Dimensionality Reduction: Nonlinear techniques

Minería de Datos (M1966)

Steven Van Vaerenbergh
Depto. de Matemáticas, Estadística y Computación
Universidad de Cantabria

Curso 2024-25

Maśter Universitario Oficial **Data Science**

con el apoyo del

Contents

Nonlinear dimensionality reduction

- Some data have **structure** that cannot be revealed by linear projections.
- PCA will fall short.
- Example: How to make a 2D map of MNIST digits?

- We need **nonlinear dimensionality reduction** techniques.
- Concrete benefits include discovering intrinsic structures, and creating a compact visualization that preserves neighborhood relations.

Multidimensional Scaling (MDS)

- Torgerson, 1952
- Nonlinear dimensionality reduction technique originally proposed for visualization in behavioral sciences
- Data may be of any kind (not just scalar), as long as we can define a **dissimilarity measure** on them:
  - E.g. respondents are asked to rate similarities between product pairs.
- MDS seeks a low-dimensional embedding that preserves pairwise dissimilarities as **squared distances**.

Multidimensional Scaling

1. Define a pairwise dissimilarity measure on the data

$$d_{ij}^* = diss(\mathbf{x}_i, \mathbf{x}_j)$$

   We want to map the data $\mathbf{x}_i \in \mathbb{R}^d \longrightarrow \mathbf{y}_i \in \mathbb{R}^r$

2. Denote the Euclidean distance on the mapped data as

$$d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$$

3. MDS searches for the mapping that minimizes a cost[1]

$$C = \sum_{i \neq j} (d_{ij}^* - d_{ij})^2$$

[1] Classical MDS uses this squared-error (“strain”), while “stress” is another popular variant: $C' = \sqrt{\sum_{i\neq j}(d_{ij}^* - d_{ij})^2 / \sum_{i\neq j}(d_{ij}^*)^2}$.

Classical MDS

Classical MDS assumes d^* to be Euclidean distances.

1. Matrix of squared dissimilarities $\boldsymbol{D}_{ij}^{(2)} = (d_{ij}^*)^2$.

2. Apply double centering:

$$\boldsymbol{B} = -\frac{1}{2}\boldsymbol{J}\boldsymbol{D}^{(2)}\boldsymbol{J}$$

J matrice delle medie

where $\boldsymbol{J}$ is the centering matrix $\boldsymbol{J} = \boldsymbol{I} - \mathbf{1}\mathbf{1}^\top/n$, with $\boldsymbol{I}$ the identity matrix and $\mathbf{1}$ the all-ones vector. Note: $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{X}^\top$ is the **Gram matrix** (inner products of centered data points).

3. Extract the r largest **eigenvectors** of $\boldsymbol{B}$ and their corresponding eigenvalues; place them in $\boldsymbol{E}_r$ and $\boldsymbol{\Lambda}_r$.
Eigen-decomposition of **B** gives us the directions of maximum variance in this distance-preserving sense.

4. MDS solution: $\boldsymbol{Y} = \boldsymbol{E}_r\boldsymbol{\Lambda}_r^{1/2}$.

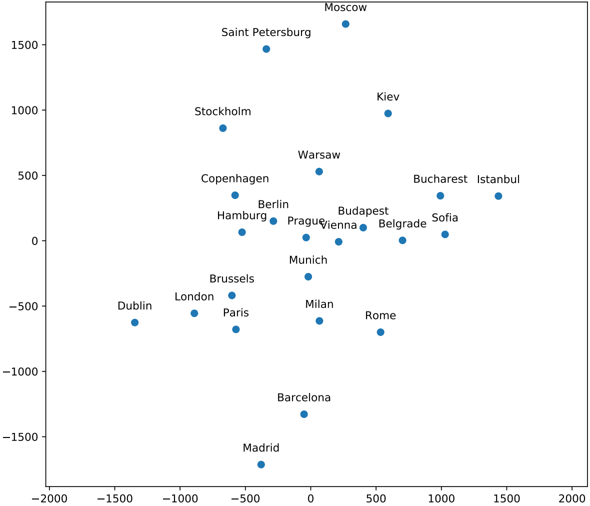
MDS example: European cities

We are given a list of distances between cities. We will use MDS to create a map.

City	Barcelona	Belgrade	Berlin	Brussels	Bucharest	...
Barcelona	0	1528.13	1497.61	1062.89	1968.42	...
Belgrade	1528.13	0	999.25	1372.59	447.34	...
Berlin	1497.61	999.25	0	651.62	1293.4	...
Brussels	1062.89	1372.59	651.62	0	1769.69	...
Bucharest	1968.42	447.34	1293.4	1769.69	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

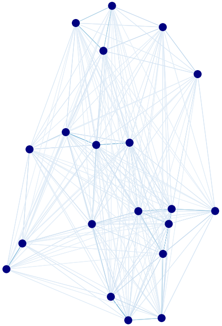
- Python code in `example_3_mds_cities.ipynb`

MDS example: European cities



Physical intuition of MDS

1. Randomly place each object on a 2D map;
2. Connect each pair $(\mathbf{x}_i, \mathbf{x}_j)$ with a spring with the length of the dissimilarity d^*_{ij};
3. Let physics take its course.

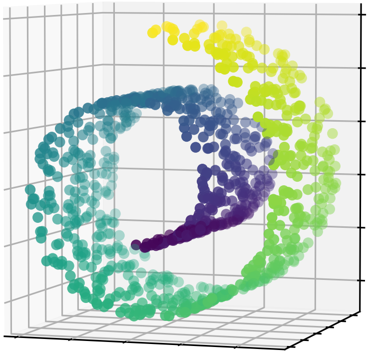


Landmark MDS

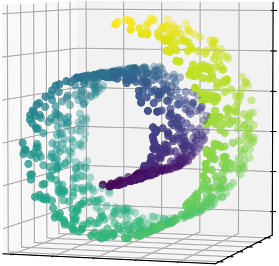
- MDS requires the eigendecomposition of the dissimilarity matrix, which is computationally expensive.
- Speedup: “Landmark MDS”
  1. Choose q data points (“landmarks”), with $r < q \ll m$;
  2. Perform MDS on landmarks;
  3. Map the remaining points to $\mathbb{R}^r$ using only their distances to landmarks.
- Landmark MDS combines MDS with the Nyström algorithm for matrix decomposition.
- By using only q landmarks, we reduce the memory from $\mathcal{O}(m^2)$ to $\mathcal{O}(mq)$). This is especially important for large datasets where a full matrix decomposition is not feasible.

Manifold Modeling

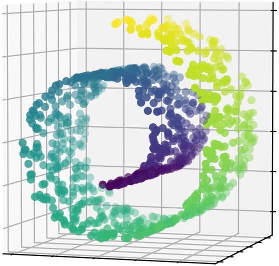
- All previous methods aim to preserve the global distance.
- What if we want to preserve **local** distance?
- Example: “Swiss roll” data:



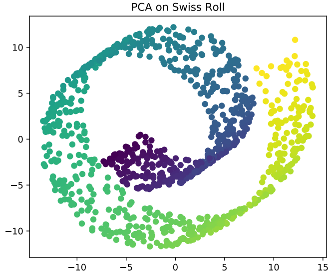
PCA on Swiss Roll data



PCA on Swiss Roll data



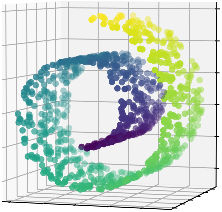
→



Dimensionality Reduction through Manifold Modeling

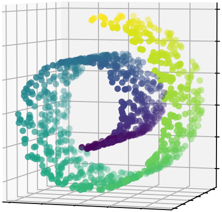
Manifold: a space that locally resembles Euclidean space.
Main idea: Find a low-dimensional representation that preserves **local properties**.

- "**local**" $\rightarrow$ with regard to neighbors.
- "**properties** $\rightarrow$ distances, linear relationships, neighborhood probabilities, etc.



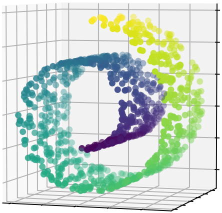
Manifold modeling

- In order to preserve the local distance we need to **model the manifold** on which the data lies.
- Example: “Swiss roll” data:

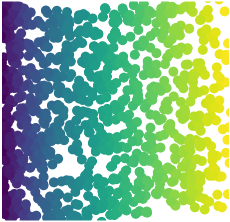


Manifold modeling

- In order to preserve the local distance we need to **model the manifold** on which the data lies.
- Example: “Swiss roll” data:



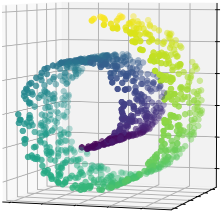
$\rightarrow$



- Local distance on the manifold: **“geodesic distance”**.

Manifold modeling

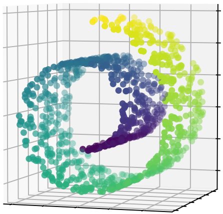
- Swiss roll data: Data on a 2D surface in 3D space.
- PCA and MDS extract a low-dimensional representation of the data but they do not explicitly model the manifold.
- PCA and MDS will fail to discover this 2D structure.



- 2 methods that do model the manifold: Isomap and LLE.

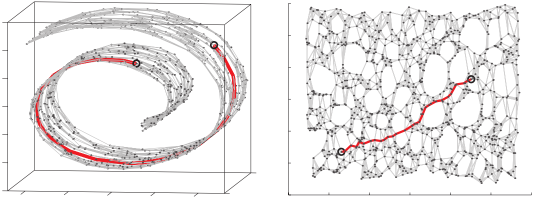
Isometric feature map (Isomap)

- Tenenbaum et al., 2000;
- Idea: Instead of true distance between two points use the **distance along the manifold**;
- This distance can be very large even if the points are close in $\mathbb{R}^d$;



Isomap: Steps

1. Construct a **graph** whose nodes are the data points, where a pair of nodes are adjacent only if the two points are close in $\mathbb{R}^d$.
2. Approximate the **geodesic distance** along the manifold between any two points as the **shortest path** in the graph;
3. Finally use **MDS** to extract the low-dimensional representation from the resulting matrix of squared distances.



Dijkstra's algorithm for shortest path calculation

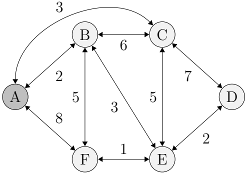
Algorithm 1 Dijkstra's Shortest Path Algorithm.

Input: nodes $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and the distances $d_{i,j}$ between pairs.
for all i **do**
  Initialize effective dist. $\bar{d}_{i,i} = 0$ and $\bar{d}_{i,j} = \infty$, for all $j \neq i$.
  Set node i as current node c.
  Mark al nodes as "unvisited".
  repeat
    for all unvisited nodes n neighboring current node **do**
      if $\bar{d}_{i,n} > \bar{d}_{i,c} + d_{c,n}$ **then**
        Update effective distance: $\bar{d}_{i,n} = \bar{d}_{i,c} + d_{c,n}$.
      end if
    end for
    Mark current node as visited.
    Next current = closest unvisited.
  until all nodes are visited.
end for
Output: effective distances $\bar{d}_{i,j}$.

Dijkstra's algorithm for shortest path calculation

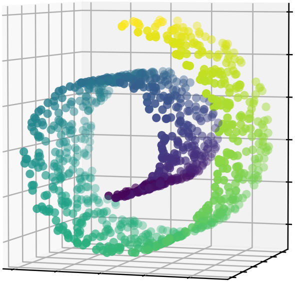
Algorithm 2 Dijkstra's Shortest Path Algorithm.

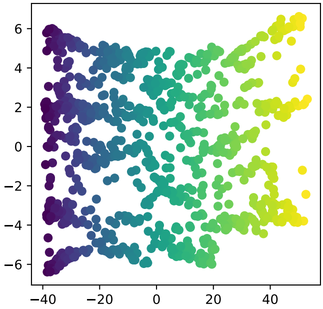
Input: nodes $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and the distances $d_{i,j}$ between pairs.
for all i **do**
  Initialize effective dist. $\bar{d}_{i,i} = 0$ and $\bar{d}_{i,j} = \infty$, for all $j \neq i$.
  Set node i as current node c.
  Mark al nodes as "unvisited".
  repeat
    for all unvisited nodes n neighboring current node **do**
      if $\bar{d}_{i,n} > \bar{d}_{i,c} + d_{c,n}$ **then**
        Update effective distance: $\bar{d}_{i,n} = \bar{d}_{i,c} + d_{c,n}$.
      end if
    end for
    Mark current node as visited.
    Next current = closest unvisited.
  until all nodes are visited.
end for
Output: effective distances $\bar{d}_{i,j}$.



Example: Isomap on Swiss roll data

Data in $\mathbb{R}^d$ $\longrightarrow$ Data in $\mathbb{R}^2$





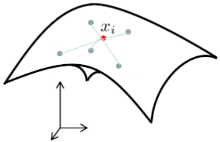
- Python code in `example_4_isomap_swiss_roll.ipynb`

Additional properties of Isomap

- Isomap includes an eigendecomposition, which is **computationally expensive**, $\mathcal{O}(N)^3$.
- Speedup: "Landmark Isomap", based on Landmark MDS and calculates the geodesic distances only to the landmarks.
- Isomap does not provide a direct functional form for the mapping $\mathcal{I} : \mathbb{R}^d \rightarrow \mathbb{R}^r$ that can simply be applied to new data. This further raises computational complexity.

Locally Linear Embedding (LLE)

- Roweis & Saul, 2004.
- Motivation: on a **local** scale the manifold can be approximated by a **linear** subspace.
- Idea: Model the manifold as a **union of linear patches**.
- Approximate each point $\mathbf{x}_i$ as a linear combination of its **neighbors**: $\mathbf{x}_i \approx \sum_{j \in \mathcal{N}(i)} W_{ij} \mathbf{x}_j$



LLE: steps

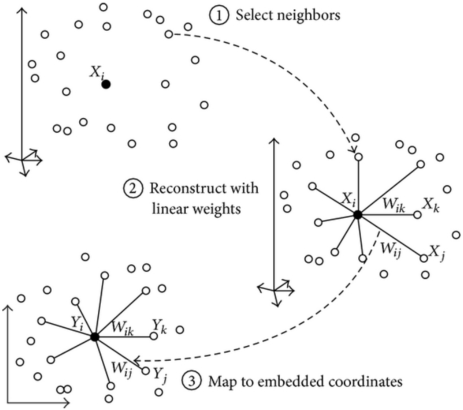
1. Index the nearest neighbors of each $\mathbf{x}_i \in \mathbb{R}^d$ as $\mathcal{N}(i)$.
2. Find the W that minimizes the reconstruction error

$$\sum_i \|\mathbf{x}_i - \sum_{j \in \mathcal{N}(i)} W_{ij}\mathbf{x}_j\|^2$$

3. Find a set of $\mathbf{y}_i \in \mathbb{R}^r$ by minimizing

$$\sum_i \|\mathbf{y}_i - \sum_{j\mathcal{N}(i)} W_{ij}\mathbf{y}_j\|^2$$

LLE steps: graphically



Python exercise

- Exercise 4:
  `exercise_4_swiss_roll_reduction_nonlinear.ipynb`

Extensions: Modified LLE

- Regularization problem in LLE:
  - Local W matrices become **rank-deficient** when the number of **neighbors** is **greater than** the number of input **dimensions**.
  - Therefore, LLE applies a regularization coefficient.
  - But regularization distorts the geometry of the manifold.

Extensions: Modified LLE

- Regularization problem in LLE:
  - Local W matrices become **rank-deficient** when the number of **neighbors** is **greater than** the number of input **dimensions**.
  - Therefore, LLE applies a regularization coefficient.
  - But regularization distorts the geometry of the manifold.
- “Modified” LLE solves this problem by using multiple weight vectors in each neighborhood.
- Several other extensions: Hessian LLE, LTSA, etc.
- Note: LLE provides a functional form for the mapping $\mathcal{I} : \mathbb{R}^d \rightarrow \mathbb{R}^r$

Advanced techniques

- Popular state-of-the-art techniques:
  - t-SNE (2008)
  - UMAP (2018)
- Capable operating on complex data.
- Popular in the data science community.

Stochastic Neighbor Embedding (SNE)

- Hinton & Roweis, 2003.
- Constructs a **probability distribution of the potential neighbors** of all $\mathbf{x}_i$ by placing a Gaussian at each location.
- Similarly, constructs a probability distribution over all $\mathbf{y}_i \in \mathbb{R}^r$.
- SNE uses gradient descent to **minimize the Kullback-Leibler divergence** between both distributions.
- Non-convex optimization problem; SNE uses several heuristics.

t-distributed SNE (t-SNE)

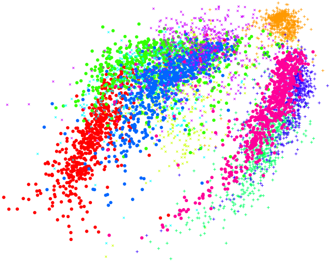
t-SNE:

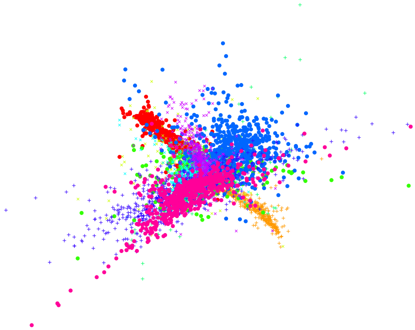
- A recent extension of SNE (van der Maaten & Hinton, 2008).
- Based on a simpler cost function than SNE and uses Student t-distributions rather than Gaussians.
- Better results than SNE and faster (converges earlier).
- Parameter “perplexity”: balance between local and global aspects.
- Very **flexible** algorithm, but **hard to interpret** and finetune.

Python exercise

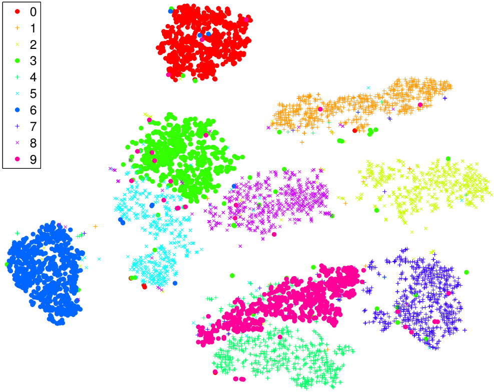
- Exercise 5: `exercise_5_digits_mapping_nonlinear.ipynb`

Example: Visualizations of MNIST


Isomap


LLE

Example: Visualizations of MNIST



t-SNE

Uniform Manifold Approximation and Projection (UMAP)

- McInnes & Healy, 2018.
- Novel manifold learning technique for dimension reduction.
- Theoretical framework based in Riemannian geometry and algebraic topology.
- Results comparable to t-SNE, but faster.
- Python code in `example_5_umap_digits.ipynb`

Interactive examples

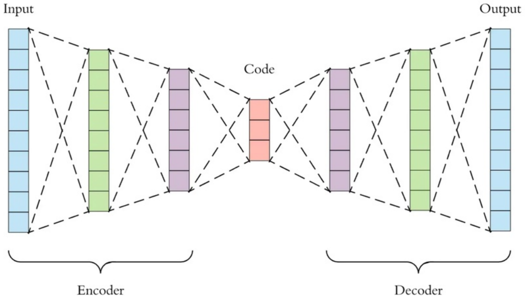
- **Isomap:** `https://plot.ly/~empet/14345.embed`
- **t-SNE:** `https://distill.pub/2016/misread-tsne/`

Interactive examples

- Isomap: `https://plot.ly/~empet/14345.embed`
- t-SNE: `https://distill.pub/2016/misread-tsne/`
- Mapping 6.5M images created by Stable Diffusion `https://atlas.nomic.ai/map/809ef16a-5b2d-4291-b772-a913f4c8ee61/9ed7d171-650b-4526-85bf-3592ee51ea31`

Autoencoder

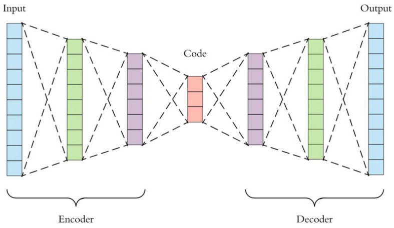
- An autoencoder is an **unsupervised** method (typically a neural network) that **compresses** the data to a lower dimension and then **reconstructs** the input back.
- 2006 seminal paper by Hinton & Salakhutdinov[2], though idea used since the 1980s.



[2] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

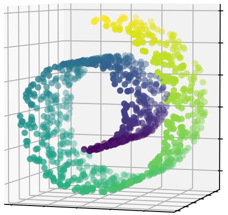
Autoencoder

- Does not make specific assumptions about the data.
- Allows to plug in any regression model as encoder/decoder.
- If a linear model is used as encoder/decoder, the solution obtained coincides with PCA. (See minimum MSE reconstruction.)

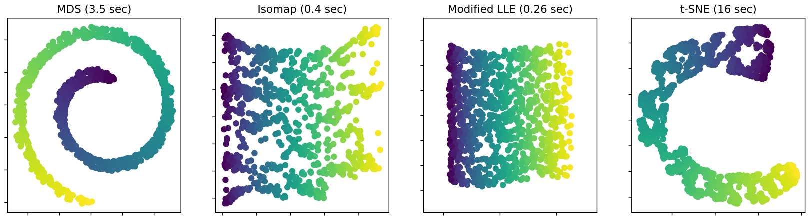


- Python code in `example_6_autoencoder.ipynb`

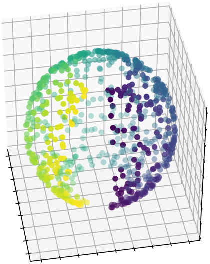
Example: Swiss roll dataset



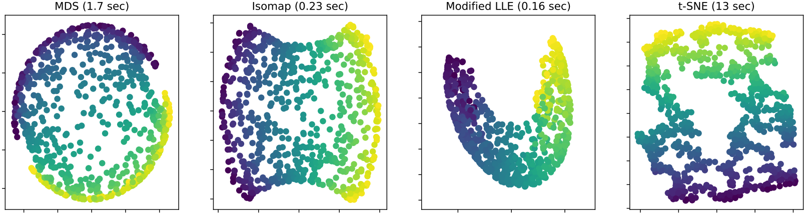
- S-curve dataset.
- Manifold Learning with 1000 points, 10 neighbors.



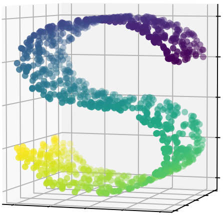
Example: Spherical dataset



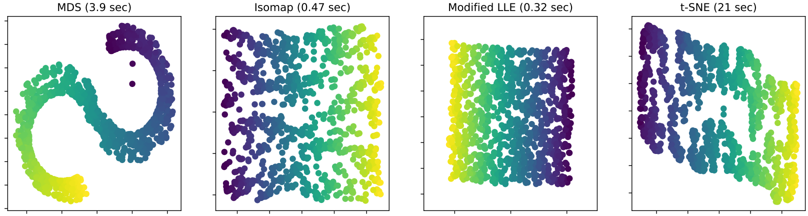
- Spherical data: poles and one slice removed.
- Manifold Learning with 1000 points, 10 neighbors.



Example: S-curve dataset



- S-curve dataset.
- Manifold Learning with 1000 points, 10 neighbors.



Comparison: Goals

What does each technique try to preserve?

- PCA: linear structure
- MDS: global geometry
- Isomap: geodesic distances (globally)
- LLE: local translations, rotations, and scalings
- t-SNE: topology (neighborhood structure)
- Autoencoder: (generic) encoding that allows for reconstruction

Comparison: Computational cost

Execution times on 1000 points of 2D toy data:

- PCA: 0.1 s
- MDS: 3 s
- Isomap: 0.2 s (Landmark Isomap)
- LLE: 0.2 s (Modified LLE)
- t-SNE: 5 s
- Autoencoder: 5 s

Comparison: Restrictions

- PCA: Unable to discover nonlinear structure.
- MDS: Requires selection of a meaningful distance.
- Isomap: Topological instability (affected by noise).
- LLE: Difficulties to treat manifolds with holes (also the case for Isomap); Requires dense manifold.
- t-SNE: Slow; hard to interpret; different convergences.
- Autoencoder: Neural net training requires lots of data.

Comparison: Guidelines

- PCA: To decorrelate data; to discover linear structure.
- MDS: We are given only a distance matrix.
- Isomap: Data is on a well-connected manifold.
- LLE: Manifold is approx. linear on a local scale.
- t-SNE: To visualize complex real-world data.
- Autoencoder: Dimensionality reduction and reconstruction.

Other nonlinear dimensionality reduction techniques

- **Kernel PCA**: nonlinear transformation of data into high-dimensional feature space, then apply PCA in that space.
- Linear Discriminant Analysis (**LDA**): Supervised dimensionality reduction that maximizes class separation.

Conclusions

- Reducing the input data dimensionality is a fundamental preprocessing step for many ML techniques.
- Related to the selection/extraction of features.
- Linear dimensionality reduction techniques:
  - **PCA**: pre-processing for regression/classification techniques; compression/storage of information.
  - **LDA**: pre-processing in problems of supervised classification.

Conclusions

- Nonlinear dimensionality reduction techniques allow to **reveal structure** that linear methods cannot.
- Higher **computational cost**.
- Often used for **visualization**, data exploration.
- Wide range of techniques: **MDS**, **Isomap**, **LLE**, **t-SNE**, **UMAP**, **Autoencoder**, etc.
- Related: Kernel-based dimensionality reduction techniques (KPCA, KCCA, etc.).

References

- Christopher J. C. Burges (2010), “Dimension Reduction: A Guided Tour”, Foundations and Trends in Machine Learning: Vol. 2: No. 4, pp 275-365.