primarily** differentiated by the **average** local bead density

2) Structures are **secondarily** differentiated by the **variance** in local bead density



*representative structures



MAY/JUNE 2023
VOLUME 3 NUMBER 3
pubs.acs.org/polymerau
ACS POLYMERS Au
AN OPEN ACCESS JOURNAL OF THE AMERICAN CHEMICAL SOCIETY
The Landscape of Single-chain Nanoparticles
ACS Publications
Most Trusted. Most Cited. Most Read.
www.acs.org

Patel, Colmenares, Webb. *ACS Polymers Au.* ***2023***

Standard DR Techniques in Scikit-learn

Sklearn.decomposition

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

- Kernel PCA
- Independent Components Analysis
- Linear Factor Analysis
- Non-negative Matrix Factorization
- Truncated SVD

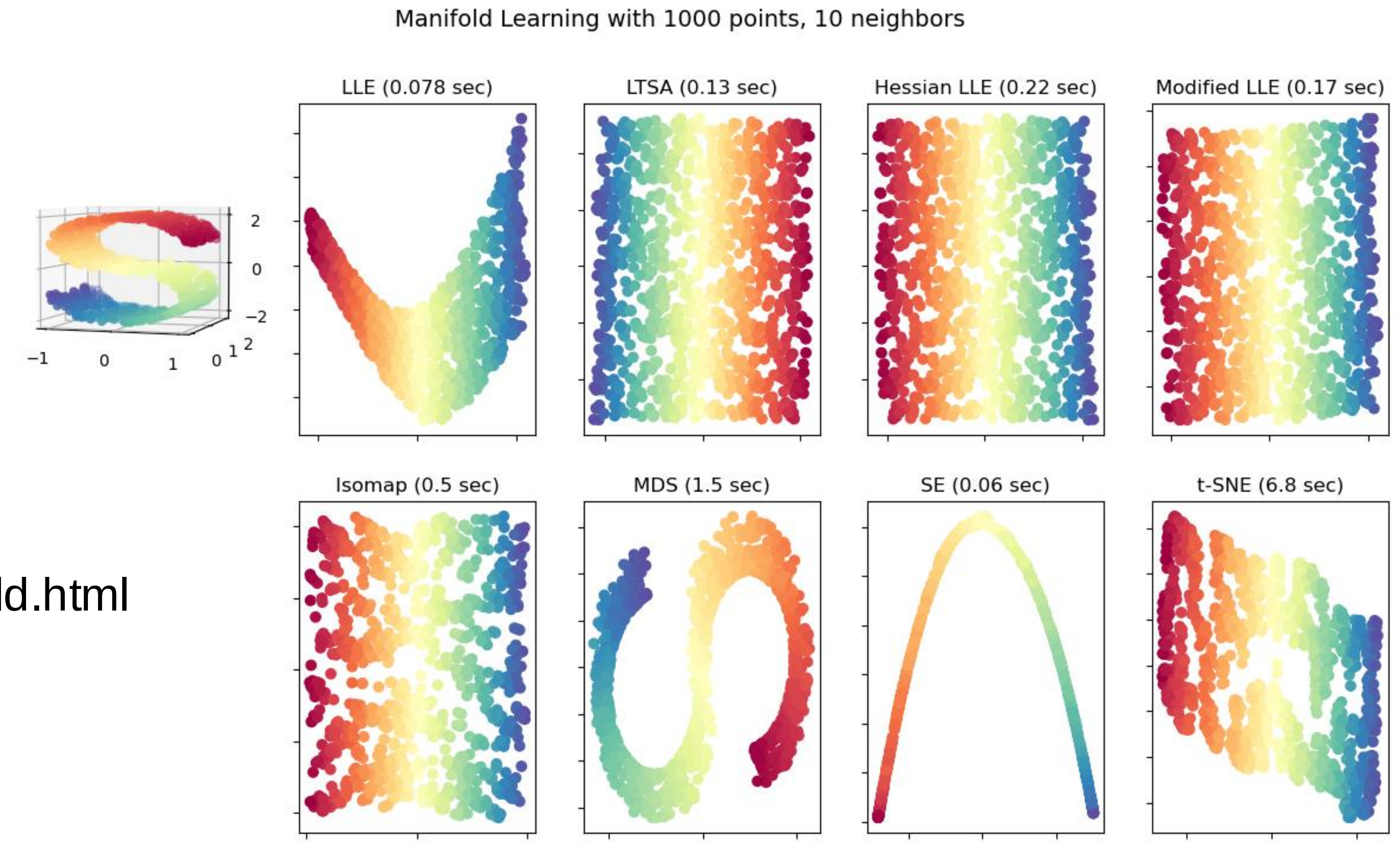
Sklearn.discriminant_analysis

- Linear Discriminant Analysis

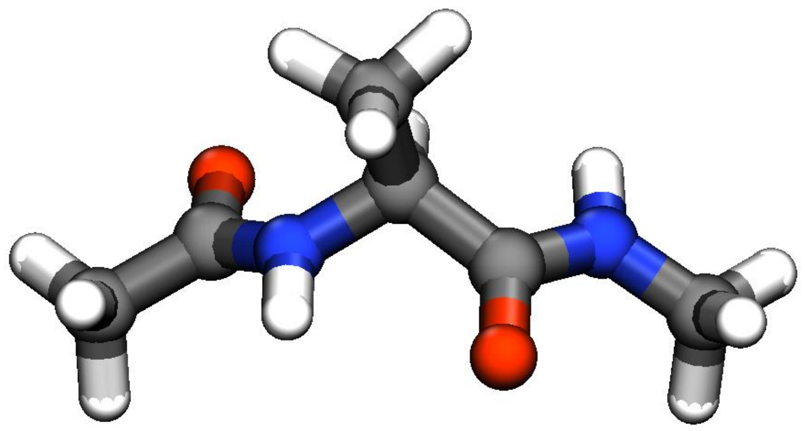
Sklearn.manifold

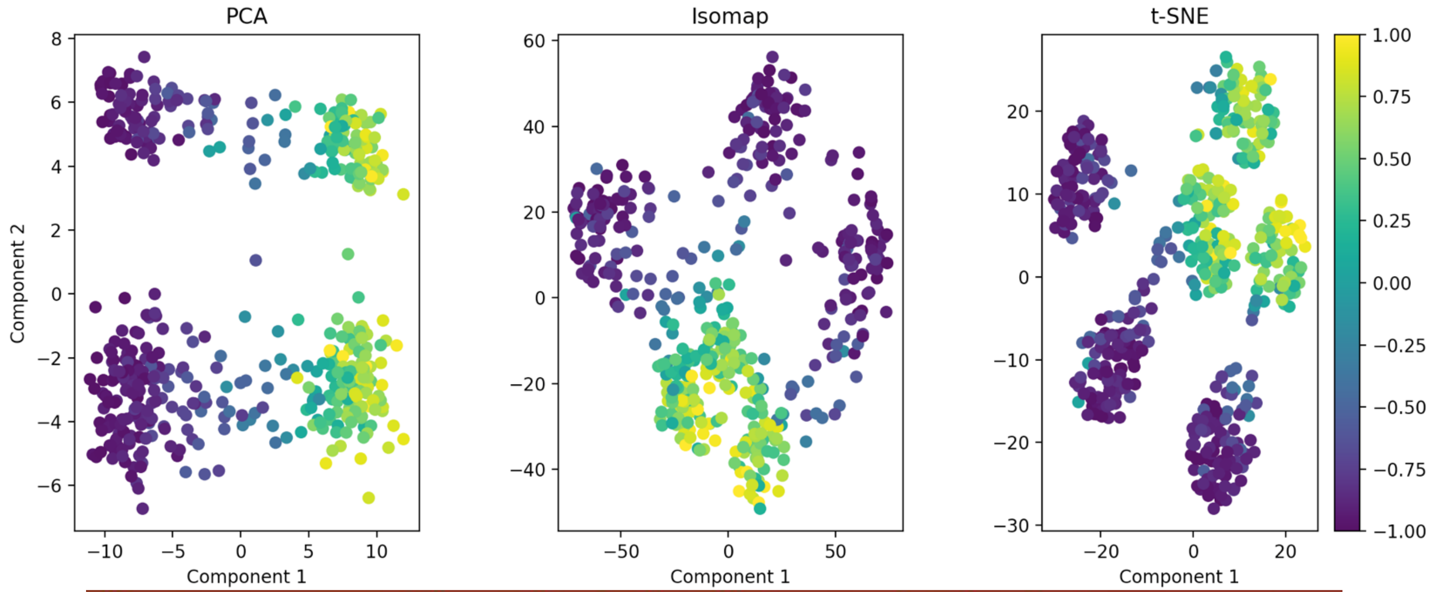
https://scikit-learn.org/stable/modules/manifold.html

- Isomap
- Locally Linear Embedding
- Spectral Embedding
- TSNE
- Multidimensional Scaling



Example on Alanine Dipeptide





1. Perform a molecular dynamics simulation (300 K)
2. Get coordinates of atoms as a function of time
3. From these coordinates compute the distance matrix amongst all atoms as a function of time
4. Use DR techniques to identify collective variables that provide a simplified description of the molecular motions

```
import os
import sys
import numpy as np
import math
from scipy.spatial.distance import cdist
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.manifold import Isomap,TSNE
```

```
#Now we can normalize all of the distances in the feature space by
subtracting the mean and dividing by std of each feature.
scaler = StandardScaler()
tdflat_norm = scaler.fit_transform(tdflat)

#Then we can do a PCA and keep the two largest components
pca = PCA(n_components=2)
pca.fit(tdflat_norm)

#Apply the PCA transformation matrix to the data
tdflat_trans = pca.transform(tdflat_norm)
```

```
#Do the same thing using Isomap
Isomap_embedding = Isomap(n_components=2)
tdflat_trans_isomap = Isomap_embedding.fit_transform(tdflat_norm)
```

```
#Do the same thing using t-SNE
TSNE_embedding = TSNE(n_components=2)
tdflat_trans_tsne = TSNE_embedding.fit_transform(tdflat_norm)
```

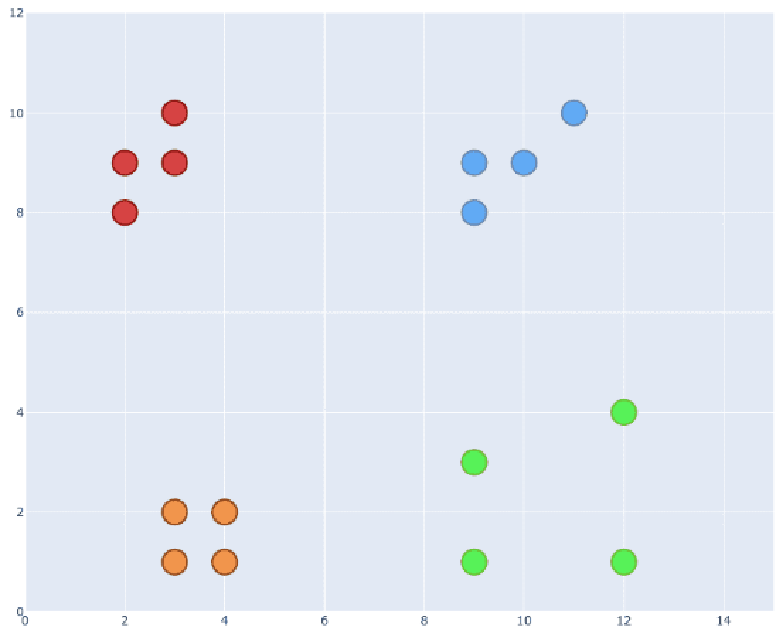
t-Distributed Stochastic Neighborhood Embedding (t-SNE)

Given a set of high dimensional data points $x_1,\ldots,x_n$ t-SNE computes the conditional probabilities, $p_{j|I}$, that are proportional to the similarity of data points x_i and x_j

For $i \neq j$, define

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

and set $p_{i|i} = 0$. Note that $\sum_j p_{j|i} = 1$ for all i.



Why can we call the similarity a conditional probability?

As explained in the original article:

“The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|I}$, that x_i would pick x_j as its neighbor if neighbors were picking in proportion to their probability density under a Gaussian centered at x_i.”

van der Maaten, L.J.P.; Hinton, G.E. (Nov 2008). "Visualizing Data Using t-SNE" (PDF). *Journal of Machine Learning Research*. **9**: 2579–2605.

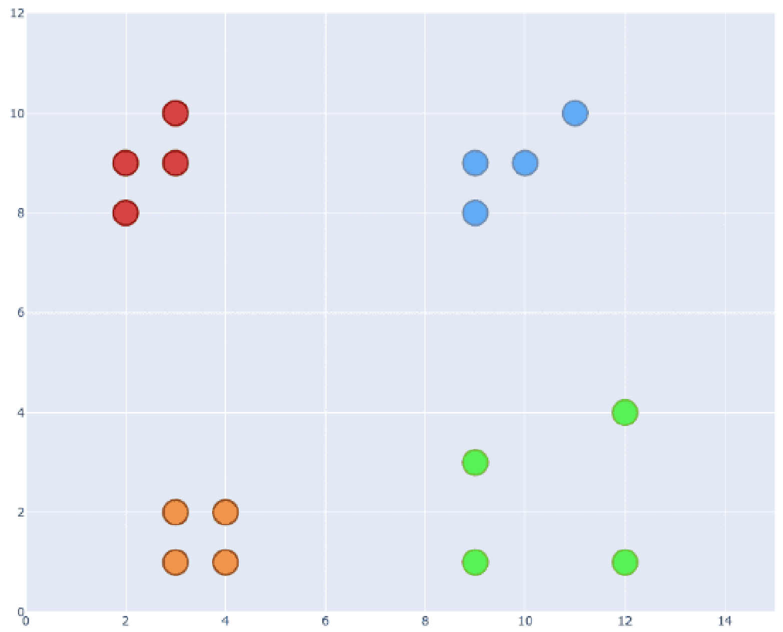
t-Distributed Stochastic Neighborhood Embedding (t-SNE)

Given a set of high dimensional data points $x_1,\ldots,x_n$ t-SNE computes the conditional probabilities, $p_{j|I}$, that are proportional to the similarity of data points x_i and x_j

For $i \neq j$, define

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

and set $p_{i|i} = 0$. Note that $\sum_j p_{j|i} = 1$ for all i.



So how the heck do we pick σ? σ effectively determines the number of nearest neighbors that any given point "feels".

In practice, this value of sigma is determined in a fairly complicated and mathematically involved fashion…

One defines the target perplexity "**k**" which is computed using:

$$H = -\sum_j p_{j|i} \log_2 p_{j|i} = \log_2 k,$$

One then finds the optimal σ by finding the value that satisfies this prespecified equation. **High k -> high σ.**

van der Maaten, L.J.P.; Hinton, G.E. (Nov 2008). "Visualizing Data Using t-SNE" (PDF). *Journal of Machine Learning Research*. **9**: 2579–2605.

t-Distributed Stochastic Neighborhood Embedding (t-SNE)

Given a set of high dimensional data points *$x_1, \ldots, x_n$* t-SNE computes the conditional probabilities, *$p_{j|I}$*, that are proportional to the similarity of data points *x_i* and *x_j*

Now – we find a lower dimensional space to which we can map the higher dimensional space that preserves these conditional probabilities as best as possible. We refer to the conditional probabilities in the low dimensional space as $q_{j|i}$.

$$q_{j|i} = \frac{e^{-\|y_i - y_j\|^2}}{\sum_{k \neq i} e^{-\|y_i - y_k\|^2}}.$$

How do we actually do this?
We can measure the mismatch between the high dimensional probabilities and the low dimensional probabilities using the Kullback-Leibler divergence as a loss function for each data point.

$$C = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}.$$

The Essence of the Algorithm:

- *Randomly place the low dimensional points and then minimize the cost function using gradient descent to find improved points. This is messy to do in practice but simple conceptually.*

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

t-Distributed Stochastic Neighborhood Embedding (t-SNE)

This isn't Exactly t-SNE. To Get to Real t-SNE You Modify a Couple of Things:

1. Symmetrize the conditional probabilities (Makes KL-divergence calculation faster)

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

Total # of high-dimensional data points.

and note that $p_{ij} = p_{ji}$, $p_{ii} = 0$, and $\sum_{i,j} p_{ij} = 1$.

2. Use a Student's t-distribution instead of a Gaussian for the low-dimensional probability distribution (Helps prevent crowding in the projection).

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

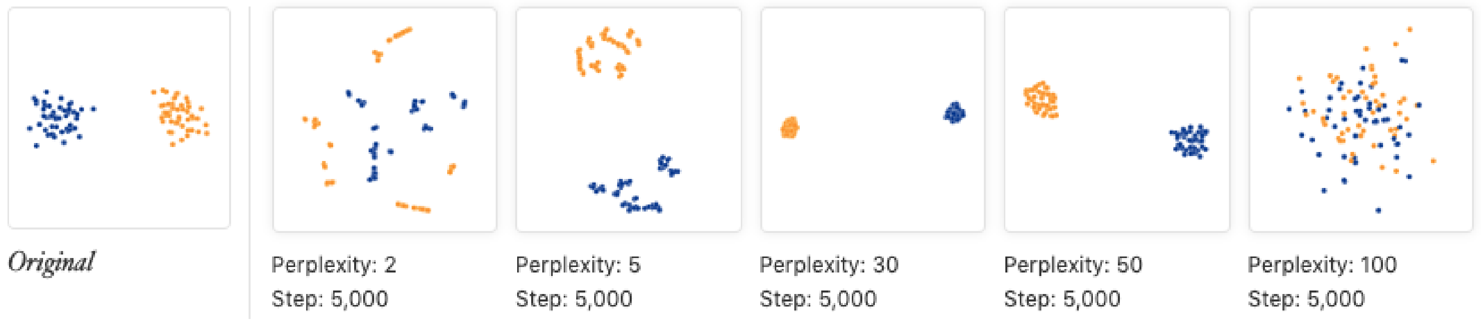
And then you do the same gradient of the KL divergence, etc.

t-Distributed Stochastic Neighborhood Embedding (t-SNE)

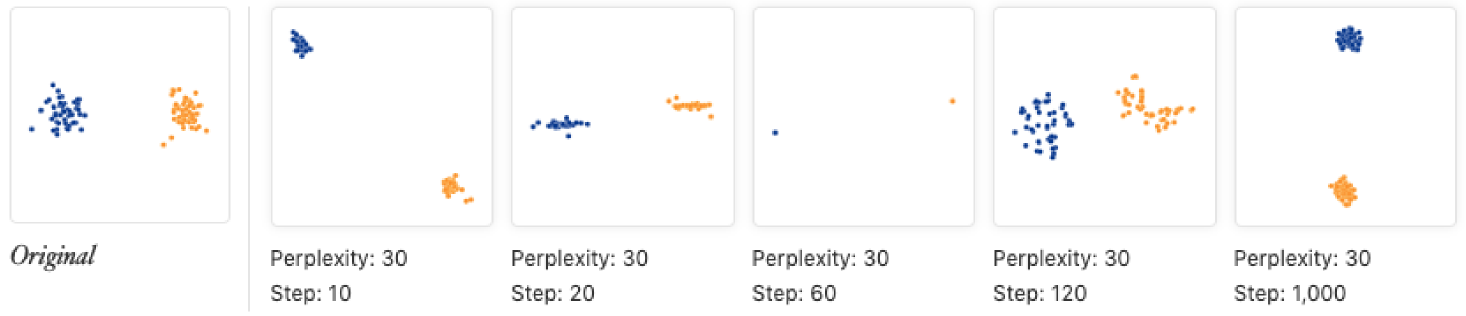
A few considerations/guidelines

Check out https://distill.pub/2016/misread-tsne/.

perplexity should be less than the number of points



convergence is something to monitor



don't over-interpret distances/shapes

Another Useful t-SNE Tip:

- Try initializing your low dimensional representation with a PCA-derived projection instead of random points. Usually this helps, and scikit-learn has a flag that will do this for you.

Uniform Manifold Approximation and Projection (UMAP)

The original paper is a pain to understand and read through, but it suffices to say that people like it because it has a more rigorous mathematical justification than t-SNE. In terms of function, it has many similarities to t-SNE.

Gaussian similarities between high dimensional data points

$$v_{j|i} = e^{-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}}$$

Symmetrization of similarities

$$v_{ij} = \left(v_{j|i} + v_{i|j}\right) - v_{j|i}v_{i|j}$$

T-distribution similarities between low dimensional data points

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

A more nuanced definition of "perplexity"

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

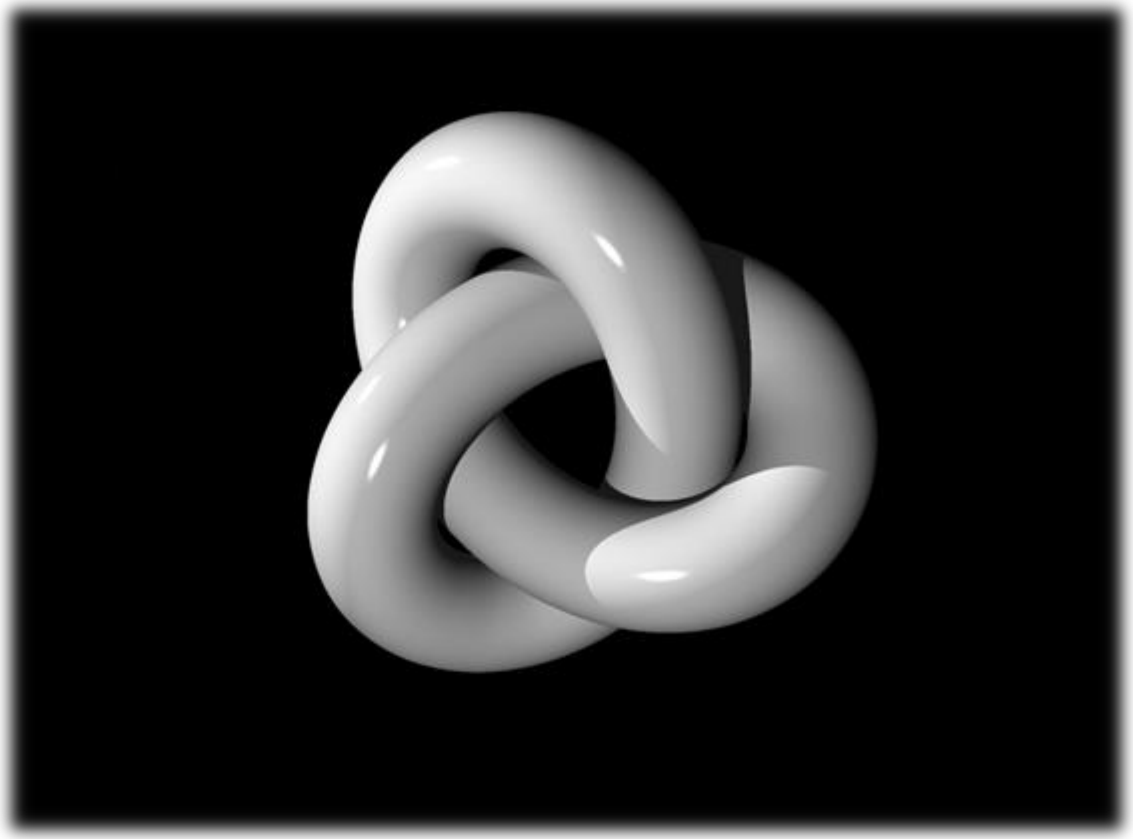
Other Points of Interest About UMAP

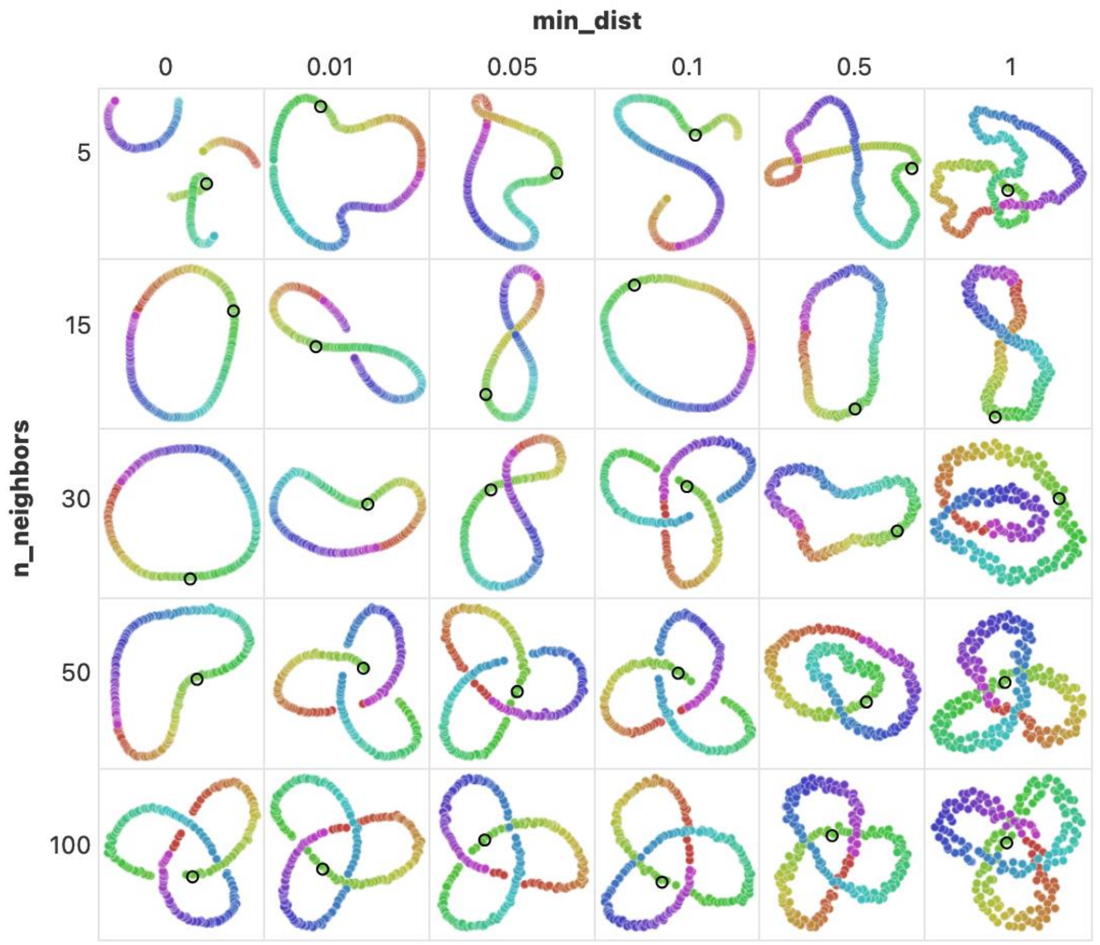
- Uses Cross-Entropy instead of KL Divergence.
- Uses Stochastic Gradient Descent instead of plain gradient descent
- Initializes the low dimensional space with a spectral method instead of random or PCA.

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

Uniform Manifold Approximation and Projection (UMAP)

Hyperparameters are again important to consider in how they dictate the resulting manifold; in UMAP there are effectively two to control





UMAP vs. t-SNE

If you would like to better understand UMAP I would recommend:

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

The biggest differences between UMAP and t-SNE are:

- UMAP has a firmer mathematical foundation than t-SNE, and thus many folks like it simply for this reason.
- UMAP uses a few clever tricks to improve numerical stability. It is also faster!!!
- UMAP is often better at preserving global structure than t-SNE - the inter-cluster relationships are potentially more meaningful than t-SNE.
- But both are very widely used!