Statistical Data Analysis

Jana de Wiljes

December 7, 2022

Unsupervised learning

Problem setting

Goal: Approximate function f, that describes the link between two random variables X and Y which have the joint distribution $\pi(z) = \pi(x, y)$

Choice of parametrisation:

- choose model class $\mathcal{H}$
- and appropriate loss functional $l(y, h(x))$

Expected Risk

For $h \in \mathcal{H}$ we define the expected Risik as follows

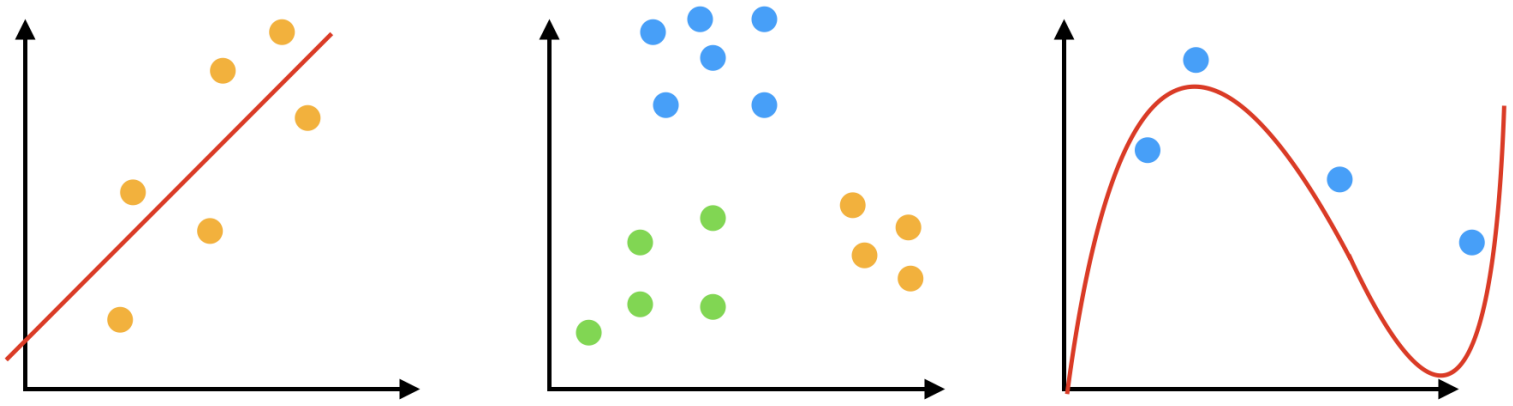
$$R(h) = \int_{\mathbf{Z}} l(y, h(x))\pi(z)dz \tag{1}$$

Approach: Want to find $h \in \mathcal{H}$ so that

$$h^* = \arg\min_{h \in \mathcal{H}} R(h) \tag{2}$$

Empirical Risk

Given in practice: independent and identical distributed Samples
$S = \{(x_i, y_i)\}_{i=1}^{N}$ with $(x_i, y_i) \sim \pi(x, y)$ for $i \in \{1, \ldots, N\}$



Empirical Risk

For a given sample set S we define the corresponding empirical risk as follows:

$$R_S(h) = \frac{1}{N} \sum_{i=1}^{N} l(y_i, h(x_i))$$

Empirical Risk-Minimizer

Empirical Risk-Minimizer

A learning algorithm $\hat{h}_N$ with $S = \{(x_i, y_i)\}_{i=1}^N$ where $(x_i, y_i) \sim \pi(x, y)$ of the form

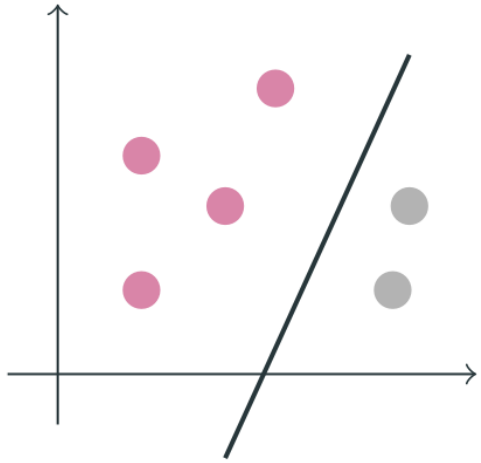
$$\hat{h}_N \in \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} l(y_i, h(x_i))$$

is called Empirical Risk-Minimizer.

Unsupervised learning

Consider:

- $f : \mathcal{X} \to \{0, 1\}$
- $f(x) = y$
- $\pi(x, y) = \begin{cases} \pi(x) & \text{für } f(x) = y \\ 0 & \text{für } f(x) \neq y \end{cases}$



But only x samples are observed

- Need to dhoose a different family of functions $\mathcal{H}$

Examples

Linear regression with regularisation

Data: $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}^h$

loss function:

$$l(x_i, f_w(x_i)) = \frac{1}{2}\|y_i - wx_i\|_2^2 + \|w_k\|_2^2$$

K-Means

Data: $x_i \in \mathbb{R}^d$, no labels y_i

loss function:

$$l(x_i, f_w(x_i)) = \min_{w_k} \frac{1}{2}\|x_i - w_k\|_2^2$$

where $w_1, \ldots w_K$ are unknown cluster centres

Perceptron

Data:$x_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$

loss function:

$$l(x_i, f_w(x_i)) = \max(0, -yw^\top x)$$

Support Vector Machines

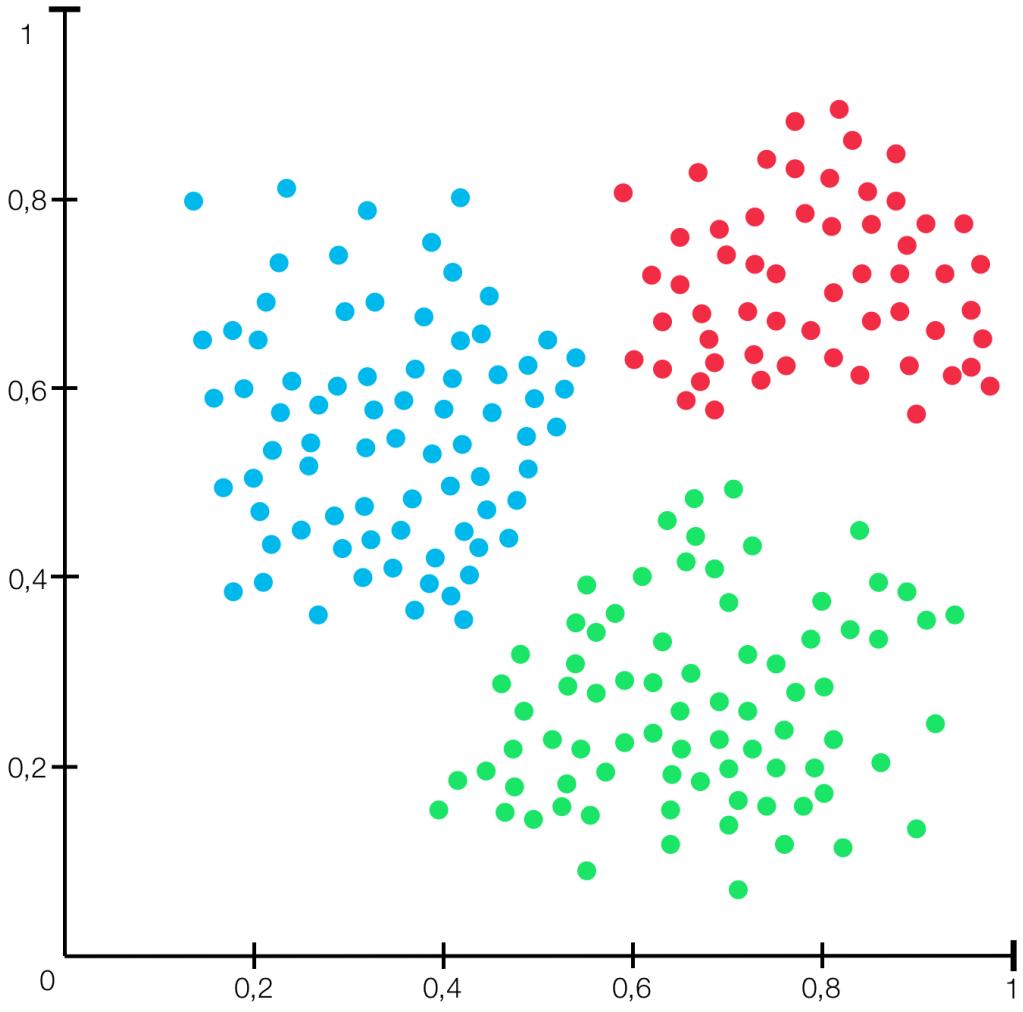
Data: $x_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$

loss function:

$$l(x_i, f_w(x_i)) = \lambda\|w\|_2^2 + \max(0, 1 - yw^\top x_i)$$

with hyper parameter $\lambda > 0$

Clustering



K-means clustering

Algorithm 1 Kmeans

Input: Anzahl an Clustern K und Menge der zu klassifizierenden Punkte $\{x_1, \ldots, x_M\}$
Output: Mengen $\mathcal{M}_k$ der verschiedenen Cluster
Initialization: Initialisiere die Cluster Zentren $\theta_1, \ldots, \theta_K \in \mathbb{R}^n$ zufällig; **while** *Bis Abbruchkriterium erfüllt* **do**

end
$k = 1 : K$
$\mathcal{M}_k := \{\ \}$
for ($m = 1 : M$ **)**

$j = \arg\min_h ||\theta_h - x_m||_2$
$\mathcal{M}_j = \mathcal{M}_j \cup \{x_m\}$

for ($k = 1 : K$ **)**
$\theta_k = \frac{1}{|\mathcal{M}_k|} \sum_{x_m \in \mathcal{M}_k} x_m$

Initialisation

- Random Partition Method
- Forgy Initialization
- kmeans++
  1. choose θ_1 uniformly at random from set of points
  2. Choose new center θ_i with probability

$$\frac{D(x_m)^2}{\sum_{x_l} D(x_l)^2} \tag{3}$$

     where $D(x_m)$ denotes the shortest distance from data point x_m to the closest center we have already chosen
  3. Repeat Step 2 until we have all K centers

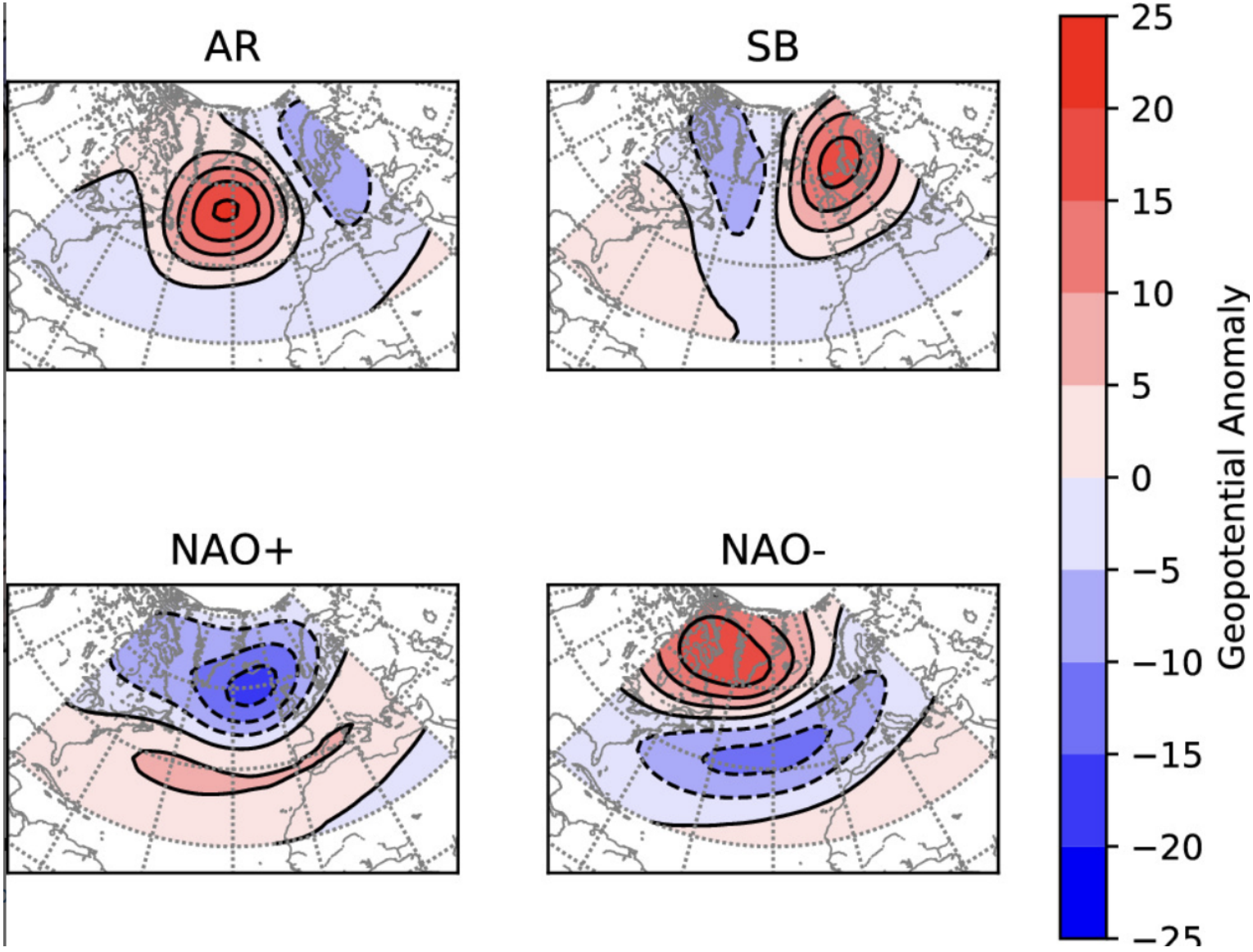
K-means clustering

Disadvantages

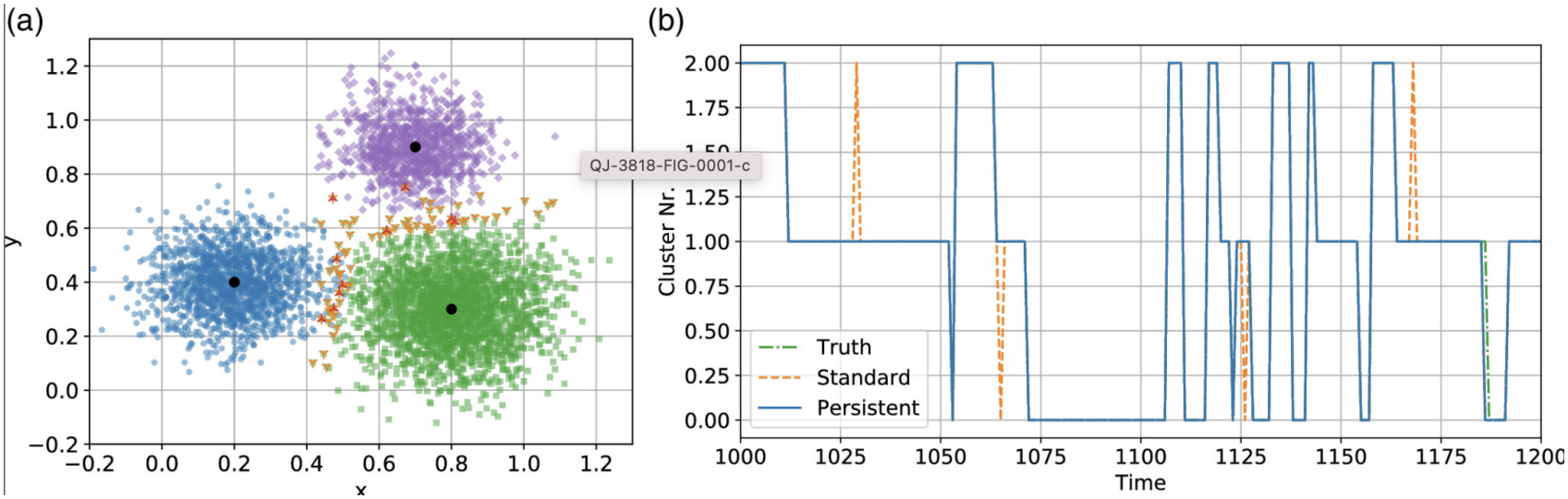
- true number of clusters K unknow (requires tuning)
- K-means algorithm dependents on the chosen initial values
- Clustering data of varying sizes and density
- Centroids can be dragged by outliers

Example: pattern recognition for atmospheric circulation regimes

Regime

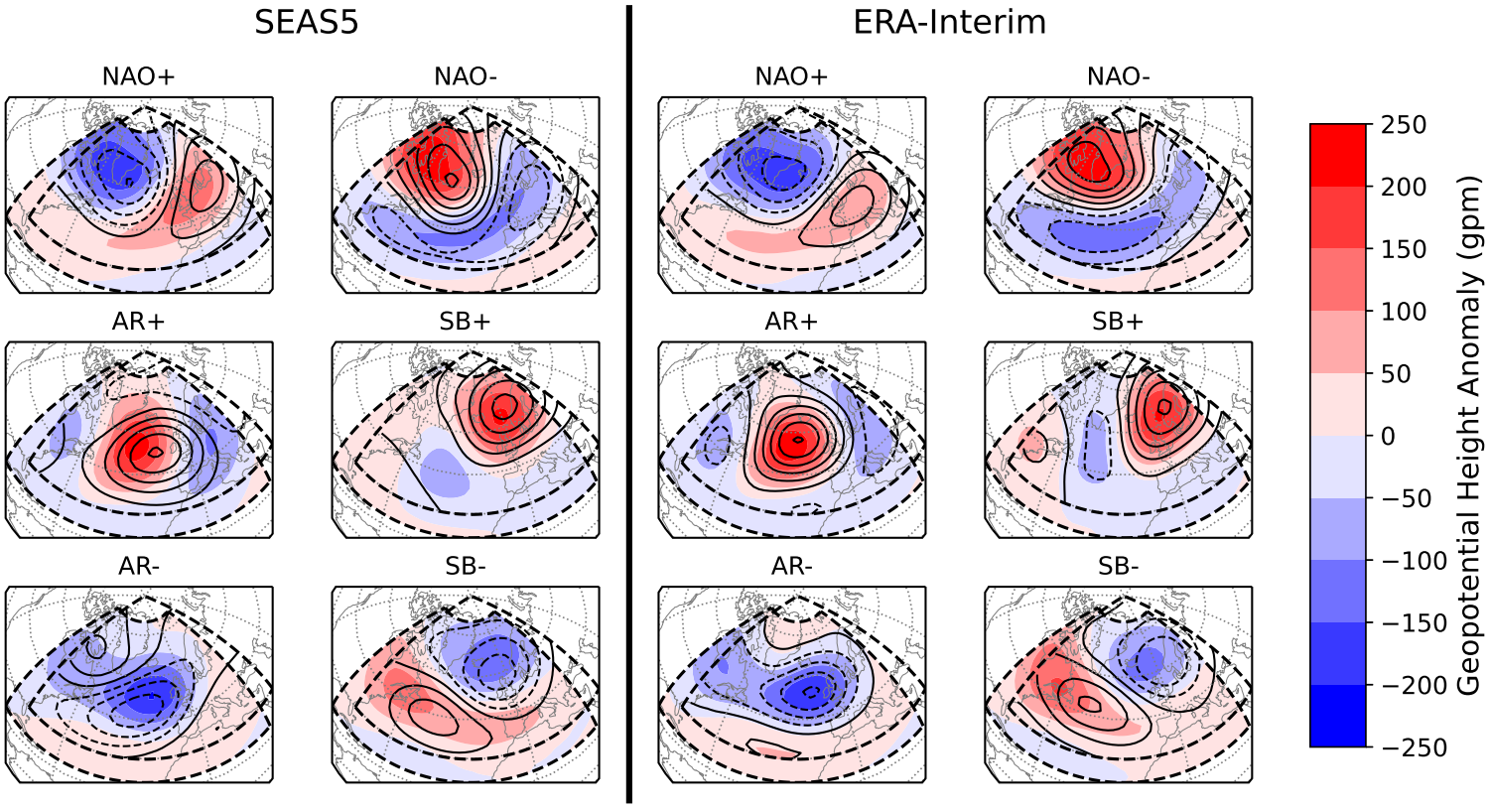


Time persistency constraint

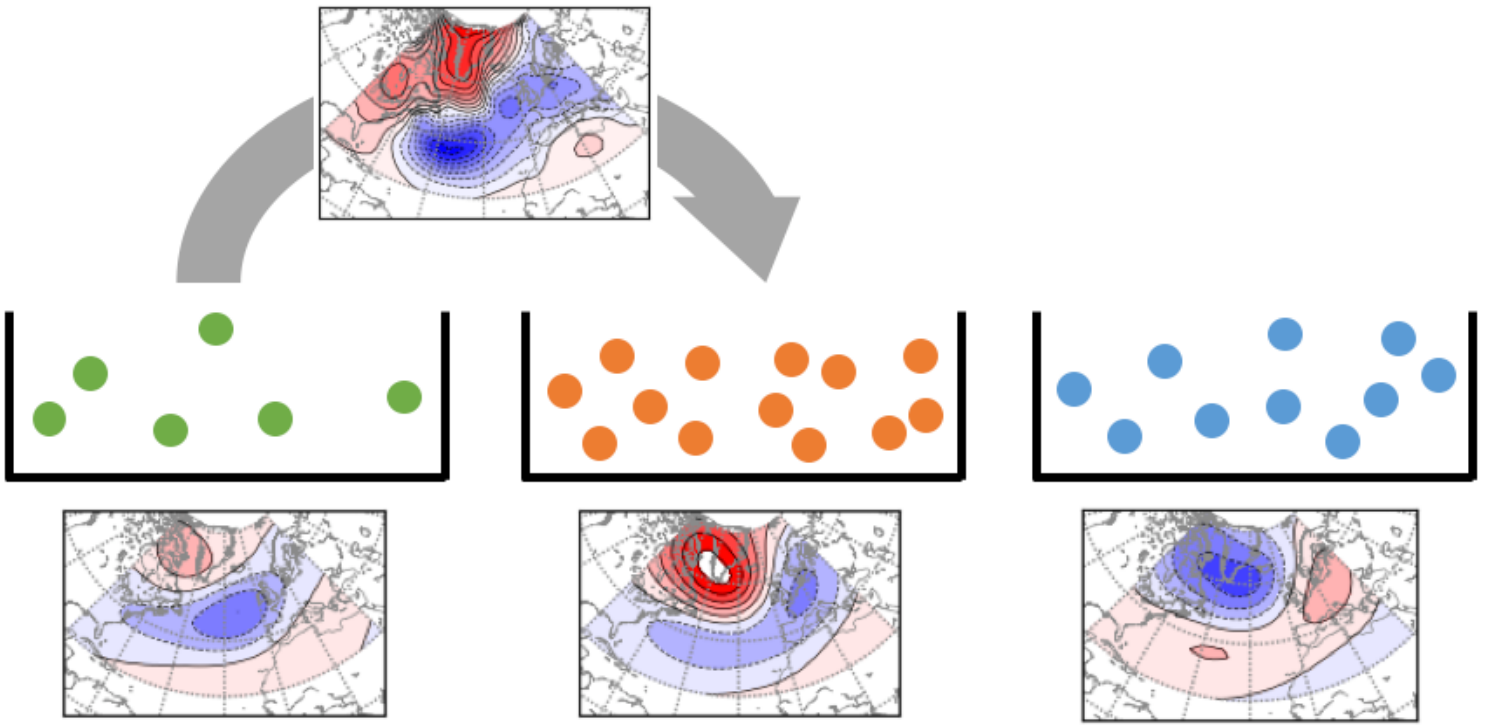


$$\sum_{t=1}^{T-1} |\gamma_k(t+1) - \gamma_k(t)| \leq N_C \quad \forall k$$

k-means clustering for different domains



k-means clustering for different domains



Optimisation problem

$$\mathbf{L}(\Theta, \Gamma) = \sum_{t=0}^{T} \sum_{n=1}^{N} \sum_{i=1}^{k} \gamma_i(t, n) \|x_{t,n} - \theta_i\|^2$$

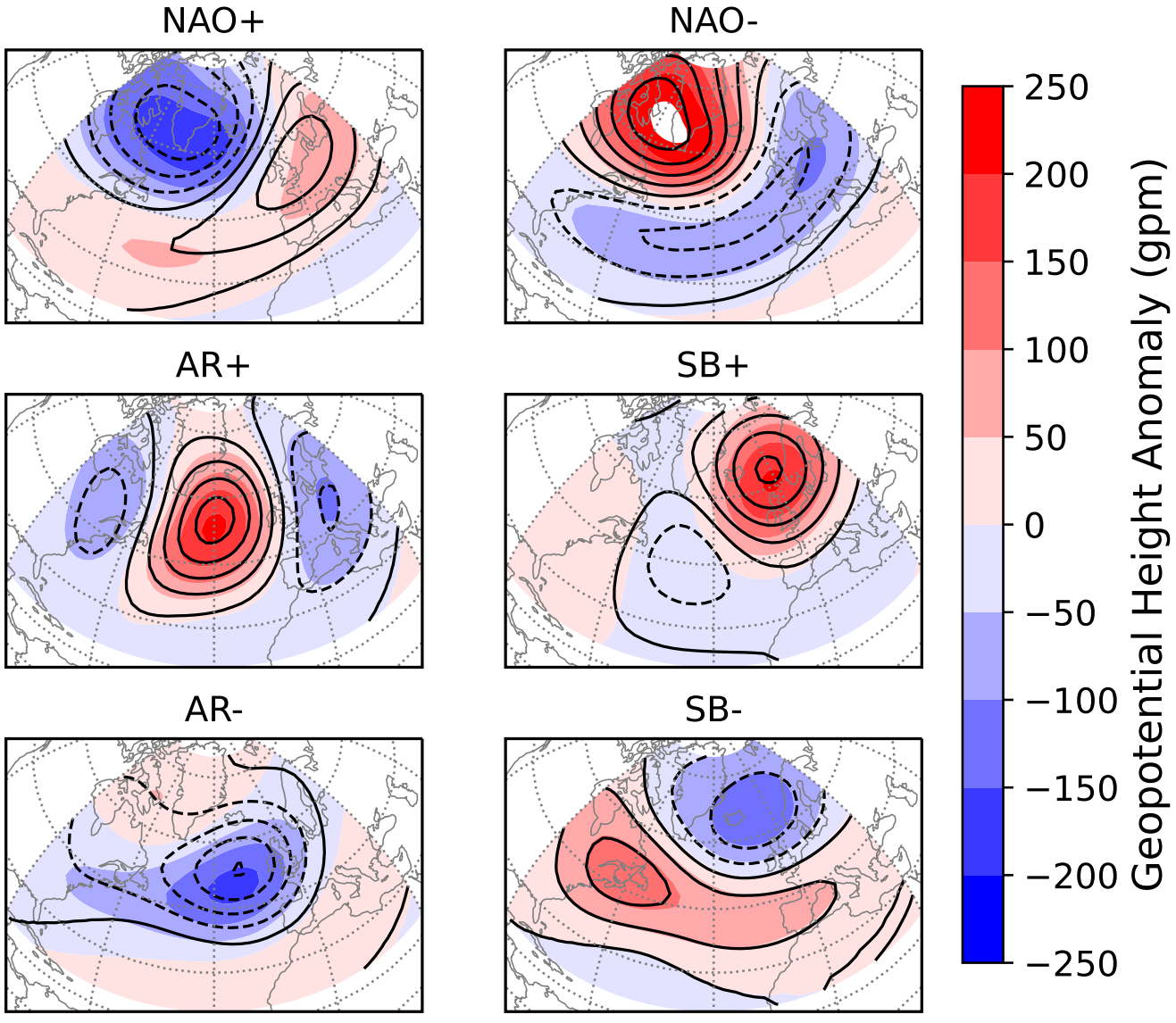
with

$$\sum_{i=1}^{k} \gamma_i(t, n) = 1, \qquad \forall t \in [0, T], \quad \forall n \in [1, N].$$

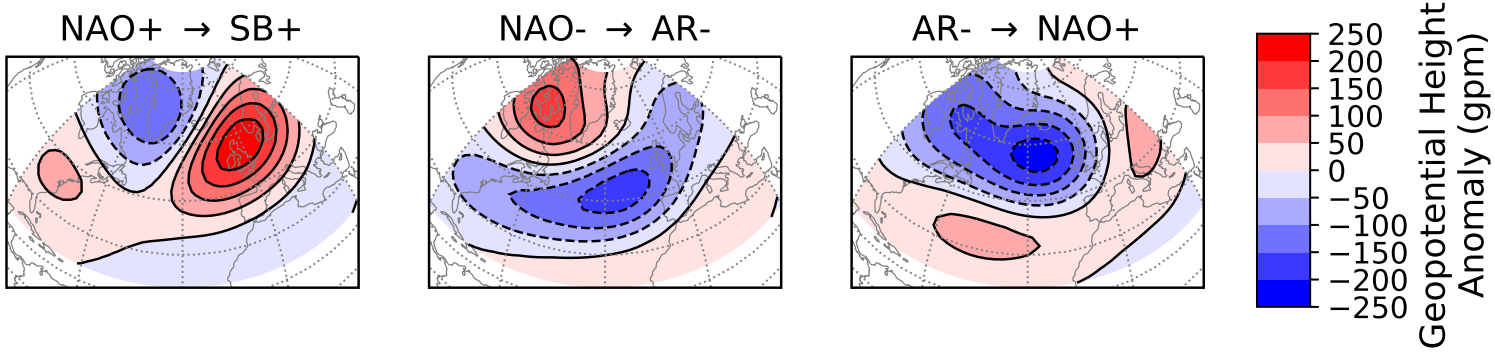
and

$$\sum_{i=1}^{k} \sum_{n_1, n_2} |\gamma_i(t, n_1) - \gamma_i(t, n_2)| \leq \phi \cdot C_{\text{eq}}, \qquad \forall t \in [0, T],$$

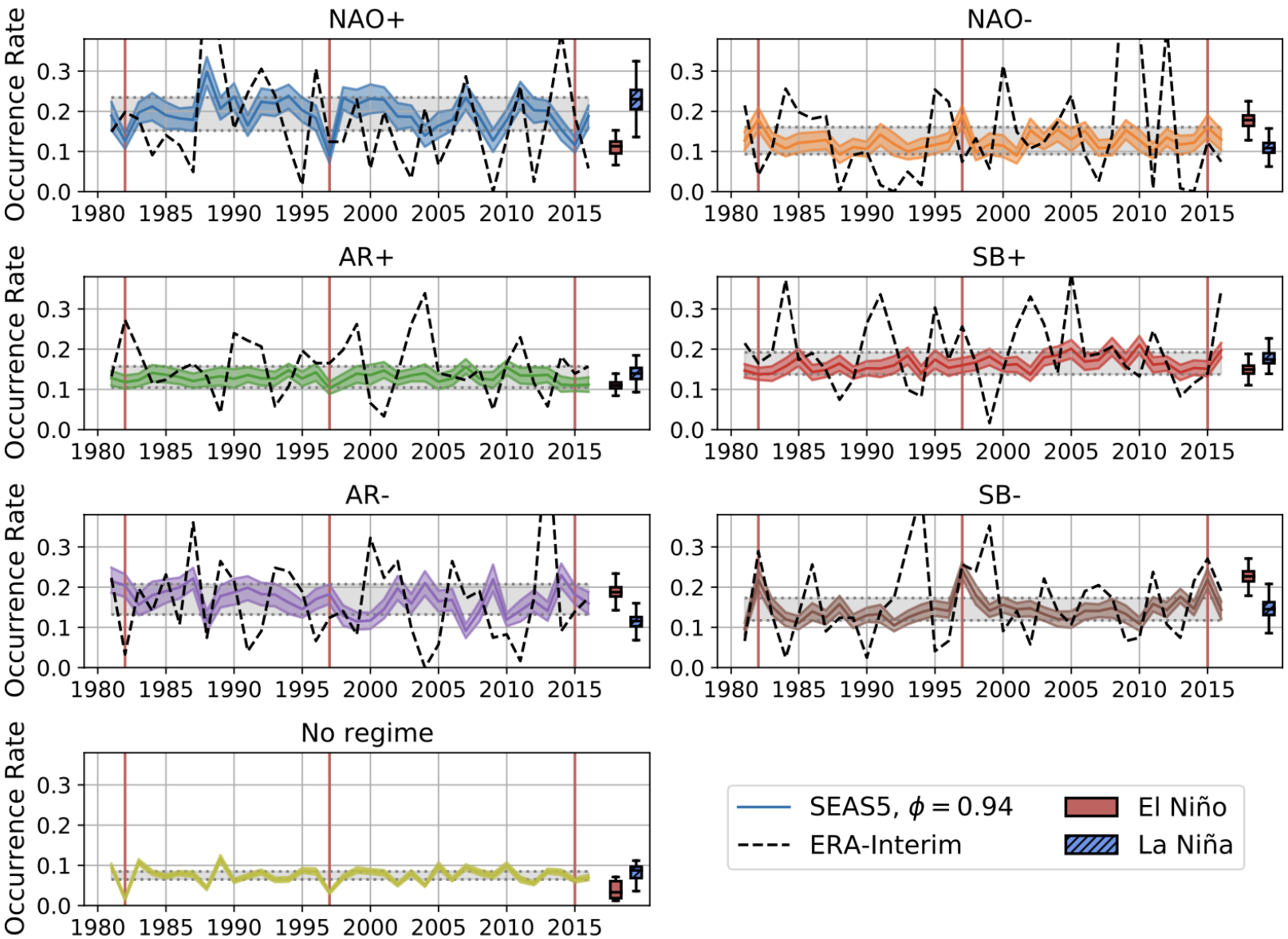
Ensemble persistency constraint



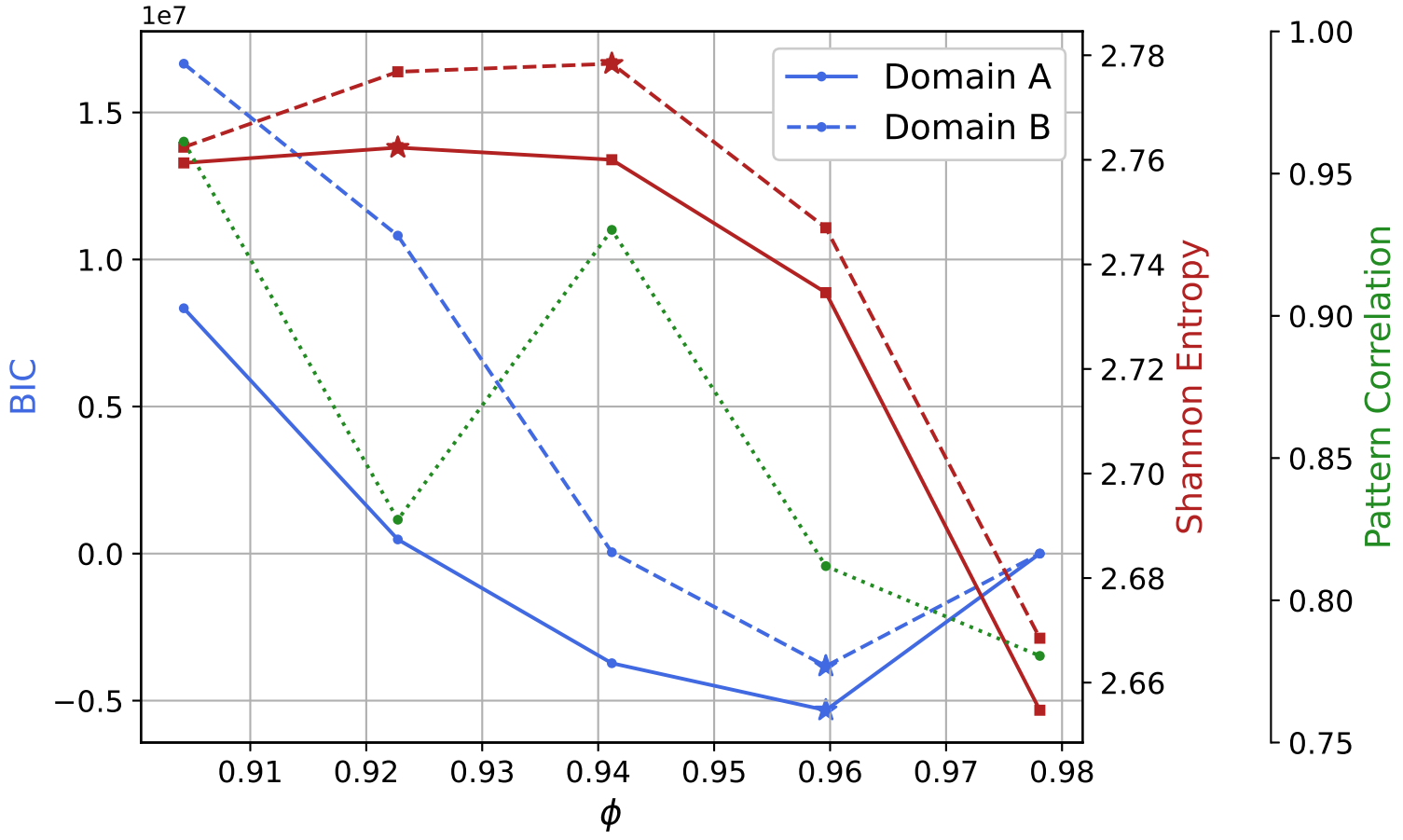
Ensemble persistency constraint



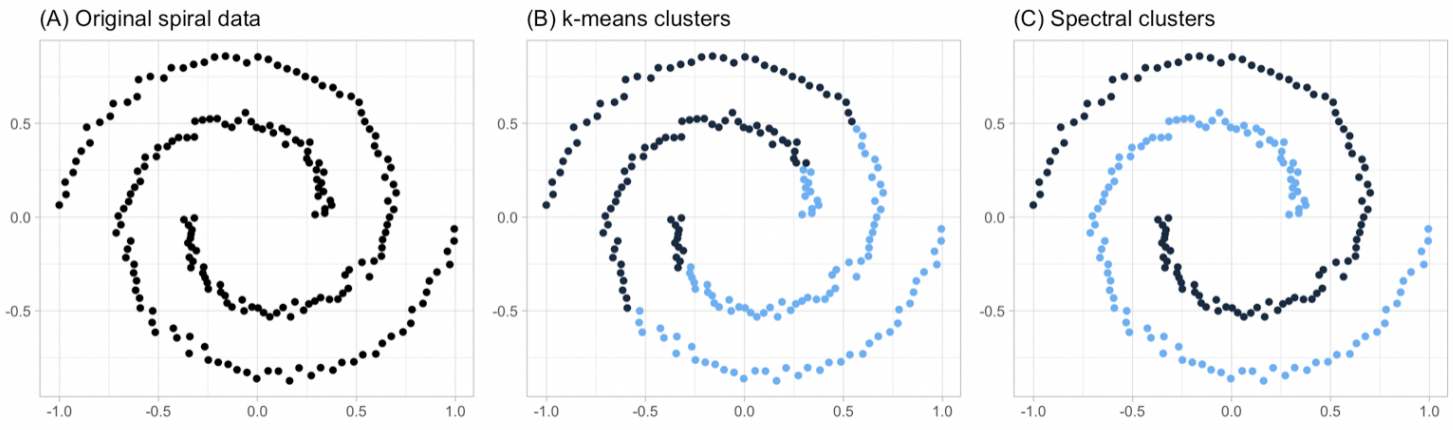
Occurrence rates



Optimal ϕ


1e7
Domain A
Domain B
BIC
Shannon Entropy
Pattern Correlation
ϕ
1.5
1.0
0.5
0.0
−0.5
0.91
0.92
0.93
0.94
0.95
0.96
0.97
0.98
2.78
2.76
2.74
2.72
2.70
2.68
2.66
1.00
0.95
0.90
0.85
0.80
0.75

K-Means vs Spectral Clustering



Spectral clustering

Eigenvalue and Eigenvectors

Definition: Let V be K-vector space, $f : V \to V$ an endomorphismus, $\lambda \in K$. A skalar λ is called **Eigenvalue** of f, if there is a vector $v \in V, v \neq 0$, so that

$$f(v) = \lambda \cdot v.$$

Such a v is then called **Eigenvector** of f corresponding to λ.

Note: An eigenvalue λ can be $0 \in K$, but an eigenvector is always $\neq 0$.

Theorem

Theorem: Let V be a K- vector space, $n = \dim V < \infty$ and $f\colon V \to V$ an Endomorphismus. Then the following two statements are equivalent:

1. V has a basis of eigenvectors of f.
2. There is a basis $\mathcal{B}$ of V, so that

$$M_{\mathcal{B}}^{\mathcal{B}}(f) = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \text{ mit } \lambda_i \in K.$$

Proof

Proof.

Let $M_{\mathcal{B}}^{\mathcal{B}}(f)$ be the representation with respect to für $\mathcal{B} = \{v_1, \ldots, v_n\}$, then the following holds

$$f(v_i) = \lambda_i v_i \quad \forall i$$
$$\Leftrightarrow \quad v_1, \ldots, v_n \text{ is a basis of eigenvectors}$$

□

Characteristic polynom

Definition: Let $A \in K^{n \times n}$ and $\lambda \in K$ arbitrary. Then

$$\text{Eig}(A, \lambda) := \{v \in K^n \mid Av = \lambda v\}$$

is called the **Eigenspace** of A with respect to λ.

$$\chi_A(t) := \det(A - tE) \in K[t]$$

is called the **characteristic polynom** of A.

Bemerkung

For a Matrix $A \in K^{n \times n}$ the following holds:

$$\lambda \in K \text{ is an Eigenvalue of } A \Leftrightarrow \text{Eig}(A, \lambda) \neq 0.$$

Theorem

Theorem: Let $A \in K^{n \times n}$ and $\lambda \in K$. Then

$$\lambda \text{ is an Eigenvalue of } A \Leftrightarrow \lambda \text{ is a root of } \chi_A(t).$$

Proof: It holds that

$$\begin{aligned}
\lambda \text{ Eigenvalue } &\Leftrightarrow Av = \lambda v \text{ for a } v \neq 0 \\
&\Leftrightarrow (A - \lambda E) \cdot v = 0 \text{ has a non trivial solution } v \neq 0 \\
&\Leftrightarrow \text{Eig}(A, \lambda) = \text{Ker}(A - \lambda E) \neq 0 \\
&\Leftrightarrow \det(A - \lambda E) = 0 \\
&\Leftrightarrow \chi_A(\lambda) = 0.
\end{aligned}$$

Example

We consider the matrix

$$\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

The characteristic polynom of A is:

$$\begin{aligned} \chi_A(t) &= \det \begin{pmatrix} 2-t & -1 \\ -1 & 2-t \end{pmatrix} \\ &= (2-t)^2 - 1 \\ &= t^2 - 4t + 3 \\ &= (t-3)(t-1), \end{aligned}$$

then the eigenvalues are $\lambda_1 = 3, \lambda_2 = 1$.

Example

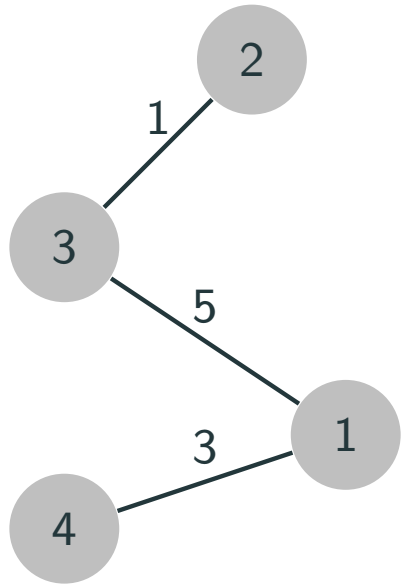
The Eigenspaces of these eigenvalues are

$$\begin{aligned}\mathrm{Eig}(A,3) &= \mathrm{Ker}(A-3E)\\ &= \mathrm{Ker}\begin{pmatrix}-1 & -1\\ -1 & -1\end{pmatrix} = \langle\begin{pmatrix}1\\ -1\end{pmatrix}\rangle,\\ \mathrm{Eig}(A,1) &= \mathrm{Ker}\begin{pmatrix}1 & -1\\ -1 & 1\end{pmatrix} = \langle\begin{pmatrix}1\\ 1\end{pmatrix}\rangle.\end{aligned}$$

What is a graph (formally)?

The objects on the following slides will play a role for spectral clustering

- $G = (V, E, \omega)$, where $V \neq \emptyset$ is a set (called the **vertex set**), $E \subset \binom{V}{2} = \{\{u, v\} : u, v \in V\}$ (called the **edge set**) and $\omega : E \to \mathbb{R}^+$, is called a **(weighted) graph**
- usually we choose (or rename) $V = \{1, 2, \ldots, n\}$ and use the notations $ij = \{i, j\}$ for $\{i, j\} \in E$ and $\omega_{ij} = \omega(ij)$
- for every $i \in V$ define $N(i) := \{j \in V : ij \in E\}$, called the **neighbourhood** of i (in G); elements of $N(i)$ are called **neighbours** of i (those elements are **adjacent** to i)
- $d_i := d(i) := |N(i)|$ is the **degree** of i



$w(\{2, 3\}) = 1$,
$N(4) = \{1\}$,
$d(1) = |\{3, 4\}| = 2$

Graph classes

Well known graph classes are:

- the **path graph** P_n has vertex set $\{1, 2, \ldots, n\}$ and edge set $\{\{1, 2\}, \{2, 3\}, \ldots, \{n-1, n\}\}$
- the **cycle graph** C_n has vertex set $\{1, 2, \ldots, n\}$ and edge set $\{\{1, 2\}, \{2, 3\}, \ldots, \{n-1, n\}, \{n, 1\}\}$
- the **complete graph** K_n consists of n vertices which are all adjacent to each other
- the **complete bipartite graph** $K_{m,n}$ has two sets V_1 and V_2 of vertices of sizes m and n, such that the edge set consists of all possible edges between V_1 and V_2

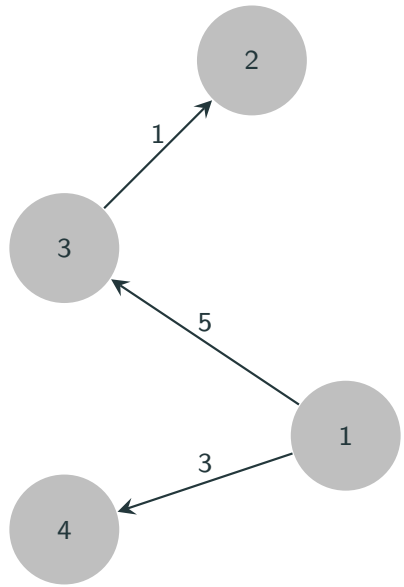
A set of vertices in a graph which are all adjacent to each other (they **induce** a complete (sub)graph), is called **clique**.

The graph $K_{1,n}$ is called a **star**.

What is a digraph (formally)?

Edges can have a direction.

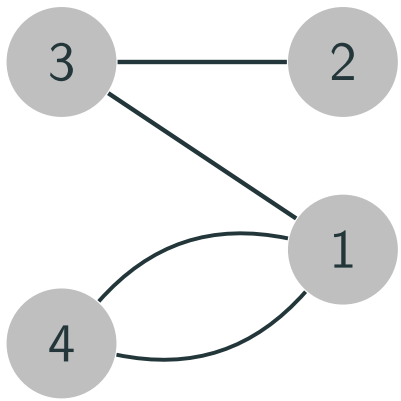
- $G = (V, E, \omega)$, where $V \neq \emptyset$ is a set, $E \subset V \times V$ (this is sometimes also called the **set of arcs**) and $\omega : E \to \mathbb{R}^+$, is called a **(weighted) digraph**
- for $(i, j) \in E$ the vertex i is called **predecessor** of j and j is called **successor** of i
- similar notation simplifications as before
- $N^+(i) := \{j \in V : (i, j) \in E\}$ is the **out-neighbourhood** of i, $N^-(i) := \{j \in V : (j, i) \in E\}$ is the **in-neighbourhood** of i
- $d^+(i) := |N^+(i)|$ is the **out-degree** of i and $d^-(i) := |N^-(i)|$ is the **in-degree** of i



$N^-(3) = \{1\}$,
$N^+(4) = \emptyset$,
$d^+(1) = 2$,
$d^-(2) = 1$

Example of a multigraph

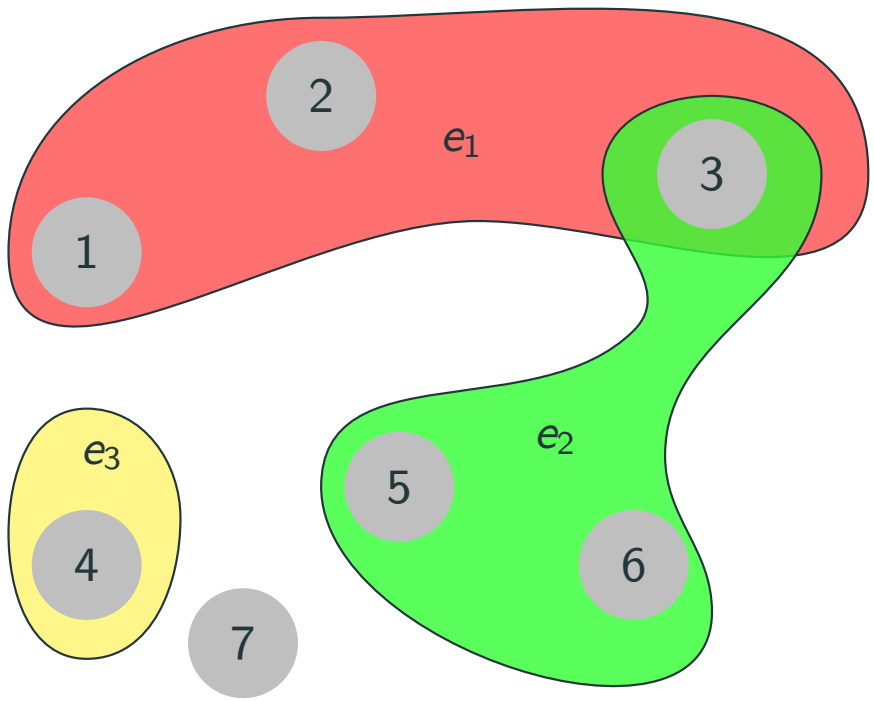
It is sometimes necessary to allow multiple edges between two vertices or a **loop** (a self-edge). In that case we use the term **multigraph**.



What is a hypergraph (formally)?

Sometimes more than two vertices need to form an edge (certain real life situations' have this property).

- natural generalisation is a **hypergraph** $H = (V, E)$, where
  - $V \neq \emptyset$ is (also) a set, but
  - E can be an arbitrary subset (the elements are called **hyperedges**) of the power set $\mathcal{P}(V)$
- if all hyperedges are of the same size r, then H is called r-**uniform**



Storing graphs

Certain matrices and lists can be associated with a graph (we will see more examples later).

- **affinity matrix** $W(G)$:
$$w_{ij} = \begin{cases} \omega_{ij} & \text{if } \{i,j\} \in E, \\ 0 & \text{else.} \end{cases}$$
- **adjacency matrix** $A(G)$: special case of $W(G)$, where $w_{ij} = 1$ for all $ij \in E$.
- **adjacency list**:
  - associate list to every vertex containing its neighbours
  - call list of these lists adjacency list of the graph (treated differently in the literature)
  - not very useful for mathematical arguments
  - especially useful (for storing) when $A(G)$ is sparse

All the above constructions are valid for directed graphs.

How to transform a digraph into a graph?

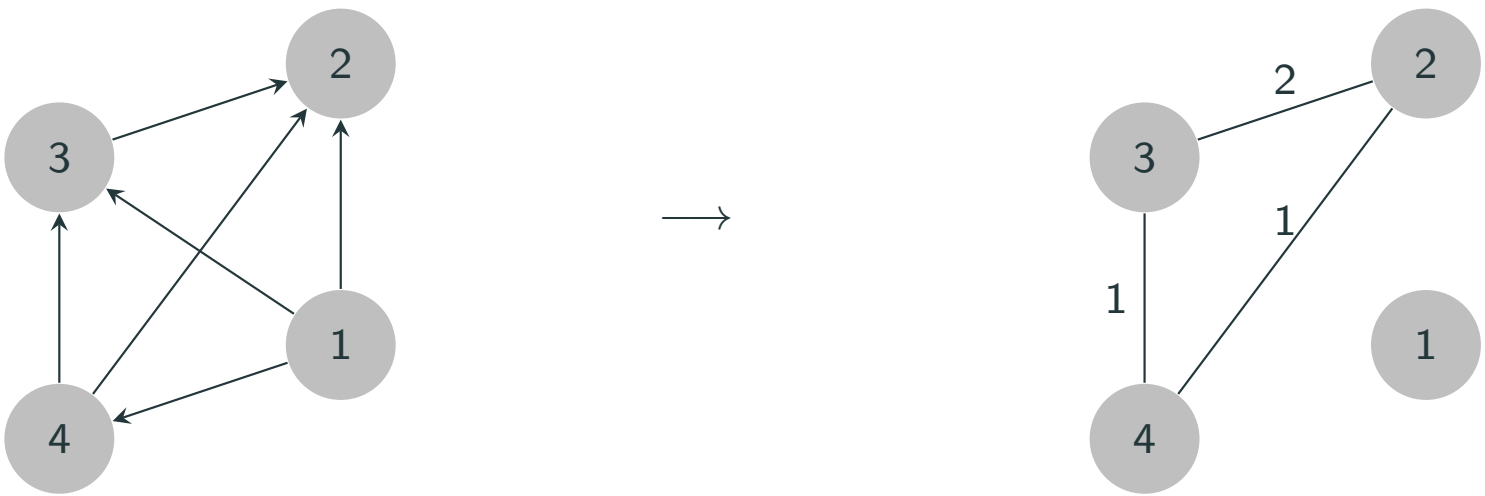
Consider the following three approaches.

- ignore the directions
- carry out **cocitation coupling**
  - existence of common predecessors induce edges
  - weights are naturally given by number of common predecessors
- carry out **bibliographic coupling**
  - existence of common successors induce edges
  - weights are naturally given by number of common successors

Cocitation coupling

A (undirected) graph is constructed via:

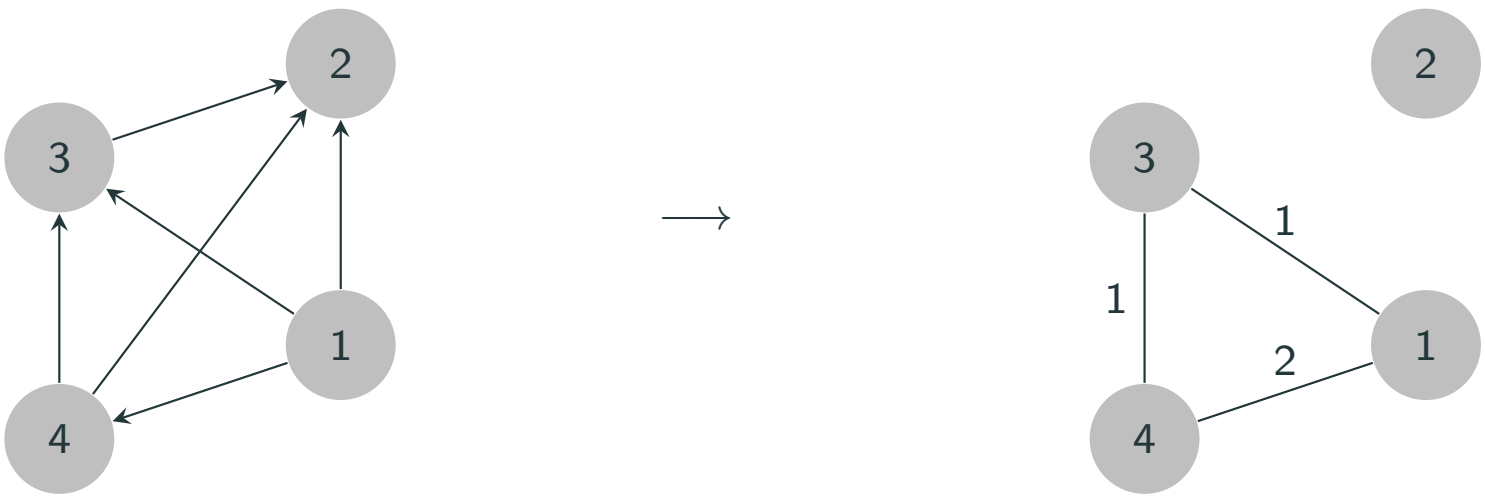
- **cocitation** c_{ij} of $i, j \in V$ is the number of common predecessors of i and j
- the **cocitation network** has vertex set V and an edge between i and j iff $c_{ij} > 0$
- it is also possible to obtain a weighted graph with weights c_{ij}
- note that $c_{ij} = \sum_{k=1}^{n} a_{ki} a_{kj}$, therefore $C = A^T A$



Bibliographic coupling

A (undirected) graph is constructed via:

- **bibliographic coupling** b_{ij} of $i, j \in V$ is the number of common successors of i and j
- the **bibliographic coupling network** has vertex set V and an edge between i and j iff $b_{ij} > 0$
- it is also possible to obtain a weighted graph with weights b_{ij}
- note that $b_{ij} = \sum_{k=1}^{n} a_{ik} a_{jk}$, therefore $B = AA^T$



How to transform a hypergraph into a graph?

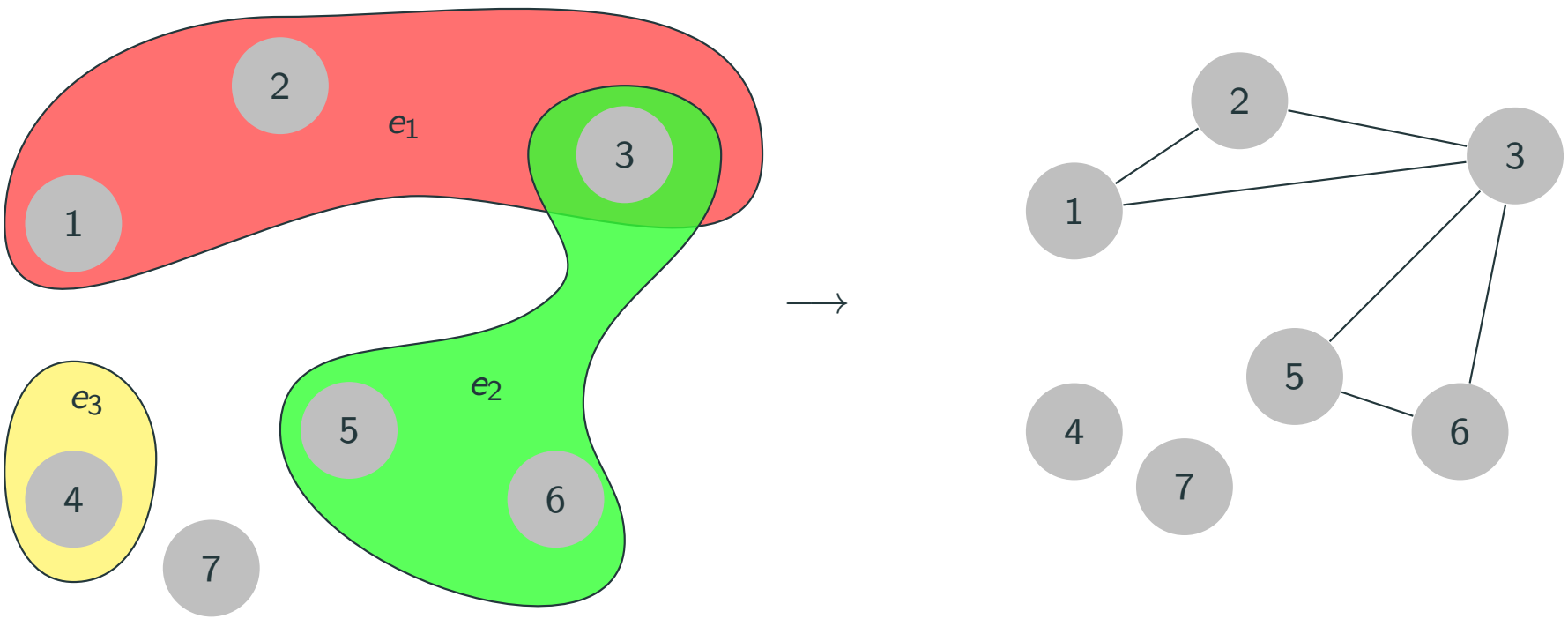
The following constructions are standard.

- clique expansion
  - the vertex set is V
  - each hyperedge e is replaced by an edge for every pair of vertices in e
  - this construction yields cliques for every hyperedge
- star expansion
  - vertex set is $V \cup E$
  - edge between u and e iff $u \in e$
  - every hyperedge corresponds to a star
- there are more...

Clique expansion

The clique expansion $G^x = (V^x, E^x)$ is constructed from $H = (V, E)$ via:

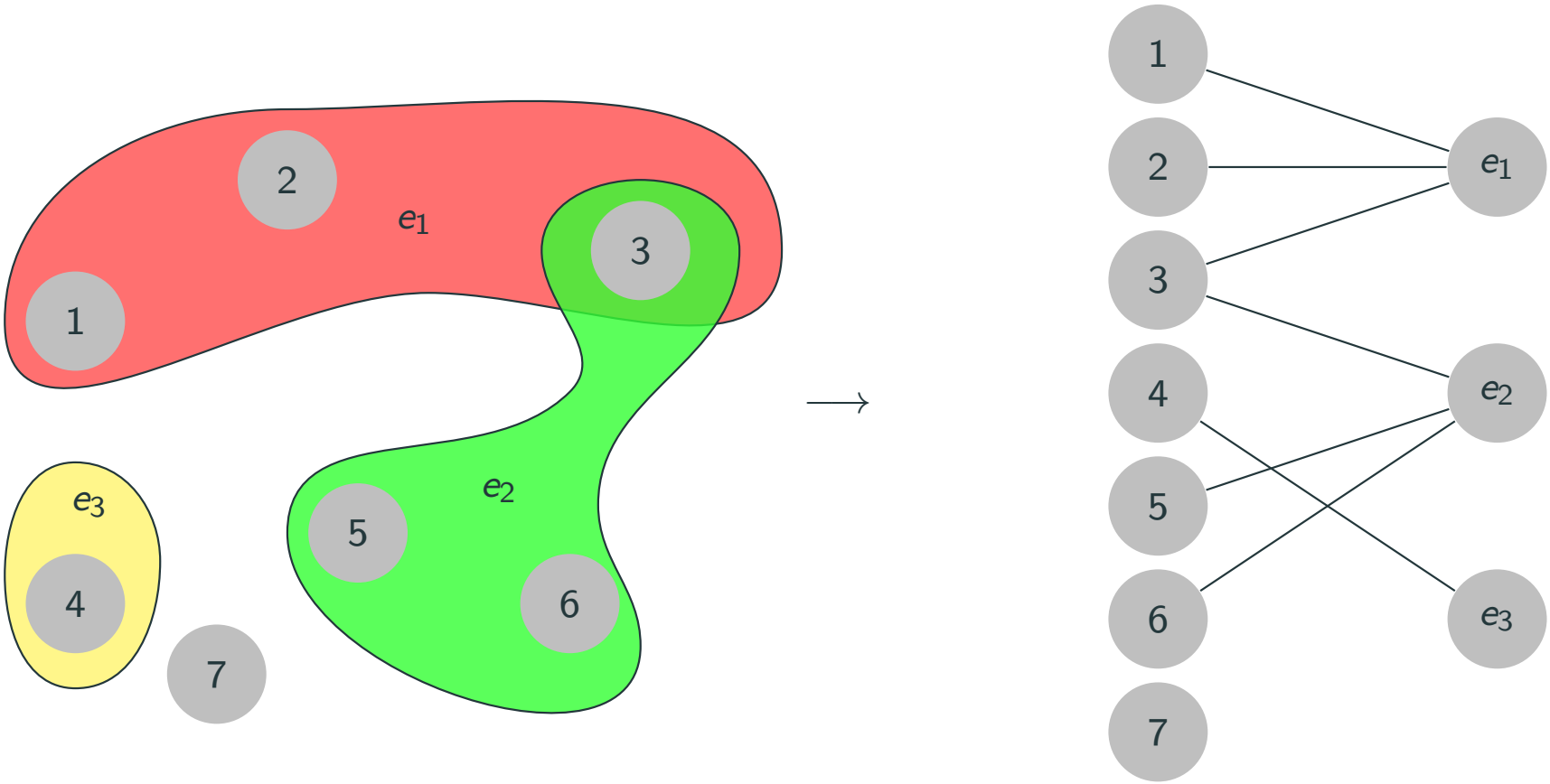
- $V^x = V$
- $E^x = \{\{i, j\} : \exists e \in E \text{ with } i, j \in e\}$



Star expansion

The star expansion $G^* = (V^*, E^*)$ is constructed from $H = (V, E)$ via:

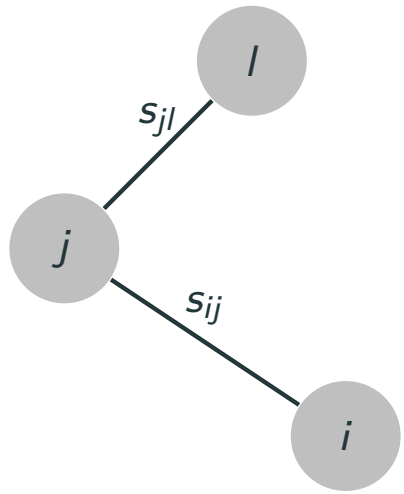
- $V^* = V \cup E$
- $E^* = \{\{i, e\} : i \in e, e \in E\}$



What if data without network structure is given?

Solution: Build your own graph!

- given a set of data points $x_1, x_2 \ldots, x_n$ and some notion of similarity[1] $s_{ij} \geq 0$ between all pairs of data points x_i and x_j
- build graph $G = (V, E)$, where the vertex i represents the data point x_i, so $V = \{1, 2, \ldots, n\}$
- $\{i, j\} \in E$ if $s_{ij} > 0$
- edge weight $\omega_{ij} = s_{ij}$ (edge weights represent similarities)
- G is called **similarity graph** (although with this particular choice of edges it is often referred to as the **fully connected graph**)



graph for $\{x_i, x_j, x_l\}$ with $s_{ij}, s_{jl} > 0$ and $s_{il} = 0$

The ε-neighbourhood graph

The ε-**neighbourhood graph** is constructed as follows:

- vertices are data points
- fix some $\varepsilon > 0$
- connect all vertices whose similarities are smaller than ε
- since ε is usually small, values of existing edges are roughly of the same scale
- hence usually unweighted

The (mutual) k-nearest neighbour graph

The k-**nearest neighbour graph** is constructed as follows:

- vertices are data points
- fix some $k > 0$
- connect i to the k nearest (w.r.t. s_{ij}) k vertices via an edge starting at i
- obtain an undirected graph by ignoring the directions

The **mutual** k-**nearest neighbour graph** is constructed as follows:

- vertices are data points
- fix some k
- connect i to the k nearest (w.r.t. s_{ij}) k vertices via an edge starting at i
- obtain an undirected graph by deleting all non symmetric edges

In both cases weights are just the similarities.