# CS6208 : Advanced Topics in Artificial Intelligence
# Graph Machine Learning

## Lecture 3 : Graph Clustering

Semester 2 2022/23


Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)


# Course lectures

- Introduction to Graph Machine Learning
- Part 1: GML without feature learning (before 2014)
  - Introduction to Graph Science
  - Graph Analysis Techniques without Feature Learning
    - Graph clustering
    - Classification
    - Recommendation
    - Dimensionality reduction
- Part 2 : GML with shallow feature learning (2014-2016)
  - Shallow graph feature learning
- Part 3 : GML with deep feature learning, a.k.a. GNNs (after 2016)
  - Graph Convolutional Networks (spectral and spatial)
  - Weisfeiler-Lehman GNNs
  - Graph Transformer & Graph ViT/MLP-Mixer
  - Benchmarking GNNs
  - Molecular science and generative GNNs
  - GNNs for combinatorial optimization
  - GNNs for recommendation
  - GNNs for knowledge graphs
  - Integrating GNNs and LLMs

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Unsupervised data clustering

- Unsupervised learning aims at designing predictive algorithms without data labels, i.e. no prior information about data classes or data properties to regress.
- Instead, these algorithms rely on general assumptions on the data distribution s.a. linearly separable data, or the task at hand e.g. identifying well-separated clusters.
- This lecture exclusively focuses on unsupervised algorithms for data clustering and graph partitioning.
- k-means[1] is the most popular unsupervised data clustering algorithm.
- Ncut[2] and Metis[3] are the most prominent techniques for unsupervised graph partitioning.
- We will reveal their underlying relationship and similarities.

[1] Lloyd, Least square quantization in PCM, 1957
[2] Shi, Malik, Normalized cuts and image segmentation, 2000
[3] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998

# Standard k-means

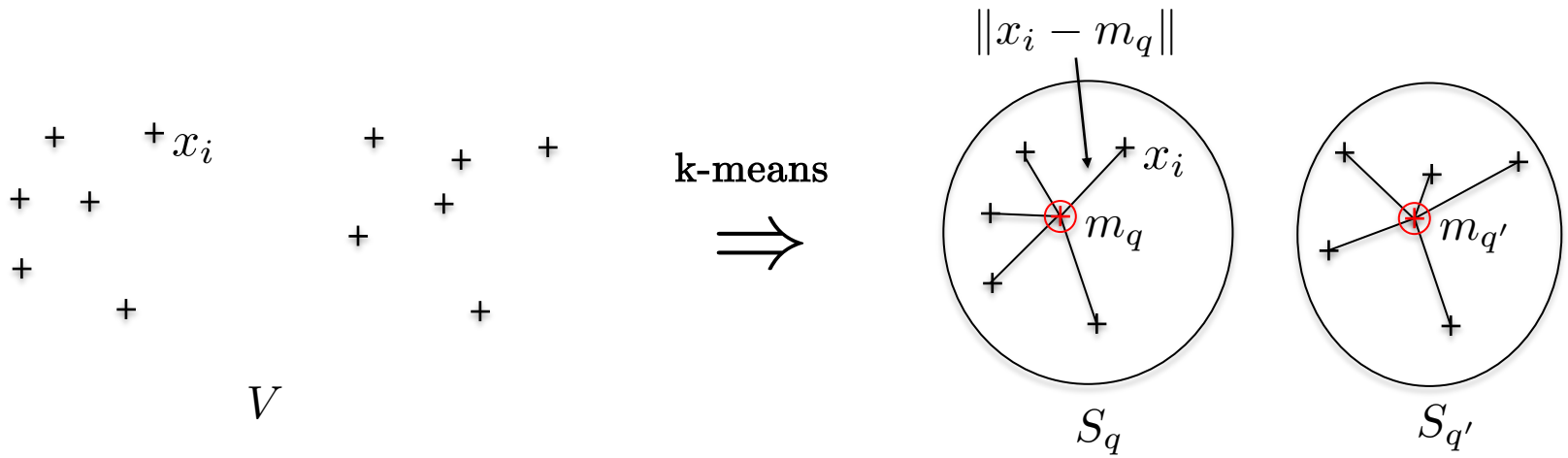
- Given n data points $V = \{x_i\}_{i=1}^{n} \in R^d$, k-means technique partitions the dataset into k clusters $\{S_1,...,S_k\}$ with their means $\{m_1,...,m_k\} \in R^d$ that minimize the least-squares objective, where the number k of clusters is arbitrary selected, i.e. not estimated :

$$L_{\text{k-means}}(\{m_q\}_{q=1}^{k}, \{S_q\}_{q=1}^{k}) = \frac{1}{kn} \sum_{q=1}^{k} \sum_{i \in S_q} \|x_i - m_q\|_2^2$$

Distance between data $x_i$ and its mean $m_q$

$q^{th}$ mean

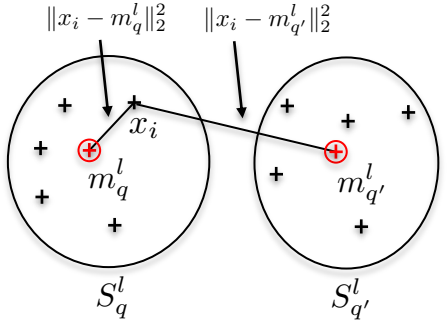
$q^{th}$ cluster



# EM algorithm

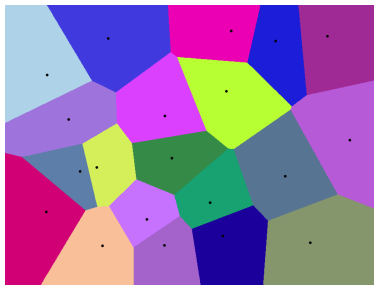
- Expectation-maximization (EM) technique[1,2]
- Initialization
  - Randomly select initial means $\{m_1,...,m_k\} \in R^d$ to be a data point in V (efficient).
  - Good initialization with k-means++[3] (with some guarantee w.r.t. optimal solution).
- Iterate until convergence : l = 0,1,2,...
  - Cluster update (expectation step)

$$S_q^{l+1} = \{x_i \in V \text{ s.t. } \|x_i - m_q^l\|_2^2 \leq \|x_i - m_{q'}^l\|_2^2, \ \forall q' \neq q\}$$




  - Mean update (maximization step)

$$m_q^{l+1} = \frac{\sum_{x_i \in S_q^{l+1}} x_i}{|S_q^{l+1}|} \in \mathbb{R}^d$$



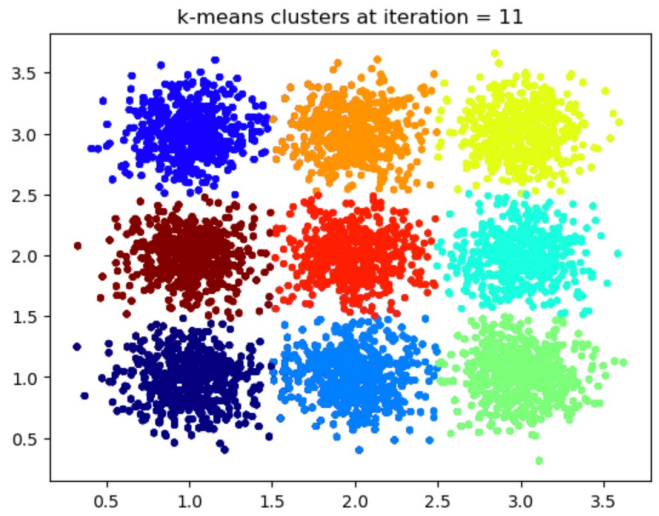
Voronoi's cells


[1] Lloyd, Least square quantization in PCM, 1957
[2] Forgy, Cluster analysis of multivariate data: efficiency versus interpretability of classifications, 1965
[3] Bradley, Fayyad, Refining Initial Points for k-Means Clustering, 1998

# Lab 1 : Standard k-means

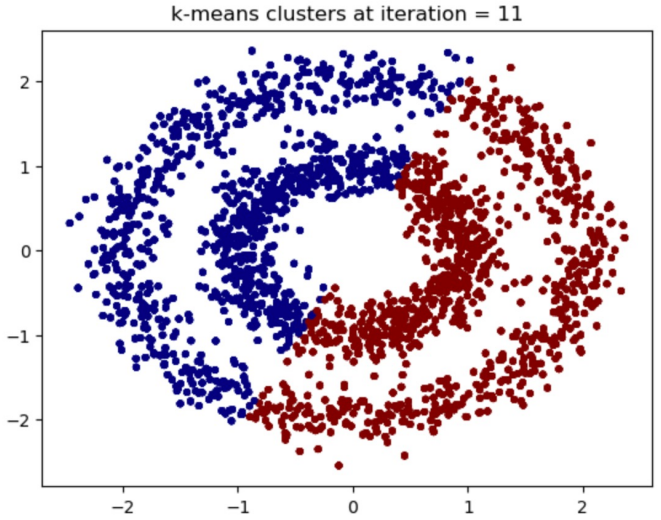
- Run code01.ipynb and analyze k-means result on
  - Linearly separable data points
  - Non-linear data points

[6]:
```
# Visualize k-means iterations
fig, ax = plt.subplots()
for k,C in enumerate(Clusters_iters):
    plt.scatter(X[:,0], X[:,1], s=10*np.ones(n), c=C, cmap='jet')
    plt.title('k-means clusters at iteration = ' + str(k+1) )
    display(fig)
    clear_output(wait=True)
```



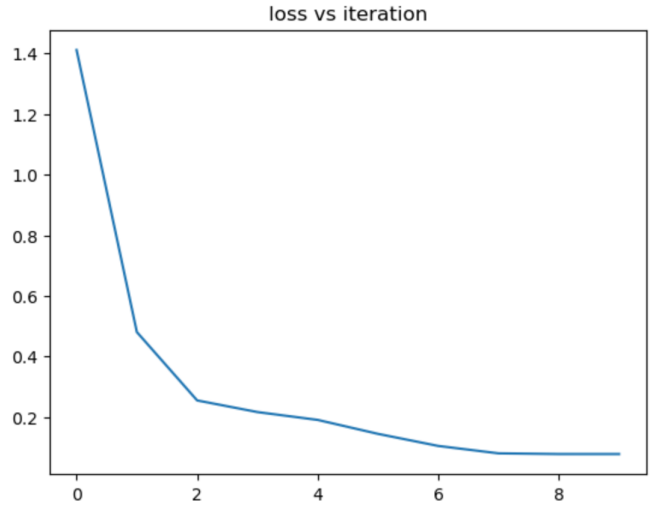

Linearly separable data points

```
# Visualize k-means iterations
fig, ax = plt.subplots()
for k,C in enumerate(Clusters_iters):
    plt.scatter(X[:,0], X[:,1], s=10*np.ones(n), c=C, cmap='jet')
    plt.title('k-means clusters at iteration = ' + str(k+1) )
    display(fig)
    clear_output(wait=True)
```



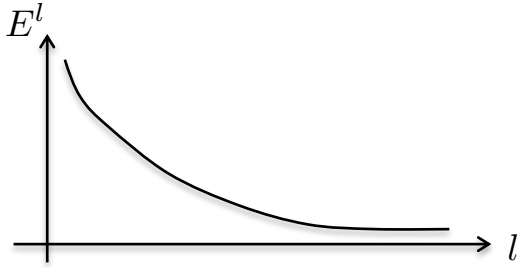

Non-linear data points

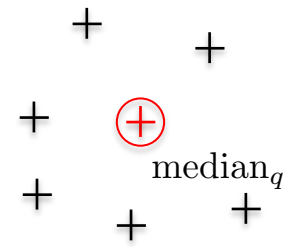
[7]:
```
# Visualize loss vs iteration
plt.figure(3)
plt.plot(En_iters)
plt.title('loss vs iteration')
plt.show()
```

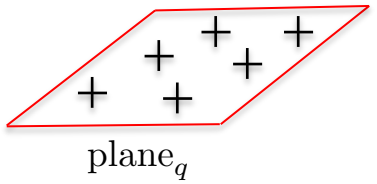


# Algorithm properties

- Advantages of the EM algorithm
  - Monotonic : $L^{l+1} \leq L^l$ for all iterations.
  - Convergence (to local minimizer) is guaranteed.
  - Speed complexity is $O(n.d.k.n_i)$,
    - where n is the number of data points, d is the data dimension, k is the number of clusters and $n_i$ is the number of iterations to convergence.
  - Easy to implement and GPU friendly.
  - Several extensions exist : k-medians[1], k-planes[2], other distances
  - k-means shares interesting connections with other important algorithms s.a. Gaussian Mixture Model (GMM)[3], PCA[4].



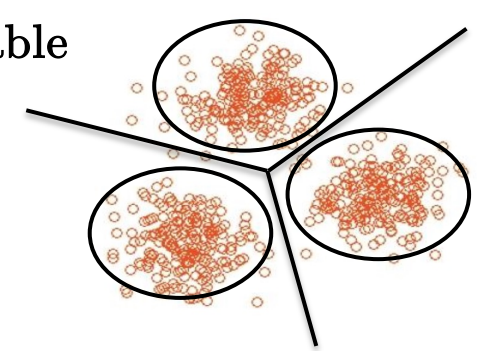





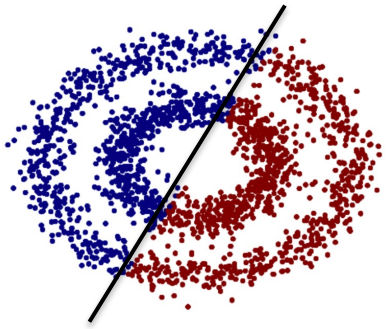
[1] Jain, Dubes, Algorithms for Clustering Data. Prentice-Hall, 1988
[2] Bradley, Mangasarian, k-plane Clustering, 2000
[3] Murphy, Machine learning : a probabilistic perspective, 2012
[4] Pearson, On Lines and Planes of Closest Fit to Systems of Points in Space, 1901

# Algorithm properties

- Limitations of the EM algorithm
  - k-means problem is NP-hard combinatorial (as all clustering problems!).
  - Initialization is thus critical for good performance :
    - Requires a good initial guess[1]
    - Alternatively, restart several times with different initializations, and pick the solution with the lowest loss value.
  - Assumption: Standard k-means assume that data points are linearly separable, s.a. dataset follows a GMM, i.e. clusters are linearly separable and spherical.
  - Standard k-means does not work for non-linear separable
  - Solution : Non-linear k-means, a.k.a. kernel k-means.



Linearly separable data points



Non-linear data points

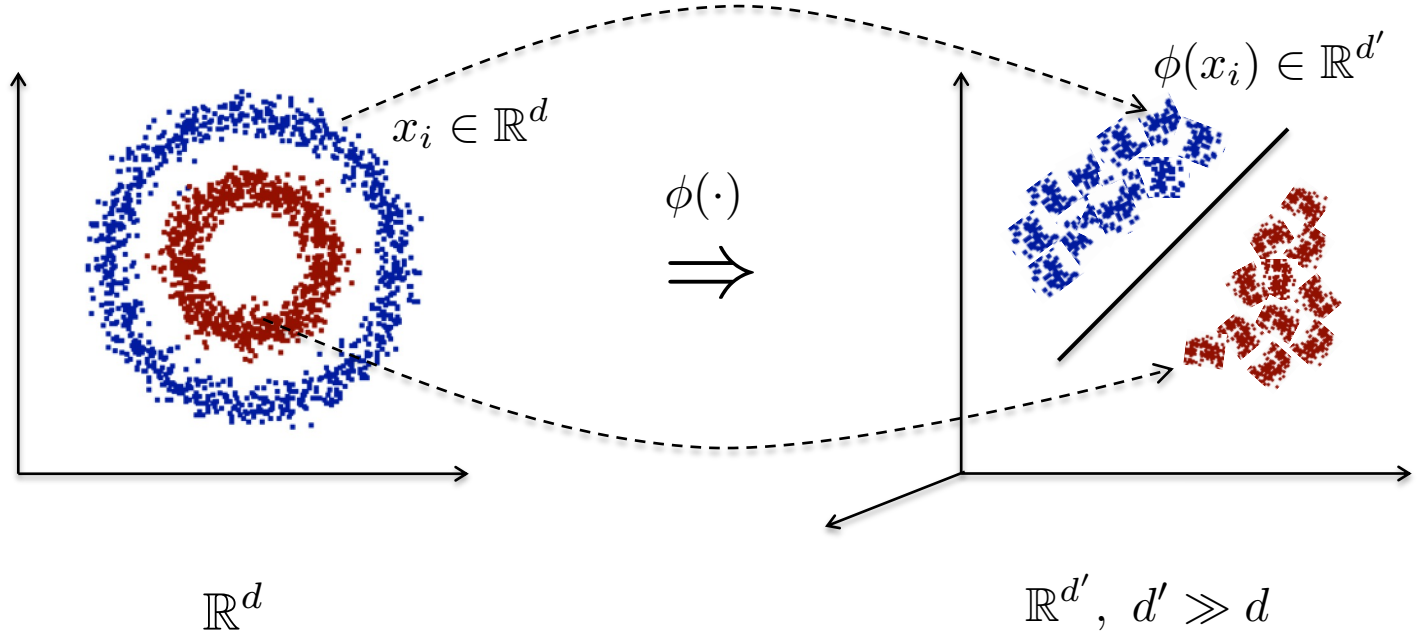
[1] Bradley, Fayyad, Refining Initial Points for k-Means Clustering, 1998

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Higher dimensional projection

- To separate non-linear data points, a common technique involves lifting the original data features to higher-dimensional spaces where the data points become linearly separable[1,2].
- This process is achieved using a so-called non-linear feature function, denoted as $\phi(.)$ :





[1] Aizerman et-al, Theoretical foundations of the potential function method in pattern recognition learning, 1964
[2] Guyon, Boser, Vapnik, Automatic capacity tuning of very large VC-dimension classifiers, 1993

# Non-linear k-means

- We update the original loss function of linear k-means by incorporating :
  - The non-linear mapping : $x \in \mathrm{R}^{\mathrm{d}} \rightarrow \phi(x) \in \mathrm{R}^{\mathrm{d}'}$ , with $\mathrm{d}' \gg \mathrm{d}$
  - Weights $\theta_i \in \mathrm{R}_+$ which control the importance of each data sample.

Weight contribution for data $\mathrm{x_i}$

$$L_{\text{nl-k-means}}(\{m_q\}_{q=1}^k, \{S_q\}_{q=1}^k) = \frac{1}{kn} \sum_{q=1}^{k} \sum_{i \in S_q} \theta_i \|\phi(x_i) - m_q\|_2^2$$

$x_i \rightarrow \phi(x_i)$

- We will introduce two approaches to minimize the non-linear k-means loss
  - EM approach
  - Spectral approach

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Indicator function of clusters

- We introduce an indicator matrix F representing the clusters $S_q$ :
  - F makes easy to use GPU for representing clusters.
  - Later, F will simplify the transition from combinatorial optimization to continuous optimization

$$F = \begin{matrix} \\ S_1 \\ \\ \\ S_2 \\ \\ \\ S_3 \end{matrix} \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{k} \in \{0,1\}^{n \times k} \quad \Big\updownarrow n$$

$F_{.,q}$ = Indicator function of the set $S_q$

# Non-linear k-means loss

- **Matrix formulation of the non-linear k-means loss[1] w.r.t. the indicator matrix F :**

$$
\begin{aligned}
L_{\text{nl-k-means}}(\{m_q\}_{q=1}^k, \{S_q\}_{q=1}^k) &= \frac{1}{kn} \sum_{q=1}^{k} \sum_{i \in S_q} \theta_i \|\phi(x_i) - m_q\|_2^2 \\
&= \frac{1}{kn} \sum_{q=1}^{k} \sum_{i \in S_q} \theta_i D_{iq}, \text{ with } D_{iq} = \|\phi(x_i) - m_q\|_2^2 \\
&= \frac{1}{kn} \operatorname{tr}(F^T \Theta D) = L_{\text{nl-k-means}}(F) \\
&\text{with } F \in \{0,1\}^{n \times k} \\
&\quad \Theta = \operatorname{diag}(\theta_1, ..., \theta_n) \in \mathbb{R}^{n \times n} \\
&\quad D \in \mathbb{R}^{n \times k}
\end{aligned}
$$

[1] Scholkopf, Smola, Muller, Nonlinear component analysis as a kernel eigenvalue problem, 1998 (10,000 citations as of 2023)

# Distance metric

- **Distance $D_{iq}$ between data $x_i$ and its mean $m_q$ :**

$$\begin{aligned} D_{iq} = \|\phi(x_i) - m_q\|_2^2 &= (\phi(x_i) - m_q)^T(\phi(x_i) - m_q) \\ &= \phi(x_i)^T\phi(x_i) - 2\phi(x_i)^T m_q + m_q^T m_q \\ &= K_{ii} - 2A_{iq} + B_{qq} \in \mathbb{R} \end{aligned}$$

with

$$K_{ii} = \phi(x_i)^T\phi(x_i)$$

$$A_{iq} = \phi(x_i)^T m_q$$

$$B_{qq} = m_q^T m_q$$

and

$$\frac{\partial L_{\text{nl-k-means}}}{\partial m_q} = 0 \quad \Rightarrow \quad m_q = \frac{\sum_{i \in S_q} \theta_i \phi(x_i)}{\sum_{i \in S_q} \theta_i} \in \mathbb{R}^{d'}$$

# Distance metric

- **Matrix formulation of metric distance D :**
  - Point-wise distance metric : $D_{iq} = K_{ii} - 2A_{iq} + B_{qq} \in \mathbb{R}$
  - First term : $K_{ii} = (\phi(x)\phi(x)^T)_{ii},\ \phi(x) \in \mathbb{R}^{n \times d'} \Rightarrow K = \phi(x)\phi(x)^T \in \mathbb{R}^{n \times n}$
  - Second term :

$$A_{iq} = \phi(x_i)^T m_q = \phi(x_i)^T \frac{\sum_{j \in S_q} \theta_j \phi(x_j)}{\sum_{j \in S_q} \theta_j} = \frac{\sum_{j \in S_q} \theta_j \phi(x_i)^T \phi(x_j)}{\sum_{j \in S_q} \theta_j} = \frac{\sum_{j \in S_q} \theta_j K_{ij}}{\sum_{j \in S_q} \theta_j}$$

$$\Rightarrow A = \phi(x)M^T \in \mathbb{R}^{n \times k} = \phi(x)\phi(x)^T \Theta F Z = K \Theta F Z \in \mathbb{R}^{n \times k},$$

$$\text{with } M = ZF^T \Theta \phi(x) \in \mathbb{R}^{k \times d'} \text{ and } Z^{-1} = \text{diag}(1_n^T \Theta F) \in \mathbb{R}^{k \times k}$$

  - Third term : $B_{qq} = (MM^T)_{qq} \Rightarrow B = ZF^T \Theta \phi(x)(ZF^T \Theta \phi(x))^T$

$$= ZF^T \Theta \phi(x)\phi(x)^T \Theta F Z = ZF^T \Theta K \Theta F Z$$

  - Finally, matrix-based distance metric :

$$D = \text{diag}(K)1_k^T - 2K\Theta FZ + 1_n \text{diag}(ZF^T \Theta K \Theta F Z) \in \mathbb{R}^{n \times k}$$

# EM algorithm

- Initialization
  - Random initial indicator $F^{l=0}$ of clusters.
- Iterate until convergence : l = 0,1,2,...
  - Cluster update (expectation step)

$$S_q^{l+1} = \{x_i \in V \text{ s.t. } D_{iq}^l \leq D_{iq'}^l,\ \forall q' \neq q\},\ D_{iq}^l = \text{dist}(x_i, m_q) = \|x_i - m_q^l\|_2^2$$

$$\Updownarrow$$

$$F_{iq}^{l+1} = \begin{cases} 1 & \text{if } D_{iq}^l = \min_{q'} D_{iq'}^l \\ 0 & \text{otherwise} \end{cases}$$

Equivalently, $S_q^{l+1} = \{x_i \in V \text{ s.t. } F_{iq}^{l+1} = 1\}$

  - Mean update (maximization step)
    - No explicit mean update required! It is implicitly done when computing D.

# Kernel trick

- Non-linear mapping $\phi(\cdot)$ enables the separation of non-linear data points.
- However, it comes with the price to apply k-means in a larger feature space than the original one. This leads to an increased complexity of $O(nkd'n_c)$ with $d' \gg d$.
- To address this issue, the kernel trick was developed, which avoids the explicit use of the mapping $\phi$.
  - Observe that computing the distance D uses the quadratic matrix $\phi\phi^T$, rather than $\phi$ individually, thus the precise expression of $\phi$ becomes irrelevant.
  - We define the kernel operator/matrix $K = \phi\phi^T$ with standard definitions as follows :

Time consuming

$$K(x_i, x_j) = x_i^T x_j \quad \text{(linear kernel for linear k-means)}$$

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \exp(-\|x_i - x_j\|_2^2/\sigma^2) \quad \text{(Gaussian kernel)}$$

$$K(x_i, x_j) = (ax_i^T x_j + b)^c \quad \text{(Polynomial kernel)}$$

Efficient kernel computation

[1] Aizerman et-al, Theoretical foundations of the potential function method in pattern recognition learning, 1964
[2] Guyon, Boser, Vapnik, Automatic capacity tuning of very large VC-dimension classifiers, 1993

# Algorithm properties

- Advantages
  - Similar to standard k-means, algorithm is monotonous and guaranteed to converge.
  - Distance computation relies on linear algebra, i.e. matrix-matrix multiplication.
  - Fast libraries such as LAPACK/BLAS for Intel and AMD, as well as CUDA for GPU, provide efficient computation.
- Limitations
  - The kernel matrix K is full : Memory complexity is $O(n^2)$ and speed is $O(n^3)$.
  - Solutions are typically local minimizers.
  - Can we obtain global minimizers?

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Non-linear k-means loss

- We reformulate the non-linear k-means loss[1] with the mean $m_q$ w.r.t. each data point $x_i$ :
  - For EM means, we have the means w.r.t. the clusters : $m_q \Rightarrow M \in R^{k \times d'}$
  - For spectral means, we will consider the alternative representation : $m_i \Rightarrow M \in R^{n \times d'}$

$$
\begin{aligned}
L_{\text{nl-k-means}}(\{m_q\}_{q=1}^k, \{S_q\}_{q=1}^k) &= \frac{1}{kn} \sum_{q=1}^{k} \sum_{i \in S_q} \theta_i \|\phi(x_i) - m_q\|_2^2 \\
&= \frac{1}{kn} \sum_{q=1}^{k} \sum_{i \in S_q} \|\theta_i^{1/2}\phi(x_i) - \theta_i^{1/2} m_q\|_2^2, \ \phi(x_i), m_q \in \mathbb{R}^{d'} \\
&= \frac{1}{kn} \sum_{i \in V} \|\theta_i^{1/2}\phi(x_i) - \theta_i^{1/2} m_i\|_2^2, \ m_i \in \mathbb{R}^{d'} \\
&= \frac{1}{kn} \sum_{i \in V} \|\theta_i^{1/2}\phi(x_i) - \theta_i^{1/2} (M)_{i,\cdot}\|_2^2, \ M \in \mathbb{R}^{n \times d'} \\
&= \frac{1}{kn} \|\theta^{1/2}\phi(x) - \theta^{1/2} M\|_F^2, \\
&\text{with } M = FZF^T \Theta \phi(x) \in \mathbb{R}^{n \times d'}
\end{aligned}
$$

[1] Dhillon, Guan, Kulis, Kernel k-means: spectral clustering and normalized cuts, 2004

# New indicator matrix of clusters

- **We introduce a new indicator matrix of clusters that forms an orthonormal basis :**

$$Y = \Theta^{1/2} F Z^{1/2} \in \mathbb{R}^{n \times k} \quad \text{s.t.} \quad Y^T Y = \mathrm{I}_k \in \mathbb{R}^{k \times k} \qquad \text{(matrix-based representation)}$$

$$Y_{ik} = \begin{cases} \sqrt{\frac{\theta_i}{\sum_{j \in S_q} \theta_j}} & \text{if } i \in S_q \\ 0 & \text{otherwise} \end{cases} \qquad \text{(point-wise representation)}$$

which implies

$$F = \Theta^{-1/2} Y Z^{-1/2} \in \mathbb{R}^{n \times k}$$

and

$$\begin{aligned} M = FZF^T \Theta \phi(x) &= \Theta^{-1/2} Y Z^{-1/2} Z Z^{-1/2} Y^T \Theta^{-1/2} \Theta \phi(x) \\ &= \Theta^{-1/2} Y Y^T \Theta^{1/2} \phi(x) \in \mathbb{R}^{n \times d'} \end{aligned}$$

# Non-linear k-means loss

- **We rewrite the non-linear k-means loss function with the new indicator Y :**

$$
\begin{aligned}
L_{\text{nl-k-means}}(Y) &= \frac{1}{kn} \|\Theta^{1/2}\phi(x) - \Theta^{1/2}M\|_F^2, \\
&\quad \text{with } M = \Theta^{-1/2}YY^T\Theta^{1/2}\phi(x) \\
&= \frac{1}{kn} \|\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x)\|_F^2 \\
&\quad \text{s.t.} \quad Y^TY = \mathrm{I}_k, \; Y \in \text{binary}^{n\times k}
\end{aligned}
$$

# Relaxation

- Let us consider the combinatorial optimization problem :

$$\min_{Y \in \text{binary}^{n \times k}} \|\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x)\|_F^2 \quad \text{s.t.} \quad Y^TY = \mathrm{I}_k$$

- Observe that the binary constraint is what makes the optimization challenging, actually rendering the problem as NP-hard.
- By relaxing the non-convex binary constraint, i.e. $\text{binary}^{n\,x\,k}$, to a convex one, i.e. $\mathrm{R}^{n\,x\,k}$, the optimization becomes continuous :

$$\min_{Y \in \mathbb{R}^{n \times k}} \|\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x)\|_F^2 \quad \text{s.t.} \quad Y^TY = \mathrm{I}_k$$

- This transition allows for a mathematically well-posed solution, as defined by the spectral theorem[1,2].

[1] Helmberg, Introduction to Spectral Theory in Hilbert Space, 1969
[2] Hawkins, Cauchy and the spectral theory of matrices, 1975

# Spectral loss

- **We simplify the new optimization problem :**

$$\min_{Y \in \mathbb{R}^{n \times k}} \|\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x)\|_F^2 \quad \text{s.t.} \quad Y^TY = \mathrm{I}_k$$

with

$$\begin{aligned}
&\|\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x)\|_F^2 \\
&= \mathrm{tr}\big((\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x))^T(\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x))\big) \\
&= \mathrm{tr}\big(\Theta^{1/2}\phi(x)^T\phi(x)\Theta^{1/2} - \Theta^{1/2}\phi(x)^T\phi(x)YY^T - \\
&\qquad\qquad YY^T\Theta^{1/2}\phi(x)^T\phi(x) + YY^T\Theta^{1/2}\phi(x)^T\phi(x)YY^T\big)
\end{aligned}$$

and using $K = \phi(x)^T\phi(x), \mathrm{tr}(AB) = \mathrm{tr}(BA), \mathrm{tr}(A+B) = \mathrm{tr}(A) + \mathrm{tr}(B), Y^TY = \mathrm{I}$

$$= \mathrm{tr}(\Theta^{1/2}K\Theta^{1/2}) - \mathrm{tr}(Y^T\Theta^{1/2}K\Theta^{1/2}Y)$$

Finally, we have

$$\min_{Y \in \mathbb{R}^{n \times k}} -\mathrm{tr}(Y^T\Theta^{1/2}K\Theta^{1/2}Y) \quad \text{s.t.} \quad Y^TY = \mathrm{I}_k$$

$$\max_{Y \in \mathbb{R}^{n \times k}} \mathrm{tr}(Y^T\Theta^{1/2}K\Theta^{1/2}Y) \quad \text{s.t.} \quad Y^TY = \mathrm{I}_k$$

# Spectral theorem

- **Spectral solution[1] of the new continuous optimization problem is given by the k largest eigenvectors of matrix A obtained by eigenvalue decomposition (EVD) :**

$$\max_{Y \in \mathbb{R}^{n \times k}} \operatorname{tr}(Y^T A Y) \quad \text{s.t.} \quad Y^T Y = \mathrm{I}_k$$

$$\text{with} \quad A = \Theta^{1/2} K \Theta^{1/2} \overset{\text{EVD}}{=} U \Lambda U^T \in \mathbb{R}^{n \times n}$$

$$\text{and solution} \quad Y^\star = U_{\cdot,1:k} \in \mathbb{R}^{n \times k} \quad (k \text{ largest eigenvectors})$$

[1] Helmberg, Introduction to Spectral Theory in Hilbert Space, 1969
[2] Hawkins, Cauchy and the spectral theory of matrices, 1975

# Understand the spectral theorem

- **Suppose a matrix A is symmetric and positive semi-definite (PSD).**
- **All kernel matrices, denoted as K, possess these properties by construction.**
- **Then, the eigenvalue decomposition of A can be expressed as :**

$$Ay_q = \lambda_q y_q, \;\; A \in \mathbb{R}^{n \times n}, y_q \in \mathbb{R}^n$$

with

$$\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq ... \geq \lambda_k \geq ... \geq \lambda_n, \text{ where } \lambda_1, ..., \lambda_k \text{ are the largest eigenvalues}$$

$$y_q^T y_{q'} = \begin{cases} 1 & \text{if } q = q' \\ 0 & \text{otherwise} \end{cases} \quad \Leftrightarrow \quad Y^T Y = \mathrm{I}_k$$

We have

$$\mathrm{tr}(Y^T A Y) = \sum_{q=1}^{k} y_q^T A y_q = \sum_{q=1}^{k} y_q^T \lambda_q y_q = \sum_{q=1}^{k} \lambda_q y_q^T y_q = \sum_{q=1}^{k} \lambda_q \text{ (k largest values)}$$

$$= \max_{Y} \; \mathrm{tr}(Y^T A Y) \;\; \text{s.t.} \;\; Y^T Y = \mathrm{I}_k$$

[1] Meyer, Matrix Analysis and Applied Linear Algebra, 2000

# Spectral clustering

- No initialization needed
- No iterative scheme needed
- Compute clusters as follows :
  - Compute matrix A : $A = \theta^{1/2} K \theta^{1/2}$
  - Perform eigenvalue decomposition of A : $Au_q = \lambda_q u_q$, $u_q \in R^{n \times k}$, $1 \leq q \leq k$
  - Form spectral solution : $Y^* = (u_1,..,u_k) \in R^{n \times k}$
  - Binarize spectral solution $Y^*$ :
    - Generally, solution $Y^*$ is not binary, i.e. no cluster can be directly identified.
    - Consider $Y^*$ as embedding coordinates of X and
    - Apply the standard k-means on $Y^*$ to identify k clusters.
- This algorithm is known as spectral clustering[1].

[1] Von Luxburg, A tutorial on spectral clustering, 2007 (w/ 10k citations as of 2023)

# Algorithm properties

- Advantages
  - Provides a global solution, i.e. independent of any initialization.
  - Offers solutions that perform well in practice.
- Limitations
  - Computes an approximate solution to the original NP-hard combinatorial optimization problem (by relaxing the indicator constraint).
  - Complexity is $O(n^2k)$, which does not scale w.r.t. the number n of data points.
  - Spectral techniques with EVD/SVD (singular value decomposition generalized EVD to non-square and non-positive semi-definite matrices) are commonly used in data analysis due to their well-understood theory.
  - However, these techniques suffer from scalability issues w.r.t. the number n of data points as EVD complexity is $O(n^2k)$, although stochastic versions of EVD/SVD have been developed with linear complexity[1].

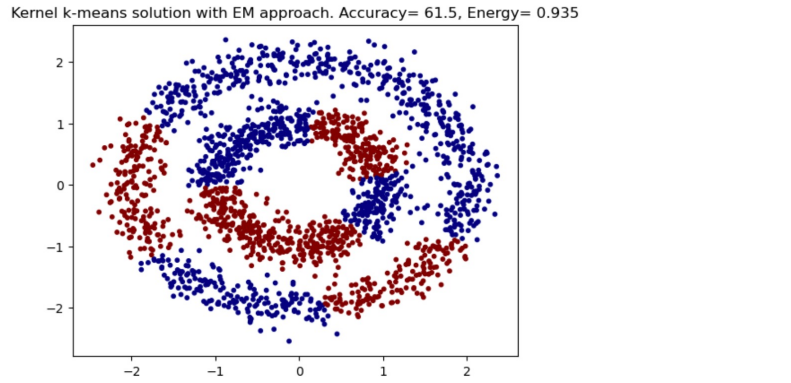
[1] Mahoney, Drineas, Randomized Algorithms for the Low-Rank Approximation of Matrices, 2010

# Lab 2 : Kernel k-means

- Run code02.ipynb and test kernel k-means with
  - EM approach
  - Spectral approach

```
# Run kernel/non-linear k-means with EM approach

# Compute linear Kernel for standard k-means
Ker = construct_kernel(X, 'kNN_gaussian', 100)
print(Ker.shape)

# Kernel k-means with EM approach
C_kmeans, En_kmeans = compute_kernel_kmeans_EM(nc, Ker, Theta, 10)

# Plot
plt.figure(3)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_kmeans, cmap='jet')
plt.title('Kernel k-means solution with EM approach. Accuracy= ' + str(compute_purity(C_kmeans,Cgt,nc))[:5] +
          ', Energy= ' + str(En_kmeans)[:5])
plt.show()
```

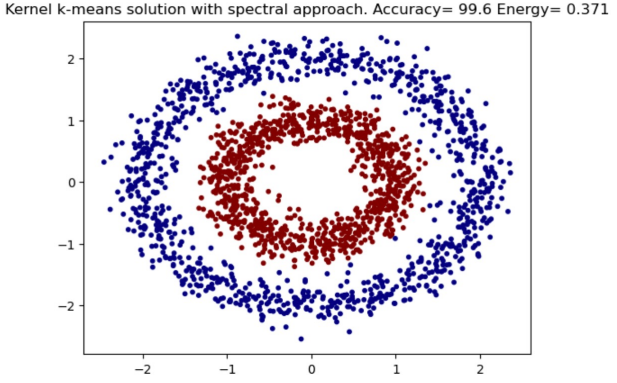
```
Construct kNN Gaussian Kernel
(2000, 2000)
```




Kernel k-means
EM approach

```
# Run kernel/non-linear k-means with spectral approach

# Compute linear kernel for standard k-means
Ker = construct_kernel(X, 'kNN_gaussian', 100)
print(Ker.shape)

# Kernel k-means with spectral approach
C_kmeans, En_kmeans = compute_kernel_kmeans_spectral(nc, Ker, Theta, 10)

# Plot
plt.figure(4)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_kmeans, cmap='jet')
plt.title('Kernel k-means solution with spectral approach. Accuracy= ' +
          str(compute_purity(C_kmeans,Cgt,nc))[:5] + ' Energy= ' + str(En_kmeans)[:5])
plt.show()
```

```
Construct kNN Gaussian Kernel
(2000, 2000)
Construct Linear Kernel
```




Kernel k-means
Spectral approach

# Outline

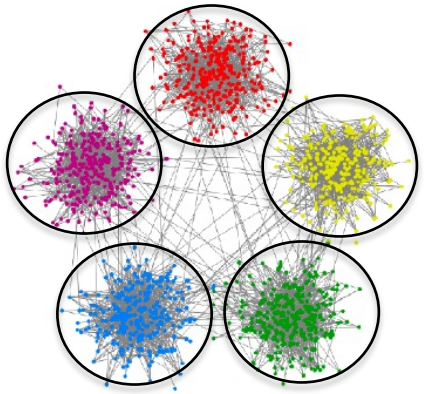
- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Data clustering

- Lloyd's EM k-means has $O(ndkn_i)$ complexity but is constrained to linear data distribution.
- Kernel's EM algorithm is $O(n^2d+n^2kn_i)$ suitable for non-linear data but with critical initialization.
- Spectral kernel k-means is $O(n^2d+n^2k)$ providing a global solution, but without guarantee of being the original combinatorial solution.
- Non-linear techniques cannot scale to millions of data points due to the full kernel matrix.
- Addressing large-scale datasets, s.a. clustering 2.8 billion monthly active Facebook users (2023) or 61 million Wikipedia articles (2023), requires a new approach.
- While the kernel operator K is full, most pairs of data points are actually not correlated.
- Graphs can offer a promising solution as it can represent sparse relationships between data.
- Interestingly, we will see that the task of data clustering is equivalent to graph partitioning.

# Graph partitioning

- Partitioning graphs is a cornerstone problem, not only for
  - Identifying connected groups, e.g. users on social networks (exploration tool to find patterns), but also plays a crucial role in
  - Balanced graph partitioning for efficient distributed processing of large-scale graphs, s.a. computing Google PageRank w/ billions of nodes.
- A notable class of unsupervised graph clustering techniques is balanced cut algorithms.
- Balanced graph cuts are instrumental in both
  - Graph theory : Define classes of networks and their properties.
  - Applications : State-of-the-art methods for unsupervised clustering e.g. Metis[1].

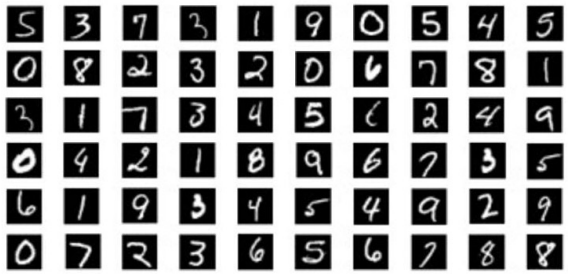


Source: Abbe, JMLR'17

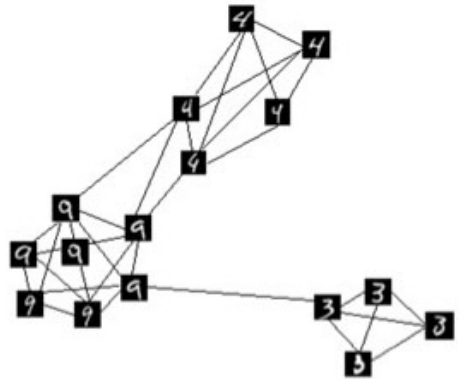
[1] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998

# Data clustering as graph partitioning

- Graph construction : Build a (sparse) k-nearest neighbors (k-NN) graph G=(V,E,A) from a dataset $V=\{x_1,...,x_n\} \in R^d$.
- The graph representation of the dataset avoids working with d-dim features directly.
- Essentially, this process transforms the n x d data features into a set of E edges, resulting in a significant compression of the dataset.
- Exact graph construction complexity is $O(n^2d)$, but faster approximate techniques exist[1].
- Memory complexity for graphs is O(E), w/ $E \ll n^2$ for real-world graphs s.a. E = O(n) for Internet.



k-NN graph construction



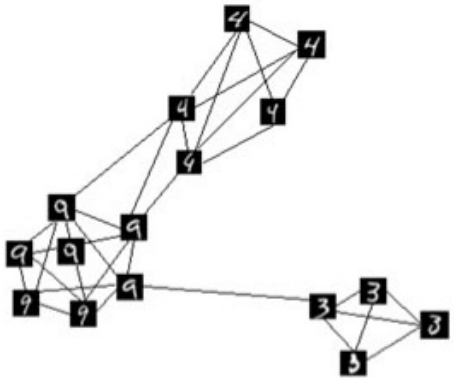
$V = \{x_1, ..., x_n\} \in \mathbb{R}^d$

$G = \{V, E, A\},\ A \in \mathbb{R}^{n \times n}$

[1] Muja, Lowe, Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, 2009

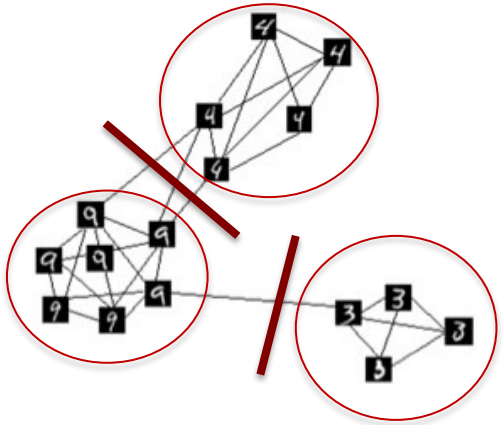
# Data clustering as graph partitioning

- Observe on a graph that close data are similar and thus form consistent clusters.
- Finding clusters within the graph can be achieved by cutting the graph at strategic locations.
- The key is to make cuts in the graph where the number of edges is minimized, clearly separating distinct clusters.



$G = (V, E, W)$

Graph partitioning



Appropriate cuts provide good clusters

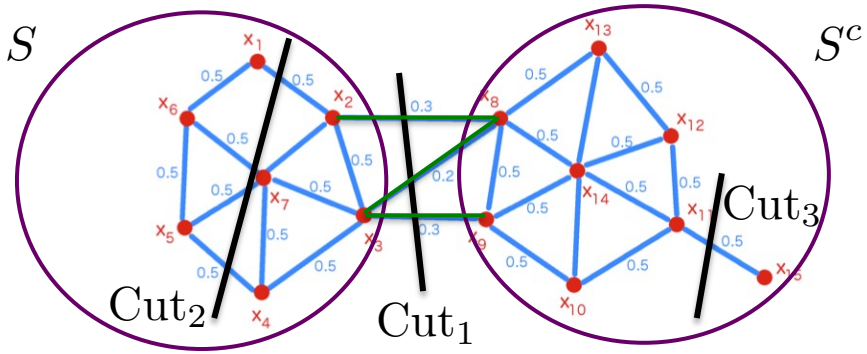
# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Cut partitioning

- Cut operator[1] : Given a graph G, a cut partitions G into two sets S and $S^c$ with value :

$$\mathrm{Cut}(S, S^c) = \sum_{i \in S, j \in S^c} A_{ij}$$




Value of $\mathrm{Cut}_1$ : $\mathrm{Cut}(S,S^c) = 0.3 + 0.2 + 0.3 = 0.8$

Value of $\mathrm{Cut}_2$ : $\mathrm{Cut}(S,S^c) = 0.5 + 0.5 + 0.5 + 0.5 = 2.0$

Value of $\mathrm{Cut}_3$ : $\mathrm{Cut}(S,S^c) = 0.5$ ⟵

- It is obvious that min cut partitioning favors small sets containing isolated points.
- A better approach is to seek clusters of similar sizes while simultaneously minimizing the cut operator :

$$\min \mathrm{Cut} \text{ and } \max \mathrm{Vol} \Leftrightarrow \min \frac{\mathrm{Cut}}{\mathrm{Vol}} \text{ a.k.a. Balanced cuts}$$

[1] Wu, Leahy, An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation, 1993

# Balanced cut partitioning

- **Cheeger Cut[1] (most popular in graph theory) :**

$$\min_{S \subset V,\ S \cup S^c = V} \frac{\text{Cut}(S, S^c)}{\min(\text{Vol}(S), \text{Vol}(S^c))}$$

- **Normalized Cut[2] (most popular in application) :**

$$\min_{S \subset V,\ S \cup S^c = V} \frac{\text{Cut}(S, S^c)}{\text{Vol}(S)} + \frac{\text{Cut}(S^c, S)}{\text{Vol}(S^c)}$$

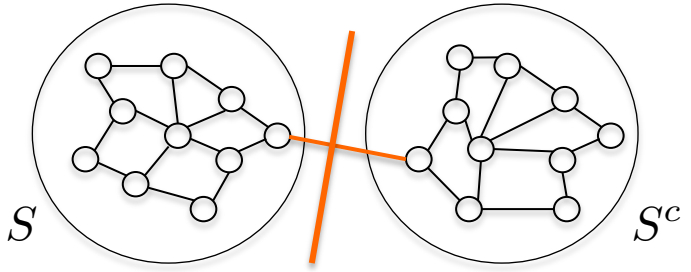
- **Normalized Association[3] (equivalent to Normalized Cuts) :**

$$\max_{S \subset V,\ S \cup S^c = V} \frac{\text{Assoc}(S, S^c)}{\text{Vol}(S)} + \frac{\text{Assoc}(S^c, S)}{\text{Vol}(S^c)}$$

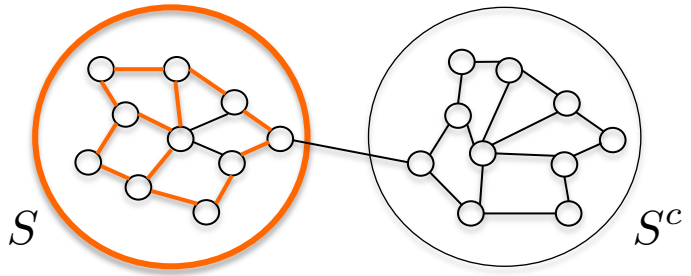
with $\text{Cut}(S, S^c) = \sum_{i \in S, j \in S^c} A_{ij}$ (num of connections between $S$ and $S^c$)

$\text{Vol}(S) = \sum_{i \in S} d_i$, with $d_i = \sum_{j \in V} A_{ij}$ (volume and degree resp.)

$\text{Assoc}(S, S^c) = \sum_{i \in S, j \in S} A_{ij}$ (num of connections inside $S$)




Partitioning by minimizing edge cuts




Partitioning by maximizing edge matching


[1] Cheeger, Pinching theorems for a certain class of Riemannian manifolds, 1969
[2] Shi, Malik, Normalized cuts and image segmentation, 2000
[3] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998

# Discrete optimization

- **Balanced cut problems are NP-hard combinatorial problems.**
- **Normalized Association for k clusters $S_q$ :**

$$\max_{\{S_q\}_{q=1}^k \text{ s.t. } \cup_q S_q = V,\ \cap_q S_q = \emptyset} \sum_{q=1}^{k} \frac{\text{Assoc}(S_q, S_q^c)}{\text{Vol}(S_q)} + \frac{\text{Assoc}(S_q^c, S_q)}{\text{Vol}(S_q^c)}$$

- **We can rewrite the discrete optimization problem with a binary indicator matrix F of the sets $S_q$ :**

$$\max_{F \in \{0,1\}^{n \times k}} \sum_{q=1}^{k} \frac{F_{\cdot,q}^T A F_{\cdot,q}}{F_{\cdot,q}^T D F_{\cdot,q}} \quad \text{s.t.} \quad \sum_{q=1}^{k} F_{i,q} = 1 \ \forall i \in V$$

$$\max_{Y \in \text{ binary}^{n \times k}} \text{tr}(Y^T B Y) \ \text{ s.t. } \ Y^T Y = \text{I}_k, \ B = D^{-1/2} A D^{-1/2}$$

$$\text{with } Y_{\cdot,q} = \frac{D^{1/2} F_{\cdot,q}}{\|D^{1/2} F_{\cdot,q}\|_2} \quad \text{(vectorial representation)}$$

$$Y_{iq} = \begin{cases} \sqrt{\frac{D_{ii}}{\text{Vol}(S_q)}} & \text{if } i \in S_q \\ 0 & \text{otherwise} \end{cases} \quad \text{(point-wise representation)}$$

# Spectral relaxation

- Directly solving discrete balanced cut problems is intractable.
- Similarly to kernel k-means, we derive an approximate solution through spectral relaxation.
- We relax the binary constraint Y in $\text{binary}^{\text{n x k}}$ to its nearest convex set, i.e. $\text{R}^{\text{n x k}}$, which renders the optimization continuous.
- Subsequently, the spectral theorem[1,2] provides the solution:
  - The k largest eigenvectors of matrix A obtained through EVD.

$$\max_{Y \in \mathbb{R}^{n \times k}} \operatorname{tr}(Y^T B Y) \quad \text{s.t.} \quad Y^T Y = \mathrm{I}_k, \; B = D^{-1/2} A D^{-1/2}$$

$$\text{with} \quad B = \Theta^{1/2} A \Theta^{1/2} \overset{\text{EVD}}{=} U \Lambda U^T \in \mathbb{R}^{n \times n}$$

$$\text{and solution} \quad Y^\star = U_{\cdot,1:k} \in \mathbb{R}^{n \times k} \quad (k \text{ largest eigenvectors})$$

[1] Helmberg, Introduction to Spectral Theory in Hilbert Space, 1969
[2] Hawkins, Cauchy and the spectral theory of matrices, 1975

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Insightful relationship

- The equivalence between kernel k-means and balanced cuts has been established[1,2].
  - **Kernel K-Means :** $\max_{Y \in \mathbb{R}^{n \times k}} \operatorname{tr}(Y^T B Y) \text{ s.t. } Y^T Y = \mathrm{I}_k,\ B = \Theta^{1/2} K \Theta^{1/2}$
  - **Balanced Cuts :** $\max_{Y \in \mathbb{R}^{n \times k}} \operatorname{tr}(Y^T B Y) \text{ s.t. } Y^T Y = \mathrm{I}_k,\ B = D^{-1/2} A D^{-1/2}$

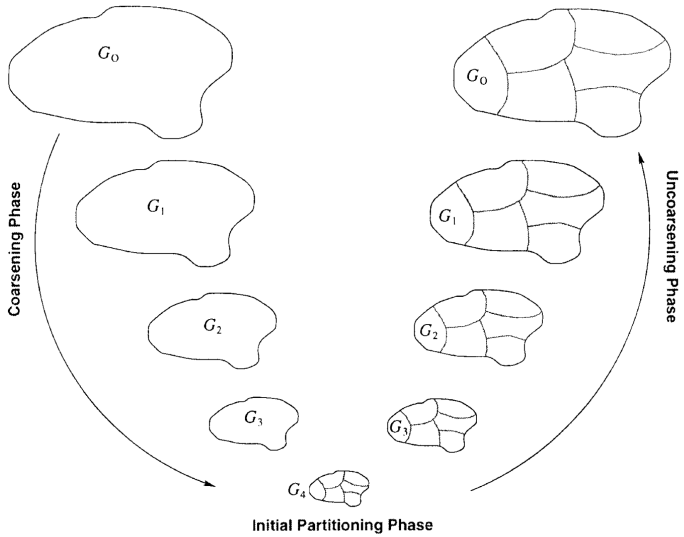
    which are equivalent for $\Theta = D^{-1}, K = A$
- This equivalence underscores the similarity of K and A : both matrices encode relationships between data points.
- The critical difference is that A is sparse, by considering only close data points, whereas K is a full matrix, assigning large values to close points and small values to distant ones.

$$K_{ij} \approx A_{ij} = \begin{cases} \exp(-\|x_i - x_j\|_2^2/\sigma^2) & \text{for } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases}$$

[1] Bach, Jordan, Learning Spectral Clustering, 2003
[2] Dhillon, Guan, Kulis, Kernel k-means: spectral clustering and normalized cuts, 2004

# Metis technique

- Leveraging the interplay between graph partitioning and data clustering, Metis[1] or Graclus[2] offer methods for partitioning graphs using kernel k-means.
- In Metis, a hierarchical graph representation is also used to suboptimal solutions.
- Notably, these approaches do not require EVD, enabling scalability to large graphs with O(E) complexity.
- As of 2023, Metis is one of the best graph partitioning techniques, striking a balance between speed and accuracy.
- An optimized multi-core CPU-version is available by Amazon in DGL[3].




[1] Karypis, Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, 1998 (w/ 7k citations as of 2023)
[2] Dhillon, Guan, Kulis, Weighted Graph Cuts without Eigenvectors: A Multilevel Approach, 2007
[3] Wang et-al, Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks, 2019

# Lab 3 : Metis

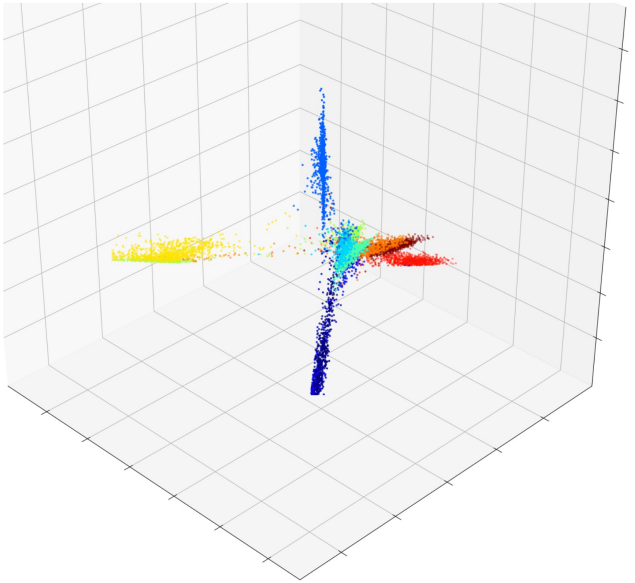
- Run code03.ipynb and test Metis on
  - Artificial balanced graph
  - Real-world USPS image graph

```
# Run Metis with DGL
#  https://docs.dgl.ai/en/0.8.x/generated/dgl.dataloading.ClusterGCNSampler.html

try: os.remove("cluster_gcn.pkl") # remove any existing partition
except: pass
num_parts = nc
G_dgl = dgl.from_scipy(W)
sampler = dgl.dataloading.ClusterGCNSampler(G_dgl, num_parts)
C_metis_dgl = torch.zeros(G_dgl.num_nodes()).long()
for idx, (idx_start, idx_end) in enumerate(zip(sampler.partition_offset[:num_parts],
    C_metis_dgl[sampler.partition_node_ids[idx_start: idx_end]] = idx
print('C_metis_dgl',C_metis_dgl)
C_metis_dgl = np.array(C_metis_dgl, dtype='int32')
acc = compute_purity(C_metis_dgl, Cgt, nc)
print('\nAccuracy Metis DGL :',acc)
```

```
Convert a graph into a bidirected graph: 0.004 seconds, peak memory: 0.000 GB
Construct multi-constraint weights: 0.000 seconds, peak memory: 0.000 GB
Metis partitioning: 0.031 seconds, peak memory: 0.000 GB
C_metis_dgl tensor([1, 0, 0,  ..., 9, 9, 9])
[23:27:38] /tmp/dgl_src/src/graph/transform/metis_partition_hetero.cc:89: Partitioned get 6607 edge cuts

Accuracy Metis DGL : 81.10346311034631
```

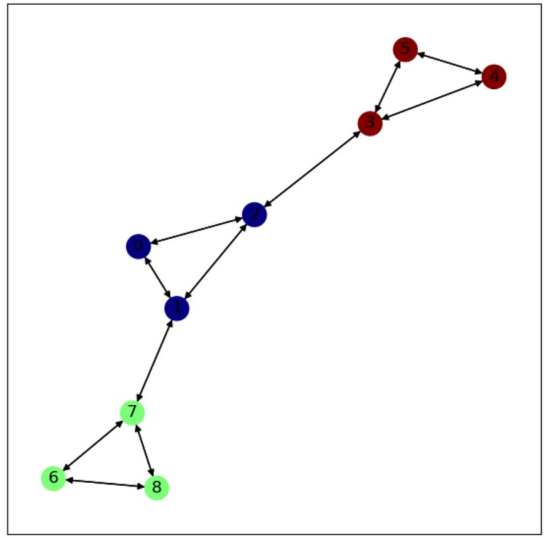
```
# Run Metis with PyG
#  https://github.com/inducer/pymetis/blob/master/pymetis/__init__.py

num_parts = nc
G_nx = nx.from_scipy_sparse_array(W)
_, part_vert = pymetis.part_graph(num_parts, adjacency=G_nx)
C_metis_pyg = np.array(part_vert,dtype='int32')
acc = compute_purity(C_metis_pyg, Cgt, nc)
print('\nAccuracy Metis PyG :',acc)
```

```
Accuracy Metis PyG : 77.16713271671327
```



```
# Run Metis with DGL
#  https://docs.dgl.ai/en/0.8.x/generated/dgl.dataloading.ClusterGCNSampler.html

try: os.remove("cluster_gcn.pkl") # remove any existing partition
except: pass
num_parts = 3
sampler = dgl.dataloading.ClusterGCNSampler(G_dgl, num_parts)
C_metis_dgl = torch.zeros(G_dgl.num_nodes()).long()
for idx, (idx_start, idx_end) in enumerate(zip(sampler.partition_offset[:num_parts],
    C_metis_dgl[sampler.partition_node_ids[idx_start: idx_end]] = idx
print('C_metis_dgl',C_metis_dgl)
G_nx = dgl.to_networkx(G_dgl)
plt.figure(figsize=[7,7])
nx.draw_networkx(G_nx, with_labels=True, node_color=C_metis_dgl, cmap='jet')
```

```
Convert a graph into a bidirected graph: 0.000 seconds, peak memory: 0.000 GB
Construct multi-constraint weights: 0.001 seconds, peak memory: 0.000 GB
Metis partitioning: 0.000 seconds, peak memory: 0.000 GB
C_metis_dgl tensor([0, 0, 0, 2, 2, 2, 1, 1, 1])
[23:27:35] /tmp/dgl_src/src/graph/transform/metis_partition_hetero.cc:89: Partition
edge cuts
```



# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Normalized cut

- **Normalized Cut[1,2] for k clusters $S_q$ :**

$$\min_{\{S_q\}_{q=1}^k \text{ s.t. } \cup_q S_q = V,\ \cap_q S_q = \emptyset} \sum_{q=1}^{k} \frac{\text{Cut}(S_q, S_q^c)}{\text{Vol}(S_q)}$$

- **We can rewrite the discrete optimization problem with a binary indicator matrix F of the sets $S_q$ :**

$$\min_{F \in \{0,1\}^{n \times k}} \sum_{q=1}^{k} \frac{F_{\cdot,q}^T L F_{\cdot,q}}{F_{\cdot,q}^T D F_{\cdot,q}}, \quad \text{with } L = D - A \quad \text{and} \quad \sum_{q=1}^{k} F_{i,q} = 1 \ \forall i \in V$$

$$\min_{Y \in \text{ binary}^{n \times k}} \text{tr}(Y^T B Y) \ \text{ s.t. } \ Y^T Y = \text{I}_k, \ B = \text{I} - D^{-1/2} A D^{-1/2}$$

$$\text{with } Y_{\cdot,q} = \frac{D^{1/2} F_{\cdot,q}}{\|D^{1/2} F_{\cdot,q}\|_2} \quad \text{(vectorial representation)}$$

$$Y_{iq} = \begin{cases} \sqrt{\frac{D_{ii}}{\text{Vol}(S_q)}} & \text{if } i \in S_q \\ 0 & \text{otherwise} \end{cases} \qquad \text{(point-wise representation)}$$

[1] Shi, Malik, Normalized cuts and image segmentation, 2000
[2] Yu, Shi, Multiclass spectral clustering, 2003

# Normalized cut

- As before, relaxing the binary constraint Y from $\text{binary}^{n \times k}$ to the nearest convex set $\mathrm{R}^{n \times k}$ makes the optimization continuous and tractable.
- This relaxation provides an approximate solution given by the spectral theorem, specifically, the k smallest eigenvectors of the graph Laplacian.
- Typically, spectral solutions do not satisfy the binary constraint, resulting in a loose relaxation.
- However, an improvment can be made to the spectral solution by enforcing the binary constraint, leading to the development of the normalized cut technique[1,2], a.k.a. as NCut.
- NCut stands out as the most popular spectral graph clustering algorithm, with 19k citations as of 2023.

[1] Shi, Malik, Normalized cuts and image segmentation, 2000
[2] Yu, Shi, Multiclass spectral clustering, 2003

# A two-step technique

- **Step 1:** **Compute the spectral solution :**

$$Y^\star = \arg\min_{Y \in \mathbb{R}^{n \times k}} \operatorname{tr}(Y^T B Y) \;\text{ s.t. }\; Y^T Y = \mathrm{I}_k,\; B = \mathrm{I} - D^{-1/2} A D^{-1/2}$$

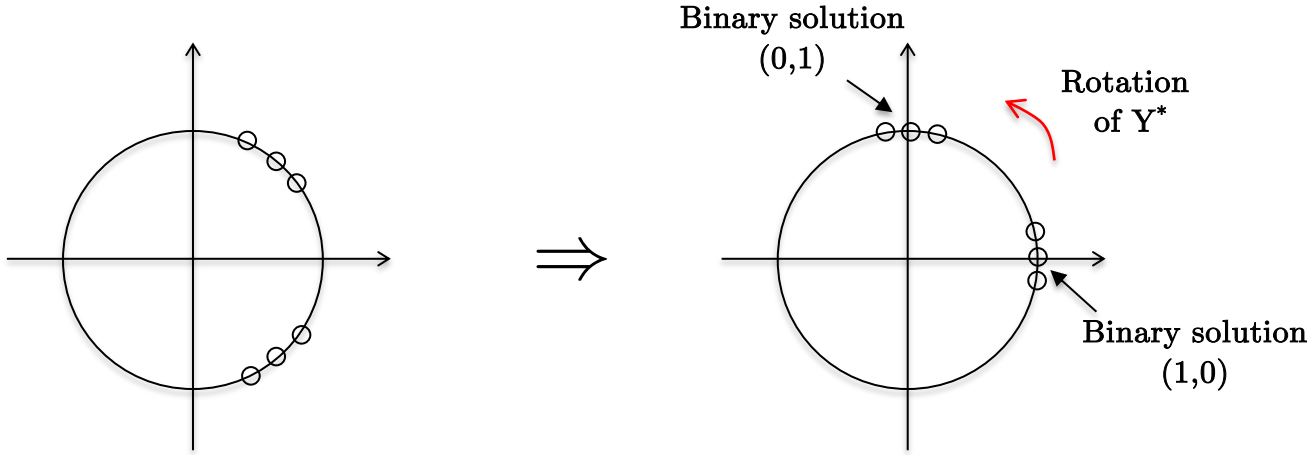
Solved by EVD.

- **Step 2:** **Spectral solutions are defined up to rotations.**

**Identifying the rotation that aligns best with the binary constraint.**

$$Z^\star = \arg\min_{Z,R} \|Z - Y^\star R\|_F^2 \;\text{ s.t. }\; R^T R = \mathrm{I}_k,\; Z \in \{0,1\}^{n \times k}$$

Solved by SVD and binarization.



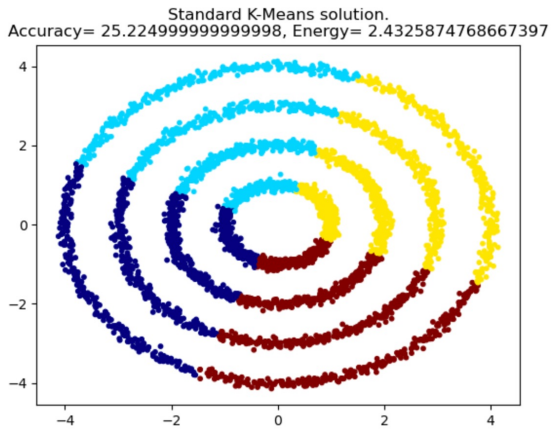

Step 1 : Y* Step 2 : Z* (rotate Y*)

# Lab 4 : NCut

- Run code04.ipynb and test Ncut on non-linear datasets with 4 classes

```
# Run standard/linear k-means with EM approach
Theta = np.ones(n) # Same weight for each data
# Compute linear Kernel for standard K-Means
Ker = construct_kernel(X, 'linear')
# Standard K-Means
C_kmeans, En_kmeans = compute_kernel_kmeans_EM(nc, Ker, Theta, 10)
# Plot
plt.figure(2)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_kmeans)
plt.title('Standard K-Means solution.\nAccuracy= ' + str(compute_purity(C_kmeans,Cgt,nc))
          ', Energy= ' + str(En_kmeans))
plt.show()
```

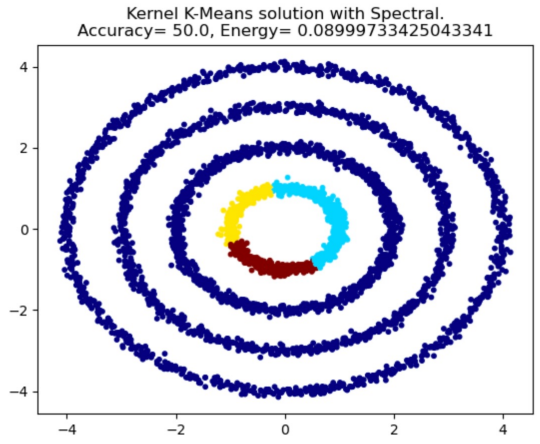
```
Construct Linear Kernel
```




Linear k-means
EM approach

```
# Run kernel/non-linear k-means with spectral approach
Ker = construct_kernel(X, 'kNN_gaussian', 100)
# Kernel K-Means with Spectral approach
C_kmeans, En_kmeans = compute_kernel_kmeans_spectral(nc, Ker, Theta, 10)
# Plot
plt.figure(3)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_kmeans, color=pyplot.jet())
plt.title('Kernel K-Means solution with Spectral.\nAccuracy= ' +
          str(compute_purity(C_kmeans,Cgt,nc)) + ', Energy= ' + str(En_kmeans))
plt.show()
```

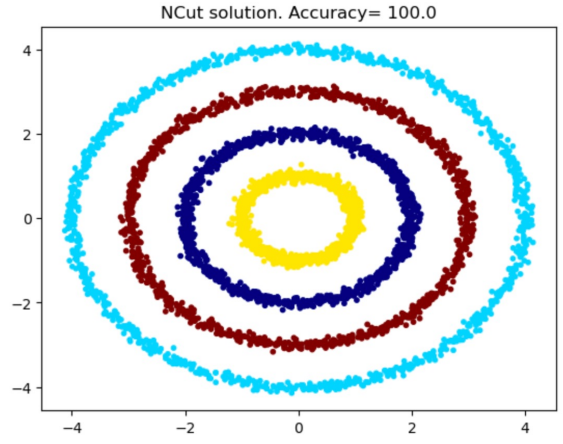
```
Construct kNN Gaussian Kernel
Construct Linear Kernel
```




Kernel k-means
EM approach

```
# Run NCut
W = construct_knn_graph(X, 50, 'euclidean_zelnik_perona')
C_ncut, acc = compute_ncut(W, Cgt, nc)

# Plot
plt.figure(4)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_ncut, color=pyplot.jet())
plt.title('NCut solution. Accuracy= ' +
          str(compute_purity(C_ncut,Cgt,nc)) )
plt.show()
```

```
k-NN graph with Zelnik-Perona technique
```



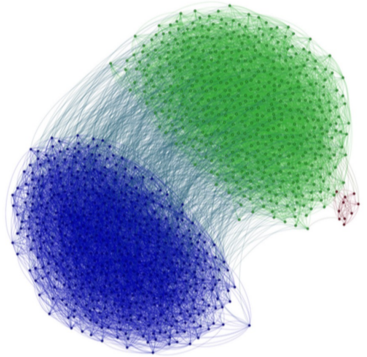

NCut
Spectral approach

# Outline

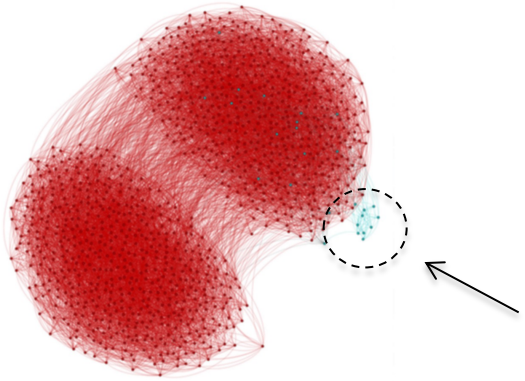
- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Product cut

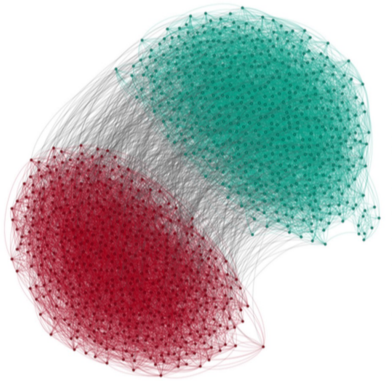
- Limitations
  - Standard cuts[1] are biased towards data outliers.
  - Balanced Cuts[2] s.a. NCut are biased towards cluster outliers.
- Product Cut[3] are specifically designed and guaranteed to maintain robustness in the presence of cluster outliers.



Two clusters with a small outlier cluster



NCut solution



PCut solution

[1] Wu, Leahy, An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation, 1993
[2] Shi, Malik, Normalized cuts and image segmentation, 2000
[3] Laurent, von Brecht, Bresson, Szlam, The Product Cut, 2016

# Product cut

- **Combinatorial optimization problem :**

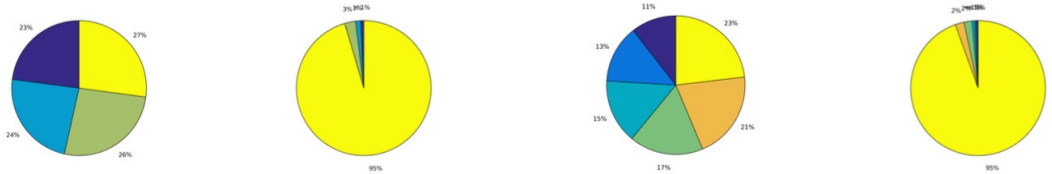
$$\min_{\{S_q\}_{q=1}^k \text{ s.t. } \cup_q S_q = V,\ \cap_q S_q = \emptyset} \frac{\prod_{q=1}^k \text{Connec}(S_q, S_q^c)}{\exp(H(\{S_q\}))}$$

$$\text{where} \quad \text{Connec}(S_q, S_q^c) = \prod_{i \in S_q} 1 + \frac{\sum_{j \in S_q^c} A_{ij}}{\sum_{j \in S_q^c} A_{ij}}$$

$$H(\{S_q\}) = -\sum_{q=1}^k p_q \log p_q, \ p_q = \frac{|S_q|}{|V|}$$

- **Real-world datasets are noisy and composed of outliers of small clusters, which bias the graph clustering algorithms to bad solutions.**

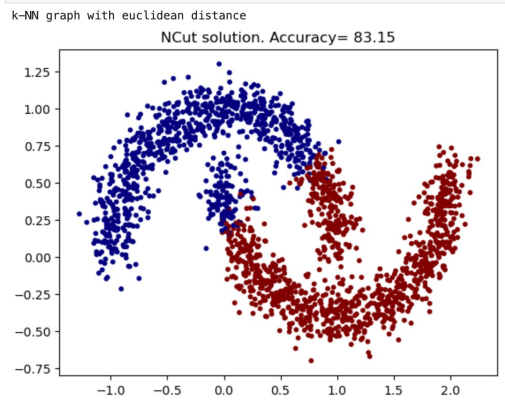
| | Partition $\mathcal{P}$ of WEBKB4 found by by the **Pcut** algo. | Partition $\mathcal{P}$ of WEBKB4 found by by the **Ncut** algo. | Partition $\mathcal{P}$ of CITESEER found by by the **Pcut** algo. | Partition $\mathcal{P}$ of CITESEER found by by the **Ncut** algo. |
|---|---|---|---|---|
| $e^{-H(\mathcal{P})}$ | .2506 | .7946 | .1722 | .7494 |
| **Pcut**$(\mathcal{P})$ | **.5335** | .8697 | **.4312** | .8309 |
| **Ncut**$(\mathcal{P})$ | .5257 | **.5004** | .5972 | **.5217** |



# Lab 5 : PCut

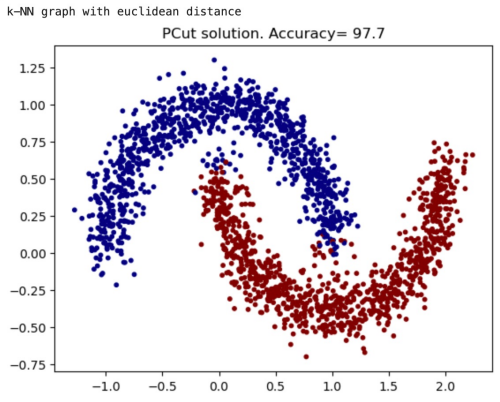
- Run code05.ipynb and test Pcut on
  - The two-moon dataset
  - USPS and MIREX datasets

```
# Run NCut
W = construct_knn_graph(X, 10, 'euclidean')
C_ncut, _ = compute_ncut(W, Cgt, nc)

# Plot
plt.figure(2)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_ncut, cmap='jet')
plt.title('NCut solution. Accuracy= ' +
          str(compute_purity(C_ncut, Cgt, nc))[:6] )
plt.show()
```




NCut solution

```
# Run PCut
W = construct_knn_graph(X, 10, 'euclidean')
C_pcut, _ = compute_pcut(W, Cgt, nc, 2, 200)

# Plot
plt.figure(3)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=C_pcut, cmap='jet')
plt.title('PCut solution. Accuracy= ' +
          str(compute_purity(C_pcut, Cgt, nc))[:6] )
plt.show()
```




PCut solution

## USPS image graph

```
# Load USPS dataset
mat = scipy.io.loadmat('datasets/USPS.mat')
W = mat['W'] # 'scipy.sparse._csc.csc_matrix'
n = W.shape[0]
Cgt = mat['Cgt']-1; Cgt=Cgt.squeeze()
nc = len(np.unique(Cgt))
print(n,nc)
```

```
9298 10
```

```
Cncut, acc = compute_ncut(W,Cgt,nc)
print('Ncut accuracy =',acc)
```

```
Ncut accuracy = 73.52118735211873
```

```
Cpcut, acc = compute_pcut(W,Cgt,nc,5,10)
print('Pcut accuracy =',acc)
```

```
Pcut accuracy = 81.24327812432782
```

## MIREX music graph

```
# Load USPS dataset
mat = scipy.io.loadmat('datasets/MIREX.mat')
W = mat['W'] # 'scipy.sparse._csc.csc_matrix'
n = W.shape[0]
Cgt = mat['Cgt']-1; Cgt=Cgt.squeeze()
nc = len(np.unique(Cgt))
print(n,nc)
```

```
3090 10
```

```
Cncut, acc = compute_ncut(W,Cgt,nc)
print('Ncut accuracy =',acc)
```

```
Ncut accuracy = 39.25566343042071
```

```
Cpcut, acc = compute_pcut(W,Cgt,nc,0.5,400)
print('Pcut accuracy =',acc)
```

```
Pcut accuracy = 45.6957928802589
```

# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- Conclusion

# Number of clusters

- Previous techniques assume that the number k of clusters is predefined.
- In scenarios where k is unknown, two approaches are commonly used :
  - Domain expertise
    - Define a quality measure of clustering.
    - Apply previously introduced techniques with various k values.
    - Pick the k value with the highest quality.
  - Simultaneous optimization of the clusters and their number
    - Treat k as a variable of the clustering problem
    - Use Louvain algorithm[1] to dynamically determine the optimal number of clusters during the clustering process.

[1] Blondel, Guillaume, Lambiotte, Lefebvre, Fast unfolding of communities in large networks, 2008

# Louvain algorithm

- **Louvain technique**[1]
  - **Popular technique in social science w/ 20k citations as of 2023.**
  - **It is basically a greedy algorithm that optimizes the modularity objective :**

$$\max_{k, C:V\to\{1,2\ldots,k\}} \sum_{ij} \left( A_{ij} - \gamma \frac{d_i d_j}{\sum_{i'j'} A_{i'j'}} \right) \delta(C_i, C_j)$$

$$\text{with} \quad \delta(C_i, C_j) = \begin{cases} 1 & \text{if } C_i = C_j \ (i \text{ and } j \text{ belong to the same cluster}) \\ 0 & \text{otherwise} \end{cases}$$

[1] Blondel, Guillaume, Lambiotte, Lefebvre, Fast unfolding of communities in large networks, 2008

# Number of clusters

- **Observe that**

$$\max_{k, C:V\to\{1,2\ldots,k\}} \sum_{ij} \Big(A_{ij} - \gamma \frac{d_i d_j}{\sum_{i'j'} A_{i'j'}}\Big)\delta(C_i, C_j)$$

is equivalent to

$$\min_{\{S_q\}_{q=1}^k \text{ s.t. } \cup_q S_q = V,\ \cap_q S_q = \emptyset} \sum_{q=1}^{k} \mathrm{Cut}(S_q, S_q^c) - \gamma \mathrm{Vol}(S_q)\mathrm{Vol}(S_q^c)$$

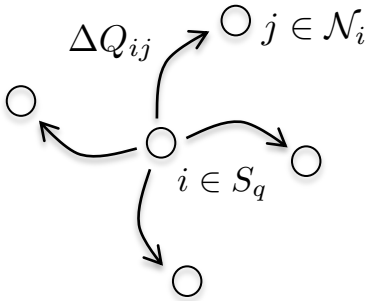
which is a relaxation of the balanced cut we call LCut:

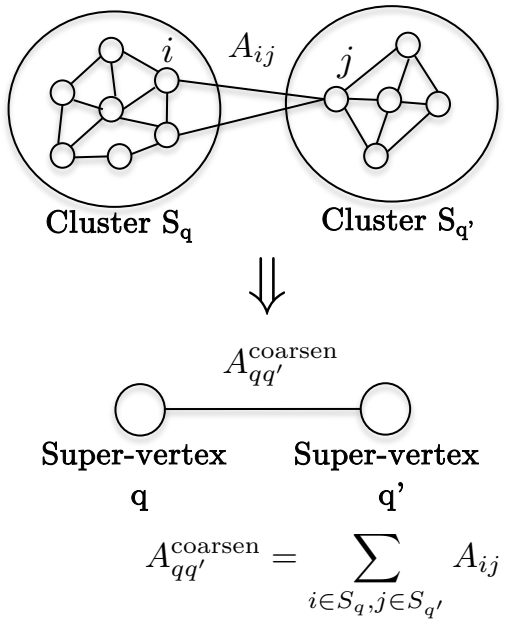
$$\min_{\{S_q\}_{q=1}^k \text{ s.t. } \cup_q S_q = V,\ \cap_q S_q = \emptyset} \sum_{q=1}^{k} \frac{\mathrm{Cut}(S_q, S_q^c)}{\mathrm{Vol}(S_q)\mathrm{Vol}(S_q^c)}, \text{ where } \gamma = \min_{\{S_q\}} \mathrm{LCut}(\{S_q\})$$

- **The parameter γ provides a loose, i.e. not exact, control over the number of clusters.**
- **When λ represents the minimum value of LCut, then modularity is equivalent to LCut.**

# Greedy algorithm

- Step 1 : Energy minimization step
  - Find communities by locally maximizing the modularity.
  - Each node is first assigned to its own community. Iteratively, each node i is reassigned to the community of its neighbor that maximizes modularity. The process is repeated until no changes occur.
- Step 2 : Graph coarsening step
  - Create a new graph by merging communities into super-vertices.
  - Construct a new adjacency matrix based on the communities identified in Step 1.
- Properties
  - Fast and parallelizable algorithm, e.g. used for Twitter community detection in 2009 w/ 2.4M nodes, 38M edges.
  - No theoretical guarantee to find the global modularity solution.




$$A^{\text{coarsen}}_{qq'} = \sum_{i \in S_q, j \in S_{q'}} A_{ij}$$

# Lab 6 : Louvain algorithm

- Run code06.ipynb and test Louvain algorithm on
  - The two-moon dataset
  - USPS dataset
- Observe that Louvain tends to over-cluster the datasets.

```
# Run Louvain
Wnx = nx.from_numpy_array(W.toarray())
partition = community.best_partition(Wnx)
nc_louvain = len(np.unique( [partition[nodes] for nodes in partition.keys()] ))
n = len(Wnx.nodes())
print('nb_data:', n , ', nb_clusters=', nc_louvain)

# Extract clusters
Clouv = np.zeros([n])
clusters = []
k = 0
for com in set(partition.values()):
    list_nodes = [nodes for nodes in partition.keys() if partition[nodes] == com]
    Clouv[list_nodes] = k
    k += 1
    clusters.append(list_nodes)

# Accuracy
acc = compute_purity(Clouv,Cgt,nc_louvain)
print('Louvain solution ',str(acc)[:5],' with nb_clusters=',nc_louvain)
```

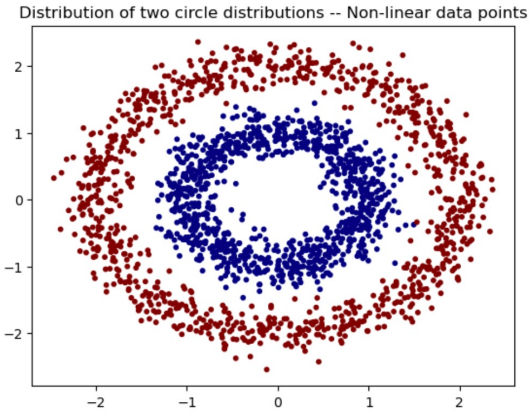
```
nb_data: 9298 , nb_clusters= 14
Louvain solution  95.71  with nb_clusters= 14
```

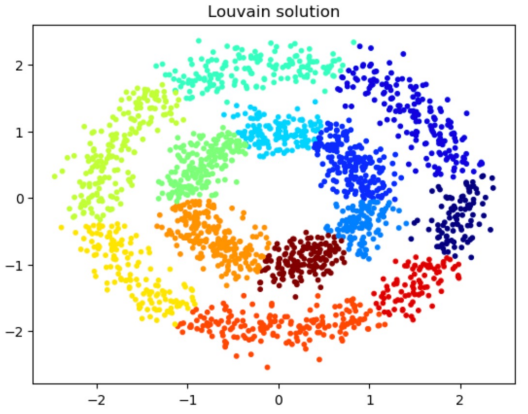
```
# Run NCut with the number of clusters found by Louvain
Cncut, acc = compute_ncut(W,Cgt,nc_louvain)
print('NCut solution:',str(acc)[:5],' with nb_clusters=',nc_louvain)
```

```
NCut solution: 86.68  with nb_clusters= 14
```

```
# Run Louvain algorithm
W = construct_knn_graph(X, 50, 'euclidean_zelnik_perona')
Wnx = nx.from_numpy_array(W)
partition = community.best_partition(Wnx)
nc_louvain = len(np.unique( [partition[nodes] for nodes in partition.keys()] ))
n = len(Wnx.nodes())
print('nb_data:', n , ', nb_clusters=', nc_louvain)

# Extract clusters
Clouv = np.zeros([n])
clusters = []
k = 0
for com in set(partition.values()):
    list_nodes = [nodes for nodes in partition.keys() if partition[nodes] == com]
    Clouv[list_nodes] = k
    k += 1
    clusters.append(list_nodes)

# Accuracy
acc = compute_purity(Clouv,Cgt,nc_louvain)
print('accuracy_louvain=',acc,' with nb_clusters=',nc_louvain)

plt.figure(2)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n), c=Clouv, cmap='jet')
plt.title('Louvain')
plt.show()
```

```
k-NN graph with Zelnik-Perona technique
nb_data: 2000 , nb_clusters= 14
accuracy_louvain= 99.8  with nb_clusters= 14
```

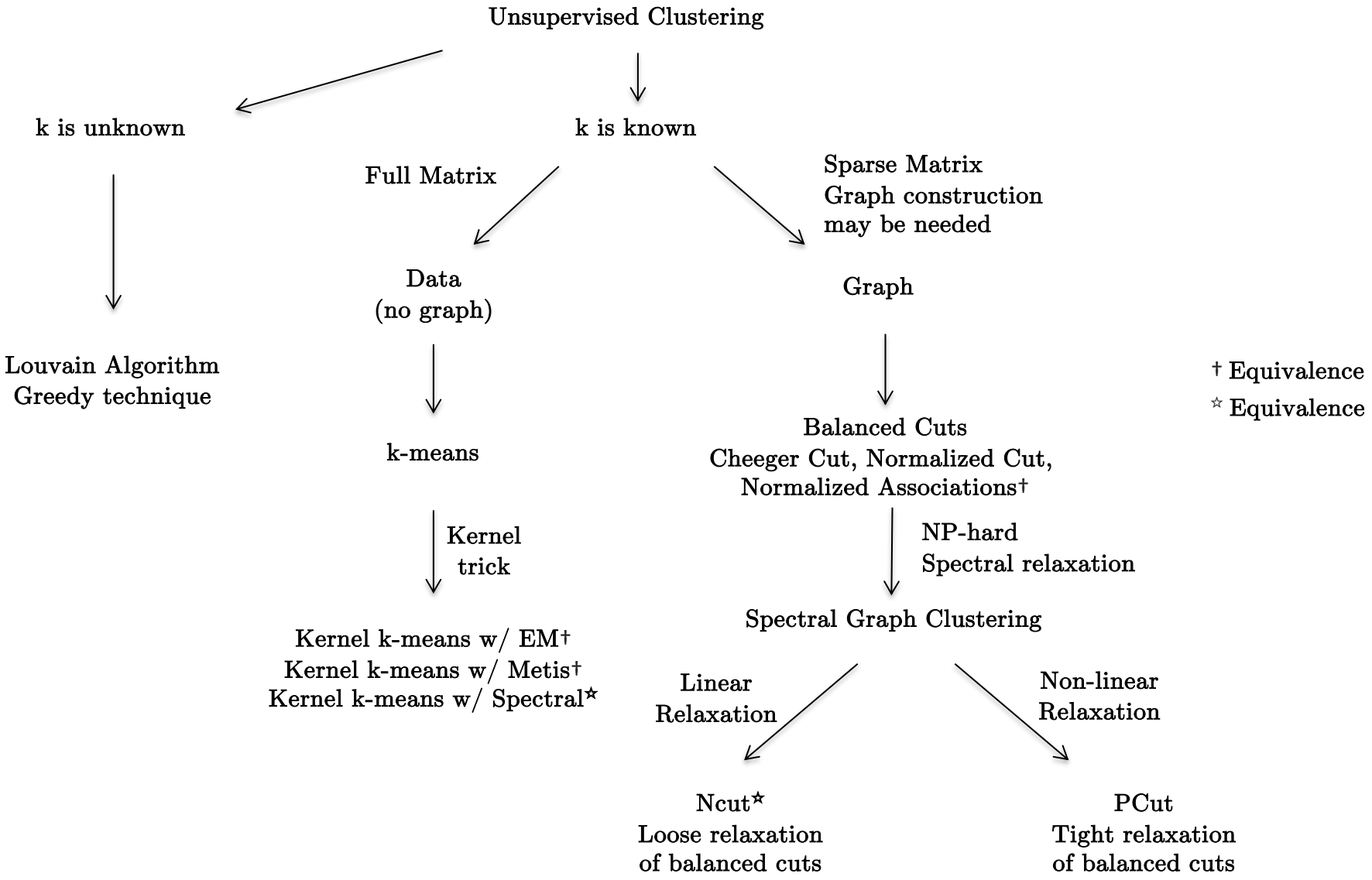


# Outline

- Data clustering
  - Standard k-means
  - Kernel k-means
    - EM approach
    - Spectral approach
- Graph clustering
  - Balanced cuts
  - Metis
  - Normalized cut
  - Product cut
  - Louvain algorithm
- **Conclusion**

# Overview of data and graph clustering



# Conclusion

- Clustering, both for data and graphs, is a cornerstone topic that beautifully connnects combinatorial and discrete optimization, continuous optimization, graph theory, and spectral solutions.
- Linear and kernel k-means is the most basic unsupervised data clustering algorithms. They are solvable with EM/greedy optimization, as well as spectral techniques.
- Unsupervised graph clustering algorithms, like NCut and Metis, focus on balanced cuts. Like k-means, they can be tackled through fast greedy algorithms or spectral optimization.
- We show the equivalence between data clustering and graph partitioning tasks, both aiming to identify communities of similar data points or nodes in graphs.
- Data clustering complexity is $O(n^2dk)$ for greedy methods, $O(n^2d+n^2k)$ for spectral techniques.
- Graph clustering has complexity $O(n^2d)$ for graph construction, $O(E)$ for Metis and $O(E^{3/2}k)$ for spectral approaches.

Questions?