Lecture 9: Manifold learning

Nonlinear dimensionality reduction

Nils Olovsson
nils.olofsson@igp.uu.se

Introduction to data science and Python programming for medical research
Uppsala University

2025

UPPSALA
UNIVERSITET

Content

Introduction

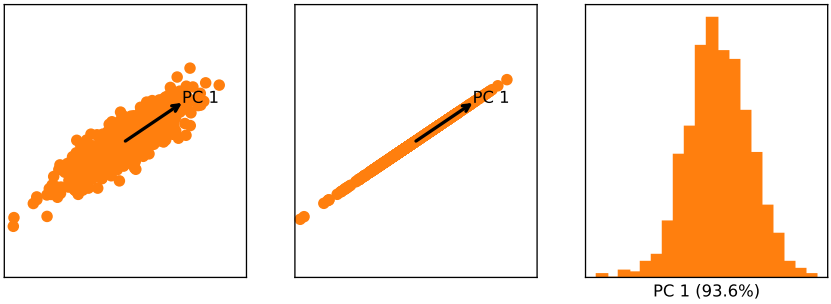
Introduction

Introduction: Problem

Dimensionality reduction with PCA is achieved by **projecting** data points on the first PC vectors.

This **embeds** the data in the **PCA coordinate system**.

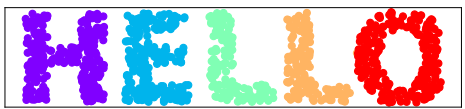
PCA is a **linear method**.



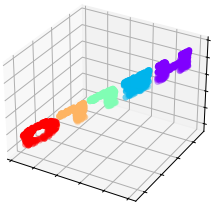
Introduction: Data with nonlinear *shape*

Data points **can be** organized in **non-linear** ways.

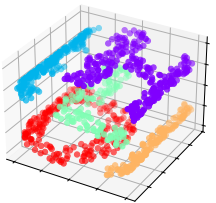
The underlying **shape** or **embedding** created by the data points can be **curved** or **folded**.



Original 2D data.



Linear 3D embedding.



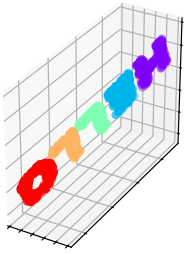
Nonlinear 3D embedding.

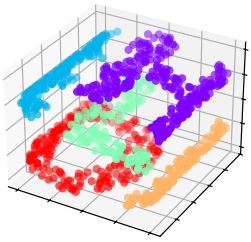
Introduction: PCA fails on non-linear data

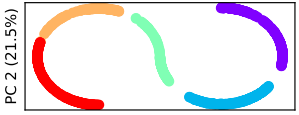
PCA can only model **linearly** embedded data.

This means that the **data** has to be lying **on a flat plane**.
(But the *plane* can have more dimensions than two.)

It will **fail** to represent **non-linear** data.







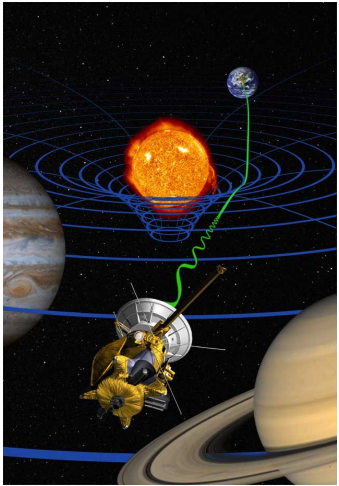
Introduction: Manifolds in physics

In physics and mathematics a **manifold** defines a topological space embedded in a higher dimension.

A **2D manifold** is simply a **surface**.

Can be **flat (linear)** or **curved and folded (non-linear)**.

A crumbled paper is a 2D manifold in 3D space.



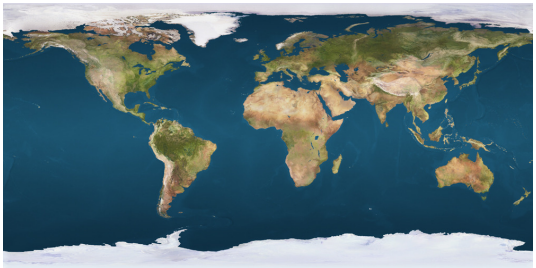
Gravity bends space.

Introduction: Locally flat

A **manifold** will look **flat** where you stand.

Manifold learning attempts to find this flat **embedding**.

Unfolding 3D surface of earth on 2D plane →



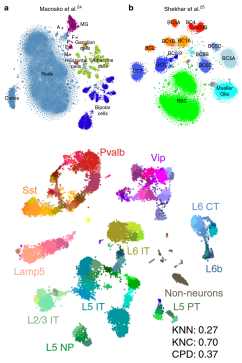
The earth is not flat. But to us on the surface it looks like it.

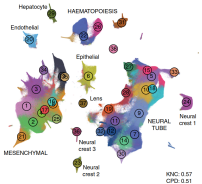
The unfolding distorts the surface, making the poles look much larger than they really are.

Introduction: Manifold analysis of biomedical data

Visualization of **very high dimensional data**, *e.g.* transcriptomics, proteomics etc.

Image **analysis** and **modeling**, *e.g.* respiratory motion.





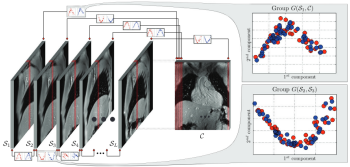


Fig. 3. Schematic of group connections through simultaneous manifold embeddings. Motion fields from neighbouring and orthogonal slices can be embedded simultaneously using appropriate similarity kernels leading to aligned embeddings. Two close-up views of aligned manifold embeddings, originating from a dataset with 50 motion fields per slice position, are shown on the right hand side.

https://doi.org/10.1038/s41467-019-13056-x

https://doi.org/10.1016/j.media.2016.06.005

Introduction: Manifold learning methods

Find mapping such that points close in original data are close after mapping. (Locally flat.)

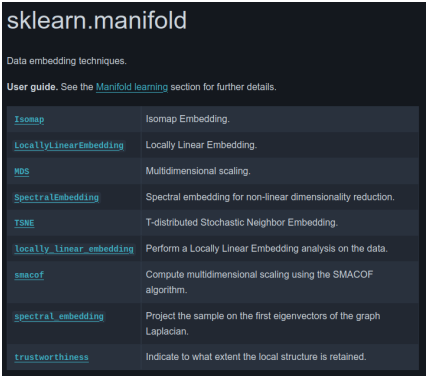
Will look at several methods that can be divided into two families.

- MDS
- Isomap
- Eigenmap
- LLE

} Spectral methods

- t-SNE
- UMAP

} Optimization (Gradient descent)



sklearn.manifold

Data embedding techniques.

User guide. See the Manifold learning section for further details.

Isomap	Isomap Embedding.
LocallyLinearEmbedding	Locally Linear Embedding.
MDS	Multidimensional scaling.
SpectralEmbedding	Spectral embedding for non-linear dimensionality reduction.
TSNE	T-distributed Stochastic Neighbor Embedding.
locally_linear_embedding	Perform a Locally Linear Embedding analysis on the data.
smacof	Compute multidimensional scaling using the SMACOF algorithm.
spectral_embedding	Project the sample on the first eigenvectors of the graph Laplacian.
trustworthiness	Indicate to what extent the local structure is retained.

https://scikit-learn.org/stable/api/sklearn.manifold.html

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

Introduction: Toy datasets

First set of methods illustrated using toy data.

Hello (Flat)

Should recover the 2D letters in *Hello*.

Hello (S-shaped)

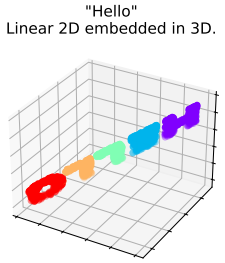
Should recover the 2D letters in *Hello*.

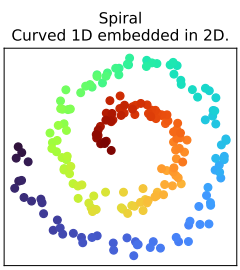
Spiral

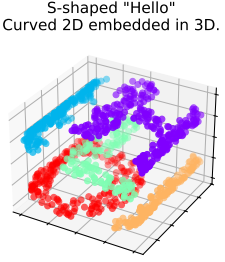
Should recover the 1D line going from blue to red.

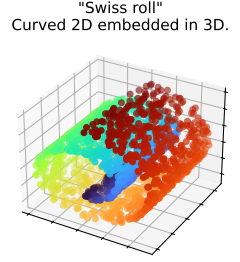
Swiss roll

Should recover the 2D sheet going from blue to red.









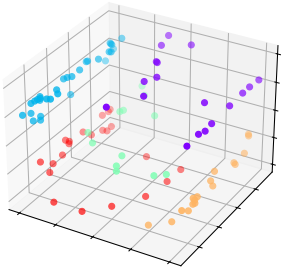
Multidimensional scaling (MDS)

Multidimensional scaling (MDS): Distance matrix

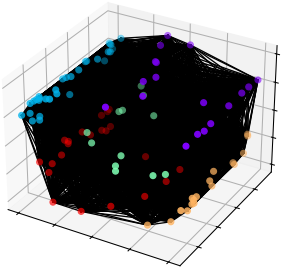
Finds a **manifold embedding** while minimizing a *strain* between data points.

Fully connected **distance matrix**.
Every point is connected to all other points.

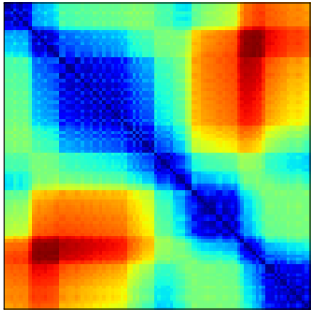
Data points
(100 points)



Dense connections



Distance matrix
(100x100 matrix)



Multidimensional scaling (MDS): Eigenvalue problem

1. Distance matrix D
2. Double center the D.

$$B = -\frac{1}{2}CDC, \qquad C = I - \frac{1}{N}J$$

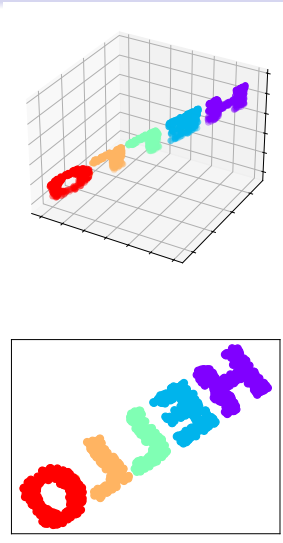
3. Decompose into eigenvectors V and eigenvalues e
4. Embedding Y in first M dimensions is defined by

$$Y = \{\sqrt{e_1}V_1, \cdots, \sqrt{e_M}V_M\}$$

```python
def fit_transform(
    self,
    X: np.ndarray,
) -> None:
    # Pairwise distance between all points
    D = sklearn.metrics.pairwise_distances(X)
    D2 = np.pow(D, 2)

    # Apply double centering
    N = D.shape[0]
    J = np.ones(D.shape, dtype=np.float64)
    I = np.eye(N, dtype=np.float64)
    C = I - (1.0/N) * J
    B = -0.5 * C @ D2 @ C

    # Eigendecomposition
    e, V = np.linalg.eigh(B)
    e = np.flip(e)
    V = np.flip(V, axis=1)
    V = V[:, 0:self._n_dimensions]
    e = e[0:self._n_dimensions]
    e = np.sqrt(e)

    # Calculate low dimension embedding Y
    Y = np.zeros(V.shape, dtype=np.float64)
    for i in range(self._n_dimensions):
        Y[:,i] = e[i] * V[:,i]
    return Y
```

Multidimensional scaling (MDS): Example (linear)

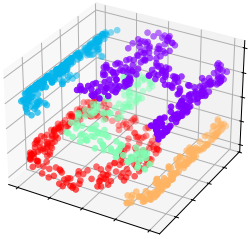
```python
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import MDS
# Create data and project into 3D
N = 1000
X, y = make_hello(n_samples=N, rseed=0)
colorize = dict(c=X[:,0],
                cmap=plt.get_cmap('rainbow', 5))
X = random_projection_to_higher_dimension(X, 3,
     rseed=0)
# Fit MDS, transform from 3D to 2D
mds = MDS(n_components=2)
X_mds = mds.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
     (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_mds[:,0], X_mds[:,1],
            marker='o', **colorize)
#plt.show()
```

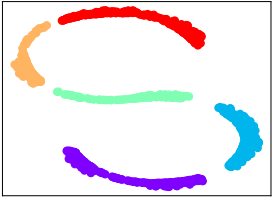


Multidimensional scaling (MDS): Example (S-shape)

Because MDS preserves all distances it **can't untangle non-linear data**.

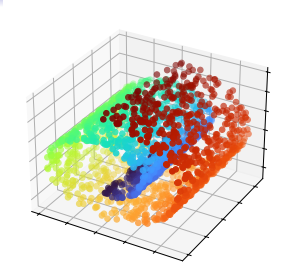
Similarly to PCA it is limited to linear data which it can recover, disregarding orientation.

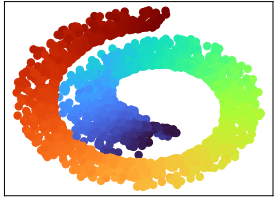




Multidimensional scaling (MDS): Example (roll)

```python
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import MDS
from sklearn.datasets import make_swiss_roll
# Create data and project into 3D
N = 2000
X, y = make_swiss_roll(n_samples=N,
                       noise=0.1,
                       random_state=0)
colorize = dict(c=y,
                cmap=plt.get_cmap('turbo'))
# Fit MDS, transform from 3D to 2D
mds = MDS(n_components=2)
X_mds = mds.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
      (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_mds[:,0], X_mds[:,1],
            marker='o', **colorize)
#plt.show()
```





Isomap

Isomap

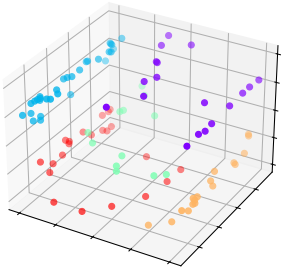
Isomap: Nearest neighbor distance matrix

Finds a **manifold embedding** while minimizing a *strain* between data points same as for MDS.

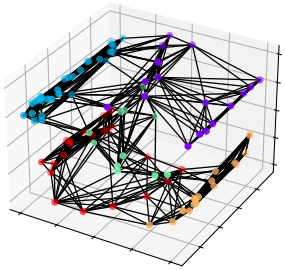
Only **connect** points to their **nearest neighbors**.

We want the manifold to preserve **local structure**!

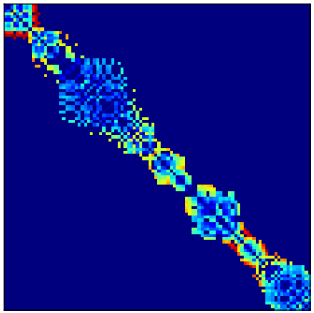
Data points
(100 points)



Connect to k=10
nearest neighbors



Distance matrix
(100x100 matrix)

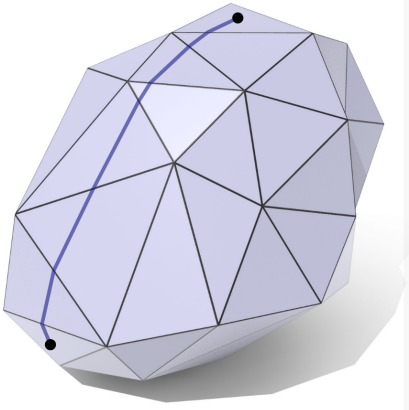


Isomap: Geodesic distance

Distances between non-connected points are now computed by finding their shortest pairwise distances on the graph.

This approximates their **geodesic** distance.

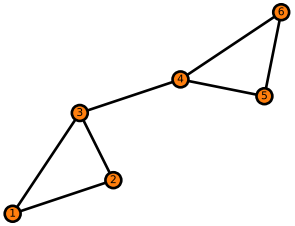
A **geodesic** is the **shortest path** in a curved space, *i.e.* on a **manifold**.



https://brickisland.net/ddg-web/

Isomap: Geodesic matrix

The zeros in the distance matrix, D are filled in by traversing the connections to find the geodesic distances and create geodesic matrix G. (Floyd-Warshall algorithm)



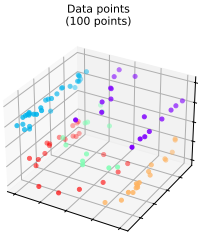
$$D = \begin{pmatrix} 0.0 & 0.6 & 0.7 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.0 & 0.4 & 0.0 & 0.0 & 0.0 \\ 0.7 & 0.4 & 0.0 & 0.6 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.6 & 0.0 & 0.5 & 0.7 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.7 & 0.5 & 0.0 \end{pmatrix}$$

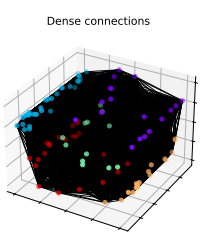
```
# For the zero entries in the distance matrix,
# compute estimates of the geodesic distances
# between pairs of points by computing
# distances on the graph with weights
# according to the Euclidean distances.
G = scipy.sparse.csgraph.floyd_warshall(D)
G2 = np.power(G, 2)
G2 = np.maximum(G2, G2.T)
```

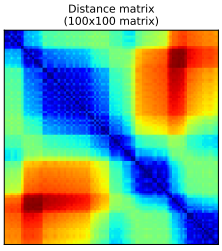
$$G_2 = \begin{pmatrix} 0.0 & 0.4 & 0.5 & \mathbf{1.8} & \mathbf{3.5} & \mathbf{4.3} \\ 0.4 & 0.0 & 0.2 & \mathbf{1.2} & \mathbf{2.5} & \mathbf{3.2} \\ 0.5 & 0.2 & 0.0 & 0.4 & \mathbf{1.3} & \mathbf{1.8} \\ \mathbf{1.8} & \mathbf{1.2} & 0.4 & 0.0 & 0.3 & 0.5 \\ \mathbf{3.5} & \mathbf{2.5} & \mathbf{1.3} & 0.3 & 0.0 & 0.3 \\ \mathbf{4.3} & \mathbf{3.2} & \mathbf{1.8} & 0.5 & 0.3 & 0.0 \end{pmatrix}$$

Isomap: Geodesic matrix vs MDS distance matrix

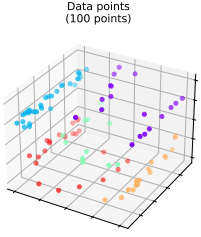
MDS:

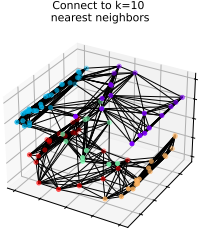


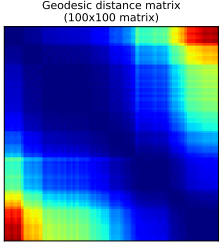




Isomap:







Isomap: Eigenvalue problem

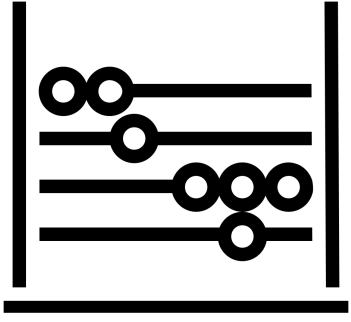
1. Sparse distance matrix D
2. Squared geodesic matrix G_2
3. Double center the G_2.

$$B = -\frac{1}{2} C G_2 C, \qquad C = I - \frac{1}{N} J$$

4. Decompose into eigenvectors V and eigenvalues e
5. Embedding Y in first M dimensions is defined by

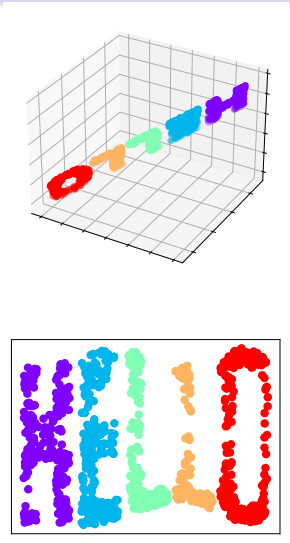
$$Y = \{\sqrt{e_1} V_1, \cdots, \sqrt{e_M} V_M\}$$

Same eigenvalue problem as for MDS but swap D for G_2!



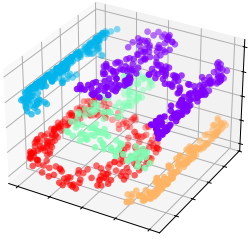
Isomap: Example (linear)

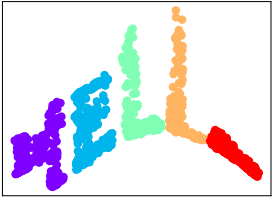
```
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import Isomap
# Create data and project into 3D
N = 1000
X, y = make_hello(n_samples=N, rseed=0)
colorize = dict(c=X[:,0],
                cmap=plt.get_cmap('rainbow', 5))
X = random_projection_to_higher_dimension(X, 3,
     rseed=0)
# Fit Isomap, transform from 3D to 2D
iso = Isomap(n_components=2,
             n_neighbors=75,
             eigen_solver='dense')
X_iso = iso.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
     (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_iso[:,0], X_iso[:,1],
            marker='o', **colorize)
#plt.show()
```



Isomap: Example (S-shape)

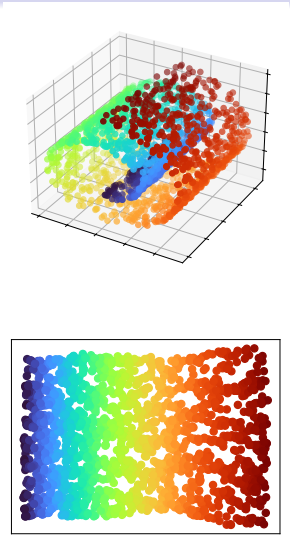
Because Isomap preserves local distances
non-linear data <u>can</u> be untangled!





Isomap: Example (roll)

```python
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import Isomap
from sklearn.datasets import make_swiss_roll
# Create data and project into 3D
N = 2000
X, y = make_swiss_roll(n_samples=N,
                       noise=0.1,
                       random_state=0)
colorize = dict(c=y,
                cmap=plt.get_cmap('turbo'))
# Fit Isomap, transform from 3D to 2D
iso = Isomap(n_components=2,
             n_neighbors=10,
             eigen_solver='dense')
X_iso = iso.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
     (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_iso[:,0], X_iso[:,1],
            marker='o', **colorize)
#plt.show()
```



Laplacian eigenmap

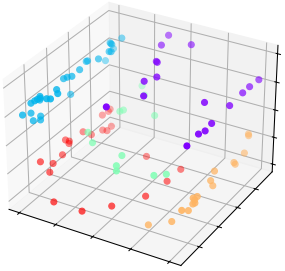
Laplacian eigenmap

Laplacian eigenmap: Nearest neighbor adjacency matrix

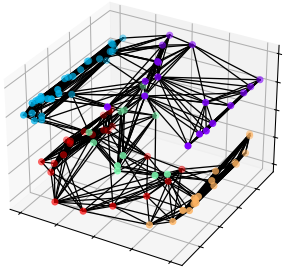
Only connect points to their neighbors.
Discard distance, keep only connections.

$$\begin{cases} 1 : \text{Points are connected} \\ 0 : \text{Points are not connected} \end{cases}$$

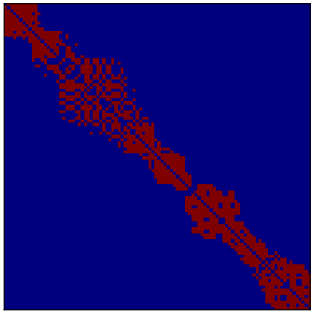
Data points
(100 points)



Connect to k=10
nearest neighbors

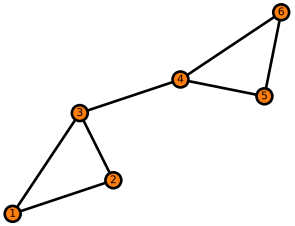


Adjacency matrix
(100x100 matrix)



Laplacian eigenmap: Adjacency matrix

Connect the k=2 nearest points.




```
N = X.shape[0]
A0 = np.zeros((N, N), dtype=np.float64)
for i in range(N):
    x_i = X[i, :]
    distances = np.linalg.norm(X - x_i, axis
      =1)
    indices = np.argsort(distances)
    indices = indices[1:k + 1]
    A0[i, indices] = 1
# Symmetrize the matrix
A = np.maximum(A0, A0.T)
```

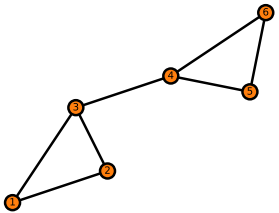
$$D = \begin{pmatrix} 0.0 & 0.6 & 0.7 & 1.3 & 1.7 & 2.0 \\ 0.6 & 0.0 & 0.4 & 0.7 & 1.0 & 1.4 \\ 0.7 & 0.4 & 0.0 & 0.6 & 1.1 & 1.3 \\ 1.3 & 0.7 & 0.6 & 0.0 & 0.5 & 0.7 \\ 1.7 & 1.0 & 1.1 & 0.5 & 0.0 & 0.5 \\ 2.0 & 1.4 & 1.3 & 0.7 & 0.5 & 0.0 \end{pmatrix}$$

$$A_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Laplacian eigenmap: Degree matrix

Degree matrix H describes **how many connections** a data point have. The matrix is **diagonal**.



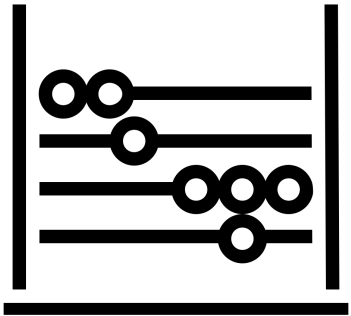
$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad H = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Graph Laplacian operator is defines as $L = H - A$.
Normalized **graph Laplacian** for this eigenvalue problem defined as
$L = H^{-\frac{1}{2}}(H - A)H^{-\frac{1}{2}}$

Laplacian eigenmap: Eigenvalue problem

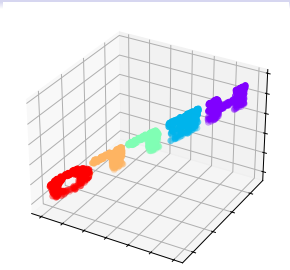
1. Distance matrix D
2. Degree matrix H.
3. Normalized graph Laplacian $L = H^{-\frac{1}{2}}(H - A)H^{-\frac{1}{2}}$
4. Decompose into eigenvectors V and eigenvalues e
5. Ignore the first eigenvector since eigenvalue is always 0. Embedding Y in first M dimensions is defined by

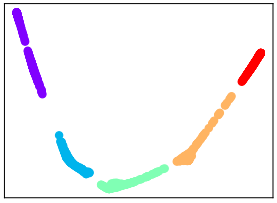
$$Y = \{\sqrt{e_2}V_2, \cdots, \sqrt{e_{M+1}}V_{M+1}\}$$



Laplacian eigenmap: Example (linear)

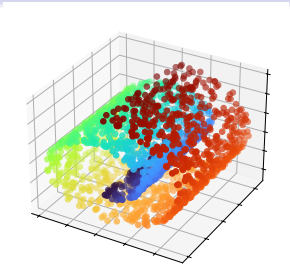
```
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import SpectralEmbedding
# Create data and project into 3D
N = 1000
X, y = make_hello(n_samples=N, rseed=0)
colorize = dict(c=X[:,0],
                cmap=plt.get_cmap('rainbow', 5))
X = random_projection_to_higher_dimension(X, 3,
    rseed=0)
# Fit Isomap, transform from 3D to 2D
lap = SpectralEmbedding(n_components=2,
                        n_neighbors=75)
X_lap = lap.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
    (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_lap[:,0], X_lap[:,1],
            marker='o', **colorize)
#plt.show()
```

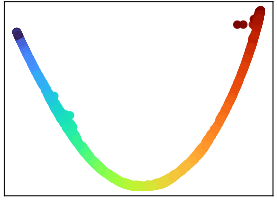




Laplacian eigenmap: Example (roll)

```python
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import SpectralEmbedding
from sklearn.datasets import make_swiss_roll
# Create data and project into 3D
N = 2000
X, y = make_swiss_roll(n_samples=N,
                       noise=0.1,
                       random_state=0)
colorize = dict(c=y,
                cmap=plt.get_cmap('turbo'))
# Fit Spec., transform from 3D to 2D
lap = SpectralEmbedding(n_components=2,
                        n_neighbors=40)
X_lap = lap.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
      (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_lap[:,0], X_lap[:,1],
            marker='o', **colorize)
#plt.show()
```





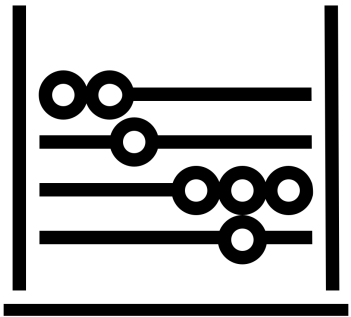
Locally linear embedding (LLE)

Locally linear embedding (LLE): Eigenvalue problem

Optimize a weight matrix W that reconstructs a data point from a weighted sum of its neighbors.

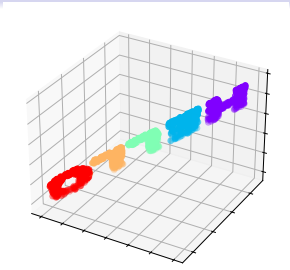
1. Weight matrix W
2. Reconstruction matrix $K = (I - W)^\intercal (I - W)$
3. Decompose into eigenvectors V and eigenvalues e
4. Embedding Y in first M dimensions is defined by

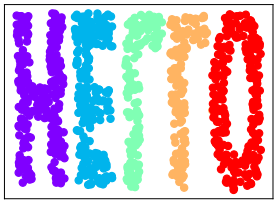
$$Y = \{\sqrt{e_1} V_1, \cdots, \sqrt{e_M} V_M\}$$



Locally linear embedding (LLE): Example (linear)

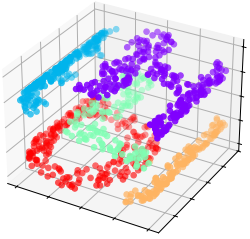
```python
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import LocallyLinearEmbedding
# Create data and project into 3D
N = 1000
X, y = make_hello(n_samples=N, rseed=0)
colorize = dict(c=X[:,0],
                cmap=plt.get_cmap('rainbow', 5))
X = random_projection_to_higher_dimension(X, 3,
     rseed=0)
# Fit LLE, transform from 3D to 2D
lle = LocallyLinearEmbedding(n_components=2,
                             n_neighbors=75,
                             method='modified')
X_lle = lle.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
     (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_lle[:,0], X_lle[:,1],
            marker='o', **colorize)
#plt.show()
```

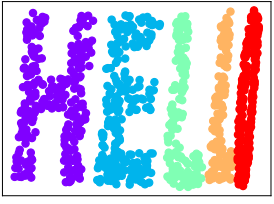




Locally linear embedding (LLE): Example (S-shape)

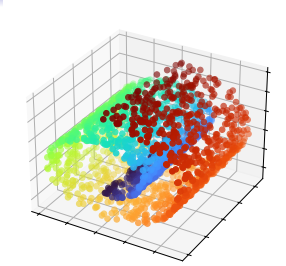
Because LLE preserves local distances
non-linear data <u>can</u> be untangled!

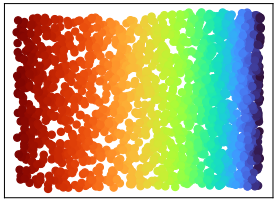




Locally linear embedding (LLE): Example (roll)

```python
import numpy as np
import matplotlib.pyplot as plt
from  matplotlib import gridspec
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.datasets import make_swiss_roll
# Create data and project into 3D
N = 2000
X, y = make_swiss_roll(n_samples=N,
                       noise=0.1,
                       random_state=0)
colorize = dict(c=y,
                cmap=plt.get_cmap('turbo'))
# Fit Spec., transform from 3D to 2D
lle = LocallyLinearEmbedding(n_components=2,
                             n_neighbors=10,
                             method='modified')
X_lle = lle.fit_transform(X)
# Plot
fig = plt.figure(figsize=1.0 * plt.figaspect
     (2.0/1.0))
gs = gridspec.GridSpec(nrows=2, ncols=1,
                       height_ratios=[1.5, 1])
# First subplot
ax0 = fig.add_subplot(gs[0], projection='3d')
ax1 = fig.add_subplot(gs[1])
ax0.scatter(X[:,0], X[:,1], X[:,2],
            marker='o', **colorize)
ax1.scatter(X_lle[:,0], X_lle[:,1],
            marker='o', **colorize)
#plt.show()
```





Comparison

Comparison

Comparison: Eigenvalue problems

All methods so far have been based on **eigen decomposition** of some **connection matrix**.

Methods are **linear** but the **embeddings** are **non-linear**.

MDS

$B = -\frac{1}{2}CDC$

Isomap

$B = -\frac{1}{2}CG_2C$

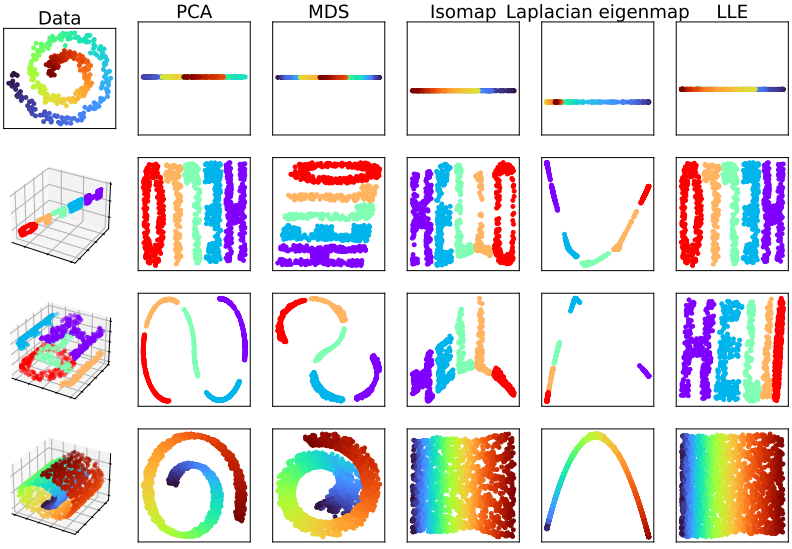
Laplacian eigenmap

$L = H^{\frac{-1}{2}}(H - A)H^{\frac{-1}{2}}$

Locally linear embedding

$K = (I - W)^{\intercal}(I - W)$

Comparison: Results



Comparison: Manifold learning vs PCA

Eigenvectors

PCA — Embed data by projecting it on eigenvectors.

Manifolds — Eigenvectors are the embedding.

Applying model

PCA — Project data on eigenvectors.

Manifolds — Data is the model. Find points close in data space and interpolate in embedded space.

Matrix size

PCA — Covariance matrix size determines by dimensionality of data.

Manifolds — Connection matrix size determined by number of data points.

(Matrix size matters for computational resources!

100 000 data points $\rightarrow$ 100 000 $\times$ 100 000 matrix requires 40 GB!)

```python
import numpy as np
from sklearn.datasets import
      make_swiss_roll
from sklearn.metrics import
      pairwise_distances
X, _ = make_swiss_roll(n_samples=1000)
C = np.cov(X.T)
D = pairwise_distances(X)
print("Data dimensions:")
print(X.shape[1])
print("Number of data points:")
print(X.shape[0])
print("Covariance matrix size:")
print(C.shape)
print("Distance matrix size:")
print(D.shape)
```

```
>> Data dimensions:
>> 3
>> Number of data points:
>> 1000
>> Covariance matrix size:
>> (3, 3)
>> Distance matrix size:
>> (1000, 1000)
```

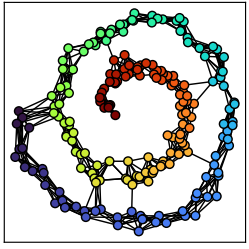
Comparison: Problems

Isomap, Laplacian eigenmaps and LLE all need a parameter for **k = number of neighbors**. Can lead to **problems**.

Manifold shortcuts

Points that should not be connected are connected.

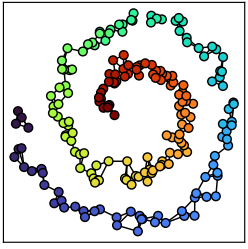
Solution: Decrease k.



Disjoint graph

The graph does not connect all points. (Eigenvalue problem is ill posed.)

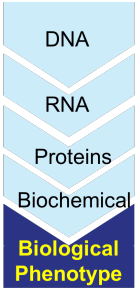
Solution: Increase k.



Real world

Real world

Real world: Single cell analysis

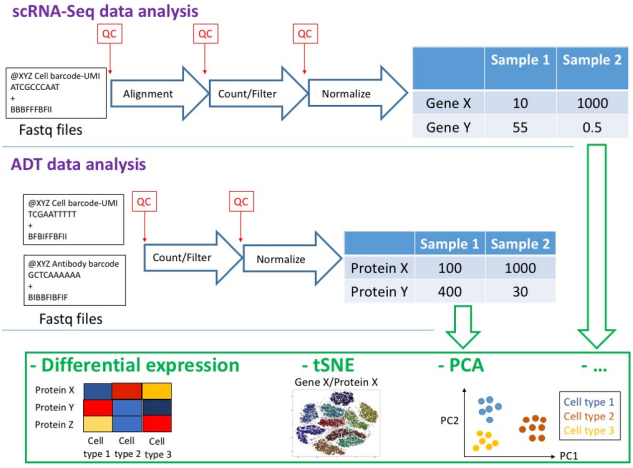


Genomics

Transcriptomics

Proteomics

Metabolomics

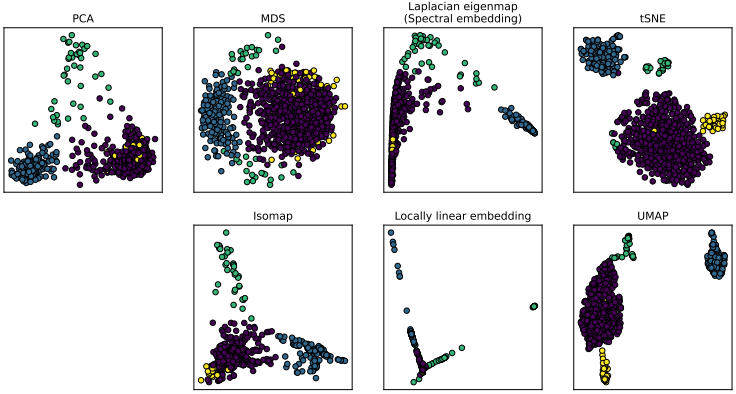


https://en.wikipedia.org/wiki/Metabolomics
https://en.wikipedia.org/wiki/CITE-Seq

Real world: Clustered data

Manifold learning used for visualization assumes **data** to be highly **clustered**, *e.g.* according to cell type.

Motivates specialized methods like **tSNE** and **UMAP**.

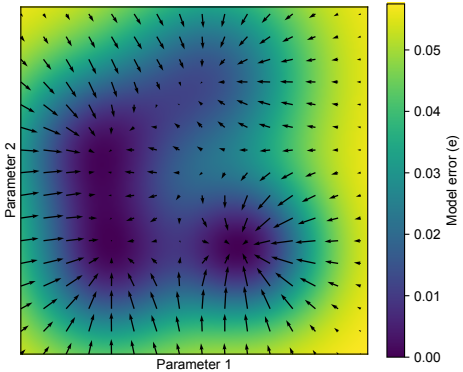


Real world: tSNE and UMAP

tSNE and UMAP are the two most commonly used manifold learning methods for visualizing data from cell assays.

Probabilistic interpretation of whether **points belong together** in **high** and **low dimensional** space.

Methods themselves are **nonlinear**, can't be solved by eigen decomposition. Must be **solved by** optimization, *e.g.* **gradient descent**.



Follow the gradient arrows to find values for parameters 1 and 2 that minimize the model error.

tSNE

tSNE

tSNE: Overview

t-distributed stochastic neighborhood embedding (tSNE).

Described by four equations:

1. Distance in high dimensional data space.
2. Perplexity
3. Distance in low dimensional embedding.
4. Cost function to minimize.

Journal of Machine Learning Research 9 (2008) 2579-2605 Submitted 5/08; Revised 9/08; Published 11/08

Visualizing Data using t-SNE

Laurens van der Maaten LVDMAATEN@GMAIL.COM
TiCC
Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands

Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
6 King's College Road, M5S 3G4 Toronto, ON, Canada

Editor: Yoshua Bengio

Abstract

We present a new technique called "t-SNE" that visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002) that is much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map. t-SNE is better than existing techniques at creating a single map that reveals structure at many different scales. This is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints. For visualizing the structure of very large data sets, we show how t-SNE can use random walks on neighborhood graphs to allow the implicit structure of all of the data to influence the way in which a subset of the data is displayed. We illustrate the performance of t-SNE on a wide variety of data sets and compare it with many other non-parametric visualization techniques, including Sammon mapping, Isomap, and Locally Linear Embedding. The visualizations produced by t-SNE are significantly better than those produced by the other techniques on almost all of the data sets.

Keywords: visualization, dimensionality reduction, manifold learning, embedding algorithms, multidimensional scaling

1. Introduction

Visualization of high-dimensional data is an important problem in many different domains, and deals with data of widely varying dimensionality. Cell nuclei that are relevant to breast cancer, for example, are described by approximately 30 variables (Street et al., 1993), whereas the pixel intensity vectors used to represent images or the word-count vectors used to represent documents typically have thousands of dimensions. Over the last few decades, a variety of techniques for the visualization of such high-dimensional data have been proposed, many of which are reviewed by de Oliveira and Levkowitz (2003). Important techniques include iconographic displays such as Chernoff faces (Chernoff, 1973), pixel-based techniques (Keim, 2000), and techniques that represent the dimensions in the data as vertices in a graph (Battista et al., 1994). Most of these techniques simply provide tools to display more than two data dimensions, and leave the interpretation of the

tSNE: (1) High dimensional distances

Probability of observing distances between points in high dimensional, data, space assumed to be normally distributed.
The probability is normalized.

Joint probability is symmetrized such that $p_{ij} = p_{ji}$.

Conditional probability:

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

Joint probability:

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

tSNE: (2) Perplexity

The *perplexity* is a value provided by the user as method parameter that influences the solution.

It is then used "in reverse" to solve for a value of the standard deviations σ_i in equation 1.

Computed from the entropy.

$$\text{perplexity} = 2^{-\sum_j \left(p_{j|i} \log_2(p_{j|i})\right)}$$

perplexity : *float, default=30.0*

The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results. The perplexity must be less than the number of samples.

tSNE: (3) Low dimensional distances

Probability of observing distances between points in low dimensional, embedding, space assumed to be t-distributed.
The probability is normalized.

(This is actually a t-distribution with one degree of freedom, also called Cauchy distribution.)

$$q_{ji} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{i \neq j} (1 + ||y_i - y_j||^2)^{-1}}$$

tSNE: (4) Cost function

The cost function is based on the **Kullback-Leibler divergence** between the probability distributions in high dimensional data space, P, and in low dimensional embedding, Q.

The embedding is optimized by minimizing the cost function.

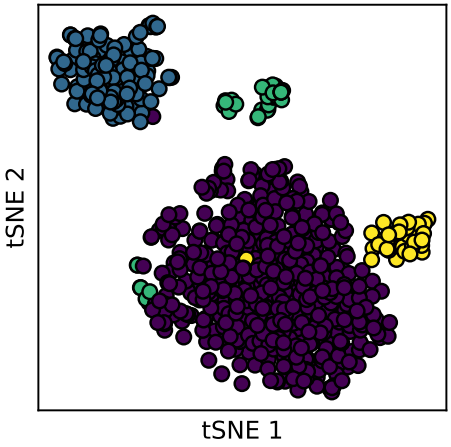
This is done by calculating the gradient of KL with respect to each embedding position.

Minimized by gradient descent.

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

tSNE: Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Cancer associated fibroblast dataset and transform
X, y = load_CAF()
X = np.log(X + 1)

# Instantiate and apply TSNE
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X)

# Plot result
ax = plt.figure(figsize=(3,3)).gca()
ax.scatter(X_tsne[:,0],
           X_tsne[:,1],
           marker='o',
           c=y.astype(int),
           edgecolor='k',
           )
ax.set_xlabel('tSNE 1')
ax.set_ylabel('tSNE 2')
```



tSNE: Problems

tSNE has some problems.

- Does not scale well with number of data points.
- Does not preserve global structure. Only inter-cluster distances are meaningful. (intra?)
- Works poorly directly on high dimensional data. PCA used to pre-process.
- Requires a lot of memory.
- Cost function is not convex. Random initialization can give different results.

UMAP

UMAP

UMAP: Overview

Uniform manifold approximation and projection (UMAP).

Alternative to tSNE that often performs better on real data.
Similarly described by four equations:

1. Distance in high dimensional data space.
2. k-nearest neighbors
3. Distance in low dimensional embedding.
4. Cost function to minimize.

UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction

Leland McInnes
Tutte Institute for Mathematics and Computing
leland.mcinnes@gmail.com

John Healy
Tutte Institute for Mathematics and Computing
jchealy@gmail.com

James Melville
jlmelville@gmail.com

September 21, 2020

Abstract

UMAP (Uniform Manifold Approximation and Projection) is a novel manifold learning technique for dimension reduction. UMAP is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. The result is a practical scalable algorithm that is applicable to real world data. The UMAP algorithm is competitive with t-SNE for visualization quality, and arguably preserves more of the global structure with superior run time performance. Furthermore, UMAP has no computational restrictions on embedding dimension, making it viable as a general purpose dimension reduction technique for machine learning.

1 Introduction

Dimension reduction plays an important role in data science, being a fundamental technique in both visualisation and as pre-processing for machine

UMAP: (1) High dimensional distances

Probability of observing distances, $d()$, between points in high dimensional, data, space assumed to be exponentially decaying.
The probability is not normalized.

Joint probability is symmetrized, but differently to tSNE.

Conditional probability:

$$p_{j|i} = \exp\left(-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}\right)$$

Joint probability:

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$$

UMAP: (2) k-nearest neighbors

The k nearest neighbors is a value provided by the user as method parameter that influences the solution.

It is then used "in reverse" to solve for a values of σ_i and ρ_i in equation 1.

$$k = 2^{\sum_i p_{ij}}$$

UMAP: (3) Low dimensional distances

Probability of observing distances between points in low dimensional, embedding, space is also similar to the one used in tSNE.

The parameters a and b are found by fitting the function for q_{ij}.

$$q_{ji} = \left(1 + a(y_i - y_j)^{2b}\right)^{-1}$$

$$q = \begin{cases} 1 & \text{if} (y_i - y_j) \leq \mu \\ \exp\left(-(y_i - y_j)\right) - \mu & \text{else} \end{cases}$$

UMAP: (4) Cost function

The cost function is based on the **binary cross entropy (BCE)** between the probability distributions in high dimensional data space, P, and in low dimensional embedding, Q.

The embedding is optimized by minimizing the cost function.

This is done by calculating the gradient of KL with respect to each embedding position.

Minimized by gradient descent.

$$\mathrm{BCE}(P, Q) = \sum_i \sum_j \left(p_{ij} \log \left(\frac{p_{ij}}{q_{ij}} \right) + (1 - p_{ij}) \log \left(\frac{1 - p_{ij}}{1 - q_{ij}} \right) \right)$$

(Same cost function as in logistic regression. Can interpret it as probability of two data points belonging together.)

UMAP: Method compared with tSNE

Both methods are based on four similar steps.

tSNE

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

$$\text{perplexity} = 2^{-\sum_j \left(p_{j|i} \log_2(p_{j|i})\right)}$$

$$q_{ji} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{i \neq j} \left(1 + ||y_i - y_j||^2\right)^{-1}}$$

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

UMAP

$$p_{j|i} = \exp\left(-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}\right)$$

$$k = 2^{\sum_i p_{ij}}$$

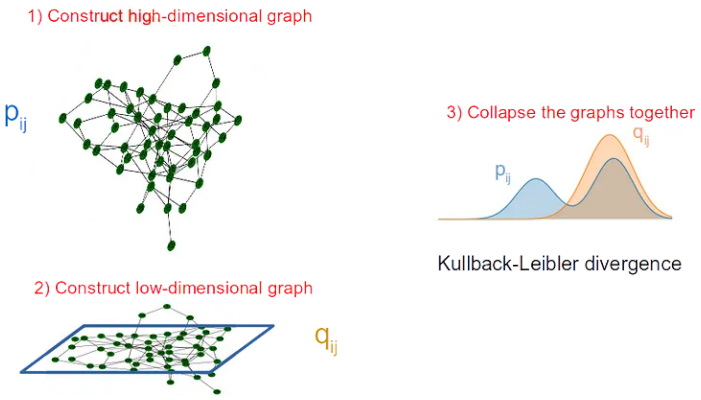
$$q_{ji} = \left(1 + a(y_i - y_j)^{2b}\right)^{-1}$$

$$q = \begin{cases} 1 & \text{if}(y_i - y_j) \leq \mu \\ \exp\left(-(y_i - y_j)\right) - \mu & \text{else} \end{cases}$$

$$\text{BCE}(P, Q) = \sum_i \sum_j \left(p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) + (1 - p_{ij}) \log\left(\frac{1 - p_{ij}}{1 - q_{ij}}\right)\right)$$

UMAP: And tSNE method overview

Both UMAP and tSNE are based on the same principle.

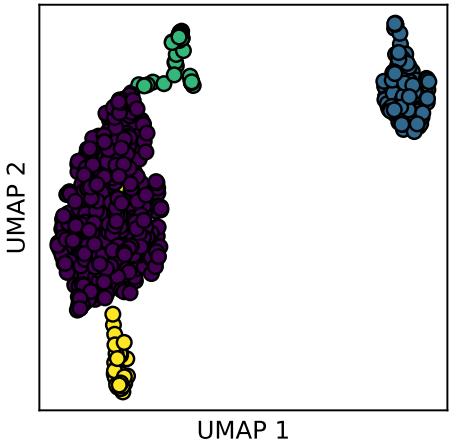


(Source: Nikolay Oskolkov)

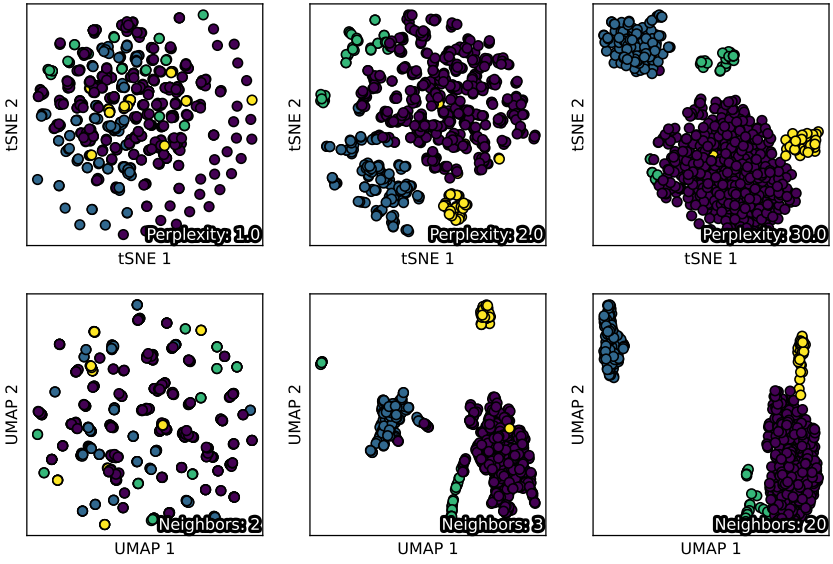
https://nikolay-oskolkov.medium.com/is-umap-accurate-fad1b3f14fb5
https://archive.is/4tOyA

UMAP: Example

```python
import numpy as np
import matplotlib.pyplot as plt
from umap import UMAP

# Cancer associated fibroblast dataset and transform
X, y = load_CAF()
X = np.log(X + 1)

# Instantiate and apply UMAP
umap = UMAP(n_components=2, random_state=0)
X_umap = umap.fit_transform(X)

# Plot result
ax = plt.figure(figsize=(3,3)).gca()
ax.scatter(X_umap[:,0],
           X_umap[:,1],
           marker='o',
           c=y.astype(int),
           edgecolor='k',
           )
ax.set_xlabel('UMAP 1')
ax.set_ylabel('UMAP 2')
```



UMAP: Results compared with tSNE



Conclusion

Conclusion

Conclusion: Distorted or exaggerated relationships in the data

tSNE and UMAP can create or exaggerate clusters in the data.

Occasionally PCA might be a better choice

because it preserves the fact that data points can

have an interpretable position between clusters.

Biologists, stop putting UMAP plots in your papers

UMAP is a powerful tool for exploratory data analysis, but without a clear understanding of how it works, it can easily lead to confusion and misinterpretation.

AUTHOR

Rafael Irizarry

PUBLISHED

Dec. 23, 2024

MATHEMATICAL STATISTICS AND MACHINE LEARNING FOR LIFE SCIENCES

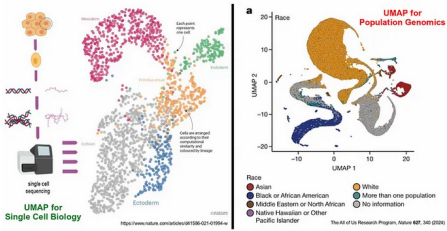
Is UMAP accurate?

Addressing some fair and unfair criticism

Nikolay Oskolkov · Subscribe

20 min read · Feb 22, 2025

238



UMAP in single cell biology vs. UMAP in population genomics. Image by author

https://nikolay-oskolkov.medium.com/is-umap-accurate-fad1b3f14fb5
https://archive.is/4tOyA
https://simplystatistics.org/posts/2024-12-23-biologists-stop-including-umap-plots-in-your-papers/

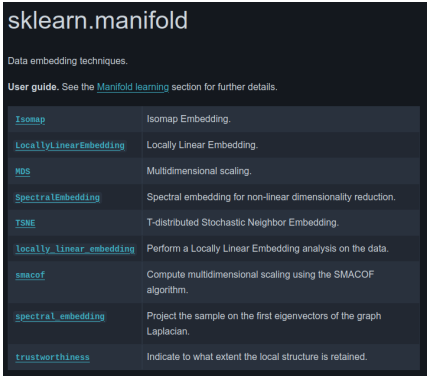
Conclusion: Example

Methods for **non-linear dimensionality reduction** in two families.

Isomap and LLE can be used when PCA seemingly fails.

Single cell data often visualized using tSNE or UMAP.

- MDS, Isomap, Eigenmap, LLE } Spectral methods
- t-SNE, UMAP } Optimization



sklearn.manifold

Data embedding techniques.

User guide. See the Manifold learning section for further details.

Isomap	Isomap Embedding.
LocallyLinearEmbedding	Locally Linear Embedding.
MDS	Multidimensional scaling.
SpectralEmbedding	Spectral embedding for non-linear dimensionality reduction.
TSNE	T-distributed Stochastic Neighbor Embedding.
locally_linear_embedding	Perform a Locally Linear Embedding analysis on the data.
smacof	Compute multidimensional scaling using the SMACOF algorithm.
spectral_embedding	Project the sample on the first eigenvectors of the graph Laplacian.
trustworthiness	Indicate to what extent the local structure is retained.

https://scikit-learn.org/stable/api/sklearn.manifold.html

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