

CS5284 : Graph Machine Learning

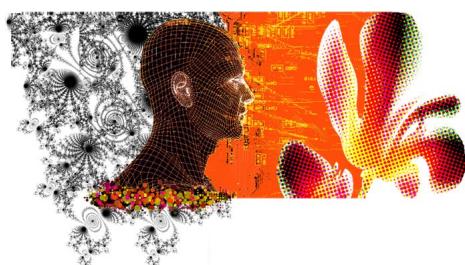
Lecture 3 : Graph Clustering

Semester 1 2024/25

Xavier Bresson

<https://x.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
 - Graph SVM
 - Recommendation on graphs
 - Graph-based visualization
- 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
 - Graph generation & molecular science
 - GNNs for combinatorial optimization
 - Integrating GNNs and LLMs
 - Benchmarking GNNs
 - GNNs for recommendation & knowledge graphs

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

K-Means loss

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

Annotations:

- q clusters: k circles around the mean m_q .
- data points assigned to cluster q : points x_i inside the circle S_q .
- q^{th} mean: center of the circle m_q .
- Distance between data x_i and its mean m_q : vector from m_q to x_i .

\Rightarrow

EM algorithm

- Expectation-maximization (EM) technique^[1,2]
- Initialization
 - Randomly select initial means $\{m_1, \dots, m_k\} \in \mathbb{R}^d$ 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 : $l = 0, 1, 2, \dots$
 - Cluster update (expectation step)

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

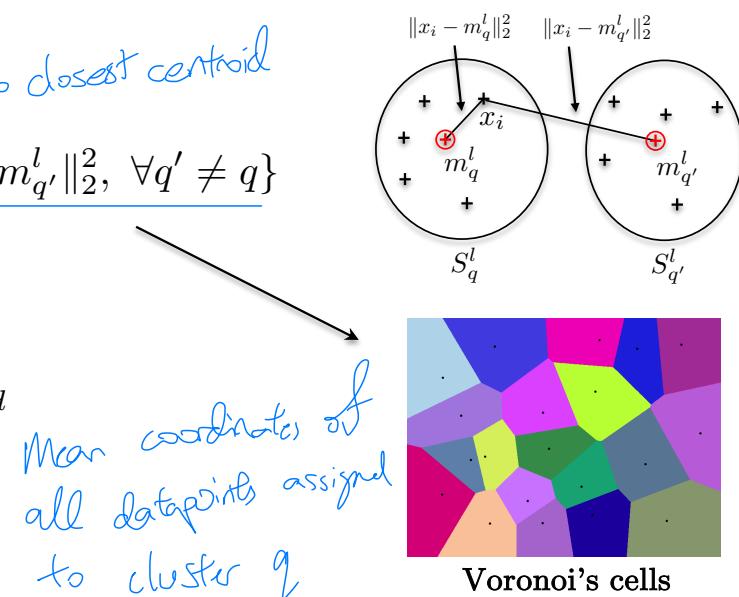
Iteration
Assign datapoint x_i to closest centroid
 - Mean update (maximization step)

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

[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



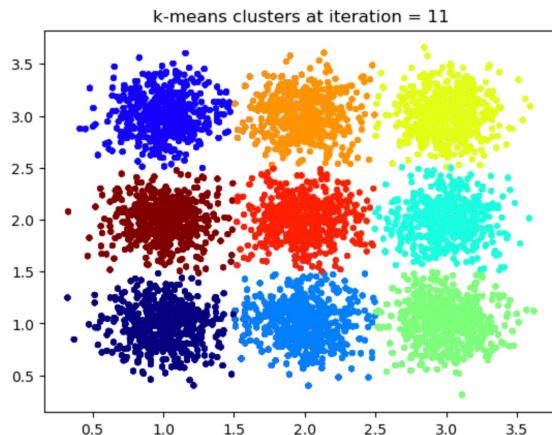
Lab 1 : Standard k-means

- Run code01.ipynb and analyze k-means result on

- Linearly separable data points *Good*

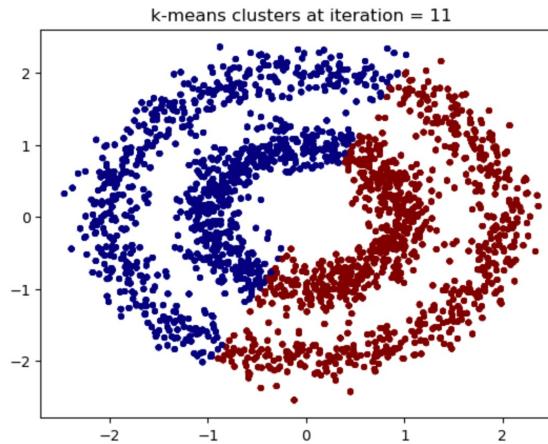
- Non-linear data points *No good* → *DBSCAN*

```
[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)
```



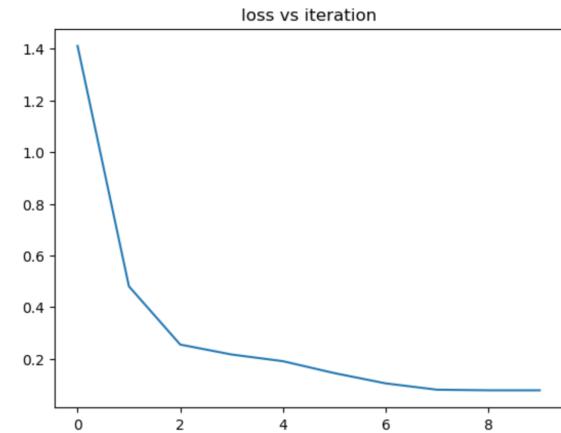
Linearly separable data points

```
[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)
```



Non-linear data points

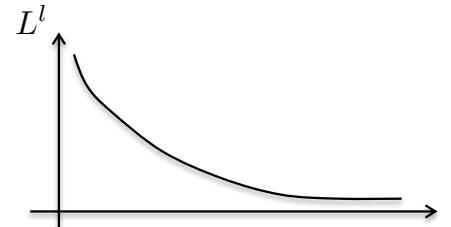
```
[7]: # Visualize loss vs iteration
plt.figure(3)
plt.plot(En_iters)
plt.title('loss vs iteration')
plt.show()
```



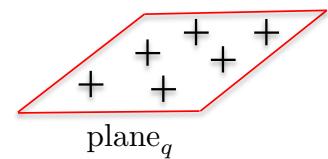
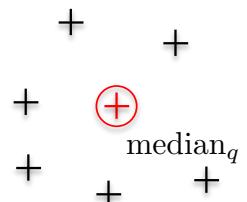
Algorithm properties

- 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. *(?) To do*
 - 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].

monotonically (guaranteed) decrease in loss



Q
To do



[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.

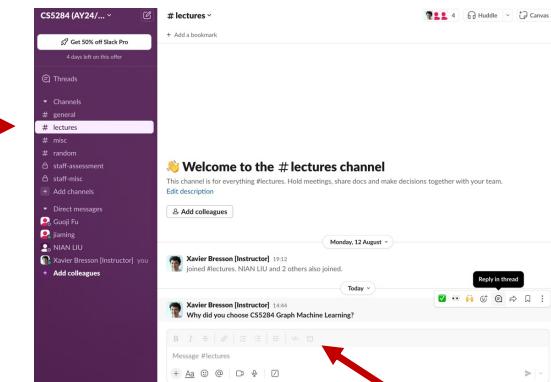
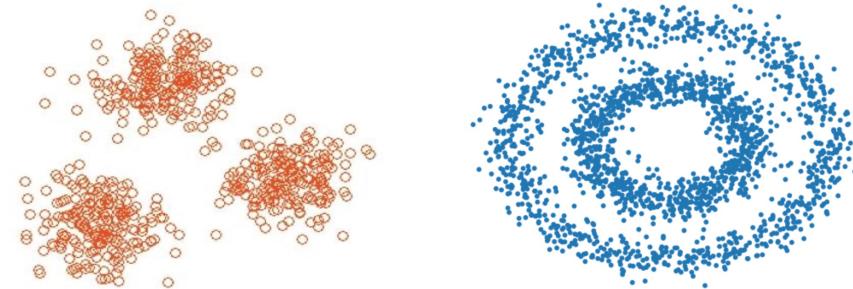
why \ominus only if we want the global minimizer \Rightarrow we need to brute force it.

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

In-lecture question

- Does k-means effectively cluster these two datasets? What assumptions does k-means make about data distribution to be successful? Justify.
- In Slack #lectures
 - Identify the question and Reply in thread with a short response

Assumptions: Data belonging to different clusters are linearly separable



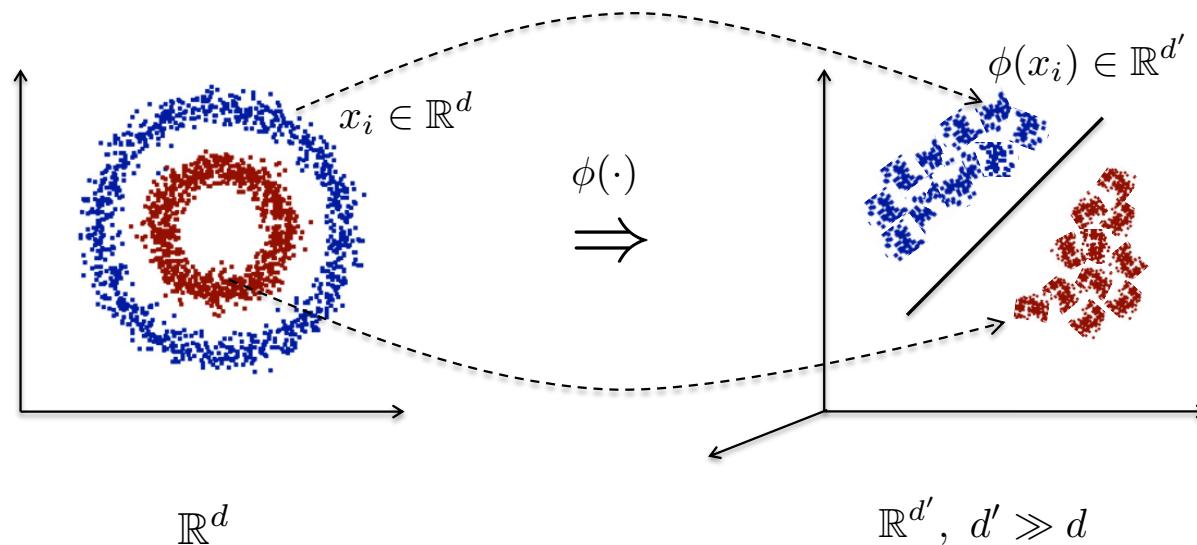
Outline

- 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 $\phi(\cdot)$:

Kernel



[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

Just replace x with $\phi(x)$

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

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

Weight contribution
for data x_i

$x_i \rightarrow \phi(x_i)$

- 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

Optimizing which combination of "cluster inclusion" (ie which node should go to which cluster) is best
 \Rightarrow Discrete optimization
 Combinatorial optimization is discrete as it optimizes the inclusion (0 or 1) of each node in each cluster. This relaxation constraint on the integer inclusion constraint makes the problem continuous

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

Cluster 1 Cluster 2 Cluster 3
 ↓ ↓ ↓
 k

datapoint 1 datapoint 2 datapoint n
 ← ← ←
 n

$F_{:,q} = \text{Indicator function of the set } S_q$

Non-linear k-means loss

Note: Each m_q is now $d' > d$ dimensions long

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

$$\begin{aligned}
 L_{\text{nl-k-means}}(\{m_q\}_{q=1}^k, \{S_q\}_{q=1}^k) &= \frac{1}{kn} \sum_{q=1}^k \sum_{i \in S_q} \theta_i \|\phi(x_i) - m_q\|_2^2 \\
 &= \frac{1}{kn} \sum_{q=1}^k \sum_{i \in S_q} \theta_i D_{iq}, \quad \text{with } D_{iq} = \|\phi(x_i) - m_q\|_2^2 \\
 &= \frac{1}{kn} \text{tr}(F^T \Theta D) = L_{\text{nl-k-means}}(F) \\
 &\text{with } F \in \{0, 1\}^{n \times k} \text{ Binary indicator matrix of shape } (n, k) \\
 &\Theta = \text{diag}(\theta_1, \dots, \theta_n) \in \mathbb{R}^{n \times n} \\
 &D \in \mathbb{R}^{n \times k} \text{ Distance matrix of each datapoint to} \\
 &\text{each cluster's centroid}
 \end{aligned}$$

Handwritten notes:

- $\text{Set of datapoints that belong to centroid } q$
- $\text{weight of datapoint } i$
- $(\Theta) = \text{diag}(1, \dots, 1) \text{ if all datapoints have the same weight}$
- $F^T \Theta D$: (nk) weighted distances
- $F^T \Theta D$: (nk) weighted distances contain the diagonals (i, i) if $F_{i,j} = 1$
- $F^T \Theta D$: (nk) weighted distances contain $\sum_j \theta_j D_{ij}$ if $F_{i,j} = 1$
- $F^T \Theta D$ should be $0 \in \mathbb{R}^{1 \times n}$ not wrong because D is matrix F . Non-diagonal elements to zeros should be 0 due to identity with F .
- other non-diagonal elements in $F^T \Theta D$ should be 0 if $i \neq j$
- Sum of elements on a square matrix's diagonal = trace

$$(\Theta)D = \begin{pmatrix} \theta_1 & 0 & \dots & 0 \\ 0 & \theta_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \theta_n \end{pmatrix} \begin{pmatrix} D_{11} & \dots & D_{1n} \\ D_{21} & \dots & D_{2n} \\ \vdots & \ddots & \vdots \\ D_{n1} & \dots & D_{nn} \end{pmatrix} = \begin{pmatrix} \theta_1 D_{11} & \theta_1 D_{12} & \dots & \theta_1 D_{1n} \\ \theta_2 D_{21} & \theta_2 D_{22} & \dots & \theta_2 D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_n D_{n1} & \theta_n D_{n2} & \dots & \theta_n D_{nn} \end{pmatrix} \quad (\Theta)D_{ij} = \theta_i D_{ij}$$

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

$$\begin{aligned}
 \therefore \text{tr}(F^T \Theta D) &= \sum_i \sum_j \theta_i D_{ij} \\
 \text{where } i &\in S_j
 \end{aligned}$$

$F^T (\Theta)D = k \text{ clusters}$

n datapoints

$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & \dots \\ 0 & 1 & 0 & 0 & 0 & \dots \\ 1 & 0 & 1 & 0 & 0 & \dots \end{pmatrix} \begin{pmatrix} 0, D_{11} & 0, D_{12} & \dots & 0, D_{1n} \\ 0, D_{21} & 0, D_{22} & \dots & 0, D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0, D_{n1} & 0, D_{n2} & \dots & 0, D_{nn} \end{pmatrix} = \begin{pmatrix} 0, D_{11} & 0 & \dots & 0 \\ 0 & 0, D_{21} + 0, D_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0, D_{nn} \end{pmatrix}$

17 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

Distance metric

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

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

map datapoint
 control q's
 coordinates

with

$$K_{ii} = \phi(x_i)^T \phi(x_i) \quad \leftarrow \text{can be pre-computed}$$

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

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

depends on m_q , which is
 optional mid-algorithm
 ↳ cannot be pre-computed

and

$$\boxed{\frac{\partial L_{\text{nl-k-means}}}{\partial m_q} = 0 \quad \Rightarrow \quad m_q = \frac{\sum_{i \in S_q} \theta_i \phi(x_i)}{\sum_{i \in S_q} \theta_i} \in \mathbb{R}^{d'} \quad \begin{matrix} \text{Formula for optimizing centroid } m_q \\ \text{Proof next page} \end{matrix}}$$

$$\text{k-means loss, } L = \frac{1}{Kn} \sum_{q=1}^K \sum_{i=1}^n F_{iq} \theta_i \|\phi(x_i) - m_q\|_2^2$$

$$= \frac{1}{Kn} \sum_{q=1}^K \sum_{i=1}^n F_{iq} \theta_i [\phi(x_i)^T \phi(x_i) - 2m_q \phi(x_i) + m_q^2]$$

$$\frac{\partial L}{\partial m_q} = \frac{1}{Kn} \sum_{q=1}^K \sum_{i=1}^n F_{iq} \theta_i \cdot \frac{2}{\partial m_q} [\phi(x_i)^T \phi(x_i) - 2m_q \phi(x_i) + m_q^2]$$

$$= \frac{1}{Kn} \sum_{q=1}^K \sum_{i=1}^n F_{iq} \theta_i \cdot [2m_q - 2\phi(x_i)] \text{ set to } = 0 \text{ to minimize } L$$

To minimize L :

~~$$\frac{1}{Kn} \sum_{q=1}^K \sum_{i=1}^n F_{iq} \theta_i \cdot [2m_q - 2\phi(x_i)] = 0$$~~

$$\sum_{q=1}^K \sum_{i=1}^n [F_{iq} \theta_i m_q - F_{iq} \theta_i \phi(x_i)] = 0$$

$$\text{For a cluster } q, \text{ Loss}_q = \sum_{i \in S_q} \theta_i m_q - \sum_{i \in S_q} \theta_i \phi(x_i) = 0$$

$$\sum_{i \in S_q} \theta_i m_q = \sum_{i \in S_q} \theta_i \phi(x_i)$$

$$m_q = \frac{\sum_{i \in S_q} \theta_i \phi(x_i)}{\sum_{i \in S_q} \theta_i}$$

This is actually pretty intuitive.
Just the weighted centre of all
datapoints in S_q
set m_q to this
to minimize L

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 : $K_{ii} = (\phi(x)\phi(x)^T)_{ii}$, $\phi(x) \in \mathbb{R}^{n \times d'}$ $K = \phi(x)\phi(x)^T \in \mathbb{R}^{n \times n}$
FYI : previous slide used $\phi(z_i) \in \mathbb{R}^{d'}$
- Second term :

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

Setting m_q to this to minimize Loss

$$\Rightarrow A = \phi(x) M^T \in \mathbb{R}^{n \times k} = \phi(x) \phi(x)^T \Theta F Z = K \Theta F Z \in \mathbb{R}^{n \times k}, \quad \Theta = \text{diag}(\theta_1, \dots, \theta_n) \in \mathbb{R}^{n \times n}$$

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

$1_n^T \Theta F$ is a \mathbb{R}^k vector with the q^{th} entry being the sum of the weights of all datapoints in cluster q . $\sum_i \Theta_{ij}$. Z serves as a normalizer to make the sum of all normalized weights sum to 1.

- Third term : $B_{qq} = (MM^T)_{qq} \Rightarrow B = Z F^T \Theta \phi(x) (Z F^T \Theta \phi(x))^T$
- Finally, matrix-based distance metric :

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

EM algorithm

~~Algorithm~~ (Just need to calculate $D \rightarrow F \rightarrow D \rightarrow F \rightarrow \dots$)
~~(or S)~~ ~~(or S)~~

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

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

\Updownarrow

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

x_i is closest to centroid q

or φ(x_i)

or φ(x_i)

All 3 mean the same thing

- 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 ϕ .
 - Observe that computing the distance D uses the quadratic matrix $\phi\phi^T$, rather than ϕ individually, thus the precise expression of ϕ becomes irrelevant. *matrix* *only this is needed, not ϕ* \checkmark
 - We define the kernel operator/matrix $K = \phi\phi^T$ with standard definitions as follows :

Common
Kernel Functions

$K(x_i, x_j) = x_i^T x_j$	Time consuming	(linear kernel for linear k-means)
$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \exp(-\ x_i - x_j\ _2^2/\sigma^2)$	(Gaussian kernel)	
$K(x_i, x_j) = (ax_i^T x_j + b)^c$	Efficient kernel computation	(Polynomial kernel)

[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

Slow:

$\forall x_i \rightarrow \text{calculate } \phi(x_i) \rightarrow \text{Construct matrix } \phi(x) \in \mathbb{R}^{n \times d'}$



To get matrix K, perform matmul $\phi(x)^T \phi(x)$

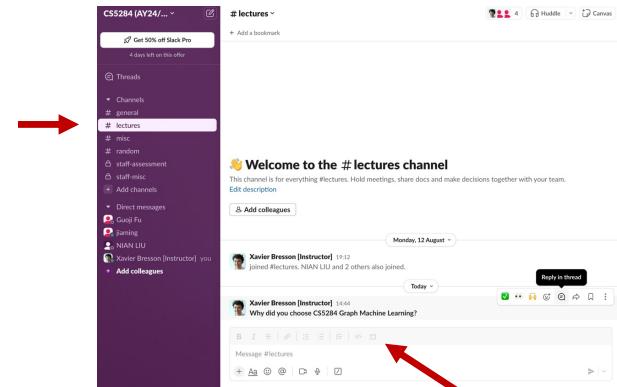
Faster:

To get matrix K, $\forall i, j \in \mathbb{Z}_{[1, n]}$ set $K_{ij} \leftarrow K(x_i, x_j)$ e.g. $\exp\left(-\frac{\|x_i - x_j\|_2^2}{\sigma^2}\right)$

calculation in low-dim.
∴ faster

In-lecture question

- Is kernel k-means guaranteed to converge with all kernels?
What is the computational complexity of this algorithm?
- In Slack #lectures
 - Identify the question and Reply in thread with a short response



Algorithm properties

- Advantages
 - 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
 - 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

Fixes the number of clusters k

a priori

Dimension of each mean m is still $\mathbb{R}^{d'}$, unchanged.

Set of datapoints assigned to cluster q

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 : $m_q \Rightarrow M \in \mathbb{R}^{k \times d'}$
 - For spectral means, we will consider the alternative representation : $m_i \Rightarrow M \in \mathbb{R}^{n \times d'}$

$$\begin{aligned}
 L_{\text{nl-k-means}}(\{m_q\}_{q=1}^k, \{S_q\}_{q=1}^k) &= \frac{1}{kn} \sum_{q=1}^k \sum_{i \in S_q} \theta_i \|\phi(x_i) - m_q\|_2^2 \\
 &= \frac{1}{kn} \sum_{q=1}^k \sum_{i \in S_q} \|\theta_i^{1/2} \phi(x_i) - \theta_i^{1/2} m_q\|_2^2, \quad \phi(x_i), m_q \in \mathbb{R}^{d'} \\
 &= \frac{1}{kn} \sum_{i \in V} \|\theta_i^{1/2} \phi(x_i) - \theta_i^{1/2} m_i\|_2^2, \quad m_i \in \mathbb{R}^{d'}, \quad \text{Just combining } \sum_{q=1}^k \sum_{i \in S_q} \text{ into } \sum_{i \in V} \\
 &= \frac{1}{kn} \sum_{i \in V} \|\theta_i^{1/2} \phi(x_i) - \theta_i^{1/2} (M)_{i,\cdot}\|_2^2, \quad M \in \mathbb{R}^{n \times d'}, \quad \text{Introducing matrix } M \\
 &= \frac{1}{kn} \|\theta^{1/2} \phi(x) - \theta^{1/2} M\|_F^2, \quad M \in \mathbb{R}^{n \times d'}, \quad \text{Def' & Frobenius norm} \\
 \text{with } M &= F Z F^T \Theta \phi(x) \in \mathbb{R}^{n \times d'} \quad M \text{ is no longer } \in \mathbb{R}^{k \times d'} \\
 &\quad \text{Previously, } M = Z F^T \phi(x)
 \end{aligned}$$

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

$$\begin{aligned}
 Y^T Y &= (\mathbb{H}^k F Z^k)^T (\mathbb{H}^k F Z^k) \\
 &= Z^k F^T (\mathbb{H}) F Z^k \quad (\because Z \text{ & } F \text{ are diagonal matrices}) \\
 &= I_k
 \end{aligned}$$

each datapoint can only be assigned to 1 cluster

New indicator matrix of clusters

Y is new indicator matrix $\therefore Y^T Y = I$

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

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

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

which implies

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

and

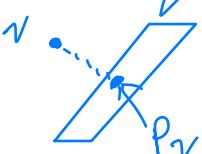
$$\begin{aligned}
 M &= F Z F^T \Theta \phi(x) = \Theta^{-1/2} Y Z^{-1/2} Z Z^{-1/2} Y^T \Theta^{-1/2} \Theta \phi(x) \\
 &= \Theta^{-1/2} Y Y^T \Theta^{1/2} \phi(x) \in \mathbb{R}^{n \times d'} \\
 &= (\mathbb{H}^k \mathbb{H}^k)^T F Z F^T \mathbb{H}^k \mathbb{H}^k \phi(x) =
 \end{aligned}$$

$$\text{Projector matrix } P = Y Y^T = (\mathbb{H}^k F Z^k)(\mathbb{H}^k F Z^k)^T = (\mathbb{H}^k F Z F^T \mathbb{H}^k)^T = A(A^T A)^{-1} A^T \text{ where } A = (\mathbb{H}^k F)$$

Xavier Bresson

$\therefore P = Y Y^T$ projects vectors onto $\text{col}(A)$ when we do Pv if $v \in \text{col}(A)$, $\text{span of col}(A)$

if $v \in \text{col}(A)$, then $Pv = v$



Non-linear k-means loss

no longer F

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

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

Relaxation

$\propto \mathcal{L}_{\text{nl-k-means}}(\mathbf{Y})$ from prev slide

- Let us consider the combinatorial optimization problem :

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

- 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. $\text{binary}^{n \times k}$, to a convex one, i.e. $\mathbb{R}^{n \times k}$, the optimization becomes continuous :

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

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

[1] Helberg, 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 :

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

with

$$\begin{aligned}
 & \|\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x)\|_F^2 \\
 &= \text{tr}((\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x))^T(\Theta^{1/2}\phi(x) - YY^T\Theta^{1/2}\phi(x))) \quad (\|\mathcal{B}\|_F^2 = \text{tr}(\mathcal{B}^T\mathcal{B}) \text{ for any matrix } \mathcal{B}) \\
 &= \text{tr}(\Theta^{1/2}\phi(x)^T\phi(x)\Theta^{1/2} - \Theta^{1/2}\phi(x)^T\phi(x)YY^T - \\
 &\quad \text{Same trace} \quad \underbrace{YY^T\Theta^{1/2}\phi(x)^T\phi(x)} + \underbrace{YY^T\Theta^{1/2}\phi(x)^T\phi(x)YY^T} \quad \text{Expanding prev line}
 \end{aligned}$$

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

$$= \text{tr}(\Theta^{1/2}K\Theta^{1/2}) - \text{tr}(Y^T\Theta^{1/2}K\Theta^{1/2}Y) \quad \text{only if } A \text{ and } B \text{ are square matrices}$$

Finally, we have

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

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

$\Downarrow A$

(new A , not $\Theta^{1/2}K\Theta^{1/2}$!!
Just notation clashes)

Only dependent on:
 1) Weights Θ - fixed
 2) Kernel $f^2 \phi(\cdot)$ - fixed
 3) Data X - fixed
 $\therefore \text{tr}(\Theta^{1/2}K\Theta^{1/2})$ cannot be minimized using Y .

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) :

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

*Mean looking for the
largest eigenvalues*

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

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

*spectral embedding
of each datapoint* *cols 1 to k
of matrix U*

[1] Helmburg, 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 :

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

with

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

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

We have

$$\begin{aligned} \text{tr}(Y^T A Y) &= \sum_{q=1}^k y_q^T A y_q = \sum_{q=1}^k y_q^T \lambda_q y_q = \sum_{q=1}^k \lambda_q y_q^T y_q = \sum_{q=1}^k \lambda_q \text{ (k largest values)} \\ &= \max_Y \text{tr}(Y^T A Y) \text{ s.t. } Y^T Y = I_k \end{aligned}$$

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

Spectral clustering

- Compute clusters as follows :
 - Compute matrix A : $A = \theta^{1/2} K \theta^{1/2}$ $= \langle H \rangle^k \phi(x)^T \phi(x) \langle H \rangle^k$
 - Perform eigenvalue decomposition of A : $Au_q = \lambda_q u_q$, $u_q \in \mathbb{R}^{n \times k}$, $1 \leq q \leq k$
 - Form spectral solution : $Y^* = (u_1, \dots, u_k) \in \mathbb{R}^{n \times k}$
 - 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].

To generate
spectral
embeddings ✓

[1] Von Luxburg, A tutorial on spectral clustering, 2007 (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].
- To construct matrix $K = \phi(x)^T \phi(x)$. Should it then be $O(n^2d^2)$?*

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

Lab 2 : Kernel k-means

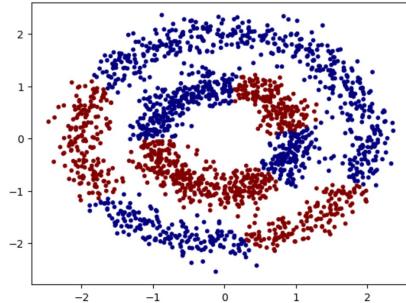
- Run code02.ipynb and test kernel k-means with
 - EM approach
 - Spectral approach

```
# 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 solution with EM approach. Accuracy= 61.5, Energy= 0.935



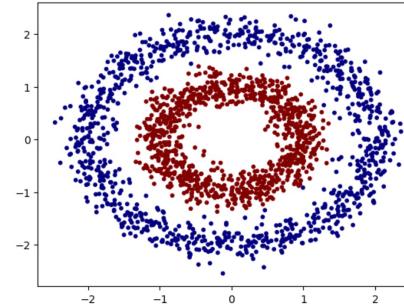
Kernel k-means
EM approach

```
# 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 solution with spectral approach. Accuracy= 99.6 Energy= 0.371



Kernel k-means
Spectral approach

NL-k-means involving projecting our weighted data $B = \bigodot^k X$ onto the column space of $A = \bigodot^k F$

$$L_{NL\text{-}k\text{-means}} = \frac{1}{kn} \left\| \underbrace{\theta^{1/2} \phi(x)}_{\text{mapped weighted data}} - \underbrace{\theta^{1/2} M}_{\text{weighted centroids/reconstructions}} \right\|_F^2,$$

From previous derivation, row q of C , $C_q = m_q^T = \begin{bmatrix} \sum_{i \in S_q} \theta_i \phi(x_i) \\ \sum_{i \in S_q} \theta_i \end{bmatrix}^T$

$$C = ZF^T \bigodot \Xi$$

To construct M where row $M_i = C_{q(i)}$ ($q(i)$ = centroid assigned to datapoint i) we must broadcast each centroid c to all rows in M where $q(i) = c$. We use matrix F for this:

$$M = FC = FZC^T \bigodot \Xi$$

$$\begin{aligned} \therefore L_{NL\text{-}k\text{-means}} &= \frac{1}{kn} \left\| \bigodot^k \Xi - \bigodot^k M \right\|_F^2 \\ &= \frac{1}{kn} \left\| \bigodot^k \Xi - \underbrace{\bigodot^k F Z C^T \bigodot \Xi}_\text{Define } A := C \right\|_F^2 \\ &= \frac{1}{kn} \left\| \bigodot^k \Xi - AC \right\|_F^2 \end{aligned}$$

Key Idea: We want to project our data $\bigodot^k \phi(x)$ onto $\text{col}(A)$ because we want to find $\min_C \left\| \bigodot^k \Xi - AC \right\|_F^2$

while enforcing the piecewise-per-cluster structure from the cluster assignments, which A encodes

Since $\text{col}(A)$ encodes this structure, $\forall v \in \mathbb{R}^n$,

$Av \in \text{col}(A)$. $\therefore \forall C = [C_1 \cdots C_k] \in \mathbb{R}^{n \times k}$,

$AC_j \in \text{col}(A) \quad \forall j = 1, \dots, k$.

In other words, Av is guaranteed to be piecewise constant per cluster

This is desirable because we want each datapoint/row in M to be equal (after unweighting by doing a left multiple of $\bigodot^{1/2}$ onto $\bigodot^{1/2} M = AC$)

across all datapoints with the same assignment in A .

And this piecewise constant across all columns of AC property ensures that all rows in AC_j with the same assignment in A have the same value (after unweighting) $\sqrt{\theta_j} [AC_j]_i$. So $\forall j, [AC_j]_i$ will be equal for datapoints in the same cluster assignment.

$$\bigodot = \text{diag}(\theta_1, \dots, \theta_n) \in \mathbb{R}^{n \times n}$$

$$X = \text{original data matrix} \in \mathbb{R}^{n \times d}$$

$$\Phi = \text{mapped data matrix} \in \mathbb{R}^{n \times d}$$

$$\text{where row } \Xi_i = [\phi(x_{i1}), \dots, \phi(x_{id})]$$

$$F = \text{hard-clustering matrix} \in \{0, 1\}^{n \times k}$$

$$M = \text{mean matrix} \in \mathbb{R}^{n \times d}$$

$$\text{where row } M_i = \text{coord of assigned centroid.}$$

$$\therefore \text{There are duplicate rows in } M$$

$$\therefore \text{rank}(M) = k$$

$$C = \text{centroid matrix} \in \mathbb{R}^{k \times d}$$

$$K = XX^T = \bigodot \Xi \Xi^T \bigodot^{1/2} \in \mathbb{R}^{n \times n}$$

This is called the Kernel/Gram matrix

$$Z = \text{Normalizer matrix} \in \mathbb{R}^{k \times k}$$

$$\begin{aligned} &= \text{diag} \left(\frac{1}{\sum_{i \in S_1} \theta_i}, \dots, \frac{1}{\sum_{i \in S_k} \theta_i} \right) \\ &= (F^T \bigodot F)^{-1} \end{aligned}$$

\Leftrightarrow all rows in $A^T C$ that belong to the same cluster will be equal after left-multiplying by $(H)^{-\frac{1}{2}}$ (i.e. unweighting)

Proof that A enforces piecewise constant property. $A = (H)^{\frac{1}{2}} F \in \mathbb{R}^{n \times k}$

$$\rightarrow A_{i,j} = \begin{cases} \sqrt{\theta_i} & \text{if datapoint } i \text{ is assigned to cluster } j \text{ ie. } c(i) = j \\ 0 & \text{otherwise} \end{cases}$$

\vec{x}_j is a \mathbb{R}^n vector containing the j -th feature of all n datapoints

$\hat{\vec{x}}_j$ is our estimated reconstruction of it i.e. its projection onto $\text{col}(A)$.

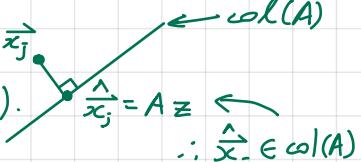
$$\exists z \in \mathbb{R}^k \text{ st. } \hat{\vec{x}}_j = Az \text{ and } \text{col}(A) \perp Az - \vec{x}_j \equiv A^T(Az - \vec{x}_j) = 0$$

WTS: $\forall q \in \mathbb{Z}_{[1,k]} \quad \forall i_1, i_2 \in \{i_1, i_2 \mid c(i) = q\},$

$$(\hat{\vec{x}}_j)_{i_1} = (Az)_{i_1} = \sum_{j=1}^k A_{i_1 j} z_j$$

$$\text{key } = \boxed{\sqrt{\theta_{i_1}} z_{c(i_1)}}$$

$$\frac{(\hat{\vec{x}}_j)_{i_1}}{\sqrt{\theta_{i_1}}} = \frac{(\hat{\vec{x}}_j)_{i_2}}{\sqrt{\theta_{i_2}}}$$



i_2 th value in vector $\hat{\vec{x}}_j$

$$\therefore (\hat{\vec{x}}_j)_{i_1} = \sqrt{\theta_{i_1}} z_{c(i_1)} \text{ and } (\hat{\vec{x}}_j)_{i_2} = \sqrt{\theta_{i_2}} z_{c(i_2)}$$

But $c(i_1) = c(i_2)$ since they belong to same cluster

$$\therefore \frac{(\hat{\vec{x}}_j)_{i_1}}{\sqrt{\theta_{i_1}}} = \frac{(\hat{\vec{x}}_j)_{i_2}}{\sqrt{\theta_{i_2}}} \quad \square$$

dim of $\phi(x)$

The above result is true for $j = \{1, \dots, d\}$,

$$\therefore \underbrace{[(H)^{-\frac{1}{2}} AC]_{i_1, \cdot}}_{\text{Row } i_1 \notin (H)^{-\frac{1}{2}} AC} = \underbrace{[(H)^{-\frac{1}{2}} AC]_{i_2, \cdot}}_{\text{Row } i_2 \notin (H)^{-\frac{1}{2}} AC}.$$

$$(H)^{-\frac{1}{2}}(AC) = (H)^{-\frac{1}{2}}((H)^{\frac{1}{2}}FC) = M := FZF^T H \Phi$$

$$\therefore \text{Row } i_1 \text{ of } M = \text{Row } i_2 \text{ of } M$$

To project our mapped data Φ onto $\text{col}(A)$, we define a projection matrix P onto $\text{col}(A)$

$$P := A(A^T A)^{-1} A^T$$

$$= (H)^{\frac{1}{2}} F Z F^T H \quad (\because A^T A = F^T H F = Z^{-1} \rightarrow (A^T A)^{-1} = Z)$$

$$\text{Project feature } j : P \Phi_{\cdot, j} \quad \text{Do this on all cols of } \Phi \quad P \Phi = (H)^{\frac{1}{2}} F Z F^T H \Phi = (H)^{\frac{1}{2}} M$$

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 $O(\underline{n^2d+n^2kn_i})$ suitable for non-linear data but with critical initialization.
Building matrix ↗ ↘ *why*
K
- 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.
complexity becomes dependent on $|E|$ rather than n ↗
graph is very sparse ↗
- 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.

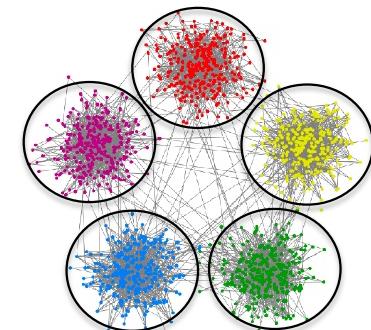
Representing the data as graphs also introduces sparsity by only forming an edge between $i & j$ iff $i & j$ are in the same cluster

If $K_{ij} = [\phi(x_i)\phi(x_j)]_{ij}$ is very small, we can introduce thresholding and set all K_{ij} to 0 if $K_{ij} < c$
↑ threshold
↓
This creates sparsity in K .

Graph partitioning

- ① Perform Graph Partitioning
- ② Do different analysis / feature extraction on these smaller partitioned graphs on different GPUs
- ③ Combine everything

- 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].



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

Source: Abbe, JMLR'17

Data clustering as graph partitioning

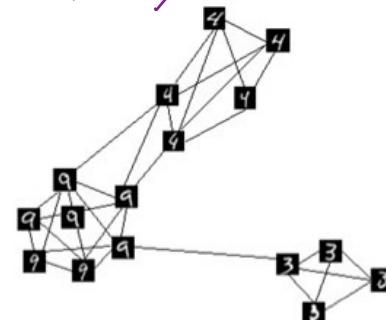
Note: Ideally we extract features from each image/dataset and we use these hand-crafted/learned features to construct G .

- Graph construction : Build a (sparse) k-nearest neighbors (k-NN) graph $G=(V,E,A)$ from a dataset $V=\{x_1, \dots, x_n\} \in \mathbb{R}^d$. *more efficient*
- The graph representation of the dataset **avoids working with d-dim features directly**.
- Essentially, this process transforms the $n \times d$ data features into a set of E edges, resulting in a significant compression of the dataset. *To do one dot product* *Bottleneck of this algorithm :: Need to compute dot product between every 2 datapoints*
- 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. *But this can be parallelized*

5	3	7	3	1	9	0	5	4	5
0	8	2	3	2	0	6	7	8	1
3	1	7	3	4	5	6	2	4	9
0	4	2	1	8	9	6	7	3	5
6	1	9	3	4	5	4	9	2	9
0	7	2	3	6	5	6	7	8	8

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

k-NN graph
construction



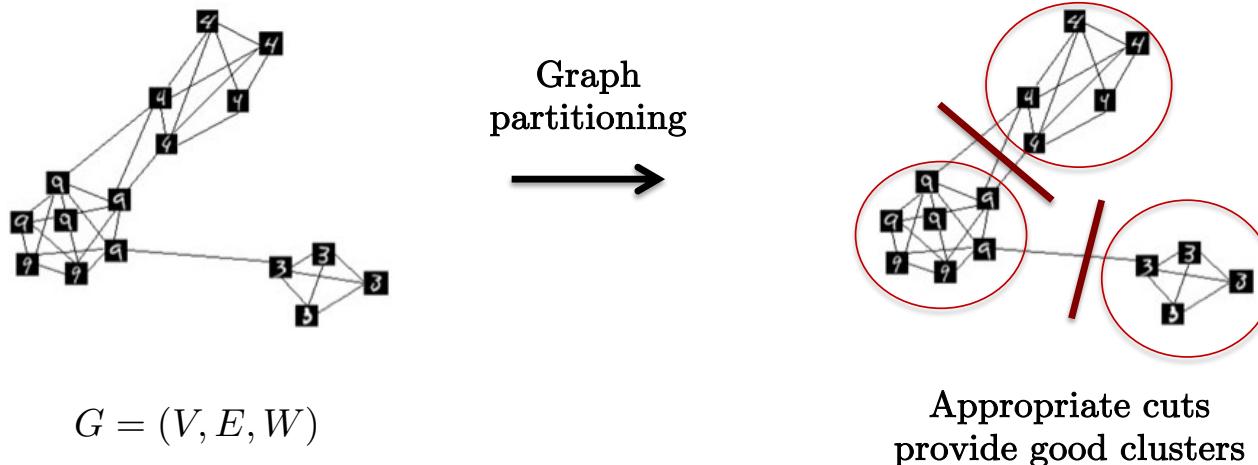
$$G = \{V, E, A\}, A \in \mathbb{R}^{n \times n}$$

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

See: KD Trees to speed this up. ?

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.



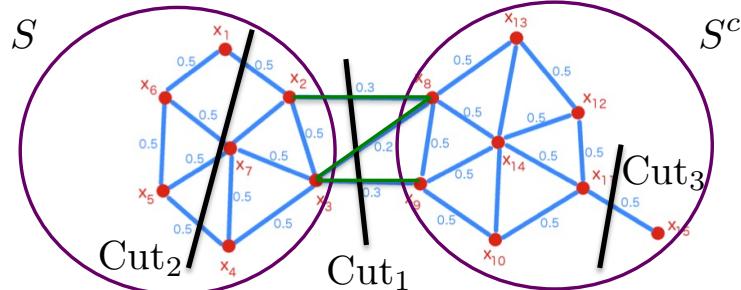
Outline

- 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 Association

Cut partitioning

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



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

sum of all the cut edges' weights

Value of Cut_1 : $\text{Cut}(S, S^c) = 0.3 + 0.2 + 0.3 = 0.8$

Value of Cut_2 : $\text{Cut}(S, S^c) = 0.5 + 0.5 + 0.5 + 0.5 = 2.0$

Value of Cut_3 : $\text{Cut}(S, S^c) = 0.5$ ← Best cut if we only focus on cut value without each partition's volume

- 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 :

① min Cut and max Vol \Leftrightarrow min $\frac{\text{Cut}}{\text{Vol}}$ 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.)

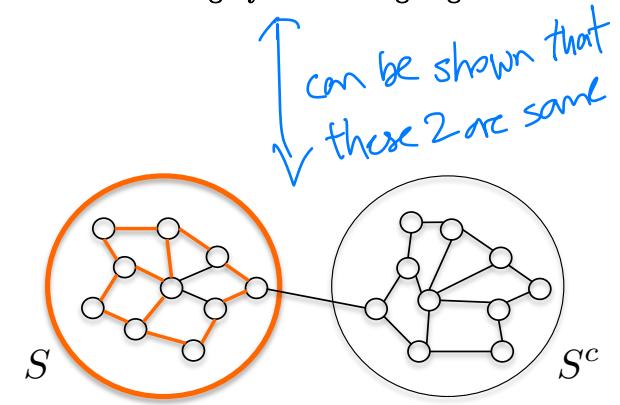
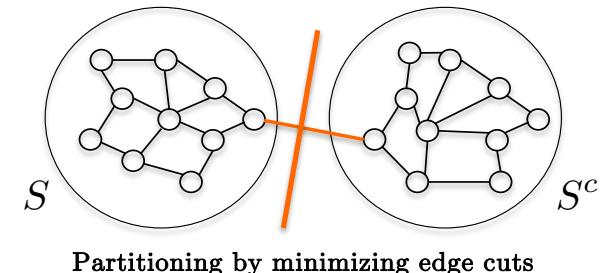
$\text{Assoc}(S, S^c) = \sum_{i \in S, j \in S} A_{ij}$ (num of connections inside S)

Sum of all degrees of nodes in the partition
(Sum of "internal" edges in the partition) + 2

Sum of edges WITHIN the cluster



not in Assoc calculation



[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 :

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

constraints

→ indicates which cluster each node belongs to

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

But why? Some problem

dega matrix

$$\max_{F \in \{0,1\}^{n \times k}} \sum_{q=1}^k \frac{F_{\cdot,q}^T A F_{\cdot,q}}{F_{\cdot,q}^T D F_{\cdot,q}}$$

Assoc *Volume*

s.t. $\sum_{q=1}^k F_{i,q} = 1 \quad \forall i \in V$

each node can only belong to 1 cluster

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

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

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

(point-wise representation)

Spectral relaxation

turn the problem to a
continuous problem

- 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 $\text{binary}^{n \times k}$ to its nearest convex set, i.e. $\mathbb{R}^{n \times k}$, which renders the optimization continuous.
- Subsequently, the spectral theorem^[1,2] provides the solution:
 - The k largest eigenvectors of matrix A obtained through EVD.

↓
 $\therefore \exists$ Global minimizer to this
relaxed optimization problem

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

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

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

[1] Helberg, 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

Similar to L_{sym}

- The equivalence between kernel k-means and balanced cuts has been established^[1,2].

- Kernel K-Means : $\max_{Y \in \mathbb{R}^{n \times k}} \text{tr}(Y^T BY) \text{ s.t. } Y^T Y = I_k, B = \Theta^{1/2} K \Theta^{1/2}$

- Balanced Cuts : $\max_{Y \in \mathbb{R}^{n \times k}} \text{tr}(Y^T BY) \text{ s.t. } Y^T Y = I_k, B = D^{-1/2} A D^{-1/2}$

which are equivalent for $\Theta = D^{-1}, K = A$

with/normalize by degree



- 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.

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

Gaussian Kernel
Recent graph construction in
Lect 2 (Slide 36)

If we apply thresholding to
set all values beyond the
distance/kernell to 0, and we use the
same mapping in A & $\phi(\cdot)$, then
 $K_{ij} = A_{ij}$

[1] Bach, Jordan, Learning Spectral Clustering, 2003

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

$K(x_i, x_j) = x_i^T x_j$	Time consuming	(linear kernel for linear k-means)
$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \exp(-\ x_i - x_j\ _2^2/\sigma^2)$		(Gaussian kernel)
$K(x_i, x_j) = (ax_i^T x_j + b)^c$	Efficient kernel computation	(Polynomial kernel)

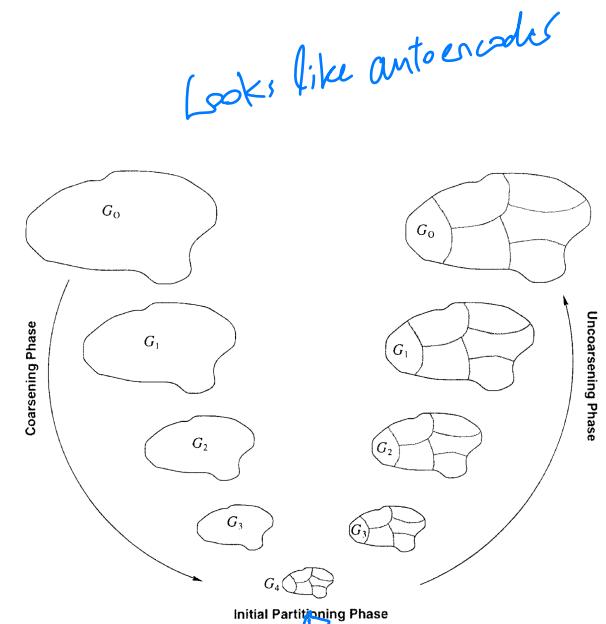
most for uscd

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].

Deep Graph Library

Graph Coarsening → Think data compression



[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.data_loading.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.data_loading.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],
                                               sampler.partition_node_ids[idx_start:idx_end])):
    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)
```

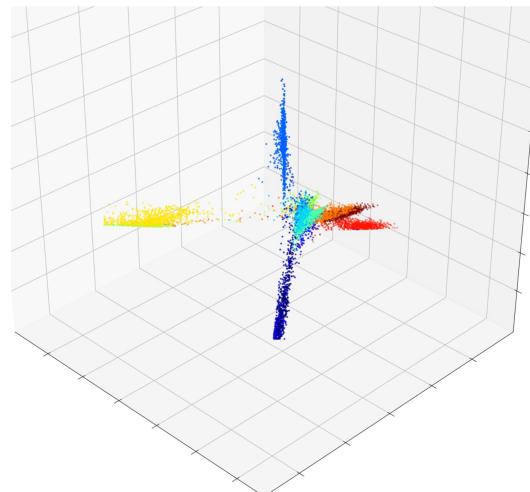
Convert a graph into a bidirected graph: 0.004 seconds, peak memory: 0.000 GB
 Construct multi-constraint weights: 0.000 seconds, peak memory: 0.000 GB
 Metis partitioning: 0.031 seconds, peak memory: 0.000 GB
`C_metis_dgl tensor([1, 0, 0, ..., 9, 9, 9])`
`[23:27:38] /tmp/dgl_src/src/graph/transform/metis_partition_hetero.cc:89: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

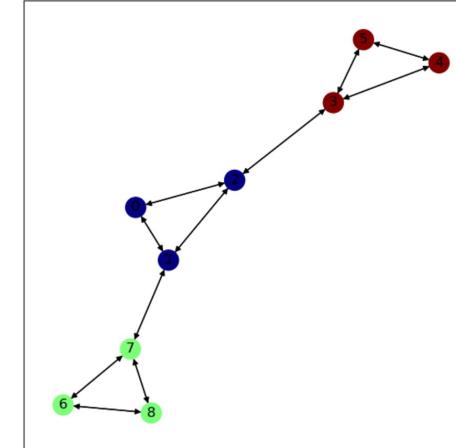
Accuracy Metis PyG : 77.16713271671327



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

try: os.remove("cluster_gcn.pkl") # remove any existing partition
except: pass
num_parts = 3
sampler = dgl.data_loading.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],
                                               sampler.partition_node_ids[idx_start:idx_end])):
    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, 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 :

$$\begin{aligned} & \min_{F \in \{0,1\}^{n \times k}} \sum_{q=1}^k \frac{F_{\cdot,q}^T L F_{\cdot,q}}{F_{\cdot,q}^T D F_{\cdot,q}}, \quad \text{with } L = D - A \quad \text{and} \quad \sum_{q=1}^k F_{i,q} = 1 \quad \forall i \in V \\ & \min_{Y \in \text{binary}^{n \times k}} \text{tr}(Y^T B Y) \quad \text{s.t.} \quad Y^T Y = I_k, \quad B = I - \underbrace{D^{-1/2} A D^{-1/2}}_{\text{Normalized Laplacian } L_{\text{sym}}} \\ & \quad \text{with } Y_{\cdot,q} = \frac{D^{1/2} F_{\cdot,q}}{\|D^{1/2} F_{\cdot,q}\|_2} \quad (\text{vectorial representation}) \\ & \quad Y_{iq} = \begin{cases} \sqrt{\frac{D_{ii}}{\text{Vol}(S_q)}} & \text{if } i \in S_q \\ 0 & \text{otherwise} \end{cases} \quad (\text{point-wise representation}) \end{aligned}$$

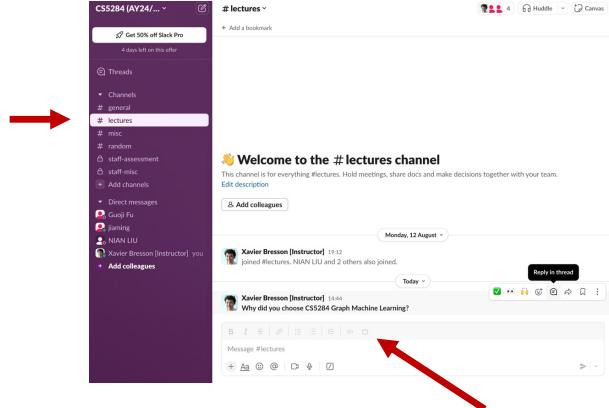
[1] Shi, Malik, Normalized cuts and image segmentation, 2000
[2] Yu, Shi, Multiclass spectral clustering, 2003

In-lecture question

- How do you compute an approximate solution of the Normalized Cut optimization problem? What is the operator B?

$$\min_{F \in \{0,1\}^{n \times k}} \sum_{q=1}^k \frac{F_{\cdot,q}^T L F_{\cdot,q}}{F_{\cdot,q}^T D F_{\cdot,q}}, \text{ with } L = D - A \text{ and } \sum_{q=1}^k F_{i,q} = 1 \forall i \in V$$
$$\min_{Y \in \text{binary}^{n \times k}} \text{tr}(Y^T B Y) \text{ s.t. } Y^T Y = I_k, B = I - D^{-1/2} A D^{-1/2}$$

- In Slack #lectures
 - Identify the question and Reply in thread with a short response



Normalized cut

- As before, relaxing the binary constraint Y from binary $^{n \times k}$ to the nearest convex set $R^{n \times k}$ 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 improvement 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 :

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

Solved by EVD.

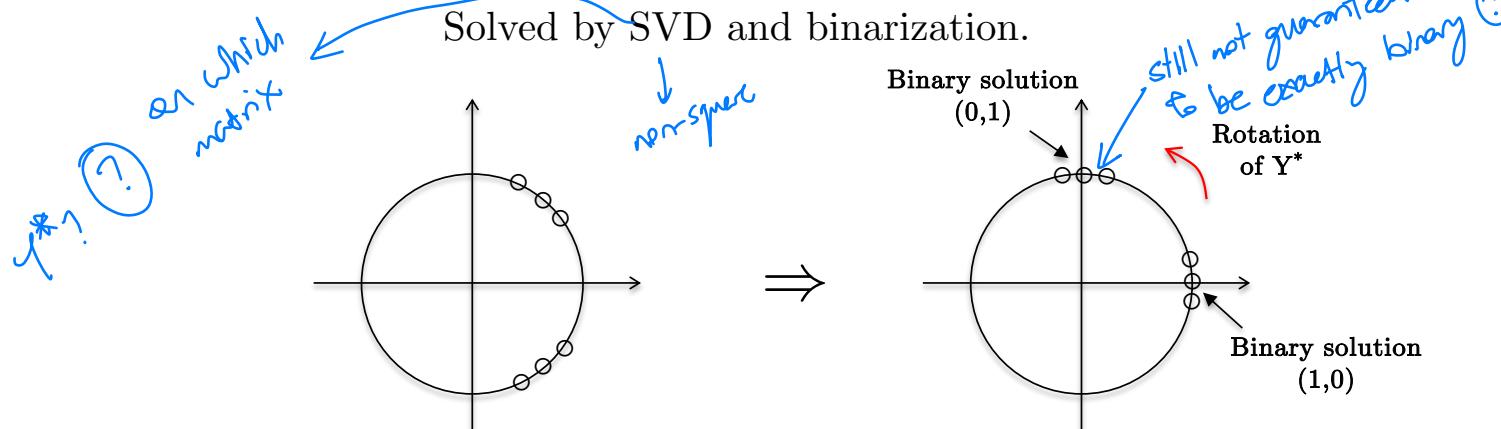
Seen before

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

Identifying the rotation that aligns best with the binary constraint.

$$Z^* = \arg \min_{Z, R} \|Z - Y^* R\|_F^2 \text{ s.t. } R^T R = I_k, Z \in \{0, 1\}^{n \times k}$$

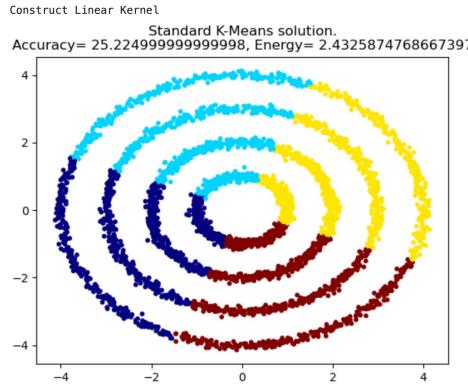
Solved by SVD and binarization.



Lab 4 : NCut

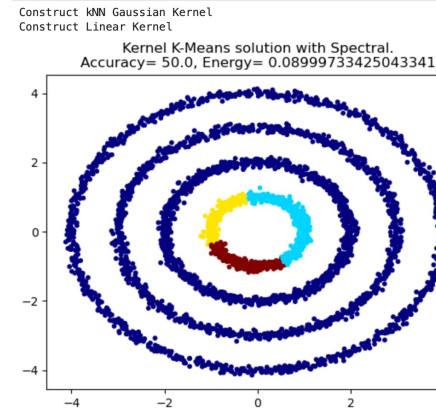
- Run code04.ipynb and test Ncut on non-linear datasets with 4 classes

```
# 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()
```



Linear k-means
EM approach

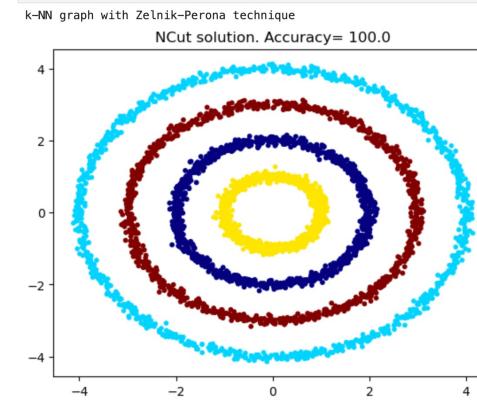
```
# Run kernel/non-linear k-means with spectral approach
Ker = construct_kernel(X, 'kNN_gaussian', 100)
# Kernel K-Means with Spectral approach
C_kmeans, En_kmeans = compute_kernel_kmeans_spectral(nc, Ker, Theta, 10)
# Plot
plt.figure(3)
size_vertex_plot = 10
plt.scatter(X[:,0], X[:,1], s=size_vertex_plot*np.ones(n),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



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=plt.cm.jet())
plt.title('NCut solution. Accuracy= ' +
str(compute_purity(C_ncut,Cgt,nc)) )
plt.show()
```



NCut
Spectral approach

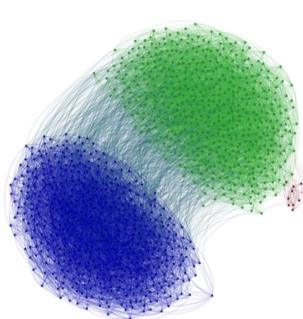
Outline

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

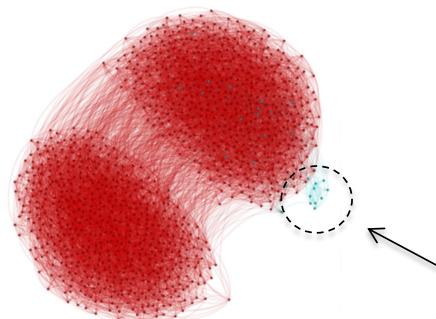
|| Not in
text

Product cut

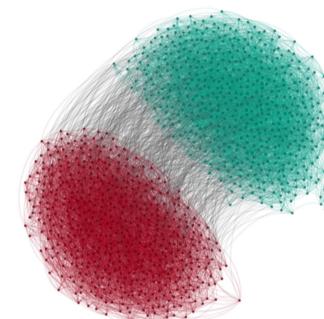
- Limitations
 - Standard cuts^[1] are biased towards data outliers.
 - Balanced Cuts^[2] s.a. NCut are biased towards cluster outliers.
- Product Cut^[3] are specifically designed and guaranteed to maintain robustness in the presence of cluster outliers.



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 :

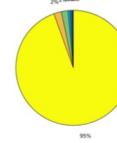
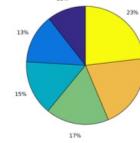
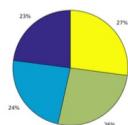
$$\min_{\{S_q\}_{q=1}^k \text{ s.t. } \cup_q S_q = V, \cap_q S_q = \emptyset} \frac{\prod_{q=1}^k \text{Connec}(S_q, S_q^c)}{\exp(H(\{S_q\}))}$$

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

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

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

	Partition \mathcal{P} of WEBKB4 found by the Pcut algo.	Partition \mathcal{P} of WEBKB4 found by the Ncut algo.	Partition \mathcal{P} of CITESEER found by the Pcut algo.	Partition \mathcal{P} of CITESEER found 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



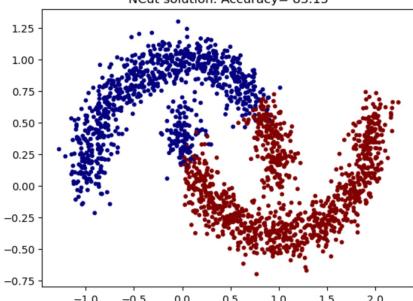
Lab 5 : PCut

- 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()

K-NN graph with euclidean distance
```

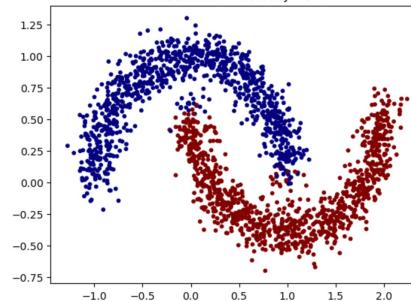


NCut solution

```
# 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()

K-NN graph with euclidean distance
```



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 USPT 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 → no need to fix k
a priori
- 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 :

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

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

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

Number of clusters

- Observe that

$$\max_{k, C: V \rightarrow \{1, 2, \dots, k\}} \sum_{ij} \left(A_{ij} - \gamma \frac{d_i d_j}{\sum_{i'j'} A_{i'j'}} \right) \delta(C_i, C_j) \quad \leftarrow \text{modularity function}$$

is equivalent to

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

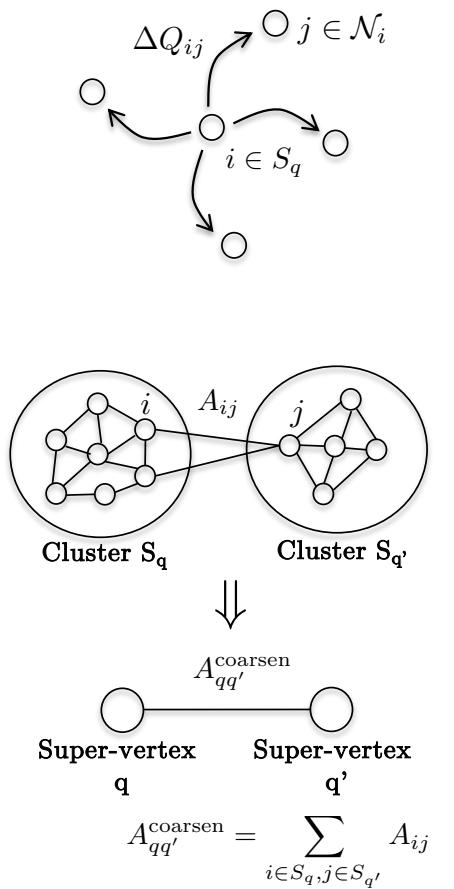
which is a relaxation of the balanced cut we call LCut:

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

- 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 i 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.



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(Clov, 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
```

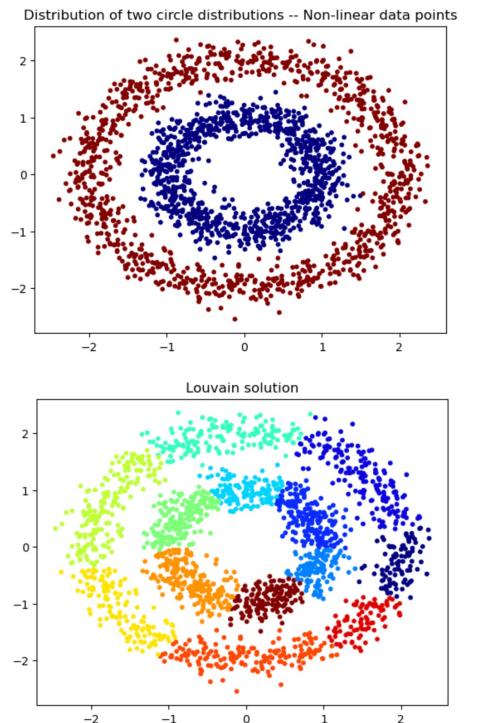
```
# Run Louvain algorithm
W = construct_knn_graph(X, 50, 'euclidean_zelnik_perona')
Wnx = nx.from_numpy_array(W)
partition = community.best_partition(Wnx)
nc_louvain = len(np.unique( [partition[nodes] for nodes in partition.keys()]))
n = len(Wnx.nodes())
print('nb_data:', n, ', nb_clusters=', nc_louvain)

# 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(Clov, 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()

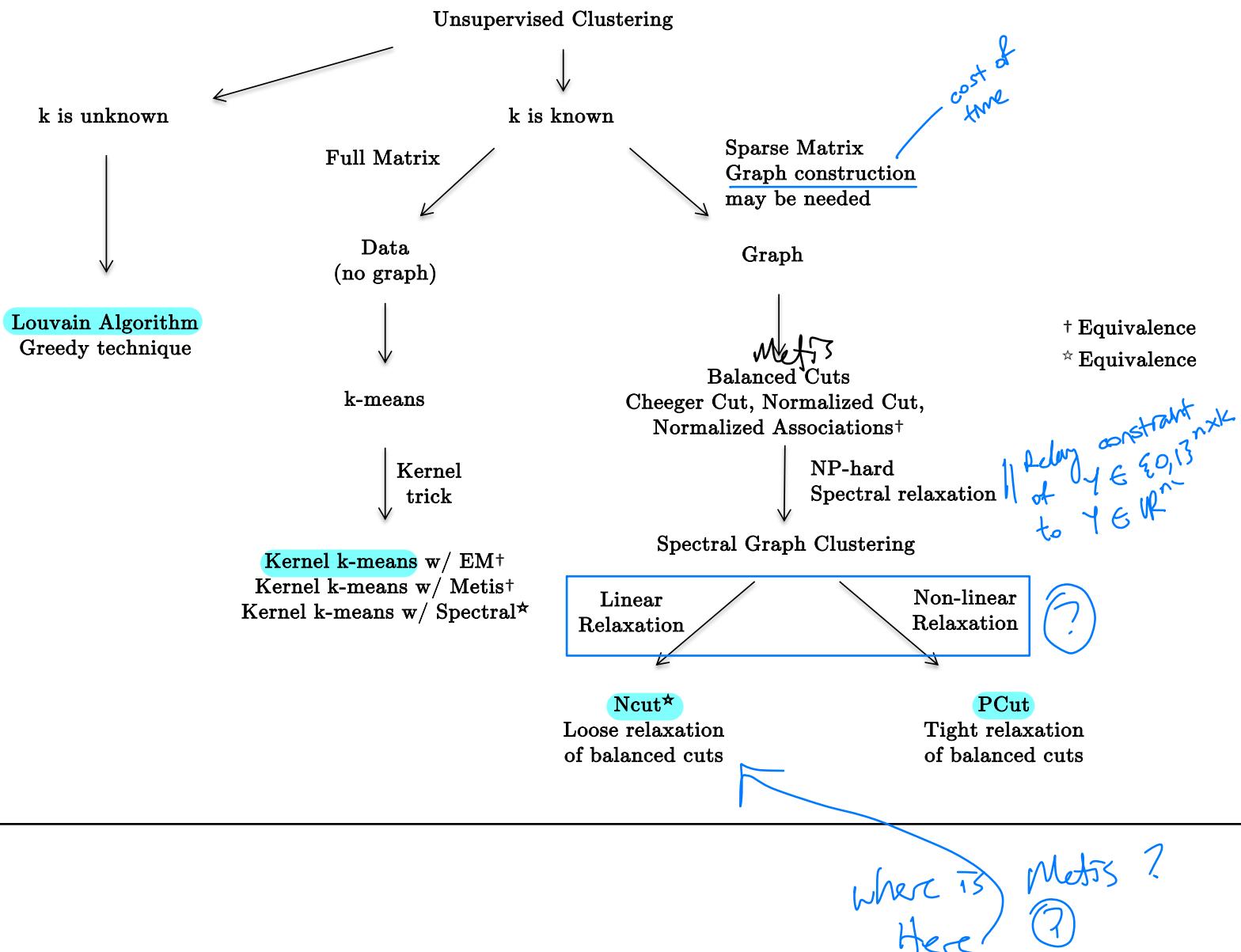
k-NN graph with Zelnik-Perona technique
nb_data: 2000 , nb_clusters= 14
accuracy_louvain= 99.8  with nb_clusters= 14
```



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 connects combinatorial and discrete optimization, continuous optimization, graph theory, and spectral solutions.
- Linear and kernel k-means are 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?