Introduction to dimensionality reduction

Dmitry Girdyuk

June 22, 2022

Huawei, STIL

Table of contents

Problem overview

Problem statement

The task of *dimensionality reduction* is to transform high-dimensional data into a meaningful representation of reduced dimensionality while preserving the distances between (individual) observations as much as possible.

- Having matrix X of dimension $m \times n$.
- We assume that the initial data dimension is d, and it represent a mapping into a space of higher dimension n, $d \ll n$.
- We want to get the original representation Y of dimension $m \times d$ (also called embedding).
- It is desirable to be able to get the original representation for new data without complete retraining.

For what purpose is it used

- Getting rid of noise and multicollinearity.
- Less time spent on data processing.
- Visualization.

Common problems

The same problems as with clustering

- Lots of approaches with specific objective functions.
- Domain knowledge required.
- Difficulties in tuning hyperparameters.
- Most methods are non-parametric.

General classification of algorithms

Expanding horizons [12, 13] i

- Principal Component Analysis (PCA, Probabilistic PCA, Kernel PCA)
- Classical multidimensional scaling (MDS)
- Sammon mapping
- Linear Discriminant Analysis (LDA, Generalized DA)
- Factor Analysis (FA, Coordinated FA)
- Isometric feature mapping a.k.a ISOMAP (Landmark Isomap)
- Local Linear Embedding (LLE, Hessian LLE, Conformal Eigenmaps, Maximum Variance Unfolding)
- Laplacian Eigenmaps
- Local Tangent Space Alignment (LTSA, Linear LTSA)
- Maximum Variance Unfolding (MVU, LandmarkMVU, FastMVU)

Expanding horizons [12, 13] ii

- Diffusion maps
- Neighborhood Preserving Embedding (NPE)
- Locality Preserving Projection (LPP)
- Stochastic Proximity Embedding (SPE)
- Local Linear Coordination (LLC)
- Manifold charting
- Gaussian Process Latent Variable Model (GPLVM)
- Stochastic Neighbor Embedding (SNE, Symmetric SNE, t-SNE)
- LargeVis
- Neighborhood Components Analysis (NCA)
- Maximally Collapsing Metric Learning (MCML)
- Large-Margin Nearest Neighbor (LMNN)
- Uniform Manifold Approximation and Projection (UMAP)
- Various Autoencoders and less-known DL approaches

Algorithm's taxonomy

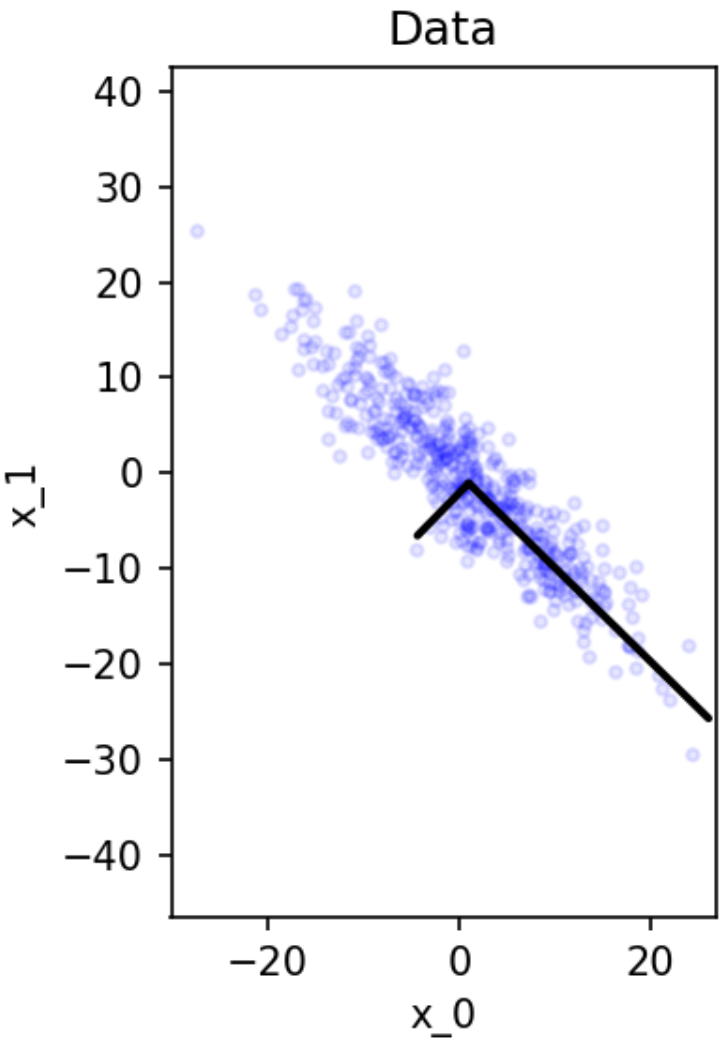
- Methods that optimize a convex objective function without local minima
  - PCA, Kernel PCA, Multidimensional scaling, ISOMAP, MVU, Diffusion maps, Laplacian eigenmaps, LLE, Hessian LLE, LTSA
- Methods that optimize a non-convex objective function with local minima:
  - Sammon mapping, LLC, Manifold charting, Deep autoencoders, t-SNE, LargeVis, UMAP

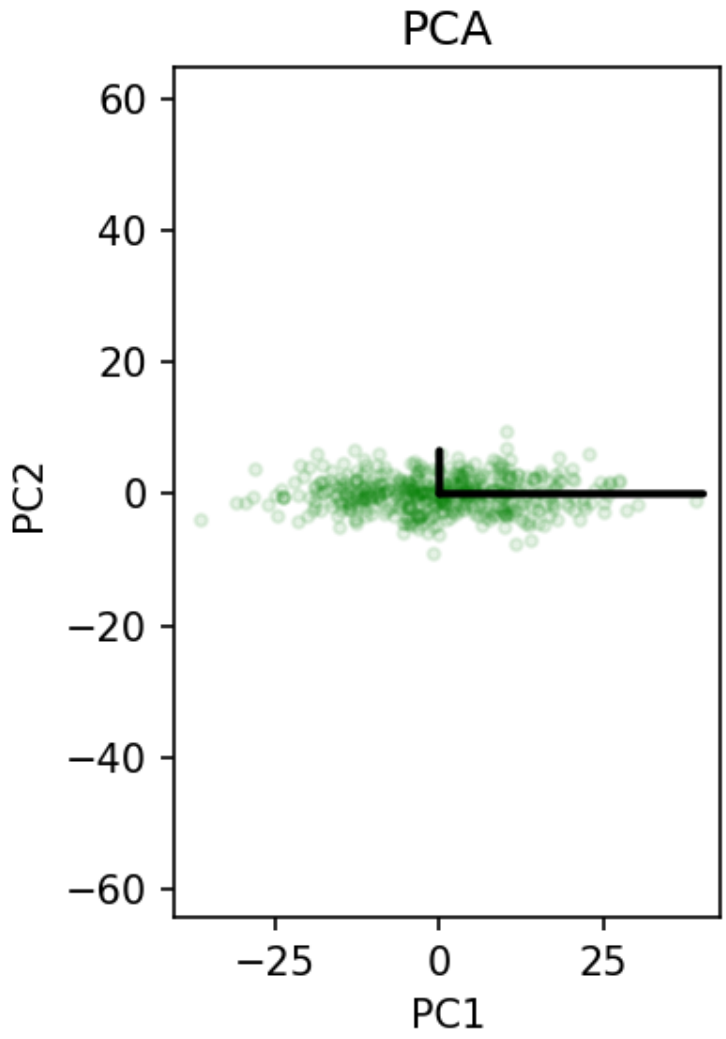
PCA

Principal Component Analysis (PCA)

- The principal components of some dataset $X_{[m \times n]}$ are a sequence of n vectors, each of which fits the data in the best way (in the sense of minimizing the mean squared distances between observations and the current vector), while each i-th vector is orthogonal to previous $i - 1$ vectors.
- New orthogonal basis? New orthogonal basis!
- Principal component analysis (PCA) is a dimensionality reduction method based on constructing a set of the first k of such orthogonal vectors.

PCA: example


Data
40
30
20
10
0
−10
−20
−30
−40
x_1
−20
0
20
x_0


PCA
60
40
20
0
−20
−40
−60
PC2
−25
0
25
PC1

PCA: theory 1

- We want to obtain such a normalized orthogonal set of vectors $v_j \in R^n, j = 1, 2, \ldots, d$, which can approximate the original vectors $x_i \in R^n, i = 1, 2, \ldots, m$

$$x_i \approx \sum_{j=1}^{d} c_{ij} v_j, \quad i = 1, 2, \ldots, m$$

- Normalize (optionally scale) the data!
- Objective function (minimizing the points projection variance on the principal component) in the case of $d = 1$

$$\frac{1}{m} \sum_{i=1}^{m} ||x_i \perp v||_2^2 \longrightarrow \min_{||v||_2 = 1}$$

PCA: theory 2

- The Pythagorean theorem allows you to transform the target function

$$||x_i \perp v||_2^2 + \langle x_i, v\rangle^2 = ||x_i||_2^2 \implies$$

$$\implies \frac{1}{m}\sum_{i=1}^{m}\langle x_i, v\rangle^2 = \frac{1}{m}(Xv)^T(Xv) = \frac{1}{m}v^T X^T X v =$$

$$= \frac{1}{m}v^T A v \longrightarrow \max_{||v||_2=1} \quad (1)$$

- The symmetric matrix $A = X^T X$ is a covariance (given scaling, also a correlation) matrix. An important property: the eigenvalues of such a matrix are non-negative.
- Objective function for the case $d > 1$, projection of x_i onto a vector subspace $V = \{v_1, \ldots, v_k\}$

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{d}\langle x_i, v_j\rangle^2 \longrightarrow \max_{V:||v_j||_2=1}$$

PCA: theory 3

- Now consider the decomposition of the matrix A

$$A = VDV^T,$$

where matrix V is an orthogonal matrix and D is a diagonal one.

- Note that if the matrix $A = \text{diag}(\lambda_1, \ldots, \lambda_n)$ (let $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_n \geqslant 0$)

$$v^T Av = \sum_{j=1}^{n} \lambda_j v_j^2$$

then the maximum for (1) is achieved for $v = e_1$

- For an arbitrary matrix A setting $v_1 = Ve_1$ we obtain

$$v_1^T Av_1 = v_1^T VDV^T v_1 = e_1^T V^T VDV^T Ve_1 = e_1^T De_1 = \lambda_1$$

PCA: theory 4

- Moreover, for any other vector $\hat{v}$, $\hat{v}^T A v \leqslant \lambda_1$.
- For the case $d > 1$ everything is deduced in a similar way. Optimal solution is first d columns of matrix V.
- Question: how to find matrix V?
- In fact, the matrix V in the decomposition $A = VDV^T$ is nothing else than a matrix whose columns are eigenvectors of matrix $A = X^T X$

$$Av_i = AVe_i = VDV^T Ve_i = VDe_i = \lambda_i Ve_i = \lambda_i v_i$$

- The last thing worth noting here is the uniqueness of the solution: if all eigenvalues are unique, then the decomposition is unique. Otherwise, there is a whole subspace of eigenvectors that solves the problem.

PCA: connection with the SVD

- Singular Value Decomposition

$$X = USV^T$$

  where U and V are orthogonal matrices of dimension $m \times m$ and $n \times n$ respectively, and matrix S is a diagonal matrix of dimension $m \times n$ with entries on the diagonal sorted in descending order.

- The columns of the matrix U are called the left singular vectors of the matrix X, the columns of the matrix V are called the right singular vectors of the matrix X.

PCA: connection with the SVD

- The left singular vectors of the matrix X are nothing but the eigenvectors of the matrix XX^T. Similarly, right singular vectors – eigenvectors of the matrix X^TX.
- Indeed

$$XX^T = USV^TVS^TU^T = USS^TU^T = UD_1U^T \implies$$
$$\implies XX^TU = UD_1$$

$$X^TX = VS^TU^TUSV^T = VS^TSV = VD_2V^T \implies$$
$$\implies X^TXV = VD_2$$

- This implies an algorithm for constructing the decomposition: use the linear algebra libraries to construct the eigendecompositions (more precisely, spectral decomposition) of matrices XX^T and X^TX.

PCA: Power Iteration 1

Power iteration is a simple algorithm to compute SVD and PCA.

Algorithm 1: Iterative principal component search algorithm

input : Matrix $A = X^T X$

Choose an arbitrary normalized vector u_0;

for $i = 1, 2, \ldots$ **do**

  $u_i = A^i u_0$;

  if $u_i / ||u_i||_2 \approx u_{i-1} / ||u_{i-1}||_2$ **then**

    Return u_i;

- It is important to note (proof by induction):
  $A^{i+1} = A^i A = V D^i V^T V D V^T = V D^{i+1} V^T$.
- As soon as the first component is found, we project the data orthogonally to the found principal component, i.e. set $x_i := x_i - \langle x_i, v_1 \rangle v_1$ and run the iterative algorithm again.

PCA: Power Iteration 2

Why does successive multiplication by vector u_0 of matrix A lead us to the principal component?

- $A = VDV^T$.
- $AV = VD$.
- Given that the columns of the matrix V have eigenvectors that form an orthogonal basis, then any vector in this basis can be written as $u_0 = c_1 v_1 + \ldots + c_n v_n$.
- We get

$$A^i u_0 = c_1 A^i v_1 + \ldots + c_n A^i v_n = c_1 \lambda_1^i v_1 + \ldots + c_n \lambda_n^i v_n = \\ = \lambda_1^i (c_1 v_1 + c_2 \frac{\lambda_2^i}{\lambda_1^i} v_2 + \ldots + c_n \frac{\lambda_n^i}{\lambda_1^i} v_n)$$

- Hence, for $i \to \infty$ with $\lambda_1 \geqslant \ldots \geqslant \lambda_n$, $A^i u_0 \to C(i) v_1$.

PCA: discussion i

- A simple interpretation of PCA: components are sequentially chosen so that the data projection variance (on components) is maximal. It follows from the centering of the data and the form of the objective function (1).
- We select the number of principal components based on the explained variance equal to the cumulative sum of the normalized eigenvalues corresponding to the eigenvectors (principal components). But for this we need to know all the eigenvalues, hence get full decomposition.
- There is an alternative, named Kaiser's rule:

$$\lambda_i > \frac{1}{n}\text{tr A}$$

- Principal component interpretation is often difficult.

PCA: discussion ii

- Nonlinear data structure? Use different method. For example, Kernel PCA! The only difference is that we find the eigenvectors not of the covariance matrix $X^T X$, but of the kernel matrix $K : k_{ij} = \kappa(x_i, x_j)$
- Final note: $Y = XV$, but we can take only d principal components and get the approximation $Y_d = XV_d$.

Multidimensional Scaling

Multidimensional Scaling [1]

- Multidimensional scaling is a family of (generally non-linear) dimensionality reduction techniques whose goal is to preserve distances between observations in a space of lower dimensionality.
- The algorithm doesn't work with points directly, but with a proximity (similarity/dissimilarity) matrix.
- Mostly used for visualization purposes.
- There are many variations on this general idea. Let's consider the classic algorithm and what is currently used in the corresponding packages.

Classical MDS: theory 1

- Having Euclidean distance matrix $D_{[m\times m]} = \{d_{ij}\}$, looking for such $y_j \in R^d, d < n, j = 1, 2, \ldots, m$, so

$$d_{ij} \approx ||y_i - y_j||_2$$

- Normalize the data before calculating the distances!
- It is clear that the solution is not unique: $Y + const$ will also satisfy the main requirement. Therefore, an additional restriction is introduced.

$$\sum_{i=1}^{m} y_{ip} = 0, \quad p = 1, \ldots, d \tag{2}$$

Classical MDS: theory 2

- Now consider the Gram matrix $B = YY^T$ (not Y^TY!)

$$||y_i - y_j||_2^2 = y_i y_i^T - 2y_i y_j^T + y_j y_j^T \implies d_{ij}^2 = b_{ii} - 2b_{ij} + b_{jj} \quad (3)$$

- In addition, from (2)

$$\sum_{i=1}^{m} b_{ij} = \sum_{i=1}^{m}\sum_{p=1}^{d} y_{ip}y_{jp} = \sum_{p=1}^{d} y_{jp} \sum_{i=1}^{m} y_{ip} = 0, \quad j = 1, \ldots, m$$

- From (3)

$$\sum_{i=1}^{m} d_{ij}^2 = \mathrm{tr}B + mb_{jj}, \quad \sum_{j=1}^{m} d_{ij}^2 = \mathrm{tr}B + mb_{ii}, \quad \sum_{j=1}^{m}\sum_{i=1}^{m} d_{ij}^2 = 2m \times \mathrm{tr}B \quad (4)$$

Classical MDS: theory 3

- Finally, from (3), (4) we have the following

$$b_{ij} = -\frac{1}{2}(d_{ij}^2 - \frac{1}{m}d_{.j}^2 - \frac{1}{m}d_{i.}^2 + \frac{1}{m^2}d_{..}^2)$$

- Then for the resulting matrix B, as in PCA, we find the decomposition $B = YY^T = U\Lambda U^T$.
- What gives us $Y = U\Lambda^{1/2}$, but as in PCA we can take only d eigenvectors that gives us $Y_d = U_d\Lambda_d^{\frac{1}{2}}$.

Metric MDS

- Metric MDS. The problem is reformulated as a minimization of the functional (called «Stress»)

$$\text{Stress}(Y) = \sigma(Y) = \sum_{i<j\leqslant m} w_{ij} \left(\delta_{ij} - \widehat{d}_{ij}(Y)\right)^2 \longrightarrow \min_Y \qquad (5)$$

  where δ_{ij} is the dissimilarities of observations i and j.
- To find a low-level representation using Metric MDS, the most commonly used iterative algorithm is Scaling by MAjorizing a COmplicated Function, SMACOF (implemented in sklearn).

Metric MDS: Iterative majorization

The central idea of majorization method is to replace iteratively the original complicated function $f(x)$ by an auxiliary function$g(x, z)$, where z in $g(x, z)$ is some fixed value. The function g has to meet the following requirements

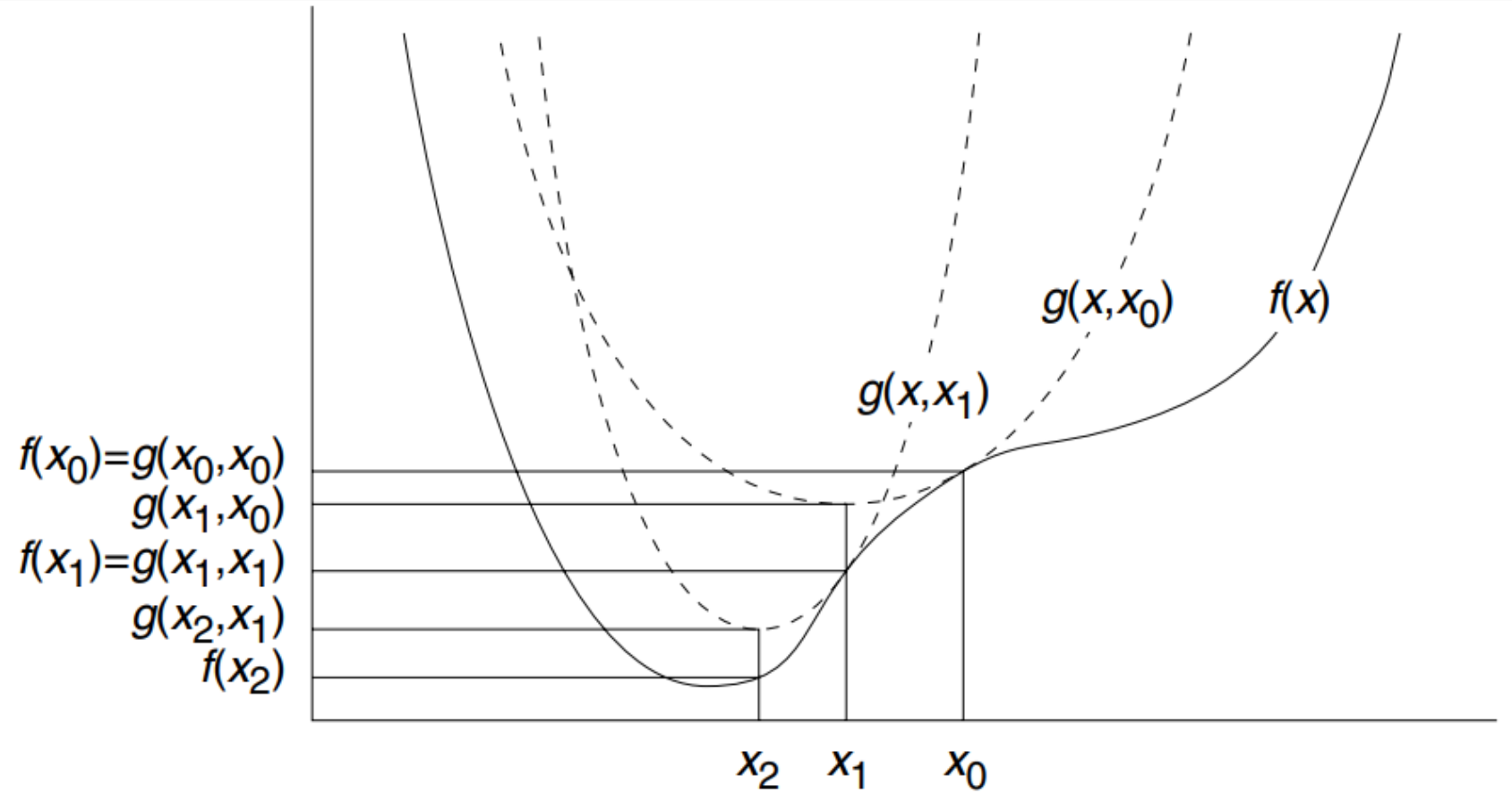
- Obviously, $g(x, z)$ should be simpler than $f(x)$. Quadratic or even linear.
- $f(x) \leq g(x, z)$.
- The auxiliary function should touch the surface at the so-called «supporting point»: $f(z) = g(z, z)$.

These rules implies

$$f(\widehat{x}) \leq g(\widehat{x}, z) \leq g(z, z) = f(z)$$

where $\widehat{x}$ is the minimum of $g(x, z)$.

Metric MDS: Iterative majorization visualization [1]



Metric MDS: SMACOF 1

- This function is upper bounded (using Cauchy–Bunyakovsky [1])

$$\sigma(Y) = \sum_{i<j} w_{ij}\delta_{ij}^2 + \sum_{i<j} w_{ij}\widehat{d}_{ij}^2(Y) - 2\sum_{i<j} w_{ij}\delta_{ij}^2\widehat{d}_{ij}^2(Y) =$$

$$= \text{Const} + \text{tr}\left[Y^T\widehat{W}Y\right] - 2\text{tr}\left[Y^T B(Y)Y\right] \leqslant$$

$$\leqslant \text{Const} + \text{tr}\left[Y^T\widehat{W}Y\right] - 2\text{tr}\left[Y^T B(Z)Z\right] = \tau(Y,Z) \quad (6)$$

where Z is the supporting point and $B(Z)$ defined as follows

$$b_{ij} = -\frac{w_{ij}\delta_{ij}}{\widehat{d}_{ij}(Z)} \text{ for } \widehat{d}_{ij}(Z) \neq 0, i \neq j$$

$$b_{ij} = \varepsilon \text{ for } \widehat{d}_{ij}(Z) = 0, i \neq j$$

$$b_{ij} = -\sum_{j=1|j\neq i}^{m} b_{ij}$$

Metric MDS: SMACOF 2

Since the majorization function $\tau(Y, Z)$ is quadratic, it could be minimized analytically.

Algorithm 2: SMACOF

Randomly initializing matrix Y_0 $(m \times d)$;
for $i = 1, 2, \ldots$ **do**
    $Z = Y_{i-1}$;
    $Y_k = \arg\min_Y \tau(Y, Z)$ (6);
    if $\sigma(Y_{i-1}) - \sigma(Y_i) < \varepsilon$ **then**
        Return Y_i;

Nonmetric (Ordinal) MDS

- There is also Nonmeric (Ordinal) MDS, where the distance metric $f(y_i, y_j)$ is non-Euclidean, only monotonic and preserves the order relation. Most often ranks (instead of observations itself) are used).
- And the problem of finding the embeddings extends with approximating $f(y_i, y_j)$. For those who is interested check this book [1].

MDS: discussion

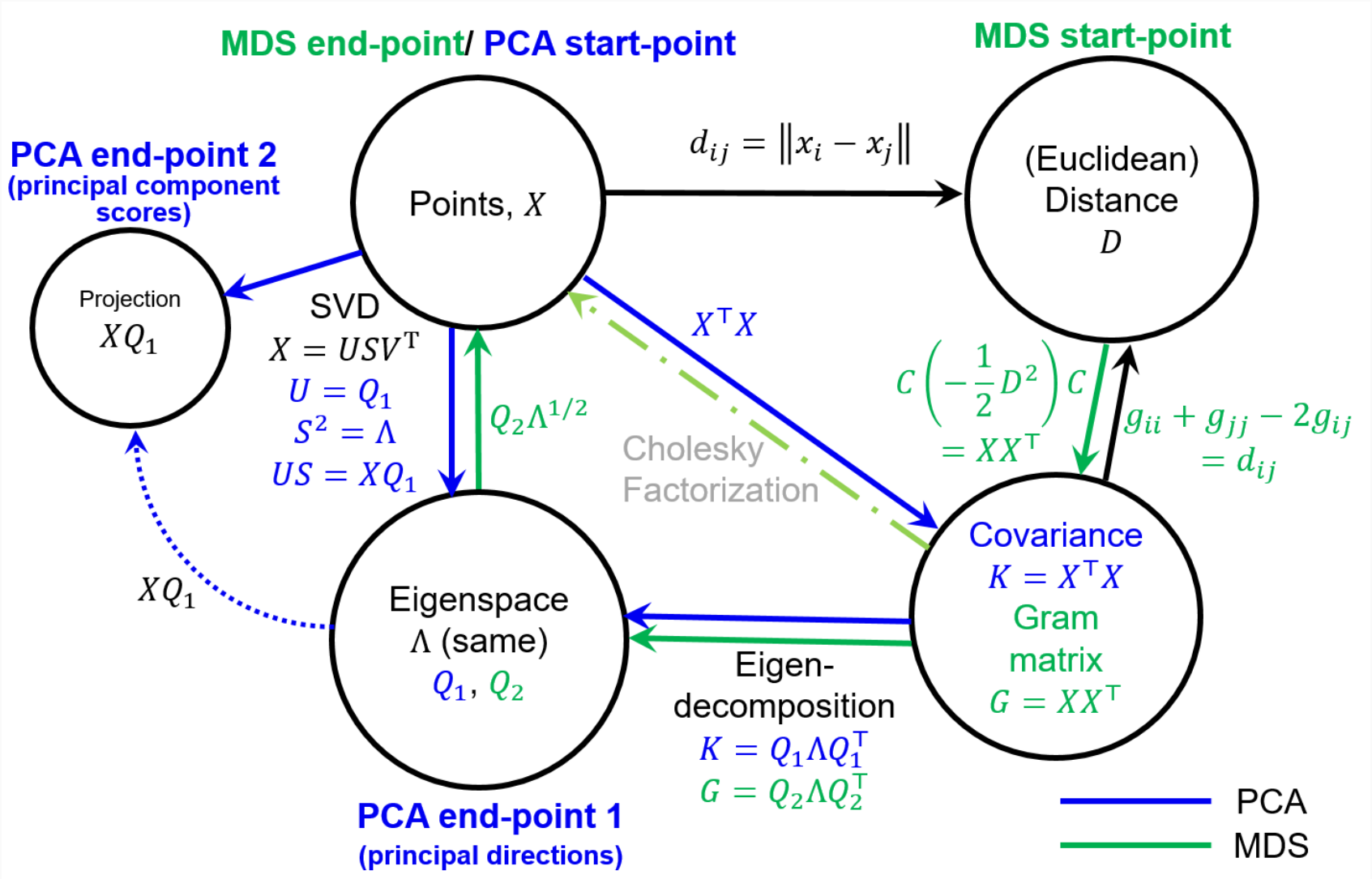
- Most often used for visualization.
- An established and still relevant set of algorithms with a long history.
- Algorithms have problems with the essential non-linearity of manifold.

Classical MDS equivalence to PCA

By the way, classical MDS is equivalent to PCA.

- Let $X^T X = Q_1 D_1 Q_1^T$ and $X X^T = Q_2 D_2 Q_2^T$.
- Let $X = USV^T$ and remembering that U and V are orthogonal, we get $X^T X = VS^T SV^T$. But $X X^T = USS^T U^T$. Therefore, $Q_1 = V$, $Q_2 = U$, and since S is diagonal $D_{1\,[n\times n]} \equiv D_{2\,[m\times m]}$.
- Finally, $X_{[m\times n]} V_{[n\times m]} = U_{[m\times m]} S_{[m\times n]}$ that proves the equivalence.

Classical MDS equivalence to PCA [8]

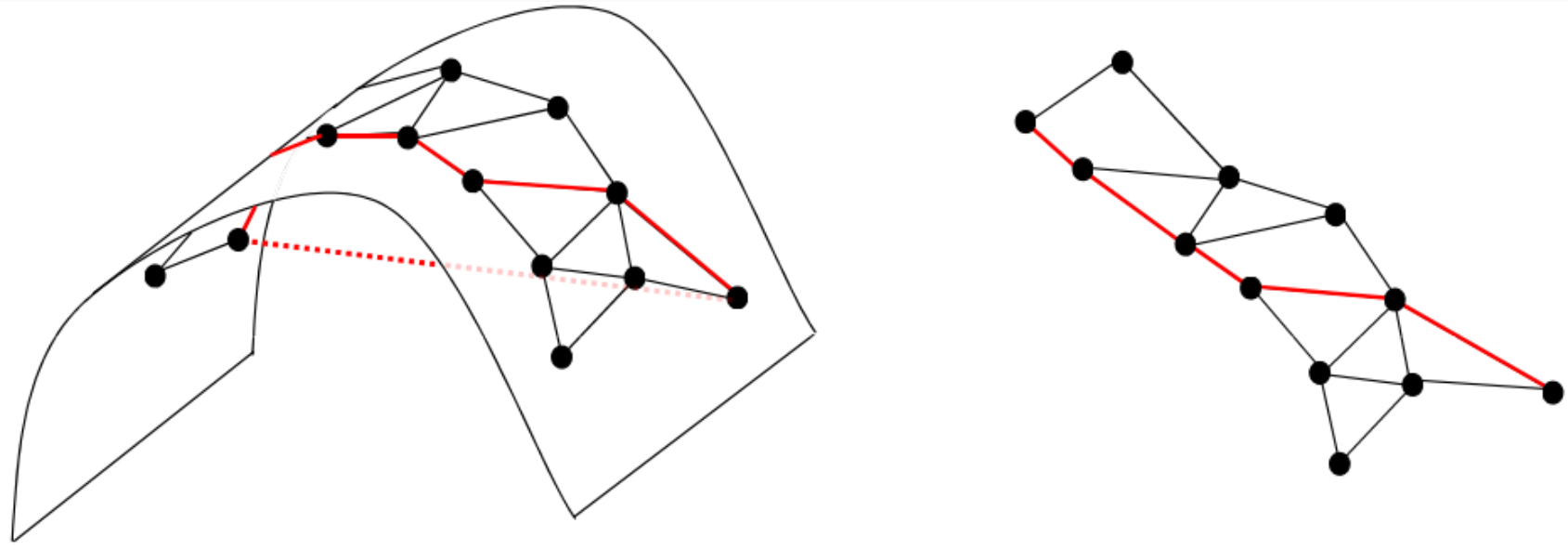


ISOMAP, LLE, and Laplasian Eigenmaps

ISOMAP [10]

- ISOMAP (Isometric feature mapping) is an example of a non-linear dimensionality reduction algorithm that actually generalizes MDS by working not with Euclidean distances between observations, but with geodesic distances based on observations k-nearest neighbors weighted graph.
- Formalizing,
  - Build a graph based on k-nearest neighbors for matrix X.
  - Construct a pairwise distance matrix for observations x_i, x_j by applying some algorithm for finding shortest paths on a graph (Dijkstra, Floyd–Warshall, etc).
  - Put the resulting matrix into a classical MDS.

ISOMAP: visualization [15]



ISOMAP: discussion

- In sklearn, instead of MDS, PCA is used, which, as already noted, is equivalent to classical MDS.
- A good example of algorithm generalization, nevertheless suffering from some shortcomings [13].
    1. Topological instability–the problem with closed loops–can significantly degrade the performance of the method.
    2. Problems with holes in manifold.
    3. Problems with non-convex manifolds.

LLE [9, 3]

- Local Linear Embedding (LLE) is a non-linear dimensionality reduction technique based on constructing a k-nearest neighbor graph and trying to find local weights to recover the original observations from their nearest neighbors.
- The main idea is that the weights that allow one to recover an observation from its neighbors can be used for the same purpose in a space of lower dimension (due to the assumption that the observation and its nearest neighbors lie on a linear manifold).

LLE: theory 1

- Having a graph of neighbors in hand, we construct a matrix W such that

$$\mathcal{E}(W) = \sum_{i=1}^{m} \left| x_i - \sum_{j: x_j \in Nb(x_i)} w_{ij} x_j \right|^2 \longrightarrow \min_W$$

  moreover, $w_{ij} = 0$ if x_i and x_j are not neighbors, and also $\sum_j w_{ij} = 1$.
- Constrained weights have an important property: when rotated, scaled or translated, they remain unchanged. Rotation and scaling is obvious, translation is due to sum-to-one contraint.

LLE: theory 2

- To find the weights, the following consideration is applied

$$\varepsilon_i = \left| x_i - \sum_j w_{ij}\eta_j \right|^2 = \left| \sum_j w_{ij}(x_i - \eta_j) \right|^2 = \\ = \sum_{j,p} w_{ij} w_{ip} \langle x_i - \eta_j, x_i - \eta_p \rangle \quad (7)$$

- And the conditional quadratic optimization problem should be solved.

LLE: theory 3

- Having found the weights, we consider a similar optimization problem

$$\Phi(Y) = \sum_{i=1}^{m} \left| y_i - \sum_j w_{ij} y_j \right|^2 \longrightarrow \min_Y$$

- As in the case of MDS, a constraint on Y is imposed

$$\sum_{i=1}^{m} y_{ip} = 0, \quad p = 1, \ldots, d$$

- And also constraine on unit covariance is added

$$\frac{1}{m} \sum_{i=1}^{m} y_i y_i^T = E$$

- The last constraint expresses an assumption that reconstruction errors for different coordinates in the embedding space should be measured on the same scale.

LLE: theory 4

- Note that the objective function is quadratic form ($\text{tr}(AA^T) = \text{tr}(A^T A)$)

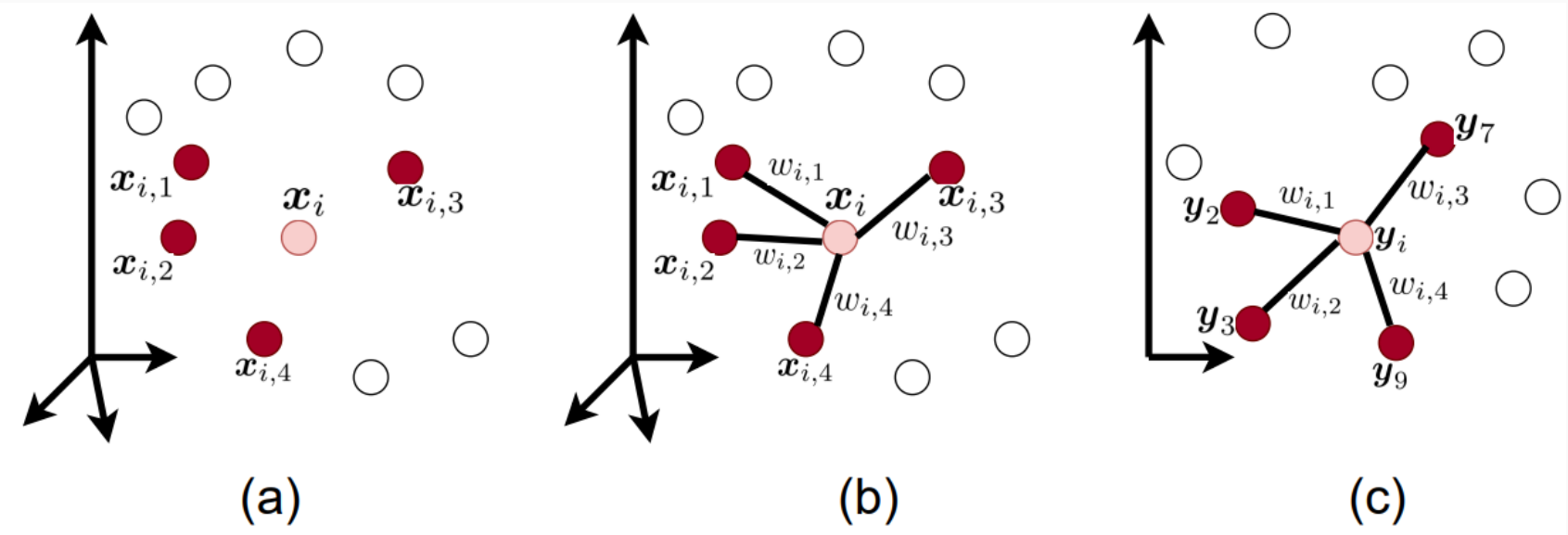
$$\Phi(Y) = \sum_{i=1}^{m} ||Y^T 1_i - Y^T w_i||_2^2 = ||Y^T (E - W)^T||_F^2 =$$
$$= \text{tr}\left(Y^T ((E - W) Y Y^T (E - W)^T\right) =$$
$$= \text{tr}\left(Y^T (E - W)^T (E - W) Y\right) = \text{tr}\left(Y^T M Y\right) \quad (8)$$

- Solving constrained optimization problem by Lagrange method we will come up with

$$MY = Y(\frac{1}{m}\Lambda)$$

  where $\Lambda = \text{diag}(\lambda_1, \ldots \lambda_d)$
- Hence, columns of Y are the eigenvectors of M with eigenvalues equal to Lagrangian multipliers divided by m which are the desired embeddings.

LLE: visualization [15]



LLE: discussion

- A simple and intuitive way to reduce dimensionality.
- The first eigenvector is a vector of all 1s (0-valued eigenvalue, Laplacian $E - W$) and should be excluded.
- Comparing to ISOMAP, LLE is scale-free, so the original distance scale is lost, but the local structure is preserved.
- There are a lot of extensions and generalizations, e.g. Hessian LLE, LTSA and others.

Laplacian eigenmaps

- Laplacian eigenmaps is similar to LLE in many ways: searching for a low-level representation of the data while trying to preserve the local properties of the desired manifold.
- In this case, local properties are the distances between an observation and its k-nearest neighbors.
- Studied during (spectral) clustering seminar: spectral embedding of Laplacian that is clustered in the end is Laplacian eigenmaps itself.

Spectral Clustering: RECAP

The Laplacian of a graph is a matrix $L = D - W$ with a set of interesting properties

- $z^\intercal Lz = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2$ foreach $z \in R^n$;
- L is symmetric and positive semidefinite;
- L n has no non-negative eigenvalues: $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$;
- The smallest eigenvalue λ_1 is always equal to 0. The corresponding eigenvector consists of 1 if the graph is connected. If the graph has p connected components, then the multiplicity of λ_1 is equal to p, and the eigenvectors are indicator vectors.

Spectral Clustering: RECAP

Algorithm 3: Spectral Clustering

input : Similarity matrix $S \in R^{n \times n}$ and the number of clusters k.

Build a similarity graph. Let W be its adjacency matrix;

Compute the Laplacian L;

Find the first d eigenvectors $u_1, \ldots, u_k$. Make a matrix $U \in R^{n \times d}$ out of them;

k-means(U, k);

t-SNE and UMAP

t-SNE

- t-distributed Stohastic Neighbor Embedding (t-SNE) [11] is a non-linear dimensionality reduction algorithm designed for visualization in (usually) 2D/3D embedding space.
- It is based on building a probability distribution of distances between pairs of observations in the original space and embeddings space, and calculating them [embeddings] by optimizing the Kullback–Leibler divergence between these two distributions.

t-SNE: theory 1

- Probability distribution for distances between pairs of observations i and j in the original space

$$p_{j|i} = \frac{\exp(-d(x_i, x_j)^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-d(x_i, x_k)^2/2\sigma_i^2)}, \quad p_{i|i} = 0 \tag{9}$$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2m} \tag{10}$$

- From (10)

$$\sum_{j=1}^{m} p_{ij} > \frac{1}{2m}$$

- Probability distribution (based on t-Student distribution with 1 degree of freedom) for distances in embedding space

$$q_{ij} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||y_k - y_l||^2\right)^{-1}}, \quad q_{i|i} = 0. \tag{11}$$

t-SNE: theory 2

- The standard deviation σ_i is fitted based on a hyperparameter called «perplexity»

$$\text{Perp}(P_i) = 2^{H(P_i)} = 2^{-\sum_j p_{j|i} \log p_{j|i}},$$

  where $H(P_i)$ is Shannon entropy of the random variable P_i (induced by σ_i over all other datapoints).
- Most often, a binary search for the value of σ_i is launched, due to the fact that a monotonic increase in the variance leads to a monotonic increase in perplexity.

t-SNE: theory 3

- The embeddings $y_i, i = 1, \ldots, m$ are calculated based on the optimization of the KL divergence between the distributions P and Q, which are joint distributions over all distances between pairs of observations

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

- The divergence gradient is

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_k - y_l||^2)^{-1} \qquad (12)$$

t-SNE: crowding problem [11]

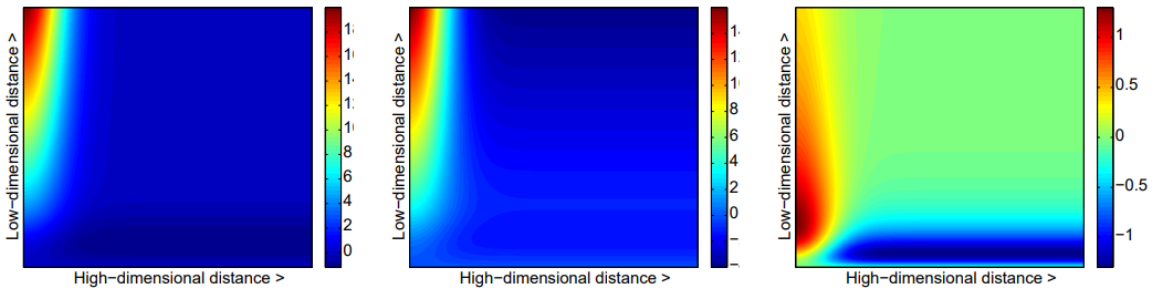
- The t-distribution improves the situation with the so-called «crowding problem». Observations that are at an average distance from the considered observation in the original space, in e.g. two-dimensional space will have to be located much further relative the observations that [in original space] placed closer to the considered observation.
- Therefore, if we model small distances accurately in the map, most of the points with average distance will be placed way too far. During the optimization, all such points will crush to the center of the low-dimensional space (the majority of p_{ij} becomes equal to a constant).

t-SNE: notes on original SNE

- Original SNE uses normal distribution with $\sigma_i = \frac{1}{\sqrt{2}}$ instead of a Student's t distribution for modeling $q_{j|i}$.
- SNE is minimizing the sum of the KL divergences between conditional probabilities $p_{j|i}$ and $q_{j|i}$, but these divergences are not symmetric. Hence, different types of errors in the pairwise distances in the low-dimensional spece are not weighted equally.
- E.g. larger cost for using [in low dimension] separated points to represent nearby datapoints [in original space] comparing with case where nearby points represent widely separated datapoints. Hence, only local structure is retained (as in LLE).
- Moreover, optimization process was highly unstable and required lots of tricks (adding Gaussian noise, changing this noise variance during the process, momentum term, etc).

t-SNE: notes on t-Student distribution

- Since t-SNE match probabilities, there is a natural way to alleviate the crowding problem by using normal distribution in the original and heavy tailed t-Student distribution in the embedding space.
- Why degree of freedom is 1? Simple formula to compute, and (almost) inverse square law for large pairwise distances.



(a) Gradient of SNE. (b) Gradient of UNI-SNE. (c) Gradient of t-SNE.

Figure 1: Gradients of three types of SNE as a function of the pairwise Euclidean distance between two points in the high-dimensional and the pairwise distance between the points in the low-dimensional data representation.

- t-SNE puts emphasis on (1) modeling dissimilar datapoints by means of large pairwise distances, and (2) modeling similar datapoints by means of small pairwise distances.

t-SNE: optimization

- Initialize y_i usually either with values from a normal distribution with zero mean and very small variance, or based on PCA.
- As for gradient descent, in addition to the term with the learning rate and anti-gradient, autors added momentum

$$Y^t = Y^{t-1} - \eta \frac{\partial C}{\partial y_i} + \alpha(t)(Y^{t-1} - Y^{t-2}) \tag{13}$$

t-SNE: algorithm

Algorithm 4: t-SNE

input : $u, \eta, \alpha(t)$

Calculate p_{ij} using (10);

Randomly initialize $m \times d$ matrix Y_0;

for $i = 1, 2, \ldots$ **do**

  Calculate q_{ij} using (11);

  Calculate gradient $\frac{\partial C}{\partial y_i}$ using (12);

  Update Y^i using (13) ;

t-SNE: algorithm's optimization tricks

- At the initial stage of optimization, p_{ij} is multiplied by a factor called «early exaggregation». This allows to work with more dense and well-separated clusters in a space of smaller dimensions relative to the clusters in the original data.
- Recommended values for perplexity range from 5 to 50. However, this value can be increased for large datasets. The perplexity itself can be considered as the effective number of neighbors of the observation under consideration.

How to Use t-SNE Effectively [14]

- Hyperparameters are of fundamental importance (special attention, of course, to perplexity).
- Cluster sizes in embedding space don't matter.
- Distance between clusters may not matter either.
- Random noise can sometimes form something «unusual».
- Sometimes interesting forms of clusters may appear in the embedding space, which are not characteristic of the original data.
- To make sure that there is a specific data topology, it is necessary to build plots with different values of perplexity.

t-SNE: algorithm's extensions

- The complexity of t-SNE is largely due to the quadratic complexity in calculating p_{ij} and the gradient (12).
- The first idea for improvement is to build $\lfloor 3u \rfloor$-nearest neighbors for each observation and compute p_{ij_k}. The rest of the values set to zero. It is proposed to search for neighbors by building a VP-tree (vantage point). Computational complexity is $O(un \log n)$.
- A modification of the algorithm based on the Barnes-Hut algorithm [4] makes it possible to approximate the values of gradients by constructing a tree of a special form for storing y_i, reducing the computational complexity of this step to $O(n \log n)$.

t-SNE: Barnes–Hutt algorithm 1

- Let's write the gradient (12) as follows

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

$$= 4\sum_j (p_{ij} - q_{ij})q_{ij}Z(y_i - y_j) =$$

$$= 4\left(\sum_j p_{ij}q_{ij}Z(y_i - y_j) - \sum_j q_{ij}^2 Z(y_i - y_j)\right) \quad (14)$$

  where $Z = \sum_{k \neq l} \left(1 + ||y_k - y_l||^2\right)^{-1}$.
- The first term is considered to be $O(un)$, taking into account the first idea on P approximation and since $q_{ij}Z = (1 + ||y_i - y_j||^2)^{-1}$.
- The second term is $O(n^2)$ and can be approximated in $O(n \log n)$ by building a quadtree for 2D embeddings. For the 3D case, you can use an octtree.

t-SNE: Barnes–Hutt algorithm 2

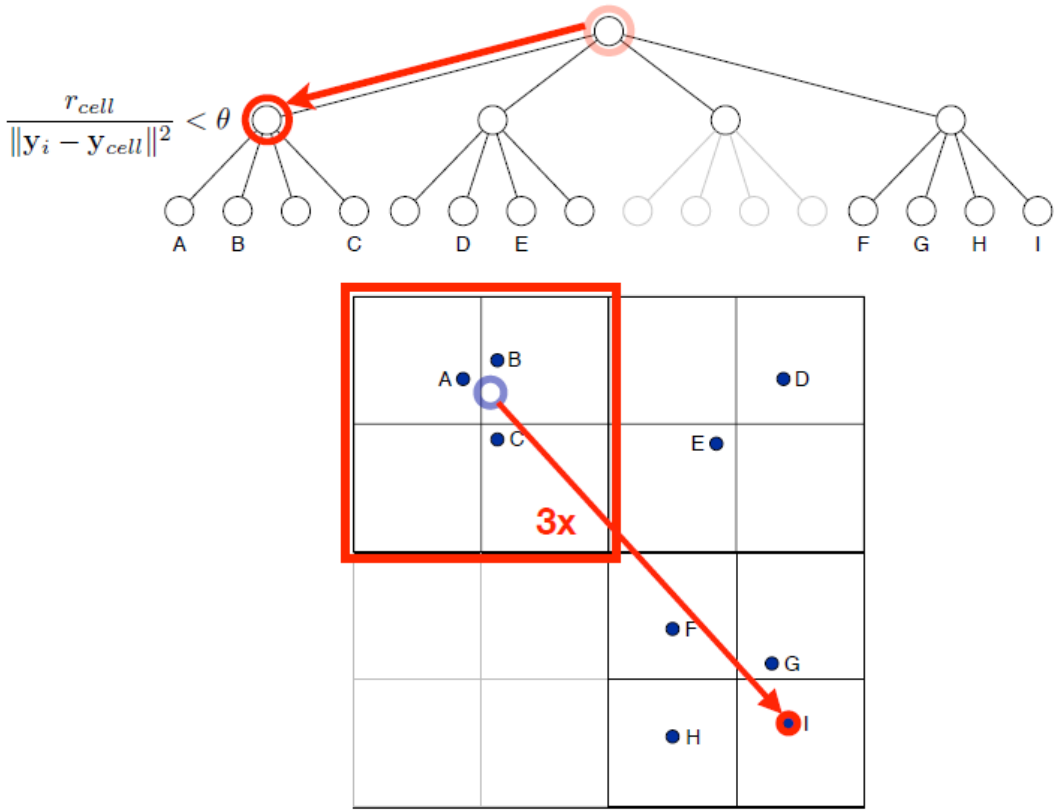


Figure 2: Illustration of the Barnes-Hut approximation. To evaluate the t-SNE gradient for point I, the Barnes-Hut algorithm performs a depth-first search on the embedding quadtree, checking at every node whether or not the node may be used as a "summary". In the illustration, the cell containing points A, B, and C satisfies the summary-condition: the force between the center-of-mass of the three points (which is stored in the quadtree node) and point I is computed, multiplied by the number of points in the cell (*i.e.*, by three), and added to the gradient for point I. All children of the summary node are pruned from the depth-first search.

t-SNE: Barnes–Hutt algorithm 3

- Using the constructed tree, we can approximate the second term as follows

$$-q_{ij}^2 Z(y_i - y_j) \approx -n_{cell} q_{i,cell}^2 Z(y_i - y_{cell}),$$

  where n_{cell} is the number of points in the quadrant, and y_{cell} is the points center of gravity in the quadrant.
- A small hack: during the depth-first search on the quadtree, we find an approximation for the value of $-q_{ij}^2 Z^2(y_i - y_j)$ and divide by Z (its approximate value we also calculate in the search process) due to the form of $q_{ij}Z$.
- The stopping criterion for the quadrant is as follows

$$\frac{r_{cell}}{||y_i - y_{cell}||^2} < \theta,$$

  where r_{cell} is the quadrant's diagonal.
- Note: θ is the same with «angle» parameter in sklearn.

t-SNE: discussion 1

- Perhaps the most popular non-linear method for visualizing multidimensional data in (usually) 2D/3D space.
- The method is flexible, the number of parameters is greater than that of the «average» non-linear dimensionality reduction algorithm.
- However, it's usually enough to look at different perplexity values (and try to vary the learning rate when optimizing KL divergence).

t-SNE: discussion 2

Disadvantages:

- Difficulties in interpretation [14].
- Time-consuming (in the classical implementation of the algorithm, the computational complexity is $O(n^2)$).
- Modification based on VP-tree and Barnes-Hut algorithm (default in sklearn) reduces this number to $O(un \log n)$, but limits its use for 2D/3D dimensional embeddings.

UMAP

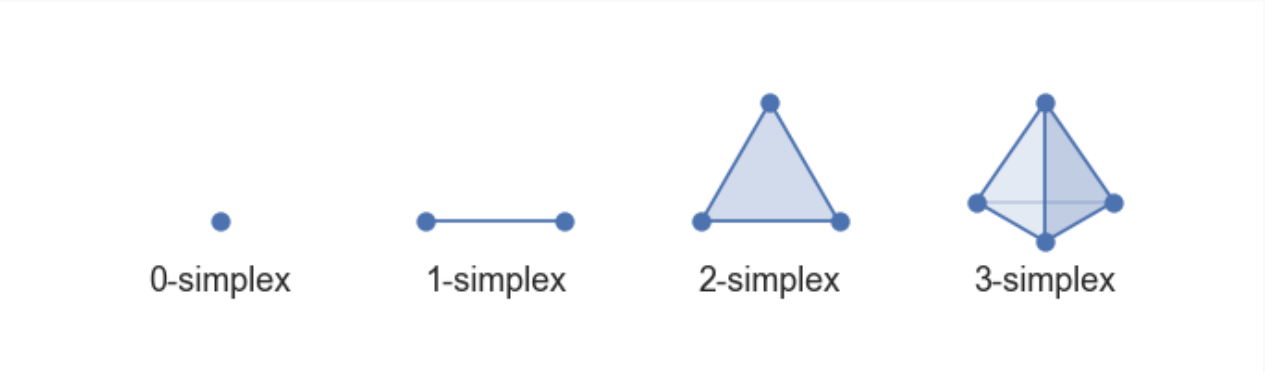
- Uniform Manifold Approximation and Projection, UMAP [5] is a quite recent (2018) non-linear dimensionality reduction algorithm.
- It is developed based on results from Riemannian geometry, algebraic topology, topological data analysis, and elements of fuzzy algebra.
- Nevertheless, deep understanding is not necessary for a general understanding of the algorithm. The method belongs to the class of algorithms that build a graph of k-nearest neighbors, in the manner of Laplacian eigenmaps, ISOMAP or t-SNE.

UMAP: one slide description [5]

- At a high level, UMAP uses local manifold approximations and patches together their local fuzzy simplicial set representations to construct a topological representation of the high dimensional data.
- Given some low dimensional representation of the data, a similar process can be used to construct an equivalent topological representation.
- UMAP then optimizes the layout of the data representation in the low dimensional space, to minimize the cross-entropy between the two topological representations.
- Now let's try to understand what does all this mean without deep dive into math behind it (with friendly explanation from the author of the algorithm [6]).

UMAP: simplicial complexes and sets

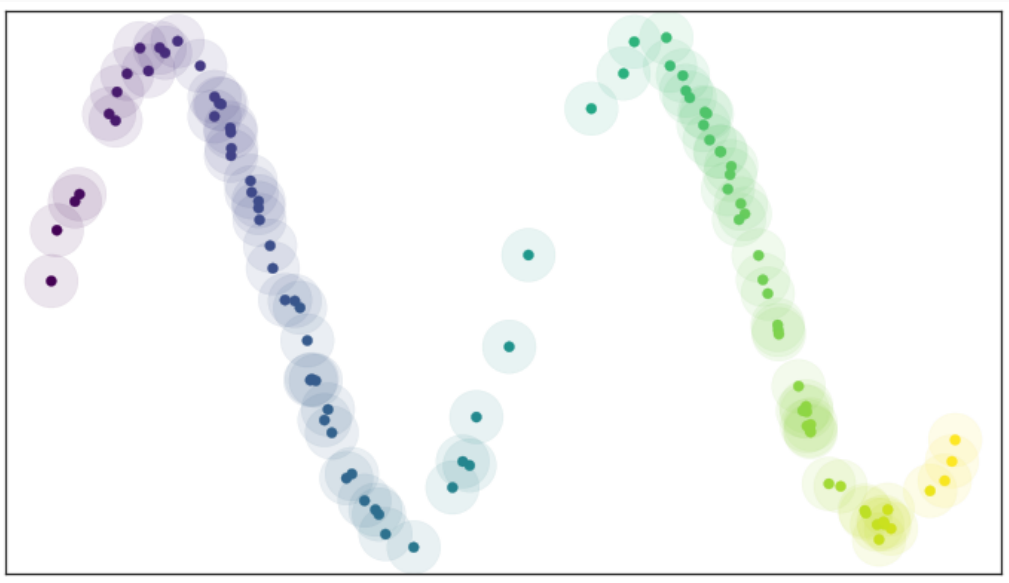
- Simplicial complexes are a means to construct topological spaces out of simple combinatorial components.
- This allows to reduce the complexities of dealing with the continuous geometry of topological spaces to the task of relatively simple combinatorics.
- Simplicial complex is a set of "simplices" (simple combinatorial blocks) joined together along faces.
- Simplicial set is just a higher abstraction of simplicial complex in modern homotophy theory (Note: this starts the lack of explanation, but it was planned to skip category theory things anyway).

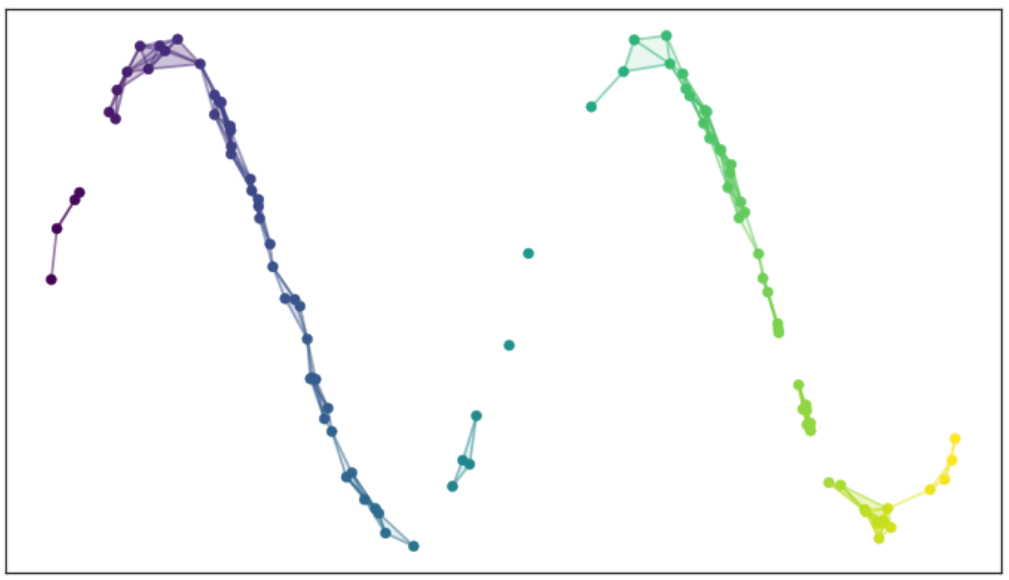


UMAP: from topological space to simplicial complex 1

- There are different ways to construct simplicial complex given the topological space. For those who want to read about it, look on Čech complex, Vietoris-Rips complex, and Nerve theorem.
- All we need to know is that the background topological theory actually provides guarantees about how well this process can produce something that represents the topological space itself in a meaningful way.
- Specifically, the Nerve theorem says that the simplicial complex will be (homotopically) equivalent to the union of the open cover.
- Let's consider the visualization of the process on the finite set of data.

UMAP: from topological space to simplicial complex 2



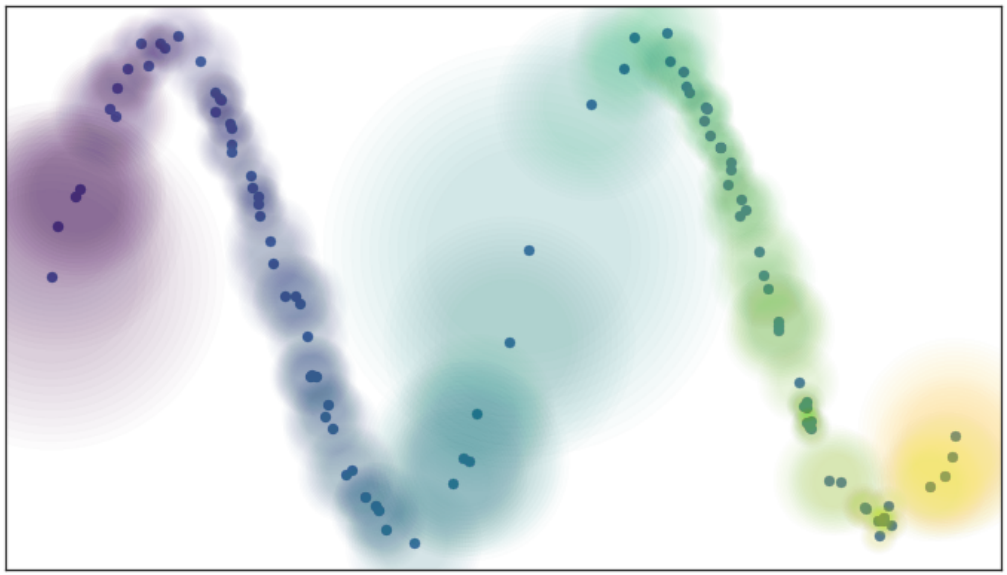


UMAP: from topological space to simplicial complex 3

- By the way, if we only care about the 0- and 1-simplices, then simplicial complex is just a graph, and finding a low dimensional representation can be described as a graph layout problem. The funny thing is that UMAP–despite all the theory–focuses on this particular case.
- Anyway, how to deal with the real data? We study topological spaces using open covers, balls with predefined radius. On practice it is hard to chose propper radius since the data is not uniformly sampled.
- Choosing small radius will result in a set of disconnected components. Choosing large radius will result in covering "too much".

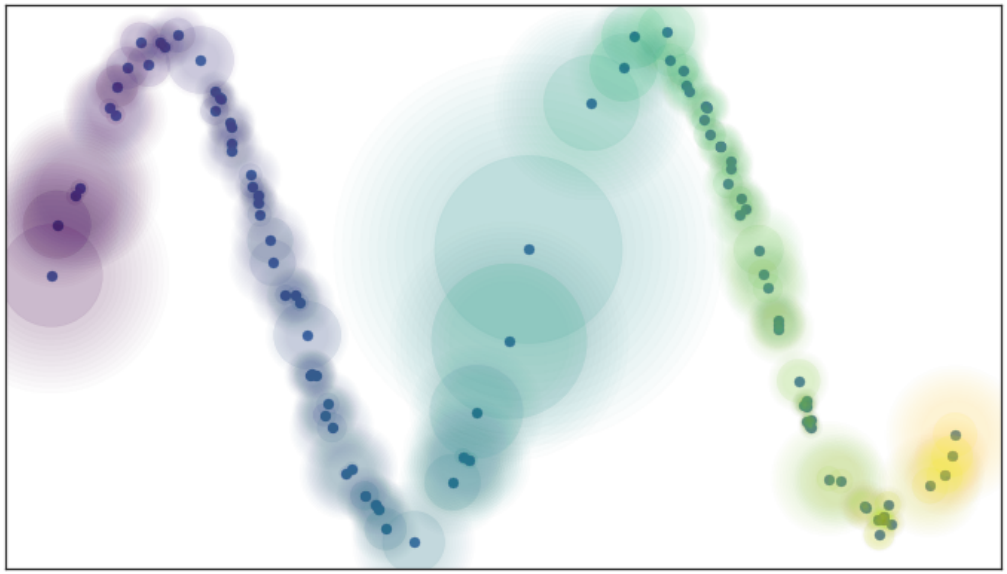
UMAP: make it great for real data 1

- How to resolve the problem of not having uniformly distributed data? Let's consider local distance for every point by making use of a little standard Riemannian geometry. On practice, consider a unit ball to the k-th neighboor of the given point with it's own metric.
- Moreover, having local Riemannian metric we can consider a fuzzy cover by weightening the distance (details will be provided later) in the local space



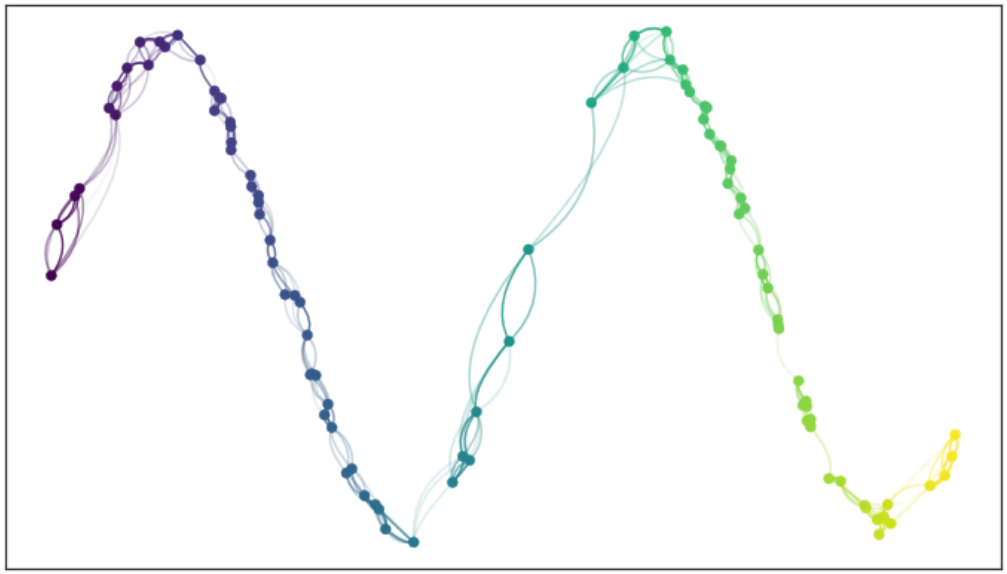
UMAP: make it great for real data 2

- Still need some adjustments. Working with fuzzy cover in the real data we can face tons of isolated points. To solve it there is a need for a local connectivity.
- In terms of fuzzy open sets, there should be complete confidence that the open set extends as far as the nearest neighbor of each point.



UMAP: make it great for real data 3

- So how to use it? Metrics are local, distances between two particular points disagree.
- There is a bunch of ways to define fuzzy union, but since the weights on the edges can be considered as probabilities, authors suggest to use probablistic triangle conorm $a + b - a \times b$ to join local fuzzy simplical sets in a single fuzzy simplical complex.
- And the first part of algorithm is finalized with the construction of weighted undirected graph of neighboors.



UMAP: finding embedding

- We have a way to represent our data as a graph.
- Ideally we want the low dimensional representation to have as similar a fuzzy topological structure as possible.
- Well, the procedure of constructing it is almost the same with the original data. We want to embed it to the Euclidean space R^d, no more problems with local metric, it is just Euclidean distance.
- The other quirk is that the distances to the nearest neighbors were used. This is a property we would like to be globally true across the manifold as we optimize toward a good low dimensional representation, so we will have to accept it as a hyper-parameter "min-dist" to the algorithm.
- Finally, for optimization fuzzy cross-entropy between the topological representations can be used.

UMAP: asumptions/axioms

- What was stated on the slides above contained a set of assumtions about the original data, and how these assumtions were used to derive the algorithm.
  1. There exists a manifold on which the data would be uniformly distributed.
  2. The underlying manifold of interest is locally connected.
  3. Preserving the topological structure of this manifold is the primary goal.
- Now let's proceed with the computational part of the algorithm.

UMAP: theory 1

The first stage is to build a weighted graph based on observations

- Find k-nearest neighbors for each $x_i \in X$
- For each x_i find ρ_i and σ_i

$$\rho_i = \min \left\{ d(x_i, x_{i_j}) | j = 1, \dots, k \right\}$$

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

UMAP: theory 2

- What is ρ_i? Distance to nearest neighbor. What for? To exactly have at least one edge in a graph with weight 1. And this, in turn, allows us to be sure that the general fuzzy simplicial set (which we will define as a graph) is locally connected at the point x_i.
- What is σ_i for? To set a proper Riemannian metric near x_i (a hack for approaching the satisfiability of the first axiom about the uniform distribution of data on a manifold). It is found by binary search.

UMAP: theory 3

Build the graph $G = (V, E, w)$

- Nodes $V = X$.
- Edges E are links between k-nearest neighbors.
- The weight function w is defined as follows

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

- Then build a symmetric adjacency matrix B based on the non-symmetric adjacency matrix A, consisting of the elements $w((x_i, x_{i_j}))$

$$B = A + A^T - A \odot A^T,$$

where the operation $\odot$ is the componentwise product of matrices.

UMAP: theory 4

- Elements of matrix A, i.e. weights $w((x_i, x_{i_j}))$ can be considered as the probabilities of the existence of an edge between observations. Obviously they are not symmetrical since the metrics are not symmetrical.
- Then the elements of the symmetric adjacency matrix B can be understood as the probability that at least one of the edges between x_i and x_j exists.
- As a result, we have a weighted undirected graph.
- As for the theoretical foundations, why do we combine all this in this way (different local topological structures with different metrics)... You have to deal with the theory of fuzzy simplicial sets described in the theoretical section of the UMAP paper [5].

UMAP: theory 5

The second stage is the construction of embeddings

- Strictly speaking, the constructed graph G is a representation of the fuzzy set A with the membership function $\mu(a), a \in A$.
- We want to get a graph in a space of lower dimension corresponding to the fuzzy set A with the membership function $\nu(a), a \in A$
- Cross-entropy for fuzzy sets

$$C((A,\mu),(A,\nu)) = \sum_{a\in A} \mu(a)\log\left(\frac{\mu(a)}{\nu(a)}\right) + (1-\mu(a))\log\left(\frac{1-\mu(a)}{1-\nu(a)}\right) = \sum_{a\in A}(\mu(a)\log(\mu(a)) + (1-\mu(a))\log(1-\mu(a))) - \sum_{a\in A}(\mu(a)\log(\nu(a)) + (1-\mu(a))\log(1-\nu(a))) \quad (15)$$

UMAP: theory 6

- Since the first part of the sum depends only on the known $\mu(a)$, it can be discarded, i.e. optimize only

$$-\sum_{a \in A} \left(\mu(a) \log\left(\nu(a)\right) + \left(1 - \mu(a)\right) \log\left(1 - \nu(a)\right)\right)$$

- For optimization, it is proposed to use a variation of stochastic gradient descent using edge sampling from graphs for both terms under the sum. In the case of the second term, the so-called negative sampling will be used (word2vec, anyone?).

UMAP: theory 7

- To do this, you need to decide what to do with $\nu(a)$. Well, find a differentiable approximation of this function. The authors do it in the manner of t-SNE

$$\Phi(y_i, y_j) = \left(1 + a||y_i - y_j||_2^{2b}\right)^{-1}$$

- The parameters are selected based on the non-linear least squares fitting against the curve

$$\Psi(y_i, y_j) = \begin{cases} \exp(-(||y_i - y_j||_2 - \text{min-dist})), & \text{otherwise} \\ 1, & ||y_i - y_j||_2 \leqslant \text{min-dist} \end{cases}$$

UMAP: algorithm pseudocode

Algorithm 1 UMAP algorithm

function UMAP(X, n, d, min-dist, n-epochs)

 # Construct the relevant weighted graph

for all $x \in X$ **do**

fs-set$[x] \leftarrow$ LocalFuzzySimplicialSet(X, x, n)

top-rep $\leftarrow \bigcup_{x \in X}$ fs-set$[x]$ *# We recommend the probabilistic t-conorm*

 # Perform optimization of the graph layout

$Y \leftarrow$ SpectralEmbedding(top-rep, d)

$Y \leftarrow$ OptimizeEmbedding(top-rep, Y, min-dist, n-epochs)

return Y

UMAP: graph building algorithm pseudocode

Algorithm 2 Constructing a local fuzzy simplicial set

function LocalFuzzySimplicialSet(X, x, n)

knn, knn-dists $\leftarrow$ ApproxNearestNeighbors(X, x, n)

$\rho \leftarrow$ knn-dists[1] # *Distance to nearest neighbor*

$\sigma \leftarrow$ SmoothKNNDist(knn-dists, n, ρ) # *Smooth approximator to knn-distance*

fs-set$_0 \leftarrow X$

fs-set$_1 \leftarrow \{([x, y], 0) \mid y \in X\}$

for all $y \in$ knn **do**

$d_{x,y} \leftarrow \max\{0, \text{dist}(x, y) - \rho\}/\sigma$

fs-set$_1 \leftarrow$ fs-set$_1 \cup ([x, y], \exp(-d_{x,y}))$

return fs-set

UMAP: σ_i initialization and spectral embedding algorithms pseudocode

Algorithm 3 Compute the normalizing factor for distances σ

function SMOOTHKNNDIST(knn-dists, n, ρ)

Binary search for σ such that $\sum_{i=1}^{n} \exp(-(\text{knn-dists}_i - \rho)/\sigma) = \log_2(n)$

return σ

Algorithm 4 Spectral embedding for initialization

function SPECTRALEMBEDDING(top-rep, d)

$A \leftarrow$ 1-skeleton of top-rep expressed as a weighted adjacency matrix

$D \leftarrow$ degree matrix for the graph A

$L \leftarrow D^{1/2}(D - A)D^{1/2}$

evec $\leftarrow$ Eigenvectors of L (sorted)

$Y \leftarrow \text{evec}[1..d+1]$ *# 0-base indexing assumed*

return Y

UMAP: SGD pseudocode

Algorithm 5 Optimizing the embedding

function OPTIMIZEEMBEDDING(top-rep, Y, min-dist, n-epochs)

$\alpha \leftarrow 1.0$

Fit Φ from Ψ defined by min-dist

for $e \leftarrow 1, \ldots,$ n-epochs **do**

for all $([a, b], p) \in$ top-rep$_1$ **do**

if RANDOM( ) $\leq p$ **then** # *Sample simplex with probability* p

$y_a \leftarrow y_a + \alpha \cdot \nabla(\log(\Phi))(y_a, y_b)$

for $i \leftarrow 1, \ldots,$ n-neg-samples **do**

$c \leftarrow$ random sample from Y

$y_a \leftarrow y_a + \alpha \cdot \nabla(\log(1 - \Phi))(y_a, y_c)$

$\alpha \leftarrow 1.0 - e/$n-epochs

return Y

UMAP: hyperparameters

- Embedding dimension d.
- Number of k-nearest neighbors. Smaller value? Accurate local topology of the manifold. Higher? Perhaps a fairly accurate boundary structure. 10 is ok to start, right?
- "min-dist" sets the level of separation between points in the embedding space. The smaller, the more densely packed local areas with embeddings will be. The higher, the more the points will be scattered over the space of a smaller dimension.

Understanding UMAP [2]

- Hyperparameters really matter (did anyone doubt it?).
- Cluster sizes in a UMAP plot mean nothing.
- Distances between clusters might not mean anything.
- Random noise doesn't always look random.
- You may need more than one plot.

UMAP: discussion 1

Pros

- Provides a general framework for manifold learning and dimensionality reduction.
- Strong mathematical foundation [5].
- Computationally efficient: $O(N^{1.14})$.
- for arbitrary size of embedding.

UMAP: discussion 2

Cons

- Still the same interpretation problems.
- Finding structures in noise since one of the axioms states that there exists low-dimensional manifold in the data.
- Even with the possibility to set big value of k, UMAP tends to focus on the local structure, therefore, in the case when it is necessary to more accurately preserve the general data structure, it is recommended to use a different algorithm (e.g. ISOMAP).
- Well, various kinds of approximations (search for neighbors, approximation of the membership function in a space of smaller dimensions) can spoil the overall picture of the algorithm. Especially for a small sample size.

UMAP: discussion 3

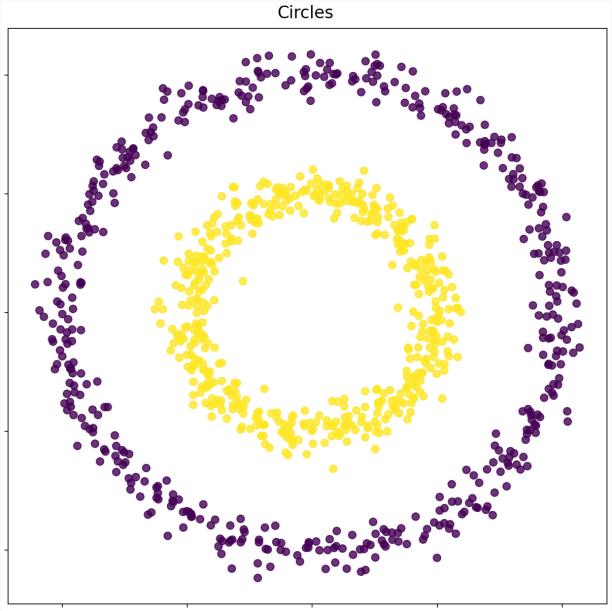
- Lots of similarities with t-SNE (and other k-neighbors graph algorithms): weighted graph, metric is close to the t-Student (except a and b), SGD for optimization fuzzy cross-entropy (KL divergence in t-SNE).
- As for advantages over t-SNE: high performance, strong mathematical background, better global structure (t-SNE is trying it best–remember the gradients–but UMAP is better), easy-to-udnerstand hyperparameters.

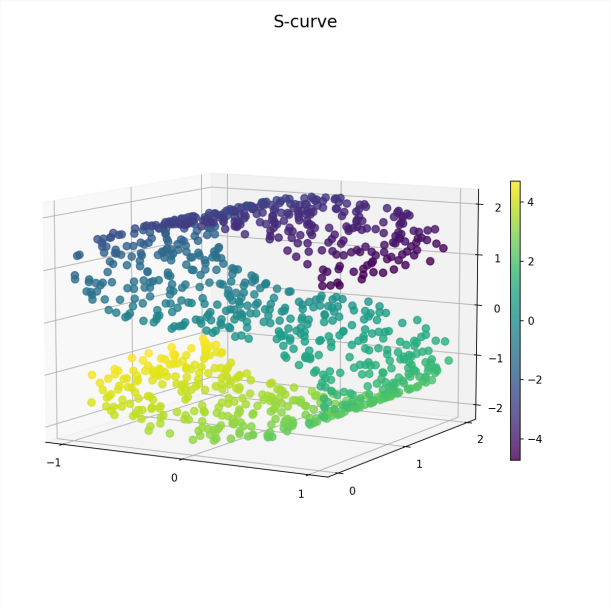
UMAP: discussion 4

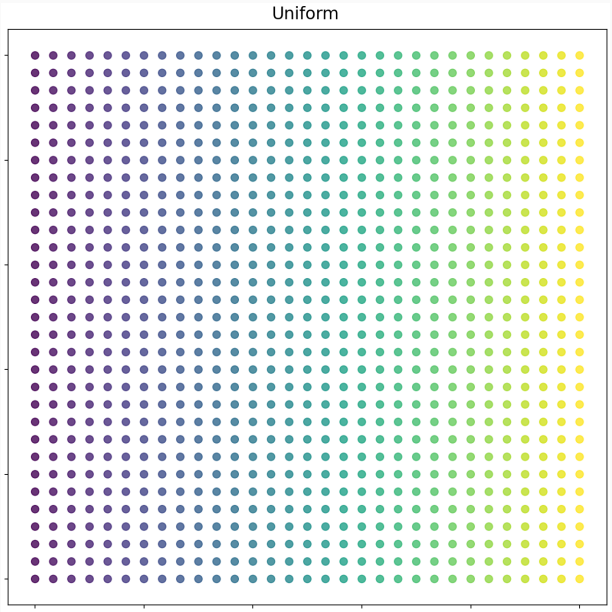
- There's a way to use train labels and consider it as a supervised/semi-supervised problem. There's also an implementation of Parametric UMAP: it replaces the second step, minimizing the same objective function as UMAP, but learning the relationship between the data and embedding using a neural network.
- UMAP implementation has "transform" method [7] (!).
    1. It first initialize the positions of new points relative to the strengths of their neighbors in the source data. If a point is in the original data set it embeds at the original points coordinates. If a point has no neighbours in the original dataset it embeds as the np.nan vector. Otherwise a point is the weighted average of it's neighbours embedding locations.
    2. Second step – optimize the fuzzy cross-entropy between old and new data embeddings using 1-simplices of the old and new original data.

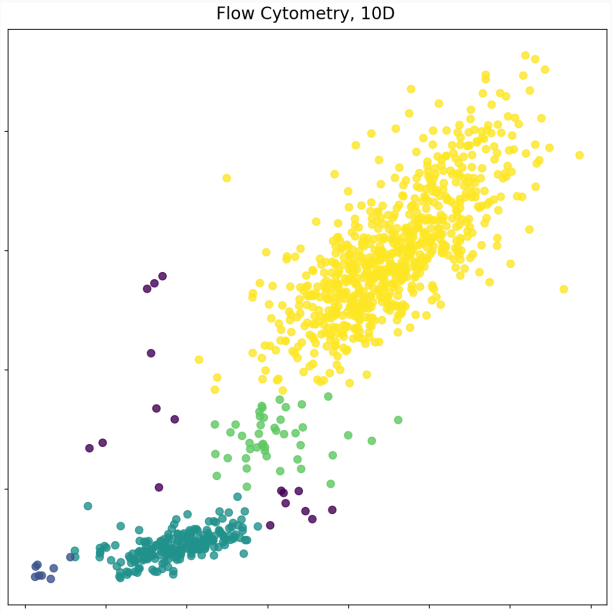
Visualizations

Datasets

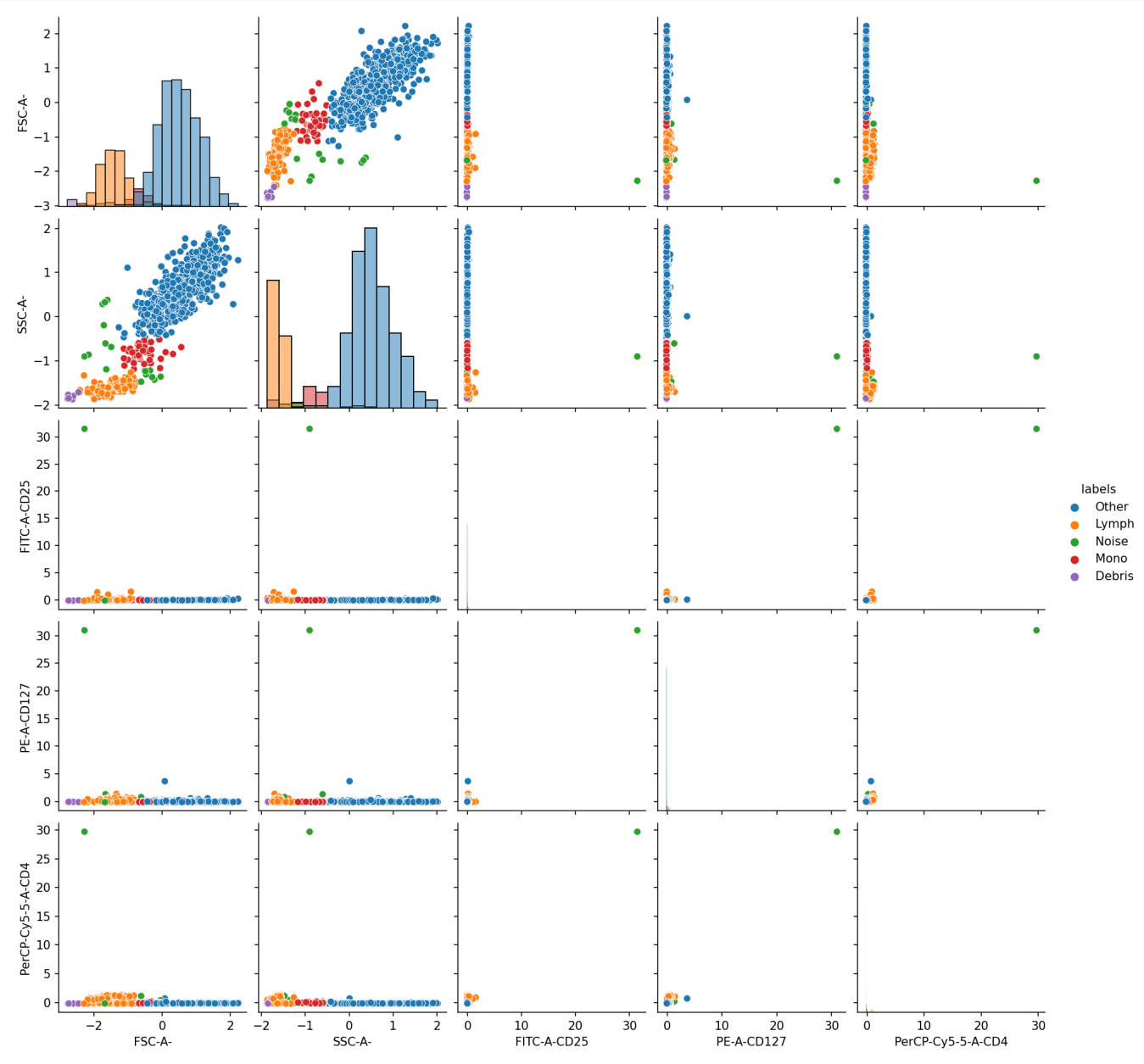




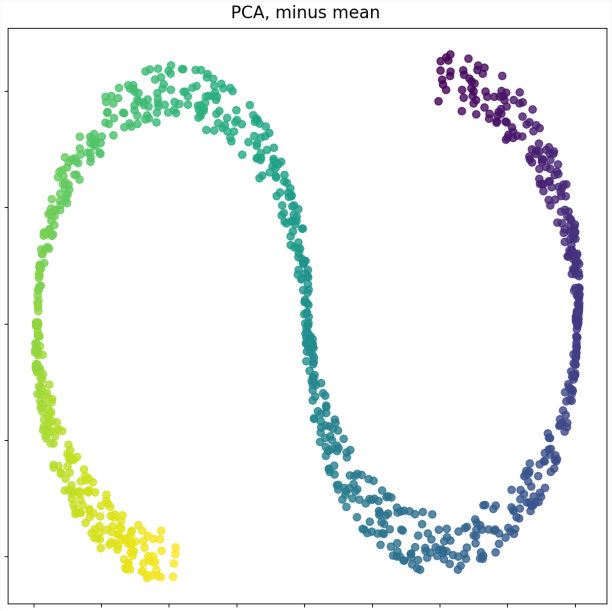


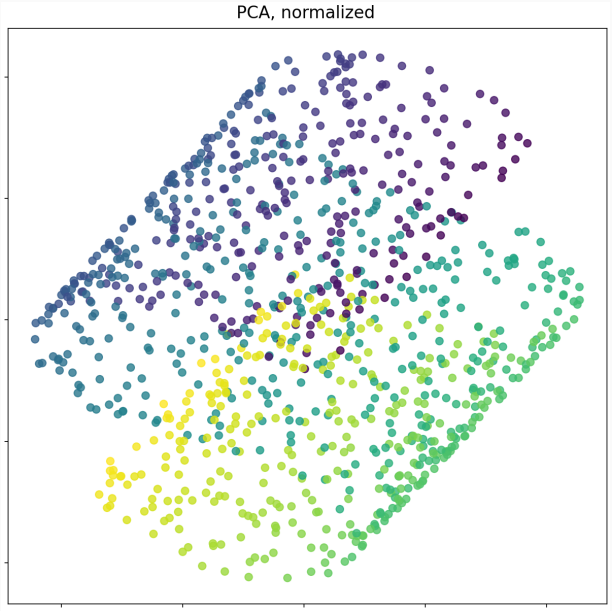


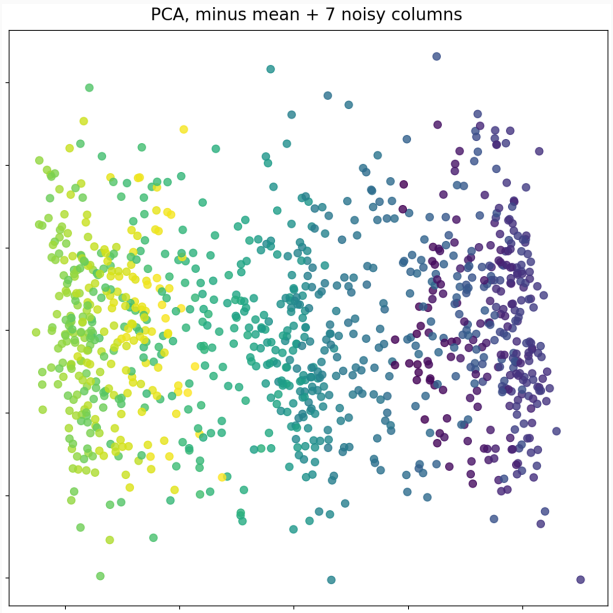
Datasets: flow cytometry pairplot



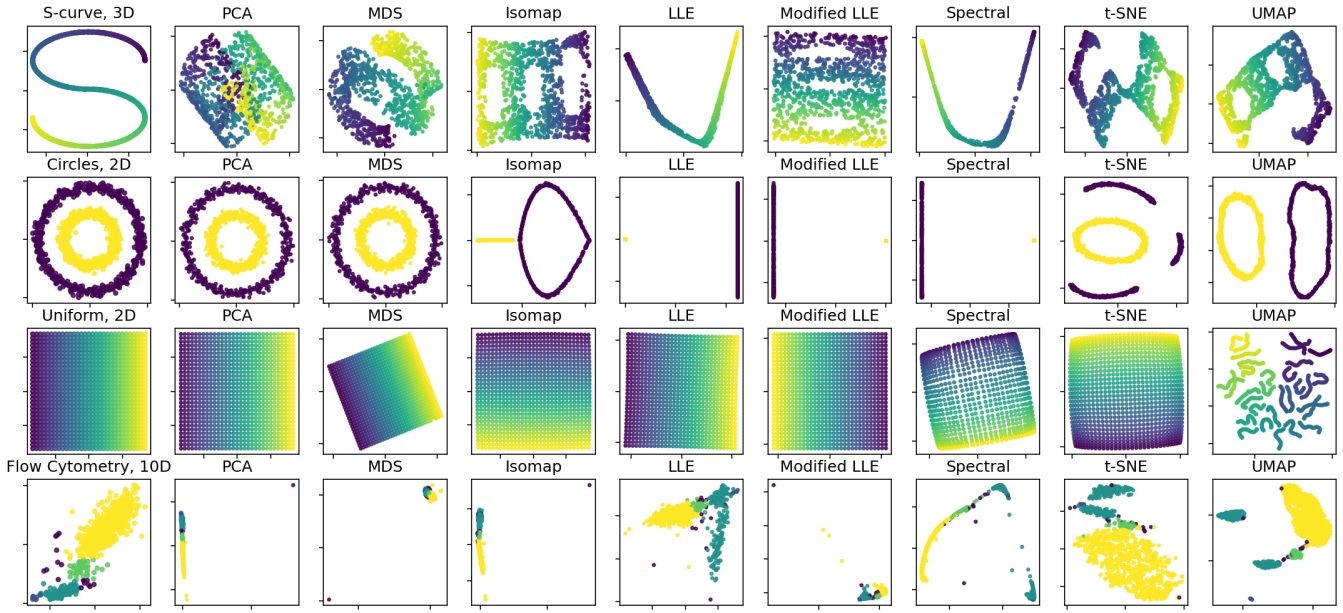
PCA, nonlinearity, noise



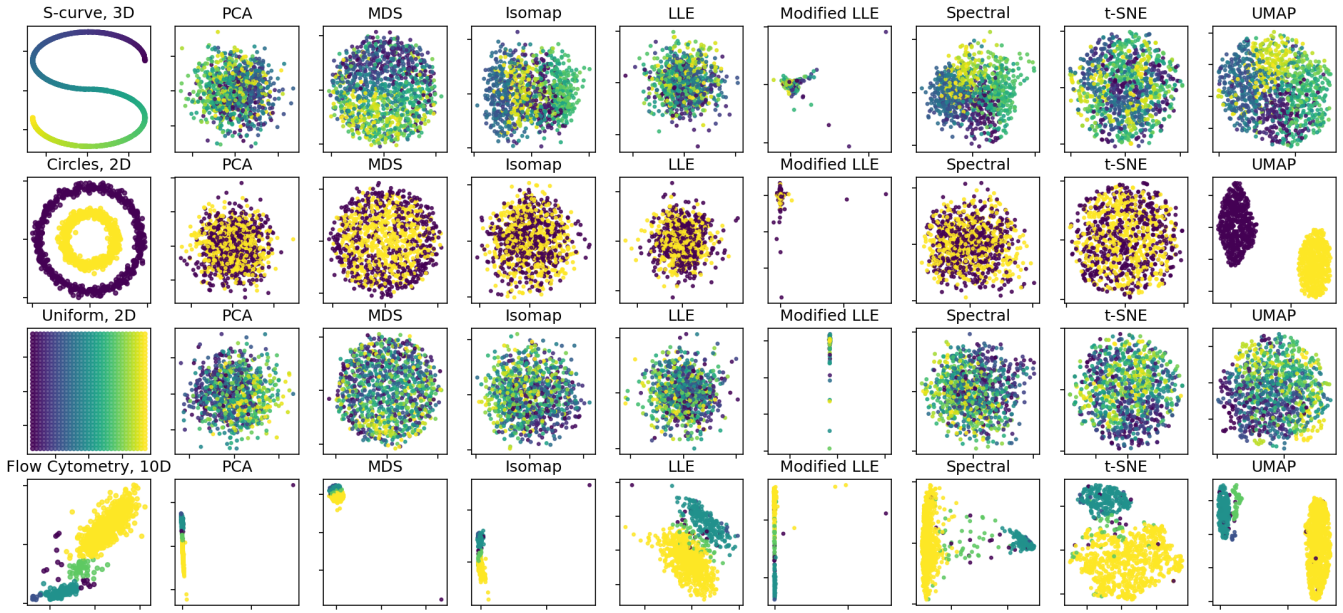




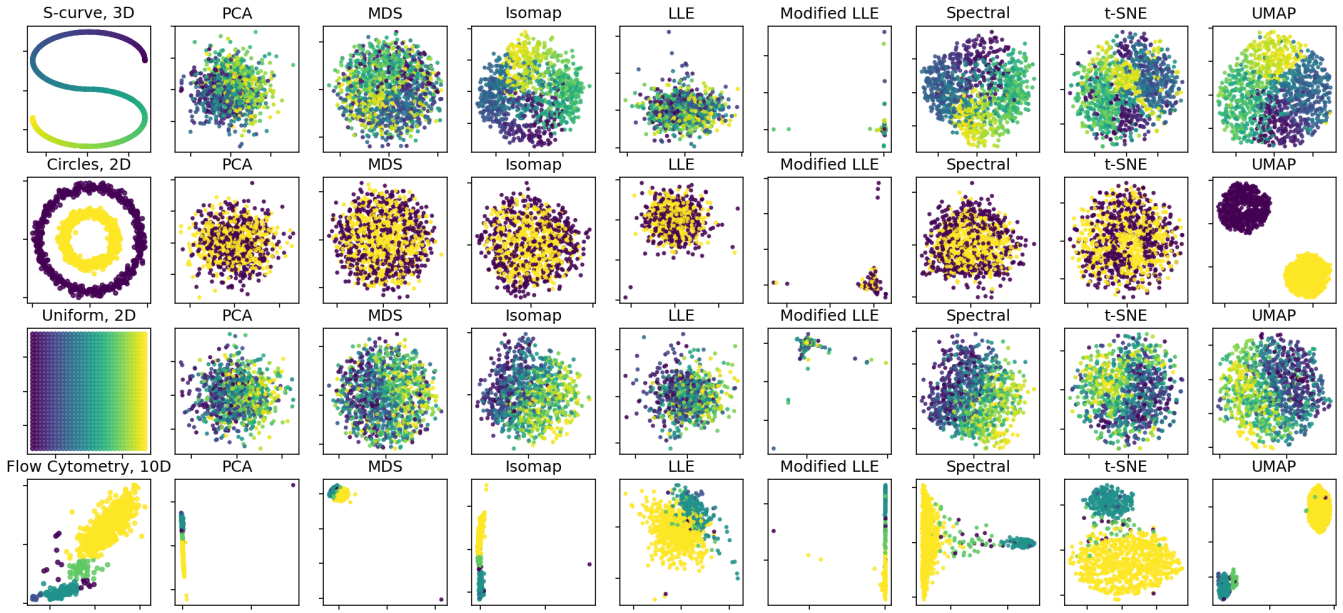
Full comparison: scaled



Full comparison: scaled, uniform noise columns (15D)



Full comparison: scaled, normal noise columns (15D)



Conclusions

- PCA suffers from nonlinear manifolds.
- LLE has issues when the number of neighbors is higher than data dimension. There are some regularization techniques and even methods to overcome it. One is called Modified LLE and uses several weight vectors for every neighborhood.
- Laplasian eigenmaps implementation consider the case of disconnected components and handle it automatically.
- UMAP performance looks really good. t-SNE is also fine. Both of them can deal with noisy columns, that ruin performance for most of the algorithms.
- t-SNE and UMAP perfect guides with visualizations for hyperparams can be found here [14, 2].

Summary

Considered algorithms summary/inter-relationships

- PCA is linear method. Others are generally non-linear.
- Classical MDS with Euclidean distances is equivalent to PCA.
- ISOMAP is simply MDS plus geodesic distances.
- The general pattern of k-nearest neighbor graph algorithms (ISOMAP, LLE, Laplacian eigenmaps, UMAP, and even t-SNE) is as follows:
  1. Building a weighted k-nearest neighbors graph and applying some transformation on distances between vertices to reflect the local data structure (don't forget about symmetry).
  2. Choose a special objective function (reflecting the desired properties) to optimize the embedding in the low-dimensional space.

Additional notes

- Hyperparameters really maters:
  1. dimension d;
  2. the way to construct the graph and the number of nearest neighbors k (ISOMAP, LLE, UMAP);
  3. perplexity in t-SNE;
  4. min-dist in UMAP.
- No discrepancies between theory and implementation of algorithms in sklearn were found. However, there are details associated with various optimization tricks (t-SNE and the Barnes-Hut algorithm), the construction of the Laplacian of the graph (also called the Kirchhoff matrix), etc.
- UMAP looks very promising. And although there is a really good written article, as well as detailed documentation, it is quite difficult to understand the code base.

Other algorithms/approaches 1

- Methods that find subset of most important features.
  - Univariate feature selection. Methods of this group consider features separately and uses different statistical scores/tests for selection. E.g. linear relation with the target using F-score (regression), ANOVA (classification), χ^2 (categorical feature in classification).
  - Model-based. E.g. random forest has some out-of-bag samples during the fit stage that can be used to compute feature improtance.
  - Sequential feature selection. Forward inclusion or backward elimination of features using different metrics for model comparison (NLL, AIC, BIC, etc).

Other algorithms/approaches 2

- Various Autoencoders. Except for the number of study cases, autoencoders usually hard to train and the final profit is low (but this is the topic for another seminar).
- Most of other algorithms noted (but not described) in "General classification of algorithms" section are different extensions of considered algorithms here. Having the ideas described in this seminar it will be much easier (well, I hope so) to understand the math behind them.

Использованные источники i

[1] Ingwer Borg и Patrick J. F. Groenen. *Modern Multidimensional Scaling Theory and Applications*. New York: Springer, 2005. ISBN: 038728981X. DOI: 10.1007/0-387-28981-X.

[2] Andy Coenen и Adam Pearce. *Understanding UMAP*. URL: https://pair-code.github.io/understanding-umap/.

[3] Benyamin Ghojogh и др. "Locally Linear Embedding and its Variants: Tutorial and Survey". В: *ArXiv* abs/2011.10925 (2020).

[4] Laurens van der Maaten. "Accelerating t-SNE using Tree-Based Algorithms". В: *Journal of Machine Learning Research* 15.93 (2014), с. 3221—3245. URL: http://jmlr.org/papers/v15/vandermaaten14a.html.

Использованные источники ii

[5] L. McInnes и J. Healy. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. В: *ArXiv* abs/1802.03426 (2018).

[6] Leland McInnes. *How UMAP Works*. URL: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html.

[7] Leland McInnes. *UMAP transforming new data*. URL: https://umap-learn.readthedocs.io/en/latest/transform.html.

[8] Yen Kaow Ng. *cMDS equivalence to PCA*. URL: https://www.quora.com/Whats-the-difference-between-MDS-and-PCA.

[9] Lawrence Saul и Sam Roweis. “An introduction to locally linear embedding”. В: *Journal of Machine Learning Research* 7 (янв. 2001).

Использованные источники iii

[10] Joshua B. Tenenbaum, Vin de Silva и John C. Langford. “A Global Geometric Framework for Nonlinear Dimensionality Reduction”. В: *Science* 290.5500 (2000), с. 2319—2323. ISSN: 0036-8075. DOI: 10.1126/science.290.5500.2319. URL: https://science.sciencemag.org/content/290/5500/2319.

[11] L.J.P. van der Maaten и G.E. Hinton. “Visualizing High-Dimensional Data Using t-SNE”. В: *Journal of Machine Learning Research* 9 (2008), с. 2579—2605.

[12] Laurens Van der Maaten. *Dimensionality reduction techniques list*. URL: https://lvdmaaten.github.io/drtoolbox/.

[13] Laurens Van Der Maaten, Eric Postma и Jaap Van den Herik. “Dimensionality reduction: a comparative review”. В: *J Mach Learn Res* 10 (2009), с. 66—71.

Использованные источники iv

[14] Martin Wattenberg, Fernanda Viegas и Ian Johnson. *How to use t-SNE Effectively*. URL: https://distill.pub/2016/misread-tsne/.

[15] Prof. Richard C. Wilson. *Similarities, Distances and Manifold Learning*. URL: http://simbad-fp7.eu/images/tutorial/02-ECCV2012Tutorial.pdf.