Chapter 5 - Unsupervised Learning

Dimensionality Reduction, Clustering and Density Estimation

Machine Learning -
Foundations and Algorithms WS21/22

Prof. Gerhard Neumann

KIT, Institut für Anthrophomatik und Robotik

Wrap-up: Where are we?

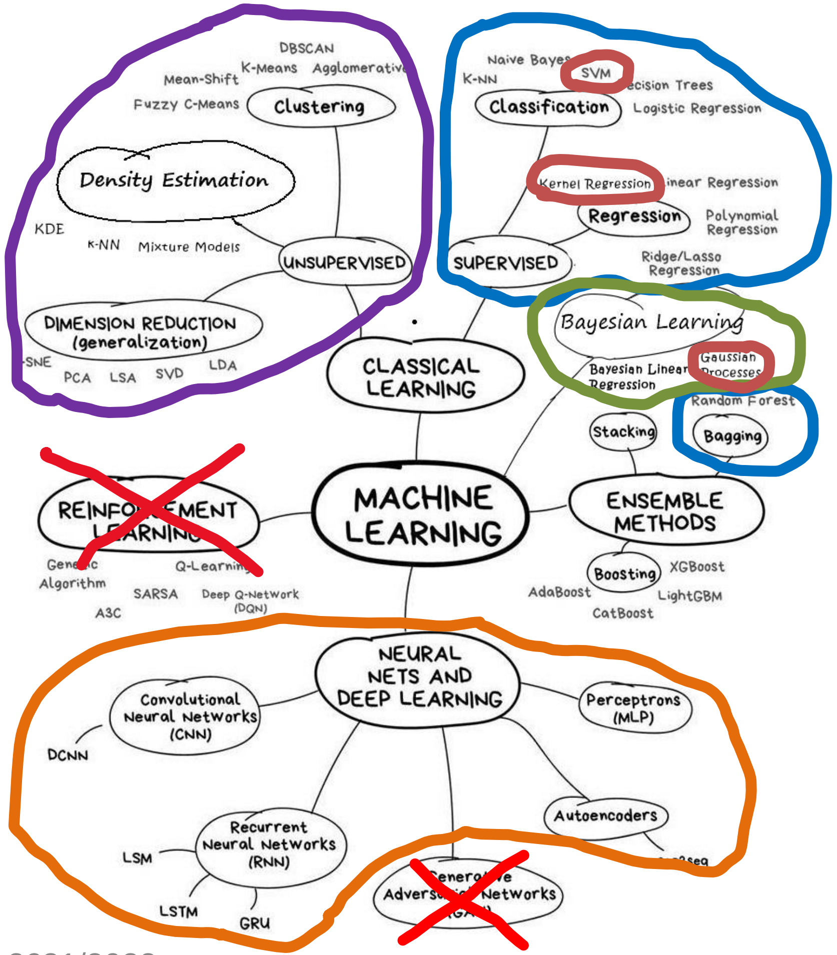
Chapter 1: Classical Supervised Learning
- Lecture 1: Linear Regression, Ridge Regression
- Lecture 2: Linear Classification
- Lecture 3: Model Selection
- Lecture 4: k-Nearest Neighbors, Trees and Forests

Chapter 2: Kernel Methods
- Lecture 5: Kernel-Regression
- Lecture 6: Support Vector Machines

Chapter 3: Bayesian Learning
- Lecture 7: Bayesian Linear Regression and Gaussian Processes

Chapter 4: Neural Networks
- Lecture 8: Neural Networks and Backpropagation
- Lecture 9: CNNs and LSTMs

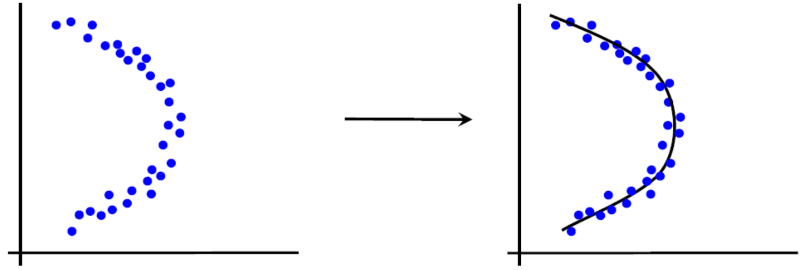
Chapter 5: Unsupervised Learning
- Lecture 10: Dimensionality Reduction, Clustering and Density Estimation
- Lecture 11: Expectation Maximization
- Lecture 12: Variational Auto-Encoders



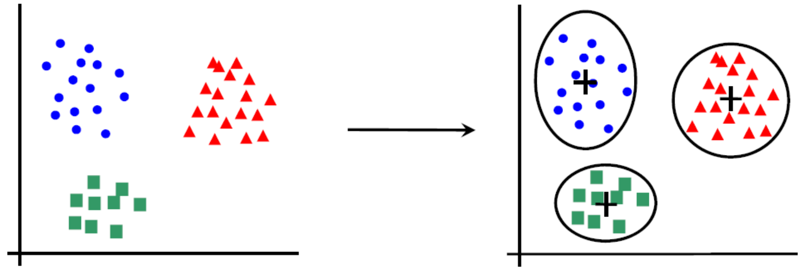
Unsupervised Learning

Trainings data does not include target values, find "structure" in the data

(1) Dimensionality reduction:



(2) Clustering



(3) Density estimation: Generative model of the data

Dimensionality Reduction

Learning Outcomes

- Understand what dimensionality reduction means and why do use it
- Understand what we mean with a “projection” of a vector
- What makes a dimensionality reduction a “good” reduction
- What are the principal components in the data and what is the relation to the covariance matrix
- Learn about constraint convex optimization

Today‘s Agenda!

Dimensionality Reduction:

- Linear Dimensionality Reduction
- Linear Orthogonal Projections
- Reproduction Error
- Principal Component Analysis

Basics: Convex Constraint Optimization

- Lagrangian Multipliers and Constraint Optimization
- Dual Optimization Problem

Slides are largely based on
Slides from Jan Peters

Dimensionality Reduction

Supervised Learning:

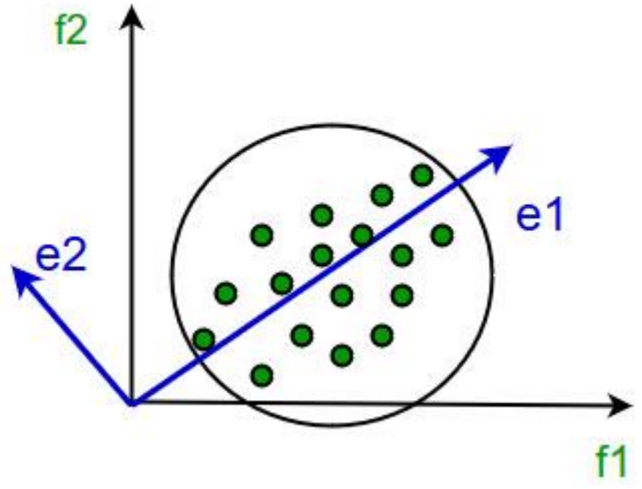
- Learn a mapping from input x to output y

Sometimes, it is quite helpful to analyze the data points themselves

- Unsupervised learning
- Particularly:
  - Reduce the dimensionality of the data

Possible application:

- Visualization of the data
- Preprocessing for any learning algorithm



Motivation from Linear Least-squares Regression

- In least-squares linear regression the parameters are computed as

  $$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

  where $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ and $\boldsymbol{y} \in \mathbb{R}^{n \times 1}$

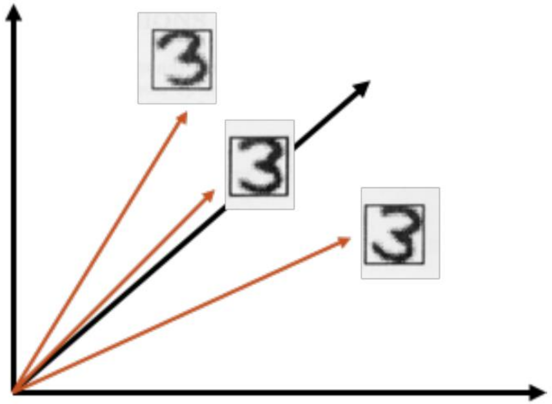
- We need to invert a d × d matrix, which naively costs $O(d^3)$

- Hence, it would be helpful to find a new d_{new} << d to gain computational advantage while not loosing prediction performance

Dimensionality Reduction

- How can we find more efficient representations for our data?
- How can we capture the "essence" of the data?

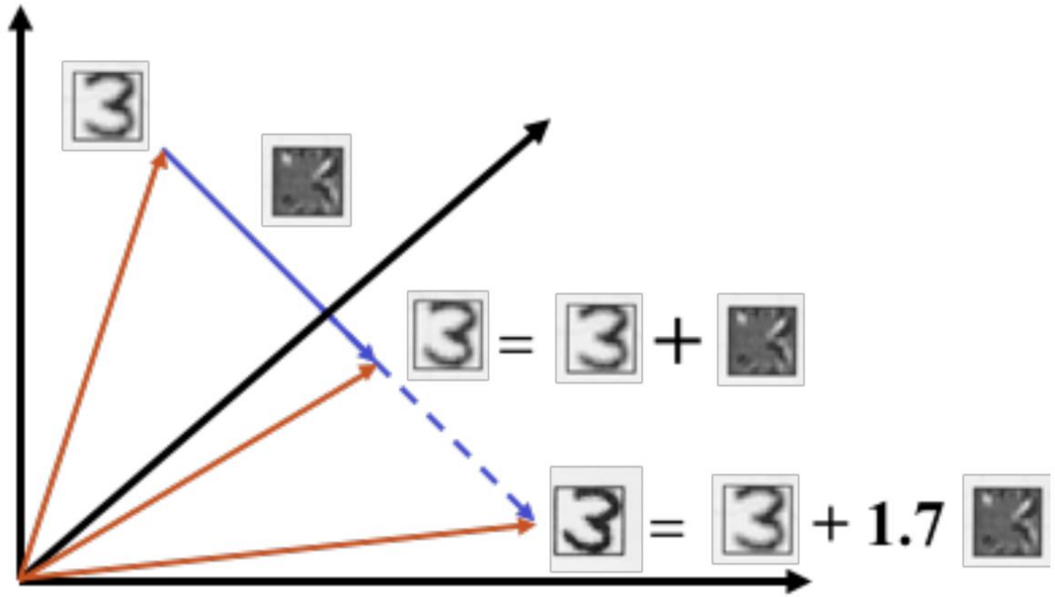
Example: images of the digit 3



- The images can be represented as points in a high-dimensional space (e.g., with one dimension per pixel, in a 4k image there are around 9 million dimensions!)

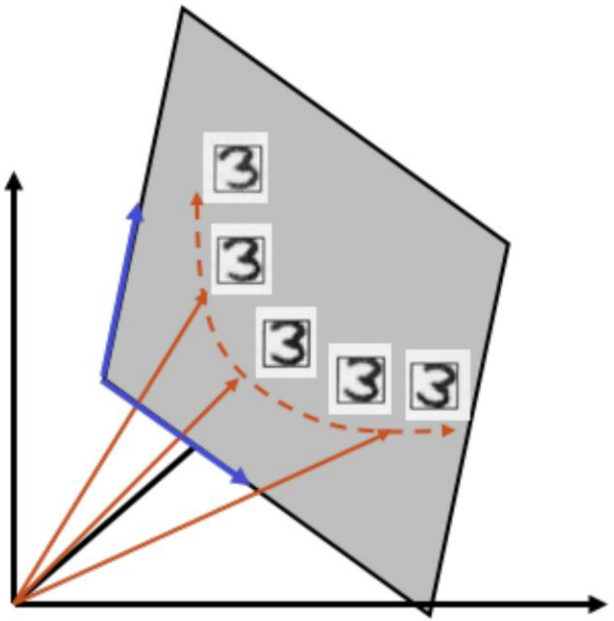
Linear Dimensionality Reduction

To make things easier, we will once again assume linear models. A data point (here: one image) can be written as a linear combination of bases (here: basis images)



Linear Dimensionality Reduction

- What linear transformations of the data can be used to define a lower-dimensional subspace that captures most of the structure?



Linear Dimensionality Reduction

Problem definition:

- Original data point i: $\boldsymbol{x}_i \in \mathbb{R}^D$
- Low-dimensional representation of data point i: $\boldsymbol{z}_i \in \mathbb{R}^M$ with D >> M
- **Goal:** find a mapping

$$\boldsymbol{x}_i \to \boldsymbol{z}_i$$

- Restrict this mapping to be a linear function

$$\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{x}_i, \text{ with } \boldsymbol{W} \in \mathbb{R}^{M \times D}$$

Orthonormal Basis Vectors

We can always write a vector in terms of an orthonormal basis coordinate system

$$\boldsymbol{x} = \sum_{i=1}^{D} z_i \boldsymbol{u}_i, \text{ where } \boldsymbol{u}_i^T \boldsymbol{u}_j = \delta_{ij} \text{ and } \delta_{ij} = 1 \text{ if } i = j, \ 0 \text{ otherwise}$$

- **Orthonormality condition:** The product of 2 different basis vectors is 0. The norm of each basis vector is 1.

Example:

$$\begin{bmatrix} 3 \\ 7 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 7 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Projections with an orthonormal basis

The coefficients z_i can be obtained by projecting **x** on the basis vector u_i

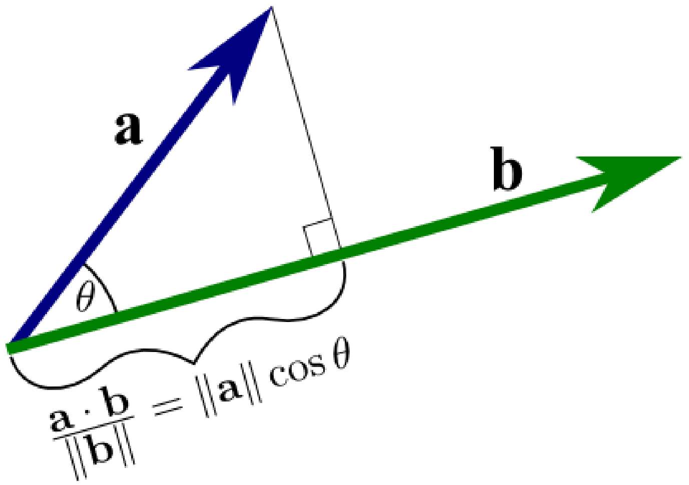
$$\underbrace{z_i}_{\text{scalar coefficient}} = \underbrace{\boldsymbol{u}_i^T \boldsymbol{x}}_{\text{projection}}$$

Example:

$$\boldsymbol{x} = z_1 \boldsymbol{u}_1 + z_2 \boldsymbol{u}_2$$

$$\boldsymbol{u}_1^T \boldsymbol{x} = z_1 \underbrace{\boldsymbol{u}_1^T \boldsymbol{u}_1}_{=1} + z_2 \underbrace{\boldsymbol{u}_2^T \boldsymbol{u}_1}_{=0} = z_1$$

Projection of 2 vectors


$\mathbf{a}$
$\mathbf{b}$
θ
$\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{b}\|} = \|\mathbf{a}\| \cos\theta$

Decomposition

Use M << D basis vectors:

$$\boldsymbol{x} = \underbrace{\sum_{i=1}^{M} z_i \boldsymbol{u}_i}_{\tilde{\boldsymbol{x}} \approx \boldsymbol{x}} + \underbrace{\sum_{j=M+1}^{D} z_j \boldsymbol{u}_j}_{\text{skip}}$$

Find the M basis vectors u_i that minimize the mean squared reproduction error:

$$\underset{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_M}{\arg\min}\, E(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_M) = \underset{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_M}{\arg\min} \sum_{i=1}^{N} ||\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i||^2$$

Minimizing the error

Assuming a single basis vector, the error can be written as

$$E(\boldsymbol{u}_1) = \sum_{i=1}^{N} ||\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i||^2 = \sum_{i=1}^{N} ||\boldsymbol{x}_i - \underbrace{(\boldsymbol{u}_1^T \boldsymbol{x}_i)}_{z_{i1}} \boldsymbol{u}_1||^2$$

$$= \sum_{i=1}^{N} \boldsymbol{x}_i^T \boldsymbol{x}_i - 2(\boldsymbol{u}_1^T \boldsymbol{x}_i)^2 + (\boldsymbol{u}_1^T \boldsymbol{x}_i)^2 \boldsymbol{u}_1^T \boldsymbol{u}_1 = \sum_{i=1}^{N} \boldsymbol{x}_i^T \boldsymbol{x}_i - (\boldsymbol{u}_1^T \boldsymbol{x}_i)^2$$

$$= \sum_{i=1}^{N} \boldsymbol{x}_i^T \boldsymbol{x}_i - z_{i1}^2$$

Minimizing the error

The error can be written as

$$E(\boldsymbol{u}_1) = \sum_{i=1}^{N} \boldsymbol{x}_i^T \boldsymbol{x}_i - z_{i1}^2$$

$$\Rightarrow \arg\min_{\boldsymbol{u}_1} E(\boldsymbol{u}_1) = \arg\max_{\boldsymbol{u}_1} \sum_{i=1}^{N} z_{i1}^2 = \arg\max_{\boldsymbol{u}_1} \sum_{i=1}^{N} (\boldsymbol{u}_1^T \boldsymbol{x}_i)^2$$

- Minimizing the error is equivalent to maximizing the variance of the projection. (Assuming a zero mean on the data)
- We can ensure a zero mean projection by subtracting the mean from the data

$$\bar{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{\mu}$$

Principle component analysis

- The first principal direction **u₁** is the direction along which the variance of the projected data is maximal

$$\boldsymbol{u}_1 = \arg\max_{\boldsymbol{u}} \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{u}^T \underbrace{(\boldsymbol{x}_i - \boldsymbol{\mu})}_{\bar{\boldsymbol{x}}_i} \right)^2 \quad \text{s.t. } \boldsymbol{u}^T \boldsymbol{u} = 1$$

  – The directions all have unit norm

- The second principal direction maximizes the variance of the data in the orthogonal complement of the first principal direction

Derivation…

- **Objective in matrix form…**

$$E(\boldsymbol{u}) = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{u}^T(\boldsymbol{x}_i - \boldsymbol{\mu})\right)^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{u}^T(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T\boldsymbol{u}\right)$$

$$= \boldsymbol{u}^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T\right)}_{\text{covariance } \boldsymbol{\Sigma}} \boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u}$$

  - The objective can be written in terms of the sample covariance!

Back to the PCA Derivation…

We obtain the following constrained optimization problem

$$\boldsymbol{u}_1 = \arg\max_{\boldsymbol{u}} \ \boldsymbol{u}^T \boldsymbol{\Sigma} \boldsymbol{u} \quad \text{s.t. } \boldsymbol{u}^T \boldsymbol{u} = 1$$

- We now know what to do… Lagrangian optimization

Optimality condition for u:

$$\boldsymbol{\Sigma} \boldsymbol{u} = \lambda \boldsymbol{u}$$

This is an Eigen-value problem!

Basics: Eigenvalues and Eigenvectors

- Let the Eigenvectors and Eigenvalues of **C** be **u$_k$** and λ_k for $k \leq D$ i.e.,

  $$\boldsymbol{C}\boldsymbol{u}_k = \lambda_k \boldsymbol{u}_k \quad \text{with } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$$ Ordered list of Eigenvalues

- In matrix form:

  $$\boldsymbol{C}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{\Lambda} \quad \text{with } \boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_D) \text{ and } \boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D]$$

- Because **U** is orthonormal (eigenvectors have unit norm), we know that $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}$

- This means that we can decompose **C** as

  $$(\boldsymbol{C}\boldsymbol{U})\boldsymbol{U}^T = (\boldsymbol{U}\boldsymbol{\Lambda})\boldsymbol{U}^T \Rightarrow \boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$$

Basics: Eigenvalues and Eigenvectors

Every positive definite symmetric matrix can be decomposed in its Eigendecomposition

$$\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \underbrace{\begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_D \end{bmatrix}}_{\text{Eigenvectors}} \underbrace{\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_D \end{bmatrix}}_{\text{Eigenvalues}} \begin{bmatrix} \boldsymbol{u}_1^T \\ \vdots \\ \boldsymbol{u}_D^T \end{bmatrix}$$

Back to PCA

Eigenvalues-Eigenvectors of the covariance matrix

$$\mathbf{\Sigma} \boldsymbol{u} = \lambda \boldsymbol{u}$$

- The largest Eigenvalue gives us the maximal variance
- The corresponding Eigenvector gives us the direction with maximal variance

Principal Component Analysis

- **Observation:** If $\lambda_k \approx 0$ for $k > M$ for some $M << D$, then we can use the subset of the first D eigenvectors to define a basis for approximating the data vectors with loosing accuracy

$$\boldsymbol{x}_i - \boldsymbol{\mu} = \underbrace{\sum_{j=1}^{M} z_{ij}\boldsymbol{u}_j}_{\tilde{\boldsymbol{x}}} + \underbrace{\sum_{j=M+1}^{D} z_{ij}\boldsymbol{u}_j}_{\text{close to } 0} \Rightarrow \boldsymbol{x}_i \approx \boldsymbol{\mu} + \sum_{j=1}^{M} z_{ij}\boldsymbol{u}_j$$

- This representation has the minimal mean squared error (MSE) of all linear representations of dimension D

$$\underset{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_M}{\arg\min}\, E(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_M) = \underset{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_M}{\arg\min} \sum_{i=1}^{N} ||\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i||^2$$

Principal Component Analysis

Now we know how we can represent our data in a lower dimensional space in a principled way

- Center the data around the mean (compute the mean of the data and subtract it)
- Compute the covariance matrix, decompose it, and choose the first M largest Eigenvalues and corresponding Eigenvectors
- This gives us an (Eigen)basis for representing the data
  - **Projection to low-D:** $$\boldsymbol{z}_i = \boldsymbol{W}^T(\boldsymbol{x}_i - \boldsymbol{\mu})$$
  - **Reprojection to high-D:** $$\tilde{\boldsymbol{x}}_i = \boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}_i$$

  with $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_M \end{bmatrix}$
- It is also common to normalize the variance of each dimension (i.e. unit variance)

How to choose M

- A larger M leads to a better approximation. In the limit, when M = D we stay in the initial data dimensions
- There are at least 2 good possibilities for choosing M
  - Choose D based on application performance, i.e. choose the smallest M that makes the application work well enough
  - Choose D so that the Eigenbasis captures some fraction of the variance (for example η = 0.9).
    The eigenvalue λ_i describes the marginal variance captured by $\mathbf{u}_i$

$$\text{Choose } D \text{ s.t. } \sum_{i=1}^{M} \lambda_i = \eta \underbrace{\sum_{i=1}^{D} \lambda_i}_{\text{Total variance of the data}}$$

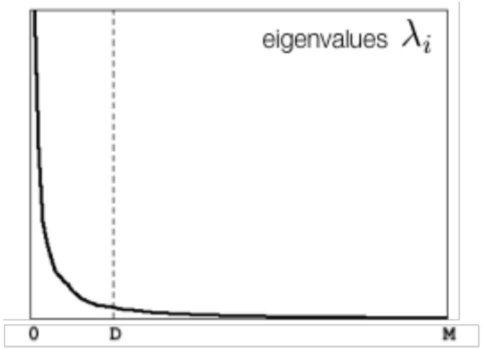


Image representation with PCA

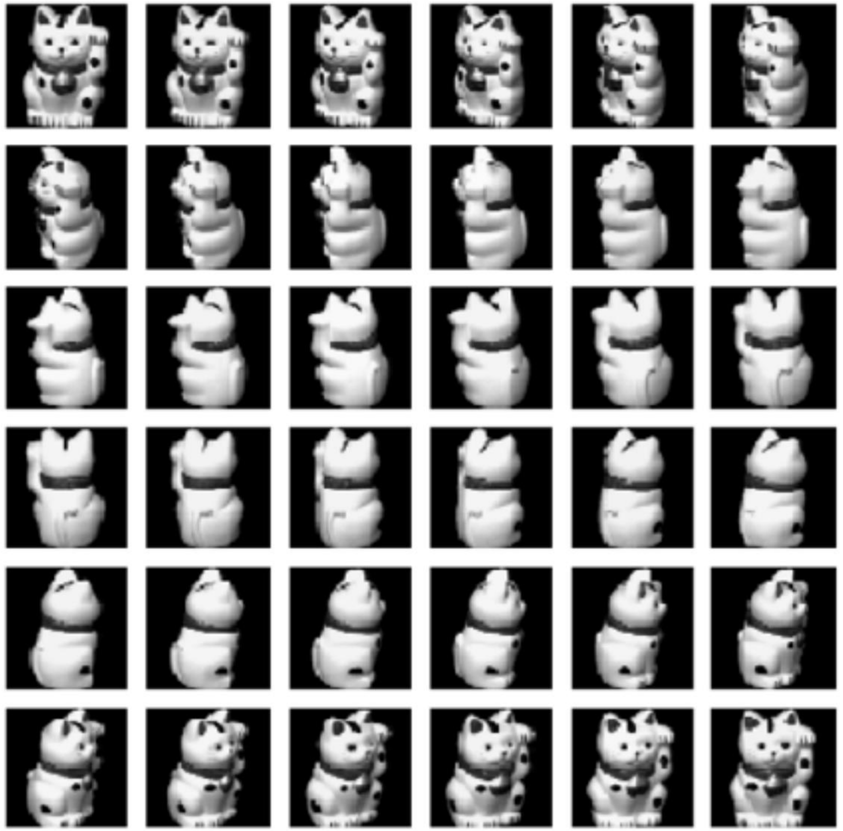
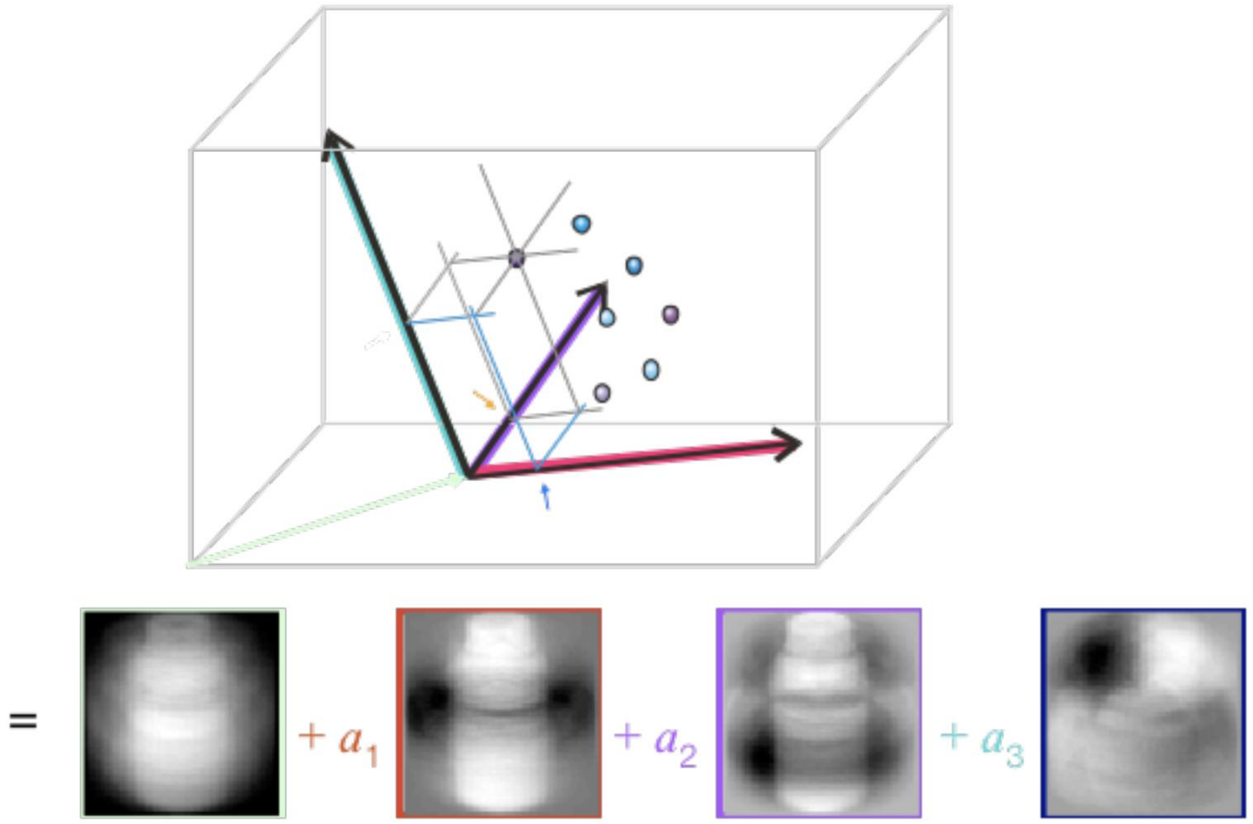


Image representation with PCA



Eigenfaces

- The first popular use of PCA for object recognition was for the detection and recognition of faces [Turk and Pentland, 1991]
- Collect a face ensemble
- Normalize for contrast, scale, & orientation
- Remove backgrounds
- Apply PCA & choose the first M eigen-images that account for most of the variance of the data

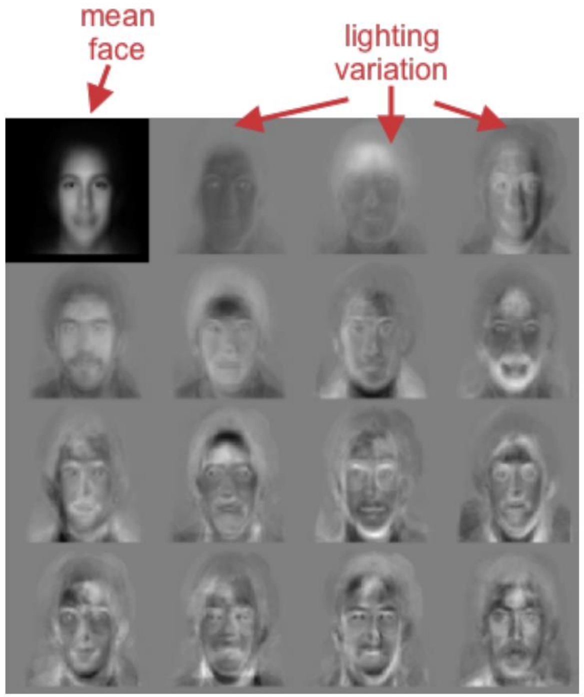
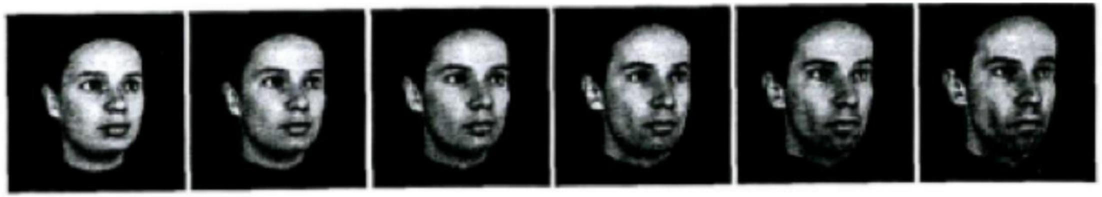
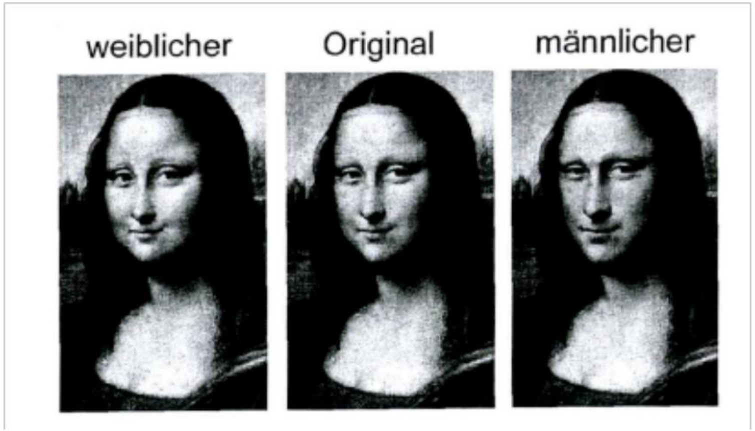


Image Morphing with PCA



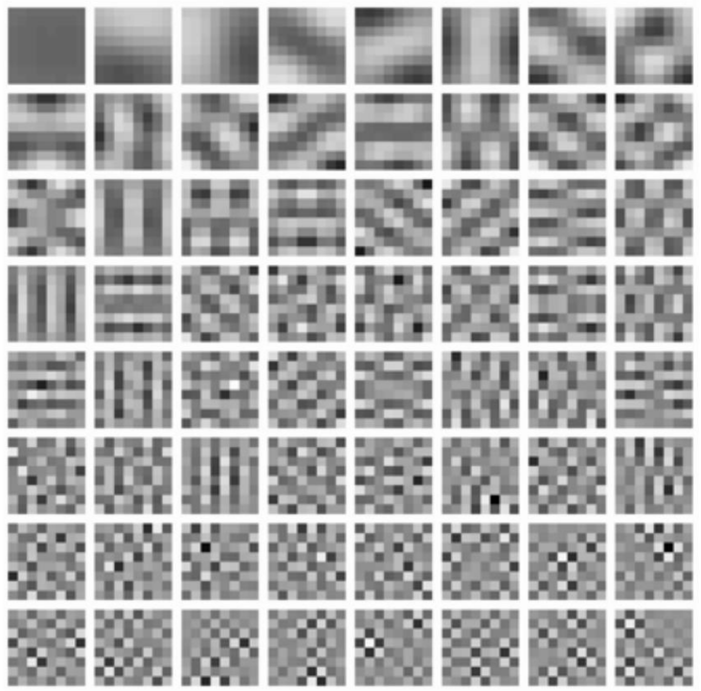


Generic Image Ensembles

Is there a low-dimensional model describing natural images?

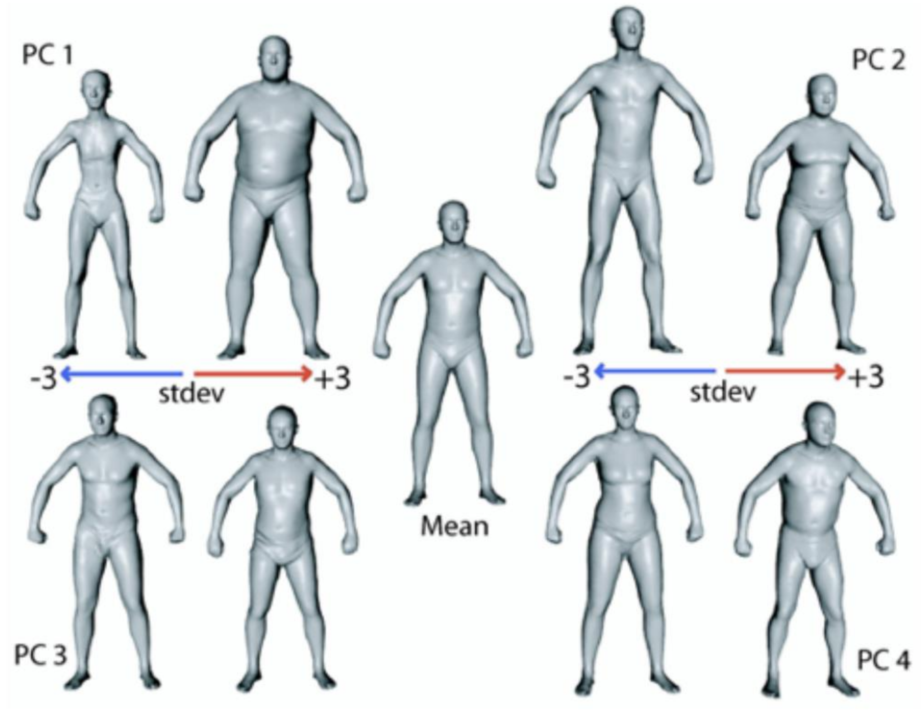
PCA of natural image patches

8x8 image patches



PCA Model of body shapes

- PCA on a detailed triangle model of human bodies [Anguelov et al. 05]



Wrap-up

Summary:

- PCA projects the data into a linear subspace
- PCA maximizes the variance of the projection
- PCA minimizes the error of the reconstruction
- We just covered the most simple linear dimensionality reduction technique
  - Many more sophisticated techniques exist
  - Kernel PCA, Auto-Encoders, t-SNE, non-negative matrix factorization (interesting, but no time to cover those...)

Applications:

- PCA allows us to transform a high-dimensional input space to a low-dimensional feature space, while capturing the essence of the data
- PCA finds a more natural coordinate system for the data
- PCA is a very common preprocessing step for high-dimensional input data

Self-test questions

What have we learned today?

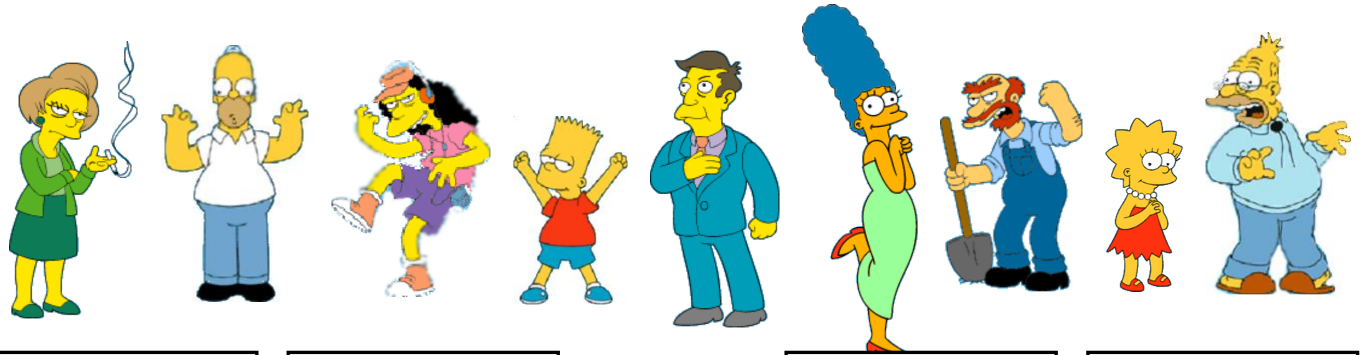
- What does dimensionality reduction mean?
- How does linear dimensionality reduction work?
- What is PCA? What are the three things that it does?
- What are the roles of the Eigenvectors and Eigenvalues in PCA?
- Can you describe applications of PCA?

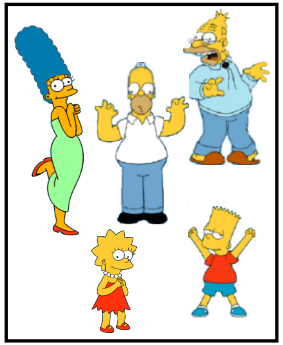
Clustering

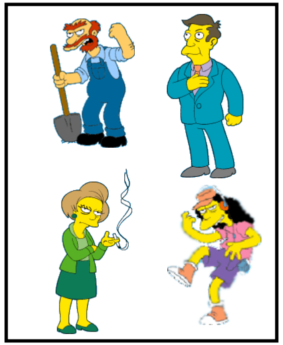
Slides based on lectures from Stefan Roth (TU Darmstadt)
and Carla Brodley (Tulft University)

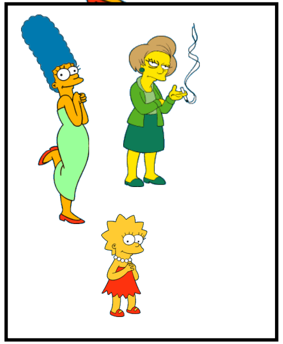
Clustering is Subjective

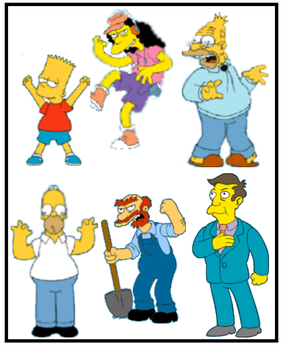
What is the natural grouping of these objects?




Simpson's Family


School Employees


Females


Males

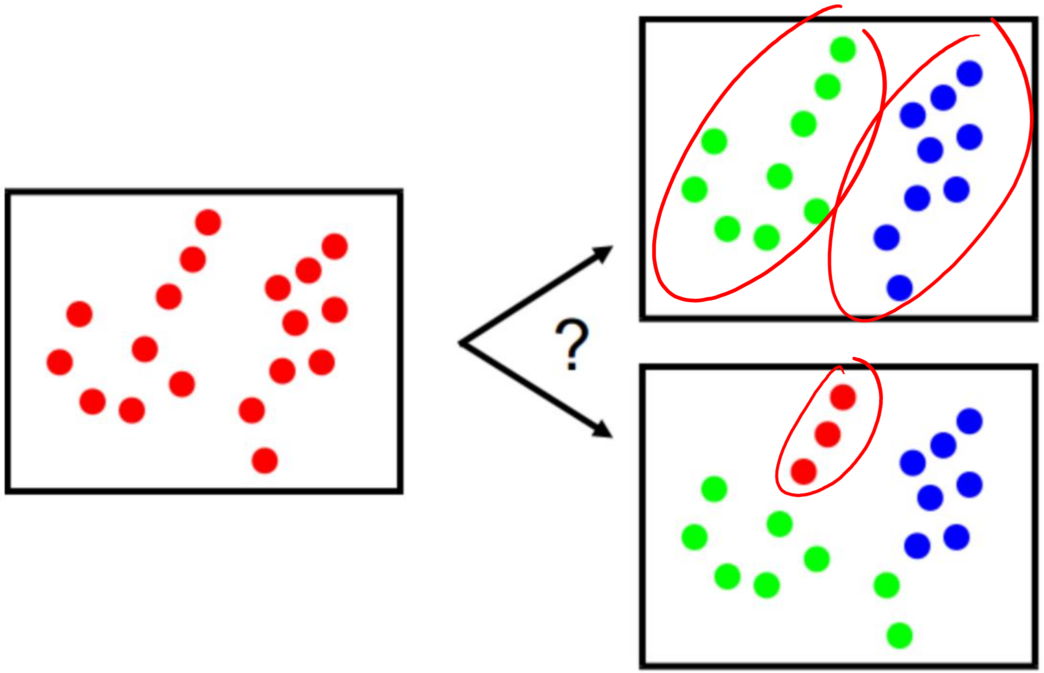
Clustering: Finding structure in the data

What are the correct clusters?

- Ground truth often not available

Similarity measure

- clustering relies on measure of similarity
- e.g. position in space (Euclidean vs. log-polar coordinates), weighting of different dimensions (features)...

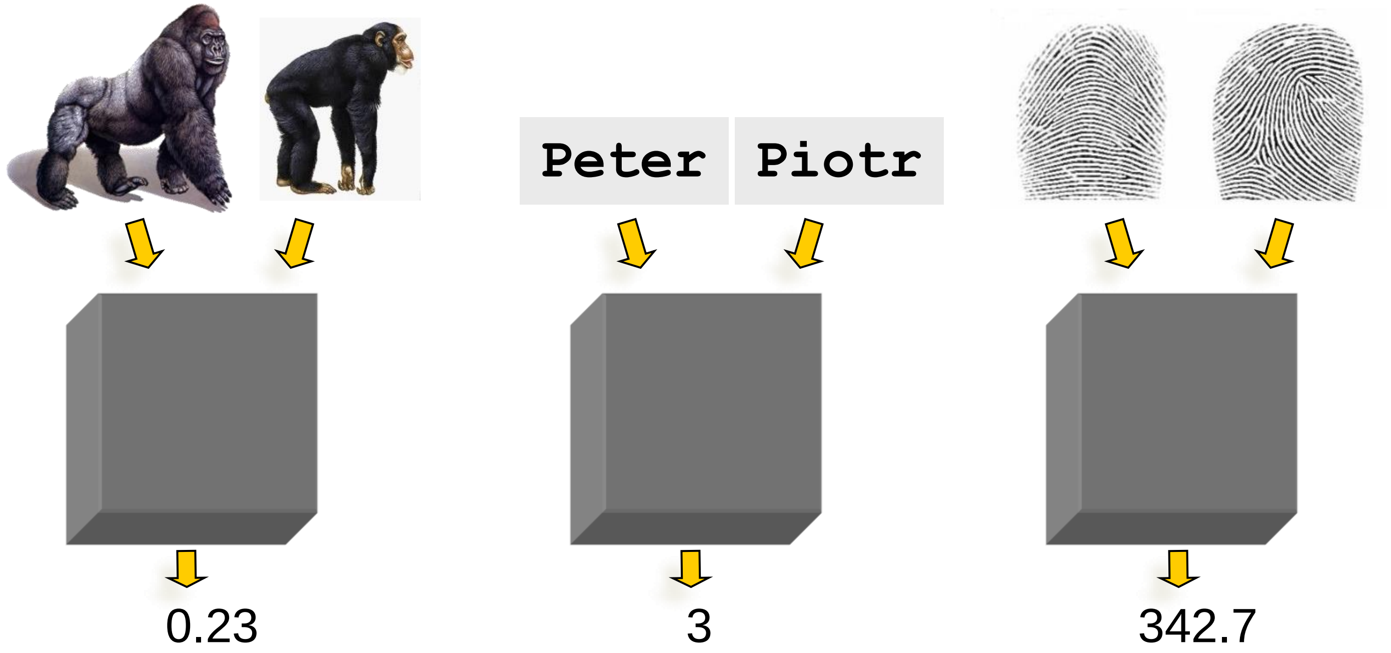


What is similarity?

Similarity is hard to define, but…
“*We know it when we see it*”

Defining Distance Measures

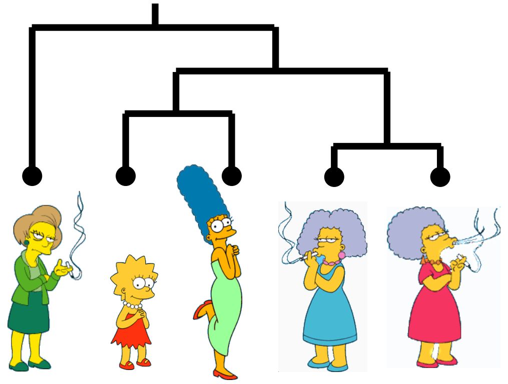
- **Definition**: Let O_1 and O_2 be two objects from the universe of possible objects. The distance (dissimilarity) between O_1 and O_2 is a real number denoted by $D(O_1,O_2)$



Basic Clustering Algorithms

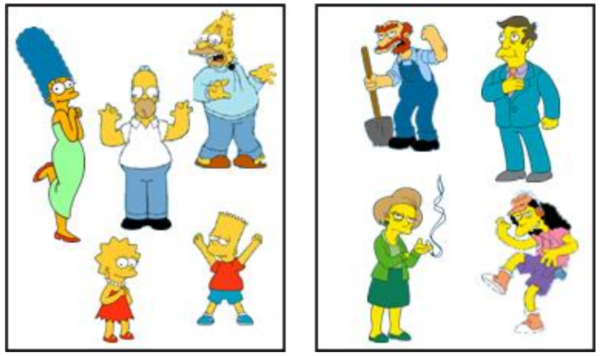
Hierarchical clustering methods (not covered)

- Bottom-up (merging)
- Top-down (splitting)



Flat clustering algorithms

- K-Means
- Mixture models (see density estimation lecture)



Other clustering methods:

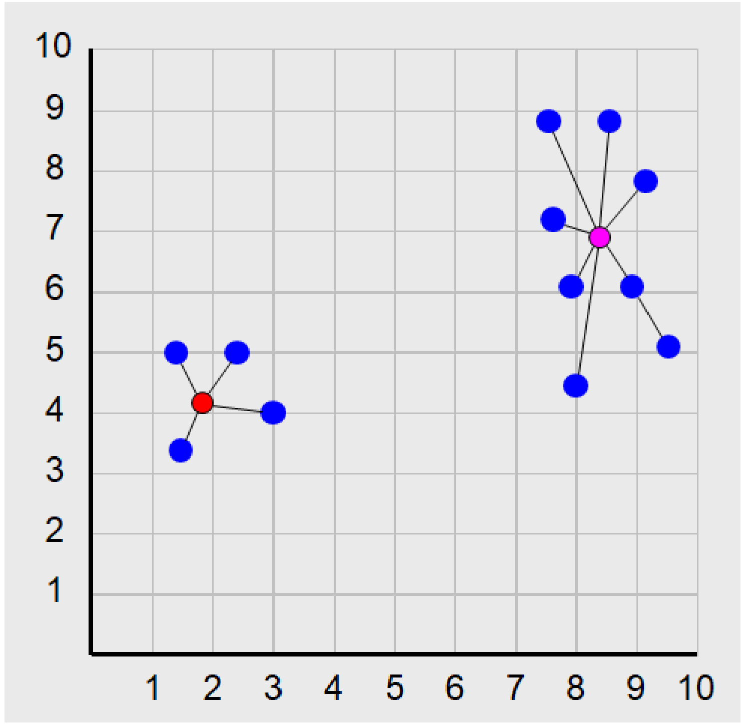
- Spectral clustering (not covered)

K-Means Algorithm

Goal: minimize quantization error!

- Given data $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$
- Search for cluster centers/prototypes/centroid $C = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k\}$
- Denote with c(x) the closest centroid vector $\boldsymbol{c} \in C$ to $\boldsymbol{x}$
- Sum of squared distances (SSD) (or sum of squared error) denotes quantization error

$$\mathrm{SSD}(C;\mathcal{D}) = \sum_{i=1}^{n} d(\boldsymbol{x}_i, c(\boldsymbol{x}_i))^2$$



K-Means Algorithm

Iterative Procedure

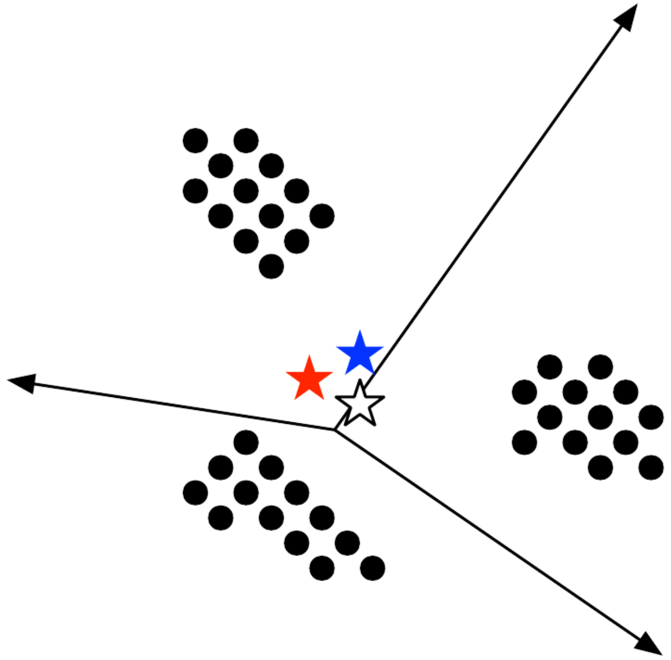
1. Pick K arbitrary cluster centers
2. Assign each sample to its closest centroid

$$z_n = \arg\min_k ||\boldsymbol{c}_k - \boldsymbol{x}_n||^2$$

3. Adjust the centroids to be the means of the samples assigned to them

$$\boldsymbol{c}_k = \frac{1}{|X_k|} \sum_{\boldsymbol{x}_i \in X_k} \boldsymbol{x}_i, \quad X_k = \{\boldsymbol{x}_n | z_n = k\}$$

4. Goto step 2 until no change



Step 1: The stars are cluster centers, randomly assigned at first.

K-Means Algorithm

Iterative Procedure

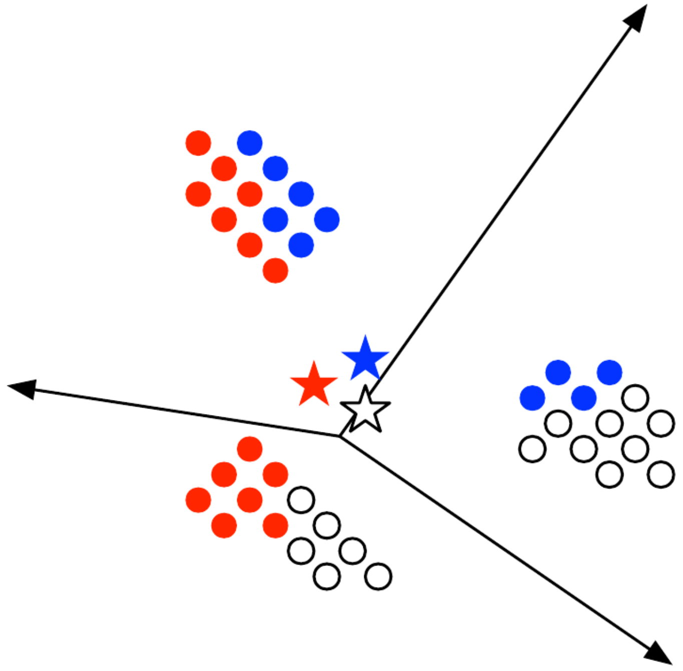
1. Pick K arbitrary cluster centers
2. Assign each sample to its closest centroid

$$z_n = \arg\min_k ||\boldsymbol{c}_k - \boldsymbol{x}_n||^2$$

3. Adjust the centroids to be the means of the samples assigned to them

$$\boldsymbol{c}_k = \frac{1}{|X_k|} \sum_{\boldsymbol{x}_i \in X_k} \boldsymbol{x}_i, \quad X_k = \{\boldsymbol{x}_n | z_n = k\}$$

4. Goto step 2 until no change



Step 2: Assign each example to its nearest cluster center.

K-Means Algorithm

Iterative Procedure

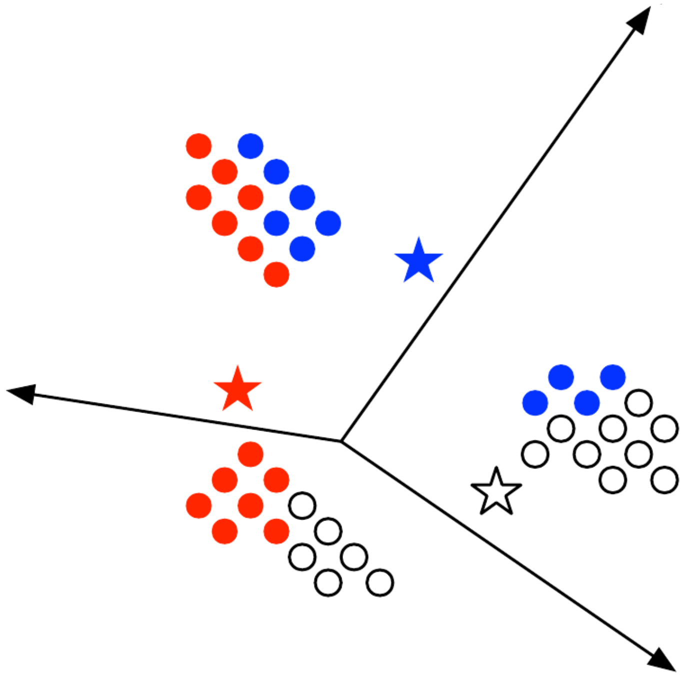
1. Pick K arbitrary cluster centers
2. Assign each sample to its closest centroid

$$z_n = \arg\min_k ||\boldsymbol{c}_k - \boldsymbol{x}_n||^2$$

3. Adjust the centroids to be the means of the samples assigned to them

$$\boldsymbol{c}_k = \frac{1}{|X_k|} \sum_{\boldsymbol{x}_i \in X_k} \boldsymbol{x}_i, \quad X_k = \{\boldsymbol{x}_n | z_n = k\}$$

4. Goto step 2 until no change



Step 3: Adjust the centroids to be the means of the samples assigned to them.

K-Means Algorithm

Iterative Procedure

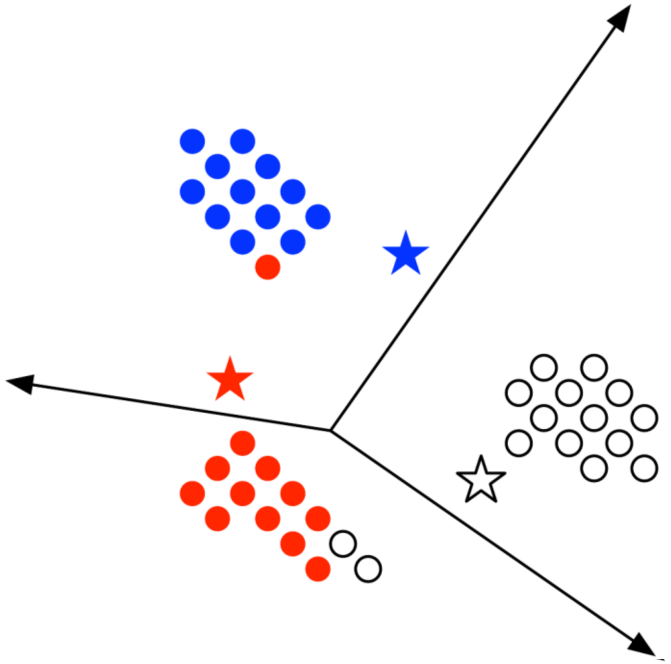
1. Pick K arbitrary cluster centers
2. Assign each sample to its closest centroid

$$z_n = \arg\min_k ||\boldsymbol{c}_k - \boldsymbol{x}_n||^2$$

3. Adjust the centroids to be the means of the samples assigned to them

$$\boldsymbol{c}_k = \frac{1}{|X_k|} \sum_{\boldsymbol{x}_i \in X_k} \boldsymbol{x}_i, \quad X_k = \{\boldsymbol{x}_n | z_n = k\}$$

4. Goto step 2 until no change



Step 4: Assign each example to its nearest cluster center.

K-Means Algorithm

Iterative Procedure

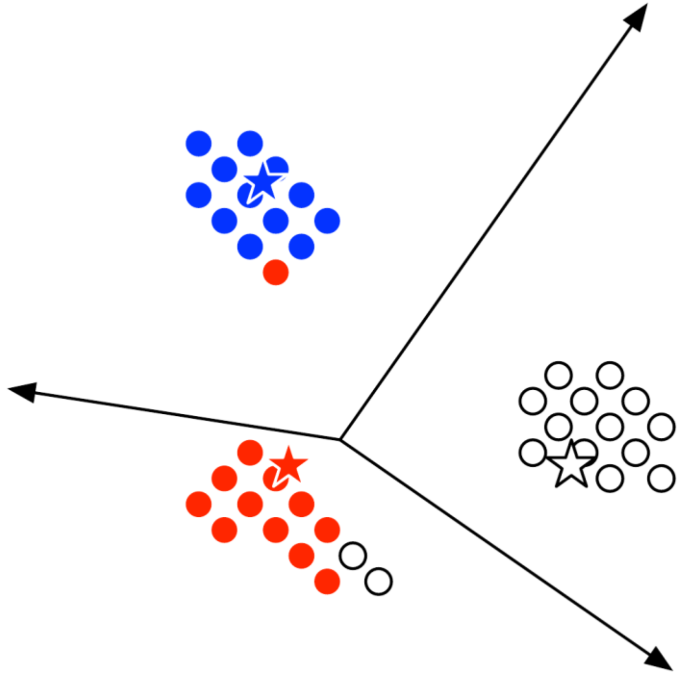
1. Pick K arbitrary cluster centers
2. Assign each sample to its closest centroid

$$z_n = \arg\min_k ||\boldsymbol{c}_k - \boldsymbol{x}_n||^2$$

3. Adjust the centroids to be the means of the samples assigned to them

$$\boldsymbol{c}_k = \frac{1}{|X_k|} \sum_{\boldsymbol{x}_i \in X_k} \boldsymbol{x}_i, \quad X_k = \{\boldsymbol{x}_n | z_n = k\}$$

4. Goto step 2 until no change



Step 5: Adjust the centroids to be the means of the samples assigned to them.

K-Means Algorithm

Iterative Procedure

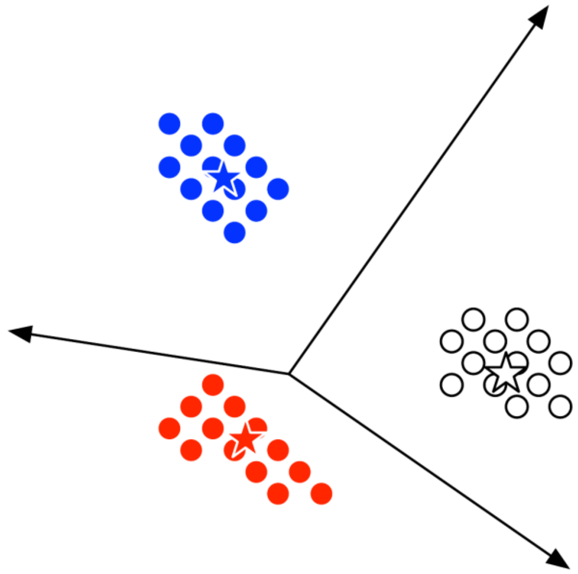
1. Pick K arbitrary cluster centers
2. Assign each sample to its closest centroid

$$z_n = \arg\min_k ||\boldsymbol{c}_k - \boldsymbol{x}_n||^2$$

3. Adjust the centroids to be the means of the samples assigned to them

$$\boldsymbol{c}_k = \frac{1}{|X_k|} \sum_{\boldsymbol{x}_i \in X_k} \boldsymbol{x}_i, \quad X_k = \{\boldsymbol{x}_n | z_n = k\}$$

4. Goto step 2 until no change



Step N: Convergence...

Does K-Means converge?

To analyze convergence, we write SSD in terms of assignments z_n

$$\mathrm{SSD}(C;\mathcal{D}) = \sum_{i=1}^{n} d(\boldsymbol{x}_i, c(\boldsymbol{x}_i))^2 = \sum_{i=1}^{n} \sum_{k} q_{ik} d(\boldsymbol{x}_i, \boldsymbol{c}_k)^2,$$

where $q_{ik} = \mathbb{I}(z_i = k)$ is 1 if the i-th example is assigned to the k-th cluster and 0 otherwise (1-hot coding)

- **Assignment Step:** Minimizes SSD w.r.t. z_i
  - Sets q_{ik} of nearest cluster to 1, all other values are 0
- **Adjustment Step:** Minimizes SSD w.r.t. centroids $\mathbf{c}_k$

$$\boldsymbol{c}_k = \frac{1}{\sum_j q_{jk}} \sum_{i=1}^{n} q_{ik} \boldsymbol{x}_i$$

  - Average vector is the vector with minimum squared distance to all assigned samples

K-Means analysis

Does K-Means converge? Yes, it (locally) minimizes the SSD!

- We have only a finite number of possible values for the centroid
- Every assignment or adjustment step is reducing the SSD (or it stays constant)

Does K-Means converge to the global minimal cost solution? No!

- The objective is an NP-Hard problem, so we can't expect any algorithm to minimize the cost without essentially checking (near to) all assignments.
- It heavily depends on the initialisation of the centroids

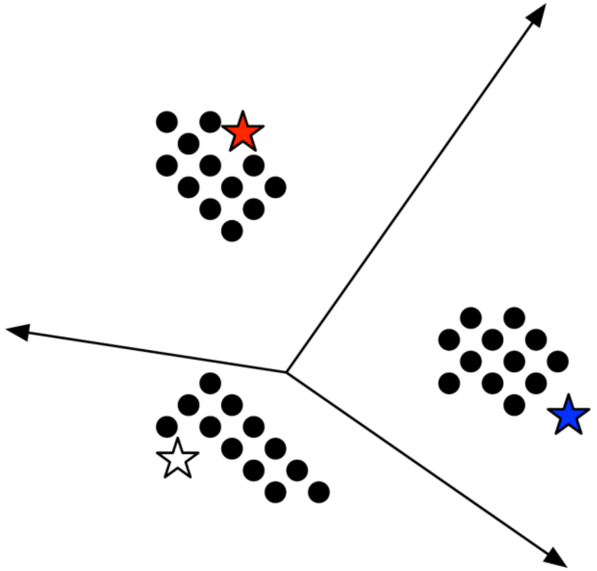
K-means++

Furthest First Initialization:

- Pick a random data-point as first center
- **for k ∈ {2, . . . , K} do**
  - find the example that is furthest from all previously selected means

$$\text{let } n = \arg\max_{i \in \{1,\ldots,n\}} \left(\min_{k' \in \{1,\ldots k-1\}} ||\boldsymbol{x}_i - \boldsymbol{c}'_k||^2 \right)$$

  - Assign centroid: $\boldsymbol{c}_k = \boldsymbol{x}_i$



Furthest first initialization in action...

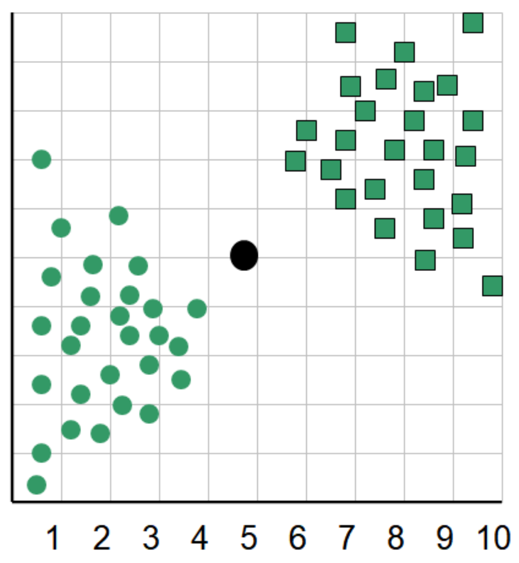
- Converges (in this case) after 1 adjustment

Number of clusters

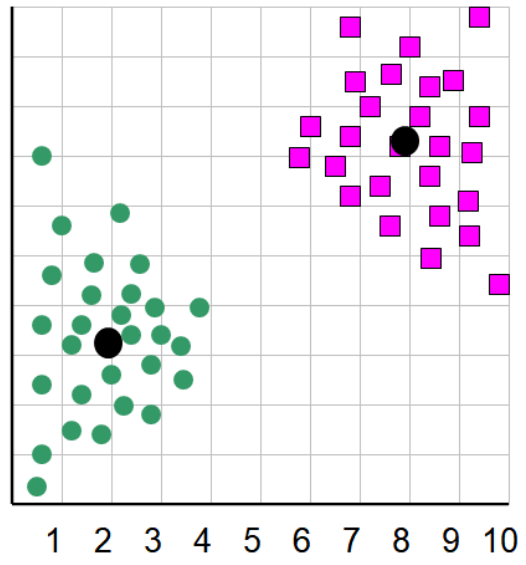
How to choose K?

- Based on 'good' function value decrease on 'holdout' set, cross validation (good but expensive)
- "Knee-finding method" (similar to PCA, heuristic but cheap)

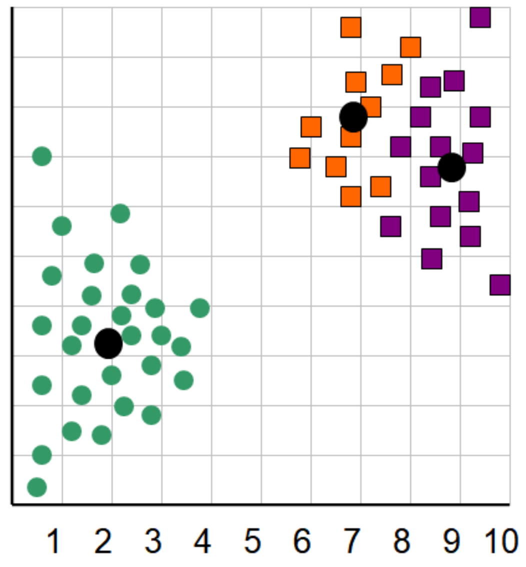
When k = 1, the objective function is 873.0



When k = 2, the objective function is 173.1

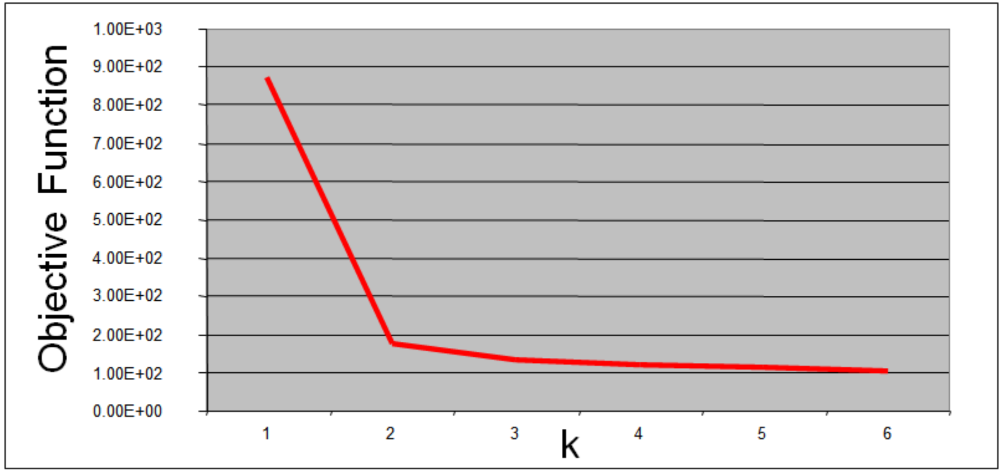


When k = 3, the objective function is 133.6



“Knee-finding” method

- We can plot the objective function (SSD) values for k equals 1 to 6…
- SSD will decrease with higher k (on average)
- The abrupt change at k = 2, is highly suggestive of two clusters in the data.
- This technique for determining the number of clusters is known as “knee finding” or “elbow finding”



Wrap-Up

Strengths:

- K-means usually converges very quickly in practice.
- K-means++ still not guaranteed to find the global optima
  - in practice, we can get stuck.
  - often try multiple initializations (use a little randomness in K-means++ and run the algorithm multiple times).

Weaknesses:

- Applicable only when mean is defined, then what about categorical data?
- Unable to handle noisy data and outliers
- Not suitable to discover clusters with non-convex shapes

Self-test Questions

What you should know now:

- How is the clustering problem defined? Why is it called “unsupervised”?
- How do hierarchical clustering methods work? What is the rule of the cluster-2-cluster distance and which distances can we use?
- How does the k-mean algorithm work? What are the 2 main steps?
- Why does the algorithm converge? What is it minimizing?
- Does k-means finds a the global minimum of the objective?

Density estimation

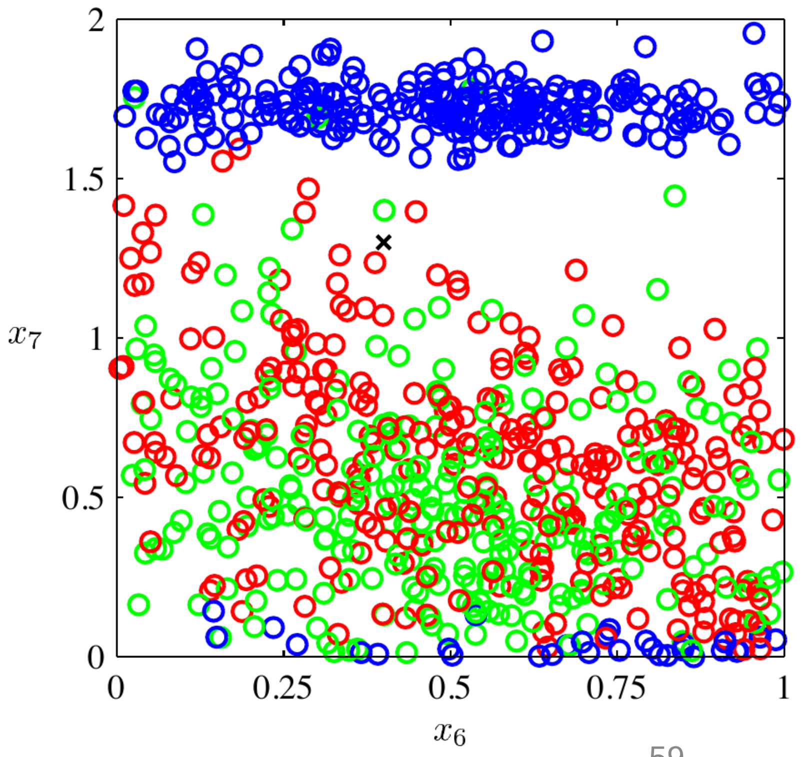
Slides based on lectures from Jan Peters (TU Darmstadt)

Density estimation

How do we get the probability distributions from this?

Applications:

- Classify (generative approaches)
- Outlier / unseen event detection
- Generate new data



Probability Density Estimation

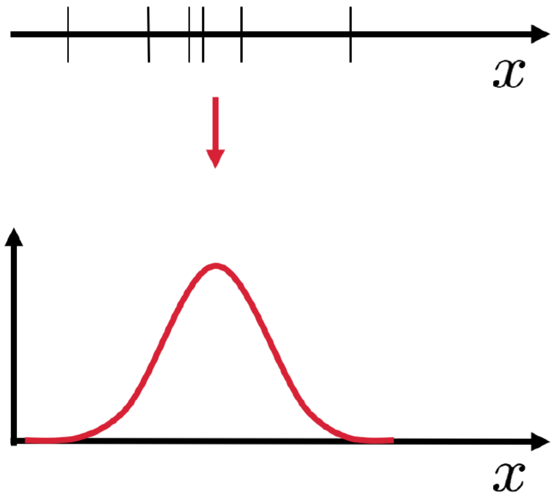
- **Training data**

$$\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$$

- **Estimation**

$$p(\boldsymbol{x})$$

- **Methods**
  - Parametric model
  - Non-parametric model
  - Mixture models



Recap: Parametric models

Most commonly used: Gaussian distribution

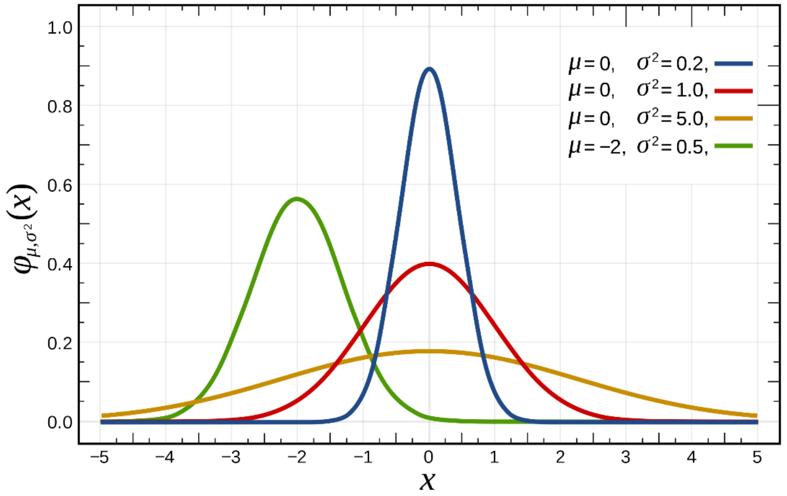
Parametric model:

$$p_{\boldsymbol{\theta}}(x) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

2 Parameters: $\boldsymbol{\theta} = \{\mu, \sigma\}$

- Mean μ
- Variance σ^2

Estimated via maximum likelihood

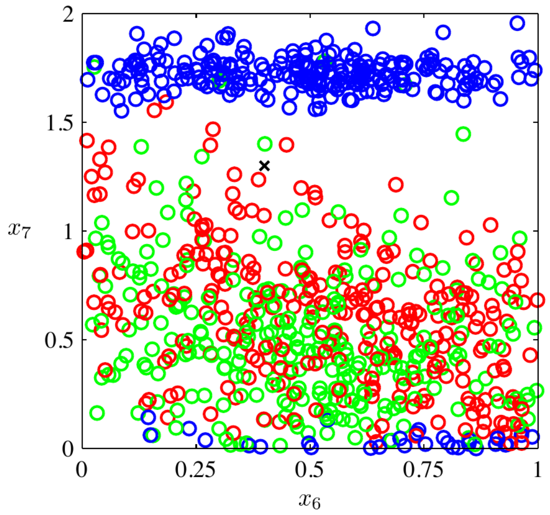


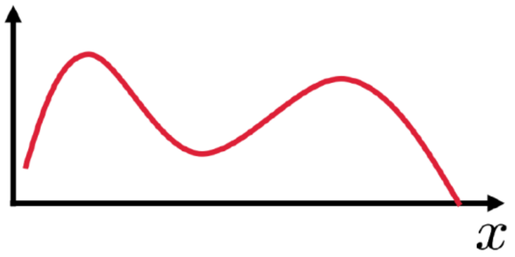
Non-parametric Models

- **Does this look Gaussian?**

- **No!** Indeed most data-sets are not Gaussian distributed. Typically we have:
  - Multi-Modality
  - Non-symmetric
  - No infinite support

- **We need more complex representations:**
  - Non-parametric
  - Mixture models





Non-parametric models

Why use Non-parametric representations?

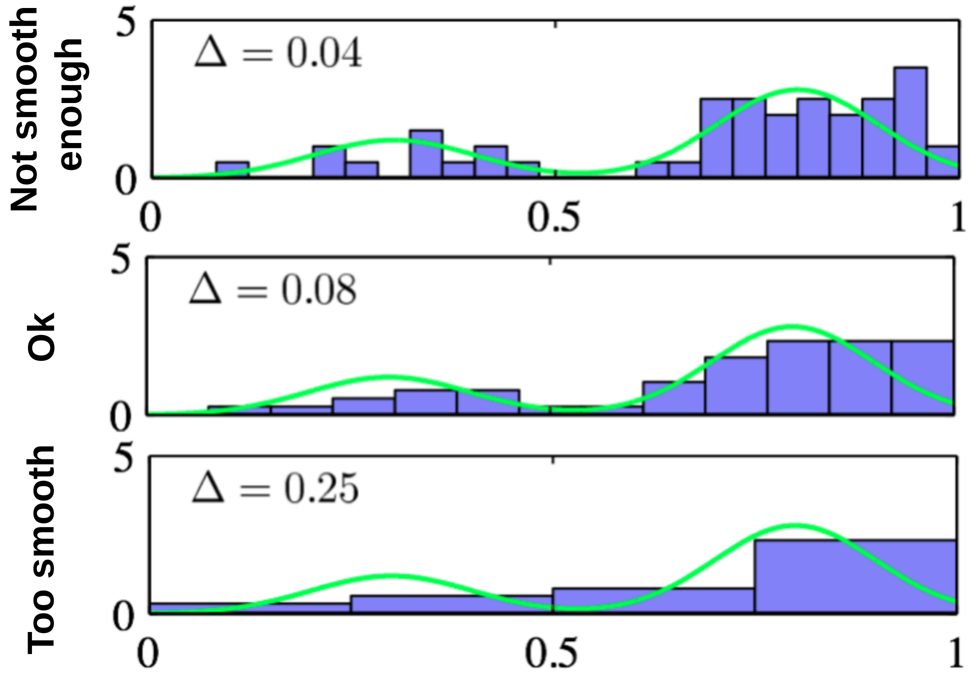
- Often we do not know what functional form the class-conditional density takes (or we do not know what class of function we need)

Probability density is estimated directly from the data (i.e. without an explicit parametric model)

- Histograms
- Kernel density estimation (Parzen windows)
- K-nearest neighbors

Histograms

- Discretize the input space into bins
- Count the samples per bin



Histograms

Properties

- They are very general, because in the infinite data limit any probability density can be approximated arbitrarily well
- At the same time it is a Brute-force method

Problems

- High-dimensional feature spaces
- Exponential increase in the number of bins
- Hence requires exponentially much data
- Commonly known as the curse of dimensionality
- How to choose the size of the bins?
  - This is again a model-selection problem!

More formal definition

- Data point **x** is sampled from probability density p(**x**)
- Probability that **x** falls in region *R*

$$p(\boldsymbol{x} \in R) = \int_R p(\boldsymbol{x}) d\boldsymbol{x}$$

- If R is is sufficiently small, with volume V , then p(**x**) is almost constant

$$p(\boldsymbol{x} \in R) = \int_R p(\boldsymbol{x}) d\boldsymbol{x} \approx p(\boldsymbol{x}) V$$

- We can also compute $p(\boldsymbol{x} \in R)$ from samples (If we have sufficiently large dataset)

$$p(\boldsymbol{x} \in R) \approx \frac{K}{N} \Rightarrow p(\boldsymbol{x}) \approx \frac{K}{NV}$$

where N is the number of total points and K is the number of points falling in the region R

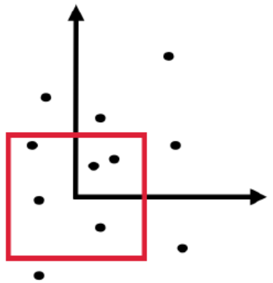
Regions

$$p(\boldsymbol{x}) \approx \frac{K}{NV}$$

- For histograms, the regions are of equal size and span across the whole input space.
- Can we find a more adaptive representation of regions?
  - Yes, make the region always centred on the input **x**!

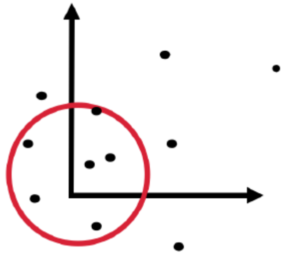
Kernel density estimation

- Fix V and determine K
- Example: determine the number of data points K in a fixed hypercube



K-nearest neighbor

- Fix K and determine V
- Example: increase the size of a sphere until K data points fall into the sphere



Kernel density estimation

A kernel $k(\boldsymbol{x}, \boldsymbol{y})$ “compares” two samples x and y

- Required properties for density estimation:
  - Non-negative: $k(\boldsymbol{x}, \boldsymbol{y}) \geq 0$
  - Distance-dependent: $k(\boldsymbol{x}, \boldsymbol{y}) = g(\underbrace{\boldsymbol{x} - \boldsymbol{y}}_{\text{difference } \boldsymbol{u}})$
- **Volume:** $V = \int g(\boldsymbol{u}) d\boldsymbol{u}$
- **Summed kernel activation:** $K(\boldsymbol{x}_*) = \sum_{i=1}^{N} g(\boldsymbol{x}_* - \boldsymbol{x}_i)$
- **Estimated density:** $p(\boldsymbol{x}_*) \approx \frac{K(\boldsymbol{x}_*)}{NV} = \frac{1}{NV} \sum_{i=1}^{N} g(\boldsymbol{x}_* - \boldsymbol{x}_i)$
- A more formal definition can be found in the kernel lecture

Parzen Window

- **Kernel function:** Hypercubes in d dimensions with edge length h

$$g(\boldsymbol{u}) = \begin{cases} 1, & |u_j| \leq h/2,\ j = 1 \dots d \\ 0, & \text{else} \end{cases}$$

bandwidth

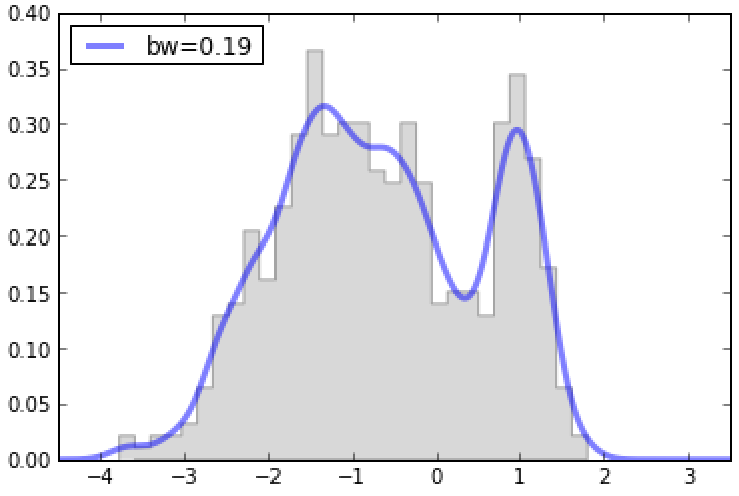
dimensionality

- **Volume:**

$$V = \int g(\boldsymbol{u}) d\boldsymbol{u} = h^d$$

- **Estimated Density:**

$$p(\boldsymbol{x}_*) \approx \frac{1}{Nh^d} \sum_{i=1}^{N} g(\boldsymbol{x}_* - \boldsymbol{x}_i)$$



✓ Simple to compute

× Not very smooth

Gaussian kernel…

- **Kernel function:**

$$g(\boldsymbol{u}) = \exp\left(-\frac{\|\boldsymbol{u}\|^2}{2h}\right)$$

bandwidth

- **Volume:**

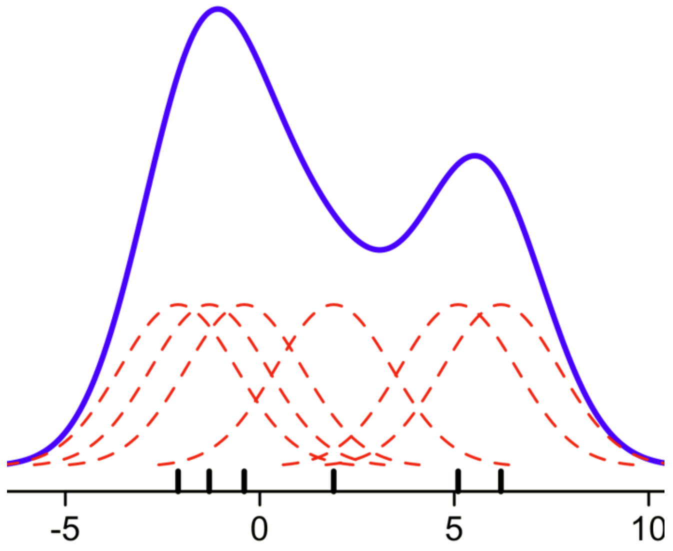
$$V = \int g(\boldsymbol{u})d\boldsymbol{u} = \sqrt{(2\pi h)^d}$$

dimensionality

- **Estimated Density:**

$$p(\boldsymbol{x}_*) \approx \frac{1}{NV}\sum_{i=1}^{N} g(\boldsymbol{x}_* - \boldsymbol{x}_i)$$

$$= \frac{1}{N\sqrt{2\pi h^d}}\sum_{i=1}^{N} \exp\left(-\frac{\|\boldsymbol{x}_* - \boldsymbol{x}_i\|^2}{2h}\right)$$



✓ Smooth

× Infinite support

× Requires a lot of computation

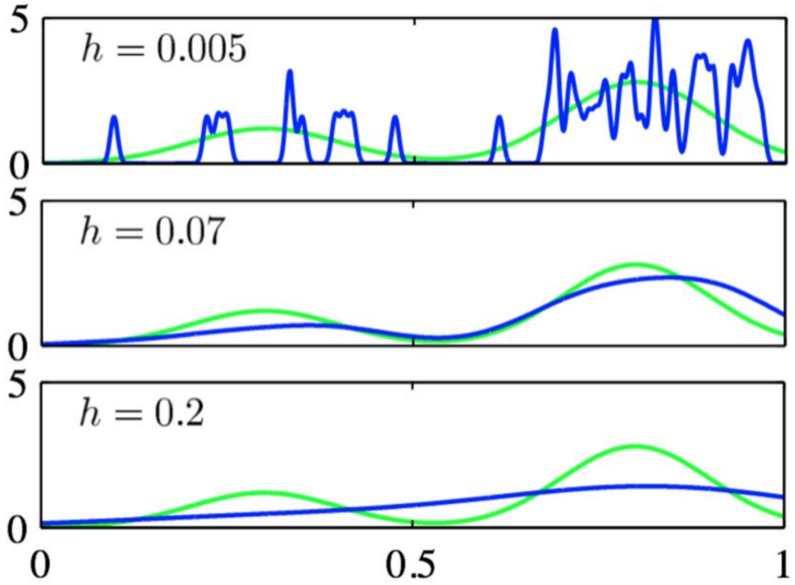
Gaussian KDE Example

- **Problem with kernel methods:** We have to select the kernel bandwidth h appropriately

Not smooth enough

About right

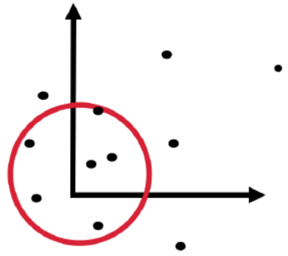
Too smooth



K-nearest neighbour density estimation

K-nearest neighbour: Fix K and determine V

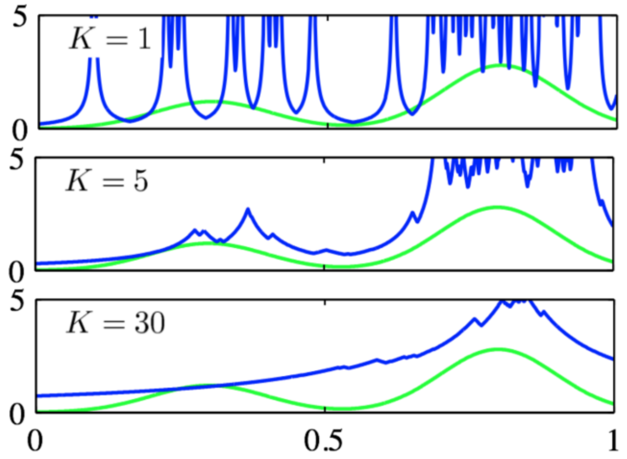
- Example: increase the size of a sphere until K data points fall into the sphere



Not smooth enough

About right

Too smooth



Model-Selection

Nonparametric probability density estimation

- **Histograms:** Size of the bins?
  - too large: too smooth
  - too small: not smooth enough
- **Kernel density estimation:** Kernel bandwidth?
  - h too large: too smooth
  - h too small: not smooth enough
- **K-nearest neighbor:** Number of neighbors?
  - K too large: too smooth
  - K too small: not smooth enough

A general problem of many density estimation approaches

- Select via cross-validation: Select model with highest likelihood on test-set

Mixture Models

Parametric models

- Gaussian, Neural Networks, ...
- ✓ Good analytic properties
- ✓ Simple
- ✓ Small memory requirements
- ✓ Fast
- × Limited representation power (most parametric distributions have only one mode)

Non-Parametric models

- Kernel-density estimation, k-NN
- ✓ General (can represent any distribution)
- × Curse of dimensionality
- × Large memory requirements
- × Slow

- Mixture models combine the advantages of both worlds
- **Key idea:** Create a complex distribution by combining simple ones (e.g. Gaussians)

Mixture model

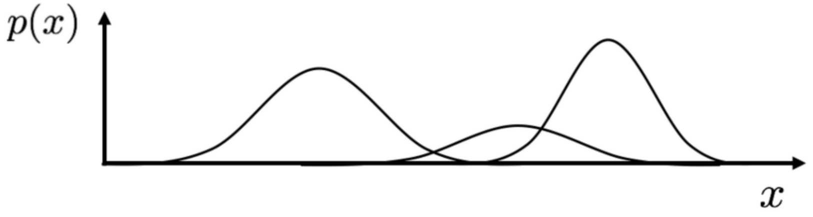
A mixture distribution is the sum of individual distributions:

$$p(\boldsymbol{x}) = \sum_{k=1}^{\overset{\boxed{\text{Number of components}}}{K}} \underbrace{p(k)}_{\boxed{\text{Mixture coefficient}}} \underbrace{p(\boldsymbol{x}|k)}_{\boxed{\text{k-th mixture component}}}$$

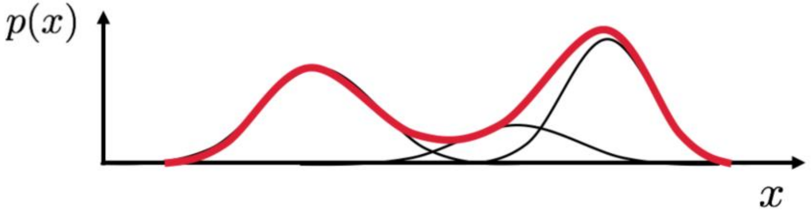
- In the limit with many / infinite components, this can approximate any smooth density

Example: Mixture of Gaussians (MoG)

- Individual Gaussians



- Sum of Gaussians



Gaussian Mixture Models (GMMs)

- **Mixture coefficient:**

$$p(k) = \pi_k, \text{ with } 0 \leq \pi_k \leq 1, \sum_k \pi_k = 1$$

- **Mixture component:**

$$p(\boldsymbol{x}|k) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- **Mixture distribution:**

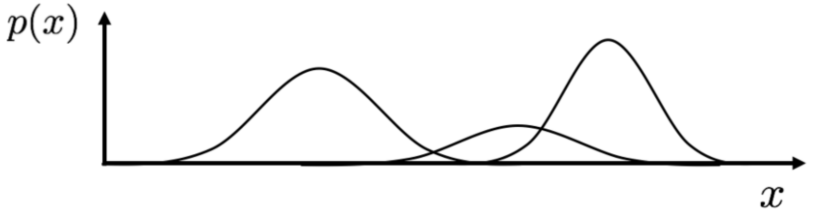
$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

  - Always integrates to 1
  - Parameters of the mixture

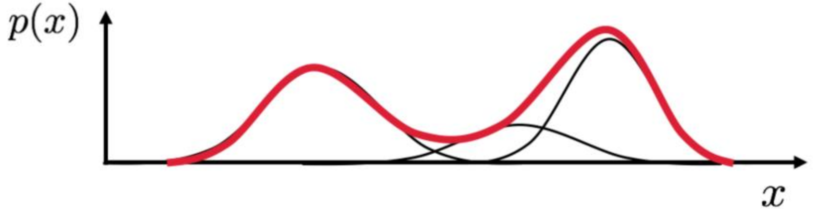
$$\boldsymbol{\theta} = \{\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \pi_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\}$$

Example: Mixture of Gaussians (MoG)

- Individual Gaussians



- Sum of Gaussians



Maximum Likelihood of a mixture

- (Marginal-)Log-Likelihood with N iid. points

$$\mathcal{L} = \log L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \underbrace{p(\boldsymbol{x}_i|\boldsymbol{\theta})}_{\text{marginal}} = \sum_{i=1}^{N} \underbrace{\log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)}_{\text{non-exponential family}}$$

- Q: Can we do gradient descent?

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_j} = \sum_{i=1}^{N} \frac{\pi_j \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_j)$$

$$= \sum_{i=1}^{N} \frac{p(j)p(\boldsymbol{x}_i|j)}{p(\boldsymbol{x}_i)} \boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_j)$$

$$= \sum_{i=1}^{N} \boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_j) p(j|\boldsymbol{x}_i)$$

× Gradient depends on all other components (cyclic dependency)

× No closed form solution

× Typically very slow convergence

► A: Yes, but the sum (marginalization) does not go well with the log

Maximum Likelihood of a mixture

Gradient Descent is possible... but very slow!

- Can we find a more specialized optimization procedure for Mixture Models?
- Yes! **Expectation-Maximization** -> Next Lecture

Wrap-Up

We have seen 3 different unsupervised learning problems today:

- **Dimensionality Reduction:** Find low-dimensional representation of the data
  - Principal Component Analysis: Computes eigen-vector of Covariance Matrix
- **Clustering:** Find structure and similarities in the data
  - K-means: Iteratively compute centroids over closest data-points
- **Density estimation:** Estimate “frequency” of data points
  - Non-parametric methods such as KDE
  - (Gaussian) Mixture Models