Vietnam National University, HCMC

International University

School of Computer Science & Engineering

Final Project

Beyond Euclidean Distance: A Theoretical Analysis and Implementation of Spectral Clustering using Unnormalized Graph Laplacians

Course: Artificial Intelligence

Student:

Tran Hoang Khiem - ITCSIU23016

December 16, 2025

Contents

Listings

List of Figures

1 Introduction

1.1 Overview

Clustering is a cornerstone technique in unsupervised machine learning and exploratory data analysis, designed to organize unlabelled data into meaningful subgroups based on similarity. Ideally, points within the same cluster should be highly similar, while points in different clusters should be distinct. Traditional algorithms, most notably K-Means, approach this problem by measuring the geometric distance between points and a central prototype (centroid). Due to its linear computational complexity and ease of interpretation, K-Means remains the default choice for many applications.

However, the reliance on Euclidean distance imposes strict geometric assumptions that often fail in complex, real-world scenarios. K-Means implicitly assumes that clusters are spherical, equally sized, and linearly separable. When data forms complex, non-linear manifolds—such as interlocking rings, spirals, or elongated shapes—distance-based metrics fail to capture the true underlying structure.

Spectral Clustering addresses these limitations by shifting the paradigm from "spatial proximity" to "graph connectivity." Instead of looking at the absolute position of data points, it analyzes the spectrum (eigenvalues and eigenvectors) of a similarity graph constructed from the data. By leveraging concepts from spectral graph theory, this method transforms the difficult problem of partitioning non-linear data in high-dimensional space into a simpler clustering problem in a lower-dimensional embedding space. This approach has proven highly effective in fields ranging from computer vision (image segmentation) to biology (gene expression analysis).

1.2 Problem Statement

The central problem addressed in this project is the inability of centroid-based algorithms to correctly partition non-convex data structures. Algorithms like K-Means operate on the assumption that clusters form convex hulls. Consequently, when presented with non-convex geometries—such as a "C" shape surrounding a dense blob—K-Means will incorrectly slice through the natural boundaries of the data to minimize variance, destroying the semantic meaning of the clusters.

Spectral Clustering offers a mathematically robust solution to this convexity problem by transforming the data into a spectral embedding where these complex manifolds become linearly separable. However, the field of spectral clustering is vast, encompassing various graph Laplacians (unnormalized, symmetric normalized, random-walk normalized) [2] and corresponding graph cut objectives (RatioCut, Normalized Cut).

Scope Limitation: Theoretical literature, such as the work by [4], suggests that normalized Laplacians often provide better stability for graphs with irregular degree distributions. However, a comprehensive implementation of all variations requires an extensive theoretical background that exceeds the time constraints of this project. Therefore, to ensure a rigorous and mathematically sound implementation, this project explicitly focuses on the **Unnormalized Graph Laplacian** and its relationship to the **RatioCut** objective. By restricting our scope, we aim to achieve a complete, first-principles understanding of the core spectral mechanisms—specifically the use of the Fiedler vector for graph partitioning—rather than providing a superficial comparison of every available variant.

1.3 Scope and Objectives

This project aims to build an intelligent system that implements Spectral Clustering from scratch and evaluates its performance against standard K-Means on both synthetic and real-world data. The project is guided by the following objectives:

- **Objective 1:** Establish the theoretical foundations of Spectral Clustering. Inspired by [4], we aim to extend that work by providing a highly accessible mathematical derivation. We specifically focus on the ”beautiful idea” of using Lagrange multipliers to intuitively derive the final optimization form ($Lf = \lambda f$) without relying on overly complex graph-cut theories.

- **Objective 2:** Implement a functional Spectral Clustering algorithm from scratch using the Unnormalized Laplacian ($L = D - W$), capable of constructing similarity graphs and performing eigen-decomposition.

- **Objective 3:** Conduct a comparative analysis of Spectral Clustering versus K-Means on synthetic non-convex datasets (Moons, Circles) and high-dimensional image data (MNIST) to empirically demonstrate the trade-off between geometric flexibility and computational cost.

2 Related Work

The problem of data clustering has been studied extensively, evolving from simple distance-based heuristics to sophisticated graph-theoretic approaches. This section reviews the progression of these techniques and establishes the theoretical context for Spectral Clustering.

2.1 Traditional Distance-Based Clustering

The most ubiquitous algorithm in clustering is K-Means, widely used due to its computational efficiency ($O(N)$) and ease of implementation. K-Means operates by partitioning data into Voronoi cells, assigning points to the nearest centroid based on Euclidean distance.

However, K-Means rests on strong geometric assumptions: it assumes that clusters are convex and linearly separable. This limits its effectiveness on complex manifolds. As noted in the literature, K-Means fails to detect concentric circles, spirals, or interlocking shapes because it prioritizes compactness over connectivity. This limitation drives the need for approaches that model the *structural relationship* between points rather than just their spatial proximity.

2.2 Graph Partitioning Approaches

To overcome the convexity limitation, clustering can be reframed as a Graph Partitioning problem. Here, data points are vertices in a graph $G = (V, E)$, and edges represent similarity.

2.2.1 The Minimum Cut (MinCut)

One of the earliest graph-based methods is the Minimum Cut, which partitions a graph by minimizing the sum of weights of edges crossing the boundary between clusters. While theoretically sound, Wu and Leahy [5] demonstrated that MinCut is practically flawed: it favors cutting ”isolated” vertices with low degrees rather than bisecting the graph into meaningful groups. This tendency results in highly unbalanced clusters, often consisting of a single outlier point and the rest of the dataset.

2.2.2 Balanced Cuts: RatioCut and Normalized Cut

To rectify the bias of MinCut, objective functions were introduced to enforce balanced partition sizes:

- **RatioCut:** Proposed by Hagen and Kahng [1], this method scales the cut cost by the number of vertices in each cluster ($|A_i|$), penalizing tiny clusters.
- **Normalized Cut (Ncut):** Shi and Malik [3] proposed scaling by the total *volume* of edges ($vol(A_i)$), which accounts for widely varying vertex degrees.

While these balanced cut formulations solve the outlier problem, minimizing them exactly is NP-Hard due to the discrete combinatorial nature of the search space (2^n possible partitions).

2.3 Spectral Relaxation

Spectral Clustering emerges as the computational solution to this NP-Hardness. By relaxing the discrete cluster indicator vector into a continuous real-valued vector, the problem transforms into a standard eigenvalue problem (finding eigenvectors of the Graph Laplacian). This allows us to find a globally optimal solution to the relaxed problem using standard linear algebra, which serves as an excellent approximation for the discrete graph partition.

3 Methodology

3.1 Data Representation

Before discussing spectral clustering, we first establish the fundamental graph definitions. A graph $G = (V, E)$ consists of a set of vertices V and edges E. The connectivity of these vertices is described by an Adjacency Matrix.

3.1.1 Standard Degree (Unweighted Graph)

In a standard unweighted graph, the adjacency matrix A is binary. If there is a connection between vertex i and vertex j, then $A_{ij} = 1$, otherwise $A_{ij} = 0$.
The **Degree** (d_i) of a vertex v_i is simply the count of edges connected to it. This is calculated as the sum of the row in the adjacency matrix:

$$d_i = \sum_{j=1}^{n} A_{ij}$$

The **Degree Matrix** D is a diagonal matrix where $D_{ii} = d_i$.

Example: Simple graph with 6 nodes.

Consider the specific graph structure shown in Figure 1:

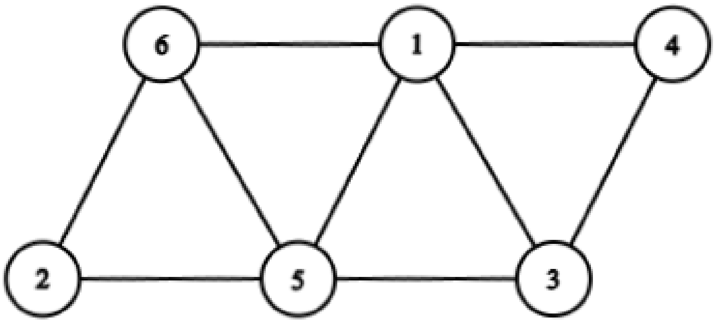

Figure 1: An unweighted graph with 6 nodes.

Based on the connections in this graph, the Adjacency Matrix A and the corresponding Degree Matrix D are defined as follows:

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Here, Node 1 is connected to 4 other nodes, so $D_{11} = 4$. Node 2 is connected to only 2 nodes, so $D_{22} = 2$.

3.1.2 Weighted Similarity Graph

For clustering tasks, we extend this concept to model data relationships using continuous weights rather than binary connections. Given a dataset $X = \{x_1, x_2, ..., x_n\}$ with n data points, we represent the data as a weighted undirected graph $G = (V, E)$, where each vertex v_i represents a data point x_i.

We define the "closeness" between two points x_i and x_j using the Gaussian Kernel (RBF) similarity function. This function ensures that nearby points have high similarity (close to 1) and distant points have low similarity (close to 0):

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

Where:

- $||x_i - x_j||^2$ is the squared Euclidean distance.
- σ is a scaling parameter that controls the width of the neighborhood.

To ensure the graph captures local structures (e.g., following the curve of a spiral), we do not connect every point to every other point. Instead, we use a k-Nearest Neighbors (kNN) approach. The weighted adjacency matrix $W = (w_{ij})$ is defined as follows:

$$w_{ij} = \begin{cases} s_{ij} \text{ if } x_j \in \text{kNN}(x_i) \\ 0 \quad \text{otherwise} \end{cases}$$

The degree d_i of a vertex v_i is now the sum of the **weights** of all edges connected to it. The Degree Matrix D remains a diagonal matrix where the diagonal elements are:

$$d_{ii} = \sum_{j=1}^{n} w_{ij}$$

Example: Simple weighted graph.

Figure 2 illustrates this concept using a synthetic 2D dataset. Unlike the unweighted graph where edges are simply present or absent, the edges here visually represent the magnitude of the similarity weight w_{ij}.

The visual intensity of the connections corresponds to the Gaussian Kernel output:

- **Thick, dark lines** indicate points that are spatially very close (small Euclidean distance), resulting in a weight w_{ij} close to 1.
- **Thin, faint lines** indicate neighbors that are further apart but still within the k-nearest range. These have a lower weight (closer to 0).

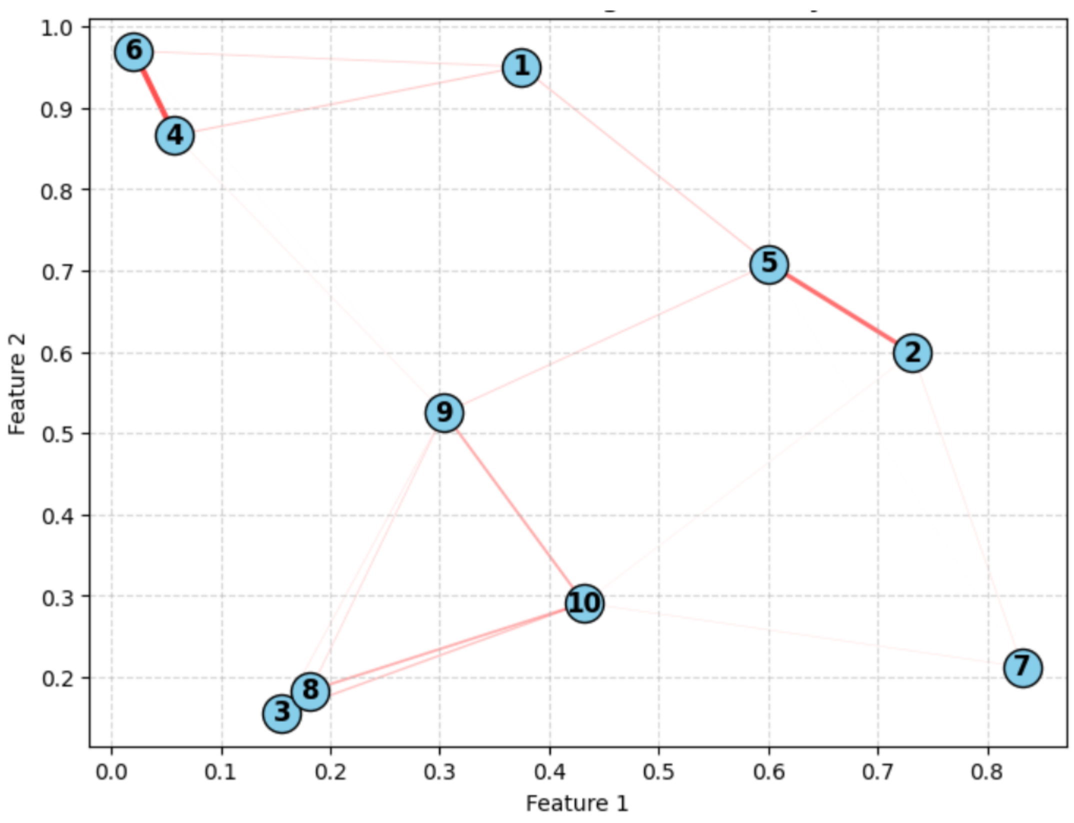

Figure 2: Weighted Similarity Graph with $k = 3$.

3.2 The Graph Laplacian

The unnormalized graph Laplacian L is defined as the difference between the degree matrix and the adjacency matrix:

$$L = D - W.$$

A fundamental property of the Laplacian is its associated quadratic form. This identity is crucial because it relates the algebraic spectrum of the matrix to the geometric "smoothness" of a vector over the graph. For any vector $f \in \mathbb{R}^n$, we derive the relationship as follows:

$$\begin{aligned} f^T L f &= f^T (D - W) f \\ &= f^T D f - f^T W f \\ &= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} w_{ij} f_i f_j. \end{aligned}$$

Substituting the definition of degree $d_i = \sum_{j=1}^{n} w_{ij}$, the first term becomes $\sum_{i,j=1}^{n} w_{ij} f_i^2$. Since the graph is undirected ($w_{ij} = w_{ji}$), we can symmetrize the expression:

$$\begin{aligned} f^T L f &= \frac{1}{2} \left(\sum_{i,j=1}^{n} w_{ij} f_i^2 - 2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i,j=1}^{n} w_{ij} f_j^2 \right) \\ &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i^2 - 2 f_i f_j + f_j^2) \\ &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2. \end{aligned}$$

This result shows that minimizing $f^T L f$ is equivalent to forcing connected nodes (where w_{ij} is large) to have similar values ($f_i \approx f_j$).

To build intuition, we can model the graph as a physical system of masses connected by springs.

- **Nodes as Masses:** Each data point x_i is a mass constrained to move on a line. Its position is given by the value f_i.
- **Edges as Springs:** An edge with weight w_{ij} acts as a spring connecting mass i and mass j with stiffness $k = w_{ij}$.

From Hooke's Law, the potential energy of a spring stretched between f_i and f_j is $\frac{1}{2} k (distance)^2 = \frac{1}{2} w_{ij} (f_i - f_j)^2$. Therefore, the Laplacian quadratic form $f^T L f$ represents the **total potential en-**

ergy of the system.

Minimizing this energy (finding the eigenvalues) corresponds to finding the "vibration modes" of the system. The lowest energy state ($\lambda_1 = 0$) occurs when all springs are relaxed ($f_i = f_j$ for all i, j), corresponding to the constant vector. The next lowest state (λ_2) corresponds to the fundamental frequency, where the graph splits into two heavy masses moving in opposite directions, stretching the springs between them as little as possible.

3.3 Dimensionality Reduction (Spectral Embedding)

While the Laplacian matrix L captures the connectivity of the graph, our ultimate goal is to find a representation of the data that makes clustering trivial. This process is called Spectral Embedding. It transforms the data from the high-dimensional input space (where clusters are complex and non-linear) to a lower-dimensional space where clusters become linearly separable.

This transformation is derived directly from the graph partition problem known as the Minimum Cut. Our goal is to separate the data into k clusters such that the edges between clusters are weak (dissimilar points) and the edges within clusters are strong (similar points).

3.3.1 The Minimum Cut Problem

The most direct way to partition a graph is to solve the MinCut problem. For a partition of the graph into k disjoint sets $A_1, \ldots, A_k$, the "cut" is defined as the sum of the weights of edges connecting different sets:

$$\mathbf{cut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i=1}^{k} W(A_i, \bar{A}_i).$$

where $W(A, B) = \sum_{i \in A, j \in B} w_{ij}$.

However, minimizing this objective directly often fails in practice. The solution tends to separate individual isolated vertices from the rest of the graph rather than finding meaningful clusters, as cutting a single vertex requires cutting very few edges.

3.3.2 Balanced Partitions (RatioCut)

To avoid trivial solutions, we must impose a constraint that the clusters be "balanced" in size. We introduce the RatioCut objective, which explicitly penalizes small clusters by scaling the cut cost by the inverse of the cluster size $|A_i|$:

$$\textbf{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} = \sum_{i=1}^{k} \frac{\textbf{cut}(A_i, \bar{A}_i)}{|A_i|}$$

Minimizing this function ensures that all clusters A_i are reasonably large. However, introducing this balancing constraint makes the problem NP-Hard. The combinatorial nature of selecting discrete partitions implies a search space of 2^n, which is impossible to solve exactly for large datasets. Instead, we perform a spectral relaxation.

3.3.3 Approximating RatioCut for $k = 2$

We define a discrete cluster indicator vector $f \in \mathbb{R}^n$ with specific values designed to balance the cluster sizes:

$$f_i = \begin{cases} \sqrt{\frac{|\bar{A}|}{|A|}} & \text{if } v_i \in A \\ -\sqrt{\frac{|A|}{|\bar{A}|}} & \text{if } v_i \in \bar{A} \end{cases}$$

We start with the Laplacian quadratic form derived in Section 3.2:

$$f^T L f = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2.$$

We observe that if two vertices v_i and v_j belong to the same cluster (both in A or both in $\bar{A}$), then $f_i = f_j$, and the term $(f_i - f_j)^2$ is zero. The sum therefore only includes edges connecting A to $\bar{A}$:

$$f^T L f = \frac{1}{2}\sum_{i \in A, j \in \bar{A}} w_{ij}(f_i - f_j)^2 + \frac{1}{2}\sum_{i \in \bar{A}, j \in A} w_{ij}(f_i - f_j)^2.$$

Substituting the values of f_i and f_j:

$$\begin{aligned}
(f_i - f_j)^2 &= \left(\sqrt{\frac{|\bar{A}|}{|A|}} - \left(-\sqrt{\frac{|A|}{|\bar{A}|}}\right)\right)^2 \\
&= \left(\sqrt{\frac{|\bar{A}|}{|A|}} + \sqrt{\frac{|A|}{|\bar{A}|}}\right)^2 \\
&= \frac{|\bar{A}|}{|A|} + 2 + \frac{|A|}{|\bar{A}|} \\
&= \frac{|\bar{A}| + |A|}{|A|} + \frac{|A| + |\bar{A}|}{|\bar{A}|} \quad (\text{using } |\bar{A}| + |A| = n) \\
&= \frac{n}{|A|} + \frac{n}{|\bar{A}|}.
\end{aligned}$$

Substituting this constant back into the sum:

$$\begin{aligned}
f^T L f &= \frac{1}{2} \sum_{i \in A, j \in \bar{A}} w_{ij} \left(\frac{n}{|A|} + \frac{n}{|\bar{A}|}\right) + \frac{1}{2} \sum_{i \in \bar{A}, j \in A} w_{ij} \left(\frac{n}{|A|} + \frac{n}{|\bar{A}|}\right) \\
&= \left(\frac{n}{|A|} + \frac{n}{|\bar{A}|}\right) \left(\frac{1}{2} \sum_{i \in A, j \in \bar{A}} w_{ij} + \frac{1}{2} \sum_{i \in \bar{A}, j \in A} w_{ij}\right) \\
&= n \left(\frac{1}{|A|} + \frac{1}{|\bar{A}|}\right) \mathbf{cut}(A, \bar{A}) \\
&= n \cdot \mathbf{RatioCut}(A, \bar{A}).
\end{aligned}$$

Next, we verify that this specific vector f satisfies two important geometric properties.

Property A: Orthogonality to the constant vector (Sum is 0)

$$\begin{aligned}
\sum_{i=1}^{n} f_i &= \sum_{i \in A} f_i + \sum_{i \in \bar{A}} f_i \\
&= \sum_{i \in A} \sqrt{\frac{|\bar{A}|}{|A|}} + \sum_{i \in \bar{A}} \left(-\sqrt{\frac{|A|}{|\bar{A}|}}\right) \\
&= |A|\sqrt{\frac{|\bar{A}|}{|A|}} - |\bar{A}|\sqrt{\frac{|A|}{|\bar{A}|}} \\
&= \sqrt{|A| \cdot |\bar{A}|} - \sqrt{|\bar{A}| \cdot |A|} \\
&= 0.
\end{aligned}$$

This implies $f \perp \mathbf{1}$.

Property B: Norm of the vector

$$\begin{aligned}
\|f\|^2 &= \sum_{i=1}^{n} f_i^2 \\
&= \sum_{i \in A} \left(\sqrt{\frac{|\bar{A}|}{|A|}} \right)^2 + \sum_{i \in \bar{A}} \left(-\sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 \\
&= |A| \frac{|\bar{A}|}{|A|} + |\bar{A}| \frac{|A|}{|\bar{A}|} \\
&= |\bar{A}| + |A| \\
&= n.
\end{aligned}$$

The discrete optimization problem is thus:

$$\min_{A \subset V} f^T L f \quad \text{subject to } f \perp \mathbf{1},\ \|f\|^2 = n,\ f \text{ is discrete.}$$

Since finding the discrete partition is NP-Hard, we **relax** the discreteness constraint and allow f to take arbitrary real values. The relaxed problem is:

$$\min_{f \in \mathbb{R}^n} f^T L f \quad \text{subject to } f \perp \mathbf{1},\ \|f\|^2 = n.$$

We solve this using the method of **Lagrange Multipliers**. We define the Lagrangian function $\mathcal{L}$ with a multiplier λ:

$$\mathcal{L}(f, \lambda) = f^T L f - \lambda (f^T f - n).$$

To find the stationary points, we take the derivative with respect to f and set it to zero:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial f} &= \frac{\partial}{\partial f}(f^T L f) - \lambda \frac{\partial}{\partial f}(f^T f - n) \\
&= 2Lf - 2\lambda f = 0.
\end{aligned}$$

Rearranging terms yields the standard eigenvalue equation:

$$Lf = \lambda f.$$

This implies that the optimal f must be an eigenvector of L. The value of the objective function at this solution is:

$$f^T L f = f^T(\lambda f) = \lambda(f^T f) = \lambda \cdot n.$$

To minimize the objective, we must choose the smallest possible eigenvalue λ.

- The smallest eigenvalue $\lambda_1 = 0$ corresponds to the constant eigenvector $\mathbf{1}$. However, this is forbidden by the constraint $f \perp \mathbf{1}$.
- Therefore, the optimal solution is the eigenvector corresponding to the **second smallest eigenvalue** λ_2.

3.3.4 Approximating RatioCut for Arbitrary k

For the general case where we wish to partition the data into $k > 2$ clusters $A_1, \dots, A_k$, we cannot use a single indicator vector. Instead, we define a set of k indicator vectors $h_1, \dots, h_k$, where the vector $h_j \in \mathbb{R}^n$ represents the j-th cluster. The entries are defined as:

$$h_{i,j} = \begin{cases} \frac{1}{\sqrt{|A_j|}} & \text{if } v_i \in A_j \\ 0 & \text{otherwise} \end{cases}$$

We assemble these vectors into an indicator matrix $H \in \mathbb{R}^{n \times k}$ such that the j-th column is h_j:

$$H = [h_1, h_2, \dots, h_k].$$

We first verify that the columns of H are orthonormal (i.e., $H^T H = I$). Let us calculate the dot product between two columns h_p and h_q:

$$h_p^T h_q = \sum_{i=1}^{n} h_{i,p} h_{i,q}.$$

- **Case 1: $p \neq q$.** Since the clusters are disjoint ($A_p \cap A_q = \emptyset$), a vertex v_i cannot belong to both A_p and A_q. Therefore, for any index i, at least one of $h_{i,p}$ or $h_{i,q}$ must be 0. The product is always 0.

- **Case 2:** $p = q$**.** We sum the squared entries for vertices in A_p:

$$h_p^T h_p = \sum_{i \in A_p} \left(\frac{1}{\sqrt{|A_p|}} \right)^2 = \sum_{i \in A_p} \frac{1}{|A_p|} = |A_p| \cdot \frac{1}{|A_p|} = 1.$$

Thus, the columns are orthonormal, satisfying the constraint $H^T H = I$.

Next, we relate the RatioCut to the Laplacian quadratic form. For a specific cluster vector h_j, we have:

$$h_j^T L h_j = \frac{1}{2} \sum_{p,q=1}^{n} w_{pq} (h_{p,j} - h_{q,j})^2.$$

The term $(h_{p,j} - h_{q,j})^2$ is non-zero only if one vertex is in A_j and the other is not (i.e., an edge in the cut). Specifically, if $v_p \in A_j$ and $v_q \notin A_j$:

$$(h_{p,j} - h_{q,j})^2 = \left(\frac{1}{\sqrt{|A_j|}} - 0 \right)^2 = \frac{1}{|A_j|}.$$

Summing over all edges in the cut:

$$h_j^T L h_j = \sum_{(p,q) \in \mathbf{cut}(A_j, \bar{A}_j)} w_{pq} \cdot \frac{1}{|A_j|} = \frac{\mathbf{cut}(A_j, \bar{A}_j)}{|A_j|}.$$

The total RatioCut is the sum over all k clusters:

$$\mathbf{RatioCut}(A_1, \ldots, A_k) = \sum_{j=1}^{k} h_j^T L h_j.$$

Using the properties of the trace operator, where $\mathbf{Tr}(A) = \sum a_{ii}$, we can rewrite this sum as the trace of the matrix product:

$$\sum_{j=1}^{k} h_j^T L h_j = \sum_{j=1}^{k} (H^T L H)_{jj} = \mathbf{Tr}(H^T L H).$$

The discrete optimization problem is:

$$\min_{A_1, \ldots, A_k} \mathbf{Tr}(H^T L H) \quad \text{subject to } H^T H = I, \; H \text{ discrete}.$$

Since the discrete constraint is NP-Hard, we relax it to allow H to be any real-valued matrix with orthonormal columns:

$$\min_{H \in \mathbb{R}^{n \times k}} \mathbf{Tr}(H^T L H) \quad \text{subject to } H^T H = I.$$

We solve this using **Matrix Lagrange Multipliers**. We construct the Lagrangian function $\mathcal{L}$, introducing a symmetric matrix of multipliers $\Lambda \in \mathbb{R}^{k \times k}$ to enforce the constraint $H^T H - I = 0$:

$$\mathcal{L}(H, \Lambda) = \mathbf{Tr}(H^T L H) - \mathbf{Tr}(\Lambda(H^T H - I)).$$

We take the derivative with respect to the matrix H.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial H} &= \frac{\partial}{\partial H}\mathbf{Tr}(H^T L H) - \frac{\partial}{\partial H}\mathbf{Tr}(\Lambda H^T H) \\ &= (L + L^T)H - H(\Lambda + \Lambda^T). \end{aligned}$$

Since L is symmetric ($L = L^T$) and we can choose the multiplier matrix Λ to be symmetric ($\Lambda = \Lambda^T$), this simplifies to:

$$2LH - 2H\Lambda = 0 \implies LH = H\Lambda.$$

This equality $LH = H\Lambda$ implies that the columns of H are the **eigenvectors** of L, and the diagonal elements of Λ are the corresponding eigenvalues. Finally, we evaluate the objective function at this solution:

$$\mathbf{Tr}(H^T L H) = \mathbf{Tr}(H^T H \Lambda) = \mathbf{Tr}(I\Lambda) = \mathbf{Tr}(\Lambda) = \sum_{i=1}^{k} \lambda_i.$$

To minimize the objective function (the sum of eigenvalues), we must select the eigenvectors corresponding to the k **smallest eigenvalues** of L.

3.3.5 The Spectral Embedding Matrix U

The construction of the matrix U represents the critical transition from the spectral domain (eigenvectors) back to the spatial domain (data points). This step is often performed mechanically, but its geometric justification relies on the preservation of vertex indices throughout the linear algebra operations.

We begin with the first k eigenvectors $u_1, \dots, u_k$ of the Laplacian L.

- Each eigenvector u_j is a column vector of dimension $n \times 1$.
- Mathematically, u_j assigns a scalar value to every vertex in the graph corresponding to the j-th vibration mode of the graph.

We stack these column vectors side-by-side to form the matrix $U \in \mathbb{R}^{n\times k}$:

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_k \\ | & | & & | \end{bmatrix}.$$

Once U is constructed, we shift our perspective from the columns to the **rows**. Let $y_i \in \mathbb{R}^k$ denote the i-th row of U:

$$y_i = (U_{i,1}, U_{i,2}, \dots, U_{i,k}).$$

It is crucial to understand why this specific row vector y_i serves as the representation for the original data point x_i.

- **Index Preservation:** The graph construction process maintains a strict bijective mapping between the data index i and the graph vertex v_i. The element L_{ij} in the Laplacian describes the relationship between data point x_i and data point x_j.
- **Eigenvector Entries:** When we solve the equation $Lu = \lambda u$, the resulting eigenvector u is a vector of length n. The i-th entry of this vector, denoted as $u(i)$, explicitly represents the value assigned to the i-th vertex (and thus the i-th data point x_i). By taking the i-th row of U, we are essentially collecting the coordinates of data point x_i across all k principal spectral dimensions.

$$U = \begin{array}{c} \begin{bmatrix} u_1(1) & u_2(1) & \dots & u_k(1) \\ \vdots & \vdots & & \vdots \\ \mathbf{u_1(i)} & \mathbf{u_2(i)} & \dots & \mathbf{u_k(i)} \\ \vdots & \vdots & & \vdots \\ u_1(n) & u_2(n) & \dots & u_k(n) \end{bmatrix} \end{array} \begin{array}{l} \leftarrow \text{Embedding for } x_1 \\ \\ \leftarrow \textbf{Embedding for } \mathbf{x_i} \\ \\ \leftarrow \text{Embedding for } x_n \end{array}$$

Thus, while the columns of U represent the global "cuts" or partitions of the graph, the rows of U represent the specific coordinates of each individual node within those partitions. This change of representation maps the data x_i from a complicated non-linear manifold in $\mathbb{R}^d$ to a simple point y_i in $\mathbb{R}^k$ where Euclidean distance reflects structural similarity.

3.4 Clustering Algorithm

The final stage of the algorithm maps the real-valued solution back to discrete clusters. Since the eigenvectors of the Laplacian approximate the optimal graph partition, we use them as a new feature representation for the data [4].

1. **Eigen-decomposition:** Compute the first k eigenvectors $u_1, \dots, u_k$ of the unnormalized Laplacian L corresponding to the smallest eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k$.

2. **Spectral Embedding:** Construct a matrix $U \in \mathbb{R}^{n \times k}$ containing these eigenvectors as columns.

3. **Feature Representation:** Treat each row $y_i \in \mathbb{R}^k$ of the matrix U as a new data point corresponding to the original vertex x_i.

4. **K-Means:** Apply the k-means clustering algorithm to the points $Y = \{y_1, \dots, y_n\}$ to partition them into k clusters $C_1, \dots, C_k$.

The original data point x_i is assigned to cluster j if and only if the corresponding row y_i is assigned to cluster j by k-means.

4 Implementation and Results

In this chapter, we translate the mathematical framework developed in Chapter 3 into a functional Python implementation. We developed a custom class, `SpectralClustering`, which encapsulates the entire pipeline from graph construction to the final clustering step. The implementation relies on standard scientific computing libraries: `numpy` for matrix operations, `scikit-learn` for nearest-neighbor search and k-means, and `matplotlib` for visualization.

4.1 Dataset Generation

To rigorously evaluate the performance of our implementation, we generated four distinct synthetic datasets (Figure 3). Each dataset tests a specific geometric property:

1. **Concentric Circles:** Two circles, one inside the other. This dataset is not linearly separable.
2. **Interleaved Moons:** Two crescent shapes that interlock. This tests the algorithm's ability to follow "snake-like" manifolds.
3. **Standard Blobs:** Three distinct, spherical clusters. This serves as a control group where simple distance-based methods should succeed.
4. **Anisotropic Blobs:** Three clusters that are stretched diagonally. This tests the algorithm's response to non-spherical (elliptical) density distributions.

4.2 Algorithm Implementation

The core logic is implemented in the `fit_predict` method of our class. This function proceeds sequentially through the four stages of spectral clustering.

4.2.1 Construction of the Similarity Graph

We first calculate the pairwise squared Euclidean distances between all points in the dataset. To convert these distances into similarities, we apply the Gaussian Radial Basis Function (RBF) kernel as defined in Equation (3.4):

```
euDist = squareform(pdist(X, 'sqeuclidean'))
S = np.exp(-euDist / (2 * self.sigma * self.sigma))
```

Here, `self.sigma` corresponds to the scaling parameter σ. This step produces a dense $N \times N$ matrix where every point is connected to every other point.

4.2.2 Sparse Connectivity and Weight Matrix

To capture the local manifold structure and reduce computational noise, we enforce a sparsity constraint using the k-Nearest Neighbors (kNN) approach. We utilize `sklearn.neighbors.kneighbors_graph` to retain only the edges between a node and its k nearest neighbors:

```
knn = kneighbors_graph(X, n_neighbors=self.neighbors, ...)
W = knn.multiply(S).toarray()
W = (W + W.T) / 2
```

The final line ensures the graph is undirected by symmetrizing the matrix ($W = W^T$), satisfying the requirements for the Laplacian derivation.

4.2.3 Laplacian and Eigen-decomposition

We compute the unnormalized Laplacian $L = D - W$. Since L is a real symmetric matrix, we utilize `np.linalg.eigh`, which is optimized for Hermitian matrices and guarantees real eigenvalues.

```
D = np.diag(np.sum(W, axis=1))
L = D - W
eigenvalues, eigenvectors = np.linalg.eigh(L)
```

The function returns the eigenvalues sorted in ascending order. We extract the first k eigenvectors (where k is the number of clusters) to form the spectral embedding matrix U.

4.2.4 Clustering in the Spectral Domain

Finally, we treat the rows of U as the new feature representation of the data. We apply standard K-Means to these low-dimensional embeddings:

```
U = eigenvectors[:, :self.clusters]
kmeans = KMeans(n_clusters=self.clusters, ...)
self.labels_ = kmeans.fit_predict(U)
```

4.3 Experimental Results

We compared our Spectral Clustering implementation against the standard K-Means algorithm.

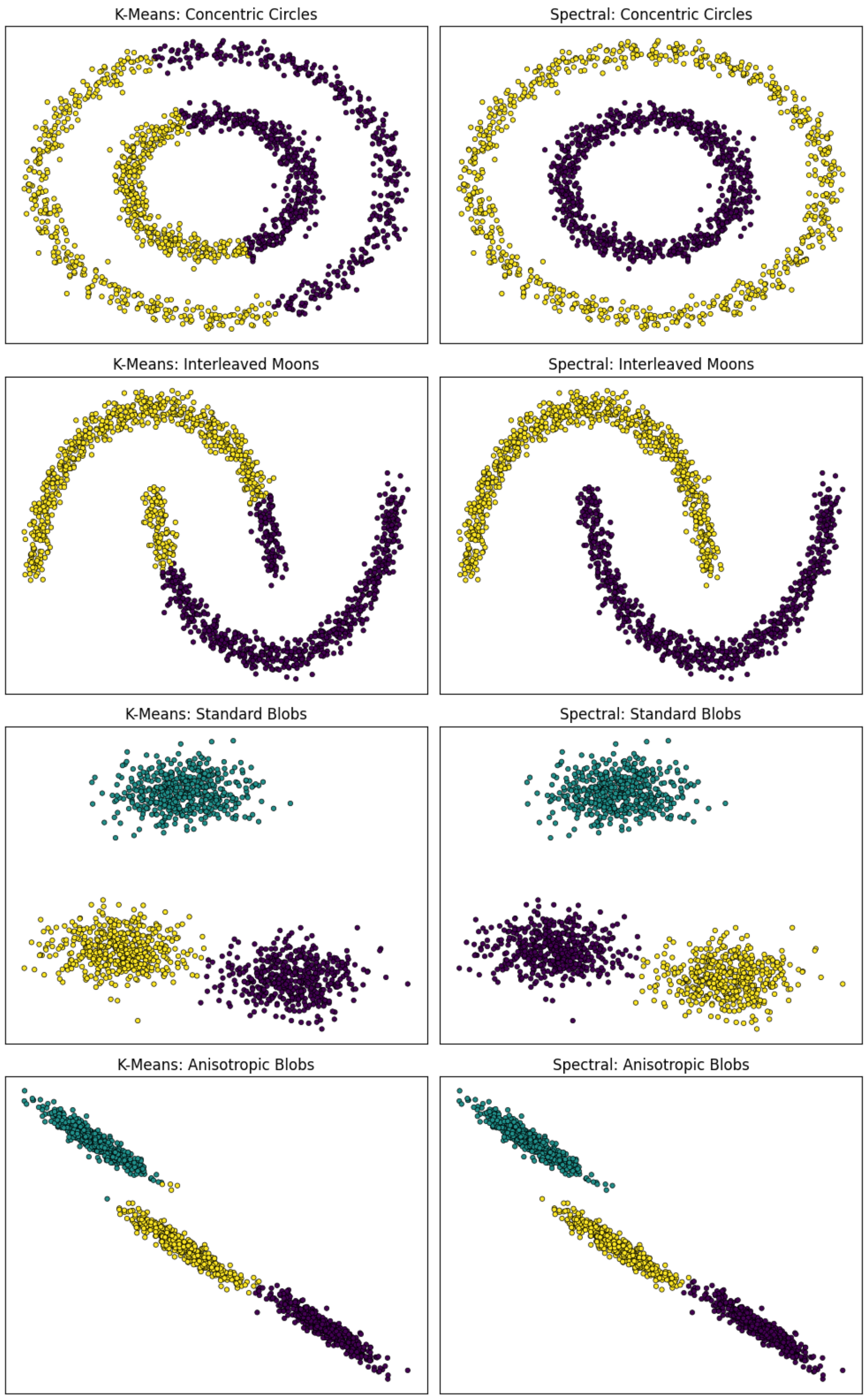

Figure 3: Comparative Analysis: Standard K-Means (Left) vs. Spectral Clustering (Right). Spectral Clustering succeeds on non-convex shapes (Rows 1-2) and correctly identifies non-spherical clusters (Row 4).

4.3.1 Analysis of Results

As illustrated in Figure 3, the performance difference is stark:

- **Non-Convex Shapes (Circles & Moons):** K-Means fails completely on the Concentric Circles and Moons (Rows 1 and 2). K-Means relies on a "centroid" assumption, creating linear boundaries (Voronoi cells). It cannot capture a cluster that surrounds another. Spectral Clustering, however, relies on *connectivity*. Since the points in the outer circle are connected to each other but not to the inner circle, the Graph Laplacian perfectly separates them.
- **Standard Blobs:** For well-separated, spherical blobs (Row 3), both algorithms perform equally well. This confirms that Spectral Clustering generalizes well to simple cases.
- **Anisotropic Blobs:** In Row 4, the clusters are stretched. K-Means assumes spherical variance and incorrectly assigns the "tails" of the stretched blobs to neighboring clusters. Spectral Clustering handles this better because the "stretched" points are still strongly connected to their main cluster in the similarity graph, preserving the true structure.

4.3.2 Spectral Analysis: Eigenvalues and Eigenvectors

To understand how the algorithm selects clusters, we analyzed the spectral properties of the Laplacian matrix across four different datasets: Concentric Circles, Interleaved Moons, Standard Blobs, and Anisotropic Blobs.

The eigenvalues of the Laplacian matrix provide a heuristic for determining the optimal number of clusters k. According to our visualization result, if a graph contains k distinct connected components, the first k eigenvalues will be approximately zero. A significant jump in value occurs between the k-th and $(k+1)$-th eigenvalue.

Figure 4 displays the eigenvalue spectrum for our four test cases.

- **Circles & Moons ($k = 2$):** In the top two plots, we observe that λ_1 and λ_2 are very close to zero. A sharp increase occurs at λ_3, correctly indicating that there are 2 natural clusters.

- **Blobs ($k = 3$):** In the bottom two plots, the first three eigenvalues remains low, with the gap appearing between index 3 and 4. This confirms that the Eigengap method is robust even for non-spherical (anisotropic) shapes.

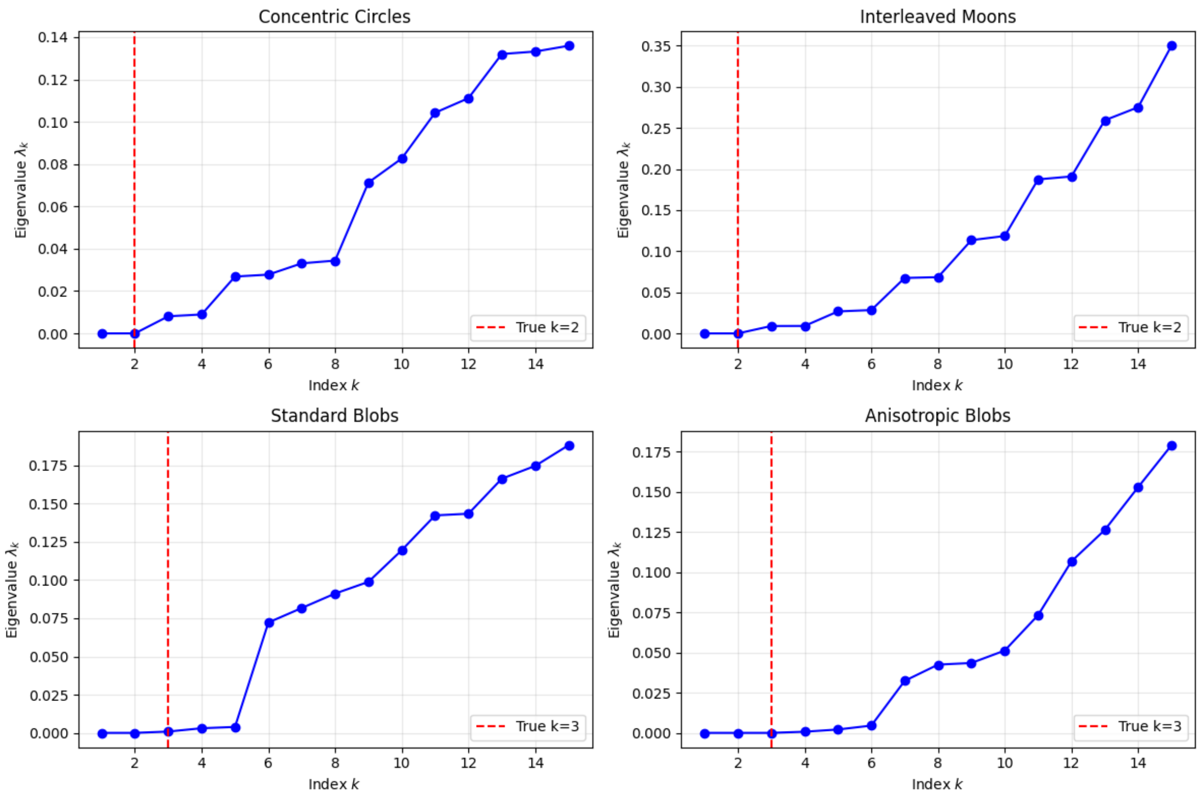

Figure 4: Eigenvalue plots for four datasets. The red dashed line marks the optimal number of clusters k, which consistently coincides with the first significant "jump" in eigenvalue magnitude.

4.3.3 Comparison on Real-World Data (MNIST)

While synthetic datasets provide clear geometric intuitions, the true test of a clustering algorithm is its performance on high-dimensional, real-world data. We evaluated both algorithms on a subset of the MNIST dataset (10% of the data, approx. 7,000 images) to test their ability to group handwritten digits based on pixel similarity.

Figures 5 and 6 visualize the cluster centers (or archetypes) found by each algorithm. Each row represents a single cluster, displaying the 10 data points closest to that cluster's centroid.

Baseline: K-Means Performance

Figure 5 illustrates the results of standard K-Means. The algorithm relies strictly on Euclidean distance in the 784-dimensional pixel space. This limitation leads to several notable errors:

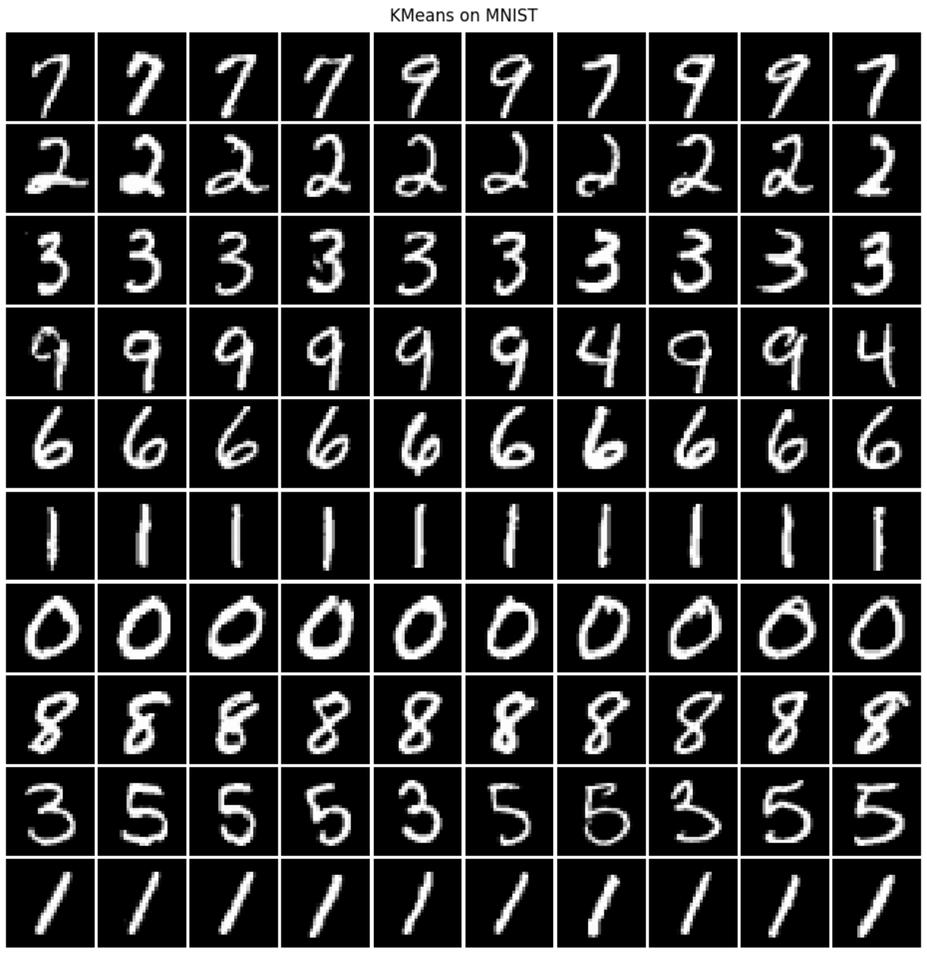

Figure 5: K-Means Clustering on MNIST. Note the confusion between topologically similar digits (Row 9 mixes 3s and 5s) and the splitting of the digit '1' into separate clusters based on slant (Row 6 and Row 10).

- **Fragmentation of Classes:** The digit "1" is split into two distinct clusters (Row 6 and Row 10). K-Means perceives a vertical "1" and a slanted "1" as being far apart in Euclidean space, failing to recognize they belong to the same semantic category.
- **Confusion of Similar Shapes:** Row 9 shows a significant mixture of the digits "3" and "5". These digits share substantial pixel overlap, making them difficult to separate without manifold information. Similarly, Row 1 struggles to distinguish between "7" and "9".

Spectral Clustering Performance

Figure 6 displays the results of our Spectral Clustering implementation. By clustering on the spectral embedding rather than raw pixels, the algorithm successfully captures the underlying manifold structure of the digits.

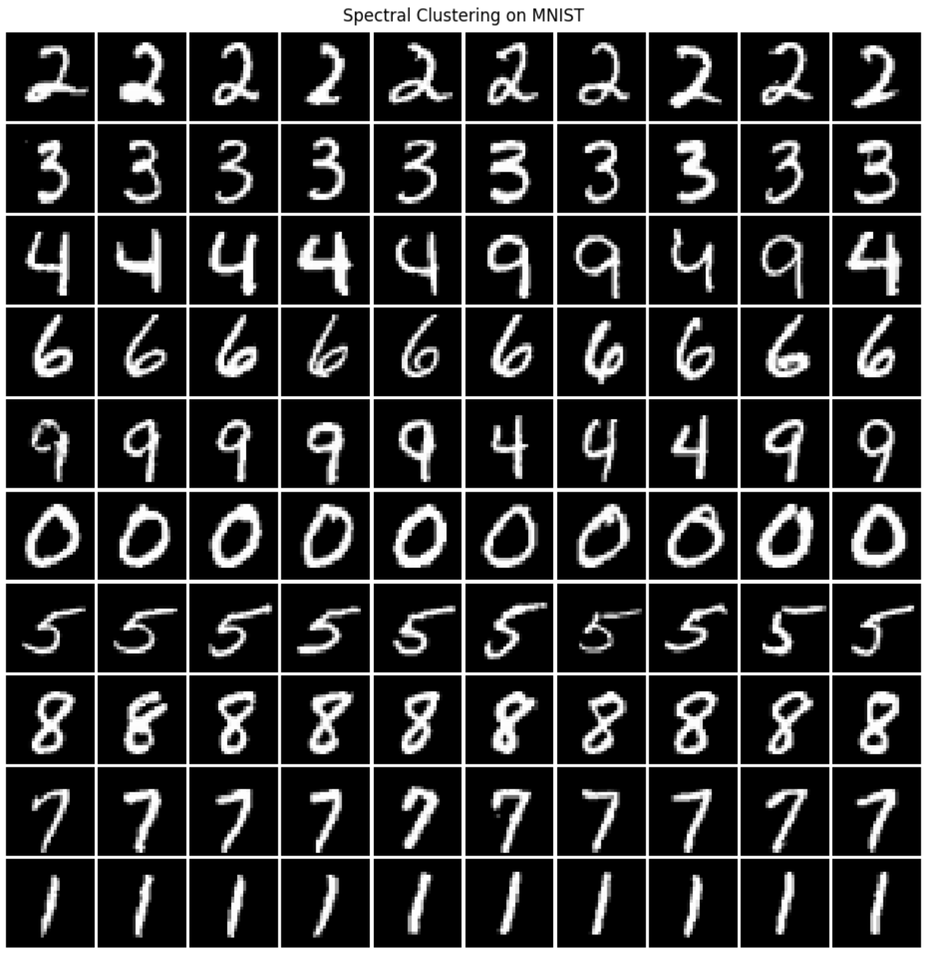

Figure 6: Spectral Clustering on MNIST. The algorithm achieves higher purity within clusters. Note that 1s are consolidated into a single group (Row 10), and 3s (Row 2) are clearly separated from 5s (Row 7).

- **Manifold Invariance:** Unlike K-Means, Spectral Clustering successfully groups all variations of the digit "1" into a single coherent cluster (Row 10). This confirms that the graph Laplacian captures the continuous transformation (rotation/slant) connecting these data points.
- **Improved Separation:** The algorithm achieves much cleaner separation between structurally similar digits. The "3"s are isolated in Row 2, while the "5"s are distinct in Row

7. The confusion between "7" and "9" observed in K-Means is largely resolved, with Row 9 containing almost exclusively "7"s.

This comparison validates that Spectral Clustering is superior for high-dimensional image data where the "distance" between points is better defined by connectivity on a manifold than by straight-line Euclidean distance.

5 Evaluation and Conclusion

5.1 Evaluation

The implementation demonstrates that Spectral Clustering is a powerful technique for analyzing data with complex, non-linear geometries. However, the evaluation highlights several trade-offs:

1. **Geometric Flexibility vs. Computational Cost:** While Spectral Clustering outperforms K-Means on non-convex shapes (as seen in Section 4.3.1), it is computationally more expensive. The eigen-decomposition step has a time complexity of $O(N^3)$, whereas K-Means is $O(N \cdot k \cdot i)$. This makes Spectral Clustering less suitable for extremely large datasets ($N > 10,000$) without approximation techniques.

2. **Sensitivity to Parameters:** The success of the algorithm depends heavily on the construction of the Similarity Graph.
   - If the scaling parameter σ is too small, the graph becomes disconnected, leading to over-segmentation.
   - If the number of neighbors k in kNN is too large, the graph "short-circuits" the manifold (connecting different moons together), causing the algorithm to merge distinct clusters.

3. **Global Optimality:** Unlike K-Means, which can get stuck in local minima and requires multiple restarts, the core step of Spectral Clustering (finding eigenvectors) is a deterministic linear algebra problem with a globally optimal solution. This provides more stable results.

5.2 Conclusion

In this project, we successfully designed and implemented a Spectral Clustering system to address the fundamental limitations of distance-based algorithms like K-Means. By shifting the paradigm from *spatial proximity* (Euclidean distance) to *connectivity* (Graph Laplacian), we demonstrated that complex, non-linear data structures can be effectively partitioned without prior knowledge of their shape.

Our experimental results confirm the theoretical advantages of this spectral approach:

- **Geometric Robustness:** On synthetic datasets, Spectral Clustering successfully "unrolled" non-convex manifolds such as Concentric Circles and Interleaved Moons, tasks where standard K-Means failed completely due to its linearity assumption.
- **Real-World Application:** On the MNIST dataset, the algorithm demonstrated superior semantic grouping. It correctly unified topologically similar digits (like rotated '1's) and distinguished between structurally similar digits (like '3' and '5') that share high pixel overlap.
- **Automated Model Selection:** We validated that the "Eigengap Heuristic" provides a reliable, mathematically grounded method for determining the optimal number of clusters k, a significant advantage over the trial-and-error approach often required for other algorithms.

However, this flexibility comes at a cost. Our evaluation highlighted the $O(N^3)$ computational complexity of the eigen-decomposition step, which makes the standard algorithm prohibitively slow for large-scale datasets compared to the linear time complexity of K-Means.

5.3 Future Work

To bridge the gap between theoretical elegance and industrial scalability, future work will focus on:

1. **Normalized Graph Laplacian:** Currently, our implementation uses the Unnormalized Laplacian ($L = D - W$). Future iterations should implement the Normalized Laplacian ($L_{sym} = I - D^{-1/2}WD^{-1/2}$ or $L_{rw} = I - D^{-1}W$). Theoretical literature suggests that normalized spectral clustering is often superior, particularly when the graph contains vertices with widely varying degrees, as it prevents the algorithm from trivially partitioning the graph by cutting off low-degree outlier vertices.

2. **Approximation Techniques:** To handle larger datasets ($N > 10,000$), we propose implementing the *Nyström Method* or *Landmark Spectral Clustering*. These techniques approximate the full similarity matrix using a small subset of samples, reducing the time complexity of eigen-decomposition from $O(N^3)$ to roughly $O(N)$.

3. **Auto-Tuning Hyperparameters:** Developing an adaptive heuristic for the scaling parameter σ (such as "local scaling") would allow the algorithm to handle datasets with varying densities automatically, eliminating the need for manual hyperparameter tuning.

References

[1] Lars Hagen and Andrew B. Kahng. A new approach to effective circuit clustering. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '92, page 422–427, Washington, DC, USA, 1992. IEEE Computer Society Press.

[2] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[3] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[4] Ulrike von Luxburg. A tutorial on spectral clustering, 2007.

[5] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(11):1101–1113, November 1993.