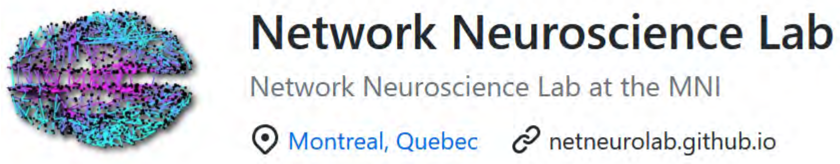
IPN Summer School 2021

Advanced Analytics for Neuroscience

Dimensionality Reduction



Network Neuroscience Lab

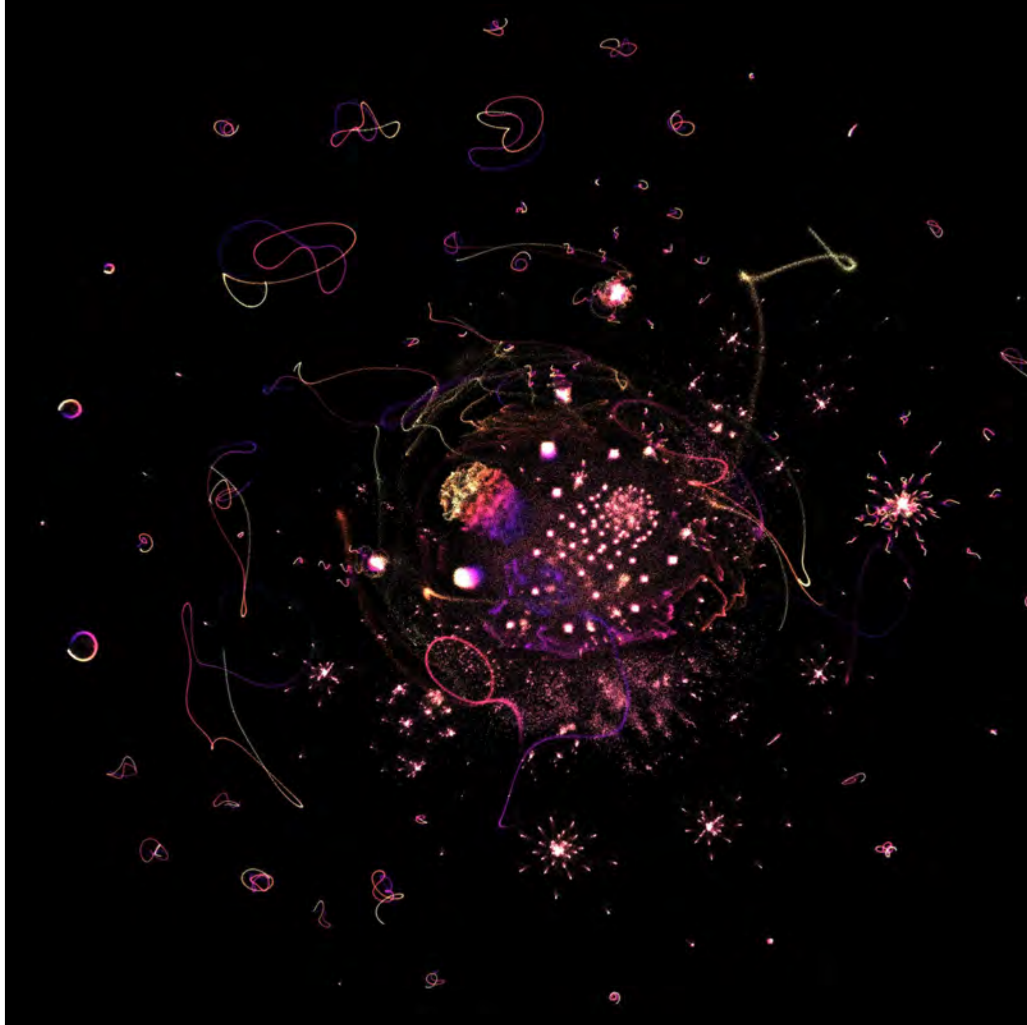
Network Neuroscience Lab at the MNI

Montreal, Quebec netneurolab.github.io

Zhen-Qi Liu

PhD student

zhenqi.liu@mail.mcgill.ca



One million integers embedded into 2D space with UMAP
johnhw.github.io

Outline for today

Goal: ***filling the missing part of other introductions***

What you will learn:

- What is dimensionality reduction
- Why do we need dimension reduction

1. From variable selection to construction
   - PCA & FA & ICA
2. From linear to nonlinear
   - Diffusion map
3. From decomposition to approximation
   - tSNE & UMAP
4. From dimensions to categories

- Back to the future

Outline for today

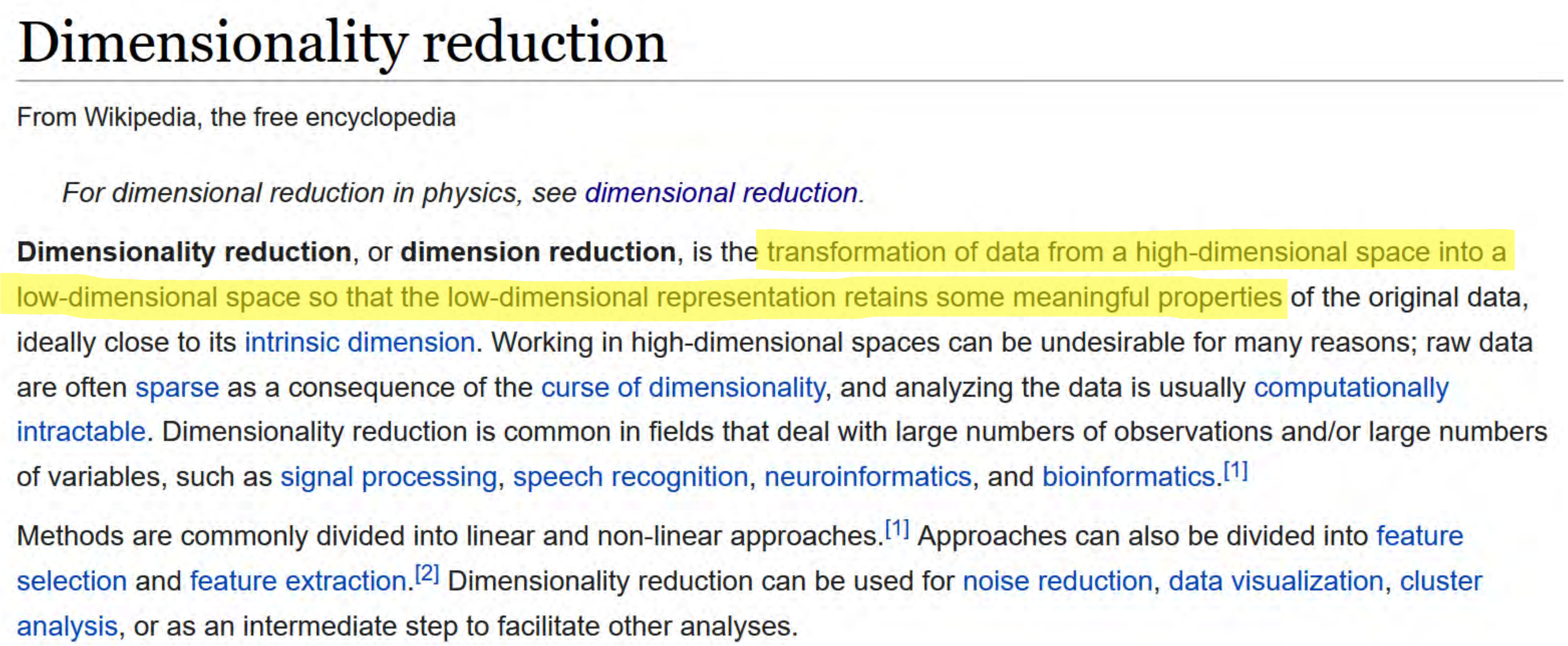
Goal: ***filling the missing part of other introductions***

What you will learn:

- What is dimensionality reduction
- Why do we need dimension reduction

What is dimensionality reduction

Back to where you first met it #1: Wikipedia



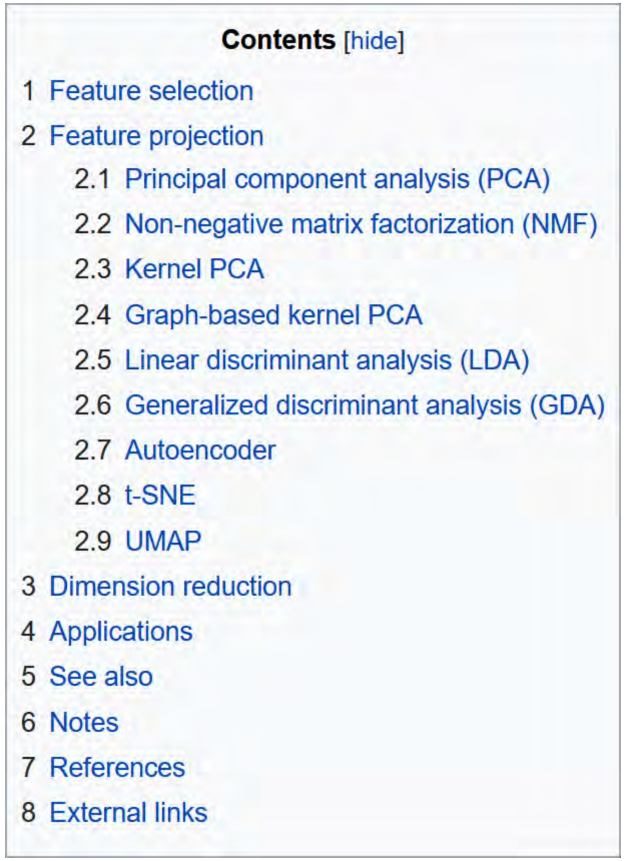
Dimensionality reduction

From Wikipedia, the free encyclopedia

For dimensional reduction in physics, see dimensional reduction.

Dimensionality reduction, or **dimension reduction**, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Working in high-dimensional spaces can be undesirable for many reasons; raw data are often sparse as a consequence of the curse of dimensionality, and analyzing the data is usually computationally intractable. Dimensionality reduction is common in fields that deal with large numbers of observations and/or large numbers of variables, such as signal processing, speech recognition, neuroinformatics, and bioinformatics.[1]

Methods are commonly divided into linear and non-linear approaches.[1] Approaches can also be divided into feature selection and feature extraction.[2] Dimensionality reduction can be used for noise reduction, data visualization, cluster analysis, or as an intermediate step to facilitate other analyses.



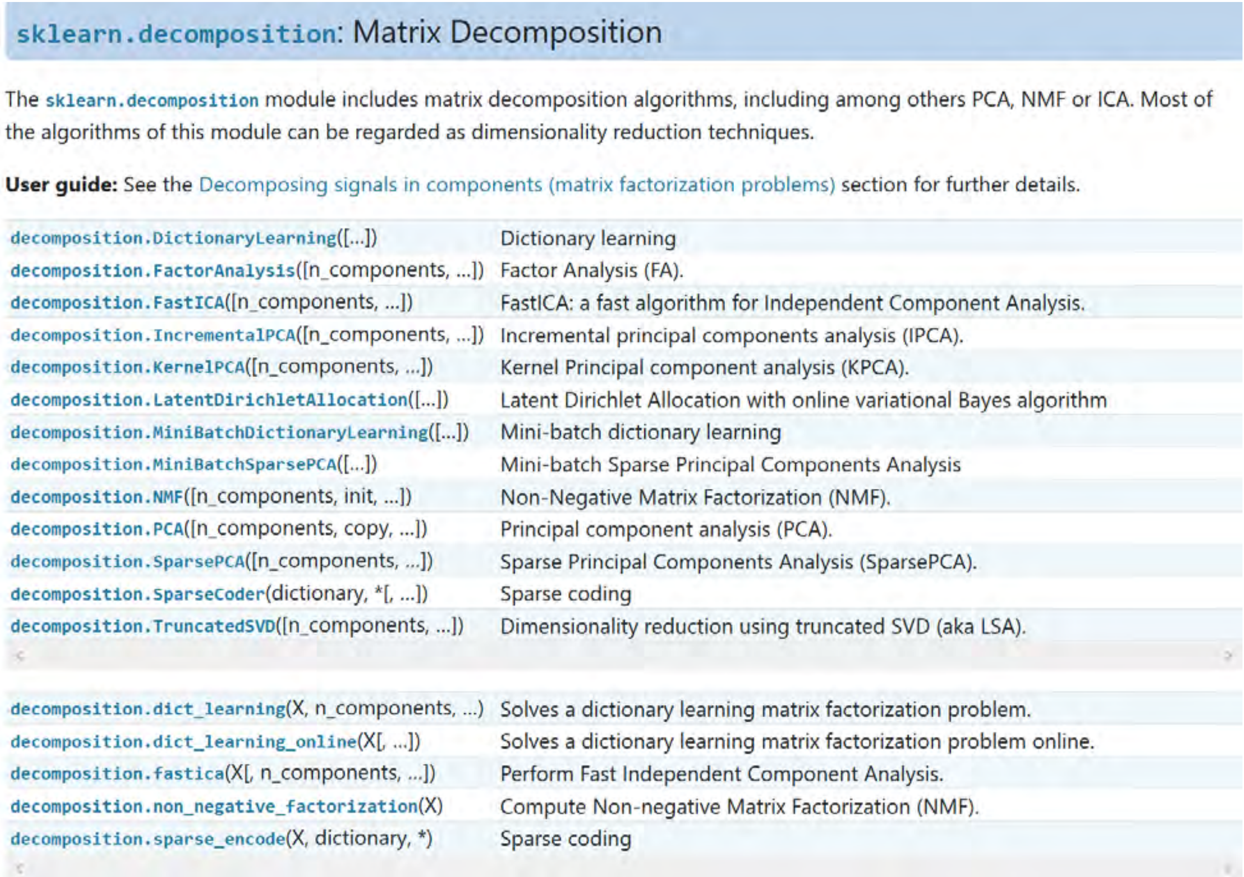
Contents [hide]

1 Feature selection
2 Feature projection
2.1 Principal component analysis (PCA)
2.2 Non-negative matrix factorization (NMF)
2.3 Kernel PCA
2.4 Graph-based kernel PCA
2.5 Linear discriminant analysis (LDA)
2.6 Generalized discriminant analysis (GDA)
2.7 Autoencoder
2.8 t-SNE
2.9 UMAP
3 Dimension reduction
4 Applications
5 See also
6 Notes
7 References
8 External links

(https://en.wikipedia.org/wiki/Dimensionality_reduction)

What is dimensionality reduction

Back to where you first met it #2: Scikit-learn



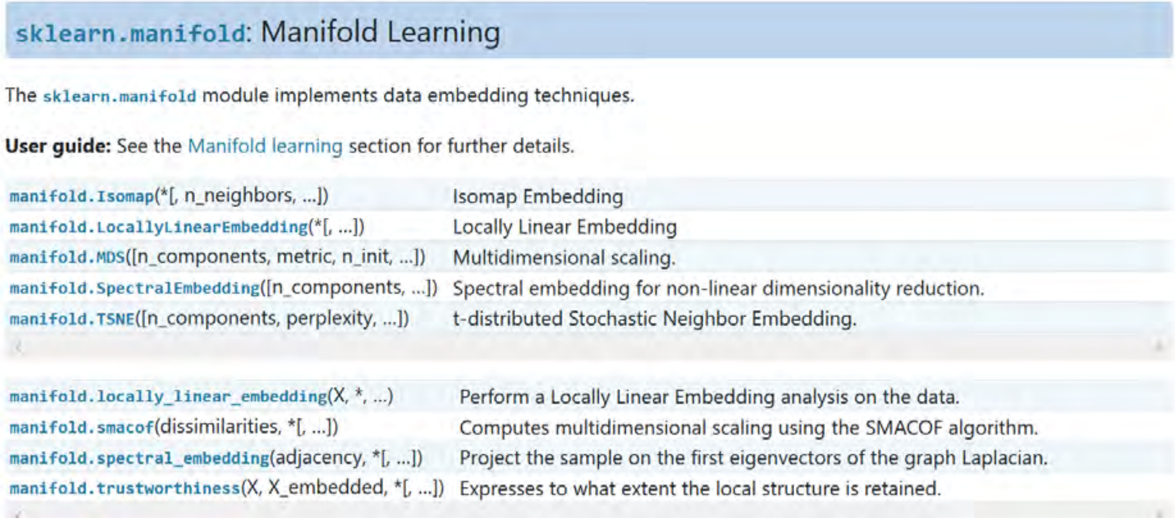
sklearn.decomposition: Matrix Decomposition

The sklearn.decomposition module includes matrix decomposition algorithms, including among others PCA, NMF or ICA. Most of the algorithms of this module can be regarded as dimensionality reduction techniques.

User guide: See the Decomposing signals in components (matrix factorization problems) section for further details.

decomposition.DictionaryLearning([...])	Dictionary learning
decomposition.FactorAnalysis([n_components, ...])	Factor Analysis (FA).
decomposition.FastICA([n_components, ...])	FastICA: a fast algorithm for Independent Component Analysis.
decomposition.IncrementalPCA([n_components, ...])	Incremental principal components analysis (IPCA).
decomposition.KernelPCA([n_components, ...])	Kernel Principal component analysis (KPCA).
decomposition.LatentDirichletAllocation([...])	Latent Dirichlet Allocation with online variational Bayes algorithm
decomposition.MiniBatchDictionaryLearning([...])	Mini-batch dictionary learning
decomposition.MiniBatchSparsePCA([...])	Mini-batch Sparse Principal Components Analysis
decomposition.NMF([n_components, init, ...])	Non-Negative Matrix Factorization (NMF).
decomposition.PCA([n_components, copy, ...])	Principal component analysis (PCA).
decomposition.SparsePCA([n_components, ...])	Sparse Principal Components Analysis (SparsePCA).
decomposition.SparseCoder(dictionary, *[, ...])	Sparse coding
decomposition.TruncatedSVD([n_components, ...])	Dimensionality reduction using truncated SVD (aka LSA).

decomposition.dict_learning(X, n_components, ...)	Solves a dictionary learning matrix factorization problem.
decomposition.dict_learning_online(X[, ...])	Solves a dictionary learning matrix factorization problem online.
decomposition.fastica(X[, n_components, ...])	Perform Fast Independent Component Analysis.
decomposition.non_negative_factorization(X)	Compute Non-negative Matrix Factorization (NMF).
decomposition.sparse_encode(X, dictionary, *)	Sparse coding



sklearn.manifold: Manifold Learning

The sklearn.manifold module implements data embedding techniques.

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

manifold.Isomap(*[, n_neighbors, ...])	Isomap Embedding
manifold.LocallyLinearEmbedding(*[, ...])	Locally Linear Embedding
manifold.MDS([n_components, metric, n_init, ...])	Multidimensional scaling.
manifold.SpectralEmbedding([n_components, ...])	Spectral embedding for non-linear dimensionality reduction.
manifold.TSNE([n_components, perplexity, ...])	t-distributed Stochastic Neighbor Embedding.

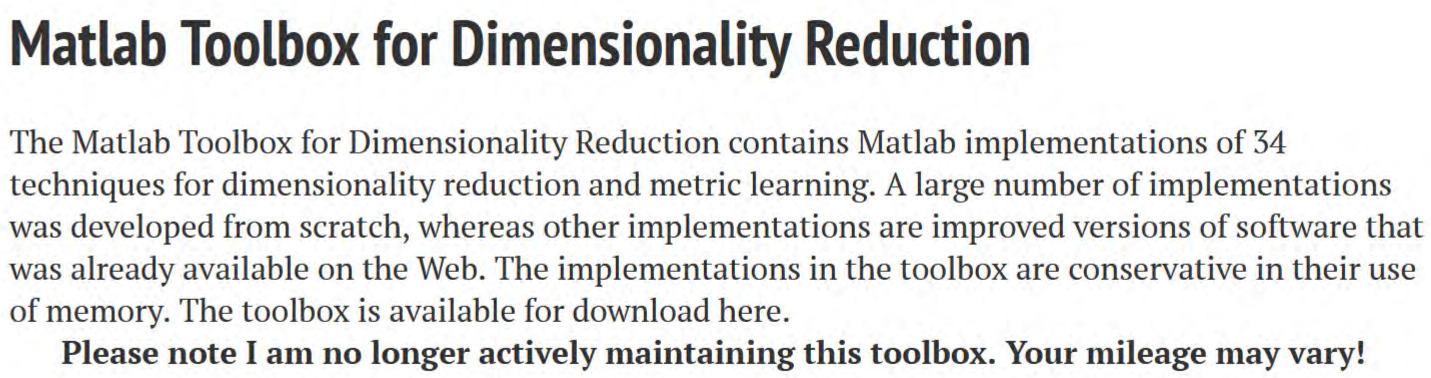
manifold.locally_linear_embedding(X, *, ...)	Perform a Locally Linear Embedding analysis on the data.
manifold.smacof(dissimilarities, *[, ...])	Computes multidimensional scaling using the SMACOF algorithm.
manifold.spectral_embedding(adjacency, *[, ...])	Project the sample on the first eigenvectors of the graph Laplacian.
manifold.trustworthiness(X, X_embedded, *[, ...])	Expresses to what extent the local structure is retained.

(https://scikit-learn.org)

What is dimensionality reduction

Back to where you first met it #2’

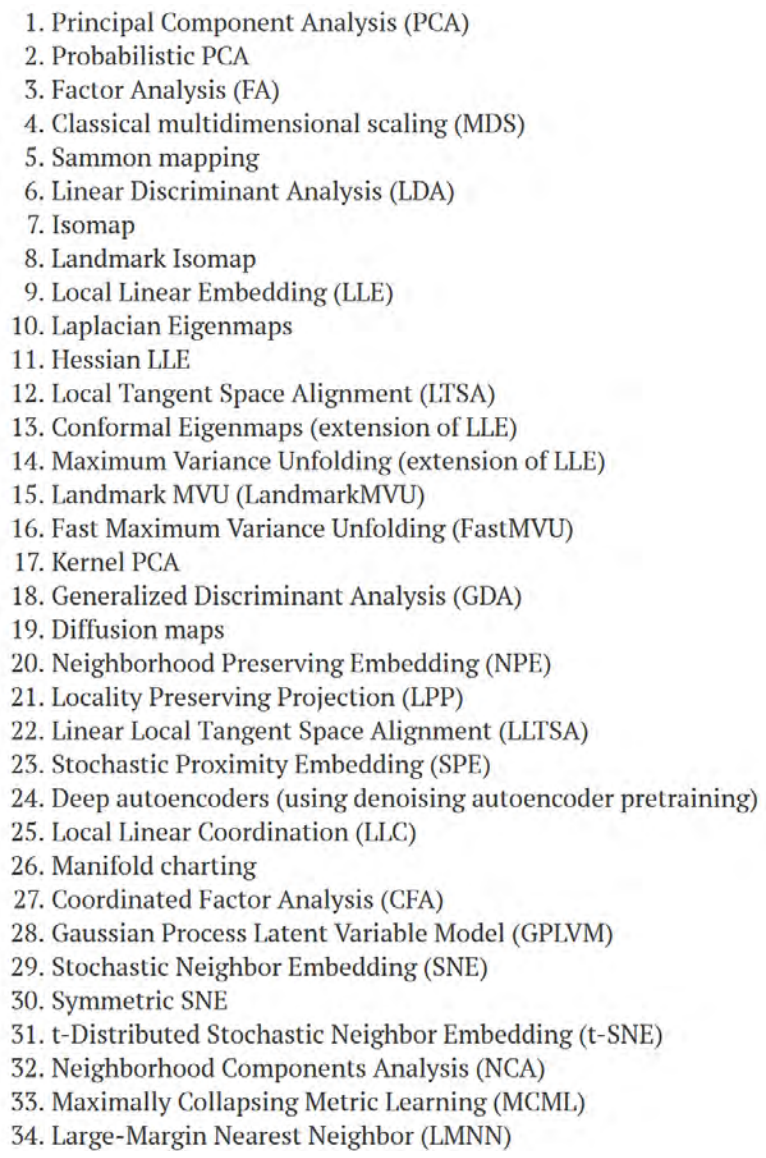
- If you use MATLAB



Matlab Toolbox for Dimensionality Reduction

The Matlab Toolbox for Dimensionality Reduction contains Matlab implementations of 34 techniques for dimensionality reduction and metric learning. A large number of implementations was developed from scratch, whereas other implementations are improved versions of software that was already available on the Web. The implementations in the toolbox are conservative in their use of memory. The toolbox is available for download here.

Please note I am no longer actively maintaining this toolbox. Your mileage may vary!



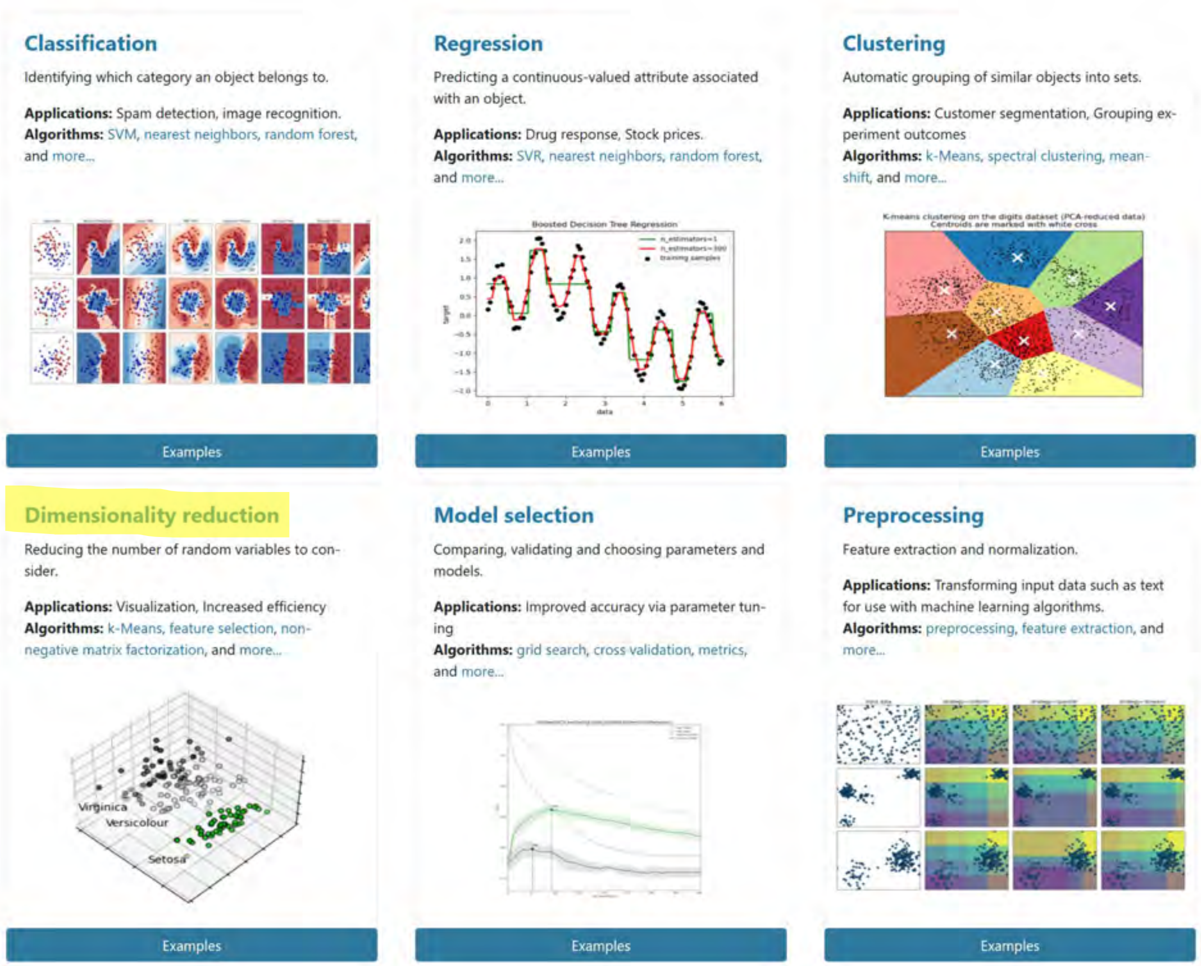
1. Principal Component Analysis (PCA)
2. Probabilistic PCA
3. Factor Analysis (FA)
4. Classical multidimensional scaling (MDS)
5. Sammon mapping
6. Linear Discriminant Analysis (LDA)
7. Isomap
8. Landmark Isomap
9. Local Linear Embedding (LLE)
10. Laplacian Eigenmaps
11. Hessian LLE
12. Local Tangent Space Alignment (LTSA)
13. Conformal Eigenmaps (extension of LLE)
14. Maximum Variance Unfolding (extension of LLE)
15. Landmark MVU (LandmarkMVU)
16. Fast Maximum Variance Unfolding (FastMVU)
17. Kernel PCA
18. Generalized Discriminant Analysis (GDA)
19. Diffusion maps
20. Neighborhood Preserving Embedding (NPE)
21. Locality Preserving Projection (LPP)
22. Linear Local Tangent Space Alignment (LLTSA)
23. Stochastic Proximity Embedding (SPE)
24. Deep autoencoders (using denoising autoencoder pretraining)
25. Local Linear Coordination (LLC)
26. Manifold charting
27. Coordinated Factor Analysis (CFA)
28. Gaussian Process Latent Variable Model (GPLVM)
29. Stochastic Neighbor Embedding (SNE)
30. Symmetric SNE
31. t-Distributed Stochastic Neighbor Embedding (t-SNE)
32. Neighborhood Components Analysis (NCA)
33. Maximally Collapsing Metric Learning (MCML)
34. Large-Margin Nearest Neighbor (LMNN)

(https://lvdmaaten.github.io/drtoolbox/)

What is dimensionality reduction

Back to where you first met it #3

- Where do I find them



Boosted Decision Tree Regression
n_estimators=1
n_estimators=300
training samples
target
data
K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross
Virginica
Versicolour
Setosa

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...

Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

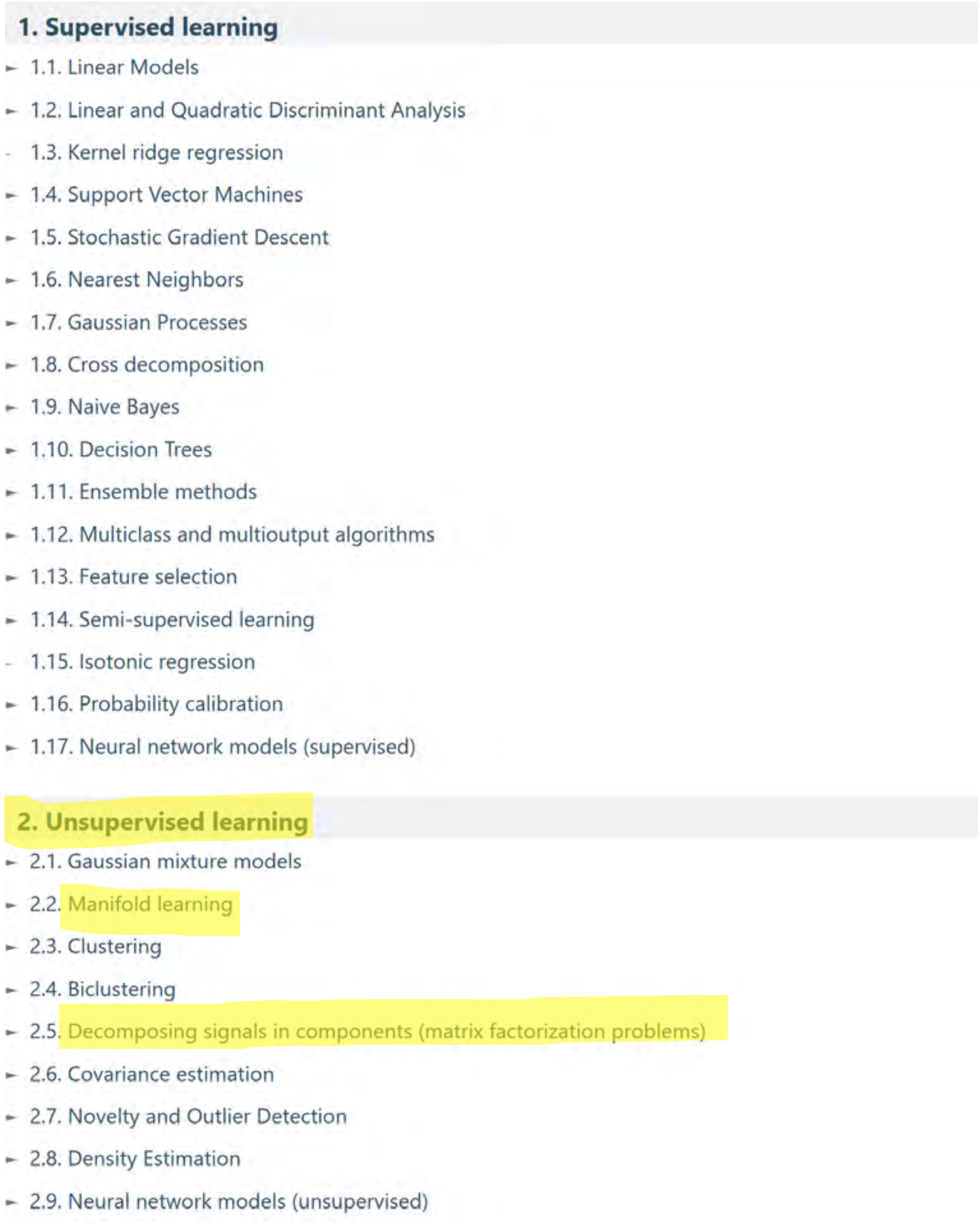
Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...

Examples



1. Supervised learning

- 1.1. Linear Models
- 1.2. Linear and Quadratic Discriminant Analysis
- 1.3. Kernel ridge regression
- 1.4. Support Vector Machines
- 1.5. Stochastic Gradient Descent
- 1.6. Nearest Neighbors
- 1.7. Gaussian Processes
- 1.8. Cross decomposition
- 1.9. Naive Bayes
- 1.10. Decision Trees
- 1.11. Ensemble methods
- 1.12. Multiclass and multioutput algorithms
- 1.13. Feature selection
- 1.14. Semi-supervised learning
- 1.15. Isotonic regression
- 1.16. Probability calibration
- 1.17. Neural network models (supervised)

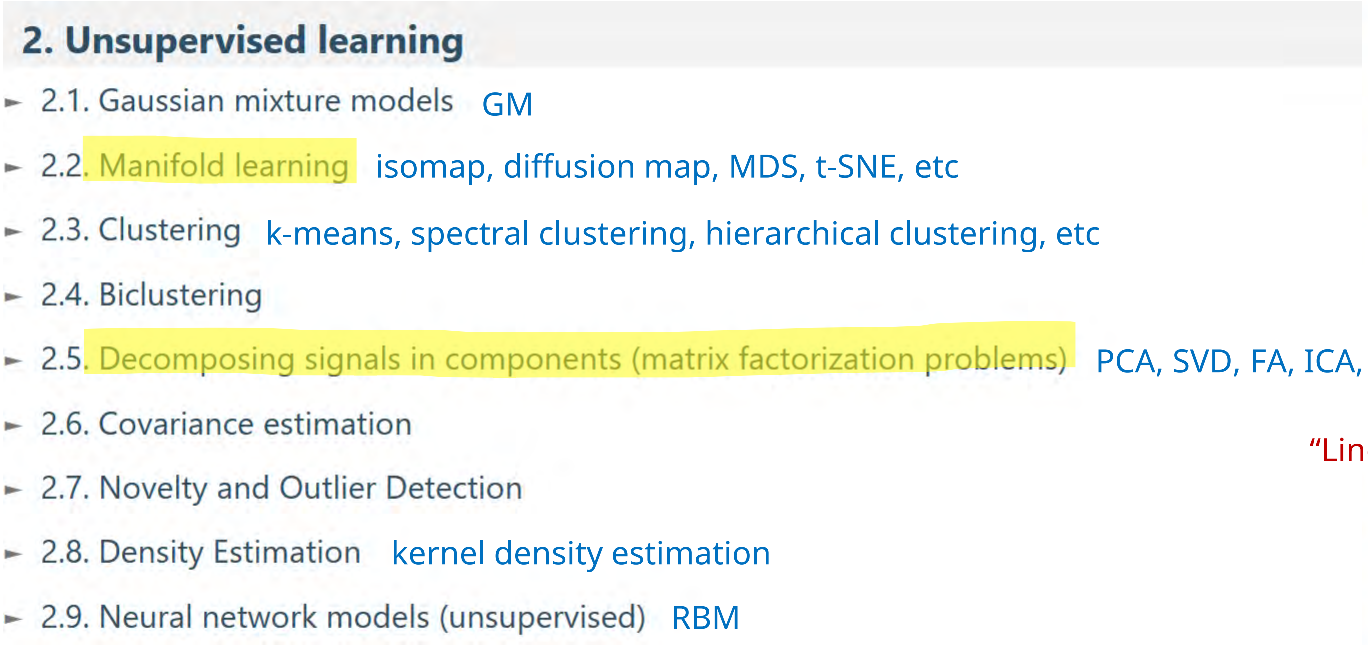
2. Unsupervised learning

- 2.1. Gaussian mixture models
- 2.2. Manifold learning
- 2.3. Clustering
- 2.4. Biclustering
- 2.5. Decomposing signals in components (matrix factorization problems)
- 2.6. Covariance estimation
- 2.7. Novelty and Outlier Detection
- 2.8. Density Estimation
- 2.9. Neural network models (unsupervised)

What is dimensionality reduction

Back to where you first met it #3’

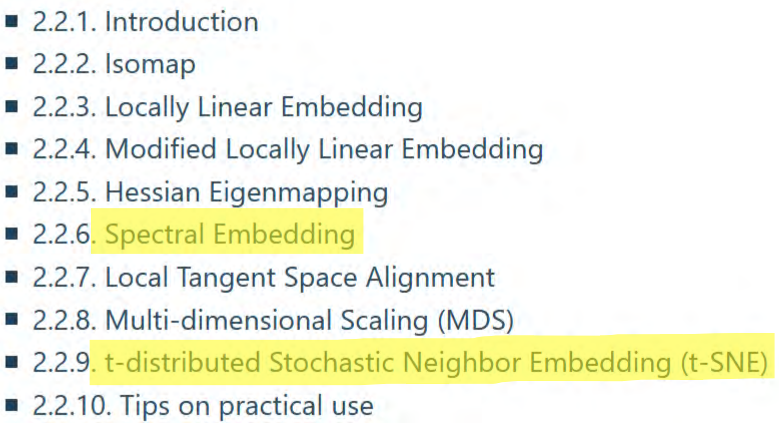
- Where do I find them: A closer look



2. Unsupervised learning

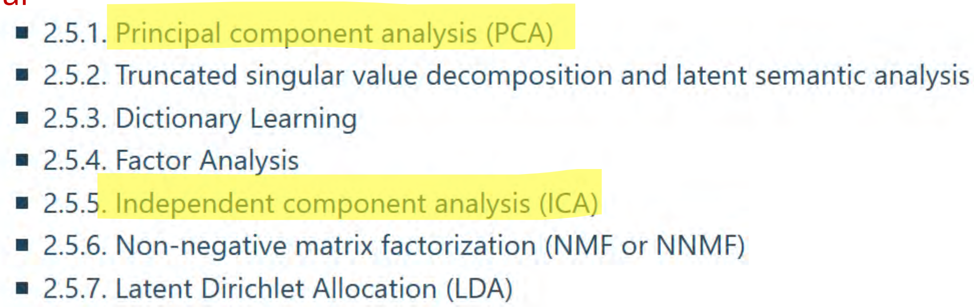
- 2.1. Gaussian mixture models GM
- 2.2. Manifold learning isomap, diffusion map, MDS, t-SNE, etc
- 2.3. Clustering k-means, spectral clustering, hierarchical clustering, etc
- 2.4. Biclustering
- 2.5. Decomposing signals in components (matrix factorization problems) PCA, SVD, FA, ICA, NMP, LDA, etc
- 2.6. Covariance estimation
- 2.7. Novelty and Outlier Detection
- 2.8. Density Estimation kernel density estimation
- 2.9. Neural network models (unsupervised) RBM

“Nonlinear”



- 2.2.1. Introduction
- 2.2.2. Isomap
- 2.2.3. Locally Linear Embedding
- 2.2.4. Modified Locally Linear Embedding
- 2.2.5. Hessian Eigenmapping
- 2.2.6. Spectral Embedding
- 2.2.7. Local Tangent Space Alignment
- 2.2.8. Multi-dimensional Scaling (MDS)
- 2.2.9. t-distributed Stochastic Neighbor Embedding (t-SNE)
- 2.2.10. Tips on practical use

“Linear”



- 2.5.1. Principal component analysis (PCA)
- 2.5.2. Truncated singular value decomposition and latent semantic analysis
- 2.5.3. Dictionary Learning
- 2.5.4. Factor Analysis
- 2.5.5. Independent component analysis (ICA)
- 2.5.6. Non-negative matrix factorization (NMF or NNMF)
- 2.5.7. Latent Dirichlet Allocation (LDA)

Why do we need dimensionality reduction

What we don’t always realize…

Humans are notoriously bad at understanding n>3 dimensions

We want to “understand the data”, “find the patterns”

“All” data-driven analyses are effectively some form of dimensionality reduction

- True for pattern recognition, machine learning, deep learning, etc.
- True for regression, classification, clustering, etc.

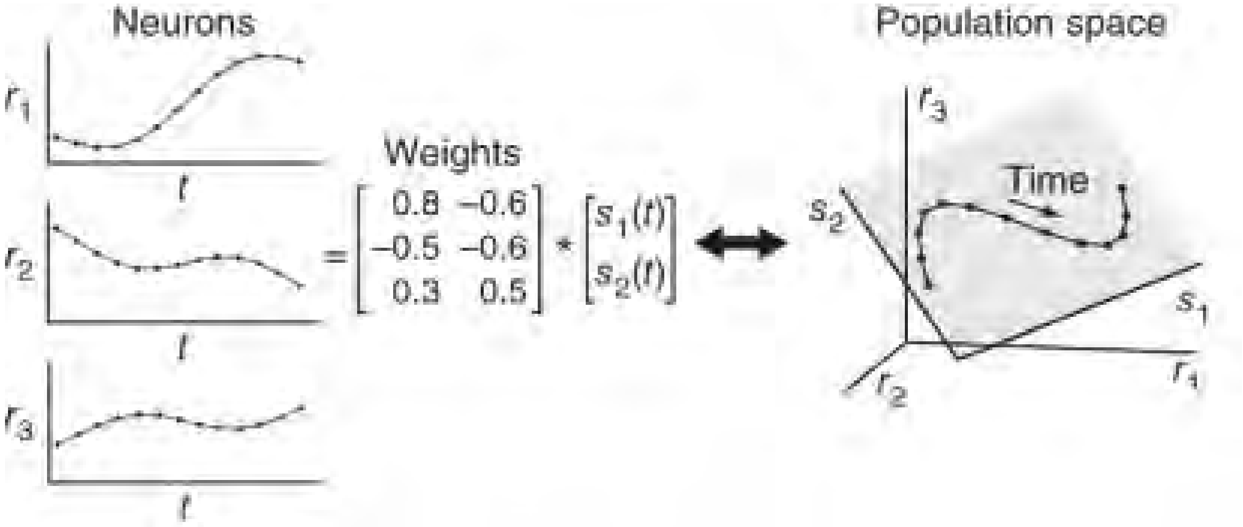
Why do we need dimensionality reduction

What we were actually assuming…

Raw high-dimensional data is often sparse

They often reside in a lower-dimensional “manifold”

We want to find that “manifold” (~ the latent variable)


Neurons
r_1
t
r_2
t
r_3
t
Weights
$= \begin{bmatrix} 0.8 & -0.6 \\ -0.5 & -0.6 \\ 0.3 & 0.5 \end{bmatrix} * \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$
Population space
r_3
Time
s_2
s_1
r_2
r_1

Bonus points:

- Computationally easier for following analysis
- Explainable features or patterns

(Cunningham & Yu, 2014)

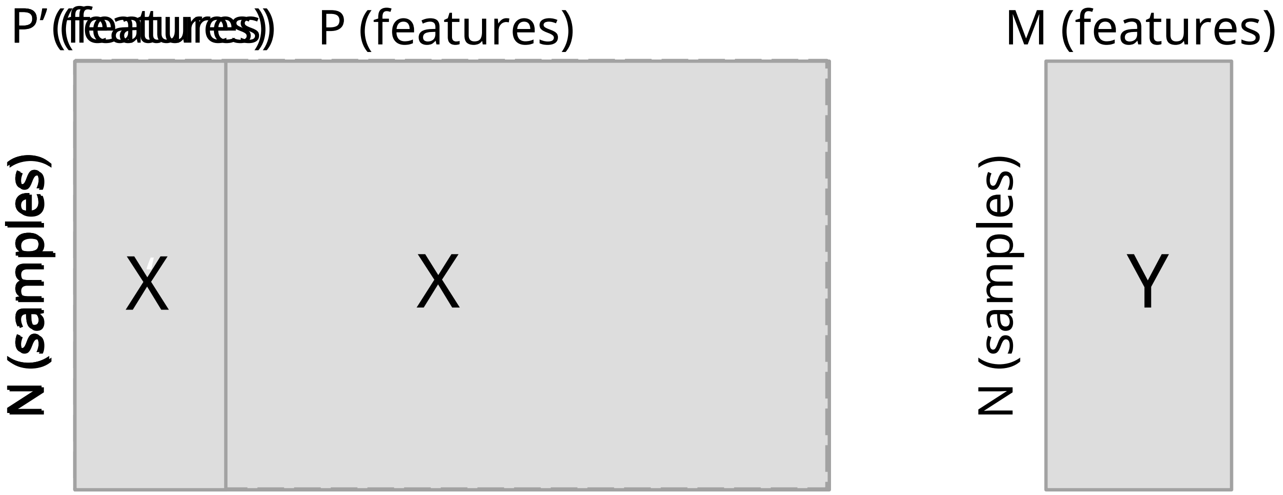
Why do we need dimensionality reduction

Actual problems in research data

- N >> P (desired)
- N ~ P (workable)
- N << P (VERY BAD!!)
  - More unknowns than observations
  - Ill-posed, under-determined, overfitting
  - “Select a few columns”

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

von Neumann



(Tian et al., 2020)

Why do we need dimensionality reduction

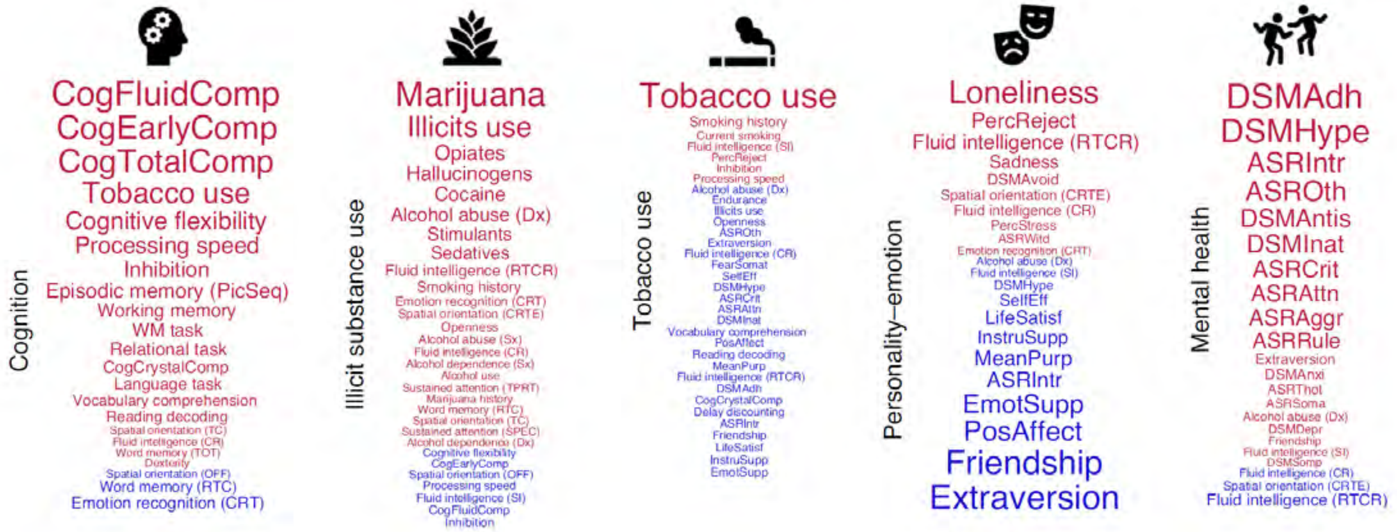
Problems when inspecting high-dimensional data

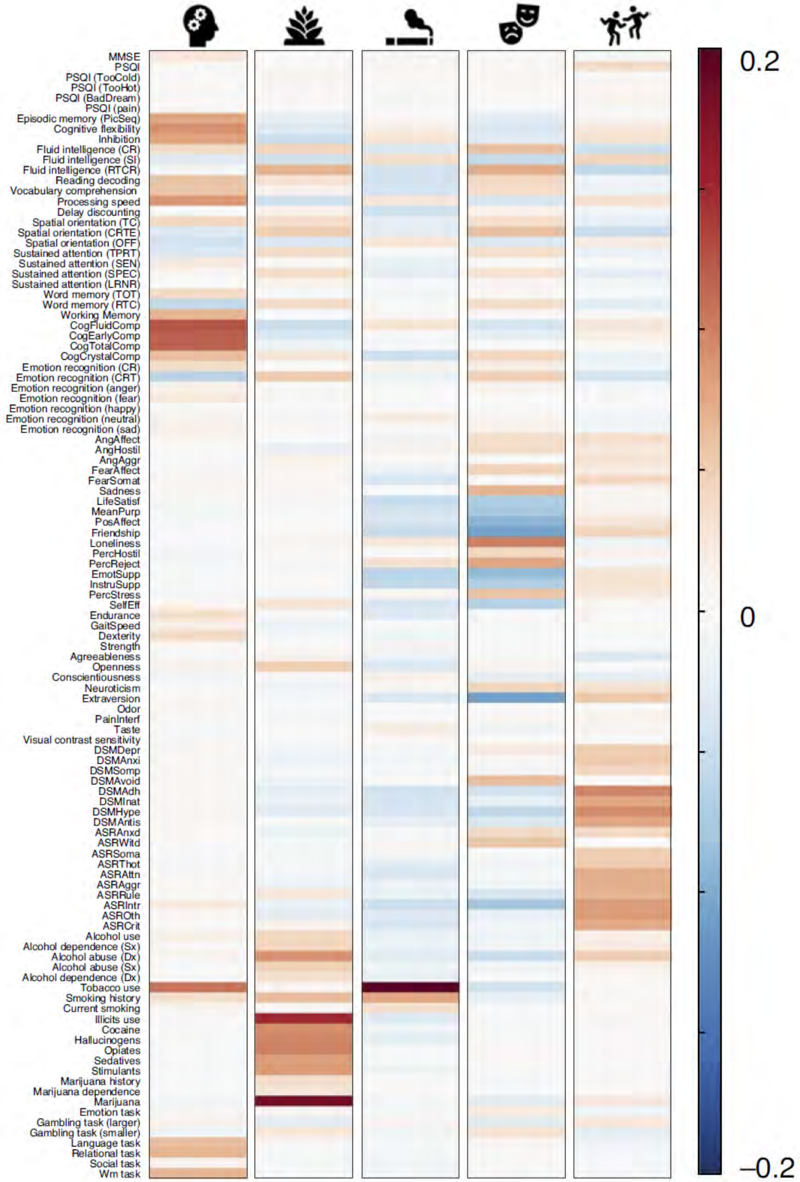
- Raw high-dimensional data is often sparse
- It can be computationally hard to run analysis
- Feature contributions will be hard to explain

Why do we need dimensionality reduction

Lots of examples #1: sparse behavioral data

- Raw high-dimensional data is often sparse
- It can be computationally hard to run analysis
- Feature contributions will be hard to explain





(Tian et al., 2020)

Why do we need dimensionality reduction

Lots of examples #2: brain-computer interface

- Raw high-dimensional data is often sparse
- It can be computationally hard to run analysis
- Feature contributions will be hard to explain

Article

High-performance brain-to-text communication via handwriting

https://doi.org/10.1038/s41586-021-03506-2

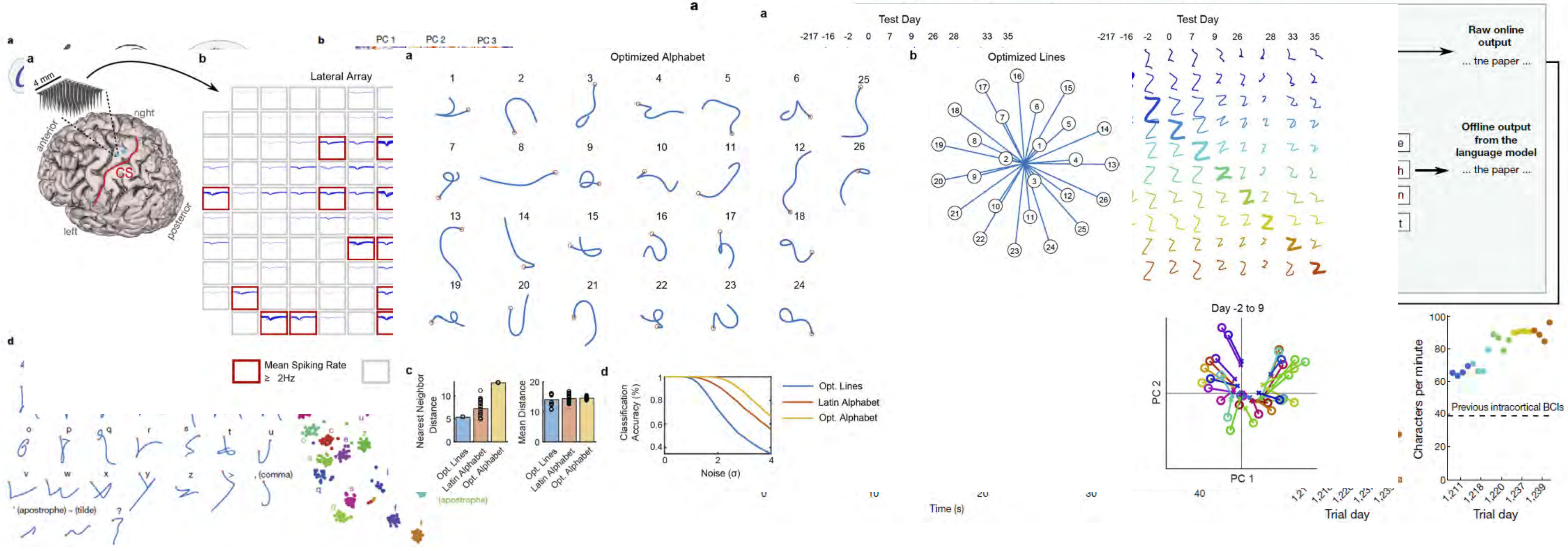
Received: 2 July 2020

Accepted: 26 March 2021

Published online: 12 May 2021

Francis R. Willett[1,2,3], Donald T. Avansino[1], Leigh R. Hochberg[4,5,6,7], Jaimie M. Henderson[2,8,9,12] & Krishna V. Shenoy[1,3,8,9,10,11,12]

Brain–computer interfaces (BCIs) can restore communication to people who have lost the ability to move or speak. So far, a major focus of BCI research has been on

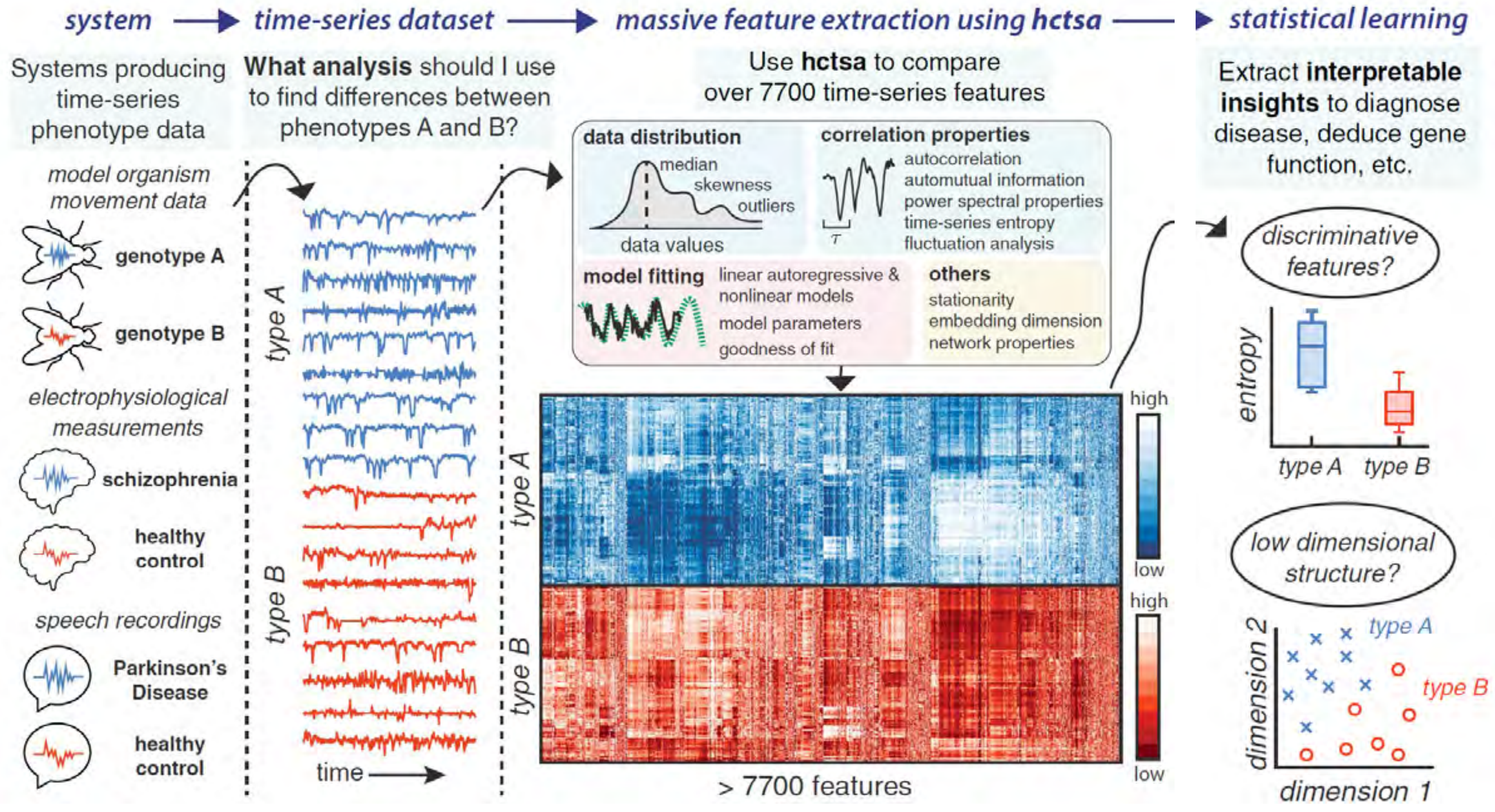


(Willett et al., 2021)

Why do we need dimensionality reduction

Lots of examples #3: massive feature extraction

- Raw high-dimensional data is often sparse
- It can be computationally hard to run analysis
- Feature contributions will be hard to explain

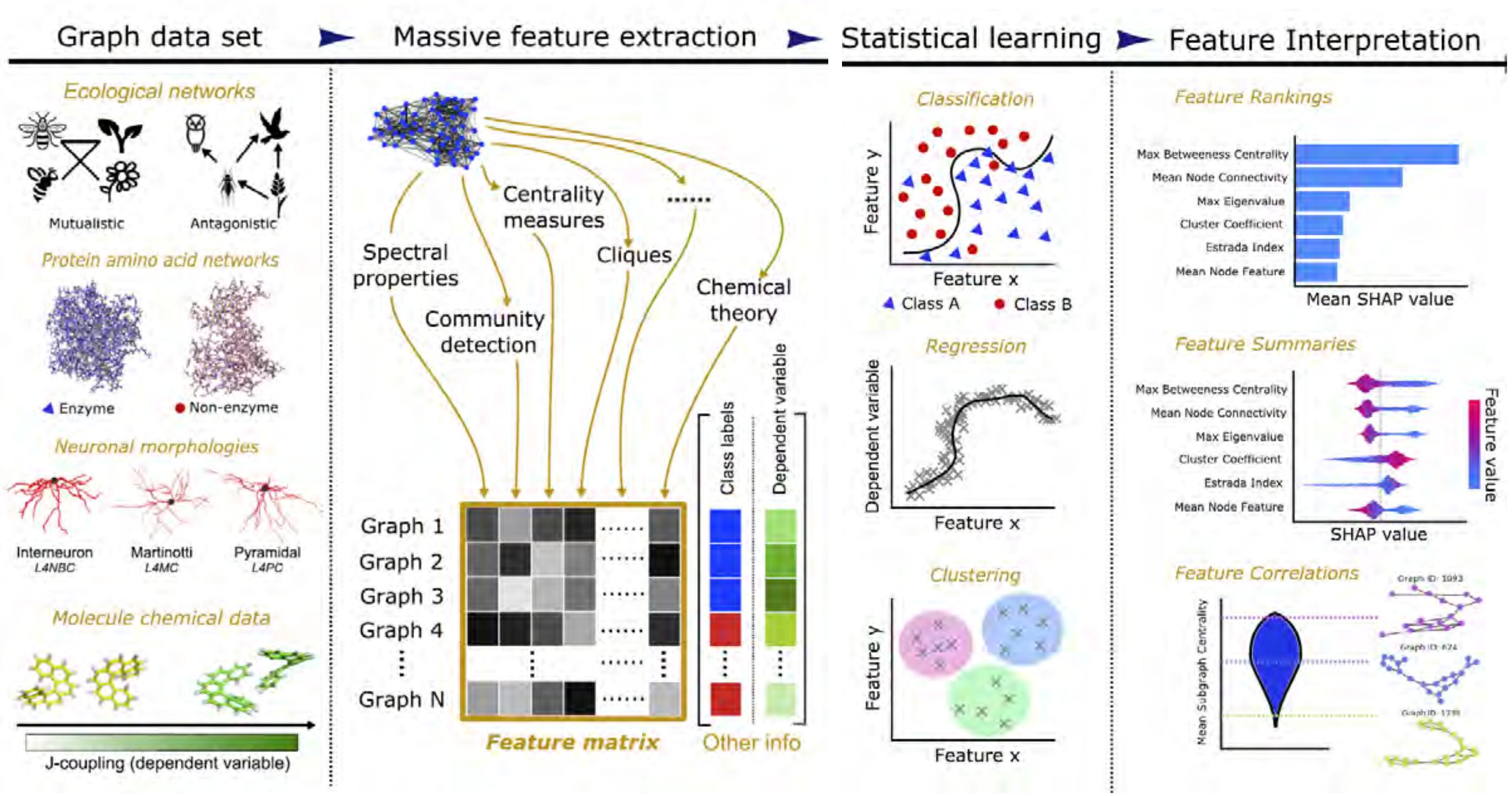


(Fulcher and Jones, 2017)

Why do we need dimensionality reduction

Lots of examples #3’: massive feature extraction (one more)

- Raw high-dimensional data is often sparse
- It can be computationally hard to run analysis
- Feature contributions will be hard to explain



(Peach et al., 2021)

Why do we need dimensionality reduction

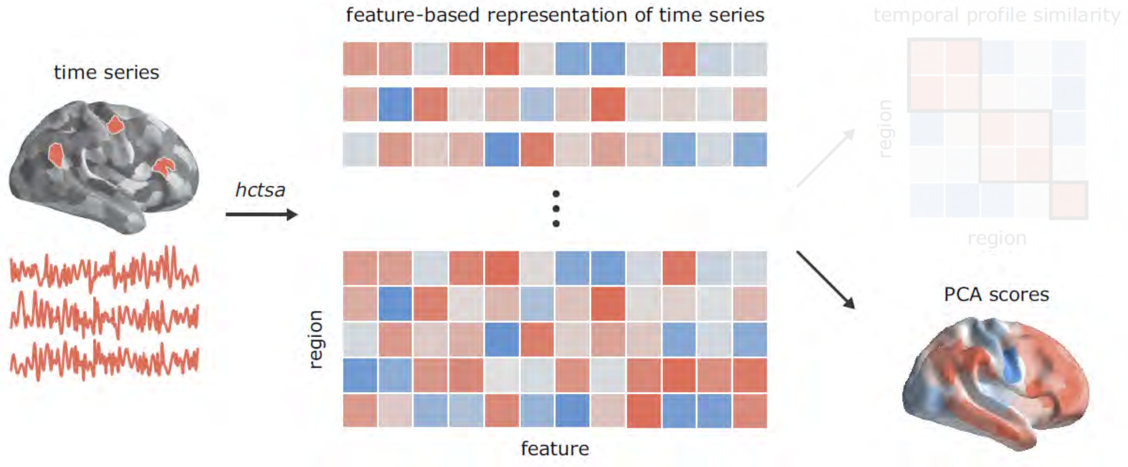
Lots of examples #4: meaningful brain-related findings

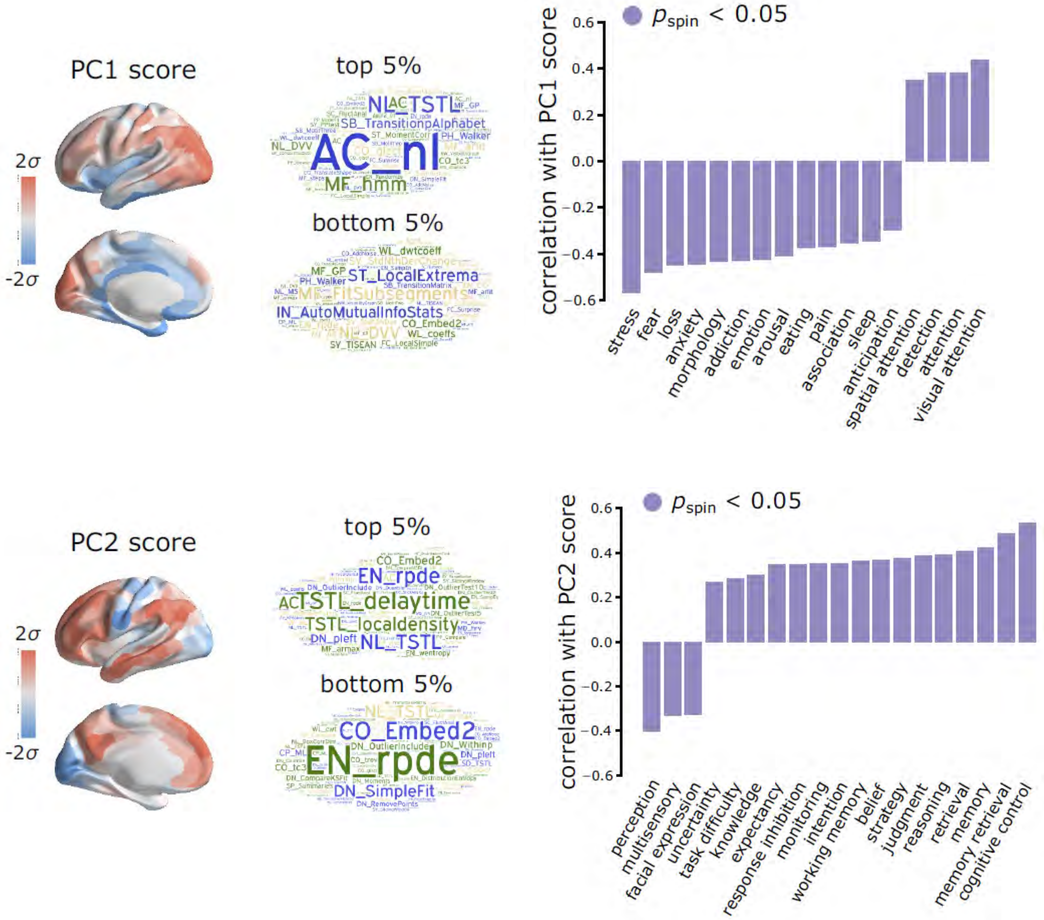
- Raw high-dimensional data is often sparse
- It can be computationally hard to run analysis
- Feature contributions will be hard to explain

Topographic gradients of intrinsic dynamics across neocortex

Golia Shafiei[1]*, Ross D Markello[1], Reinder Vos de Wael[1], Boris C Bernhardt[1], Ben D Fulcher[2], Bratislav Misic[1]*

[1]McConnell Brain Imaging Centre, Montréal Neurological Institute, McGill University, Montréal, Canada; [2]School of Physics, The University of Sydney, Sydney, Australia


time series
hctsa
feature-based representation of time series
region
feature
temporal profile similarity
region
region
PCA scores



(Shafiei et al., 2020)

Outline for today

Goal: ***filling the missing part of other introductions***

What you will learn:

- What is dimensionality reduction
- Why do we need dimension reduction

Outline for today

Goal: ***filling the missing part of other introductions***

What you will learn:

- What is dimensionality reduction
- Why do we need dimension reduction

1. From variable selection to construction
   - PCA & FA & ICA
2. From linear to nonlinear
   - Diffusion map
3. From decomposition to approximation
   - tSNE & UMAP
4. From dimensions to categories

- Back to the future

We select variables

Best subset, ridge, LASSO…

We make new variables

PCA, FA, ICA…

We distort the data

Diffusion map, Isomap, LLE…

We make guesses

tSNE, UMAP…

What's next?

Outline for today

Goal: ***filling the missing part of other introductions***

What you will learn:

- What is dimensionality reduction
- Why do we need dimension reduction

1. From variable selection to construction
    - PCA & FA & ICA
2. From linear to nonlinear
    - Diffusion map
3. From decomposition to approximation
    - tSNE & UMAP
4. From dimensions to categories

- Back to the future

Side Notes

- I would focus on the big picture
    - …and leave the details & references
- You could ignore the math part!
    - …and come back when you need it
- You don't have to understand all of them!
    - …but I'm sure they'll be useful in the future

Starting point: variable selection

Most primitive: best-subset

More continuity: shrinkage methods

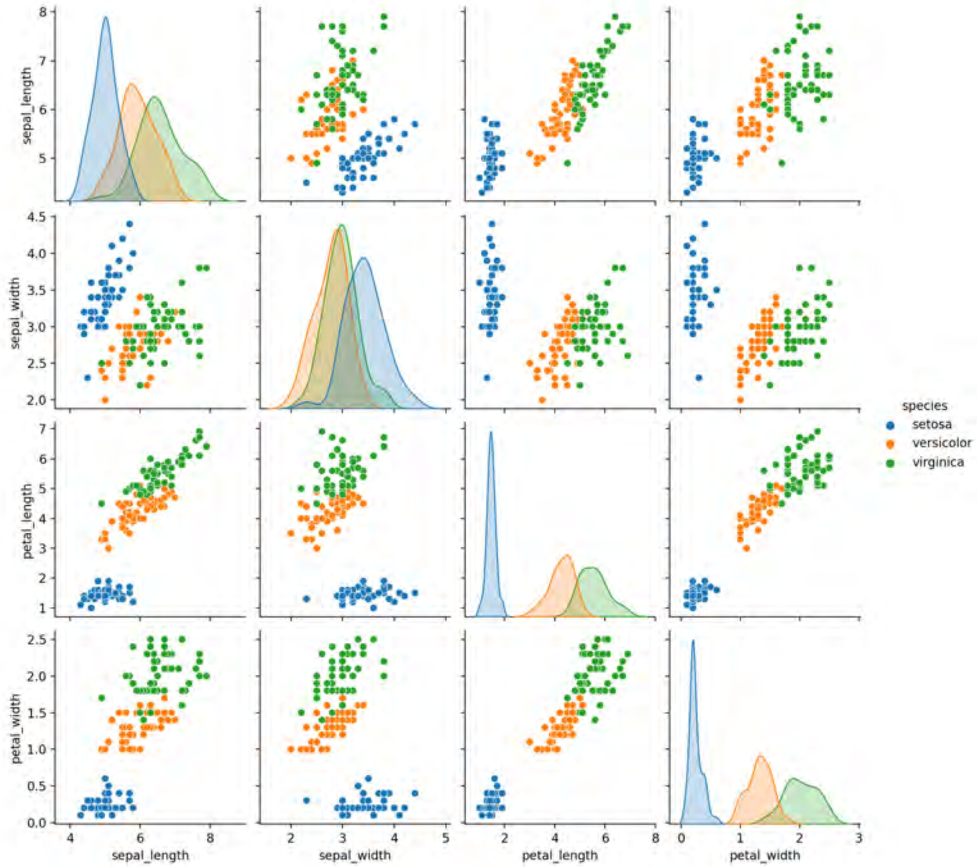
What-if:

- Don’t have a designated task/label (unsupervised)
- More explainable
- Computationally cheaper

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j.$$

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \Big\{ \sum_{i=1}^{N} \big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\big)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \Big\}.$$

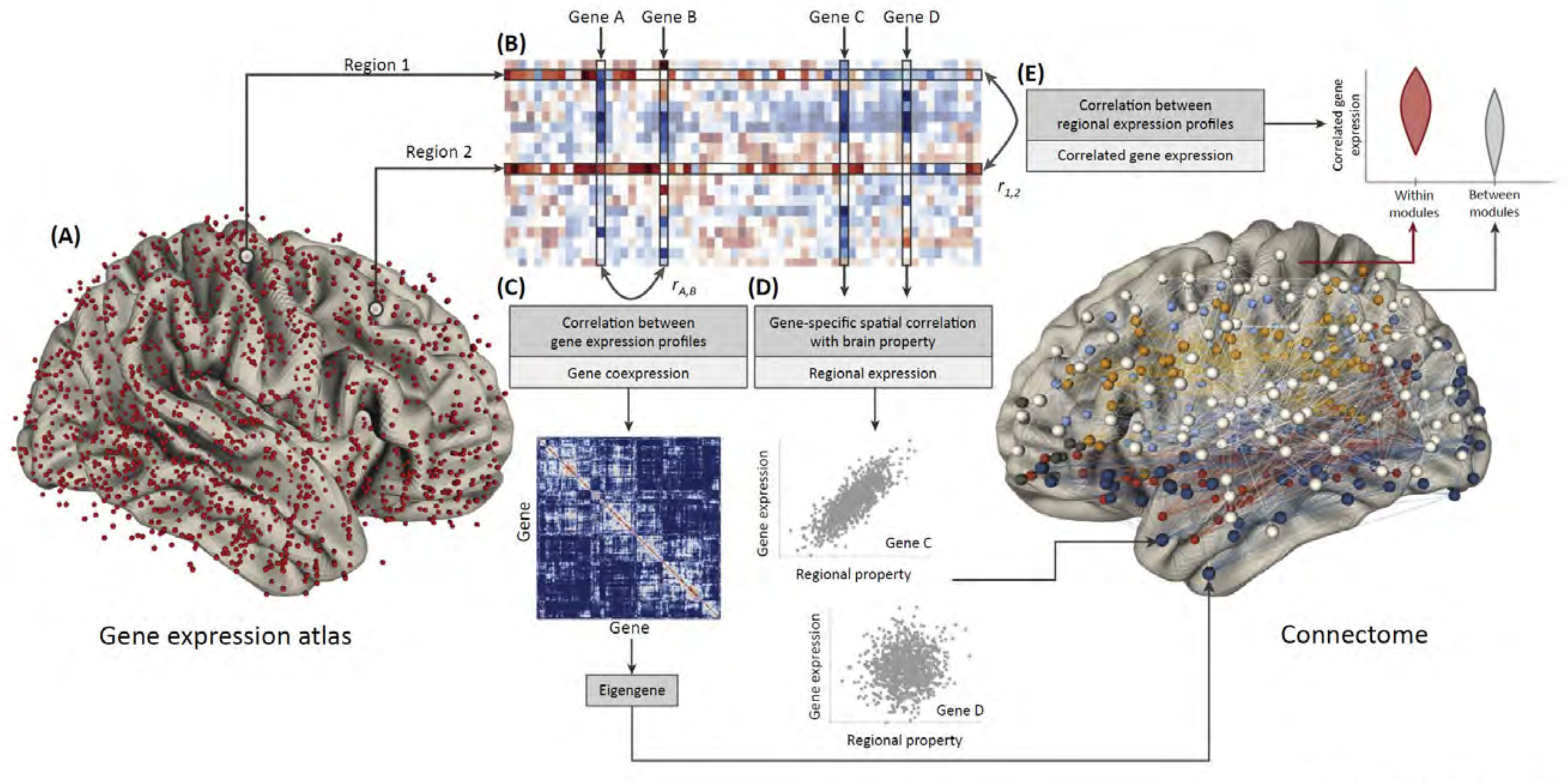
$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \Big\{ \frac{1}{2} \sum_{i=1}^{N} \big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\big)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \Big\}.$$



Starting point: variable selection

What-if:

- It’s just impossible to select variables by hand



Trends in Cognitive Sciences

(Fornito et al., 2019)

Evolving ideas of dimensionality reduction

From variable selection to construction

- ~~Starting point: variable selection~~
- PCA & FA & ICA: intuitions & practical
- PCA & FA & ICA: differences

From linear to nonlinear

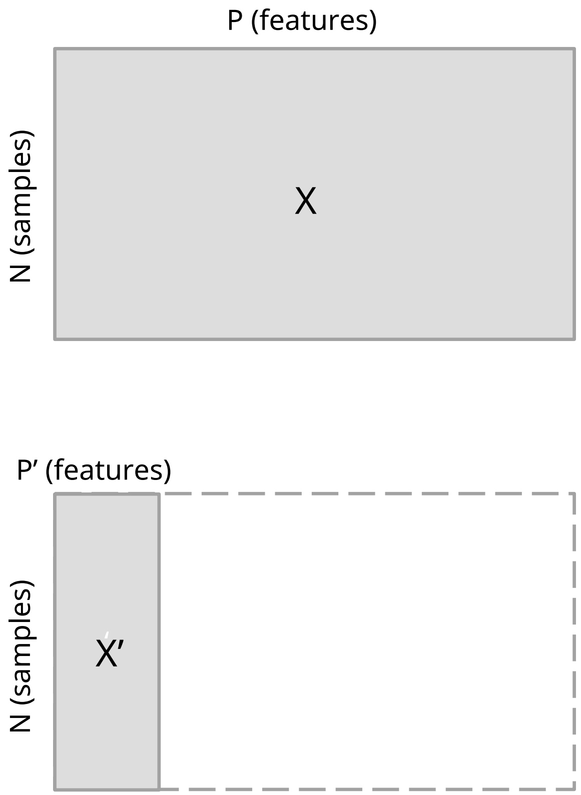
- Diffusion map

From decomposition to approximation

- tSNE & UMAP: intuitions & cautions

From dimensions to categories

Back to the future

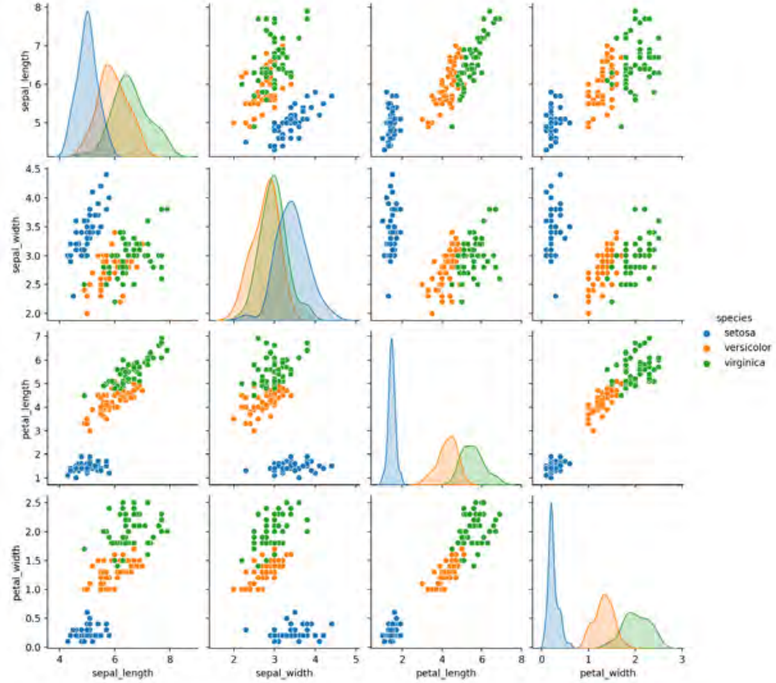


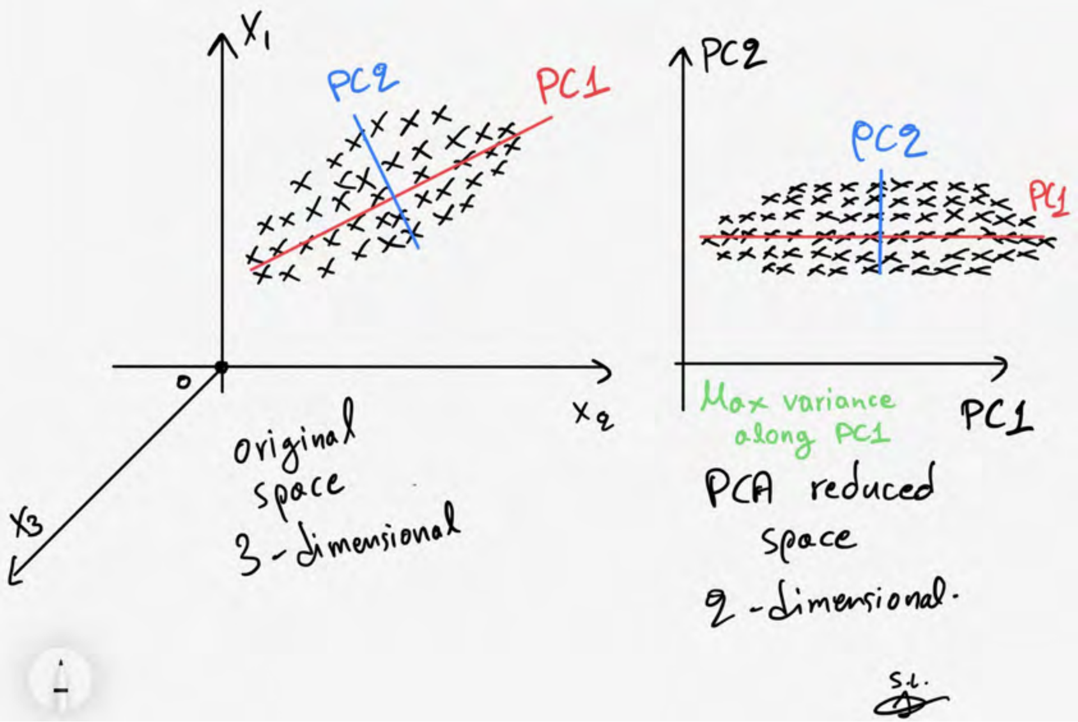
From variable selection to construction

Existing variables (axes) are not enough for pattern-finding

Making a set of “new” directions using linear combination

This is effectively a rotation of axes





X1
PC2
PC1
O
X2
X3
original space 3-dimensional
PC2
PC2
PC1
Max variance along PC1
PC1
PCA reduced space 2-dimensional.

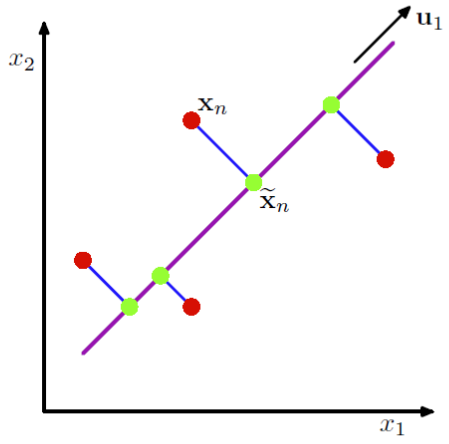
(Serafeim Loukas @ towardsdatascience, 2020)

PCA: intuitions

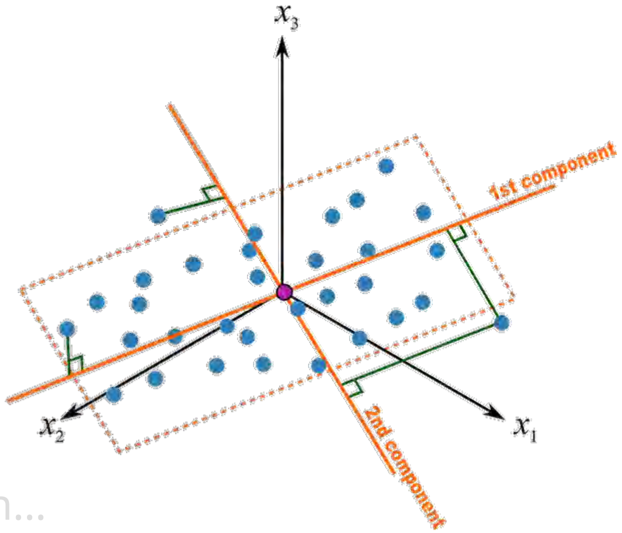
Maximizing variance (or minimizing error)

- Original variables $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots]$ are correlated
- We want a new set of variables $[\boldsymbol{z}_1, \boldsymbol{z}_2, \dots]$ that
  - Capture the most variance
  - Mutually uncorrelated (orthogonal)
- Want to project $\boldsymbol{x}$ to $\boldsymbol{z}$ by a rotation $\boldsymbol{u}$, so $\boldsymbol{z} = \boldsymbol{u}^T\boldsymbol{X}$
- Find $\boldsymbol{u}$, so it would $\max_{\boldsymbol{u}} Var(\boldsymbol{z})\,, s.t.\, \boldsymbol{u}^T\boldsymbol{u} = 1$
- $Var(\boldsymbol{z}) = \frac{1}{N}\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}$, where $\boldsymbol{R} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X}$
- Construct Lagrange multiplier $\mathrm{L} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} - \lambda(\boldsymbol{u}^T\boldsymbol{u} - 1)$
- Let $\frac{\partial L}{\partial \boldsymbol{u}} = 2\boldsymbol{R}\boldsymbol{u} - 2\boldsymbol{u}\lambda = 0$, we have $\boldsymbol{R}\boldsymbol{u} = \lambda\boldsymbol{u}$, $(\boldsymbol{R} - \lambda\boldsymbol{I})\boldsymbol{u} = 0$
- Here λ is the (first) eigenvalue and $\boldsymbol{u}$ is the (first) eigenvector
- The others are calculated by subtracting the first principal components
- Overall, $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}$

We want to find (make) the new variables!
“the linear transform”



How do we find them?
“max variance direction”



After some magic optimization…
“Got them!”

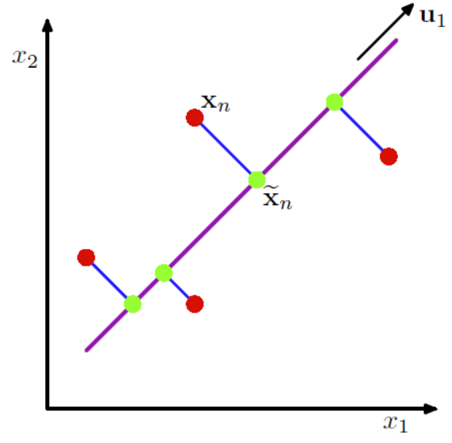
(Dunn, 2021)

PCA: intuitions

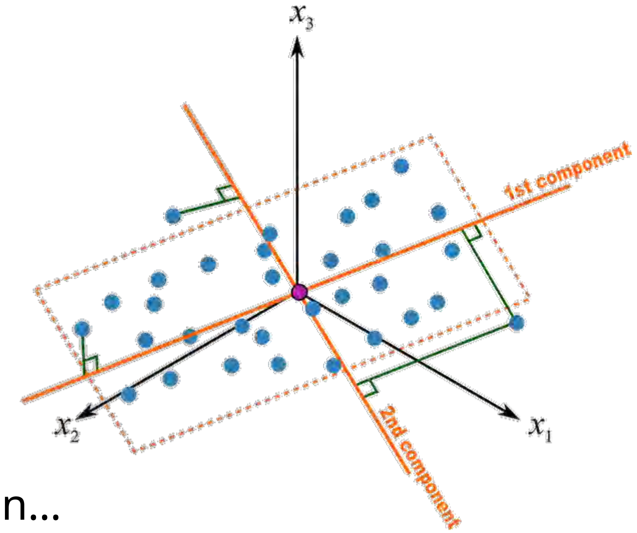
Maximizing variance (or minimizing error)

- Original variables $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \dots]$ are correlated
- We want a new set of variables $[\boldsymbol{z_1}, \boldsymbol{z_2}, \dots]$ that
  - Capture the most variance
  - Mutually uncorrelated (orthogonal)
- Want to project $\boldsymbol{x}$ to $\boldsymbol{z}$ by a rotation $\boldsymbol{u}$, so $\boldsymbol{z} = \boldsymbol{u}^T\boldsymbol{X}$
- Find $\boldsymbol{u}$, so it would $\max_{\boldsymbol{u}} Var(\boldsymbol{z})\,, s.t.\ \boldsymbol{u}^T\boldsymbol{u} = 1$
- $Var(\boldsymbol{z}) = \frac{1}{N}\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}$, where $\boldsymbol{R} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X}$
- Construct Lagrange multiplier $\mathrm{L} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} - \lambda(\boldsymbol{u}^T\boldsymbol{u} - 1)$
- Let $\frac{\partial L}{\partial \boldsymbol{u}} = 2\boldsymbol{R}\boldsymbol{u} - 2\boldsymbol{u}\lambda = 0$, we have $\boldsymbol{R}\boldsymbol{u} = \lambda\boldsymbol{u}$, $(\boldsymbol{R} - \lambda\boldsymbol{I})\boldsymbol{u} = 0$
- Here λ is the (first) eigenvalue and $\boldsymbol{u}$ is the (first) eigenvector
- The others are calculated by subtracting the first principal components
- Overall, $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}$

We want to find (make) the
new variables!
“the linear transform”



How do we find them?
“max variance direction”



After some magic optimization…
“Got them!”

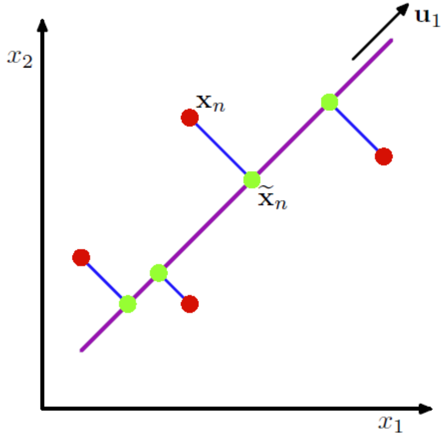
(Dunn, 2021)

PCA: intuitions

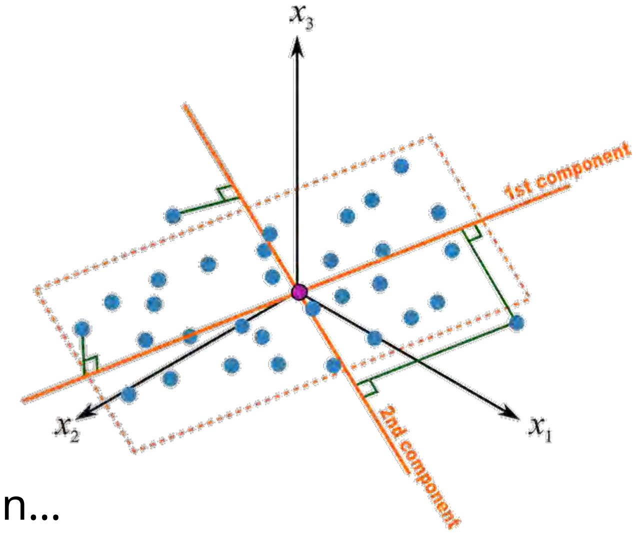
Maximizing variance (or minimizing error)

- Original variables $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots]$ are correlated
- We want a new set of variables $[\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots]$ that
  - Capture the most variance
  - Mutually uncorrelated (orthogonal)
- Want to project $\boldsymbol{x}$ to $\boldsymbol{z}$ by a rotation $\boldsymbol{u}$, so $\boldsymbol{z} = \boldsymbol{u}^T\boldsymbol{X}$
- Find $\boldsymbol{u}$, so it would $\max_{\boldsymbol{u}} Var(\boldsymbol{z})\,, s.t.\, \boldsymbol{u}^T\boldsymbol{u} = 1$
- $Var(\boldsymbol{z}) = \frac{1}{N}\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}$, where $\boldsymbol{R} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X}$
- Construct Lagrange multiplier $\mathrm{L} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} - \lambda(\boldsymbol{u}^T\boldsymbol{u} - 1)$
- Let $\frac{\partial L}{\partial \boldsymbol{u}} = 2\boldsymbol{R}\boldsymbol{u} - 2\boldsymbol{u}\lambda = 0$, we have $\boldsymbol{R}\boldsymbol{u} = \lambda\boldsymbol{u}$, $(\boldsymbol{R} - \lambda\boldsymbol{I})\boldsymbol{u} = 0$
- Here λ is the (first) eigenvalue and $\boldsymbol{u}$ is the (first) eigenvector
- The others are calculated by subtracting the first principal components
- Overall, $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}$

We want to find (make) the
new variables!
“the linear transform”



How do we find them?
“max variance direction”



After some magic optimization…
“Got them!”

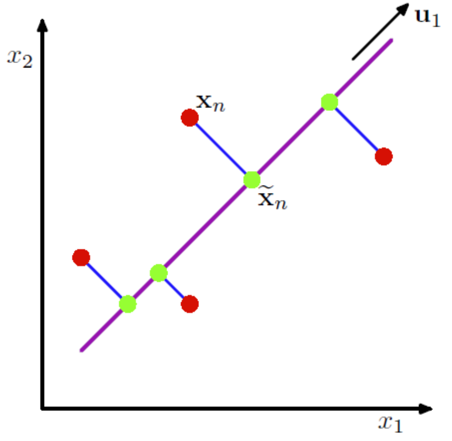
(Dunn, 2021)

PCA: intuitions

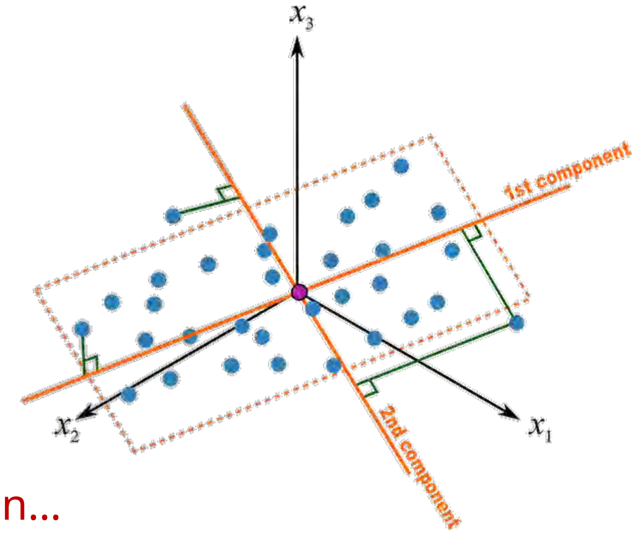
Maximizing variance (or minimizing error)

- Original variables $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots]$ are correlated
- We want a new set of variables $[\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots]$ that
  - Capture the most variance
  - Mutually uncorrelated (orthogonal)
- Want to project $\boldsymbol{x}$ to $\boldsymbol{z}$ by a rotation $\boldsymbol{u}$, so $\boldsymbol{z} = \boldsymbol{u}^T\boldsymbol{X}$
- Find $\boldsymbol{u}$, so it would $\max\limits_{\boldsymbol{u}} Var(\boldsymbol{z}), s.t.\, \boldsymbol{u}^T\boldsymbol{u} = 1$
- $Var(\boldsymbol{z}) = \frac{1}{N}\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}$, where $\boldsymbol{R} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X}$
- Construct Lagrange multiplier $\mathrm{L} = \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} - \lambda(\boldsymbol{u}^T\boldsymbol{u} - 1)$
- Let $\frac{\partial L}{\partial \boldsymbol{u}} = 2\boldsymbol{R}\boldsymbol{u} - 2\boldsymbol{u}\lambda = 0$, we have $\boldsymbol{R}\boldsymbol{u} = \lambda\boldsymbol{u}$, $(\boldsymbol{R} - \lambda\boldsymbol{I})\boldsymbol{u} = 0$
- Here λ is the (first) eigenvalue and $\boldsymbol{u}$ is the (first) eigenvector
- The others are calculated by subtracting the first principal components
- Overall, $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}$

We want to find (make) the
new variables!
“the linear transform”



How do we find them?
“max variance direction”



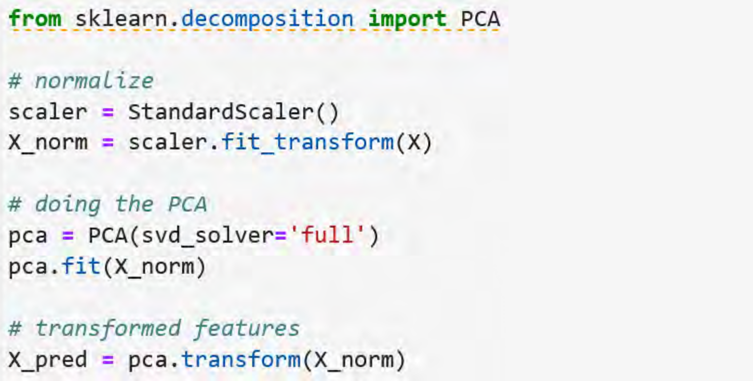
After some magic optimization…
“Got them!”

(Dunn, 2021)

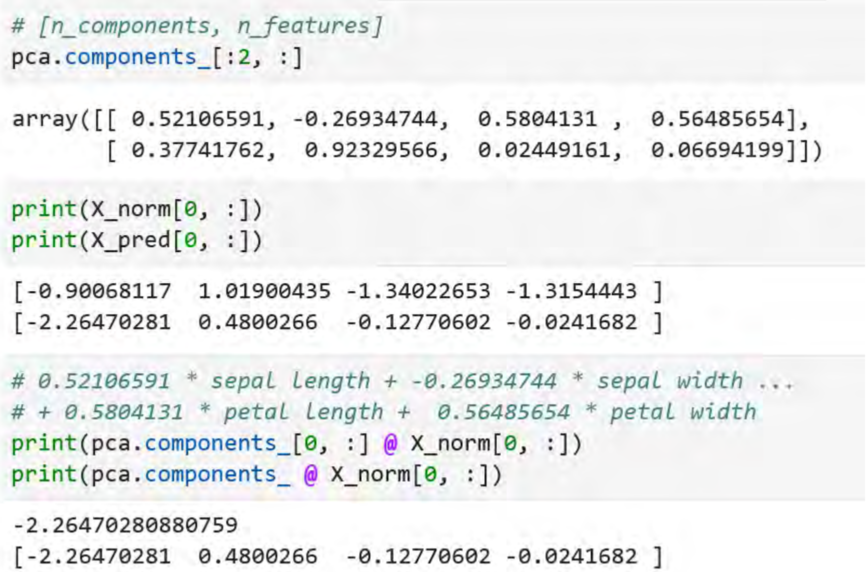
PCA: practical

Doing the PCA

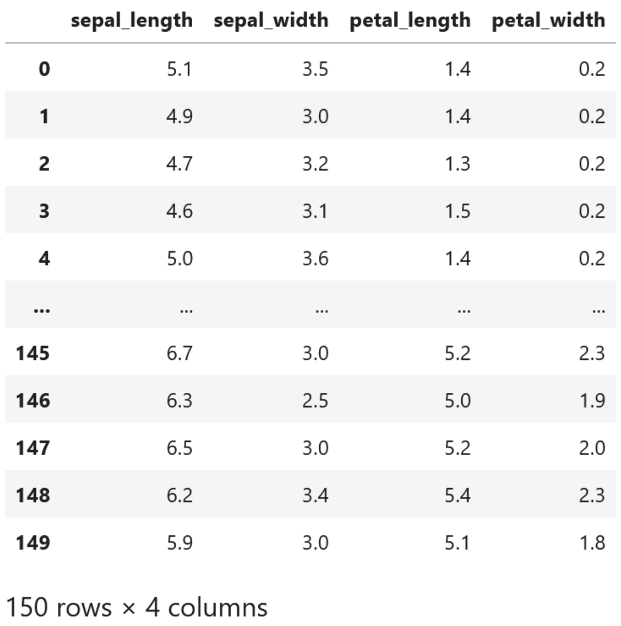
- What do we use and what we will get



```
from sklearn.decomposition import PCA

# normalize
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

# doing the PCA
pca = PCA(svd_solver='full')
pca.fit(X_norm)

# transformed features
X_pred = pca.transform(X_norm)
```

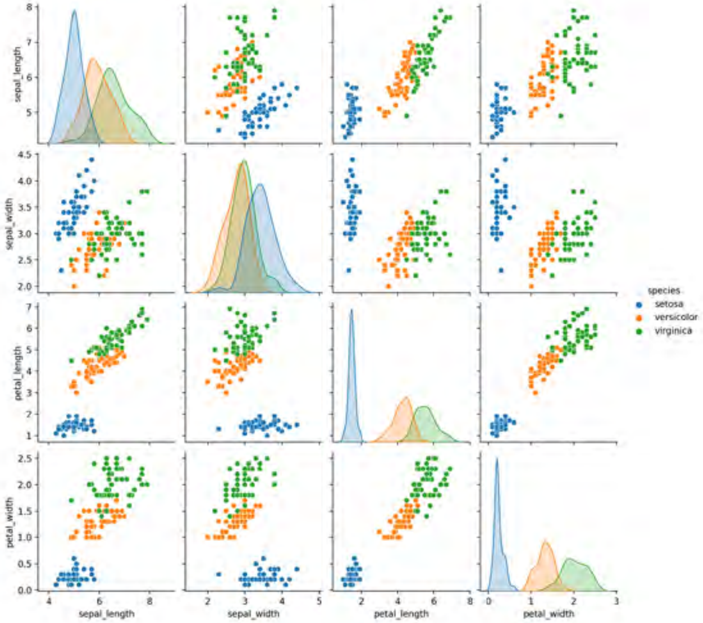


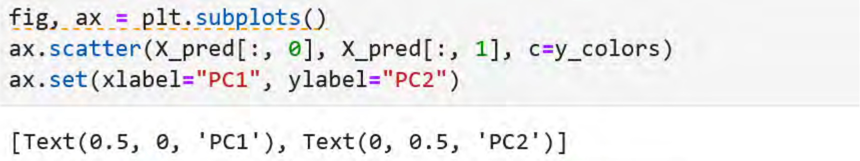
```
# [n_components, n_features]
pca.components_[:2, :]

array([[ 0.52106591, -0.26934744,  0.5804131 ,  0.56485654],
       [ 0.37741762,  0.92329566,  0.02449161,  0.06694199]])

print(X_norm[0, :])
print(X_pred[0, :])

[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]

# 0.52106591 * sepal length + -0.26934744 * sepal width ...
# + 0.5804131 * petal length +  0.56485654 * petal width
print(pca.components_[0, :] @ X_norm[0, :])
print(pca.components_ @ X_norm[0, :])

-2.26470280880759
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]
```



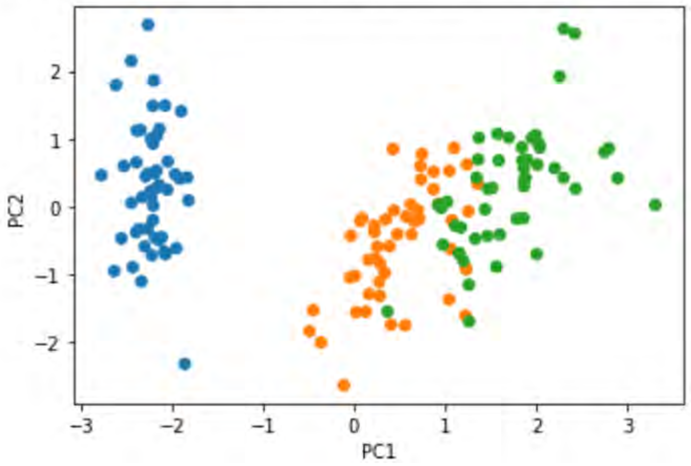
	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns





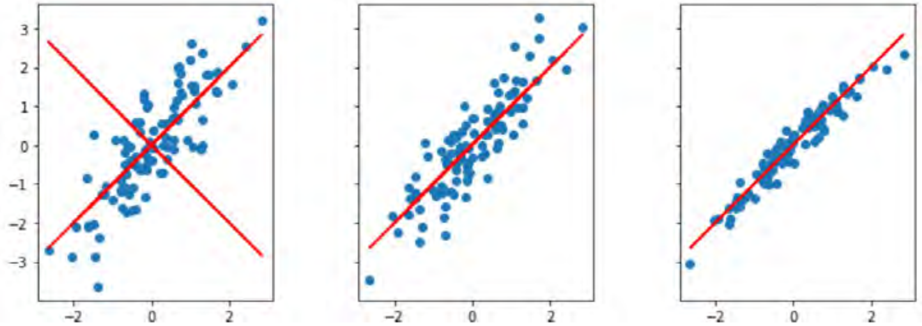
```
fig, ax = plt.subplots()
ax.scatter(X_pred[:, 0], X_pred[:, 1], c=y_colors)
ax.set(xlabel="PC1", ylabel="PC2")

[Text(0.5, 0, 'PC1'), Text(0, 0.5, 'PC2')]
```

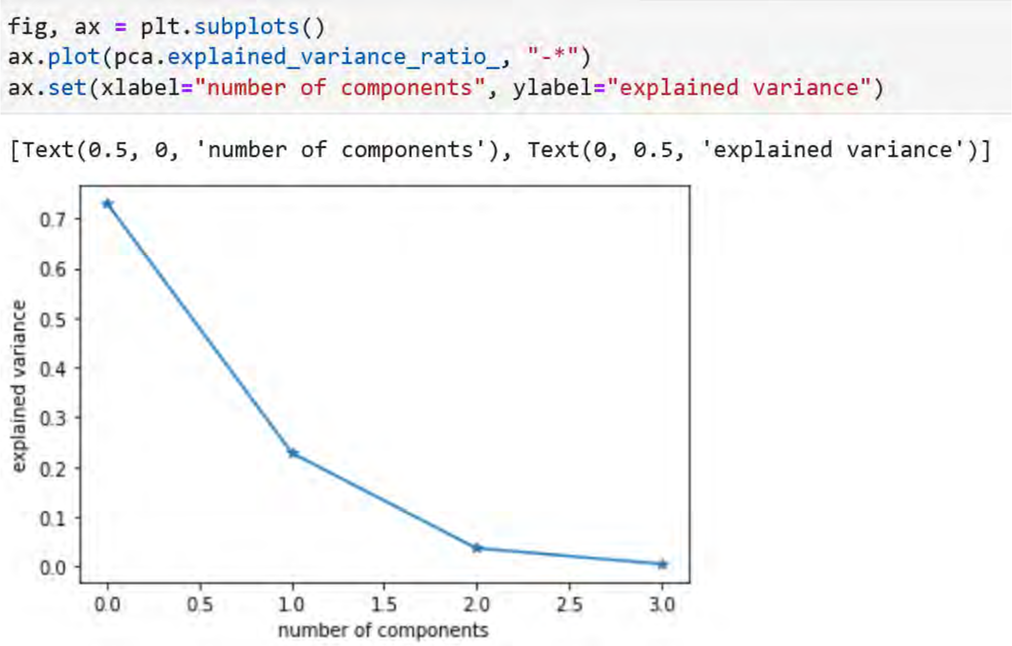


PCA: practical

Doing the PCA

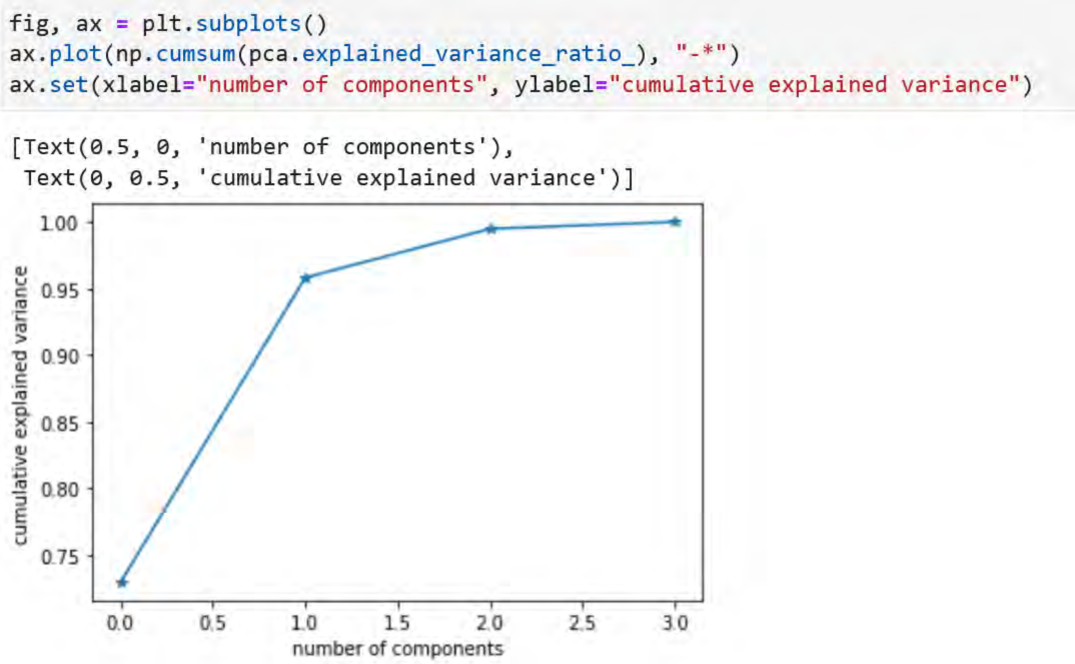
- How many components to keep
- “Scree plot”
- Find the “elbow”





```
fig, ax = plt.subplots()
ax.plot(pca.explained_variance_ratio_, "-*")
ax.set(xlabel="number of components", ylabel="explained variance")
```

```
[Text(0.5, 0, 'number of components'), Text(0, 0.5, 'explained variance')]
```



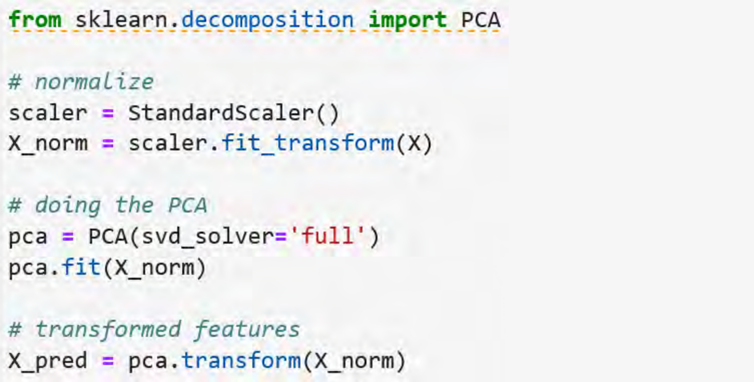
```
fig, ax = plt.subplots()
ax.plot(np.cumsum(pca.explained_variance_ratio_), "-*")
ax.set(xlabel="number of components", ylabel="cumulative explained variance")
```

```
[Text(0.5, 0, 'number of components'),
 Text(0, 0.5, 'cumulative explained variance')]
```

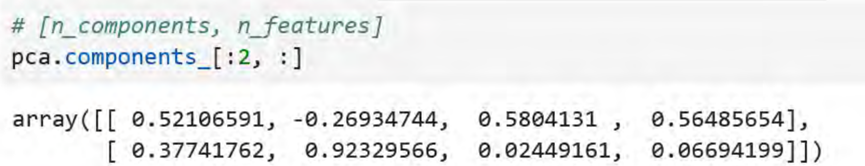
PCA: practical

Doing the PCA

- Corresponding to SVD output

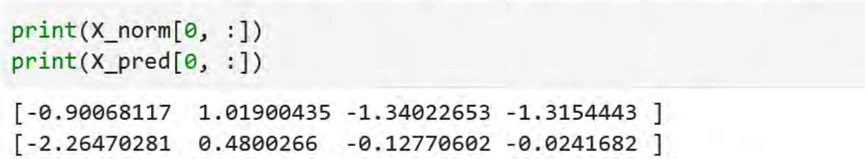


```
from sklearn.decomposition import PCA

# normalize
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

# doing the PCA
pca = PCA(svd_solver='full')
pca.fit(X_norm)

# transformed features
X_pred = pca.transform(X_norm)
```



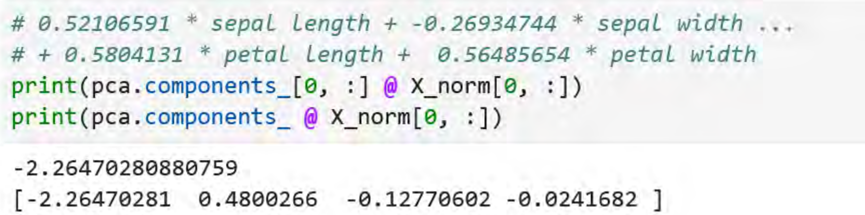
```
# [n_components, n_features]
pca.components_[:2, :]
```

```
array([[ 0.52106591, -0.26934744,  0.5804131 ,  0.56485654],
       [ 0.37741762,  0.92329566,  0.02449161,  0.06694199]])
```



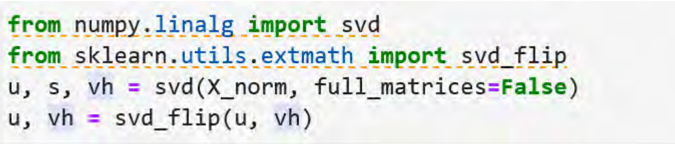
```
print(X_norm[0, :])
print(X_pred[0, :])
```

```
[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]
```



```
# 0.52106591 * sepal length + -0.26934744 * sepal width ...
# + 0.5804131 * petal length +  0.56485654 * petal width
print(pca.components_[0, :] @ X_norm[0, :])
print(pca.components_ @ X_norm[0, :])
```

```
-2.26470280880759
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]
```



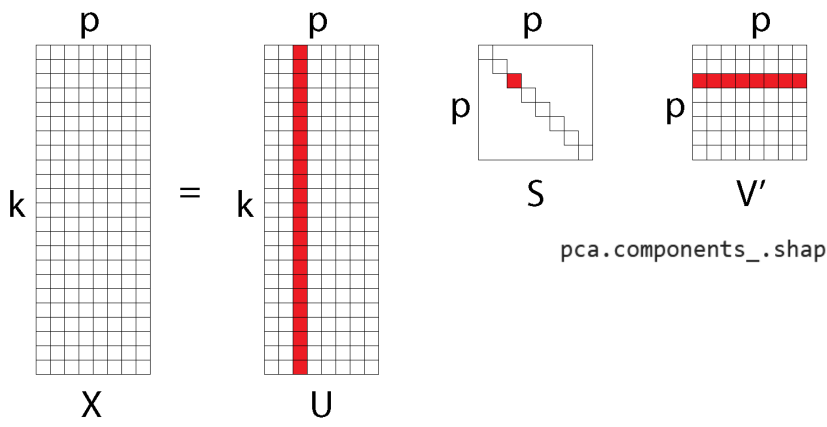
```
from numpy.linalg import svd
from sklearn.utils.extmath import svd_flip
u, s, vh = svd(X_norm, full_matrices=False)
u, vh = svd_flip(u, vh)
```

$$SVD(X) = USV^T$$

$$\boldsymbol{Z = XV = USV^T V = US}$$

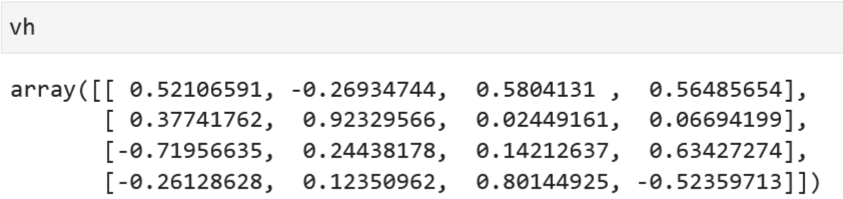
`pca.singular_values_.shape = (4,)`
`pca.explained_variance_.shape = (4,)`

`X_norm.shape = (150, 4)`



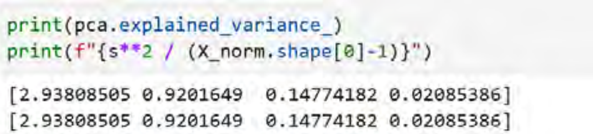
`pca.components_.shape = (4, 4)`

`X_pred.shape = (150, 4)`



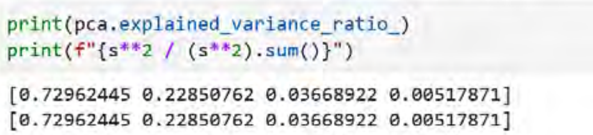
```
vh
```

```
array([[ 0.52106591, -0.26934744,  0.5804131 ,  0.56485654],
       [ 0.37741762,  0.92329566,  0.02449161,  0.06694199],
       [-0.71956635,  0.24438178,  0.14212637,  0.63427274],
       [-0.26128628,  0.12350962,  0.80144925, -0.52359713]])
```



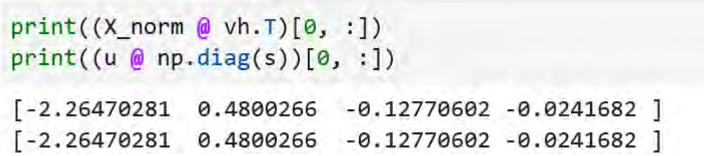
```
print(pca.explained_variance_)
print(f"{s**2 / (X_norm.shape[0]-1)}")
```

```
[2.93808505 0.9201649  0.14774182 0.02085386]
[2.93808505 0.9201649  0.14774182 0.02085386]
```



```
print(pca.explained_variance_ratio_)
print(f"{s**2 / (s**2).sum()}")
```

```
[0.72962445 0.22850762 0.03668922 0.00517871]
[0.72962445 0.22850762 0.03668922 0.00517871]
```



```
print((X_norm @ vh.T)[0, :])
print((u @ np.diag(s))[0, :])
```

```
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]
```

PCA: intuitions

PCA through SVD: a way for calculation

Singular value decomposition $SVD(X) = USV^T$

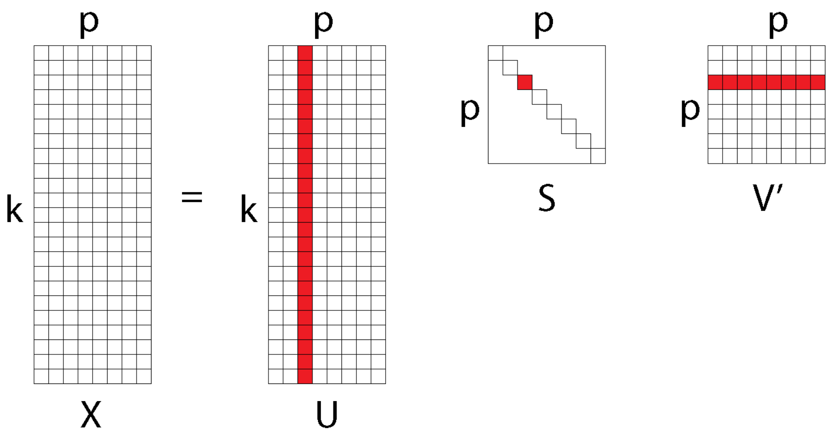
Eigen decomposition $EIG(A) = Q\Lambda Q^{-1}$, for real symmetric, $Q^{-1} = Q^T$

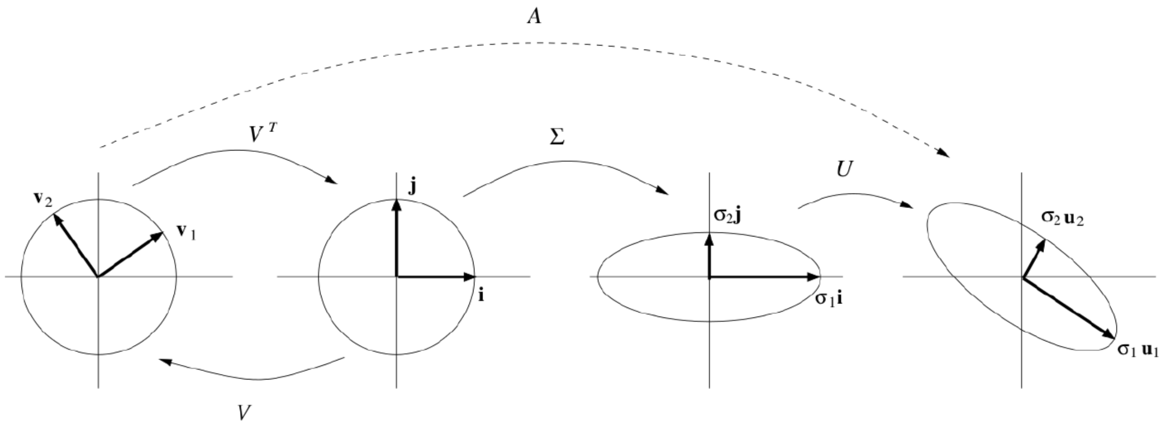
$$X^TX = (VS^TU^T)(USV^T) = VS^T(U^TU)SV^T = V(S^TS)V^T = V\Lambda V^T$$

$$XX^T = (USV^T)(VS^TU^T) = US(V^TV)S^TU^T = U(SS^T)U^T = U\Lambda U^T$$

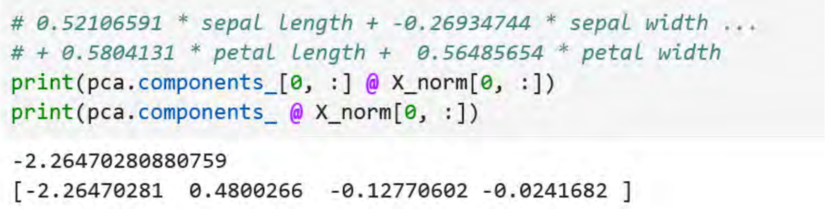
$$EIG(X^TX) = V\Lambda V^T, EIG(XX^T) = U\Lambda U^T$$

$$\boldsymbol{Z = XV = USV^TV = US}$$

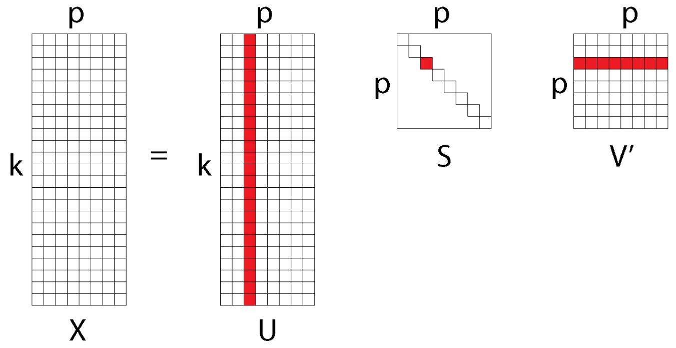




PCA: practical



```
# 0.52106591 * sepal length + -0.26934744 * sepal width ...
# + 0.5804131 * petal length +  0.56485654 * petal width
print(pca.components_[0, :] @ X_norm[0, :])
print(pca.components_ @ X_norm[0, :])

-2.26470280880759
[-2.26470281  0.4800266  -0.12770602 -0.0241682 ]
```

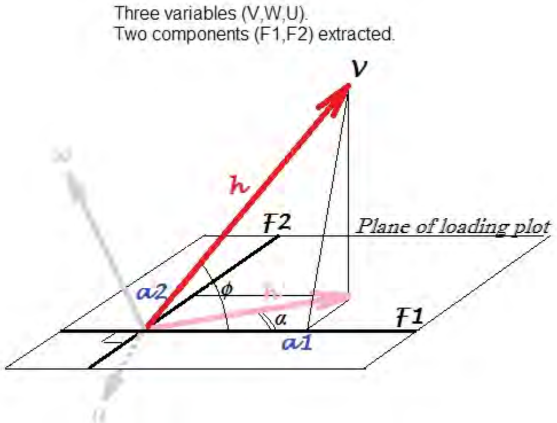


Analyzing PCA results

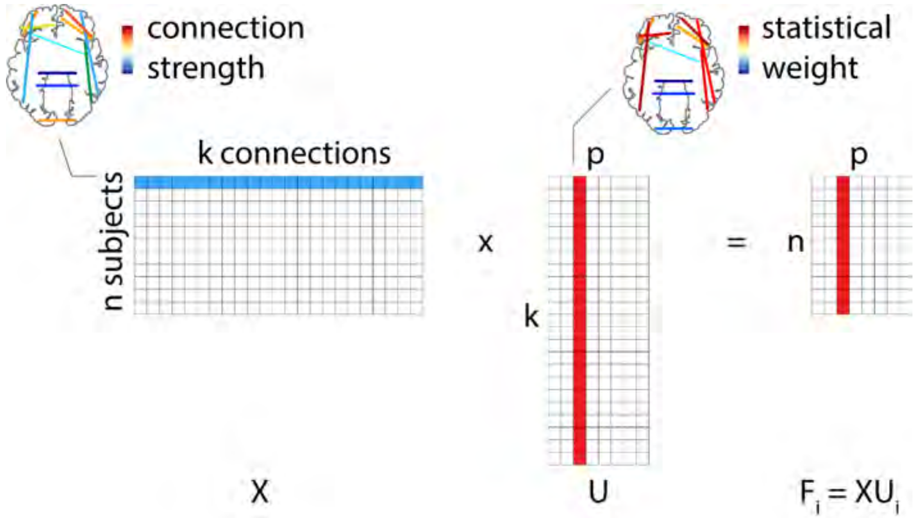
- Weights, scores and loadings
- Weights: how much each variable contributes to the pattern
- $SVD(\boldsymbol{X}) = \boldsymbol{USV}^T$ $\quad\quad$ $\boldsymbol{Z} = \boldsymbol{XV} = \boldsymbol{USV}^T\boldsymbol{V} = \boldsymbol{US}$

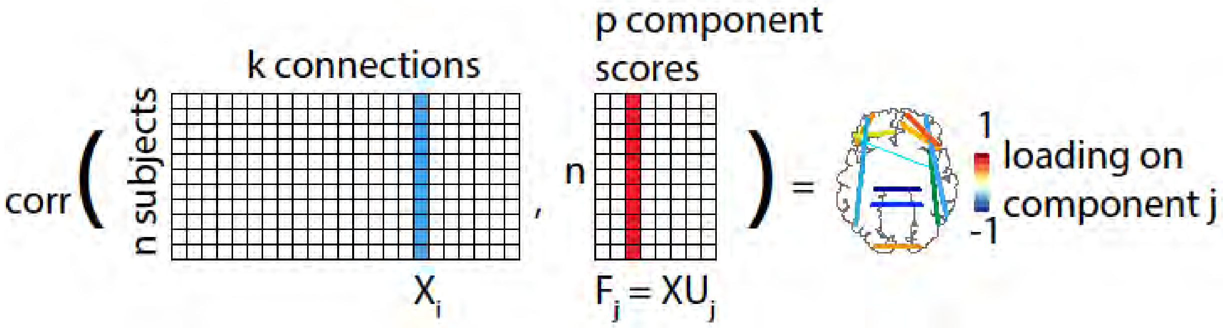
$$\mathbf{X} = \mathbf{US}\cdot\mathbf{V}^\top = \text{Scores}\cdot\text{Principal directions.}$$

$$\mathbf{X} = \sqrt{n-1}\mathbf{U}\cdot(\mathbf{VS}/\sqrt{n-1})^\top = \mathbf{U}\cdot\mathbf{L}^\top = \text{Standardized scores}\cdot\text{Loadings.}$$



- Project individual participants
  - Scores: how do data express the overall pattern, projecting data onto the latent variables
  - Loadings: correlations between the original variables and the expression of latent variable





PCA: bonus

PCA may be implemented by brain plasticity

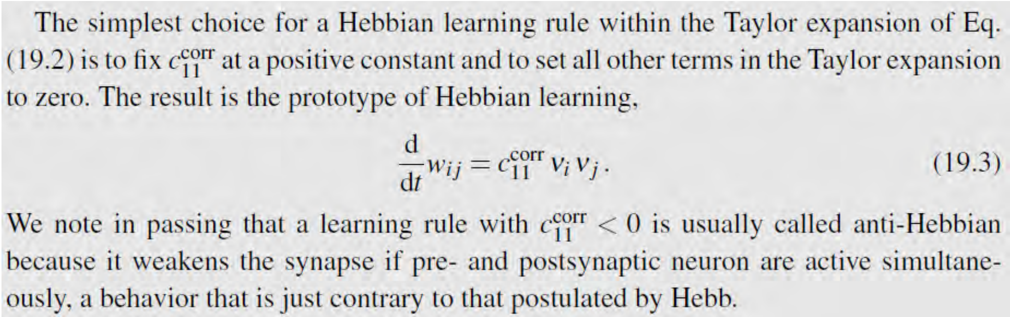
Hebbian learning

- “fire together, wire together”
- Finds first principal component, but saturates

Oja’s learning rule

- “add forgets” (proportional to weight and output squared)

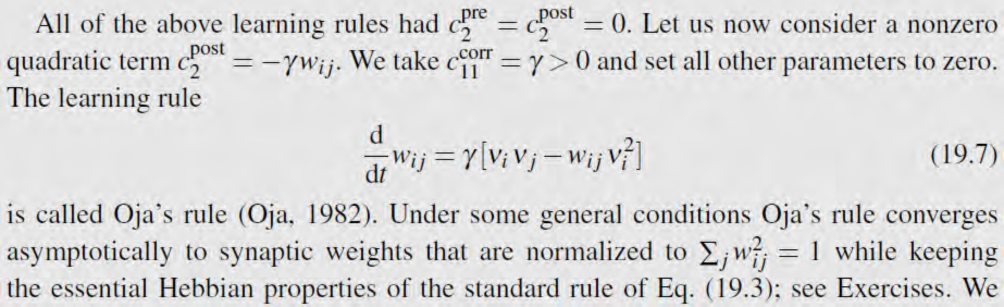
Other rules also exist & ICA can be derived



The simplest choice for a Hebbian learning rule within the Taylor expansion of Eq. (19.2) is to fix c_{11}^{corr} at a positive constant and to set all other terms in the Taylor expansion to zero. The result is the prototype of Hebbian learning,

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = c_{11}^{\text{corr}}\, \nu_i\, \nu_j\,. \qquad (19.3)$$

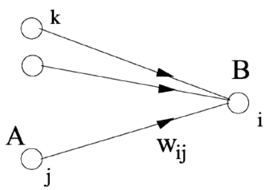
We note in passing that a learning rule with $c_{11}^{\text{corr}} < 0$ is usually called anti-Hebbian because it weakens the synapse if pre- and postsynaptic neuron are active simultaneously, a behavior that is just contrary to that postulated by Hebb.



All of the above learning rules had $c_2^{\text{pre}} = c_2^{\text{post}} = 0$. Let us now consider a nonzero quadratic term $c_2^{\text{post}} = -\gamma w_{ij}$. We take $c_{11}^{\text{corr}} = \gamma > 0$ and set all other parameters to zero. The learning rule

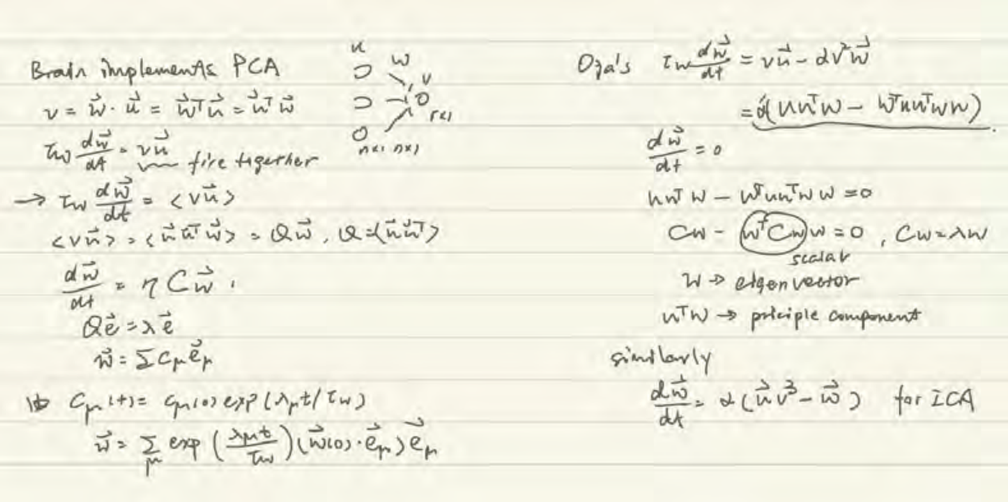
$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = \gamma\,[\nu_i\, \nu_j - w_{ij}\, \nu_i^2] \qquad (19.7)$$

is called Oja’s rule (Oja, 1982). Under some general conditions Oja’s rule converges asymptotically to synaptic weights that are normalized to $\sum_j w_{ij}^2 = 1$ while keeping the essential Hebbian properties of the standard rule of Eq. (19.3); see Exercises. We



$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = F(w_{ij}; \nu_i, \nu_j)\,.$$

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = c_0(w_{ij}) + c_1^{\text{pre}}(w_{ij})\,\nu_j + c_1^{\text{post}}(w_{ij})\nu_i + c_2^{\text{pre}}(w_{ij})\,\nu_j^2 + c_2^{\text{post}}(w_{ij})\,\nu_i^2 + c_{11}^{\text{corr}}(w_{ij})\,\nu_i\,\nu_j + \mathcal{O}(\nu^3)\,.$$



Brain implements PCA

$v = \vec{w}\cdot\vec{u} = \vec{w}^T\vec{u} = \vec{u}^T\vec{w}$

$\tau_w \frac{d\vec{w}}{dt} = v\vec{u}$ fire together

→ $\tau_w \frac{d\vec{w}}{dt} = \langle v\vec{u}\rangle$

$\langle v\vec{u}\rangle = \langle \vec{u}\vec{u}^T\vec{w}\rangle = Q\vec{w}$, $Q = \langle\vec{u}\vec{u}^T\rangle$

$\frac{d\vec{w}}{dt} = \eta C\vec{w}$

$Q\vec{e} = \lambda\vec{e}$

$\vec{w} = \sum c_\mu \vec{e}_\mu$

$c_\mu(t) = c_\mu(0)\exp(\lambda_\mu t/\tau_w)$

$\vec{w} = \sum_\mu \exp\left(\frac{\lambda_\mu t}{\tau_w}\right)(\vec{w}(0)\cdot\vec{e}_\mu)\vec{e}_\mu$

Oja's $\tau_w \frac{d\vec{w}}{dt} = v\vec{u} - \alpha v^2\vec{w}$

$= \underline{\alpha(uu^Tw - w^Tuu^Tww)}$

$\frac{d\vec{w}}{dt} = 0$

$uu^Tw - w^Tuu^Tww = 0$

$Cw - \underbrace{\boxed{w^TCw}}_{\text{scalar}}w = 0$, $Cw = \lambda w$

w → eigenvector

w^Tw → priciple component

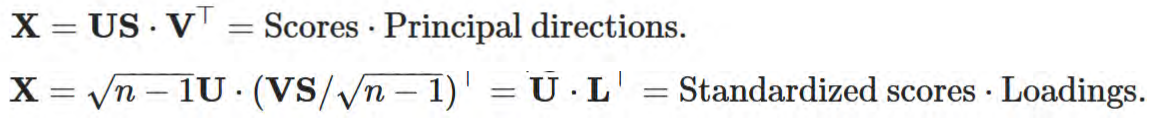
similarly

$\frac{d\vec{w}}{dt} = \alpha(\vec{u}v^3 - \vec{w})$ for ICA

Dayan and Abbott (2001), *Theoretical Neuroscience*
Gerstner et al (2004), *Neuronal Dynamics*
Hastie et al (2009), *The Elements of Statistical Learning*

(http://www.scholarpedia.org/article/Oja_learning_rule)

FA: intuitions

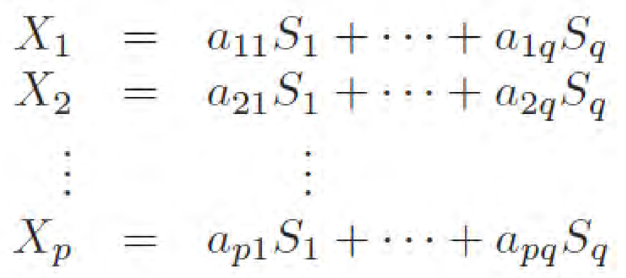


$$\mathbf{X} = \mathbf{US}\cdot\mathbf{V}^\top = \text{Scores}\cdot\text{Principal directions.}$$

$$\mathbf{X} = \sqrt{n-1}\mathbf{U}\cdot(\mathbf{VS}/\sqrt{n-1})^\top = \tilde{\mathbf{U}}\cdot\mathbf{L}^\top = \text{Standardized scores}\cdot\text{Loadings.}$$

Factor analysis (FA)

- From $\mathbf{X} = \boldsymbol{USV}^T$, let $\boldsymbol{S} = \sqrt{N}\boldsymbol{U}$ and $\boldsymbol{A}^T = \boldsymbol{SV}^T/\sqrt{N}$
- We have $\mathbf{X} = \boldsymbol{SA}^T$, which is a latent variable model.
- To make it more constrained, $X = \boldsymbol{AS} + \varepsilon$
- ε_i are uncorrelated zero-mean disturbances
- Covariance matrix $\Sigma = \boldsymbol{AA}^T + diag(Var(\varepsilon_i))$
- Calculation
  - PCA approach
  - Maximum likelihood approach



$$\begin{aligned} X_1 &= a_{11}S_1 + \cdots + a_{1q}S_q \\ X_2 &= a_{21}S_1 + \cdots + a_{2q}S_q \\ &\vdots \\ X_p &= a_{p1}S_1 + \cdots + a_{pq}S_q \end{aligned}$$

Won’t we find infinite ways to do the rotation?
“thus, infinite latent variables”

We need only one
“fit a model with error”

We are not just finding max variance
“we have factor assumptions in head”

FA: intuitions

$$\mathbf{X} = \mathbf{US} \cdot \mathbf{V}^\top = \text{Scores} \cdot \text{Principal directions.}$$

$$\mathbf{X} = \sqrt{n-1}\mathbf{U} \cdot (\mathbf{VS}/\sqrt{n-1})^\top = \tilde{\mathbf{U}} \cdot \mathbf{L}^\top = \text{Standardized scores} \cdot \text{Loadings.}$$

Factor analysis (FA)

- From $\mathbf{X} = \boldsymbol{USV}^T$, let $\boldsymbol{S} = \sqrt{N}\boldsymbol{U}$ and $\boldsymbol{A}^T = \boldsymbol{SV}^T/\sqrt{N}$
- We have $\mathbf{X} = \boldsymbol{SA}^T$, which is a latent variable model.
- To make it more constrained, $\mathrm{X} = \boldsymbol{A}S + \varepsilon$
- ε_i are uncorrelated zero-mean disturbances
- Covariance matrix $\Sigma = \boldsymbol{AA}^T + diag(Var(\varepsilon_i))$
- Calculation
  - PCA approach
  - Maximum likelihood approach

$$\begin{aligned}
X_1 &= a_{11}S_1 + \cdots + a_{1q}S_q + \varepsilon_1 \\
X_2 &= a_{21}S_1 + \cdots + a_{2q}S_q + \varepsilon_2 \\
\vdots & \qquad\quad \vdots \\
X_p &= a_{p1}S_1 + \cdots + a_{pq}S_q + \varepsilon_p,
\end{aligned}$$

Won’t we find infinite ways to do the rotation?
“thus, infinite latent variables”

We need only one
“fit a model with error”

We are not just finding max variance
“we have factor assumptions in head”

FA: intuitions

$$\mathbf{X} = \mathbf{U}\mathbf{S} \cdot \mathbf{V}^\top = \text{Scores} \cdot \text{Principal directions.}$$

$$\mathbf{X} = \sqrt{n-1}\mathbf{U} \cdot (\mathbf{V}\mathbf{S}/\sqrt{n-1})^\top = \tilde{\mathbf{U}} \cdot \mathbf{L}^\top = \text{Standardized scores} \cdot \text{Loadings.}$$

Factor analysis (FA)

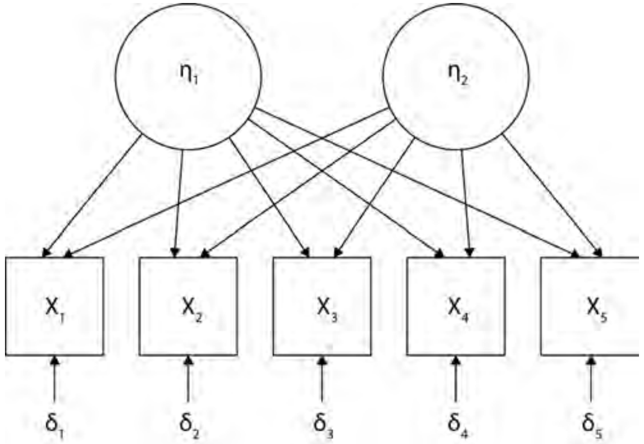
- From $\mathbf{X} = \boldsymbol{USV}^T$, let $\boldsymbol{S} = \sqrt{N}\boldsymbol{U}$ and $\boldsymbol{A}^T = \boldsymbol{SV}^T/\sqrt{N}$
- We have $\mathbf{X} = \boldsymbol{SA}^T$, which is a latent variable model.
- To make it more constrained, $\mathrm{X} = \boldsymbol{AS} + \varepsilon$
- ε_i are uncorrelated zero-mean disturbances
- Covariance matrix $\Sigma = \boldsymbol{AA}^T + diag(Var(\varepsilon_i))$
- Calculation
  - PCA approach
  - Maximum likelihood approach

Won't we find infinite ways to do the rotation?
"thus, infinite latent variables"

We need only one
"fit a model with error"

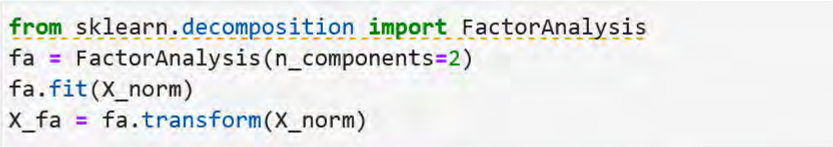
We are not just finding max variance
"we have factor assumptions in head"

$$\begin{aligned}
X_1 &= a_{11}S_1 + \cdots + a_{1q}S_q + \varepsilon_1\\
X_2 &= a_{21}S_1 + \cdots + a_{2q}S_q + \varepsilon_2\\
&\vdots\\
X_p &= a_{p1}S_1 + \cdots + a_{pq}S_q + \varepsilon_p,
\end{aligned}$$

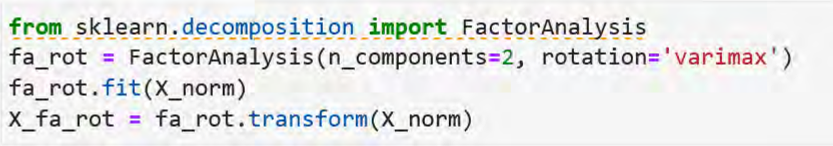


FA: practical

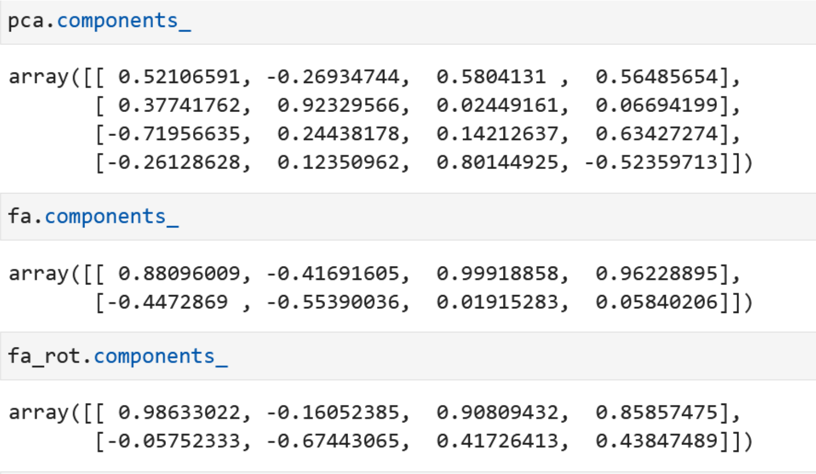
Doing the FA



```
from sklearn.decomposition import FactorAnalysis
fa = FactorAnalysis(n_components=2)
fa.fit(X_norm)
X_fa = fa.transform(X_norm)
```



```
from sklearn.decomposition import FactorAnalysis
fa_rot = FactorAnalysis(n_components=2, rotation='varimax')
fa_rot.fit(X_norm)
X_fa_rot = fa_rot.transform(X_norm)
```



```
pca.components_
```

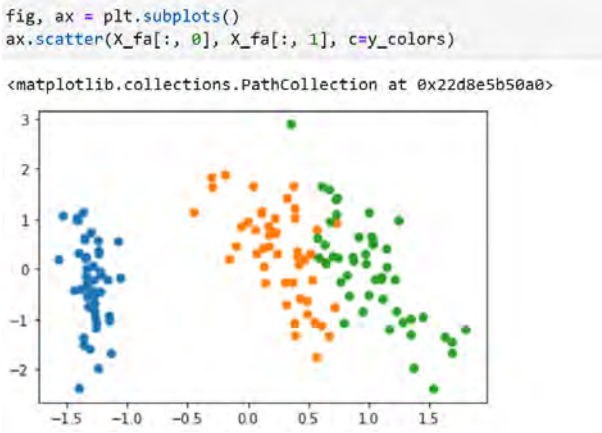
```
array([[ 0.52106591, -0.26934744,  0.5804131 ,  0.56485654],
       [ 0.37741762,  0.92329566,  0.02449161,  0.06694199],
       [-0.71956635,  0.24438178,  0.14212637,  0.63427274],
       [-0.26128628,  0.12350962,  0.80144925, -0.52359713]])
```

```
fa.components_
```

```
array([[ 0.88096009, -0.41691605,  0.99918858,  0.96228895],
       [-0.4472869 , -0.55390036,  0.01915283,  0.05840206]])
```

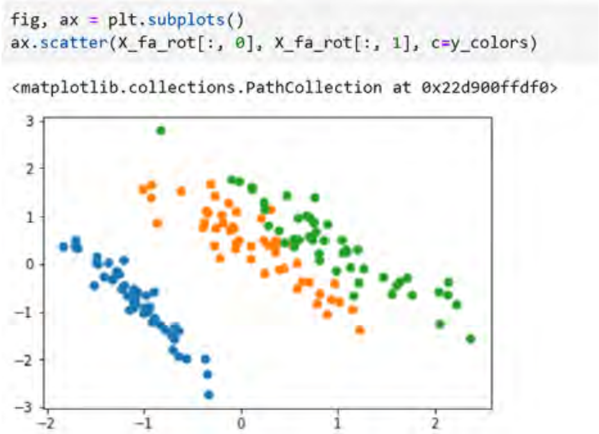
```
fa_rot.components_
```

```
array([[ 0.98633022, -0.16052385,  0.90809432,  0.85857475],
       [-0.05752333, -0.67443065,  0.41726413,  0.43847489]])
```



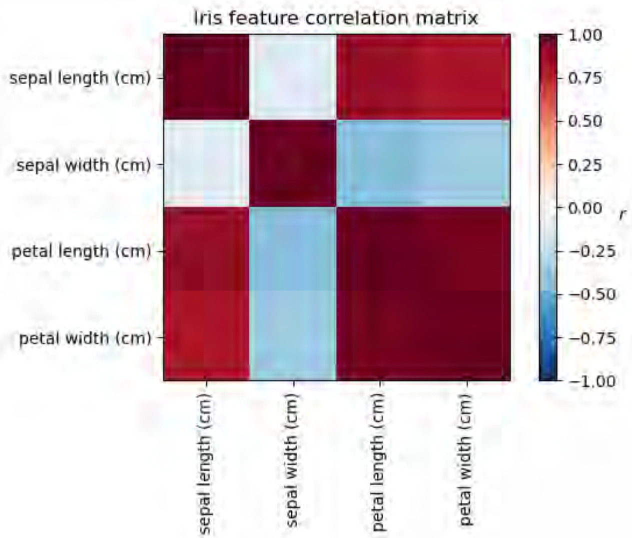
```
fig, ax = plt.subplots()
ax.scatter(X_fa[:, 0], X_fa[:, 1], c=y_colors)
```

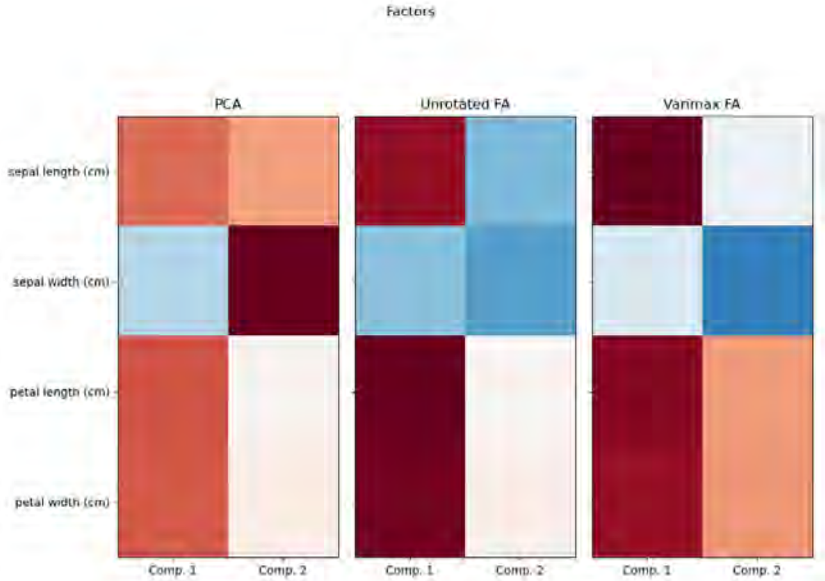
```
<matplotlib.collections.PathCollection at 0x22d8e5b50a0>
```



```
fig, ax = plt.subplots()
ax.scatter(X_fa_rot[:, 0], X_fa_rot[:, 1], c=y_colors)
```

```
<matplotlib.collections.PathCollection at 0x22d900ffdf0>
```

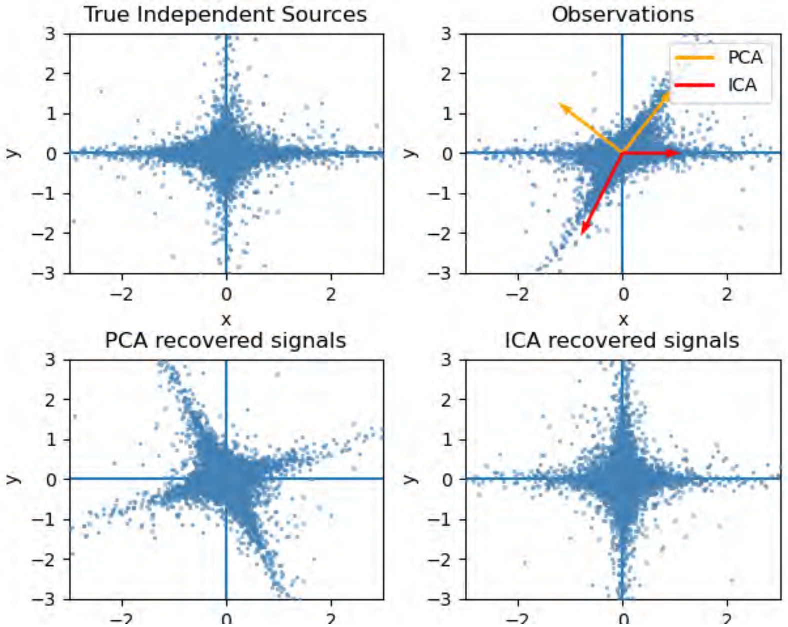




ICA: intuitions

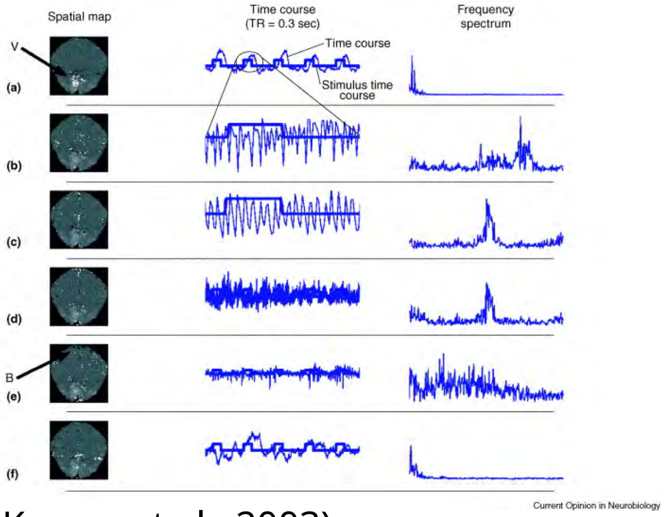
Independent component analysis (ICA)

- Finding an orthogonal $\boldsymbol{A}$ that components of $S = \boldsymbol{A}^T X$ are independent
- Any linear mixture of independent variables will be more Gaussian than original
- Create new axes that maximize non-Gaussian projections
- Many algorithms (underlying objective functions)

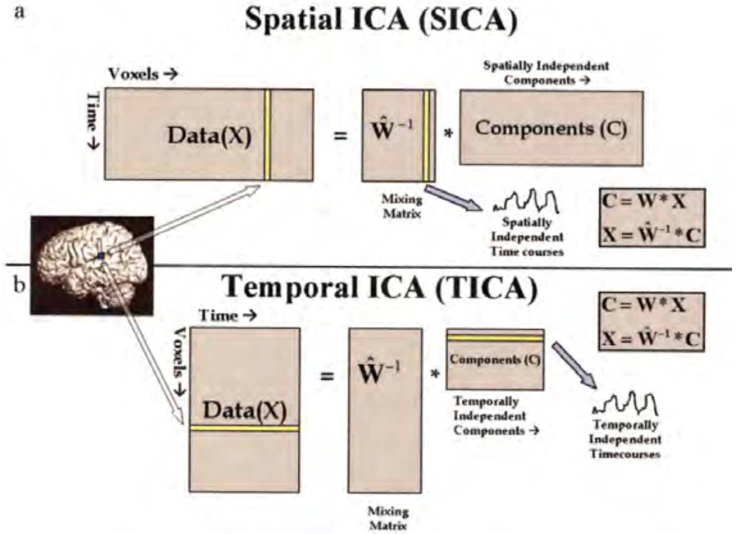


Blind source separation (the cocktail party problem)

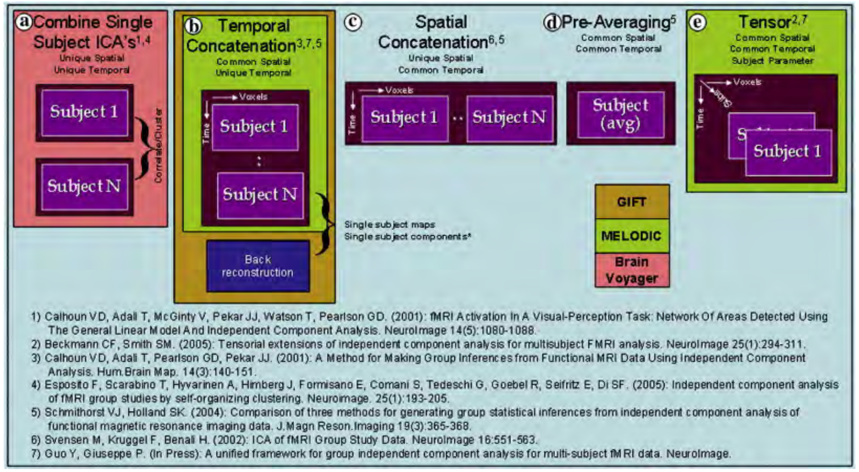
- Denoising + discovery



(McKeown et al., 2003)



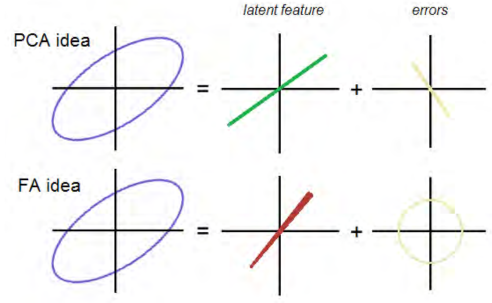
(Calhoun et al. 2001)



1) Calhoun VD, Adali T, McGinty V, Pekar JJ, Watson T, Pearlson GD. (2001): fMRI Activation In A Visual-Perception Task: Network Of Areas Detected Using The General Linear Model And Independent Component Analysis. NeuroImage 14(5):1080-1088.
2) Beckmann CF, Smith SM. (2005): Tensorial extensions of independent component analysis for multisubject FMRI analysis. NeuroImage 25(1):294-311.
3) Calhoun VD, Adali T, Pearlson GD, Pekar JJ. (2001): A Method for Making Group Inferences from Functional MRI Data Using Independent Component Analysis. Hum.Brain Map. 14(3):140-151.
4) Esposito F, Scarabino T, Hyvarinen A, Himberg J, Formisano E, Comani S, Tedeschi G, Goebel R, Seifritz E, Di SF. (2005): Independent component analysis of fMRI group studies by self-organizing clustering. Neuroimage. 25(1):193-205.
5) Schmithorst VJ, Holland SK. (2004): Comparison of three methods for generating group statistical inferences from independent component analysis of functional magnetic resonance imaging data. J.Magn Reson.Imaging 19(3):365-368.
6) Svensen M, Kruggel F, Benali H. (2002): ICA of fMRI Group Study Data. NeuroImage 16:551-563.
7) Guo Y, Giuseppe P. (In Press): A unified framework for group independent component analysis for multi-subject fMRI data. NeuroImage.

(Calhoun et al. 2009)

PCA & FA & ICA: differences



From PCA to FA

- PCA: based on 2nd order moment, linear transform
- SVD form can naturally lead to a latent representation (linear combination of uncorrelated basis), however, it's not determined
- FA adds a noise term and assume Gaussian, then use maximum likelihood to fit
- FA: based on 2nd order moment (uncorrelated), model-based generative, with noise, not determined

From FA to ICA

- FA: based on 2nd order moment, expects Gaussian, can be rotated
- ICA: statistically independent (based on all the cross-moments), unique representation, find the non-Gaussian components (thus max 4th order moment), starts with FA and look for possible rotations

PCA vs ICA

- PCA: max 2nd order moment
- ICA: max 4th order moment (or min mutual information, etc.; as independent as possible)

Evolving ideas of dimensionality reduction

From variable selection to construction

- ~~Starting point: variable selection~~
- PCA & FA & ICA: intuitions & practical
- PCA & FA & ICA: differences

From linear to nonlinear

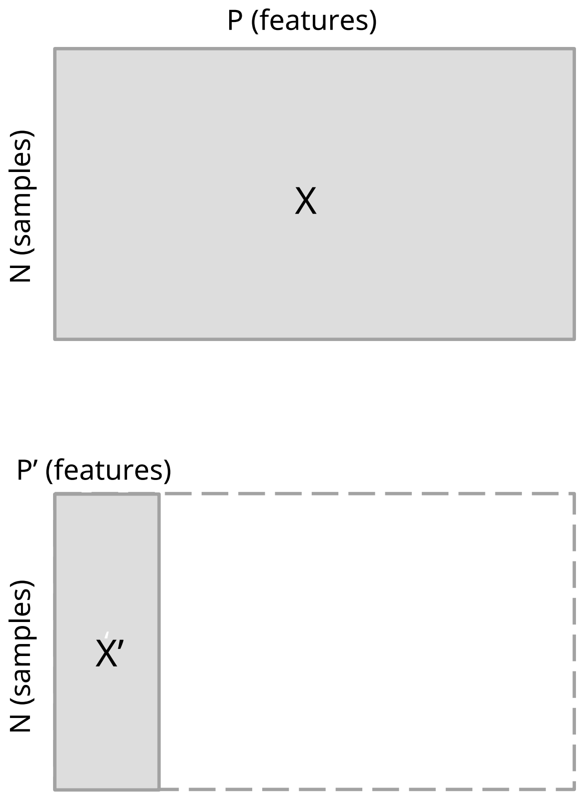
- Diffusion map

From decomposition to approximation

- tSNE & UMAP: intuitions & cautions

From dimensions to categories

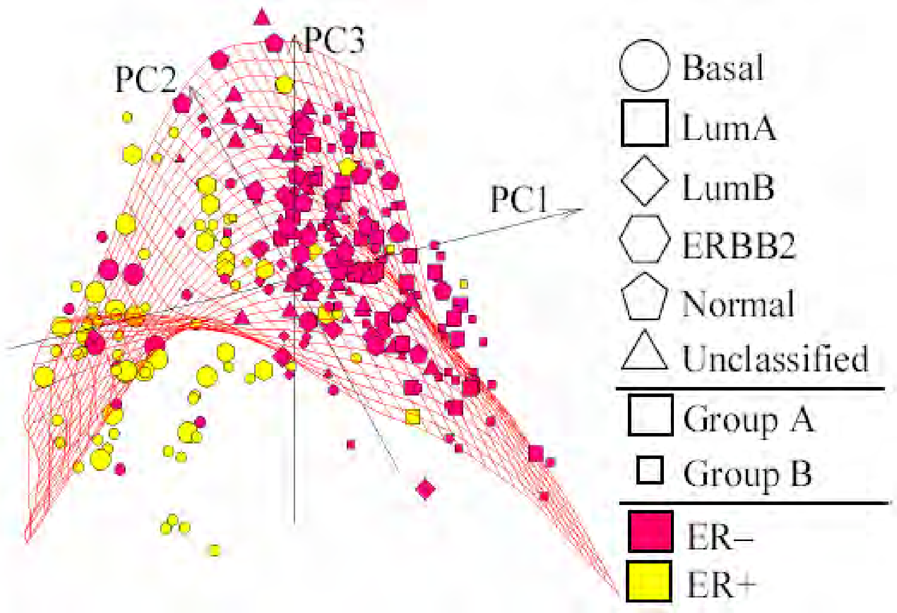
Back to the future



From linear to nonlinear

Why do we want to go nonlinear

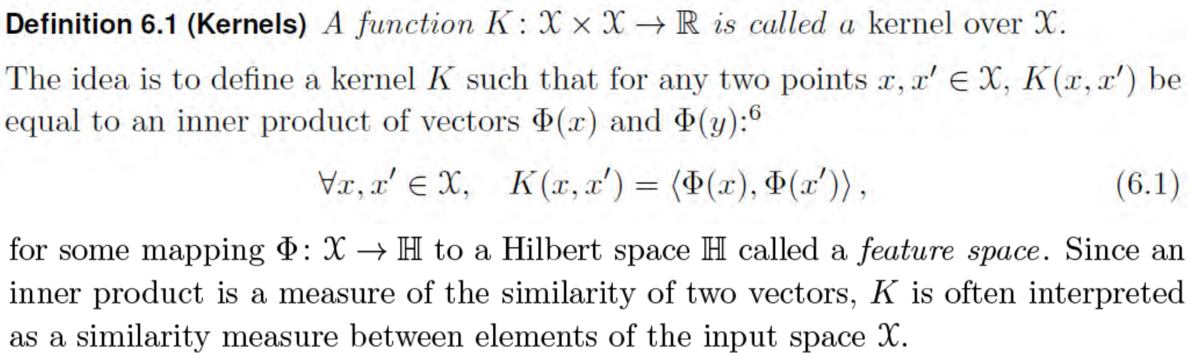
- Many data modalities are intrinsically high-dimensional and nonlinear



(Gorban and Zinovyev, 2010)

From linear to nonlinear

The kernel trick

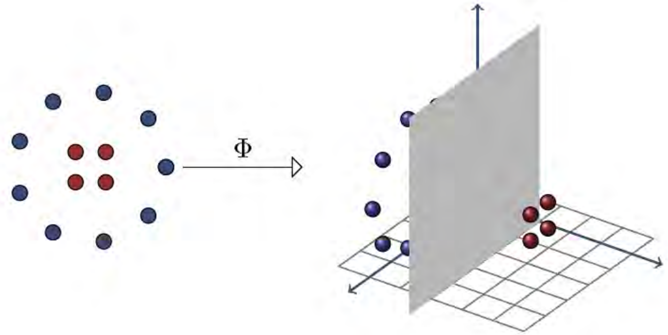


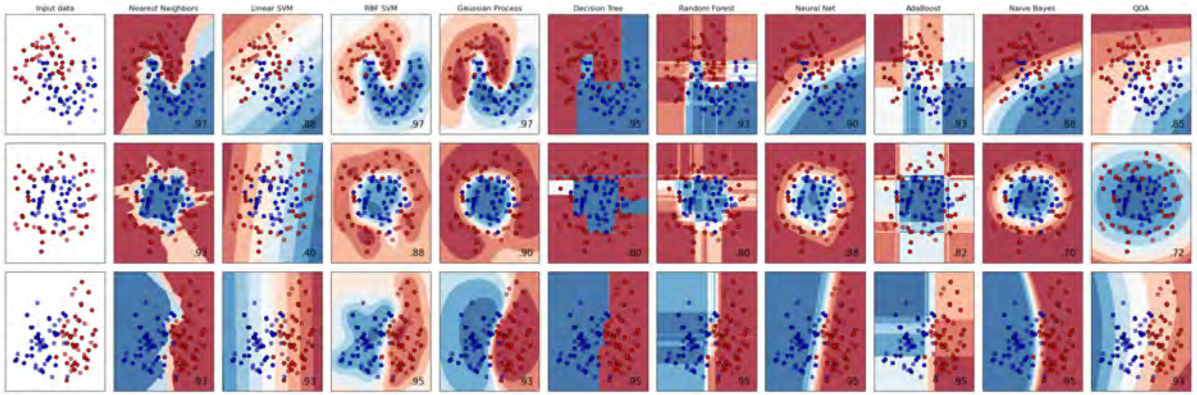
Definition 6.1 (Kernels) *A function* $K\colon \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ *is called a* kernel over $\mathcal{X}$.

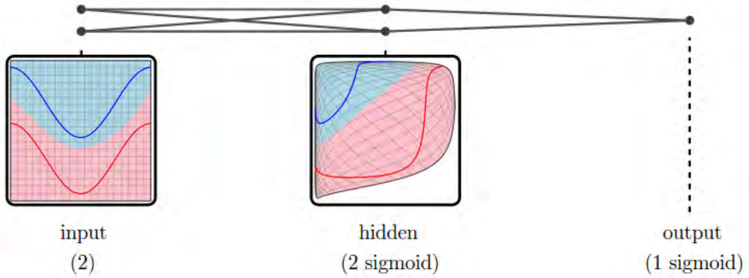
The idea is to define a kernel K such that for any two points $x, x' \in \mathcal{X}$, $K(x, x')$ be equal to an inner product of vectors $\Phi(x)$ and $\Phi(y)$:[6]

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \langle \Phi(x), \Phi(x') \rangle\,, \tag{6.1}$$

for some mapping $\Phi\colon \mathcal{X} \to \mathbb{H}$ to a Hilbert space $\mathbb{H}$ called a *feature space*. Since an inner product is a measure of the similarity of two vectors, K is often interpreted as a similarity measure between elements of the input space $\mathcal{X}$.







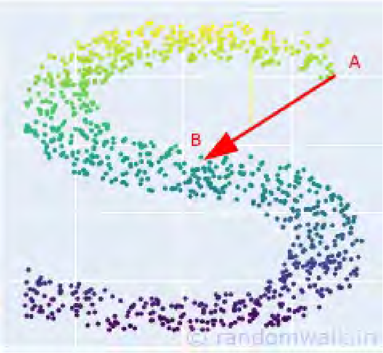
Schölkopf (1998), *Learning with Kernels*
Bishop (2006), *Pattern Recognition and Machine Learning*
Mohri et al (2018), *Foundations of Machine Learning*

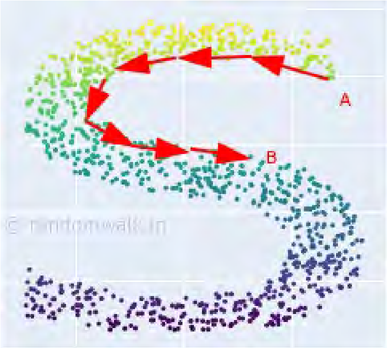
(https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/)
(https://colah.github.io/posts/2015-01-Visualizing-Representations/)

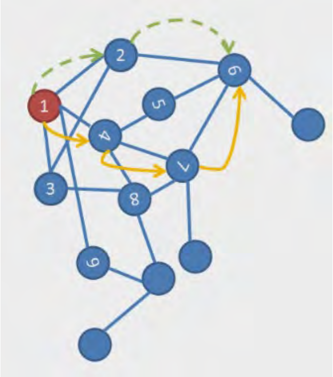
Diffusion map: intuitions

Diffusion map

- Focusing on the local “diffusion” effect, re-defining distance
- Naturally suited for connectomes





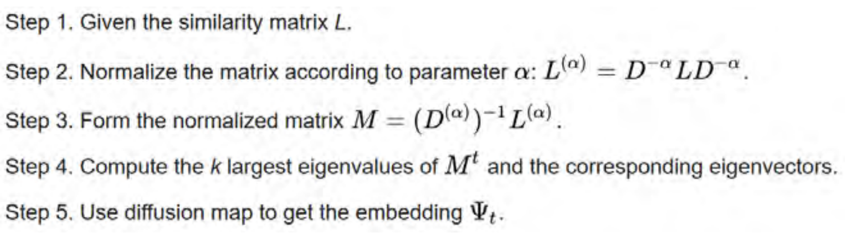


We focus on the local

$$k(x,y) = \exp\left(-\frac{||x-y||^2}{\epsilon}\right)$$

We simulate the diffusion

$$L_{i,j}^{(\alpha)} = k^{(\alpha)}(x_i, x_j) = \frac{L_{i,j}}{(d(x_i)d(x_j))^\alpha}$$



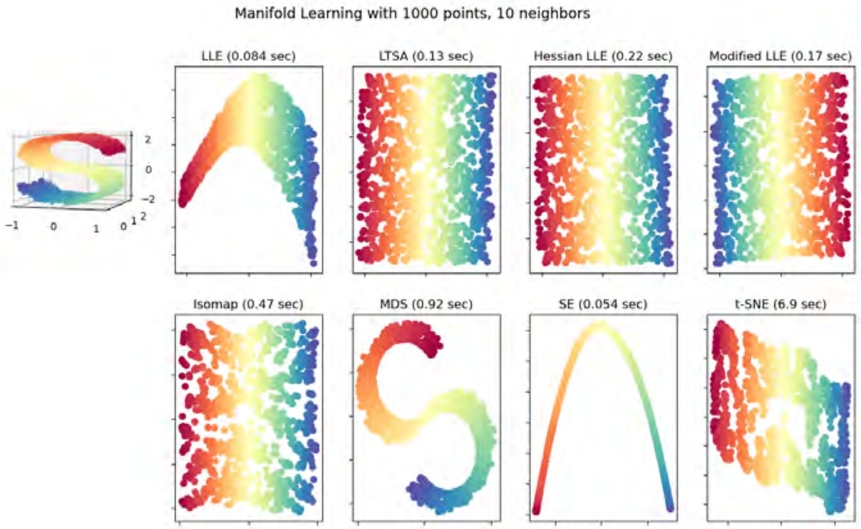
Step 1. Given the similarity matrix L.

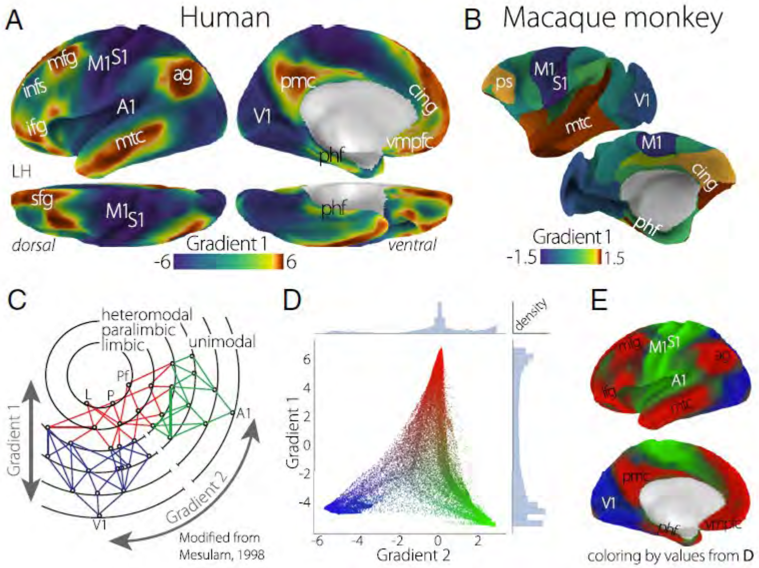
Step 2. Normalize the matrix according to parameter α: $L^{(\alpha)} = D^{-\alpha} L D^{-\alpha}$.

Step 3. Form the normalized matrix $M = (D^{(\alpha)})^{-1} L^{(\alpha)}$.

Step 4. Compute the k largest eigenvalues of M^t and the corresponding eigenvectors.

Step 5. Use diffusion map to get the embedding Ψ_t.





(Margulies et al., 2016)

Evolving ideas of dimensionality reduction

From variable selection to construction

- ~~Starting point: variable selection~~
- PCA & FA & ICA: intuitions & practical
- PCA & FA & ICA: differences

From linear to nonlinear

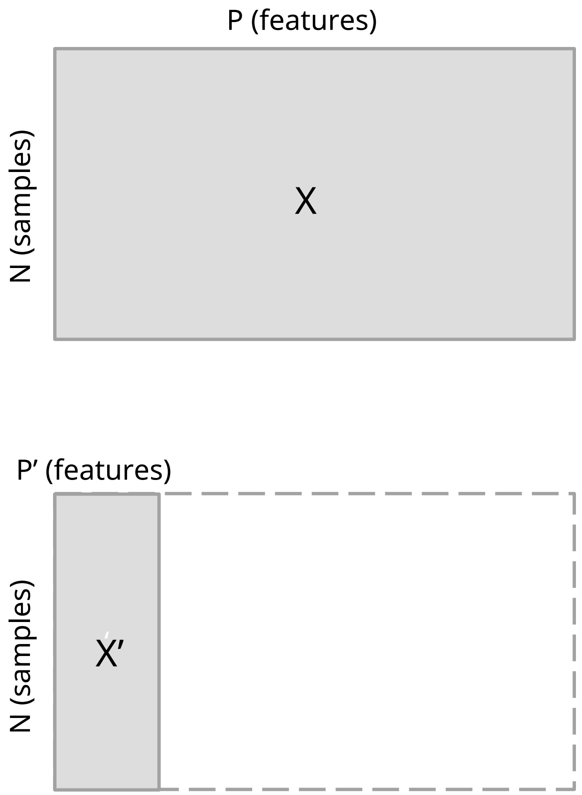
- Diffusion map

From decomposition to approximation

- tSNE & UMAP: intuitions & cautions

From dimensions to categories

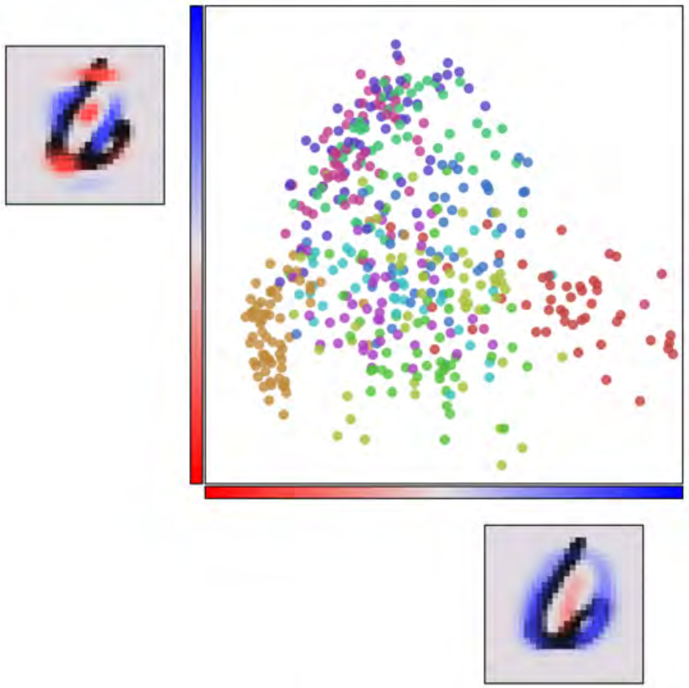
Back to the future



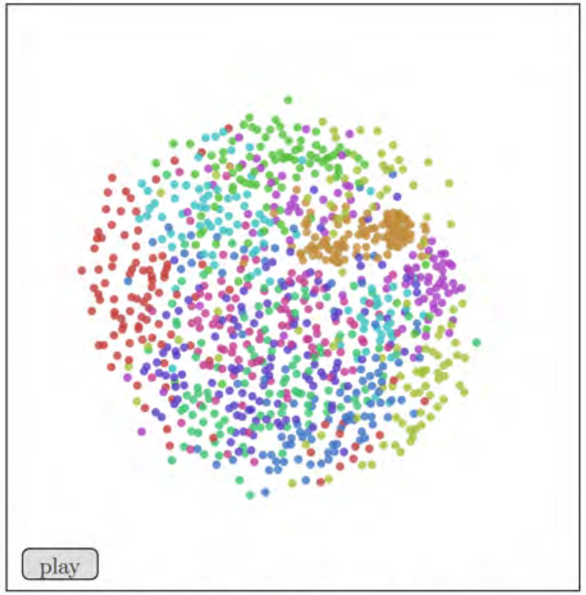
From decomposition to approximation

Moving to optimization-based dimensionality reduction

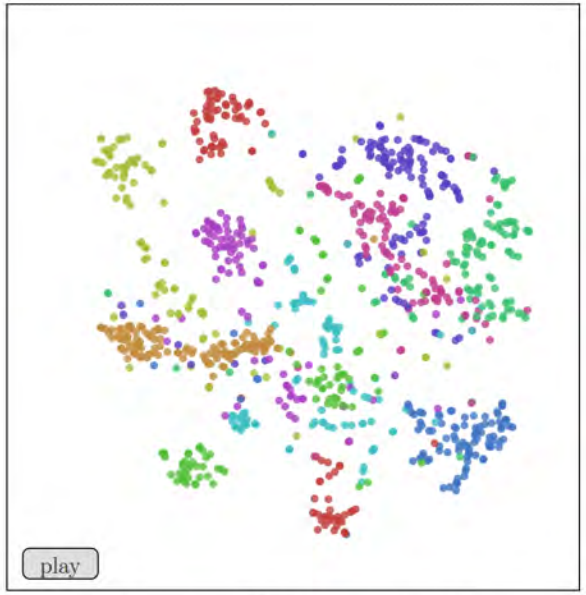
Keep “similarities” or “distances” between samples



Visualizing MNIST with PCA



Visualizing MNIST with MDS



Visualizing MNIST with t-SNE

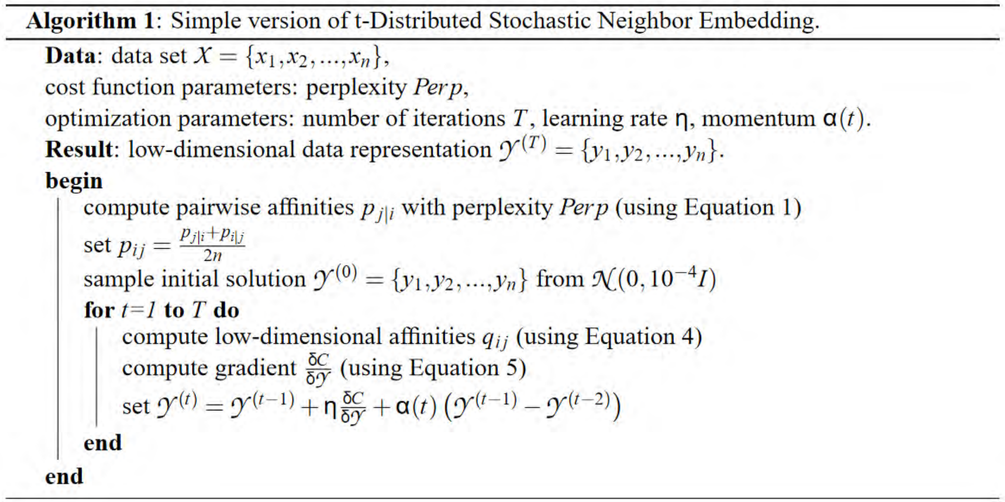
tSNE & UMAP: intuitions

t-distributed stochastic neighbour embedding

- Local closeness only, non-parametric

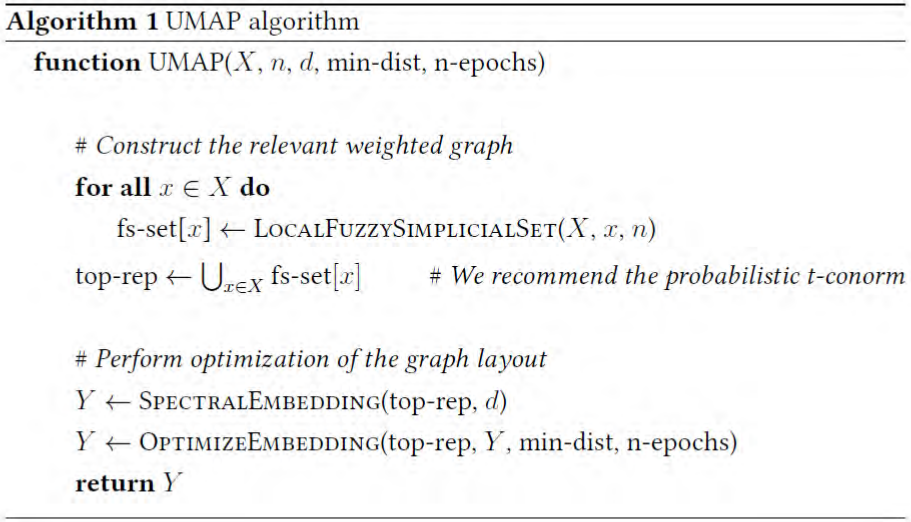
Uniform manifold approximation and projection

- Can capture the global structure, more efficient



Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

```
Data: data set $X = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
    compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
    set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
    sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4} I)$
    for $t=1$ to $T$ do
        compute low-dimensional affinities $q_{ij}$ (using Equation 4)
        compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
        set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left( \mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right)$
    end
end
```



Algorithm 1 UMAP algorithm

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

    # Construct the relevant weighted graph
    for all $x \in X$ do
        fs-set[$x$] ← LocalFuzzySimplicialSet($X$, $x$, $n$)
    top-rep ← $\bigcup_{x \in X}$ fs-set[$x$]        # We recommend the probabilistic t-conorm

    # Perform optimization of the graph layout
    $Y$ ← SpectralEmbedding(top-rep, $d$)
    $Y$ ← OptimizeEmbedding(top-rep, $Y$, min-dist, n-epochs)
    return $Y$
```

(van der Maaten and Hinton, 2008)

(McInnes et al., 2018)

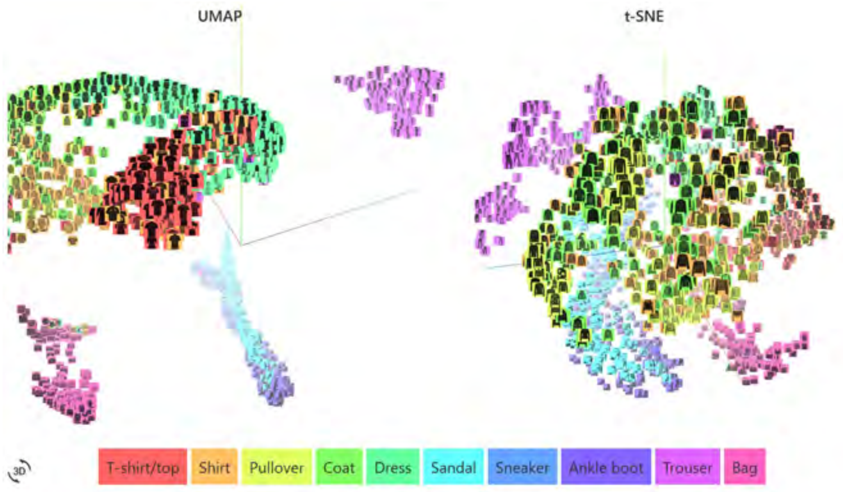
tSNE & UMAP: intuitions

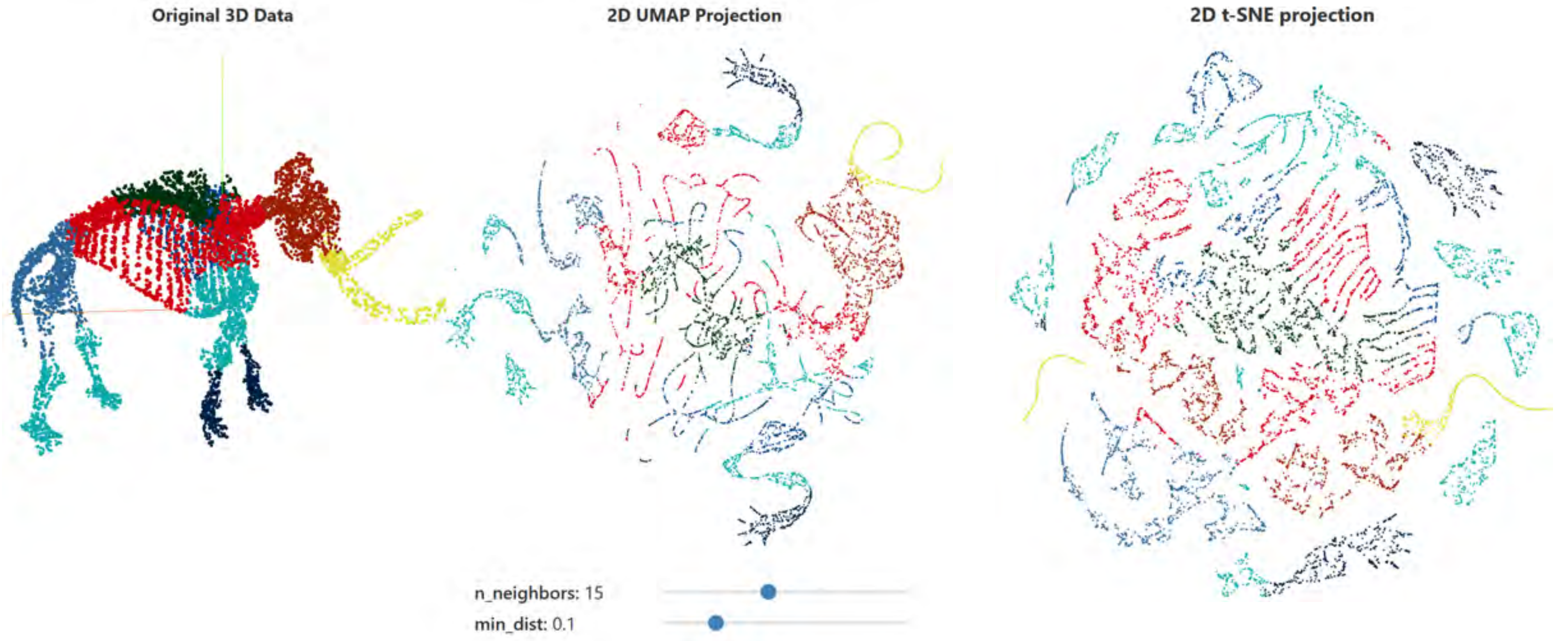
t-distributed stochastic neighbour embedding

- Local closeness only, non-parametric

Uniform manifold approximation and projection

- Can capture the global structure, more efficient





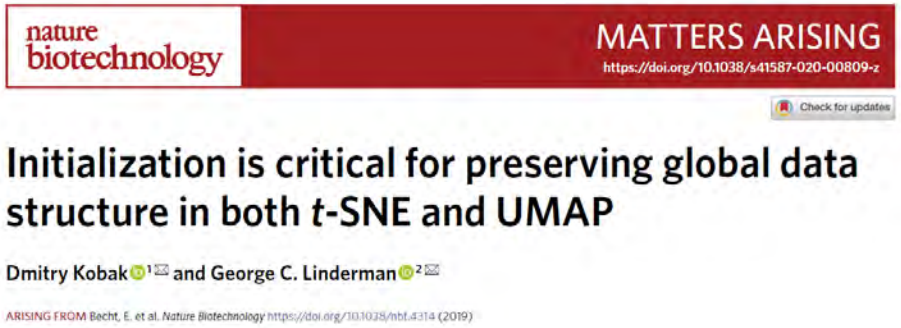
n_neighbors: 15

min_dist: 0.1

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

tSNE & UMAP: tales of caution

Explain the similarity structure with care



nature biotechnology

MATTERS ARISING

https://doi.org/10.1038/s41587-020-00809-z

Initialization is critical for preserving global data structure in both *t*-SNE and UMAP

Dmitry Kobak[1] and George C. Linderman[2]

ARISING FROM Becht, E. et al. *Nature Biotechnology* https://doi.org/10.1038/nbt.4314 (2019)

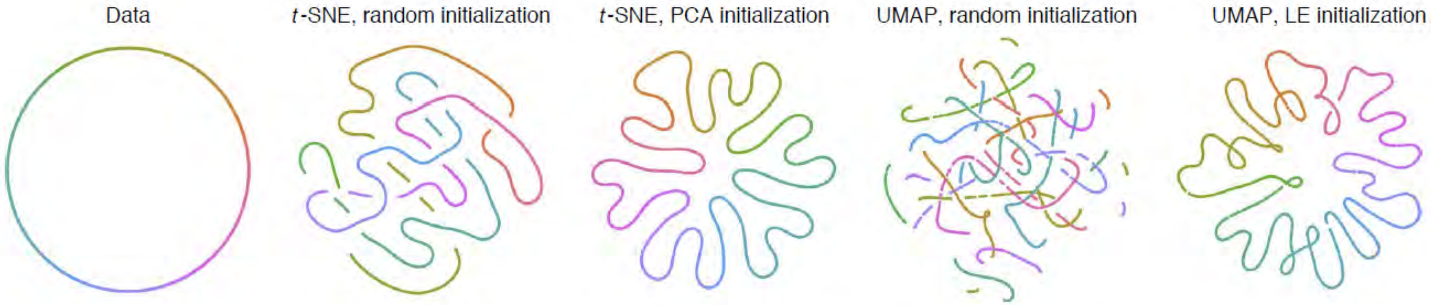


Fig. 1 | *t*-SNE and UMAP with random and non-random initialization. Embeddings of $n = 7{,}000$ points sampled from a circle with a small amount of Gaussian noise ($\sigma = r/1{,}000$, where r is the circle's radius). We used random and PCA initialization for *t*-SNE (openTSNE[11] v.0.4.4) and random and LE initialization for UMAP (v.0.4.6). All other parameters were kept as default. For this dataset, PCA and LE give the same initialization. Note that openTSNE scales PCA initialization to have s.d. = 0.0001, which is the default s.d. for random initialization in *t*-SNE[2]; similarly, UMAP scales the LE result to have a span of 20, which is the value it uses for random initialization.

Evolving ideas of dimensionality reduction

From variable selection to construction

- ~~Starting point: variable selection~~
- PCA & FA & ICA: intuitions & practical
- PCA & FA & ICA: differences

From linear to nonlinear

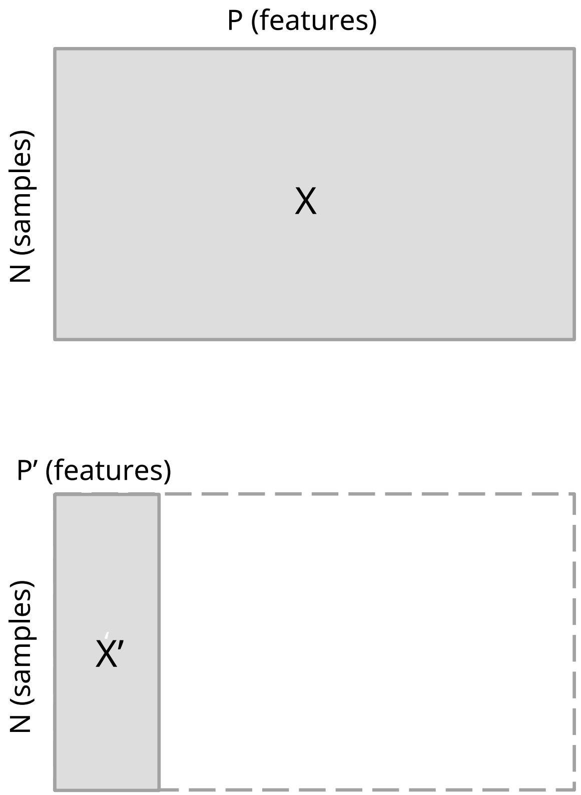
- Diffusion map

From decomposition to approximation

- tSNE & UMAP: intuitions & cautions

From dimensions to categories

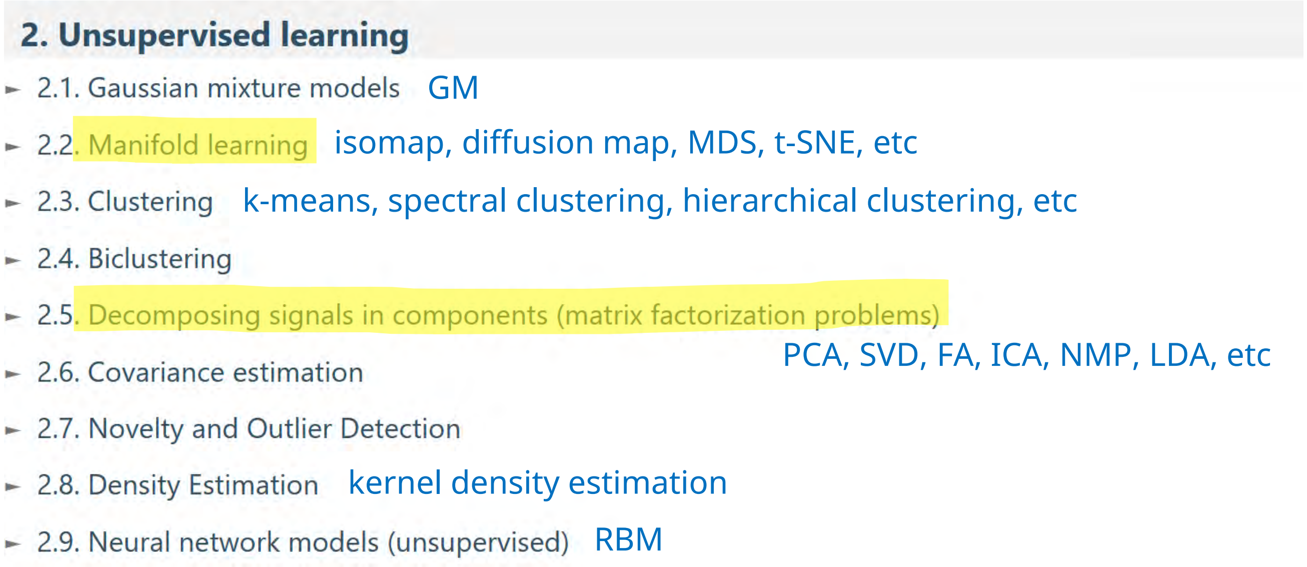
Back to the future



From dimensions to categories

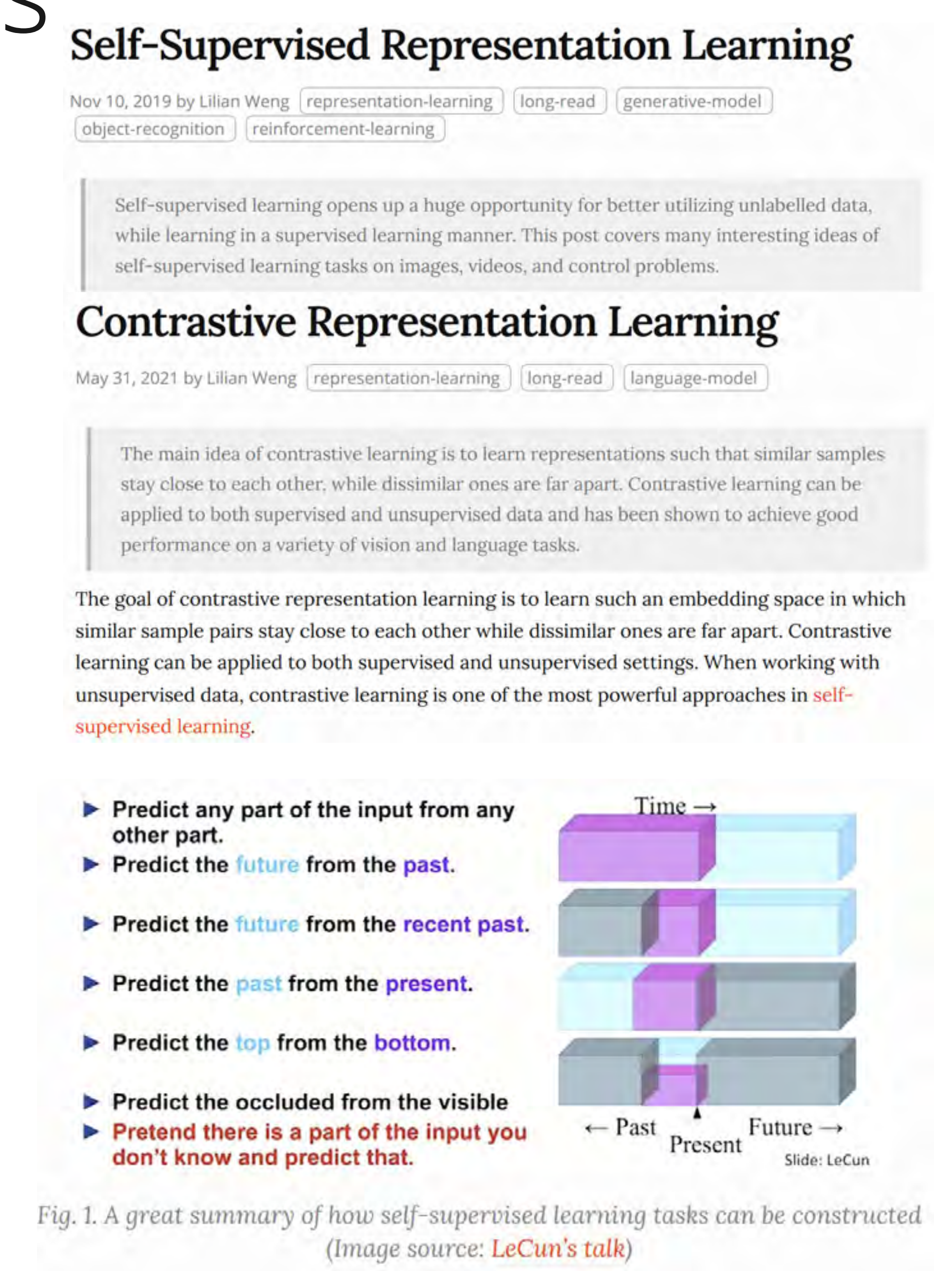
How to exploit structure in data

- Rapid development in recent years
- AI inspires neuroscience
- More challenges than knowns



2. Unsupervised learning

- 2.1. Gaussian mixture models GM
- 2.2. Manifold learning isomap, diffusion map, MDS, t-SNE, etc
- 2.3. Clustering k-means, spectral clustering, hierarchical clustering, etc
- 2.4. Biclustering
- 2.5. Decomposing signals in components (matrix factorization problems) PCA, SVD, FA, ICA, NMP, LDA, etc
- 2.6. Covariance estimation
- 2.7. Novelty and Outlier Detection
- 2.8. Density Estimation kernel density estimation
- 2.9. Neural network models (unsupervised) RBM



Self-Supervised Representation Learning

Nov 10, 2019 by Lilian Weng representation-learning long-read generative-model object-recognition reinforcement-learning

> Self-supervised learning opens up a huge opportunity for better utilizing unlabelled data, while learning in a supervised learning manner. This post covers many interesting ideas of self-supervised learning tasks on images, videos, and control problems.

Contrastive Representation Learning

May 31, 2021 by Lilian Weng representation-learning long-read language-model

> The main idea of contrastive learning is to learn representations such that similar samples stay close to each other, while dissimilar ones are far apart. Contrastive learning can be applied to both supervised and unsupervised data and has been shown to achieve good performance on a variety of vision and language tasks.

The goal of contrastive representation learning is to learn such an embedding space in which similar sample pairs stay close to each other while dissimilar ones are far apart. Contrastive learning can be applied to both supervised and unsupervised settings. When working with unsupervised data, contrastive learning is one of the most powerful approaches in self-supervised learning.

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.

Fig. 1. A great summary of how self-supervised learning tasks can be constructed (Image source: LeCun's talk)

(https://lilianweng.github.io/lil-log/)

Outline for today

Goal: ***filling the missing part of other introductions***

What you will learn:

- What is dimensionality reduction
- Why do we need dimension reduction

1. From variable selection to construction
   - PCA & FA & ICA
2. From linear to nonlinear
   - Diffusion map
3. From decomposition to approximation
   - tSNE & UMAP
4. From dimensions to categories

- **Back to the future**

We select variables

Best subset, ridge, LASSO…

We make new variables

PCA, FA, ICA…

We distort the data

Diffusion map, Isomap, LLE…

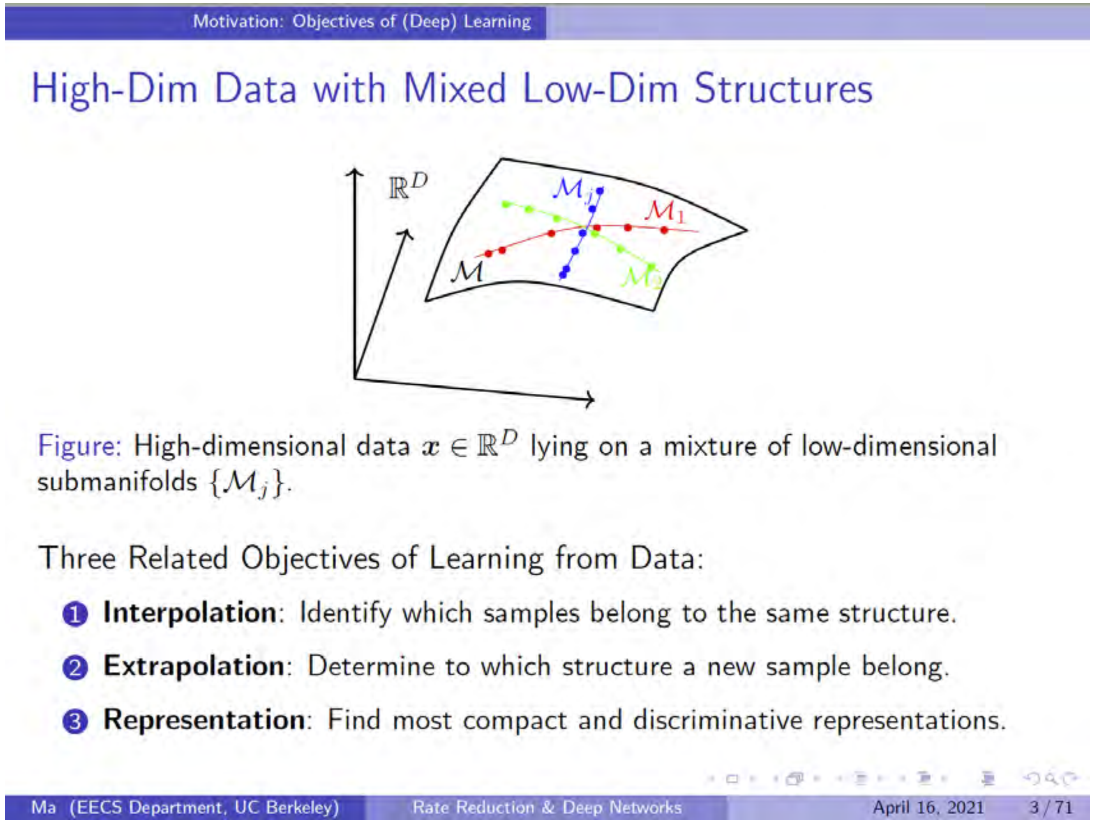
We make guesses

tSNE, UMAP…

What's next?

Back to the future

"Learning from data"

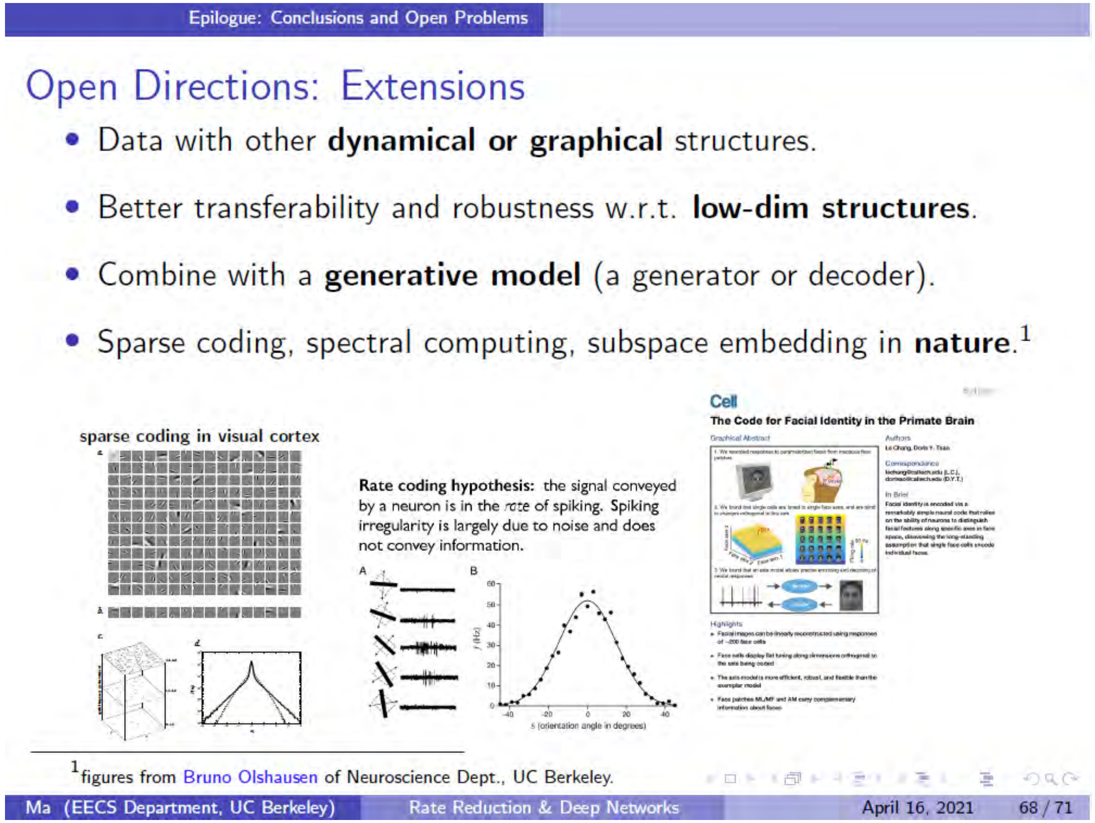


High-Dim Data with Mixed Low-Dim Structures

Figure: High-dimensional data $x \in \mathbb{R}^D$ lying on a mixture of low-dimensional submanifolds $\{\mathcal{M}_j\}$.

Three Related Objectives of Learning from Data:

1. **Interpolation**: Identify which samples belong to the same structure.
2. **Extrapolation**: Determine to which structure a new sample belong.
3. **Representation**: Find most compact and discriminative representations.



Open Directions: Extensions

- Data with other **dynamical or graphical** structures.
- Better transferability and robustness w.r.t. **low-dim structures**.
- Combine with a **generative model** (a generator or decoder).
- Sparse coding, spectral computing, subspace embedding in **nature**.[1]

Rate coding hypothesis: the signal conveyed by a neuron is in the rate of spiking. Spiking irregularity is largely due to noise and does not convey information.

[1] figures from Bruno Olshausen of Neuroscience Dept., UC Berkeley.

References: theoretical

Intro-level

- Neuromatch academy (https://academy.neuromatch.io/)
- 3Blue1Brown (https://www.3blue1brown.com/)

Quite theoretical

- Dayan and Abbott (2001), Theoretical Neuroscience
- Gerstner et al (2004), Neuronal Dynamics
- Hastie et al (2013), An Introduction to Statistical Learning: with Applications in R

Extremely theoretical

- Hastie et al (2009), The Elements of Statistical Learning
- Schölkopf (1998), Learning with Kernels
- Bishop (2006), Pattern Recognition and Machine Learning
- Mohri et al (2018), Foundations of Machine Learning

References: practical

Scikit-learn documentation (https://scikit-learn.org/stable/user_guide.html)

Cross Validated (https://stats.stackexchange.com/)

Talk to a friend with statistics background!