Spectral Clustering

Presented by: Yosser Fhal

What is Clustering?

- Clustering means **grouping similar data points.**
- Traditional clustering (like **K-Means**) uses **distance** between points.
- But what if “closeness” isn’t just about distance; it’s about **relationships**?

Example:
In a social network, two people might not be directly connected, but they may still belong to the same friend group through shared connections!

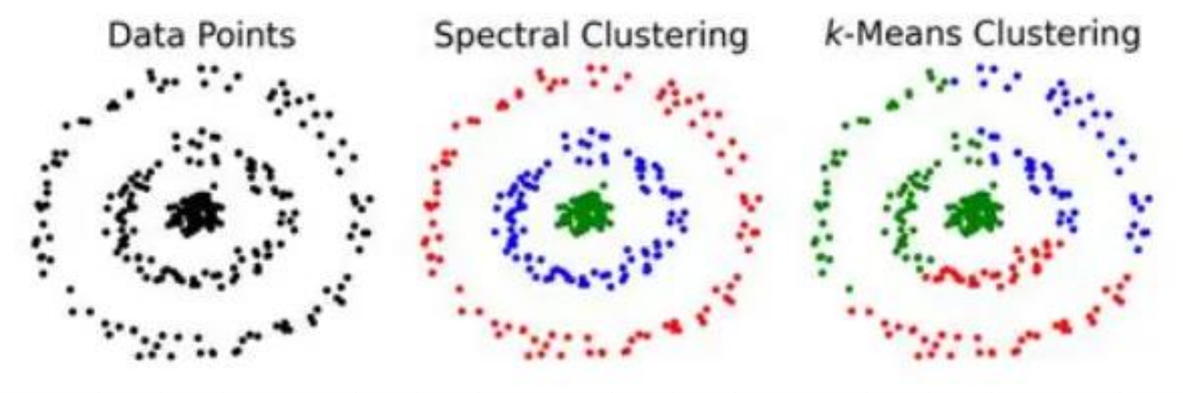
→ That’s when **Spectral Clustering** helps.

What is Spectral Clustering?

- Spectral clustering looks at how **points are connected** instead of just how far apart they are.
- It does this by:
  1. Turning the data into a **graph,** points become **nodes**, and connections between them become **edges**.
  2. Studying that graph using **math tools called eigenvalues and eigenvectors**.
  3. Finding natural groups or “communities” within the graph.
- Think of it like finding communities of friends in a big social network!

Why is Spectral Clustering powerful?

- Works with **irregular** or **non-spherical shapes**.
- Can find **complex**, **non-linear patterns**.
- Excellent for **graph-based data** (like social networks or web pages)
- Doesn’t depend only on geometric distance.



Representing Data as a Graph

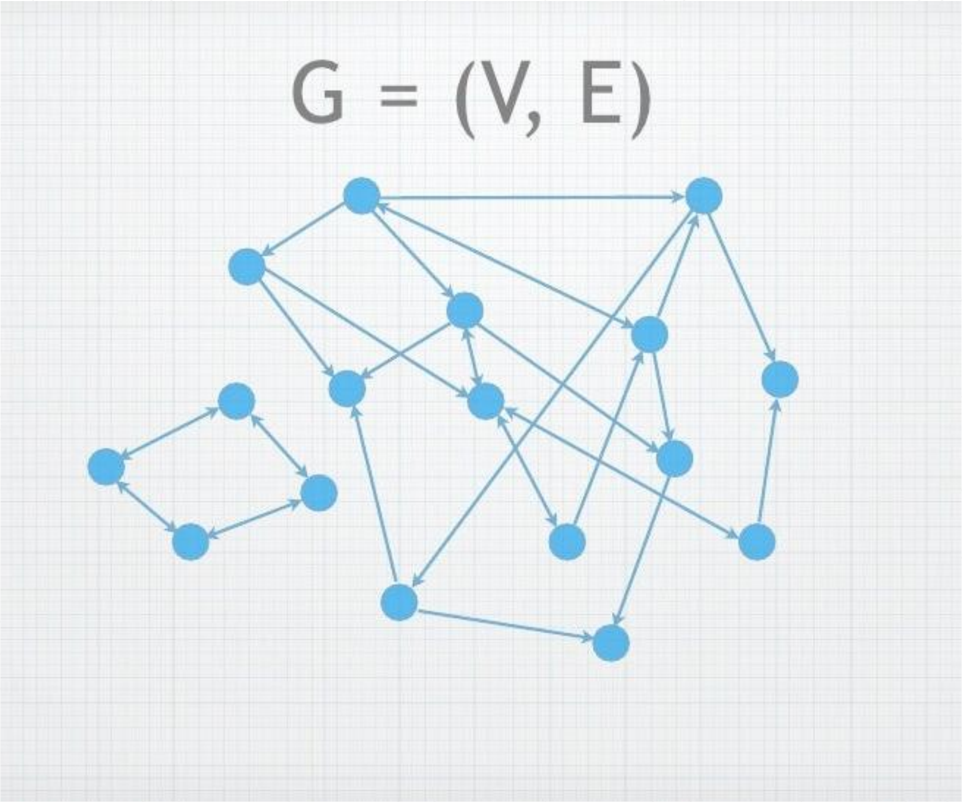
We convert the dataset into a **graph**

$$G = (V, E)$$

Where:

- **Vertices (V)** → Data points.
- **Edges (E)** → Connections or similarity between points.

Then, we use the **eigenvectors of a special matrix (the Laplacian)** to separate the graph into groups (clusters).

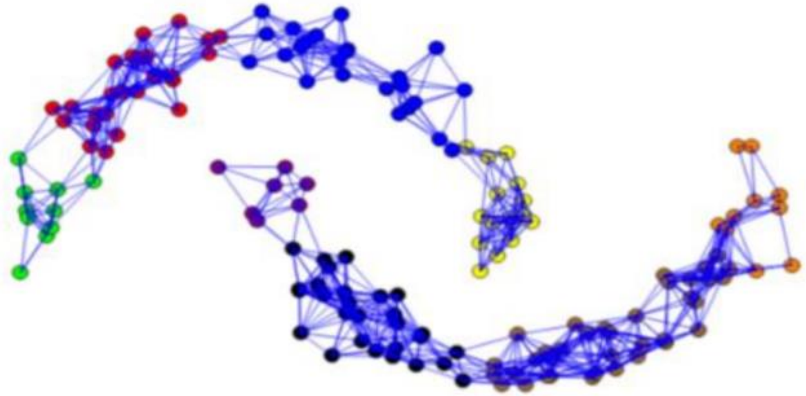


Building the Similarity Graph (1/3)

There are three main ways to connect nodes:

1. **ε-neighborhood graph**
   - Connect points if distance < ε

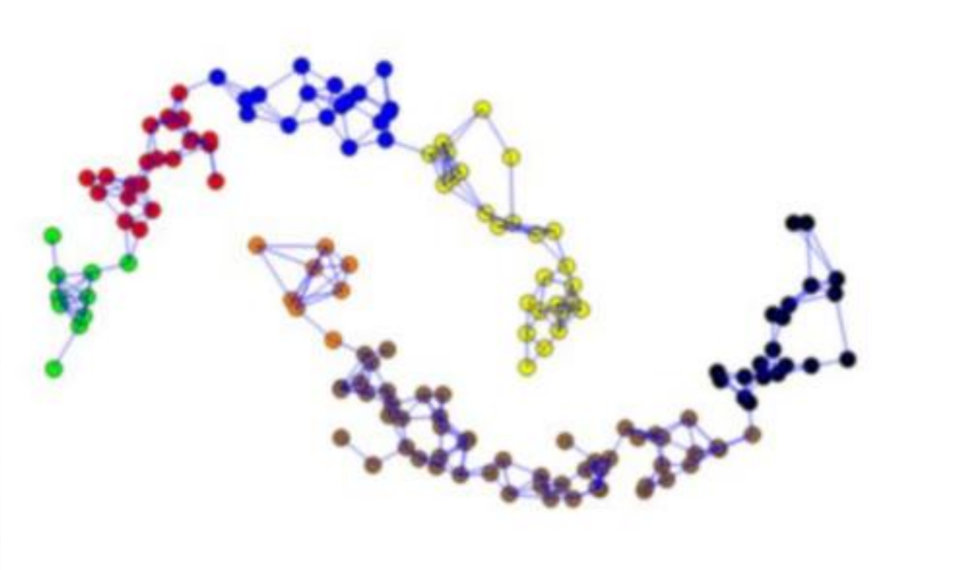
   → Simple, but sensitive to ε value



Representing Data as a Graph (2/3)

2. **k-nearest neighbours (k-NN) graph**
   - Each point connects to its *k* closest neighbours.

   → Captures local structure



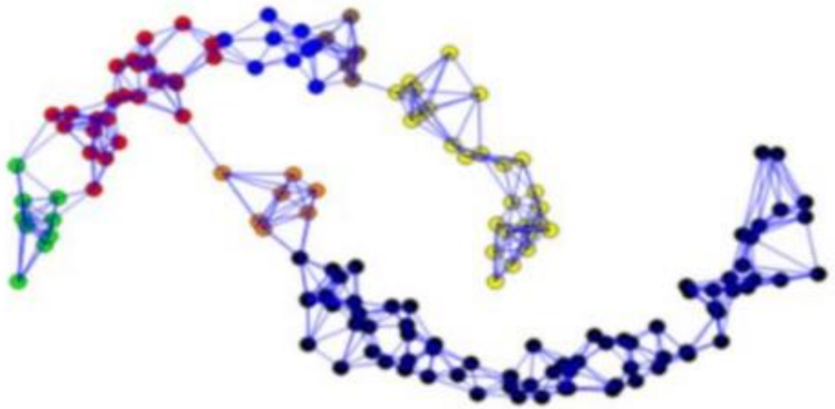
Representing Data as a Graph (3/3)

3. **Fully-connected graph**

- Every point connects to every other, with weight

$$s(x_i, x_j) = \exp\left(-\frac{\| x_i - x_j \|^2}{2\sigma^2}\right)$$

→ Captures global relationships



(x_i, x_j): *data points* | $s(x_i, x_j)$: *similarity*

$\| x_i - x_j \|^2$: *squared Euclidean distance* $= \sum_{k=1}^{d} d(xi,k - xj,k)^2$

σ: *Scale parameter* **Small σ** → similarity decays fast **Large σ** → similarity decays slowly

Matrices in Spectral Clustering (1/3)

- Once we build the graph, we represent it in **matrix form**.

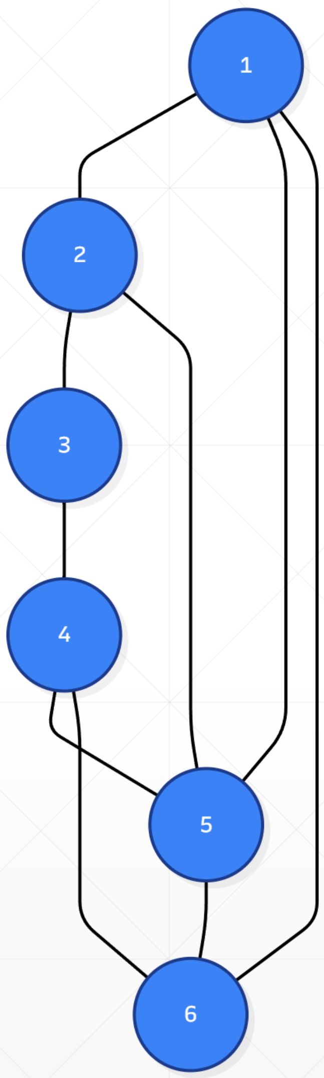
Adjacency (Affinity) Matrix A

$$A_{ij} = \begin{cases} s_{ij}, & \text{if nodes } i,j \text{ are connected} \\ 0, & \text{otherwise} \end{cases}$$

→ Represents **connection strength** between nodes.

s_{ij}: *similarity between nodes*

	1	2	3	4	5	6
1	0	1	0	0	1	1
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	1	0	1	0	1	1
5	0	1	0	1	0	1
6	0	1	1	0	1	0



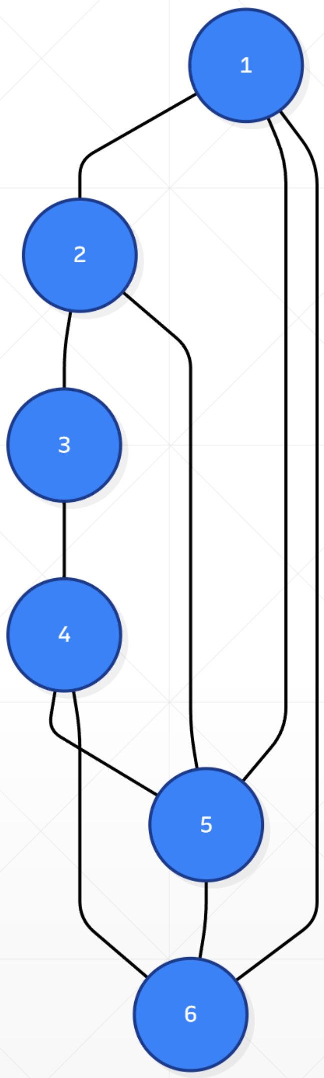
Matrices in Spectral Clustering (2/3)

Degree Matrix D

Diagonal matrix with: $$D_{ii} = \sum_j A_{ij}$$

→Represents the **strength of the connections** of each node .

	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	2	0
6	0	0	0	0	0	3



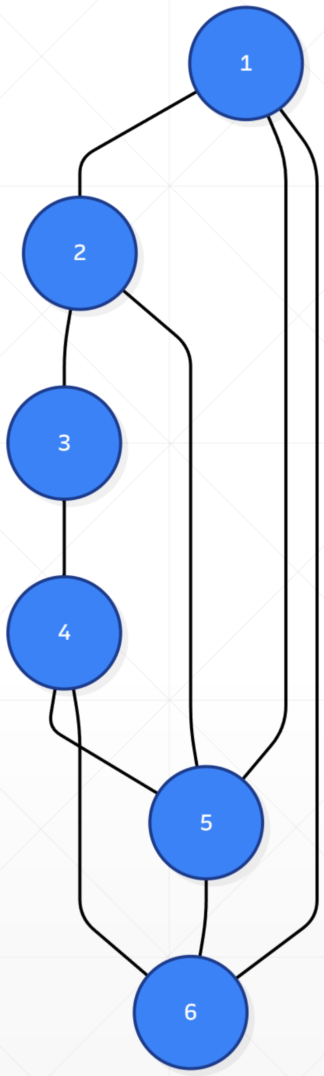
Matrices in Spectral Clustering (3/3)

Laplacian Matrix L

Core of spectral clustering: $L = D - A$

→ It measures how much each **node differs from its neighbours**.

	1	2	3	4	5	6
1	3	-1	0	-1	-1	0
2	0	2	-1	0	-1	0
3	-1	0	3	0	-1	-1
4	0	0	-1	3	0	-1
5	-1	-1	0	0	2	0
6	0	-1	0	-1	0	3



Normalized Laplacians

To handle graphs with uneven connections, we “normalize” L.

Two versions:

- **Symmetric:**

$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

- **Random Walk:**

$$L_{rw} = D^{-1} L = I - D^{-1} A$$

Why normalize? Because some nodes may have many more connections than others; normalization balances their importance.

$A: Adjacency\ matrix$ $I: Identity\ matrix$
$D: Diagonal\ matrix$ $L: Unnormalized\ Laplacian\ matrix$

The Graph Cutting Idea

The goal is to cut the graph into groups so:

- Connections *inside* each group are strong.
- Connections *between* groups are weak.

Spectral clustering finds where to “cut” the graph using eigenvectors instead of trying every possible cut (which would be too slow).

Eigenvalues and Eigenvectors (1/2)

For a matrix L:

$$Lv = \lambda v$$

- $v \longrightarrow$ **Eigenvector** (direction)
- $\lambda \longrightarrow$ **Eigenvalue** (stretch factor)

→ You can think of **eigenvectors** as special directions that show the main structure of data.
→ **Eigenvalues** tell how important each direction is.

Eigenvalues and Eigenvectors (2/2)

In spectral clustering:

- Small eigenvalues show the smoothest ways to divide the graph.
- The **second smallest eigenvalue** (called the *Fiedler value*) helps find the best split.
- The **Fiedler vector** tells how to divide the data into two groups.

→ If you see a big gap between eigenvalues, it usually means that clear clusters exist.

Spectral Clustering Algorithm

- **Unnormalized Algorithm**
  1. Form the **similarity matrix** $\boldsymbol{A}$
  2. Compute **degree matrix** $\boldsymbol{D}$
  3. Compute the **Laplacian** $L = D - \mathrm{A}$
  4. Find the first k eigenvectors of $\boldsymbol{L}$
  5. Combine them into a new matrix $\boldsymbol{U} = [u_1, u_2, \ldots, u_k]$
  6. Treat each row of U as a new data point
  7. Apply **K-Means** on rows of U
  8. Get your **final clusters**

Algorithm Variants (1/2)

- There are 3 main versions: all use the same idea, but differ in **which Laplacian** they use:

1. **Unnormalized (Simple):**

$$L = D - A$$

→ Works well for small, balanced graphs.

2. **Normalized Cut - Shi & Malik (2000) :**
   Solve:

$$Lv = \lambda Dv$$

→ Helps balance cluster sizes, avoids tiny groups.

v: **Eigenvector** (direction) | λ: **Eigenvalue** (stretch factor)

Algorithm Variants (2/2)

3. **Row Normalization - Ng, Jordan & Weiss (2002) :**
   After computing eigenvectors U:

$$T_{ij} = \frac{u_{ij}}{\sqrt{\sum_k u_{ik}^2}}$$

→Normalizes each row of U before K-Means for more stable clusters.

- U: $matrix\ of\ eigenvectors$
- u_{ij}: $element\ in\ row\ i, column\ j\ of\ U$
- k: $number$ clusters

Complexity

- **Time complexity:** about $O(n^3)$ for large data (due to eigenvalue calculation).
- **Space:** $O(n^2)$to store the similarity matrix.

Real-World Applications

- **Social Network Analysis:** Detecting communities (friend groups, influencers).
- **Image Segmentation:** Partitioning images into meaningful regions.
- **Bioinformatics:** Discovering groups of genes/proteins with similar functions.
- **Document Clustering:** Grouping similar texts or news articles.
- **Recommendation Systems:** Identifying clusters of similar users or products.

Conclusion

- Spectral Clustering = *Graph Theory* + *Linear Algebra* + *Clustering.*
- It doesn’t assume any shape: it works on **complex structures**.
- Uses **Laplacian eigenvectors** to find hidden clusters.
- Can handle **non-linear, graph-based data** better than K-Means.
- Complexity can be optimized.
- Real power lies in **interpreting relationships, not just distances.**

References

Luxburg07_tutorial_spectral_clustering.pdf

What is Spectral Clustering and how its work?

Spectral clustering – Wikipedia

Thank you for your attention!

Spectral Clustering

K-Means vs Spectral Clustering

Feature	K-Means	Spectral Clustering
Cluster Shape	Spherical (convex)	Any shape (Can be non-convex)
Basis	Distance	Connections
Data Type	Works best on numeric data	Works great on graphs or connected data
Speed	Fast	Slower (more math)
Output	Cluster centres	Groups based on connections

Complexity and Limitations

Spectral clustering involves **eigen-decomposition** of an $n \times n$ matrix.

- **Time complexity:**
  - $O(n^3)$ for full eigen decomposition
  - $O(k \cdot n^2)$if we only take top k eigenvectors (using sparse solvers)
- **Space complexity:**
  - $O(n^2)$for storing similarity matrix

→ Optimizations:

- Use **sparse graphs** (k-NN).
- Apply **approximate eigen-solvers** or **Nyström method** for large datasets.

Graph Cuts

Goal: Make **strong connections inside clusters** and **weak connections between clusters.**

1. **Cut Between Two Sets A and B:**

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

=total edge weight connecting A and B.

2. **Normalized Cut (Shi & Malik):**

$$\text{Ncut}(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)}$$

where $\text{vol}(A) = \sum_{i \in A} D_{ii}$

→Spectral clustering approximates this **cut minimization** problem using **eigenvectors**, which is much more efficient. In simple terms it “cuts” the graph where connections are weakest.

Graph Cuts

(A, B): $sets, clusters$
wij $Edge\ wight$
$cut(A, B)$ $Cut\ between\ A\ and\ B$
D $Degree\ matrix$
Dii $Node\ degree$
$vol(A)$ $Volume\ of\ set\ A$
$vol(B)$ $Volume\ of\ set\ B$
$Ncut(A, B)$ $Normalized\ Cut\ between\ A\ and\ B$