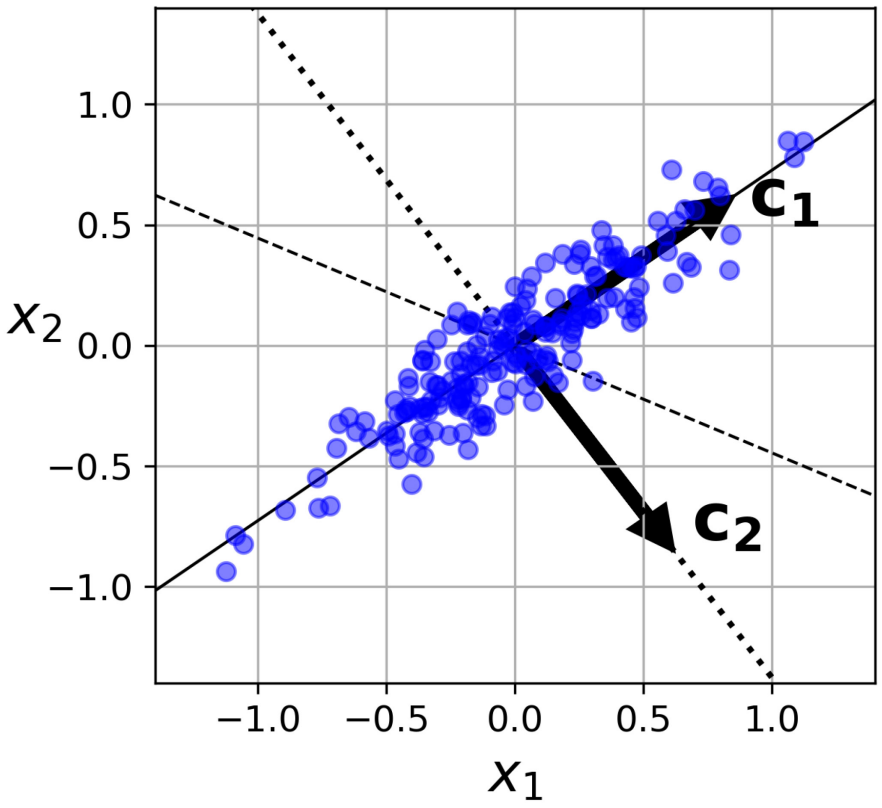


MSE MachLe
Dimensionality Reduction

Christoph Würsch
Institute for Computational Enineering ICE
Interstaatliche Hochschule für Technik Buchs, FHO

V10: To cope with the curse of dimensionality

V10: Dimensionality Reduction

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

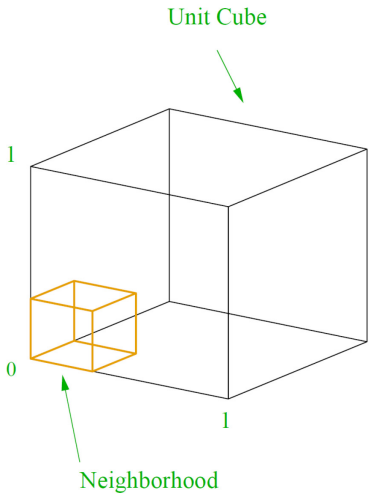
Agenda:

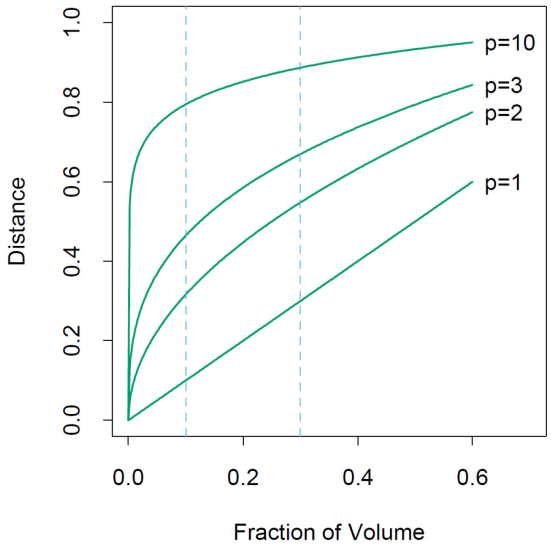
- Methods: feature **reduction** and feature **extraction**
- The **curse of dimensionality** and the mainfold hypothesis
- **The Manifold Hypothesis**
- **Principal Component Analysis**
  - PCA: Principal Component Ananlysis (linear)
  - Kernel PCA (non-linear)
- **Manifold Methods based on similarity**
  - **MDS**: Multidimensional scaling
  - **LLE**: local linear embedding
  - **Isomap**: Isometric mapping
  - **t-SNE**: t-distributed stochastic neighbor embedding

https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction#Manifold_learning_algorithms

Curse of dimensionality

- We are increasingly confronted with very high dimensional data from speech, images, and genomes and other sources.
- One of the characteristics of high-dimensional data is that the **number of dimensions** is comparable or larger than the **number of samples**.
- This has the consequence that the sample complexity of function approximation can grow exponentially.



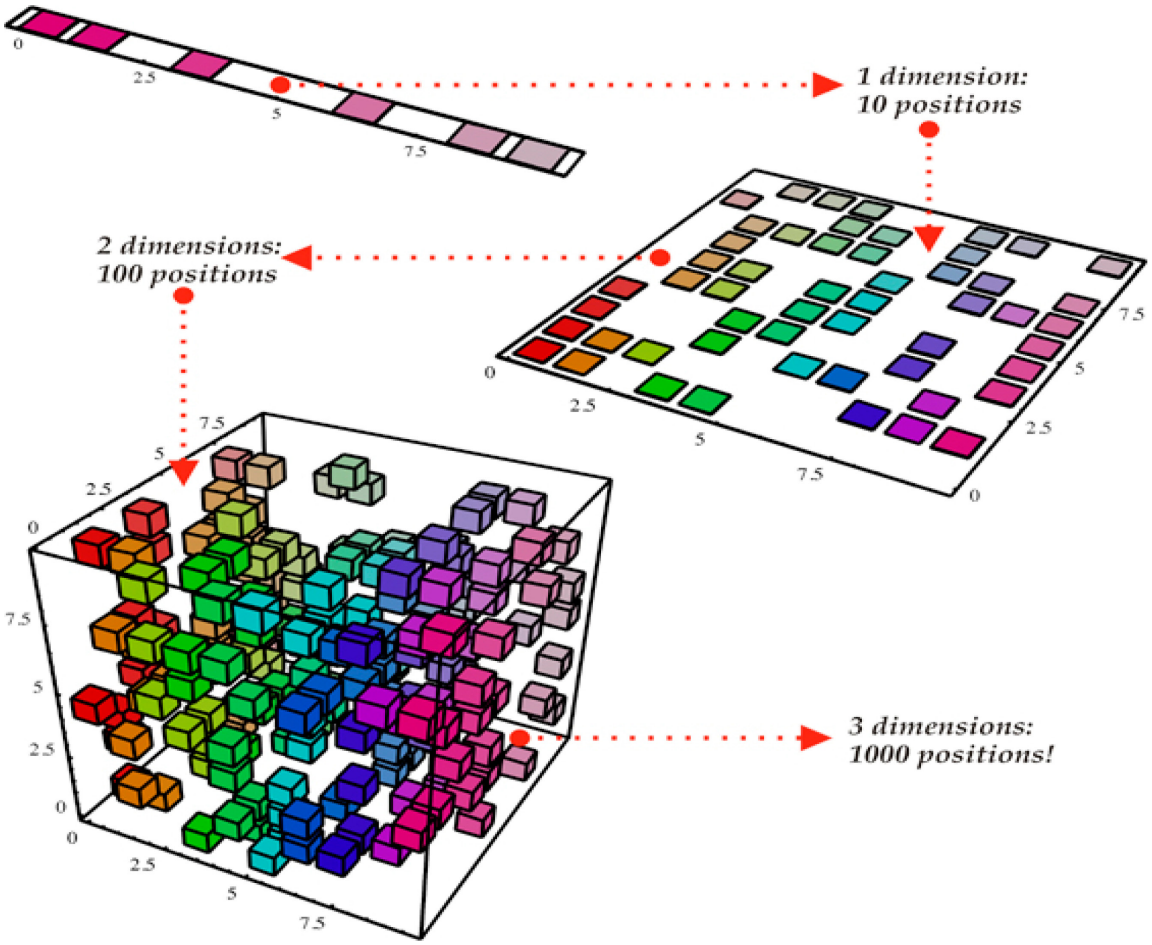


The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.

(Source: ESL)

intuition ?

- How many data are needed for a certain coverage?



dim	n for 10% coverage
1	10
2	100
3	1000
10	10^{10}
p	10^{p}

Life in High dimensional space

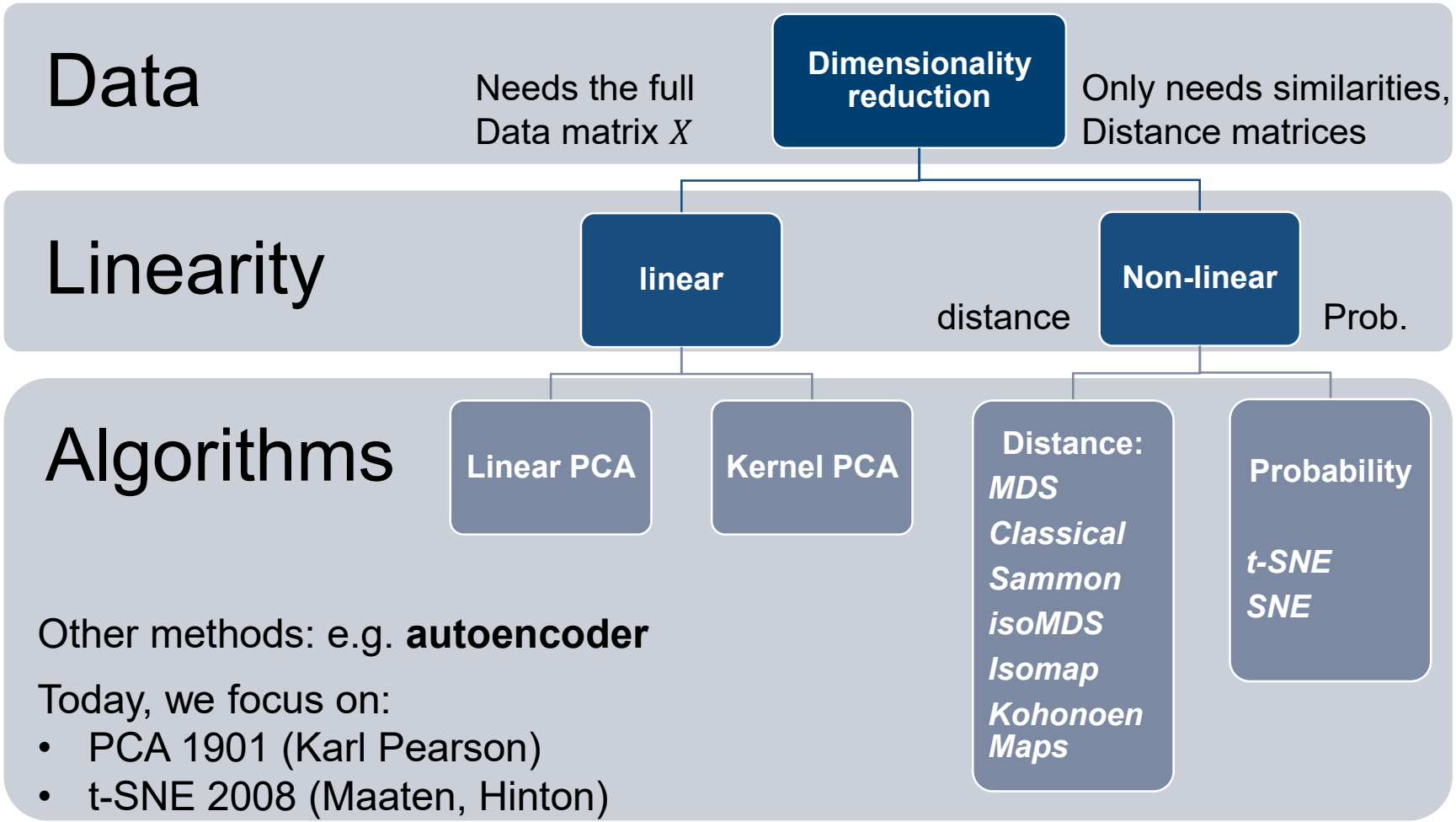
- d-dimensional unit hypercube: $H_d = [0,1]^d$
  - Calculate the pobability that a random point is closer that 0.001 from the border as function of d.
  - Calculate the average distance of two random points in the unit hypercube as function of d.
- **High dimensional data is very sparse**: most training instances are likely to be far away from each other making predictions much harder, greater risk of overfitting, joint probabilities are hard to fit.
- The number of necessary training data to reach a given density grows **exponentially** with the dimension d.
  - Whith 100 features (less than in the MINST problem), you would need $\approx 10^{100}$ training data, more than atoms in the universe, in order to be data points closer than 0.1 of each other.
- E.g. MNIST Dataset: The number of dimensions (= #pixels) can be reduced to accelerate training and to increase the robustness
  - Pixels are almost always white, strong correlation between neighbouring pixels

Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

- **Feature Reduction**: approaches try to find a subset of the original variables (also called features or attributes). There are three strategies:
  - the filter strategy (e.g. information gain),
  - the wrapper strategy (e.g. search guided by accuracy), and
  - the embedded strategy (features are selected to add or be removed while building the model based on the prediction errors).
- **Feature Extraction**: transforms the data in the high-dimensional space to a space of fewer dimensions

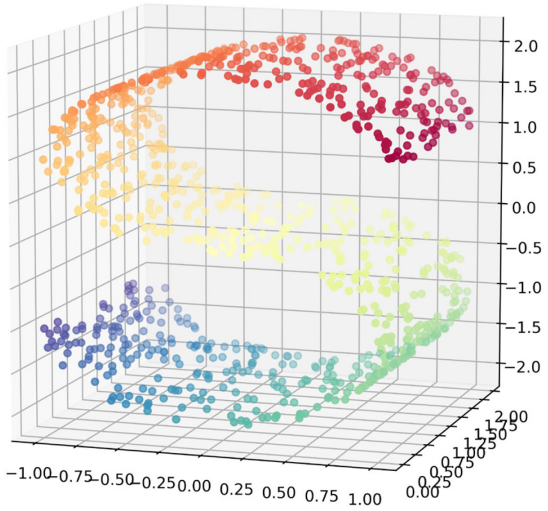
Overview

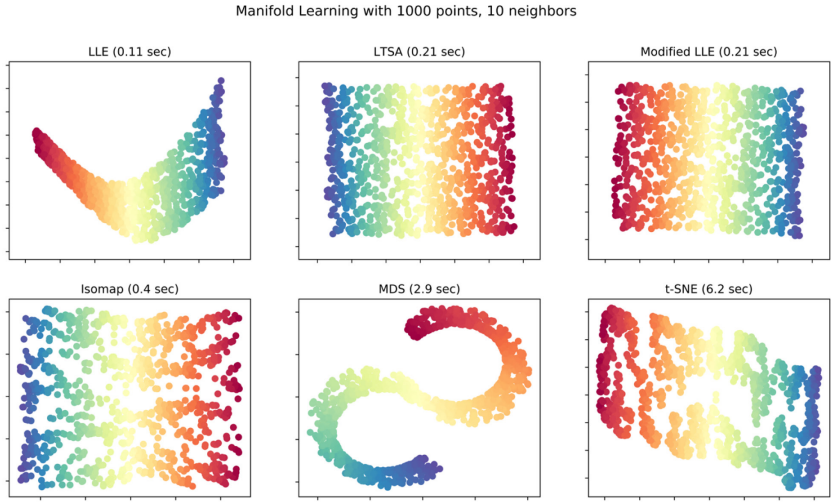


For a fine grained map see (Maarten, http://tsam-fich.wdfiles.com/local--files/apuntes/TPAMI_Paper.pdf)

Manifold Learning

- A collection of methodologies for analyzing high dimensional data based on the hypothesis that data tend to lie near a low dimensional manifold is now called **Manifold Learning**.
- Examples of low-dimensional manifolds embedded in high-dimensional spaces include: image vectors representing 3D objects under different illumination conditions and camera views and phonemes in speech signals.





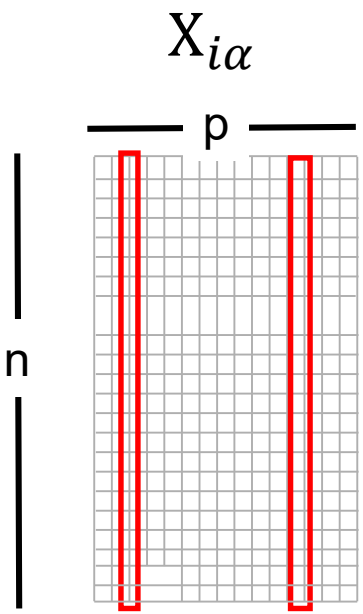
PCA: Principal Component Analysis

- The main **linear technique** for dimensionality reduction, principal component analysis PCA, performs a linear mapping of the data to a lower-dimensional space in such a way that the **variance of the data in the low-dimensional representation is maximized.**
- The eigen vectors that correspond to the largest eigenvalues (the principal components) can now be used to reconstruct a large fraction of the variance of the original data.
- PCA identifies the d-dimensional hyperplane that lies closest to the data and then projects the data on it. (d is a hyperparameter).

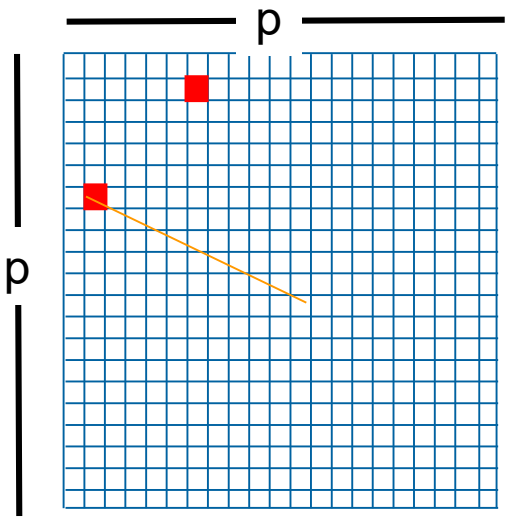
Data Matrix X and covariance Matrix C

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- Data Matrix $(i = 1 \ldots n)$



$$C_{ij} = \langle (X_i - \langle X_i \rangle)(X_j - \langle X_j \rangle) \rangle$$



Covariance Matrix

Sum over all n datapoints

$$\text{if} \quad \langle X_i \rangle = 0 \quad \Rightarrow \quad C = X^T X$$

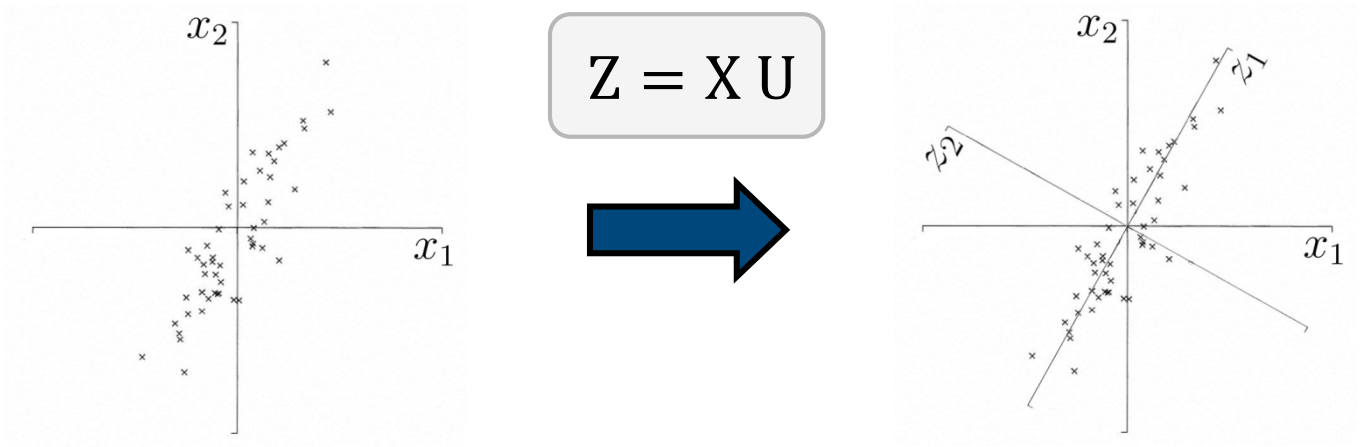
It turns out that the optimization corresponds to a **diagonalization** of the covariance matrix C.

$$C = U\, D\, U^T$$

PCA is a rotation

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- PCA is **a rotation (or reflection)** so that the first PC has the highest variance, the second is orthogonal to the first and has the second highest variance, … X is the (n,p)-dimensional data matrix column centered.



- How can one calculate this? → It turns out that one can solve either
- **the Eigenvalue Problem of X^TX (see next slide)**
- **or the Singular Value Decomposition of X**

Eigenvalues of $X^T X$

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

$$D = \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \\ \hline 0 & & & 0 \\ & \ddots & & \\ & & & 0 \end{bmatrix}$$

Diagonalize Covariance Matrix $C = X^T X$

$$C = U D U^T$$

$$U^T = U^{-1}$$

λ_i are the sorted Eigenvalues of C

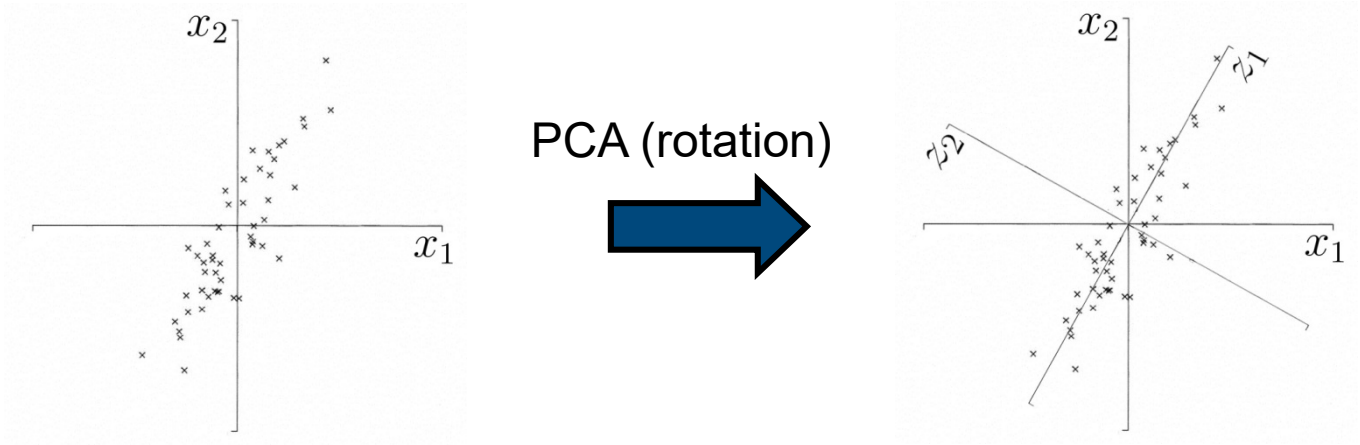
From linear algebra:

- Since **C** is positive and symmetrical **U is composed of the Eigenvectors of C.**
- U is a rotation which transforms the experiments into the new coordinate system.
  - In this new coordinate system the covariance matrix is D and is diagonal with the Eigenvalues λ_i on the diagonal and (since sorted) the first axis carries the most variance.
  - The λ_i are the variances in the new coordinate system.

It can be shown that this definition is equivalent to the definition rotate so that highest variance is in first component second highest in second.

Projection to the k most important directions (dimensions)

Situation: We now have a **rotation** into a new coordinate system.



- **No data is lost in that transformation**. The first component explains most of the variance, the second the second most,…. All components explain the whole variance.
- Now take only a few components and hope that data is explained by them.
- How good is the approximation? A measure is the **explained variance.**

Code Example

NTB Interstaatliche Hochschule für Technik Buchs

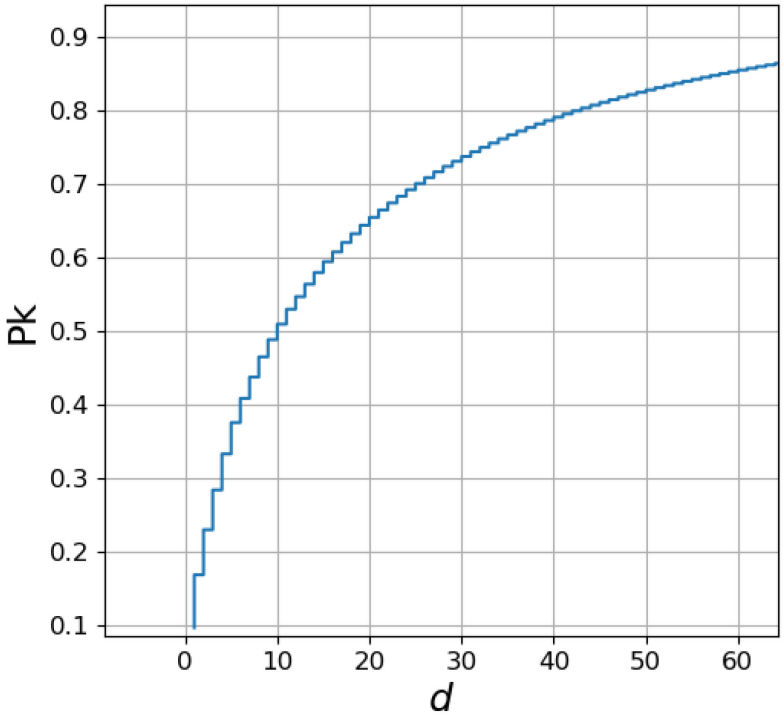
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
X3D_inv = pca.inverse_transform(X2D)

fig = plt.figure(figsize=(6, 5))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X3D_inv[:, 0], X3D_inv[:, 1], X3D_inv[:, 2],
cmap=plt.cm.hot) ax.view_init(10, -70)
ax.set_xlabel("$x_1$", fontsize=18)
ax.set_ylabel("$x_2$", fontsize=18)
ax.set_zlabel("$x_3$", fontsize=18)
```

- there is some loss of information during the projection step, so the recovered 3D points are not exactly equal to the original 3D points. Check this!

Explained Variance Ratio P_k

- The percentage the first k dimensions contribute to the variance (their importance).
- **Explained Variance Ratio:**

$$P_k = \frac{\sum_{i=1}^{k} \mathrm{var}(\mathrm{X}_i)}{\sum_{j=1}^{p} \mathrm{var}(\mathrm{X}_j)} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j}$$



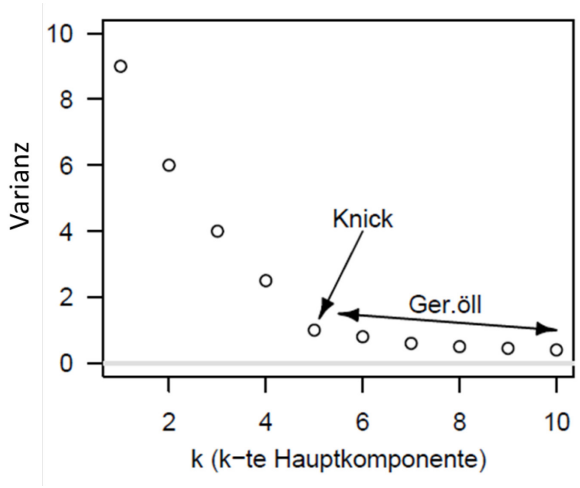

```
pca = PCA(n_components=0.80)
pca.explained_variance_ratio_
1 - pca.explained_variance_ratio_.sum()
```

How many PCs do we need? 1st and 2nd criterion

- 1st criterion: The total variance V_{tot} can be calculated as:
  (the total variance is preserved under rotation)

$$V_{\text{tot}} = \sum_{j=1}^{p} \text{var}(\text{X}_j) = \sum_{j=1}^{p} \text{var}(\text{Z}_j) = \sum_{j=1}^{p} \lambda_j$$

- 2nd criterion: P_k as fuction of the dimension: Test with MINST Dataset.



Interpretation 1: Minimization of SSE (Sum of suqared error vs. maximization of variance)

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

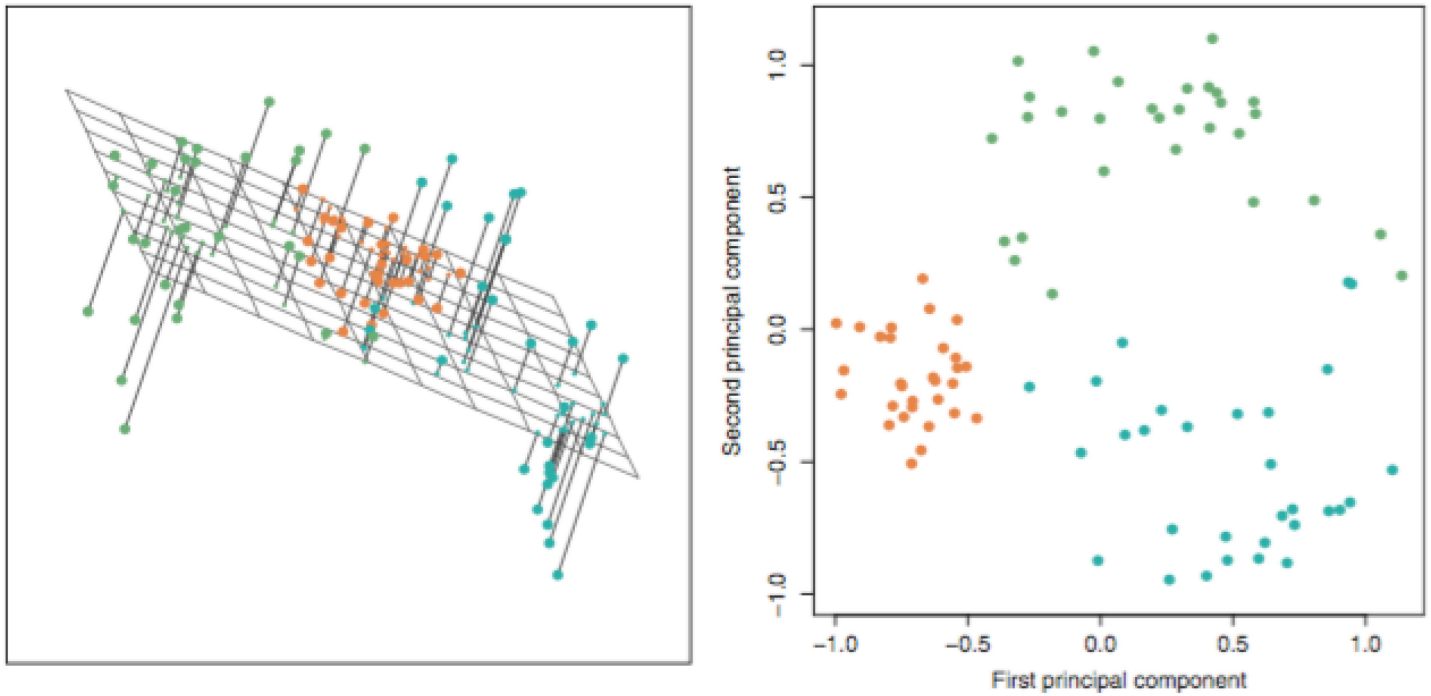


FIGURE 10.2. *Ninety observations simulated in three dimensions.* Left: *the first two principal component directions span the plane that best fits the data. It minimizes the sum of squared distances from each point to the plane.* Right: *the first two principal component score vectors give the coordinates of the projection of the 90 observations onto the plane. The variance in the plane is maximized.*

Interpretation 2: PCA minimizes the reconstruction error (autoencoder)

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- PCA Rotation can be achieved by multiplying X with an orthogonal rotation matrix U

$$Y_{(n\times p)} = X_{(n\times p)} \cdot U_{(p\times p)} \quad \textbf{(projection)}$$

$$X_{(n\times p)} = Y_{(n\times p)} \cdot U^{T}_{(p\times p)} \quad \textbf{(reconstruction)}$$

- Reconstruct X with only k<p Principal Components:

$$\hat{X}_{(n\times p)} = \left[Y_{(n\times k)}, 0_{(n\times p-k)}\right] \cdot U^{T}_{(p\times p)}$$

- **PCA minimizes the reconstruction error over all m available datapoints. The first k PCs define a k-dim. hyperplane to which the sum of squared projection errors is minimal.**

$$\min\left\{ \sum_{i=1}^{m} \left\| \hat{X}_i - X_i \right\|^2 \right\}$$

Maximum Variance Formulation I (e.g. Bishop)

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- D-Dimensional data vectors: $\{\boldsymbol{x}_n\}$ $(n = 1 \ldots N)$
- Data Matrix: $X_{(D\times N)} = [\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{N}]$
- Goal: Project the data to an M-dimensional subspace with $M < D$ while maximizing the variance of the projected data.
- Assume $M = 1 \Rightarrow$ there is a D-dimensional vector $\boldsymbol{u}_1$ that does this, the norm can be chosen to be one: $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$

The **mean** $\overline{\boldsymbol{x}_p}$ and *variance* $\mathrm{VAR_p}$ *of the projected data* are given by:

$$\overline{\boldsymbol{x}_p} = \boldsymbol{u}_1^T \overline{\boldsymbol{x}}$$

$$\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$$ **data mean**

$$\mathrm{VAR_p} = \frac{1}{N}\sum_{n=1}^{N} \{\boldsymbol{u}_1^T \boldsymbol{x}_n - \boldsymbol{u}_1^T \overline{\boldsymbol{x}}\}^2 = \boldsymbol{u}_1^T \mathrm{C}\, \boldsymbol{u}_1$$

data covariance matrix

$$\mathrm{C} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T$$

Maximum Variance Formulation II (e.g. Bishop)

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- Now maximize $\mathrm{VAR_p}$ under the constraint, that the norm of the projection operator is one: $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$
- Use a Lagrange multiplier λ_1 for a constrained optimization

$$L(\boldsymbol{u}_1, \lambda_1) = \boldsymbol{u}_1^T \mathrm{C}\, \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1) \qquad \textbf{Lagrange Function}$$

$$\left.\begin{aligned} \frac{\partial L}{\partial \boldsymbol{u}_1} &= 0 \\ \frac{\partial L}{\partial \lambda_1} &= 0 \end{aligned}\right\} \quad \begin{aligned} &\Rightarrow \quad \boxed{\mathrm{C}\, \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1} \\ &\Rightarrow \quad \boldsymbol{u}_1^T \mathrm{C}\, \boldsymbol{u}_1 = \lambda_1 \end{aligned}$$

- The variance will be maximum, if we set $\boldsymbol{u}_1$ to be the **eigenvector of C with the largest value (1st principal component).**
- We can define additional principal components λ_k in an incremental fashion by choosing each new direction that maximizes the variance amongst all possible directions orthogonal to the those already considered.
- The optimal linear projection matrix is then given by $[\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M]$

Minimum error formulation I (Bishop)

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- Define an **orthonormal** basis $\boldsymbol{u}_i$ $(i = 1 \ldots D)$ of the D-dimensional dataset. Each vector $\boldsymbol{x}_n$ can be represented exactly in this basis (rotation). The α_i are the components in this new basis $\boldsymbol{u}_i$.

$$\boldsymbol{x}_n = \sum_{i=1}^{D} \alpha_{ni} \boldsymbol{u}_i$$

- Taking the inner product with $\boldsymbol{u}_j$ and making use of the orthonomality $\boldsymbol{u}_i^T \boldsymbol{u}_j = \delta_{ij}$, we find:

$$\alpha_{nj} = \boldsymbol{x}_n^T \boldsymbol{u}_j$$

$$\boldsymbol{x}_n = \sum_{i=1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_i)\, \boldsymbol{u}_i \qquad (1)$$

- Our goal is an approximation with a restricted number $M < D$ of variables:

$$\tilde{\boldsymbol{x}}_n = \sum_{i=1}^{M} z_{ni}\, \boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i\, \boldsymbol{u}_i \qquad (2)$$

Minimum error formulation II (Bishop)

- We are free to chose $\{\boldsymbol{u}_i\}$, $\{z_{ni}\}$ and the $\{b_i\}$ to minimize the MSE:

$$\mathrm{J} = \frac{1}{N}\sum_{n=0}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 \qquad \tilde{\boldsymbol{x}}_n = \sum_{i=1}^{M} z_{ni}\,\boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i\,\boldsymbol{u}_i \qquad (3)$$

- Setting all derivatives to zero:

$$\frac{\partial \mathrm{J}}{\partial z_{nj}} = 0 \quad \Rightarrow \quad z_{nj} = \boldsymbol{x}_n^T \boldsymbol{u}_j \qquad (j = 1, \ldots M)$$

$$\frac{\partial \mathrm{J}}{\partial b_j} = 0 \quad \Rightarrow \quad b_j = \overline{\boldsymbol{x}}^T \boldsymbol{u}_j \qquad (j = M+1, \ldots D) \qquad (4)$$

- If we substitue these expressions in (3) and make use of (1), we obtain:

$$\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n = \sum_{i=M+1}^{D} \{(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T \boldsymbol{u}_i\}\,\boldsymbol{u}_i \qquad (5)$$

- The displacement vector $\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n$ ist orthogonal to the principal subspace $\boldsymbol{u}_i$ $(i = 1 \ldots M)$ because it is a linear combination of $\boldsymbol{u}_i$ with $(i = M+1 \ldots D)$

Minimum error formulation III (Bishop)

- The MSE can therefore be written:

$$\mathrm{J} = \frac{1}{N}\sum_{n=0}^{N}\|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}(\boldsymbol{x}_n^T\boldsymbol{u}_i - \bar{\boldsymbol{x}}^T\boldsymbol{u}_i)^2 = \sum_{i=M+1}^{D}\boldsymbol{u}_i^T\mathrm{C}\boldsymbol{u}_i \qquad (6)$$

- This leads to a Lagrangian as before, but for eigenvectors orthogonal to the projection subspace, e.g. for D=2, M=1 ($\boldsymbol{u}_1$ is the projection subspace, $\boldsymbol{u}_2$ is orthogonal)

$$L(\boldsymbol{u}_2, \lambda_2) = \boldsymbol{u}_2^T\mathrm{C}\,\boldsymbol{u}_2 + \lambda_2(1 - \boldsymbol{u}_2^T\boldsymbol{u}_2) \qquad (7)$$

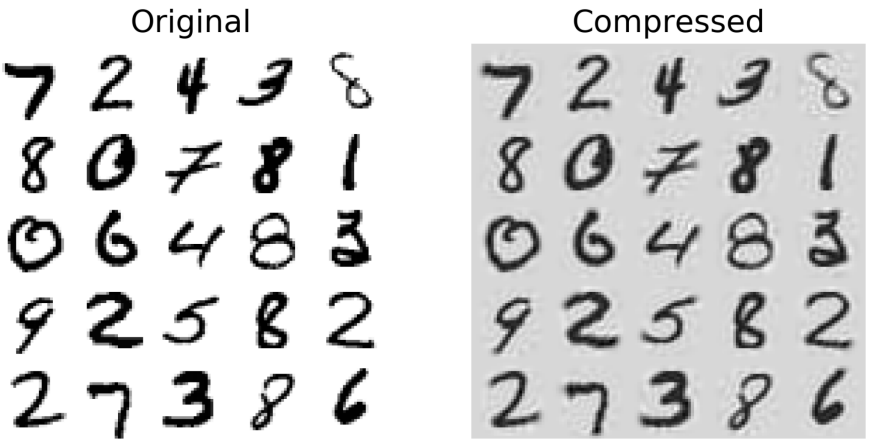
$$\Rightarrow \quad \mathrm{C}\,\boldsymbol{u}_2 = \lambda_2\boldsymbol{u}_2$$

- The general solution for J is therefore:

$$\mathrm{J} = \sum_{i=M+1}^{D}\boldsymbol{u}_i^T\mathrm{C}\,\boldsymbol{u}_i = \sum_{i=M+1}^{D}\lambda_i \qquad (5)$$

- To minimize J, we have to sum the $(D - M)$ **smallest** eigenvalues for the subspace orthogonal to the projection. Or the other way round: **the D eigenvectors corresponding to the D largest eigenvalues of C span the subspace, where the MSE error is minimized.**

PCA for data compression



- Lab: Load the MNIST dataset, select a training dataset

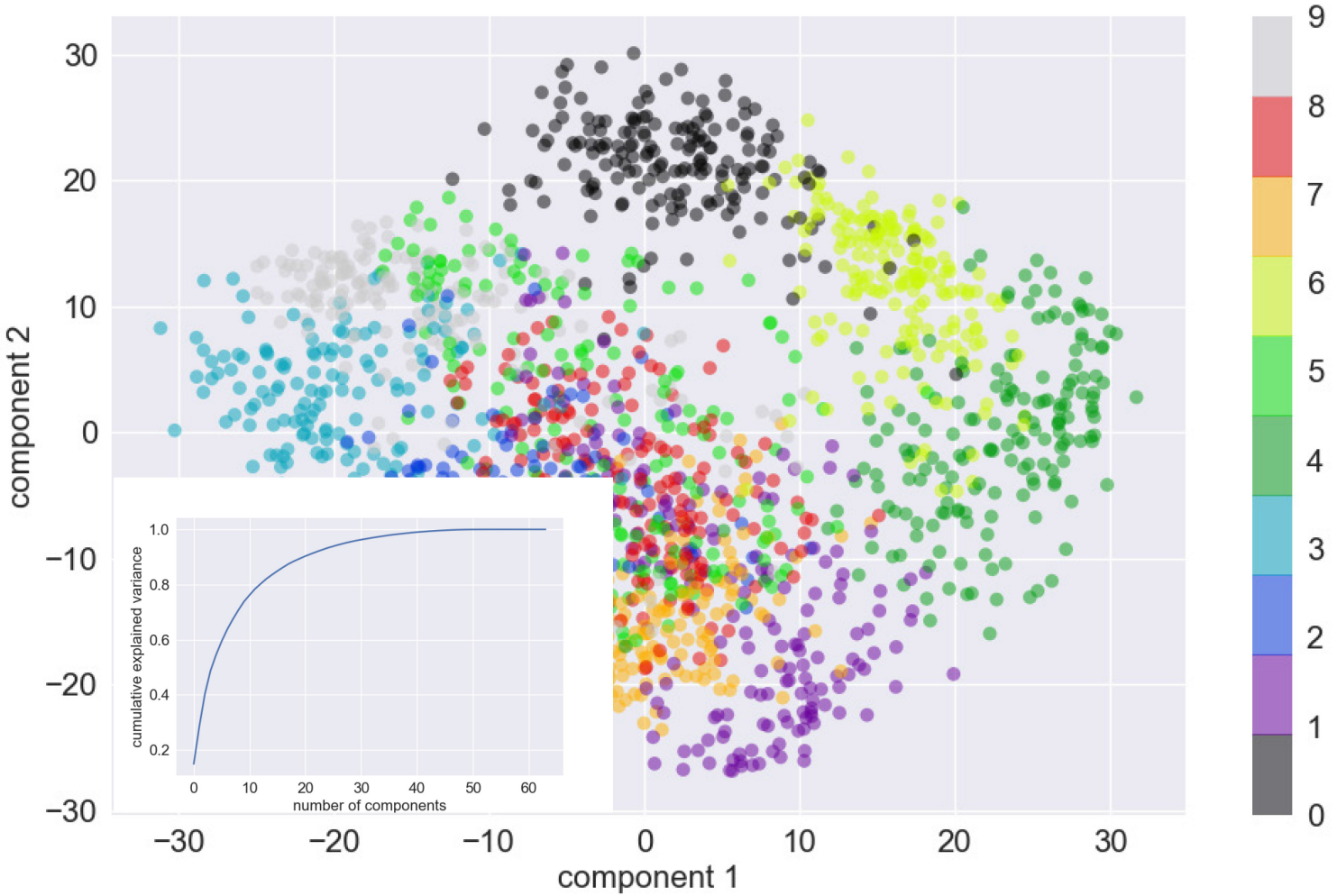
```
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
```

- PCA: Plot the explained variance as function of the dimension
- Select k for $P_k = 0.95$
- Reconstruct the original data using the projected dataset.
- Plot the reconstructed and the original data for comparison.

PCA on MNIST digits dataset

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

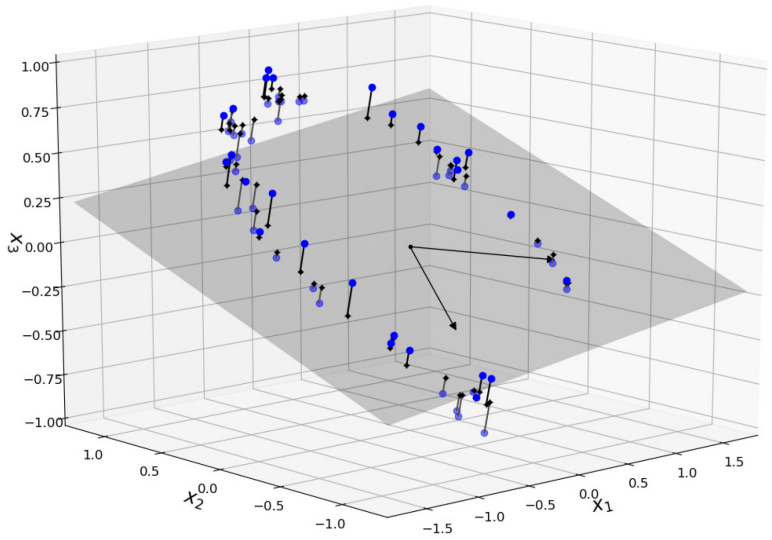
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

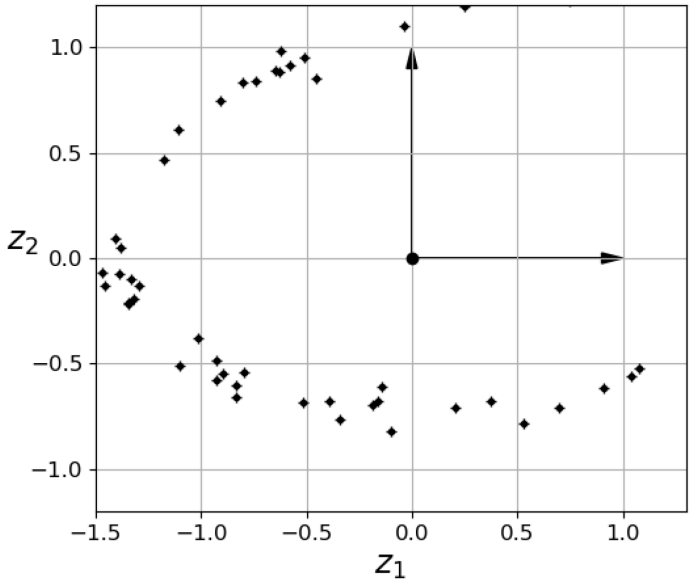


Mainly on a one dimensional manifold

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- Kernel PCA





Kernel PCA and the Kernel Trick

- Principal component analysis can be employed in a nonlinear way by means of the **kernel trick**.
- Data that inseparable in the input space can be transformed in higher dimensional space: we make the problem linearly separable by a simple mapping, e.g.: ϕ: $\mathbb{R}^2 \rightarrow \mathbb{R}^2 : (x_1, x_2) \longmapsto (x_1, x_2, x_1^2 + x_2^2)$
- High-dimensional mapping can seriously increase computation time. Can we get around this problem and still get the benefit of high-D? YES: using the Kernel-Trick:

$$K(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j)$$

- Given any algorithm that can be expressed solely in terms of **dot products**, this trick allows us to **construct different nonlinear versions** of it.

Popular kernels

- Gaussian (RBF): $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\beta \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2)$
- Laplacian: $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma |\boldsymbol{x} - \boldsymbol{x}'|_1)$
- Polynomial: $K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^p$
- Sigmoid : $K(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left(\alpha\, \boldsymbol{x}^T \boldsymbol{x}' + \delta\right)$
- Linear Kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$

Kernel PCA (KPCA)

Kernel Principal Component Analysis (KPCA) extends conventional principal component analysis (PCA) to a high dimensional feature space D using the "**kernel trick**".

1. Mapping into a nonlinear feature space: $x_i \mapsto \phi(x_i)$
2. Extract the PCA in that space → the result will be nonlinear in the original data space

Mean value and variance in feature space D:

Mean value

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)$$

Covariance matrix

$$C = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T$$

Eigenvalue problem

$$C\,v = \lambda\,v$$

How it works: Proof (optional)

- Eigenvectors of C can be expressed as linear combination of the features, i.e. they lie in the span of $\{\phi(x_1), \phi(x_2), \ldots \phi(x_n)\}$

$$v = \sum_{i=1}^{n} \alpha_i \, \phi(x_i)$$

This is only a scalar!

$$Cv = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\, \phi(x_i)^T v = \frac{1}{n}\sum_{i=1}^{n} (\phi(x_i)\, v)\, \phi(x_i)^T = \lambda v$$

for: $\lambda \neq 0$

$$v = \frac{1}{\lambda n}\sum_{i=1}^{n} \phi(x_i)\, \phi(x_i)^T v = \frac{1}{\lambda n}\sum_{i=1}^{n} (\phi(x_i)\, v)\, \phi(x_i)^T = \sum_{i=1}^{n} \alpha_i \, \phi(x_i)^T$$

- **Finding the eigenvectors v in the high dimensional feature space D is equivalent to finding the coefficients α_i in the n-dimensional data space.**

Proof II (optional)

- Substituting back, we find:

$$Cv_j = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\,\phi(x_i)^T \overbrace{\left[\sum_{j=1}^{n} \alpha_{jl}\,\phi(x_l)\right]}^{v_j} = \lambda_j \sum_{i=1}^{n} \alpha_{jl}\,\phi(x_l)$$

$$Cv_j = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i) \left[\sum_{j=1}^{n} \alpha_{jl} \underbrace{\boldsymbol{\phi(x_i)^T \phi(x_l)}}_{K(x_i, x_l)}\right] = \lambda_j \sum_{i=1}^{n} \alpha_{jl}\,\phi(x_l)$$

- Multiply this by $\phi(x_k)$ from the left:

$$\frac{1}{n}\sum_{i=1}^{n} \phi(x_k)^T \phi(x_i)\,\alpha_{jl}[K(x_i, x_l)] = \lambda_j \sum_{i=1}^{n} \alpha_{jl}\,\phi(x_k)^T \phi(x_l)$$

$$\frac{1}{n} K_{ki}\alpha_{jl}K_{il} = \lambda_j \alpha_{jl}\,K_{kl}$$

Proof III (optional) :

- In matrix form:

$$\frac{1}{n} K_{ki} \alpha_{jl} K_{il} = \lambda_j \alpha_{jl} K_{kl}$$

$$K_{ki} K_{il} \alpha_{jl} = n\lambda_j \alpha_{jl} K_{kl}$$

$$K^2 \alpha_j = n\lambda_j K \alpha_j$$

$$K \alpha_j = n\lambda_j \alpha_j$$

Eigenvalue problem for the Kernel matrix

- For a new point x, its projection y_j onto the principal component j is:

$$y_j = \phi(x)^T v_j = \sum_{i=1}^{n} \alpha_{ji} \phi(x)^T \phi(x_i) = \sum_{i=1}^{n} \alpha_{ji} K(x, x_i) \qquad (j = 1 \dots d)$$

http://sebastianraschka.com/Articles/2014_kernel_pca.html

Visualization of high dimensional data using similarities: Basic Idea (MDS, SNE)

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- **Situation**: given data $\vec{y}$ in high dimensional space with distance, (e.g. 99 features)
- **Idea**: Have distances / dissimilarities d_{ij} between many objects in high dimensional space → draw this in low dimensional space $(\mathbb{R}^2, \mathbb{R}^3)$

$$d_{ij}^* \rightarrow d_{ij} = \left\| \vec{y}_i - \vec{y}_j \right\|^2$$

Low-dimensional space

High-dimensional space

- The distances in (low-D) d_{ij}^* should match the original ones d_{ij} (high-D) as “good as possible”

Methods based on similarity

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

1. **MDS**: Multidimensional scaling
2. **LLE**: local linear embedding
3. **Isomap**: Isometric mapping
4. **t-SNE:** t-distributed stochastic neighbor embedding

Similarity is based on a metric, a generalized form of distance measure.

Overview

- **Multidimensional Scaling (MDS)** reduces dimensionality while trying to preserve the distances between *all* the instances
- **Local Linear Embedding (LLE)** reduces dimensionality while trying to preserve the distances between *close* instances only.
- **Isomap** creates a *graph* by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the geodesic distances between the instances.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space (e.g., to visualize the MNIST images in 2D).
- **Linear Discriminant Analysis (LDA)** is actually a classification algorithm. During training it learns the most discriminative axes between the classes. These axes can be used to define a hyperplane onto which to project the data. The projection will keep classes as far apart as possible, so LDA is a good technique to reduce dimensionality before running another classification algorithm such as an SVM classifier.

Similarity: metric and non-metric

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

A **metric** $d(x, y)$ is a generalized distance measure that follows the following axioms

1. Non-negativity: $d(x, y) \geq 0$
2. Coincidence: $d(x, y) = 0 \Leftrightarrow x = y$
3. Symmetry: $d(x, y) = d(y, x)$
4. Triangle inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Examples of metrics:

a. *Euclidian* and other L_p-Metric: induced metric by the norm of the vector space $\|x - y\|_p$
b. *Jaccard-Distance* (1 - Jaccard Index)
c. *Graph Distance* (shortest-path)
d. *Wasserstein* metic: between two probability distributions

Examples of metrics

- L_p metric: $d_p(x,y) = \|x-y\|_p = \sqrt[p]{\sum_i |x_i - y_i|^p}$
- L_∞ metric: $\|x\|_\infty = \max_i\{|x_i|\}$
- L_1 metric: $d_1(x,y) = \|x-y\|_1 = \sum_i |x_i - y_i|$
  (taxicab distance, Manhattan distance, chess, compressed sensing)

- **Hamming distance**: In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.

String Metrics

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- String metrics are in areas including fraud detection, fingerprint analysis, plagiarism detection, ontology merging, DNA analysis, RNA analysis, image analysis, evidence-based machine learning, data mining, data integration, and semantic knowledge integration.

Name	Example
Hamming distance	"**k**a**r**o**l**i**n**" and "**k**a**t**h**r**i**n**" is 3.
Levenshtein distance and Damerau–Levenshtein distance	**kitten** and **sitting** have a distance of 3. 1. **kitten** → **sitten** (substitution of "s" for "k") 2. **sitten** → **sittin** (substitution of "i" for "e") 3. **sittin** → **sitting** (insertion of "g" at the end).
Jaro–Winkler distance	JaroWinklerDist("MARTHA","MARHTA") = $d_j = \frac{1}{3}\left(\frac{m}{\lvert s_1\rvert} + \frac{m}{\lvert s_2\rvert} + \frac{m-t}{m}\right) = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{6-\frac{2}{2}}{6}\right) = 0.944$ • m is the number of *matching characters*; • t is half the number of *transpositions*(`"MARTHA"[3]!=H, "MARHTA"[3]!=T`).
Most frequent k characters	MostFreqKeySimilarity('research', 'seeking', 2) = 2

https://en.wikipedia.org/wiki/String_metric#List_of_string_metrics

1. MDS: Multi-Dimensional Scaling

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- Multidimensional scaling (MDS) seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space.
- MDS attempts **to model similarity or dissimilarity** data as distances in a geometric spaces. In general, is a technique used for analyzing similarity or dissimilarity data.

Modern Multidimensional Scaling, Theory and Applications
Ingwer Borg, Patrick J. F. Groenen in *Springer Series in Statistics (2005)*

1.1 Classical MDS = Principal Coordinates Analysis (PCoA)

- Classical MDS takes an input matrix I giving dissimilarities between pairs of items and outputs a coordinate matrix whose configuration minimizes a **loss function called strain**.
- For example, given the aerial distances between many cities in a matrix $D = [d_{ij}]$ where d_{ij} is the distance between the coordinates of i^{th} and j^{th} city, we want to find the coordinates x_i of the cities.
- The **strain** is given by:

$$S_D(x_1, \dots, x_N) = \left(\frac{\sum_{ij} (d_{ij} - \langle x_i | x_j \rangle)^2}{\sum_{ij} d_{ij}^2} \right)^{\frac{1}{2}}$$

$$S_D(x_1, \dots, x_N) \approx \sum_{i<j} \left\| d_{ij} - d_{ij}^* \right\|$$

- MDS attempts to find an embedding from the high dimensional objects in I into $x_i \in \mathbb{R}^N$ such that distances d_{ij} are preserved:

1.2 Steps of a Classical MDS algorithm

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

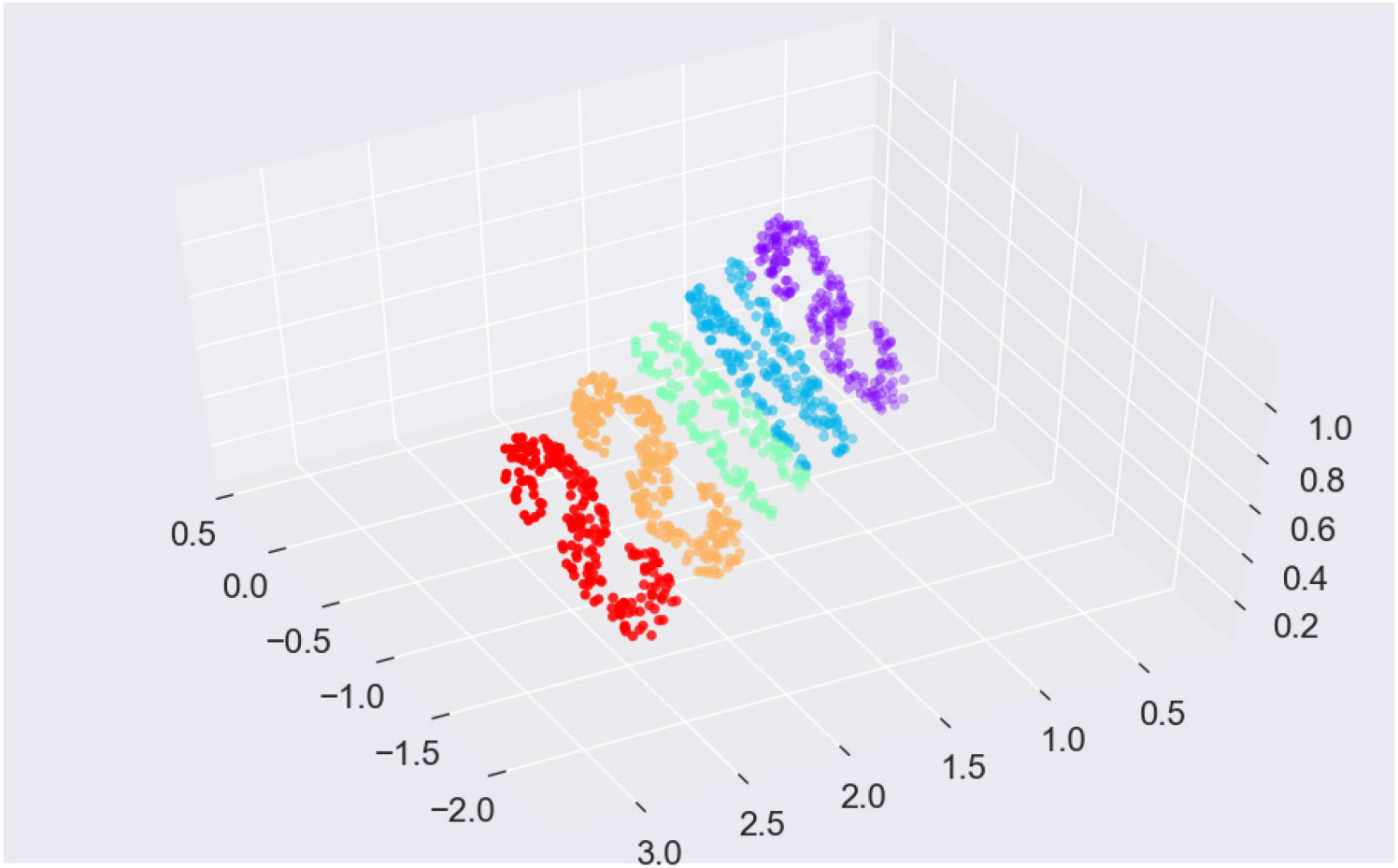
- Classical MDS uses the fact that the coordinate matrix can be derived by an eigenvalue decomposition from $B = XX^T$. And the matrix B can be computed from proximity matrix D by using **double centering**

  1. Set up the squared proximity matrix (covariance): $C = D^{(2)} = \left[d_{ij}^2\right]$
  2. Apply double centering: $B = -\frac{1}{2}HCH^T$ using the **centering matrix** $\mathrm{H} = \mathbb{I}_n - \frac{1}{n}11^T$ where n is the number of objects.
  3. Determine the **m largest eigenvalues** $\{\lambda_1, \dots, \lambda_m\}$ and corresponding **eigenvectors** $\{u_1, \dots, u_m\}$ of B (m= output dimension)
  4. Now, $X = U_m(\Lambda_m)^{1/2}$, where U_m ist the matrix of m eigenvectors $[u_1, \dots, u_m]$ and Λ_m is the diagonal matrix of meigenvalues of B

- Classical MDS assumes **Euclidean distances**.
  So this is not applicable for direct dissimilarity ratings.

1.3 Multidimensional Scaling (MDS) as mainfold earning (lab)

NTB
Interstaatliche Hochschule für Technik Buchs



1.4 Summary MDS

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- Is fast, because it is based on linear algebra
- Only distances are needed as input (as all MDS methods)
- The formulation as a cost function is valid for Euclidian distances only (internally Eigenvalues are used). If other distances (besides Euclidian) are used, nothing is guaranteed.
- MDS tries to preserve the distance between all data points, even if they are far separated → **LLE** only preserves distance between nearby points and is often able to map complex structures to low dimensions.
- For Euclidean distances, **classical MDS is equivalent to PCA** (but conceptually different)

2. LLE = local linear embedding

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- LLE describes the local properties of the manifold around a data point x_i by writing the data point as a linear combination (the so-called reconstruction weights w_{ij}) of its *k nearest neighbors*: $x_j \approx \sum_j w_{ij} x_j$
- In the low dimension, LLE attempts to **retain the reconstruction weights W as well as possible**.
- Hence, LLE fits a hyperplane through the data point x_i and its nearest neighbors, thereby **assuming that the manifold is locally linear.**
- The local linearity assumption implies that the reconstruction weights w_{ij} of the data points x_i are **invariant to translation, rotation, and rescaling.**
- Because of the invariance to these transformations, any linear mapping of the hyperplane to a space of lower dimensionality preserves the reconstruction weights in the space of lower dimensionality.
- As a consequence, finding the d-dimensional data representation Y amounts to minimizing the cost function ϕ:

$$\phi(Y) = \sum_i \left(y_i - \sum_{j=1}^{k} w_{ij}\, y_j \right)^2$$

2. LLE = local linear embedding

- first, for each training instance x_i, the algorithm identifies its k closest neighbors (e.g. k = 10), then tries to reconstruct x_i as a linear function of these neighbors: $x_j \approx \sum_j w_{ij} x_j$
- More specifically, it finds the weights w_{ij} such that the squared distance between x_i and $\sum_j w_{ij} x_j$ is as small as possible, assuming $w_{ij} = 0$ if x_j is not one of the k closest neighbors of x_i.
- Thus the first step of LLE is the **constrained optimization problem** where W is the weight matrix containing all the weights w_{ij}.
- The second constraint simply normalizes the weights for each training instance x_i

$$\widehat{W} = \underset{W}{\operatorname{argmin}} \left\{ \sum_{i=1}^{m} \left\| x_i - \Sigma_j \, w_{ij} x_j \right\|^2 \right\}$$

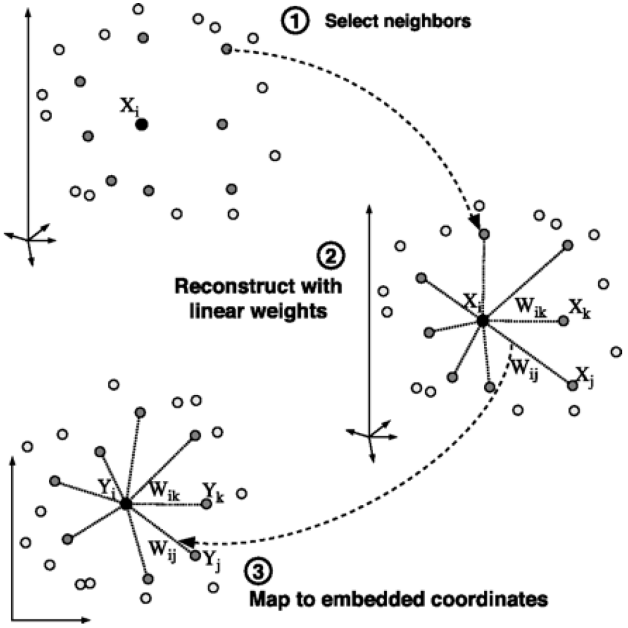
$$\text{subject to} \begin{cases} w_{ij} = 0 & \text{if } x_j \text{ is not one of the k n.n. of } x_i \\ \sum_{j=1}^{m} w_{ij} = 1 & (i = 1 \ldots m) \end{cases}$$

LLE Pseudocode

NTB Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

```
1. Find neighbours in X space [b,c].
for i=1:N
     compute the distance from Xi to every other point Xj
     find the K smallest distances
     assign the corresponding points to be neighbours of Xi
```

```
2. Solve for reconstruction weights W.
for i=1:N create matrix Z consisting of all
     neighbours of Xi[d]
     subtract Xi from every column of Z
     compute the local covariance C=Z'*Z [e]
     solve linear system C*w = 1 for w [f]
     set Wij=0 if j is not a neighbor of i
     set the remaining elements in the ith
     row of W equal to w/sum(w);
```




```
3. Compute embedding coordinates Y using weights W.
create sparse matrix M = (I-W)'*(I-W)
find bottom d+1 eigenvectors of M (corresponding to the d+1 smallest
eigenvalues) set the qth ROW of Y to be the q+1 smallest eigenvector
(discard the bottom eigenvector [1,1,1,1...] with eigenvalue zero)
```

https://cs.nyu.edu/~roweis/lle/publications.html

$$\phi(Y) = \sum_i \left(y_i - \sum_{j=1}^{k} w_{ij}\, y_j \right)^2$$

3.2 Isomap (isometric feature mapping)

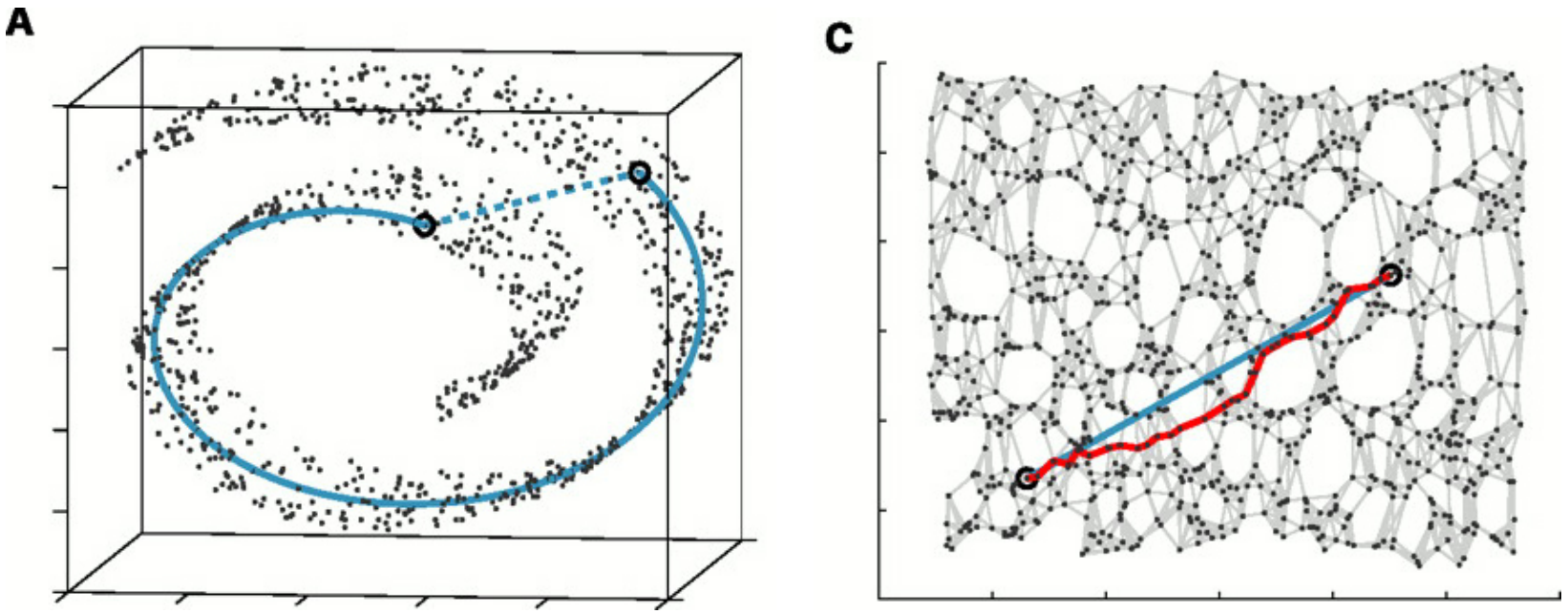
- A **nonlinear method** for dimensionality reduction
- Finds the map that preserves the global, nonlinear geometry of the data by preserving the **geodesic manifold interpoint distances**
- Geodesic: Shortest curve along the manifold connecting two points
- First approximates the geodesic interpoint distances, then runs MDS to find the projection that preserves these distances.
- *Isomap uses the same basic idea as PCA, the difference being that linearity is only preserved locally (via small neighborhoods).*

The basic steps are:

1. For each object, find a small set of neighboring objects and their distances.
2. Compute *all-pairs shortest paths* on the above neighborhood graph.
3. Run *multidimensional scaling* using the matrix of shortest-path distances.

http://science.sciencemag.org/content/290/5500/2319.full

3.3 Euclidian vs. Geodesic Distance



- **graph geodesic:** In the field of graph theory, the distance between two vertices in a graph is the number of edges in a shortest path connecting them. This is also known as the geodesic distance.

3.4 Calculation Effort

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

Step	Name	Description
1 $O(DN^2)$	Construct neighborhood graph, **G**	Compute matrix $\mathbf{D}_G=\{d_X(i,j)\}$ $d_x(i,j)$ = Euclidean distance between neighbors
2 **$O(DN^2)$**	Compute shortest paths between ***all*** pairs	Compute matrix $\mathbf{D}_G=\{d_G(i,j)\}$ $d_G(i,j)$ = sequence of hops = approx geodesic dist.
3 $O(dN^2)$	Construct k-dimensional coordinate vectors	Apply MDS to $\mathbf{D}_G$ instead of $\mathbf{D}_X$

3.5 Advantages and disadvantages of Isomap

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

Advantages:

- Nonlinear
- Non-iterative
- Globally optimal
- Parameters: k or ε (chosen fixed radius)

Disadvantages:

- Graph discreteness overestimates the geodesic distance
- k must be high to avoid “linear shortcuts” near regions of high surface curvature

4. t-SNE (t-distr. Stochastic neighbour embedding)

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

- t-SNE, is a **manifold** learning algorithm that in essence constructs a probability distribution p over the dataset X, and then another probability distribution q in a lower dimensional data space Y, making both the distribution as “close” as possible.
- Is well suited for the visualization of high-dimensional datasets.
- t-SNE uses a tuneable parameter, “**perplexity**,” which says (loosely) how to balance attention between local and global aspects of your data. The parameter is, in a sense, a guess about the number of close neighbors each point has
- t-SNE is incredibly flexible, and can often find structure where other dimensionality-reduction algorithms cannot. Unfortunately, that very flexibility makes it tricky to interpret.

https://distill.pub/2016/misread-tsne/
https://lvdmaaten.github.io/tsne/

4.1 How it works

- **Step 1:** In the high-dimensional space X, create a probability distribution $p_{i|j}$ that dictates the relationships between various neighboring points x_i and x_j
- **Step 2:** It then tries to recreate a probability distribution $q_{i|j}$ in a low dimensional space Y that follows that probability distribution $p_{i|j}$ as best as possible (KL-divergence as loss function).

- The “t” in t-SNE comes from the **t-distribution**, which is the distribution used in Step 2. The “S” and “N” (“stochastic” and “neighbor”) come from the fact that it uses a probability distribution across neighboring points.
- before looking into t-SNE, we will investigate SNE, which uses *Gaussian distribution* instead of *Student-t* distribution.

4.1 How it works (SNE)

- we construct a probability distribution q in the lower dimensional space using the following formula for pairwise points y.

$$p_{i|j} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)} \qquad q_{i|j} = \frac{e^{-\|y_i - y_j\|^2}}{\sum_{k \neq i} e^{-\|y_i - y_k\|^2}}$$

- The goal is to make sure the two probability distributions are as similar as possible. This is achieved by considering the **Kullback-Leibler (KL) divergence**. KL divergence is a measure of how different two probability distributions are from one another:

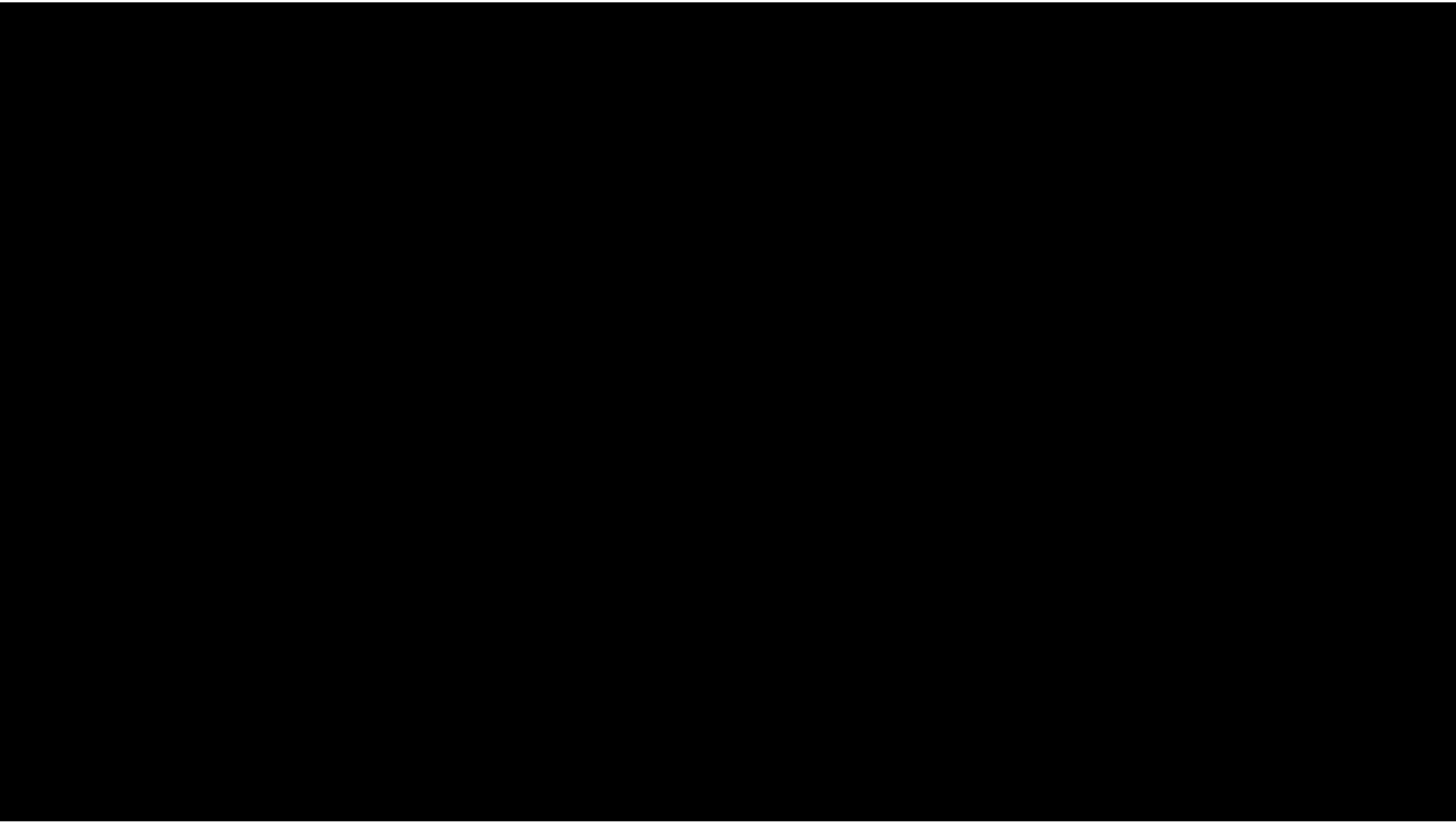
$$\mathrm{KL}(P|Q) = \sum_{i \neq q} p_{i|j} \ln\left(\frac{p_{i|j}}{q_{i|j}}\right)$$

- In essence, the KL divergence is the expected value of the log difference of the probabilities of the data points. Now that we have a notion of how to measure the similar/difference between distributions, we can optimize this by well known gradient descent methods.

A.I. Experiments: Visualizing High-Dimensional Space

https://experiments.withgoogle.com/ai

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz



Excercises (Aurélien Géron)

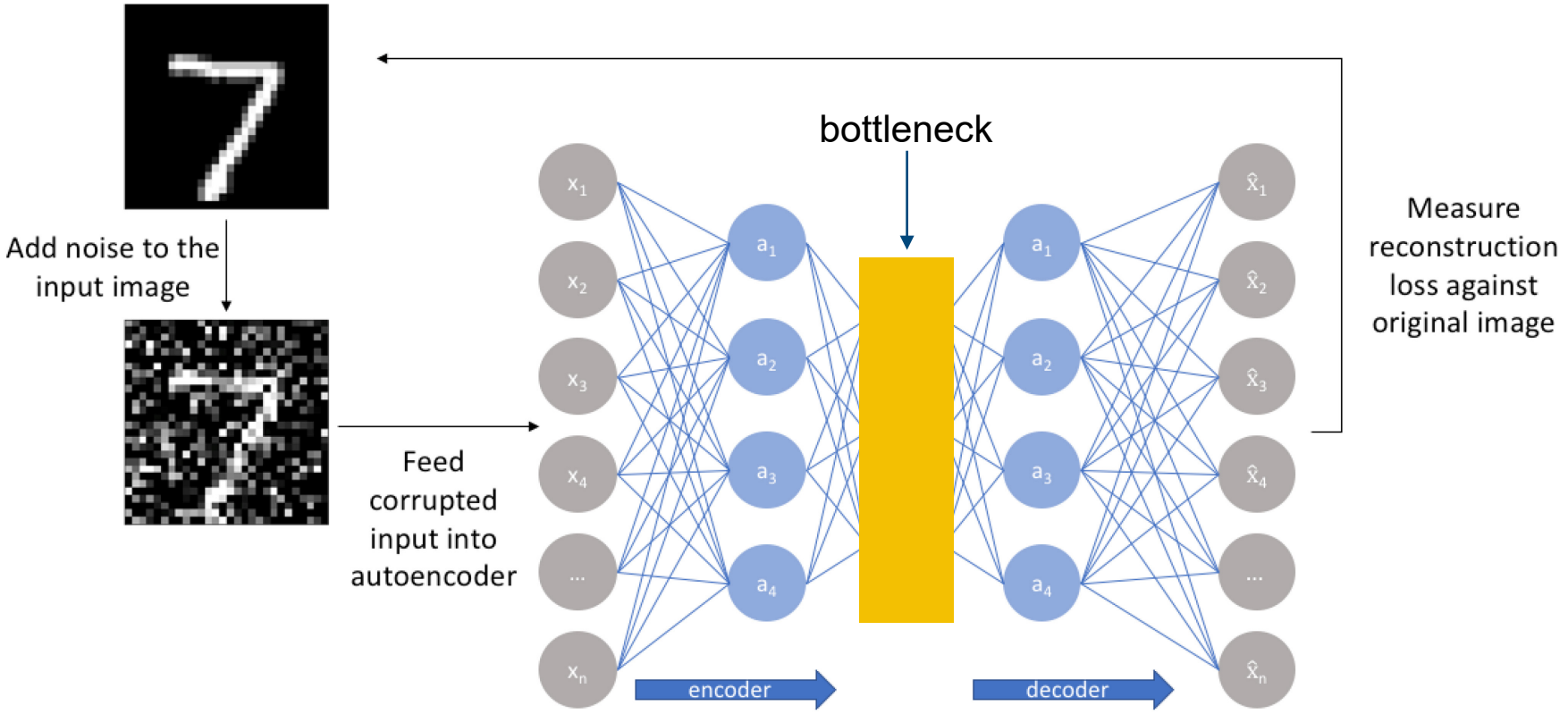
NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

1. What are the main motivations for reducing a dataset's dimensionality? What are the main applications? What are the main drawbacks?
2. What is the curse of dimensionality?
3. Once a dataset's dimensionality has been reduced, is it possible to reverse the operation? If so, how? If not, why?
4. Can PCA be used to reduce the dimensionality of a highly nonlinear dataset? Which methods can alternatively be used?
5. Suppose you perform a PCA on a 1000-dimensional dataset, setting the explained variance to 95%. How many dimensions will the resulting dataset have?
6. In which cases would you use incremental PCA, randomized PCA or kernel PCA?
7. How can you evaluate the performance of a dimensionality reduction algorithm?
8. Does it make sense to chain two different dimensionality reduction algorithms?

Autoencoder for dimensionality reduction

- **Definition: An autoencoder is a feed-forward neural network which is trained to approximate the identity function**. That is, it is trained to map from a vector of values to the same vector.
- When used for dimensionality reduction purposes, *one of the hidden layers in the network is limited to contain only a small number of network units*. Thus, the network must learn to encode the vector into a small number of dimensions and then decode it back into the original space.
- Thus, the first half of the network is a model which maps from high to low-dimensional space, and the second half maps from low to high-dimensional space. Although the idea of autoencoders is quite old, training of deep autoencoders has only recently become possible through the use of restricted Boltzmann machines and stacked denoising autoencoders.

Principle of an Autoencoder



Source: https://www.jeremyjordan.me/autoencoders

Summary PCA & Dimensionality Reduction

- **PCA** performs a **linear mapping** of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. In practice, the covariance (and sometimes the correlation) matrix of the data is constructed and the eigenvectors on this matrix are computed.
- PCA can be employed in a nonlinear way by means of the **kernel trick**. The resulting technique, kernel PCA, is capable of constructing **nonlinear mappings** that maximize the variance in the data.

Advantages of Dimensionality Reduction:

- It reduces the *time* and *storage space* required.
- *Removal of multi-collinearity* improves the interpretation of the parameters of the machine learning model.
- It becomes easier to *visualize* the data when reduced to very low dimensions such as 2D or 3D.
- It avoids the *curse of dimensionality* (e.g. for k-means).