Demystifying Spectral Clustering (Fairness and Constraints): My Learning Project Before the IIT Guwahati Internship

Hitartha Dutta
IIT Guwahati, Summer Internship 2025

Introduction

Clustering has always been introduced as one of the most important methods in the area of unsupervised learning, wherein the main objective is to place data points into useful clusters without any assigned labels. But my knowledge about clustering improved immensely during my internship period at IIT Guwahati. There, I was exposed to spectral clustering (SC), a high-tech ML clustering technique that uses the eigenvalues and eigenvectors of a similarity matrix obtained from the data to determine clusters. This technique is significantly different from traditional clustering methods such as k-means, as it takes advantage of the graph structure of the data, enabling it to detect intricate shapes and connectivity patterns that could be overlooked by other techniques.

What enhanced the experience even further was learning about the state-of-the-art extensions of spectral clustering that resolve real-world problems. An example is fairness-constrained clustering, which adds constraints to prevent the clustering result from disproportionately harming or benefiting some groups, particularly important in applications with sensitive attributes such as gender or race. Supervision-based clustering, however, involves partial supervision or user feedback during the clustering operation, merging the advantages of unsupervised and supervised learning to generate more pertinent and precise clusters.

This hands-on project training was not only an intellectual exercise but a holistic exposure that blended the solid mathematical underpinnings with coding implementation. It got me ready to tackle clustering problems not only as standalone theoretical exercises but as instruments that have to be modified and built upon to satisfy ethical considerations and application-specific needs. The knowledge and understanding I developed at this time provided a solid foundation for my later main research, allowing me to enjoy the power and the responsibility involved in contemporary clustering methods.

In this post, I'll walk you through:

- Why spectral clustering so powerful,
- How "constrained" and "fair" variants work,
- My Python implementations,
- What I learned about clustering in the wild.

Spectral Clustering: Data Science Meets Linear Algebra

Spectral Clustering differs from standard K-means or hierarchical clustering. Rather than computing distances directly, SC models the data as a graph in which points are joined when they are similar and then finds clusters by examining the spectrum (eigenvalues/eigenvectors) of a matrix that describes this graph. In simple words, Spectral clustering thinks of the data as a network or graph, where points that are similar are linked together. Then, it studies this network using special math tools called eigenvalues and eigenvectors, which help reveal the natural groups or clusters in the data. In simple terms, spectral clustering finds clusters by looking at how points connect and relate to each other in the whole network, rather than just how close they are one-to-one. This makes it especially good at finding groups that have complex shapes or patterns.

The Basic Steps:

1. **Represent data as a graph** (affinity/similarity matrix).
2. **Build a Laplacian matrix** from the graph.
3. **Find the first k eigenvectors**—using linear algebra magic.
4. **Run K-means** in this new, revealed space.

1. Represent data as a graph (affinity/similarity matrix)

Instead of treating your data as simple rows of features, spectral clustering first turns your dataset into a **graph**:

- **Nodes:** Each data point is a node.
- **Edges:** Points are connected based on how similar they are, using a chosen *similarity* (affinity) function (e.g., Gaussian kernel).
- **Affinity Matrix (A):** A square matrix where Aij*Aij* quantifies the similarity between points i*i* and j*j*; higher values mean they're more alike. This matrix can be fully connected, or you might only connect each point to its k*k* nearest neighbors for sparsity.

2. Build a Laplacian matrix from the graph

The **graph Laplacian** is central to spectral clustering. It mathematically captures the connectivity and structure of the graph:

- **Degree Matrix (D):** Diagonal matrix where *Dii* is the sum of affinities for node i*i*.
- **Unnormalized Laplacian:** $L=D-A$.
- **Normalized Laplacian:** ($Lsym=I - D^{-1/2}AD^{-1/2}$), which often performs better when clusters vary in size or density.

The Laplacian encodes how each node is connected to others, and its mathematical properties are critical to finding good clusters.

3. Find the first k*k* eigenvectors—using linear algebra magic

The power of spectral clustering comes from **spectral graph theory**:

- Compute the smallest *k* eigenvalues and their corresponding eigenvectors of the (normalized) Laplacian. **
- These eigenvectors form a new feature space. Each data point (node) is now represented as a vector of its scores on these eigenvectors; in effect, you've embedded your data into a low-dimensional space that reveals the clusters' underlying structure.
- This "linear algebra magic" transforms complex, intertwined clusters into clearly separated groups.

**** What Are Eigenvalues and Eigenvectors?**

Given any square matrix (say, the normalized graph Laplacian L_{sym}), an **eigenvector** is a special non-zero vector *v* such that multiplying L_{sym} by *v* simply stretches (or shrinks) it by some scalar λ:

$$L_{sym}v=\lambda v$$

- *v*: the eigenvector
- λ: the corresponding eigenvalue

In spectral clustering, the "smallest" eigenvectors (those associated with the smallest eigenvalues) encode the cluster structure of your graph.

How Do You Compute Them?

- For small matrices (hundreds of nodes), use direct solvers like numpy.linalg.eigh or scipy.linalg.eigh in Python.
- python
```
import numpy as np
# Assume L_sym is your Laplacian matrix
eigenvalues, eigenvectors = np.linalg.eigh(L_sym)
```

- eigenvalues is a list of all eigenvalues.
- eigenvectors[:,i] is the eigenvector corresponding to eigenvalues[i].

- For **large or sparse matrices**, use iterative solvers that efficiently calculate only the smallest *k* eigenvalues/eigenvectors (since you rarely need all of them):
- Python:
```
from scipy.sparse.linalg import eigsh
# k: number of clusters/eigenvectors you want
evals, evecs = eigsh(L_sym, k=k, which='SM') # 'SM' means smallest
magnitude
```

My Code: Pure and Simple Spectral Clustering

I thoroughly studied Ulrike von Luxburg's 2007 tutorial on standard spectral clustering before delving into more complex subjects like fairness or constraints. This experimental code served as my point of reference for comprehending the potency of spectral clustering.

Methodology and Approach

1. Establishing Affinity/Symmetry

- Objective: Illustrate the data in graphical form.
- Methodology: Each data point constitutes a node, while the similarity between points represents the edge weight.
- Affinity (Similarity) Matrix: Employ the RBF (Gaussian) kernel for continuous data:

$$A_{ij} = exp\left(-\frac{|| xi - xj||^2}{2\sigma^2}\right)$$

where σ is a bandwidth parameter (can use average nearest neighbor distance).

2. Constructing the Laplacian

- **Degree Matrix *D*:** Diagonal—sum of affinities per node.

```
D = np.diag(np.sum(A, axis=1))
```

- Laplacians:
  Unnormalized: $L=D-A$
  Symmetric Normalized: $L_{sym}=I- D^{-1/2}AD^{-1/2}$

3. Eigen Decomposition

- Compute the smallest *k* eigenvectors of the chosen Laplacian (*k*= expected number of clusters).
- The structure of these eigenvectors reveals natural cluster boundaries—nodes in the same cluster have similar vector entries.

4. Spectral Embedding & Clustering

- Form a matrix from the k*k* eigenvectors (nodes × eigenvectors).
- Row normalization (for Lsym*Lsym*): scale each row to have unit norm—for better separation.
- Run K-means on these vectors.

5. Evaluation

- Adjusted Rand Index (ARI): Compares true cluster labels with clustering result, robust to permutation.

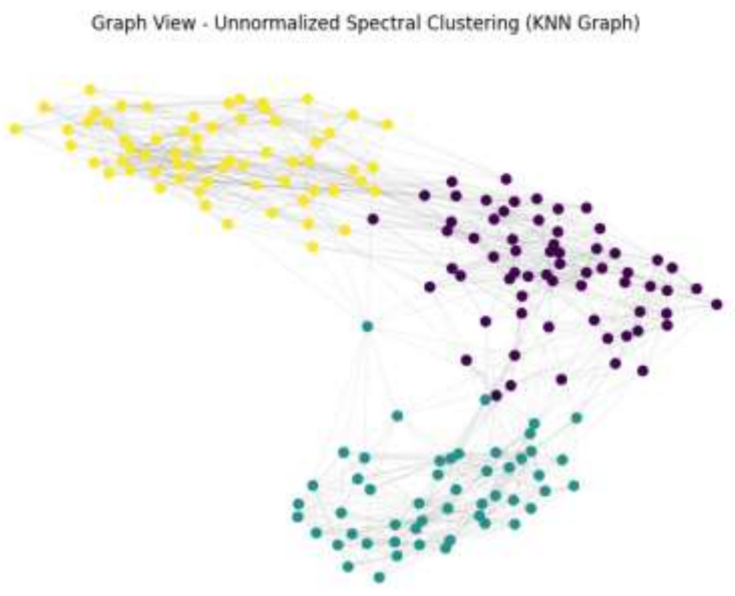
- Cut value: The sum of edge weights between clusters—a smaller cut implies cleaner separation.

6. Visualization

- Scatter plots (data space): Points colored by true and predicted cluster.
- Spectral embedding visualization: Plot first two/three eigenvector components to see cluster separation in the embedded space.
- Eigenvector heatmaps: Show structure and "smoothness" over the graph.

7. Results:

One of the result graphs is shown for reference example below:



*** TO VIEW AND ACCESS THE IMPLEMENTATION CODE REFER TO THE LINK OF THE GITHUB REPOSITORY ATTACHED IN THE 1st PAGE OF THIS DOCUMENT*

Final Thoughts:

This foundational spectral clustering implementation gave me:

- **Solid intuition** on how "graph thinking" and linear algebra combine for powerful clustering,
- The tools to compare and visualize cluster quality,
- A strong springboard to implement fairness or constraint-based extensions.

Once you've "seen the magic" of spectral clustering on basic datasets, you'll trust why the method is such a flexible, extendable, and effective tool.

The Next Level: Fair and Constrained Spectral Clustering

Real-world data analysis is rarely perfect. Organizations increasingly demand that clustering outcomes not just follow raw data structure, but also **honor fairness criteria** or **incorporate external ("side") information**. Spectral clustering naturally extends to these settings—thanks to modern advancements in constraint and fairness-aware algorithms.

1. Fair Spectral Clustering

The Motivation

In scenarios like social network analysis, education, or policy research, we often care not just about natural groups, but also about **fair representation of sensitive attributes** (e.g., gender, ethnicity, socio-economic status). Plain clustering may split groups in biased ways, reinforcing existing imbalances in the data.

Algorithmic Approach — Based on Kleindessner et al. (2019)

- **Goal:**
  Ensure that every cluster approximately preserves the demographic proportions found in the overall dataset (e.g., each cluster is roughly 50% male and 50% female if the dataset is).
- **Key Challenge:**
  Standard spectral clustering does *not* guarantee such proportionality: it seeks "natural" cuts in the graph, which may align with demographic boundaries.
- **Solution Steps:**
  1. **Encode Demographic Group Membership**
     Construct group membership vectors $f^{(s)}$ for each group s, indicating group membership of each node.
  2. **Construct Fairness Constraint Matrix F:**
     Each column is $f^{(s)} -$ *(proportion of group s)* $\times 1$*, for* $s = 1 \ldots (h-1)$ where h is number of groups.
  3. **Find Nullspace Z of F^T:**
     This matrix projects any embedding into the space orthogonal to unfair partitions.
  4. **Project Laplacian:**
     Solve the spectral problem not on the whole Laplacian L, but in the "fair subspace":

     $$Z^T LZu = \lambda u$$

     For normalized Laplacian, this is extended further (see Algorithm 3 in Kleindessner et al.).

5. **Spectral Embedding & Clustering:**
   Map the original nodes back using *H=Zu*. Apply row-normalization (if normalized Laplacian). Run K-means in this space.

- **Result:**
  The clusters obtained are as fair as possible with respect to group proportions, and often, surprising empirical findings show that "fairness" can be achieved with minimal sacrifice in cut cost or overall cluster compactness.

Practical Example

On real friendship network data (nodes: students, edges: friendships, metadata: gender), the fair spectral clustering algorithm produced clusters with **significantly higher gender-balance**—without drastically increasing the edge cuts between clusters.

2. Flexible Constrained Spectral Clustering

Motivation

Classic constrained clustering supports so-called **Must-Link (ML)** and **Cannot-Link (CL)** constraints:

- **ML:** "Points A and B must be in the same cluster."
- **CL:** "Points X and Y must not be in the same cluster."

However, many real problems need more nuance:

- **"Soft" constraints:** "These points are *likely* together," or "preferably keep them apart."
- **Incomplete or noisy supervision:** Not all constraints can or should be strictly enforced.

Algorithmic Approach — Wang & Davidson (2010)

- **Goal:**
  Allow constraints of varying strength, encoded as a real-valued matrix (not only +1, -1).
  Users can control the "importance" of side information versus natural graph structure.
- **Representation:**
  - **Constraint Matrix QQQ:**
    - $Q_{ij} > 0$: prefer ML (strength = value)
    - $Q_{ij} < 0$: prefer CL (strength = |value|)
    - $Q_{ij} = 0$: no constraint/unknown
- **Flexible Algorithm (CSP):**
  1. **Normalize Laplacian and Constraints:**

$$\bar{L} = D^{-1/2}LD^{-1/2}, \bar{Q} = D^{-1/2}QD^{-1/2}$$

2. **Trade-off Parameter (β or α):**
   User sets a threshold for required constraint satisfaction.
3. **Generalized Eigenvalue Problem:**
   Solve:

$$\bar{L}v = \lambda(\bar{Q} - \frac{\beta}{vol(G)} I)v$$

   or, in some settings:

$$L_{mod} = \bar{L} - \alpha\bar{Q}$$

   The smallest positive eigenvectors are your embedding.

4. **Spectral Embedding & K-means:** As before, project to low-d space; cluster.

- **Variations:**
  - **ModAff:** Enforce constraints by modifying affinity matrix weights (“hard” enforcement: ML = 1.0, CL = 0.0).
  - **GrBias:** Only Must-Link, “boost” those
  - **Flexible CSP:** Soft or degree-based, user-tunable—can recover previous methods as special cases.

Practical Example

On Iris and Ionosphere datasets:

- **Process:**
  - Some labels are known; pairwise constraint matrix created accordingly (based on known label agreement/disagreement).
  - Compared several algorithms (CSP, ModAff, GrBias), visualized the quality of clustering via the Rand Index.
- **Findings:**
  - **Flexible CSP** consistently delivered stable and improved clustering as more constraints were added.
  - Could gracefully handle noisy or inconsistent constraints by adjusting the constraint strength parameter.

Summary Table: Algorithmic Variations

Algorithm	Constraint Type	How Constraints Are Used	Clustering Step
Plain SC	None	-	Eigenvectors of Laplacian
Fair SC (Kleindessner)	Group prop.	Linear fairness constraints via nullspace	Projected Laplacian, K-means
ModAff	ML/CL (hard)	Alters affinity directly (ML=1, CL=0)	SC on modified Laplacian

Algorithm	Constraint Type	How Constraints Are Used	Clustering Step
GrBias	ML (boosted)	Boosts ML affinities	SC on modified Laplacian
Flexible CSP (Wang)	Real-valued/soft	Degree-of-belief, tunable via parameter	Generalized eigensolver, K-means

Key Takeaways

- **Fair Spectral Clustering** enforces proportional group representation with mathematically principled constraints.
- **Flexible Constrained Spectral Clustering** generalizes ML/CL to soft, weighted supervision and allows user control over "faithfulness" to constraints versus natural structure.
- Both algorithms use advanced linear algebra (nullspace projection, generalized eigendecomposition) but **retain the core advantage**: transforming complex, irregular data into a space where simple clustering (K-means) is remarkably effective.

This progression from standard to fairness- and constraint-aware spectral clustering gives practitioners tools to build not just accurate but also responsible and adaptable clustering models for real-world datasets.

The Learning Journey: Implementation, Challenges, and Insights

This spectral clustering training project was an eye-opening experience, transforming theoretical concepts into practical, working code and preparing me for the rigors of real research. Below, I share the key highlights of my implementation process along with the challenges I faced, how I addressed them, and the insights gained.

Key Implementation Highlights

- **Affinity Matrix and Graph Representation:**
  Building the foundation, I learned how to represent datasets as graphs using the Radial Basis Function (RBF) kernel to convert feature vectors into meaningful similarity measures. This step is critical: a well-chosen affinity metric can make or break clustering quality. Constructing the degree matrix and carefully assembling the graph Laplacian deepened my understanding of graph-based learning principles.
- **Eigen-Decomposition and Numerical Stability:**
  Computing eigenvalues and eigenvectors of large Laplacian matrices is challenging both conceptually and computationally. I explored different eigensolvers (numpy.linalg.eigh for dense, scipy.sparse.linalg.eigsh for sparse matrices) and learned the importance of normalization to ensure stable and interpretable spectral embeddings. Handling numerical precision issues—such as tiny non-zero values due to floating-point error—taught me attentiveness to the intricacies of linear algebra in practice.

- **Encoding Constraints:**
  Moving beyond vanilla spectral clustering, I tackled the task of incorporating constraints. Generating constraint matrices from given labels — both hard (must-link, cannot-link) and soft (degree-of-belief) — involved rigorously designing data structures and workflow pipelines to reflect real-world supervision. This step required meticulous alignment between theoretical formulations and engineering implementation.
- **Evaluation and Visualization:**
  I employed quantitative metrics like Adjusted Rand Index (ARI) and Rand Index to benchmark clustering outcomes. Beyond numbers, visualizing embeddings and cluster assignments in 2D plots revealed the nuanced impact of spectral transformation and constraints on data geometry, helping me gain an intuitive feel for the algorithms' behavior.

Challenges Faced and How I Overcame Them

- **Understanding Graph Laplacians and Their Variants:**
  The mathematical definitions of unnormalized versus normalized Laplacians, and their subtle differences in spectral properties, initially caused confusion. I overcame this by studying multiple sources, plotting sample matrices, and implementing both versions side-by-side to compare outcomes. This hands-on experimentation clarified the conditions under which each variant excels.

- **Scalability of Eigen-Decomposition:**
  As dataset sizes grew, computing full eigendecompositions became prohibitively slow. Utilizing sparse matrix techniques and iterative solvers (like ARPACK via scipy.sparse.linalg.eigsh) allowed practical computations. I also learned careful parameter tuning to select the minimal but sufficient number of eigenvectors for effective clustering.

- **Integrating Fairness Constraints Without Breaking Clustering Quality:**
  Adding linear constraints for fairness required modifying the spectral problem to restrict the embedding space. Initially, my code failed to converge or produced trivial solutions. By deeply understanding the nullspace projection and carefully implementing matrix operations—supported by small synthetic tests—I succeeded in achieving balanced clusters without sacrificing much cut quality.

- **Handling Noisy and Incomplete Constraints:**
  Real-world constraints are often inconsistent. Implementing flexible, degree-of-belief constraints (instead of strict binary ones) required designing generalized eigenvalue problems and new evaluation criteria. Iterative debugging and validation on benchmark datasets helped me develop a robust algorithm resilient to noisy supervision.

- **Balancing Code Modularity and Efficiency:**
  Writing modular, reusable code while ensuring computational efficiency was a recurring challenge. Refactoring functions for affinity construction, eigen-decomposition, and constraint incorporation while maintaining clarity improved long-term maintainability.

Insights Gained

- **Visualization is Key**
  Watching how spectral embedding spreads data points in low-dimensional space was a revelation. It vividly demonstrated why SC can separate clusters invisible to algorithms like K-means operating on original features.
- **Fairness and Constraints Matter**
  Even limited supervision or proportional representation constraints significantly improved cluster interpretability and social fairness. This intersection of machine learning with ethical practice is both vital and rewarding.
- **Theory and Code Synergy**
  Walking through mathematical proofs and translating them into code gave me confidence and insight: theory without implementation lacks substance; implementation without understanding is fragile.
- **Explicit Trade-offs**
  Every clustering decision involves trade-offs—between purity and fairness, simplicity and accuracy, or autonomy and supervision. These trade-offs clarify that machine learning is as much an art informed by context as a science powered by algorithms.

Final Thoughts: More than Just a Training Project

This spectral clustering foundational work was far more than a simple internship training exercise. It became the bedrock for my research at IIT Guwahati, enabling me to confidently extend classical methods to fairness- and constraint-aware clustering in complex real-world problems.

For budding data scientists and researchers diving into unsupervised learning, I emphasize:

- Explore spectral methods beyond K-means—they reveal rich geometric structure.
- Implement and visualize repeatedly—trust your eyes, not just formulas.
- Engage deeply with constraints and fairness—machine learning is becoming a catalyst for responsible technology.

This journey deepened my curiosity and sharpened my skills, setting me up for future challenges in AI research and practice.