THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 16: What else is out there?

Instructor: Sergei V. Kalinin

And one transform to rule them all…

https://en.wikipedia.org/wiki/One_Ring

Canonical Correlation Analysis

- Simple correlation: $y_1 \sim x_1$
- Multiple correlation: $y_1 \sim x_1 \quad x_2 \quad x_3$
- Canonical correlation: $y_1 \quad y_2 \quad y_3 \sim x_1 \quad x_2 \quad x_3$

- Start with multiple y and x variables: $y_1 \quad y_2 \quad y_3 \sim x_1 \quad x_2 \quad x_3$
- Construct a “canonical variate” as the combination of y variables:

$$CV_{y1} = b_1 y_1 + b_2 y_2 + b_3 y_3$$

- Construct a “canonical variate” as the combination of x variables

$$CV_{x1} = a_1 x_1 + a_2 x_2 + a_3 x_3$$

- The canonical correlation is the correlation of the variates $R_c = r_{cvy1,\ cvx1}$
- Find a’s and b’s to maximize R_c
- Iterate for second pair

Steps of CCA

1. **Start with two datasets X and Y:** Standardize if necessary (subtract the mean and divide by the standard deviation for each variable).
2. Find the **First Pair of Canonical Variables** by maximizing the correlation between linear combinations of **X** and **Y**. Use optimization algorithm (e.g., an eigenvalue or SVD) to find the weight vectors **a** and **b** for **X** and **Y** respectively, such that the correlation between **X a** and **Y b** is maximized.
3. **Orthogonalization:** Find the residuals by removing the projections on the first pair of canonical variables from **X** and **Y** respectively. This can be done by regressing each variable on the first canonical variable and taking the residuals.
4. **Next Pair of Canonical Variables:** Repeat the CCA process with the residual data to find the next pair of canonical variables. Again, find weight vectors **a2** and **b2** for the residuals of X and Y respectively, that maximize the correlation between the new linear combinations.
5. **Continue the Process** iteratively, orthogonalizing the datasets at each step and performing CCA on the residuals, until the desired number of canonical variable pairs is obtained or until the correlations for new pairs are near zero.
6. **Result**: Obtain multiple pairs of canonical variables, each pair being uncorrelated with the others.

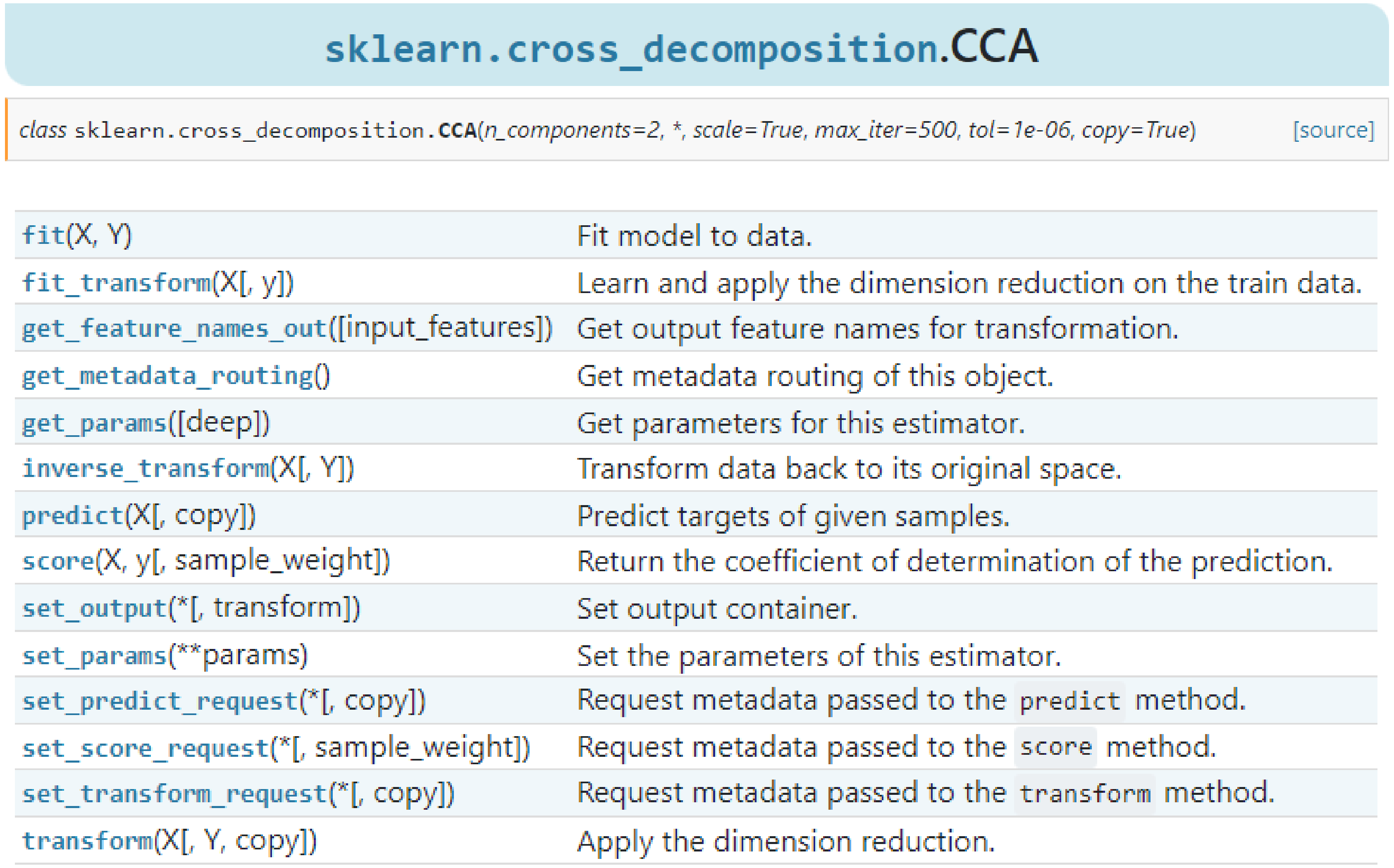
Canonical Correlation Analysis

Canonical correlation is used (instead of comparisons among selected simple and/or multiple correlations) when:

- We are interested in which combination(s) of feature variables are related to which combination(s) of target variables
- Whether the set of variables have a concentrated or a diffuse correlational structure

For example: we have two sets of predictor variables and want to show that one set is most useful for understanding the one criterion variable and the other set is most useful for understanding a different set of criterion variables

CCA in scikit-learn



sklearn.cross_decomposition.CCA

class sklearn.cross_decomposition.**CCA**(*n_components=2*, ***, *scale=True*, *max_iter=500*, *tol=1e-06*, *copy=True*) [source]

`fit`(X, Y)	Fit model to data.
`fit_transform`(X[, y])	Learn and apply the dimension reduction on the train data.
`get_feature_names_out`([input_features])	Get output feature names for transformation.
`get_metadata_routing`()	Get metadata routing of this object.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(X[, Y])	Transform data back to its original space.
`predict`(X[, copy])	Predict targets of given samples.
`score`(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
`set_output`(*[, transform])	Set output container.
`set_params`(**params)	Set the parameters of this estimator.
`set_predict_request`(*[, copy])	Request metadata passed to the `predict` method.
`set_score_request`(*[, sample_weight])	Request metadata passed to the `score` method.
`set_transform_request`(*[, copy])	Request metadata passed to the `transform` method.
`transform`(X[, Y, copy])	Apply the dimension reduction.

CCA Outputs

- **x_weights_:** Contains the canonical weights for the first dataset (X). Used to compute the canonical variables for X.
- **y_weights_:** Contains the canonical weights for the second dataset (Y). Used to compute the canonical variables for Y.
- **x_scores_:** Contains the scores of X, i.e., the canonical variables derived from X. Calculated by multiplying X with x_weights_.
- **y_scores_:** Contains the scores of Y, i.e., the canonical variables derived from Y. Calculated by multiplying Y with y_weights_.
- **x_loadings_:** Contains the canonical loadings for X. Represents the correlation between X and the canonical variables of X.
- **y_loadings_:** Contains the canonical loadings for Y. Represents the correlation between Y and the canonical variables of Y.
- **coef_:** Contains the coefficients of the linear model used to transform X to Y. Represents the relationship between the canonical variables of X and Y.

Why these methods are useful?

Pro:

1. Computational Efficiency
2. Interpretability
3. Less Data Requirement
4. Applicability to Linear Problems
5. Ease of Implementation and Use
6. Feature Reduction and Visualization
7. Foundation for More Complex Models

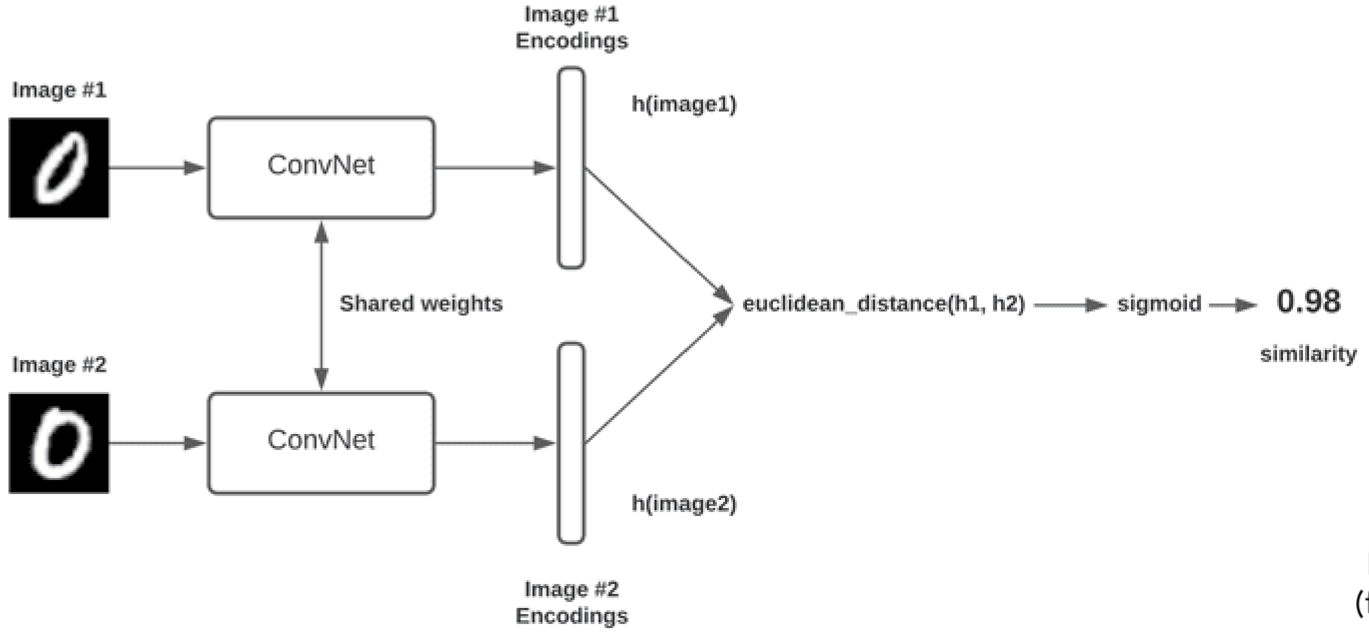
Con:

1. Assumption of Linearity
2. Assumption of Gaussian distributions
3. Limited Complexity
4. Independence Assumption in PCA and CCA
5. Vulnerability to Outliers
6. Feature Importance Ambiguity

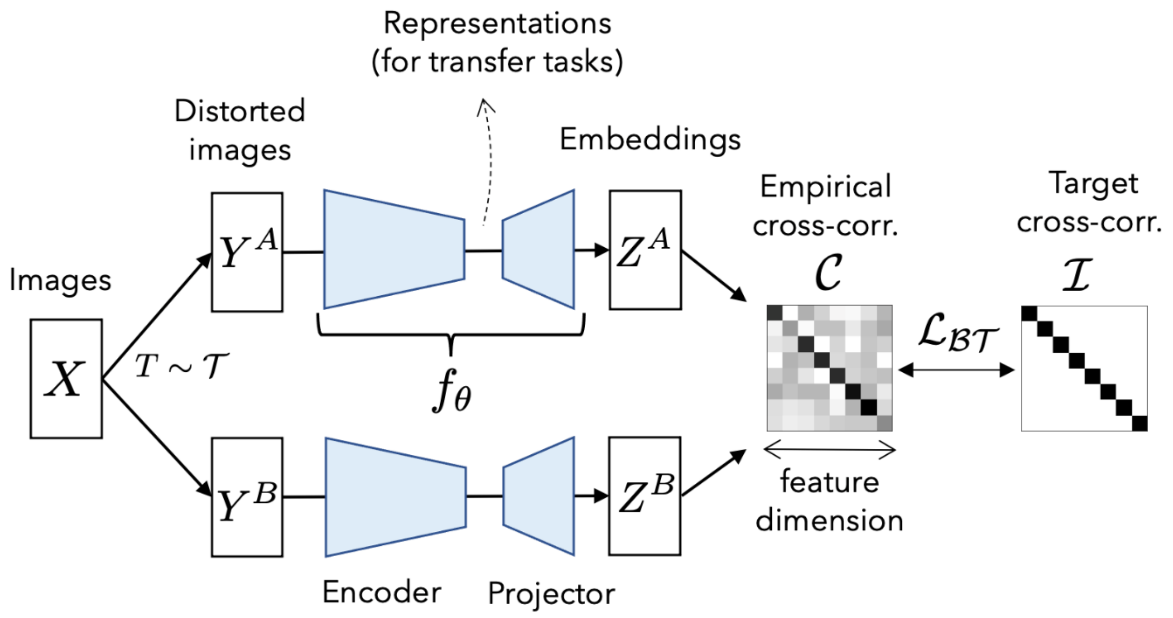
Linear methods in Deep Learning Era

1. **Preprocessing:** To reduce the dimensionality of data or extract more informative features before feeding them into a neural network.
2. **Visualization:** To visualize high-dimensional embeddings learned by neural networks.
3. **Post-Processing of Embeddings:** To further process or analyze embeddings learned by a neural network for various tasks. Specifically, **CCA** can be used to measure the similarity of learned representations from different neural networks or layers, often used in analyzing and comparing embeddings.
4. **Improving Model Robustness and Generalization:** To enhance the robustness and generalization capabilities of deep learning models: feature extraction or transformation, potentially enhancing the robustness and generalization of neural network models.
5. **Facilitating Transfer Learning:** To adapt learned representations for transfer learning applications, e.g. transform embeddings for alignment, compatibility, or adaptation
6. **Interpreting Neural Network Decisions:** To interpret and understand the decisions or representations learned by neural networks, e.g. analyze and interpret the feature representations or activations within a neural network, contributing to model interpretability and transparency.

Siamese networks



Barlow twins



https://pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

https://github.com/facebookresearch/barlowtwins

Independent Component Analysis

- **PCA:** orthogonal transformation of possibly correlated variable into a set of linearly uncorrelated variables
    - Analyzes data representing observations described by dependent variables which are inter-correlated
    - Main goal is to find true variables assuming that corrupting noise is Gaussian

- **ICA:** method for separating multivariate signal into additive subcomponents assuming statistical independence of source signals
    - Finds components that are maximally independent and non-Gaussian
    - Blind source separation – cocktail party problem

Compared to PCA, ICA can produce statistically independent non-Gaussian components by decorrelating the higher-order moments in addition to the first- and second-order moments of the statistical distribution

Mathematics of ICA

The Goal is to transform observed data into maximally independent components measured through some function $F(s_1,\ldots,s_n)$ of independence.

- Data as a set of vectors $\longrightarrow$ $x = (x_1,\ldots,x_m)^T$
- Components of the data $\longrightarrow$ $s = (s_1,\ldots,s_n)^T.$

$$s = Wx$$

An observed data vector *x* can then be represented as a sum of independent components *s* weighted by some mixing weight *a*:

$$x_i = a_{i,1}s_1 + \ldots + a_{i,k}s_k + \ldots + a_{i,n}s_n \quad \text{or} \quad x = \sum_{k=1}^{n} s_k a_k$$

In other words, data vector *x* is represented by basis vectors $a_k = (a_{1,k},\ldots,a_{m,k})^T$

that can form columns of a mixing matrix such that: $x = As$

https://en.wikipedia.org/wiki/Independent_component_analysis

Mathematics of ICA - 2

$$x = As$$

Given that our data is set of vectors *x*, we want to find both, the mixing matrix *A* and sources *s*

This can be done by calculating *w* vectors and a cost function that can maximize the Non-Gaussianity, or minimize mutual information of

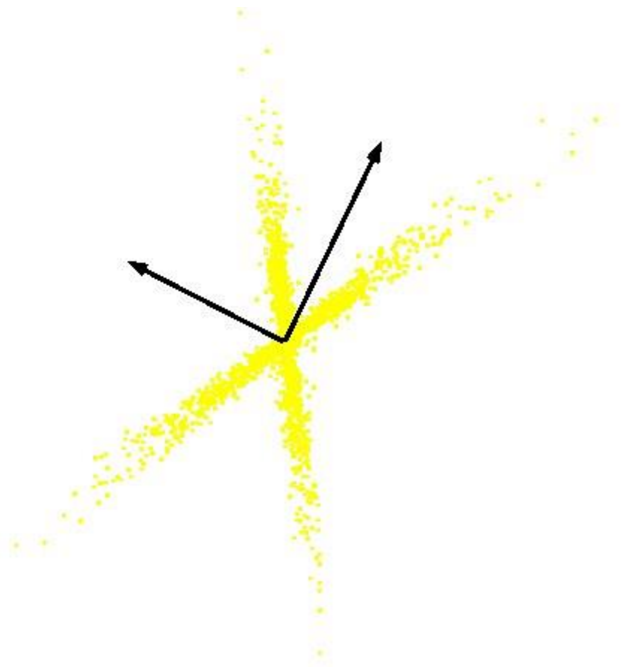
$$s_k = (w^T * x)$$

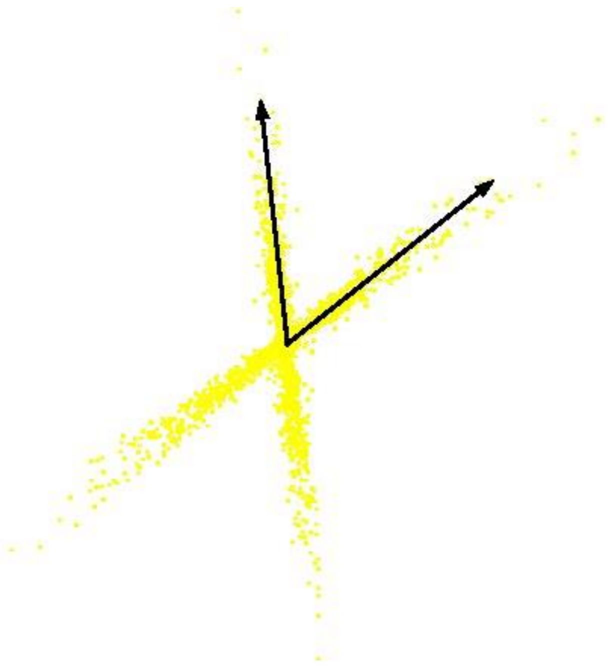
Original sources can then be recovered by multiplying observed data vectors *x* with the inverse of the mixing matrix: $W = A^{-1}$

- Not as easy to utilize as PCA, but excellent premade algorithms are readily available, e.g. Aapo Hyvärinen – FastICA*

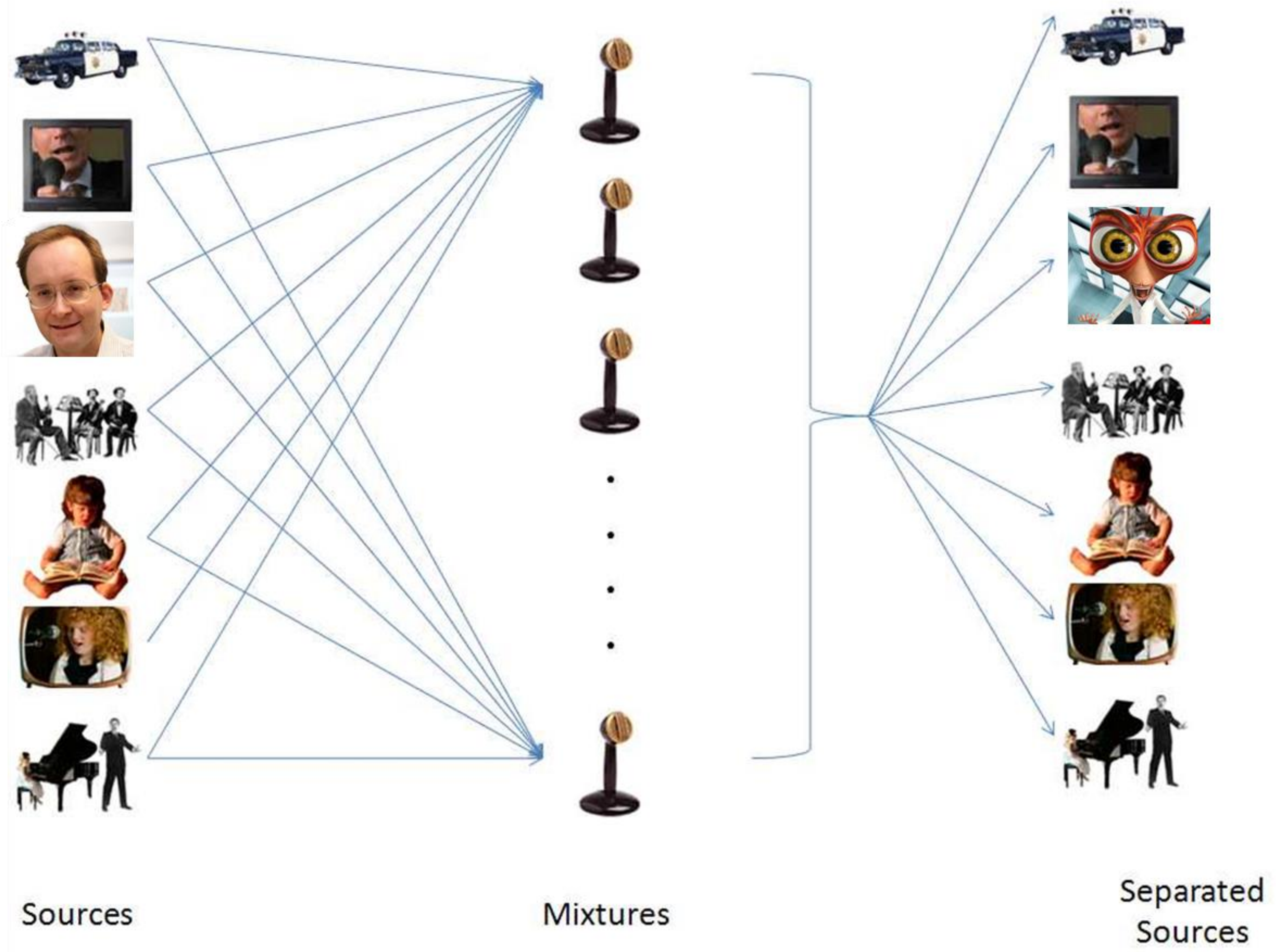
http://cis.legacy.ics.tkk.fi/aapo/papers/IJCNN99_tutorialweb/IJCNN99_tutorial3.html
http://research.ics.aalto.fi/ica/fastica/

PCA vs. ICA


PCA
(orthogonal coordinate)


ICA
(non-orthogonal coordinate)

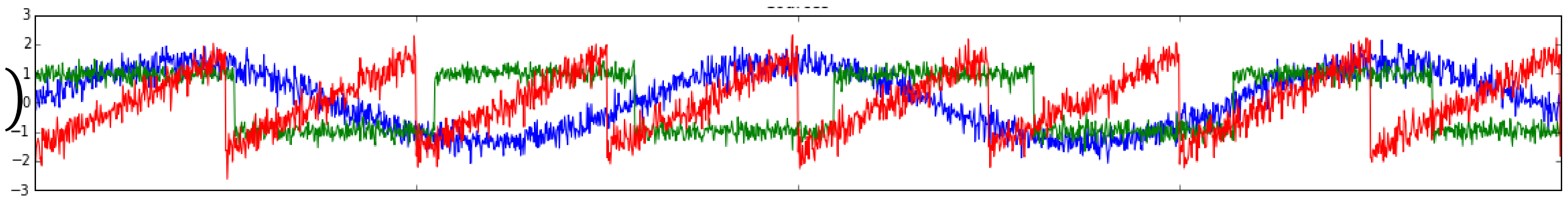
Cocktail Party Problem



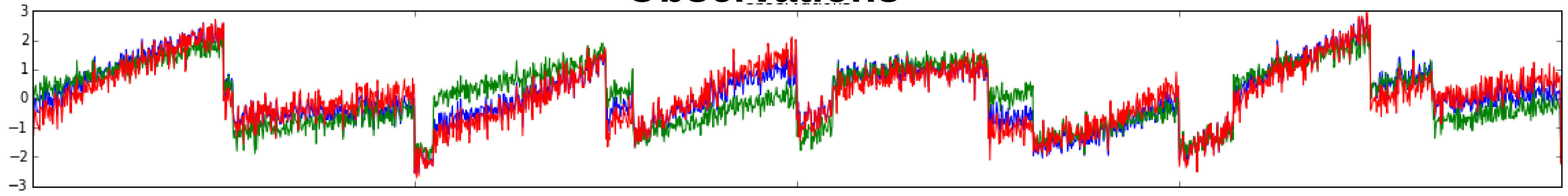
ICA Example

$$\left(R_1(t), R_2(t), \dots R_n(t)\right)$$

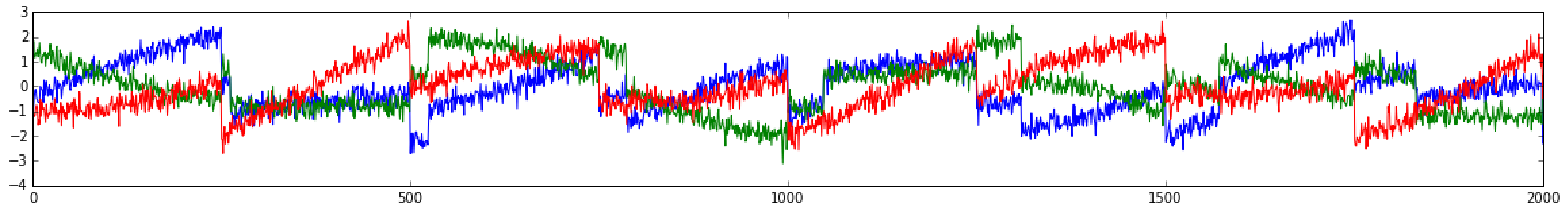
Sources



Observations

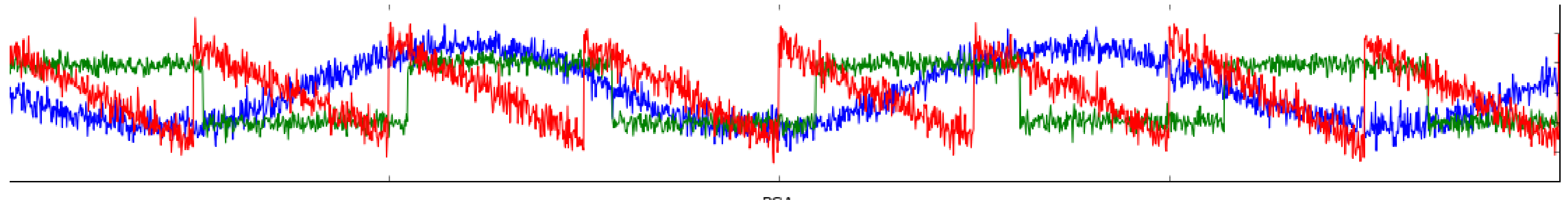


PCA Components



Components are maximally independent

ICA Components

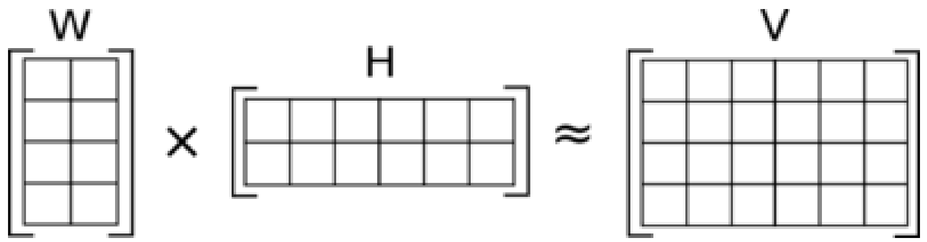


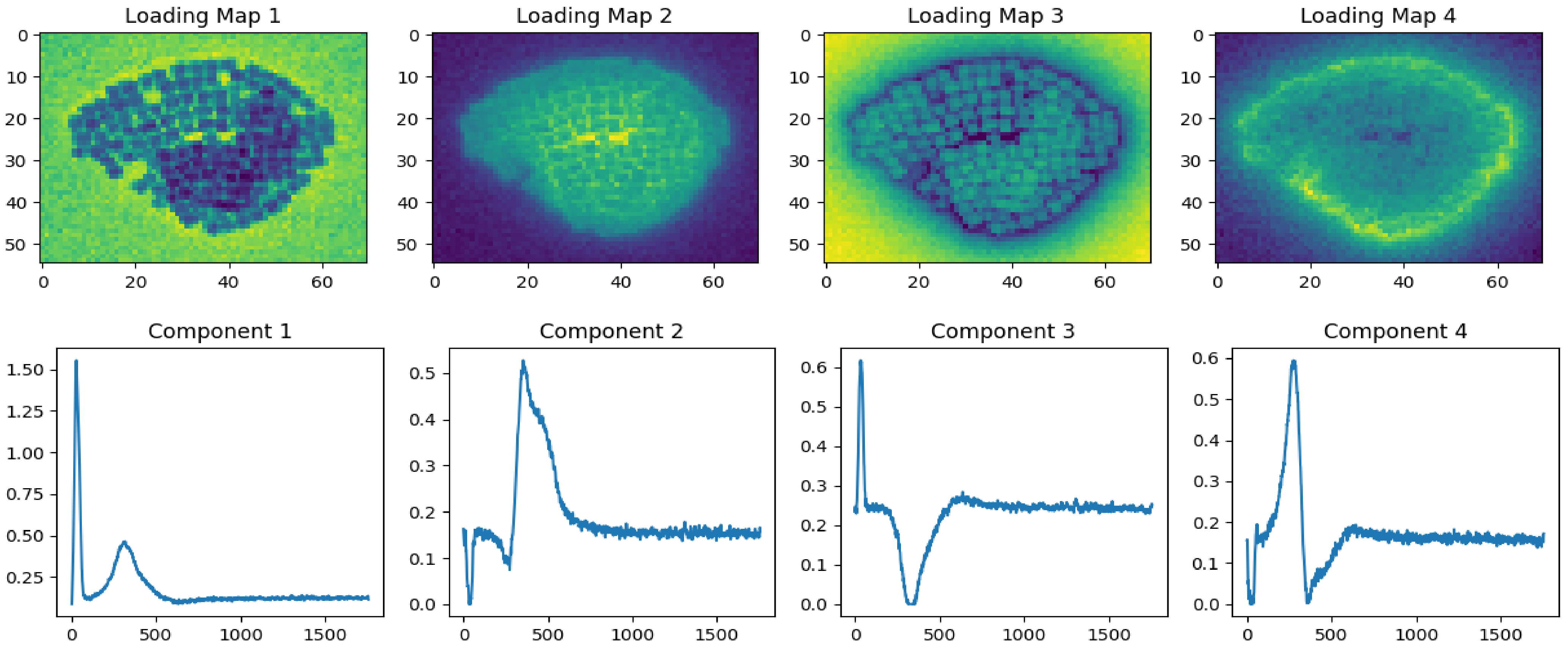
$$\begin{pmatrix} R_1(t) \\ \dots \\ R_n(t) \end{pmatrix} = \boldsymbol{A} \begin{pmatrix} s_1(t) \\ \dots \\ s_n(t) \end{pmatrix}$$

Non-Negative Matrix Factorization

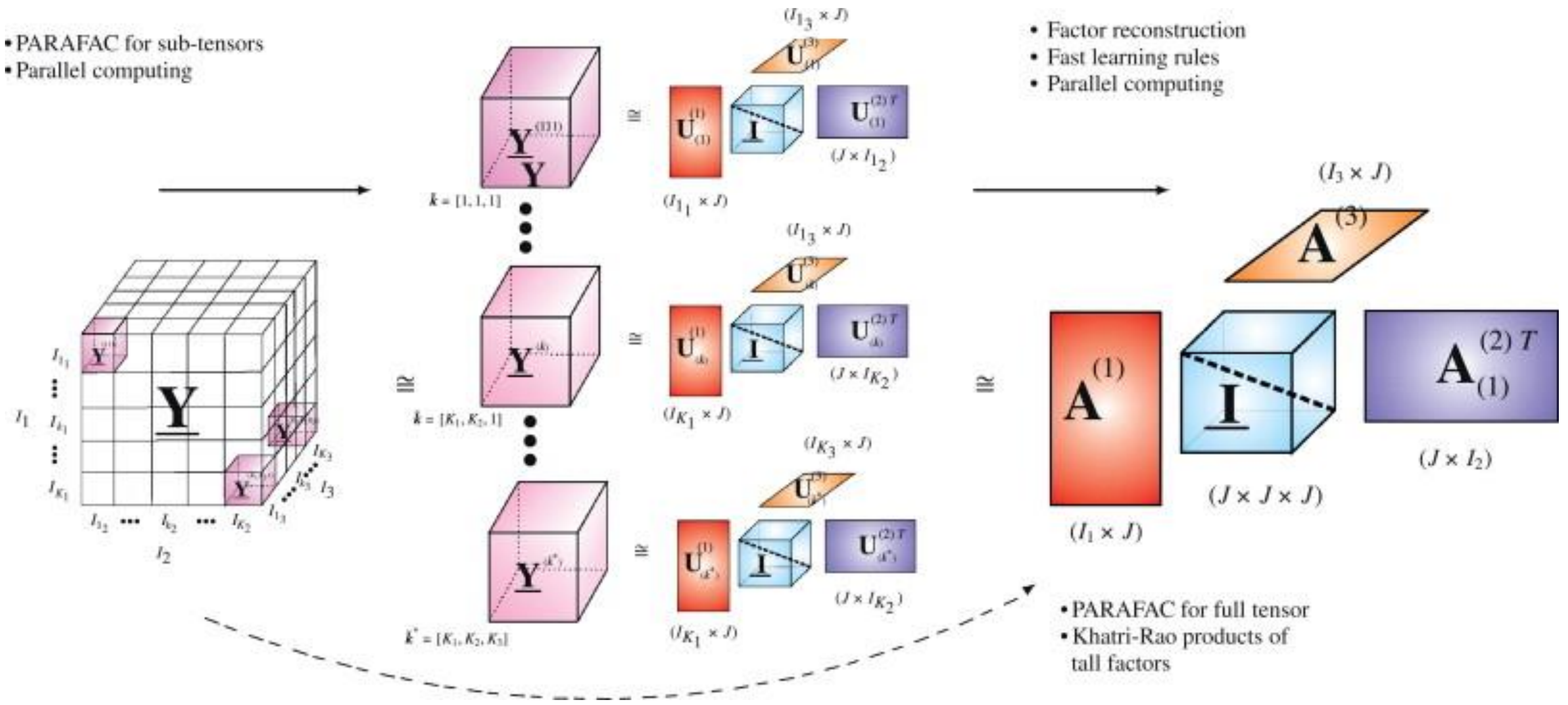
Non-negative Matrix Factorization (NMF)

- Use Case: Factorizes data into non-negative components, useful in fields like topic modeling and image analysis.
- Pros: Produces interpretable, part-based components.
- Cons: Limited by the non-negativity constraint.





Parallel Factor Analysis (PARAFAC)



https://www.sciencedirect.com/science/article/abs/pii/S0925231211000415

Factor Analysis

The factor analysis model expresses each observed variable X_i as a linear combination of the common factors F_j and an error term:

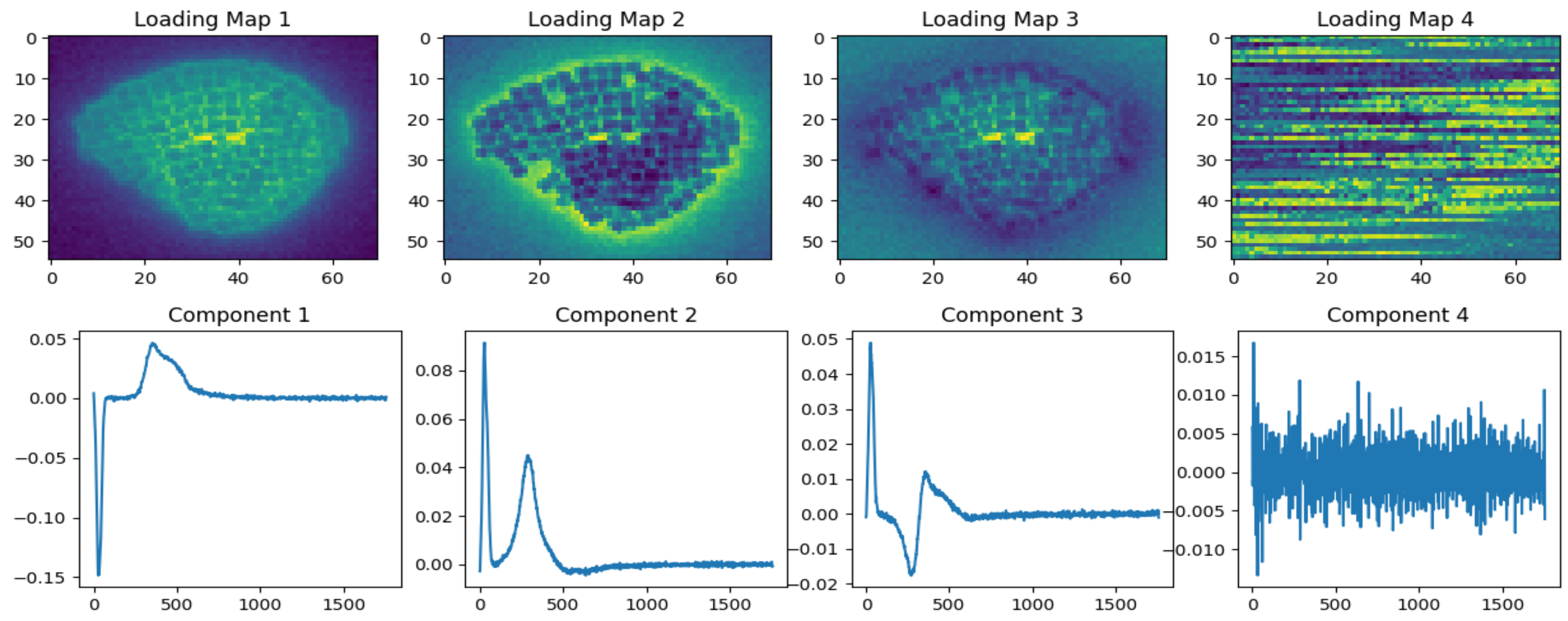
$$X_i = \lambda_{i1} F_1 + \lambda_{i2} F_2 + \cdots + \lambda_{im} F_m + \epsilon_i$$

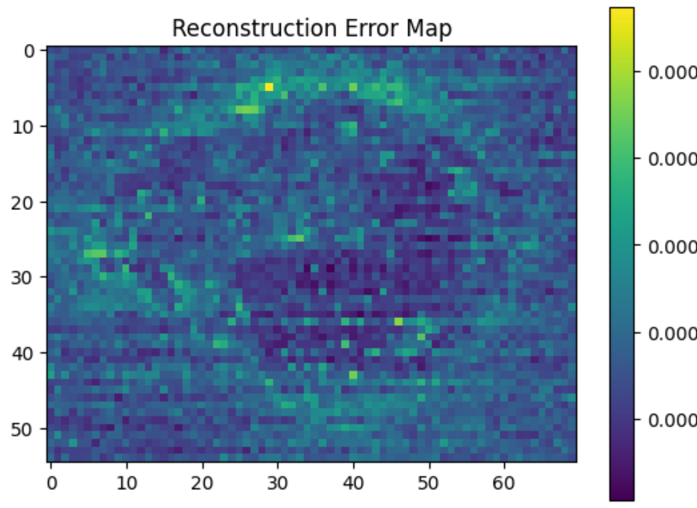
Where:

- X_i is the observed variable.
- F_j are the latent factors.
- λ_{ij} are the factor loadings (the weight of each factor on the observed variable)
- ϵ_i is the unique variance or error

Feature	**Principal Component Analysis (PCA)**	**Factor Analysis (FA)**
Purpose	Dimensionality reduction by transforming data into a new set of uncorrelated variables (principal components) that capture the most variance in the data.	FA is used to identify latent factors that explain the correlations among observed variables, primarily for uncovering underlying structures.
Modeling Goal	PCA aims to capture the maximum variance in the observed data with fewer principal components (linear combinations of the original variables).	FA aims to explain the correlations or covariances between observed variables by modeling them as linear combinations of latent factors plus error.
Variance vs. Covariance	PCA focuses on explaining **total variance** in the data.	FA focuses on explaining **shared variance** (covariation) among variables.
Dimensionality Reduction	PCA transforms the data into a lower-dimensional space while preserving the most variance.	FA reduces dimensionality by identifying latent factors that explain the shared variance in the observed variables.
Rotation of Components	Rotation is not typically applied in standard PCA, but Varimax or other rotations can be applied in some cases to make components more interpretable.	Rotation (e.g., Varimax, Promax) is often used to make the factors more interpretable by simplifying the factor loadings.

Factor Analysis





- **Communalities:** Shared variance explained by the factors.
- **Specific Variance (Uniqueness):** Variance unique to each observed variable.
- **Factor Scores:** Individual-level scores for each latent factor.
- **Eigenvalues:** Amount of variance explained by each factor.
- **Variance Explained by Each Factor:** Total variance accounted for by each factor.
- **Factor Correlation Matrix:** Correlations between factors (if oblique rotation is used).
- **Rotated Factor Loadings:** Simplified loadings for better interpretation after rotation.
- **Scree Plot:** Graphical tool to help determine the number of factors to retain.
- **Rotation Type:** Method used to rotate the factors for better interpretability.

t-SNE and UMAP

1. **t-SNE (t-Distributed Stochastic Neighbor Embedding)**

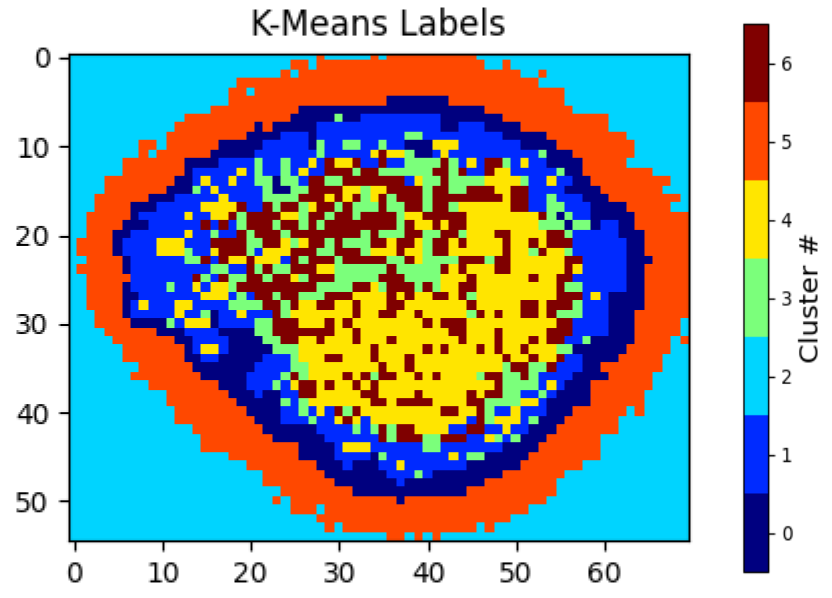
- Use Case: Effective for visualizing high-dimensional data by reducing it to 2 or 3 dimensions, especially when you want to retain local structure.
- Pros: Great for visualizing clusters
- Cons: Does not preserve global structure well

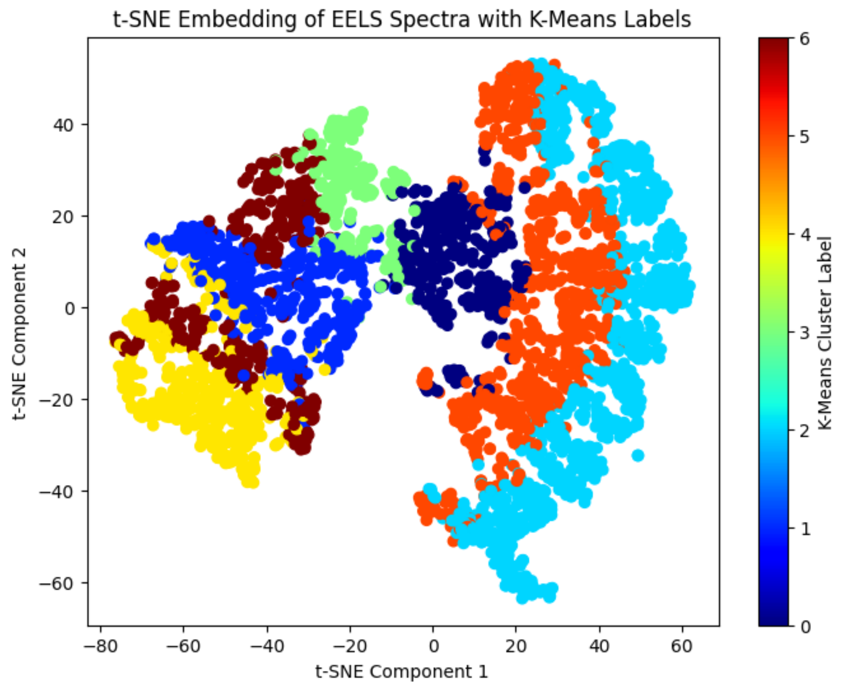
Output

- Low-Dimensional Embeddings
- Pairwise Affinities (in high- and low-dimensional space)

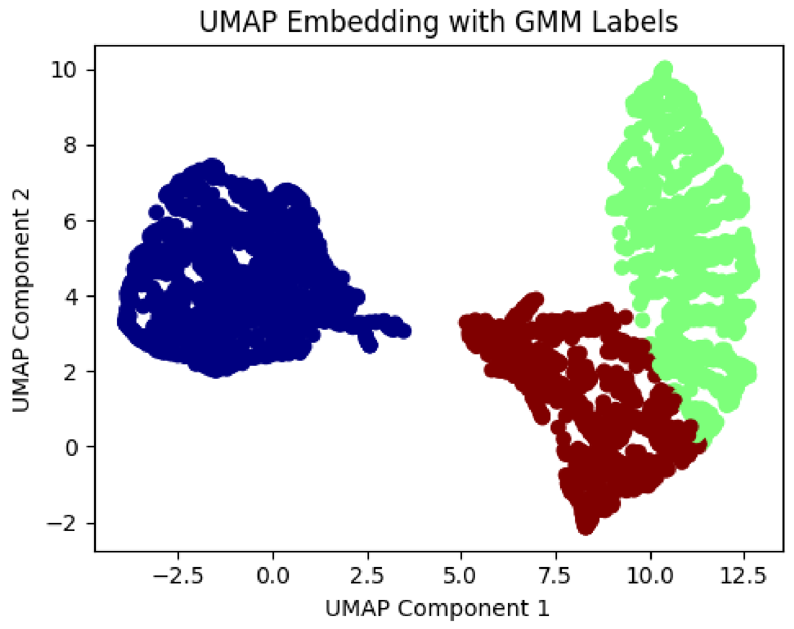
2. **UMAP (Uniform Manifold Approximation and Projection)**

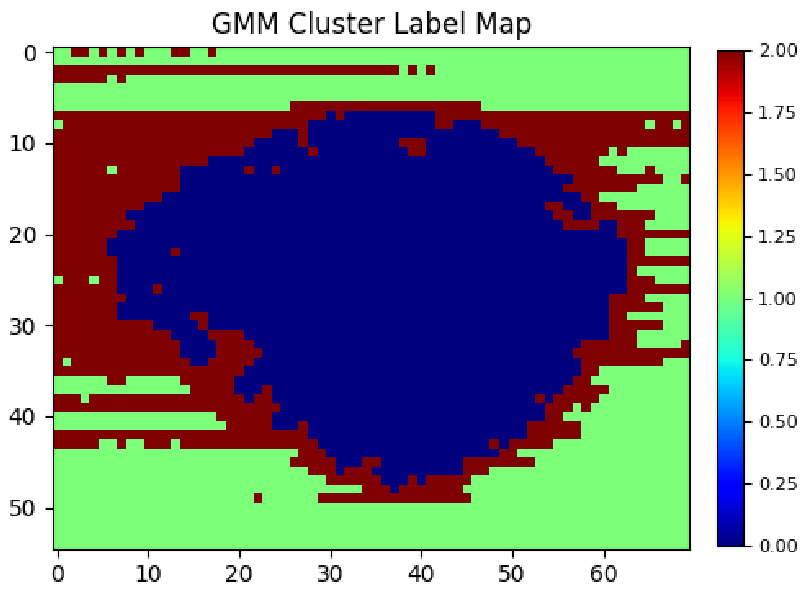
- Use Case: Similar to t-SNE but faster and better at preserving both local and global structures.
- Pros: Computationally efficient and preserves more topological structure than t-SNE
- Cons: Can be harder to tune compared to t-SNE

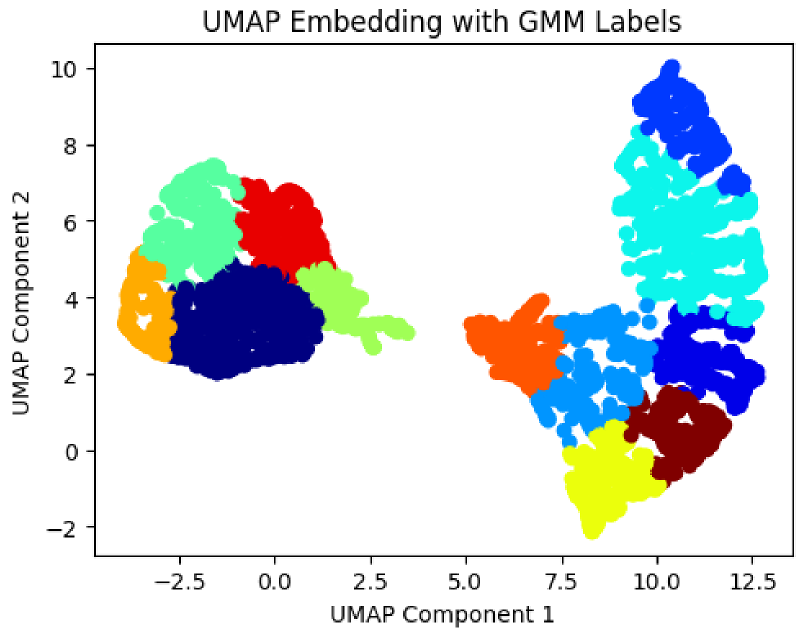


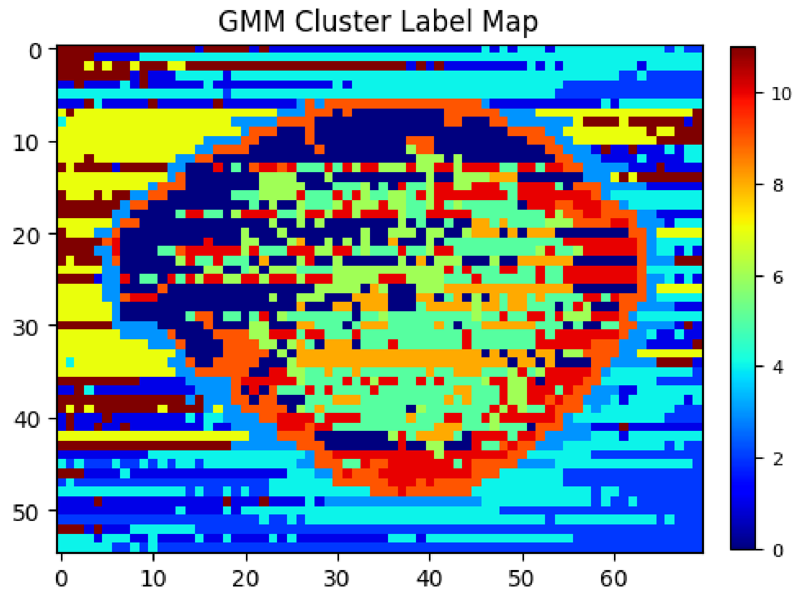


Clustering in UMAP space







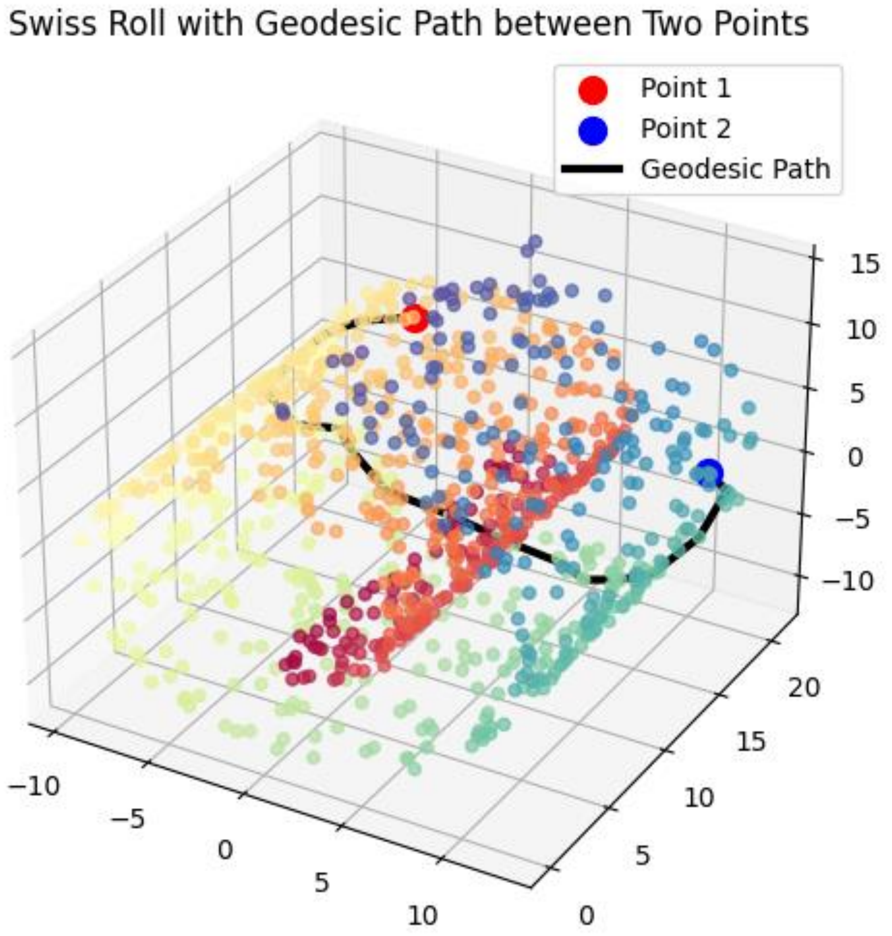


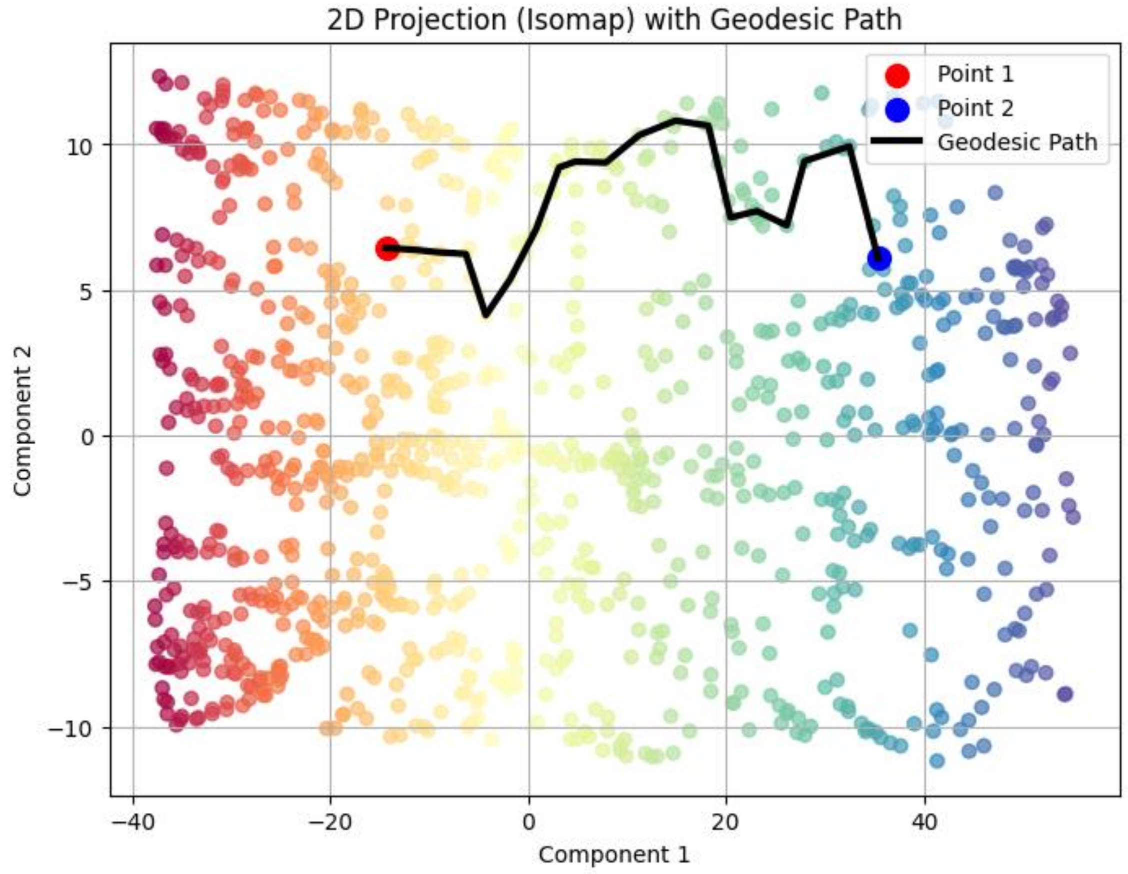
Isomap and MDS

- Isomap: Manifold learning dimensionality reduction preserving geodesic distances between points.
- MDS (Multidimensional Scaling): Aims to place objects in low-dimensional space based on pairwise dissimilarities.

Feature	**Isomap**	**Multidimensional Scaling (MDS)**
Type of Distance	Uses **geodesic distances** (distances along a manifold)	Uses **Euclidean distances** (straight-line distances)
Goal	Captures the underlying **nonlinear structure** of data	Preserves pairwise **Euclidean distances** as faithfully as possible
Nonlinearity	Specifically designed to handle **nonlinear** manifolds	Assumes data lies in a **linear** space
Neighborhood Graph	Constructs a neighborhood graph (k-nearest neighbors or ε-ball) to approximate geodesic distances	No neighborhood graph; relies directly on the pairwise distances in the original space
Distance Calculation	Geodesic distance is computed using shortest paths on a graph	Directly uses the input distance matrix (e.g., Euclidean distance)
Preservation of Structure	Better for preserving **global nonlinear** structures	Better for preserving **local linear** relationships
Algorithm	Extends classical MDS by computing shortest paths between points in a graph	Classical MDS uses eigenvalue decomposition to map distances into a lower-dimensional space
Handling of Nonlinearities	Handles **nonlinear** dimensionality reduction well	Struggles with **nonlinear** structures, better for **linear** manifolds
Applications	Useful for reducing dimensionality of data with curved, nonlinear manifolds (e.g., Swiss roll)	Good for preserving the relative distances of high-dimensional data points in lower dimensions

Isomap and MDS





Self-Organized Feature Maps

An unsupervised learning technique that projects high-dimensional data onto a lower-dimensional (usually 2D) map while preserving the topological structure. Competitive learning.

- Initialization: A grid of nodes is initialized, each with a weight vector matching the input data's dimensions.

- Input Sampling: Input vectors are presented to the grid iteratively.
- BMU Identification: The node with the closest weight vector to the input (Best Matching Unit, BMU) is found.
- Update: The BMU and its neighbors' weights are updated to resemble the input vector, with closer neighbors adjusting more.

- Decay: The learning rate and neighborhood radius decrease over time for fine-tuning.
- Iteration: This process repeats until convergence.

- SOFM is used for tasks like clustering and visualizing high-dimensional data, preserving the topological relationships between data points.

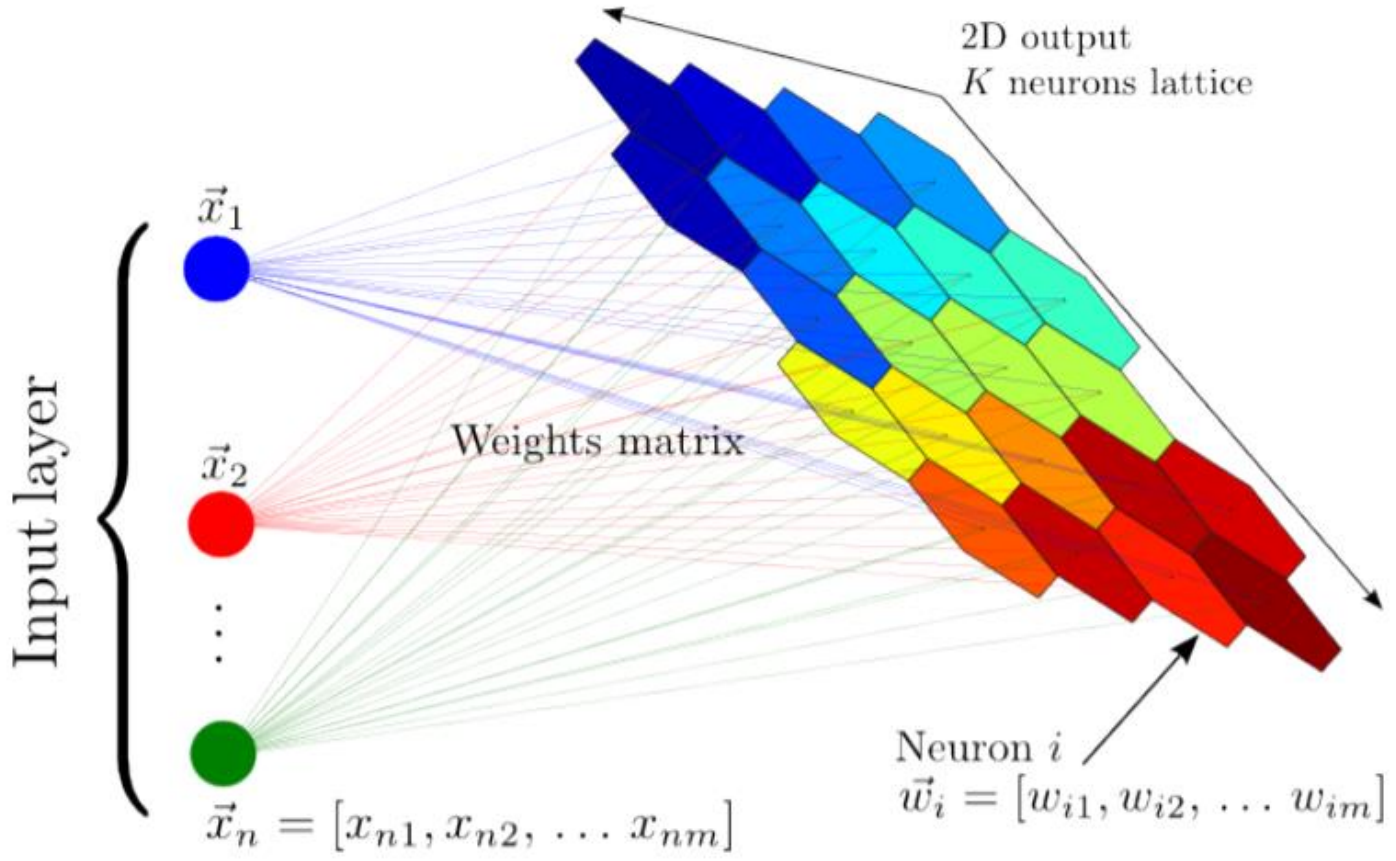
Self-Organized Feature Maps

Learning law:

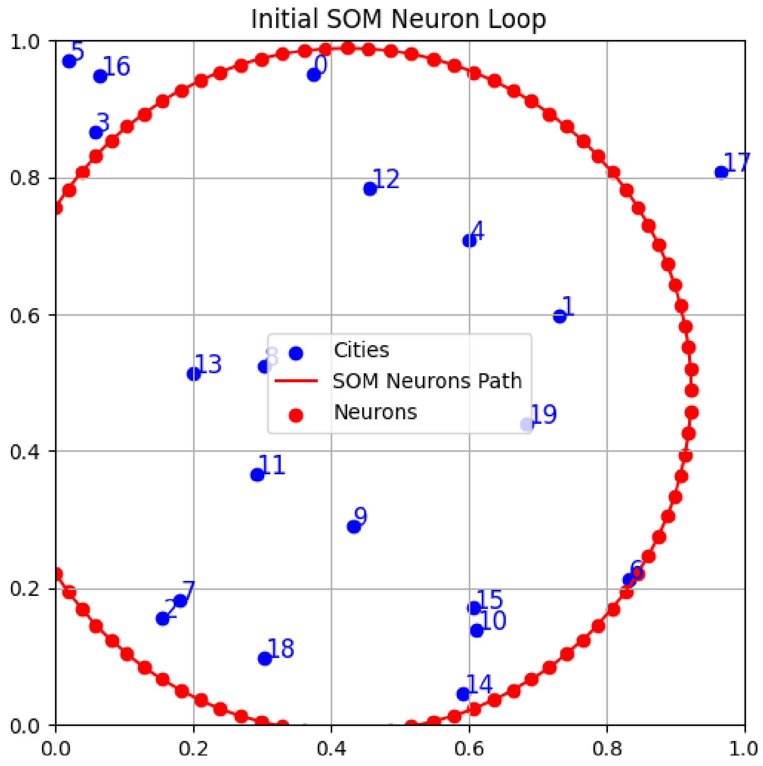
$w_i(t+1)=w_i(t)+\alpha(t)\cdot h_i\mathrm{BMU}(t)\cdot(x-w_i(t$

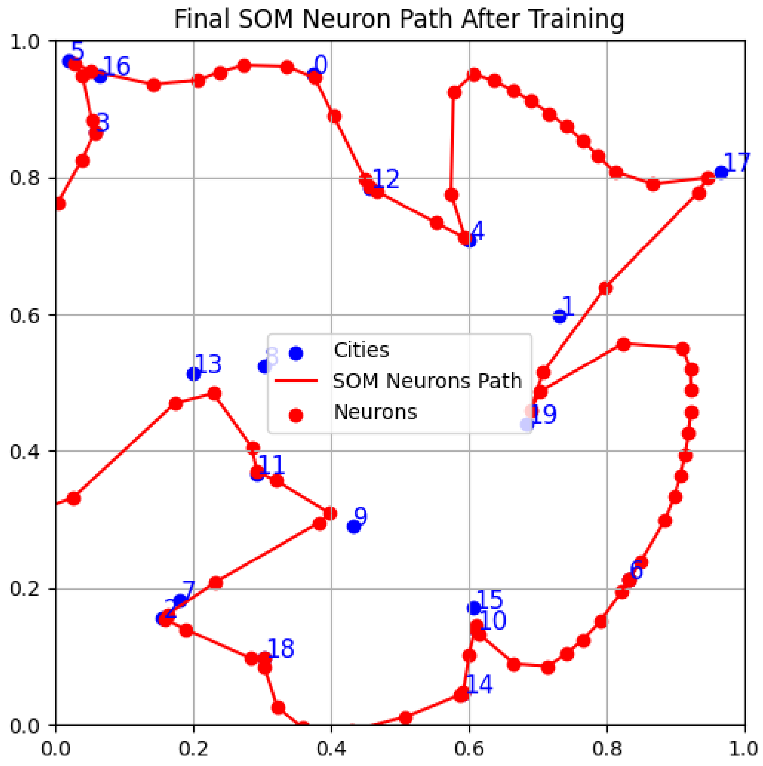
where:

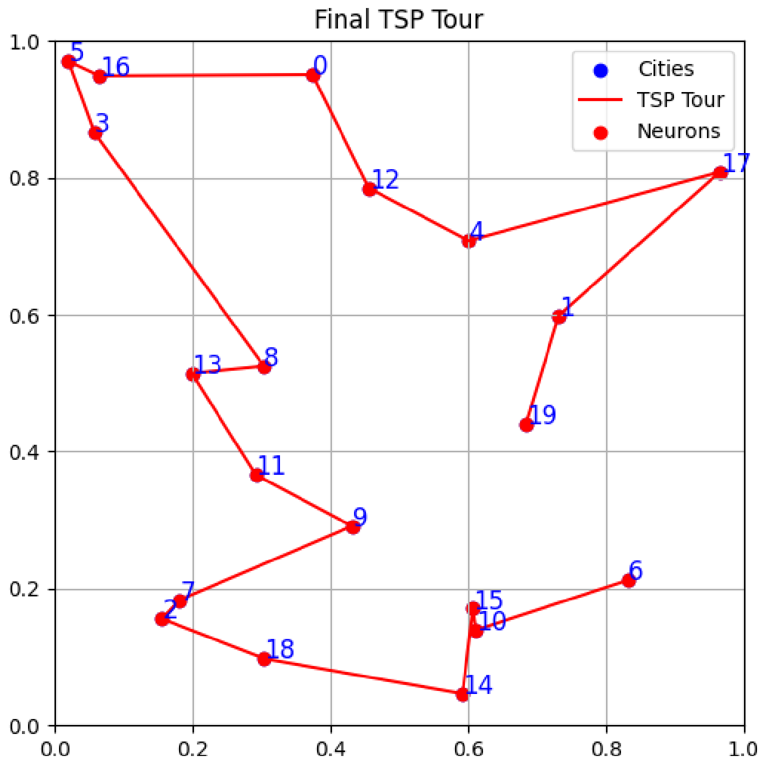
- $wi(t)$ is the weight vector of node i at iteration t,
- $\alpha(t)$ is the learning rate, which decreases over time,
- h_i BMU(t) is the neighborhood function, which determines the strength of the update for node i based on its distance to the BMU. typically decreases with the distance from the BMU and also shrinks over time.


2D output
K neurons lattice
$\vec{x}_1$
$\vec{x}_2$
Input layer
Weights matrix
Neuron i
$\vec{w}_i = [w_{i1}, w_{i2}, \ldots w_{im}]$
$\vec{x}_n = [x_{n1}, x_{n2}, \ldots x_{nm}]$

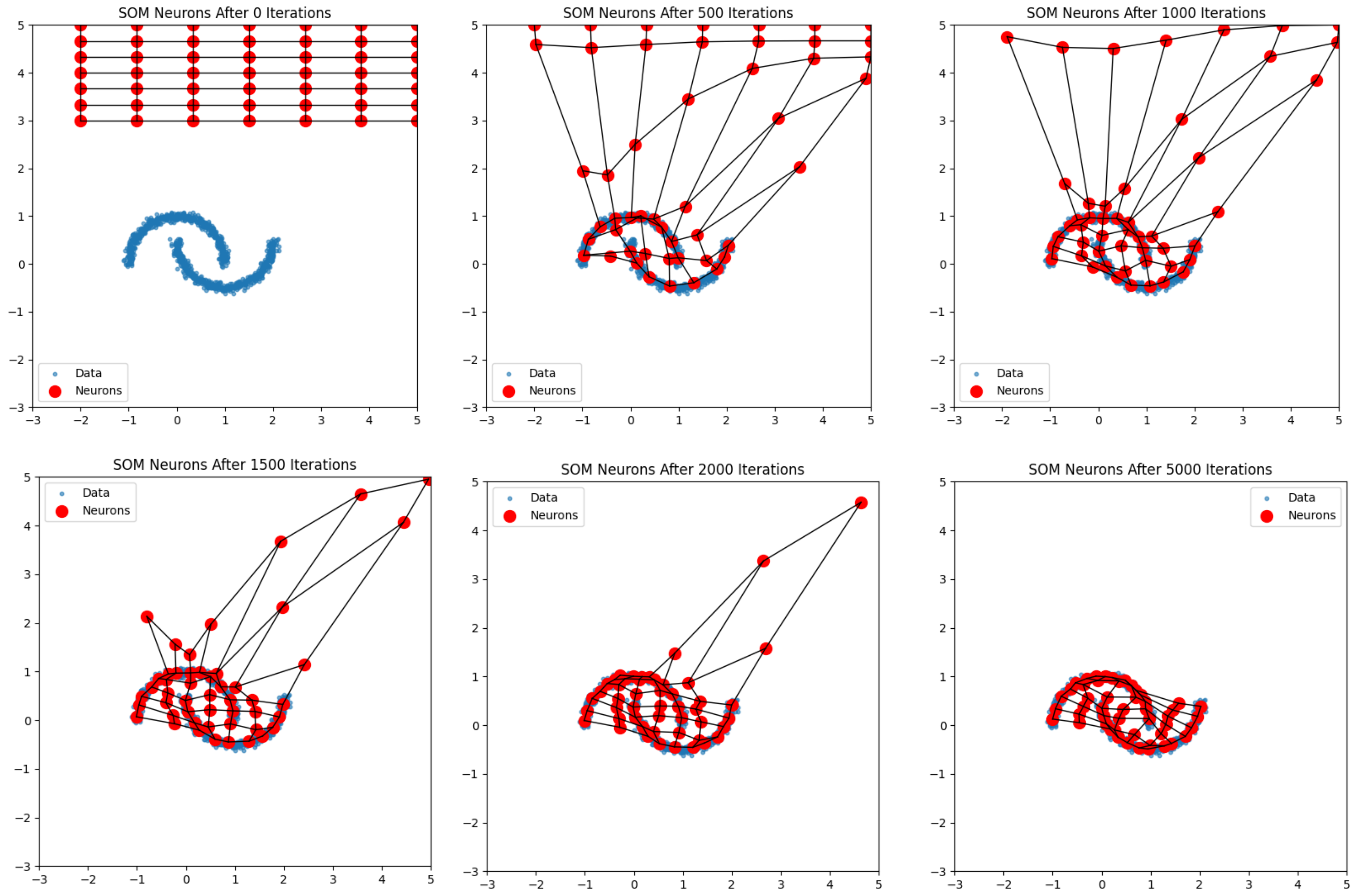
https://www.latentview.com/blog/self-organizing-maps/


Initial SOM Neuron Loop
Cities
SOM Neurons Path
Neurons
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

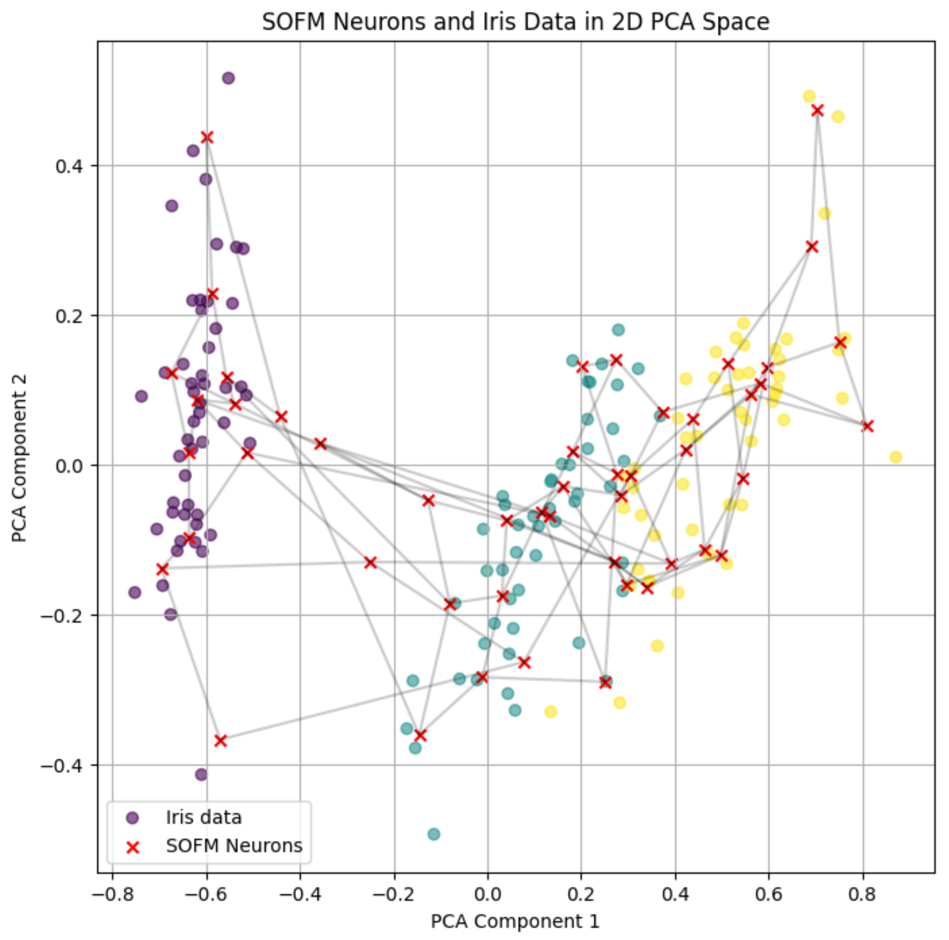

Final SOM Neuron Path After Training
Cities
SOM Neurons Path
Neurons
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

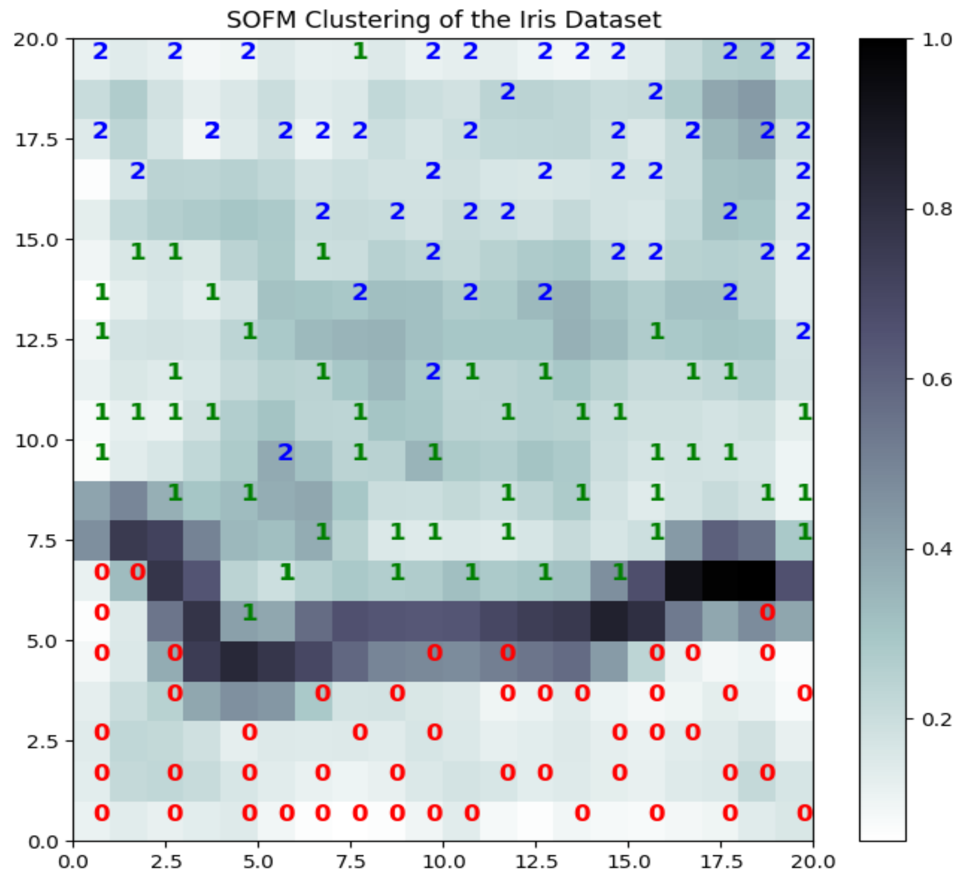

Final TSP Tour
Cities
TSP Tour
Neurons
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Self-Organized Feature Maps



Self-Organized Feature Maps





Have fun with midterm!