Otto-von-Guericke University Magdeburg

Faculty of Computer Science

Master Thesis

User Controlled Spectral Clustering: A Framework for Tailored Analysis of Different Similarity Graphs and Graph Laplacians

Author:

Imran Sheikh

August 23, 2023

Advisors:

Prof. Dr. rer. nat. habil. Gunter Saake
Database and Software Engineering Group, Otto von Guericke University
Jun.-Prof.Dr. Robert Heyer
Leibniz-Institut für Analytische Wissenschaften – ISAS – e.V.
Faculty of Technology, Bielefeld University
Dr.-Ing. David Broneske
Deutsches Zentrum für Hochschul- und Wissenschaftsforschung
M.Sc. Daniel Walke
Bioprocess Engineering, Otto von Guericke University
M.Sc. Daniel Micheel
Database and Software Engineering Group, Otto von Guericke University

Sheikh, Imran:
User Controlled Spectral Clustering: A Framework for Tailored Analysis of Different Similarity Graphs and Graph Laplacians
Master Thesis, Otto-von-Guericke University Magdeburg, 2023.

Abstract

Spectral clustering, a popular unsupervised learning technique, effectively uncovers intricate patterns in data by leveraging graph-based representations. Frameworks like scikit-learn facilitates the application of spectral clustering. However, current implementation in scikit-learn is limited. Scikit-learn's spectral clustering offers limited choices for distance metrics, similarity graph construction, and graph Laplacian types, constraining its adaptability to diverse datasets. We identified three key criteria required for a good spectral clustering framework.

Our framework tackles these limitations by offering users seven different choices of distance metrics (criteria i) and three distinct similarity graph options (criteria ii), kNN graph, ϵ-neighborhood graph, and fully connected graph. Additionally, users can choose from three variants of the Laplacian matrix (criteria iii), including the unnormalized, random walk normalized, and symmetric normalized Laplacians.

To assess our framework's capabilities, we conducted performance analyses on the Iris and CORA datasets, comparing their spectral clustering results against k-Means clustering. For Iris, our framework achieves remarkable gains with 94.7% accuracy (5.4% increase), 0.851 Adjusted Rand Index (ARI) (12.1% increase), and 0.831 Normalized Mutual Information (NMI) (9.4% increase). On CORA, our framework exhibits similar performance for accuracy at 22.2% (0.45% increse), significant improvements, were observed for an ARI of 0.23 (283.33% increase) and a NMI of 32.1 (215.69% increase).

As a final evaluation step, we compared our framework results to a prior study on spectral clustering's performance for the Iris dataset. Our framework consistently excels, delivering substantial enhancements in external and internal cluster validity measures. Particularly notable is the combination of the symmetric normalized Laplacian with the epsilon neighborhood graph, resulting in a 15.33% accuracy increase, a 53.92% increase in ARI, and a 38.39% improvement in NMI.

Keywords: Spectral clustering, k-Means, machine learning, distance metric, similarity graphs, graph Laplacian, Iris, CORA.

Acknowledgments

I would like to extend my heartfelt gratitude to Prof. Dr. rer. nat. habil. Gunter Saake for providing me with the incredible opportunity to undertake my thesis under his guidance. Your mentorship and encouragement have been instrumental in shaping my academic journey, and I am truly thankful for your unwavering support.

I am immensely grateful to Jun.-Prof.Dr. Robert Heyer and Dr.-Ing. David Broneske, for their exceptional mentorship and dedicated guidance throughout the entire duration of my thesis. Your expertise, experience, and continuous support have been invaluable in refining my academic pursuits.

A special acknowledgment goes to M.Sc. Daniel Walke for his remarkable patience, understanding, and expertise in supervising me throughout this journey. Your guidance has been invaluable in shaping my work, enhancing my understanding, and bettering my critical thinking.

I extend my sincere appreciation to my family members and close ones for their unwavering support and encouragement throughout this challenging yet rewarding journey. Your belief in me has been a constant source of inspiration.

I also wish to express my gratitude to my elder brother for his unwavering support and dedication to furthering my academic endeavors. Your encouragement has been a driving force in my pursuit of knowledge.

Last but not least, I extend my deepest gratitude to my parents for their unending support and belief in me. Your love, encouragement, and sacrifices have been the cornerstone of my achievements.

Thank you all for being an integral part of this incredible journey and for contributing to my growth as an academician.

Contents

List of Figures

List of Tables

1. Introduction

In today's data-driven world, an unprecedented volume of information is generated daily across various domains, ranging from social networks [Newman, 2004] and biological systems [Barabasi and Oltvai, 2004] to recommendation systems [Adomavicius and Tuzhilin, 2005] and image analysis [Arbelaez et al., 2010]. The intricate relationships that underlie this data often form a complex web of interconnectedness, rendering traditional analytical techniques inadequate for comprehensive insights. Consequently, a pressing need has emerged to devise novel methodologies that can unearth hidden patterns and structures within this intricate data landscape.

Graphs have emerged as a versatile and powerful abstraction for modeling and analyzing interconnected data. A graph is a mathematical representation that consists of vertices (nodes) and edges (links) connecting these vertices, aptly capturing the relationships, dependencies, and interactions among different entities. As real-world data continues to grow in complexity, graphs have proven indispensable for revealing valuable insights and enabling informed decision-making. A graph, can be defined as *"G = (V, E)"*, where V represents the set of vertices, and E represents the set of edges connecting these nodes.

Graphs have gained substantial attention in the realm of machine learning, serving as a foundational data structure for designing algorithms that leverage the intrinsic structure of data. Machine learning techniques can be broadly categorized into two main paradigms, supervised and unsupervised learning. Supervised machine learning operates on labeled datasets, where each data point is associated with a corresponding label or target value. Algorithms in this category learn to map input features to output labels by observing a set of training examples. Classic examples include linear regression [Hastie et al., 2009], decision trees [Breiman, 2017], and support vector machines [Cortes and Vapnik, 1995]. Unsupervised machine learning, on the other hand, thrives in scenarios where labeled data is scarce or entirely absent. This paradigm focuses on uncovering underlying patterns and structures within data [Bishop and Nasrabadi, 2006], often through techniques such as clustering and dimensionality reduction. Clustering algorithms group similar data points together based on intrinsic similarities, providing insights into natural

groupings within the data. Classic examples include k-Means [MacQueen et al., 1967], hierarchical clustering [Jain and Dubes, 1988], and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [Ester et al., 1996].

The traditional spectral clustering framework involves constructing a similarity graph followed by graph partitioning through eigenvalue decomposition. However, a pivotal concern in this framework is the choice of similarity graph and different normalization strategies before eigen decomposition, which significantly influence the clustering performance [Ng et al., 2002]. The lack of a one-size-fits-all solution necessitates a tailored approach that allows users to customize these choices according to the dataset characteristics. The goal of this thesis is to develop a versatile framework that empowers users to improve customization of the spectral clustering process [Shi, 2003].

This thesis attempts to address the limitations of conventional spectral clustering methods by proposing a novel framework that allows for user controlled spectral clustering. The core objective of this framework is to provide users with the autonomy to select distinct similarity graphs and graph Laplacians that align with the specific characteristics of their data. By enabling this level of customization, the framework aims to enhance the clustering performance and robustness across diverse datasets.

In contrast to our comprehensive spectral clustering framework, scikit-learn's [Pedregosa et al., 2011] spectral clustering implementation exhibits several limitations that restrict its adaptability and hinder its effectiveness in certain scenarios.

criteria (i) - Notably, scikit-learn does not provide specifying the distance metric as a hyperparameter to compute the adjacency matrix for pairwise distance calculations. Instead, users must compute the adjacency matrix in advance and then pass the adjacency matrix to the spectral clustering module. This indirect approach can be less intuitive and requires preprocessing steps.

criteria (ii) - Furthermore, scikit-learn's spectral clustering offers only two choices for similarity graphs, the k-Nearest Neighbors (kNN) graph and the Gaussian kernel. This limitation narrows the scope for users to explore different graph structures that might better reflect the underlying relationships within their data.

criteria (iii)) - Another constraint lies in scikit-learn's limited selection of graph Laplacians. The library exclusively employs the symmetric normalized Laplacian (L_{sym}) as the Laplacian matrix. This constraint prevents users from leveraging other Laplacian variants, such as the random walk normalized Laplacian (L_{rw}) and the unnormalized Laplacian (L). [Von Luxburg, 2007] suggests the use of L_{rw} over L_{sym}.

These limitations collectively hinder the exploration of alternative spectral clustering strategies within scikit-learn. Users are confined to a specific set of choices for constructing similarity graphs and graph Laplacians, leaving them unaware and unable to maximize the potential benefits that different configurations might offer for their specific datasets.

Our spectral clustering framework is strategically designed to address these shortcomings, by offering a wide array of choices for distance metrics, similarity graph construction, and graph Laplacians.

Goals of the Thesis and Research Questions

The goal of this thesis is to implement and evaluate a framework which fulfills a pre-defined criteria(i-iii) which allows us to extend the possibilities of spectral clustering in regard to the choice of distance metrics, similarity graphs, and graph laplacian. To achieve this objective, I define the following research questions:

(RQ1) How does the proposed framework enable users to customize their choices to perform spectral clustering across different similarity graphs and graph Laplacians?

(RQ2) How do the clustering results of the proposed framework compare to k-Means approach?

(RQ3) How does the performance of our user-controlled spectral clustering framework compare against existing spectral clustering approaches on the same dataset?

My thesis meticulously addresses the research questions, leveraging insights gained from experiments conducted on both the Iris and CORA dataset.

Structure of this Thesis

Chapter 2 offers essential background context, followed by a review of related work in Chapter 3. Chapter 4 delves into various methodologies and comprehensive experimental procedures. In Chapter 5, we introduce the spectral clustering framework. Our findings and discussions are presented in Chapter 6, and finally, Chapter 7 offers a summary and outlines potential directions for future research.

2. Background

In this chapter, we provide a detailed exploration and analysis of spectral clustering as a technique for effective data clustering. In section 2.1, we begin by introducing readers to graphs, their types, representations, and operations. In section 2.2, there is a brief introduction to spectral graph theory. Then we move on to the most in-depth section of this chapter, spectral clustering, in section 2.3 where we discuss the various steps involved in spectral clustering. In section 2.4 we discuss the challenges of spectral clustering. A use case for explaining the whole spectral clustering process is described in section 2.5. We discuss the various metrics used in this thesis in the final section 2.6.

2.1 Graphs

Graphs are a fundamental data structure used to model relationships between objects in computer science and mathematics. A graph consists of a set of vertices (also called nodes or points) and a set of edges (also called links, lines or arcs) connecting them. A vertex represents an object or an entity in the real world, such as a city, a person, a web page, or a molecule. Vertices are usually represented as points or circles in a graph. An edge, on the other hand, is a line connecting two vertices in a graph, indicating a relationship or a connection between the two vertices. For example, we can think of the internet as a virtual graph, with vertices representing individual webpages and edges representing hyperlinks between two webpages.

Mathematically, we can define a graph $\boldsymbol{G} = (\boldsymbol{V},\boldsymbol{E})$ where $\boldsymbol{V}$ is the set of vertices and $\boldsymbol{E}$ is the set of edges. Graphs can be visualized as a collection of vertices connected by edges in a graph or network diagram. Listed here are some definitions mentioned in the works of [Sedgewick and Wayne, 2011] that will help us understand graph structures and guide us through our discussions on graphs. We refer to (Figure 2.1) for acquainting with some of these definitions. **Self - Loop**- A self-loop is an edge that connects a vertex to itself.

Parallel Edges - Two edges are parallel if they connect the same pair of vertices.

Adjacent and Incident - When an edge connects two vertices, we say that the vertices are adjacent to one another and that the edge is incident on both vertices.

Degree - The degree of a vertex is the number of edges incident on it.

Subgraph - A subgraph is a subset of a graph's edges (and associated vertices) that constitutes a graph.

Path - A path in a graph is a sequence of vertices connected by edges, with no repeated edges.

Simple Path - A simple path is a path with no repeated vertices.

Cycle - A cycle is a path (with at least one edge) whose first and last vertices are the same.

Simple Cycle - A simple cycle is a cycle with no repeated vertices (other than the requisite repetition of the first and last vertices).

Length of Path or cycle - The length of a path or a cycle is its number of edges.

Connected Vertices - One vertex is said to be connected to another if there exists a path that contains both of them.

Connected Graph - A graph is connected if there is a path from every vertex to every other vertex.

Connected Components - A graph that is not connected consists of a set of connected components, which are maximal connected subgraphs.

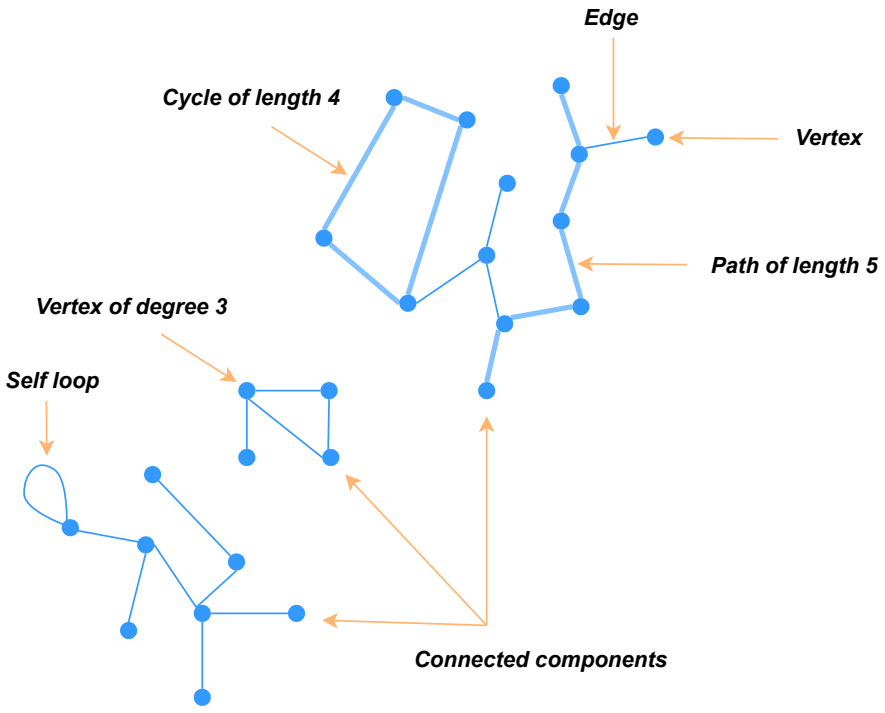


Figure 2.1: Anatomy of an undirected graph

Graphs are powerful tools for visualizing and analyzing data in various fields such as mathematics [West et al., 2001], statistics [Moore et al., 2017], economics [Mankiw,

2014], and computer science [Cormen et al., 2009b]. Graphs provide a clear and concise representation of information, allowing us to identify patterns, trends, and relationships that may not be apparent in raw data. Graphs enable us to communicate complex data in a visual format, making it easier for others to understand and interpret the information. They help us present data-driven insights, support decision-making processes, and communicate findings in a more accessible and intuitive manner.

2.1.1 Types of graphs

Graph theory is a branch of mathematics that studies the properties and relationships of graphs. Graphs can be categorized into various types based on specific characteristics and properties. Here is a brief overview of some common types of graphs in graph theory:

Directed Graph - A directed graph, also known as a digraph, is a graph that consists of a set of vertices and a set of directed edges. In a directed graph, each edge has a direction associated with it, which means that it has a starting vertex and an ending vertex. The graph in the (Figure 2.2a) has four vertices, labeled A, B, C, and D, and four directed edges, labeled AB, BC, CD, and DA. The direction of each edge is indicated by an arrowhead, which points from the starting vertex to the ending vertex.

For instance, the edge AB has its tail connected to vertex A and its head connected to vertex B, indicating that it goes from A to B. Similarly, the edge CD has an arrowhead pointing from vertex C to vertex D, indicating that it goes from C to D.

It is important to note that the direction of an edge is crucial in a directed graph because it indicates the nature of the relationship between the vertices. The edge AB in this example represents a one-way relationship from A to B, which is different from the reverse relationship from B to A.

Directed graphs can be used to represent various types of relationships, such as communication networks [Peterson and Davie, 2011], food webs [Pascual and Dunne, 2006], traffic flow [Ni, 2015], electrical circuits [Nilsson, 2014] and many more.

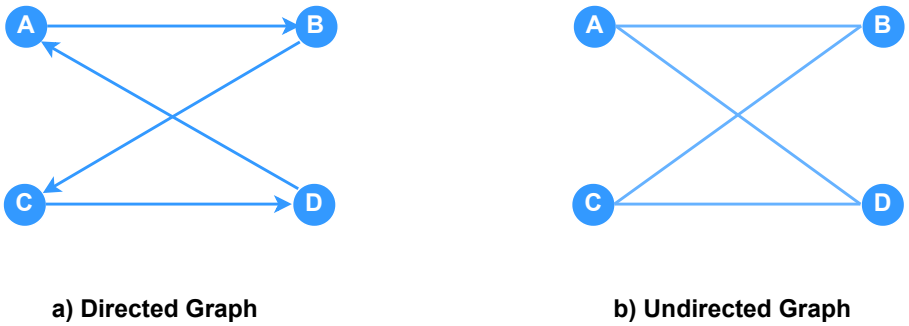
For example, consider a social network where users can follow each other. The users can be represented as vertices, and the follow relationship can be represented as directed edges from the follower to the followed user.

Undirected Graph - An undirected graph is a graph where each edge connects two vertices, but no direction is associated with the edge. In other words, the edges do not have a specific starting or ending point. In the (Figure 2.2b) the edge that connects vertex A to vertex B, also connects vertex B to vertex A.

Undirected graphs are commonly used to model pairwise relationships between objects or to represent networks in which the direction of a connection is irrelevant. They are particularly useful in situations where the relationship is symmetric.

For example, consider a group of friends who are connected by their friendships. The friends can be represented as vertices, and the friendships can be represented as undirected edges between the vertices.

[1] https://algs4.cs.princeton.edu/41graph/



a) Directed Graph b) Undirected Graph

Figure 2.2: Examples of Directed and undirected graphs

Connected Graph - A connected graph is a graph in which there is a path between any two vertices in the graph. In other words, there are no isolated vertices in the graph, and every vertex can be reached from every other vertex by following a sequence of edges. (Figure 2.3a) shows a connected graph. In this graph, there are five vertices labeled A, B, C, D, and E, and there are four edges connecting them. This graph is a connected graph because there is a path between any two vertices in the graph. For instance, there is a path between vertex A and vertex E that goes through vertices A, B, D, and E.

An example of a connected graph is a road network. In a road network, cities or locations are represented as vertices, and roads or highways between them are represented as edges. Since there is a path between any two cities or locations in a road network, although it may not be the shortest path, the graph is connected.

Analyzing the structure of a road network graph can provide insights into traffic flow, congestion, and transportation planning. For example, identifying key vertices (such as large cities or transportation hubs) and edges (such as major highways or intersections) can help prioritize infrastructure improvements and reduce bottlenecks in the network [Gross and Yellen, 2005].

Disconnected Graph - A disconnected graph is a graph that is not connected, meaning that there is at least one vertex in the graph that is not connected by a path. In other words, the graph can be split into two or more disconnected components, each of which is a connected subgraph. In the graph shown in (Figure 2.3b), there are six vertices labeled A, B, C, D, E, and F, and there are four edges connecting them. However, the graph is disconnected because there is no path between vertices A, B, and C and vertices D, E, and F. Instead, the graph can be split into two connected components: {A, B, C} and {D, E, F}.

Disconnected graphs arise in many applications. An example of a disconnected graph is a power grid. In a power grid, power stations and substations are represented as vertices, and power lines between them are represented as edges. Disconnected components in the graph can represent areas that are not well-connected to the rest of the grid, indicating areas where power outages are more likely to occur. Analyzing the structure of a power grid graph can help identify vulnerabilities and improve the resilience of the system [Dorf and Svoboda, 2010]. Identifying key vertices and edges and ensuring redundancy in the network can help prevent widespread outages in the event of failures or natural disasters.

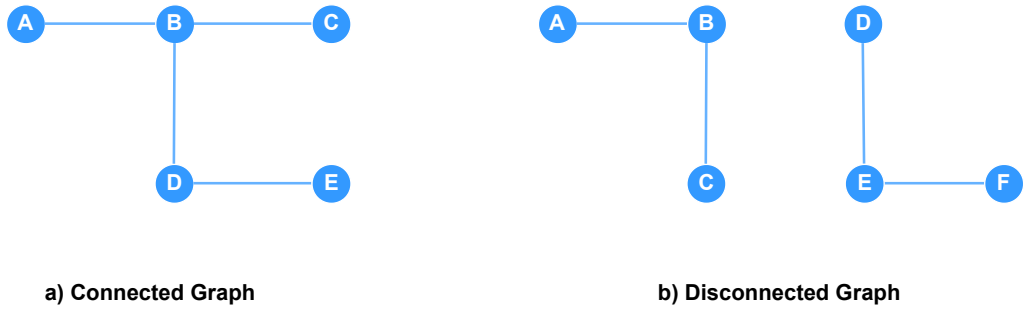


Figure 2.3: Examples of Connected and Disconnected graphs

Regular Graph - A regular graph is a graph in which every vertex has the same degree, meaning the number of edges incident to each vertex is the same. In other words, every vertex in a regular graph has the same number of adjacent vertices. A regular graph is often represented as a k-regular graph, where k is the degree of each vertex in the graph. (Figure 2.4a) shows an example of a 3-Regular graph. In this example, every vertex has three incident edges and three adjacent vertices. The vertex A is adjacent to vertices B, D, and F and the edges incident to vertex A are edges AB, AD, and AF.

An application of a regular graph is well illustrated in the work of [Brouwer and Haemers, 2012]. The authors explain that the atoms in a crystal can be viewed as vertices of a graph, and the bonds between atoms can be represented as edges of the graph. If the crystal has a regular structure, then the resulting graph will be a regular graph. By studying the spectrum of the adjacency matrix of the graph, physicists can gain insights into the properties of the crystal, such as its mechanical and electrical properties. We will see more about adjacency matrix of a graph under Graph Representations in subsection 2.1.2. Regular graphs have many interesting properties and are studied in various fields, including graph theory [Godsil and Royle, 2001], computer science [Hoory et al., 2006], and physics [Brede, 2012].

Complete Graph - A complete graph is a graph in which each pair of distinct vertices is connected by an edge. In other words, a complete graph is a graph in which every vertex is adjacent to every other vertex. A complete graph is a regular graph, which has a degree of n-1, where n is the total number of vertices. In the example shown in (Figure 2.4b) the complete graph has a degree of n -1 (6 - 1 = 5) and thus can also be called a 5-regular graph.

One example of the application of complete graphs in the study of complex networks [Börner et al., 2010] is the visualization of citation networks in scientific literature. In a citation network, each paper is represented as a vertex, and edges represent citations between papers. By constructing a complete graph between papers that have cited each other, researchers can visualize clusters of papers that are highly connected and have a strong influence on each other.

Weighted Graph - A weighted graph is a graph in which each edge is assigned a numerical weight. These weights are numerical values that could be used to represent any attribute of the edge, such as distance, time, cost, or any other metric. Weighted graphs are used to model a variety of real-world phenomena. For example,

a) 3-Regular graph b) Complete graph

Figure 2.4: Examples of d-Regular and Complete graphs

in a transportation network, the weights might represent the distance between two locations, the time required to travel between them, or the cost of traveling between them. (Figure 2.5a) shows an example of weighted graph which is also directed. In this example the numbers associated with the edges are their weights. For e.g. the edge from vertex A to vertex B has a weight of 3, the edge from vertex B to vertex H has a weight of 10, and so on.

In the work of [Sedgewick and Wayne, 2011], the authors provide an example of a weighted graph that represents an airline route network. In this graph, each vertex represents an airport, and each edge represents a flight route between two airports. The weight of each edge represents the distance or cost associated with the flight route. For instance, the edge connecting New York City and Chicago might have a weight of 800 miles, while the edge connecting New York City and Los Angeles might have a weight of 2,800 miles. By using weighted graphs, airline companies can optimize their routes to reduce costs and improve efficiency.

Bipartite Graph - A bipartite graph is a special type of graph in which the vertices can be divided into two disjoint sets such that no edge connects vertices within the same set. In other words, edges connect vertices in different sets and not in the same set. (Figure 2.5b) shows an example of a bipartite graph. In this example the vertex set V1 is {A,B,C,D,E} and the vertex set V2 is {F,G,H}. Note that all edges are between the two vertices set V1 and V2 and not within the same set.

A classic example of a bipartite graph is a graph representing the relationship between customers and products in a store [Barabási, 2016]. Customers can be represented by one set of vertices and products by another set, and an edge is drawn between a customer and a product if that customer has purchased that product. Another example of a bipartite graph is a graph representing the relationship between actors and movies. Actors can be represented by one set of vertices and movies by another set, and an edge is drawn between an actor and a movie if that actor has appeared in that movie.

Cyclic Graph - A cyclic graph is a type of graph where there is at least one cycle. A cycle is a path that starts and ends at the same vertex. In other words, a cyclic graph has at least one path that traverses some edges and vertices and returns to the starting point, forming a loop. A cycle can be of any length, including length 1, which means a single vertex that connects to itself.

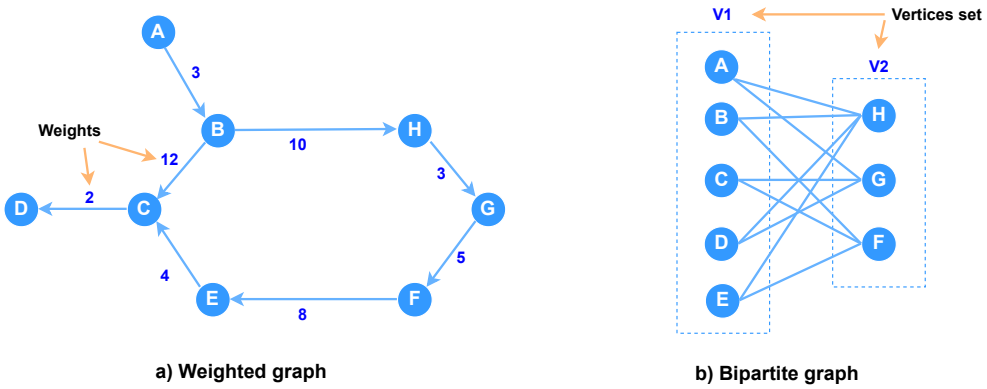


Figure 2.5: Examples of Weighted and Bipartite graphs

A real-world example of a cyclic graph is the cycle of water through the Earth's atmosphere, oceans, land, and back to the atmosphere. This cyclic process is called the water cycle or hydrological cycle [Winterwerp and Van Kesteren, 2004]. Water evaporates from the Earth's surface and forms clouds, which then precipitate as rain or snow onto the land or oceans. The water that falls on the land or oceans then flows back into rivers, lakes, and oceans, or is absorbed by plants and animals. This water is eventually evaporated back into the atmosphere, completing the cycle. (Figure 2.6) shows an example of a cyclic graph. In this graph, the different phases of the water cycle are represented as vertices. Namely, Collection, Evaporation, Condensation, and Precipitation and the edges represent the direction of water movement between different phases in the cycle.

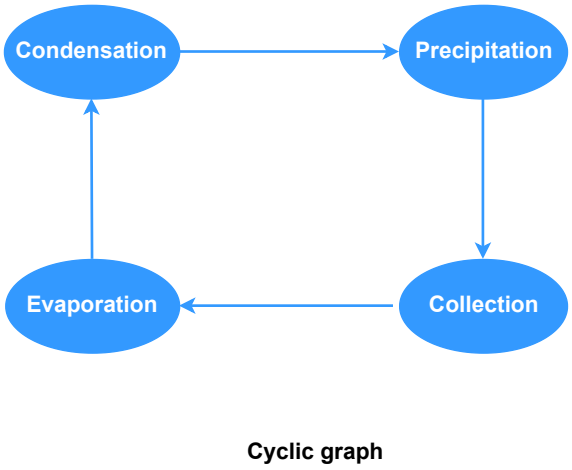


Figure 2.6: Cyclic Graph for the water cycle

Null Graph, Trivial Graph, and Empty Graph - A null graph, a trivial graph, and an empty graph are all special cases of graphs with no edges.

A null graph is a graph with one or more vertices and no edges. (Figure 2.7a) shows a null graph with five vertices and no edges between them.

A trivial graph is a graph with one vertex and no edges. It is also called a singleton graph. (Fiure 2.7b) shows a trivial graph with one vertex and no edge.

An empty graph is a graph with no vertices and no edges. Depending on the author and the text, an empty graph can sometimes interchangeably be called as a null graph. (Figure 2.7c) shows an empty graph which has no vertex or edge.

The applications of these graphs [Trudeau, 2013] are limited due to their simple nature. It is worth noting that while these graphs may not have many practical applications, they are still important in graph theory as they form the basis for more complex graphs and help to establish fundamental concepts and properties.

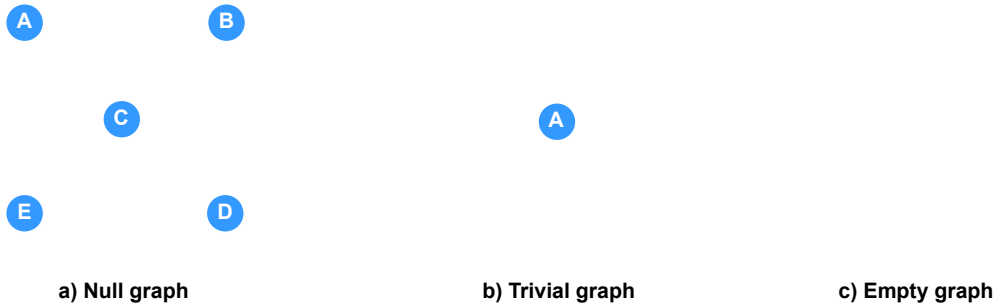


Figure 2.7: Examples of Null, Trivial, and Empty Graphs

2.1.2 Graph representations

Graph representation refers to the way in which a graph is represented or stored in a computer program or an algorithm. Graph representation is important for many graph-based algorithms, as the choice of representation can have a significant impact on the efficiency and effectiveness of the algorithm. There are several ways to represent graphs, including:

Adjacency matrix - An adjacency matrix is a way to represent a graph as a 2D matrix where the rows and columns correspond to the vertices of the graph, and the entries in the matrix indicate whether there is an edge between the corresponding vertices. An adjacency matrix for a graph with n vertices is a square matrix $(n \times n)$ with n rows and n columns. Mathematically, we can represent an adjacency matrix A for a graph G with n vertices as follows:

The element,

$A[i, j] = 1$, if there is an edge from vertex i to vertex j in G

$A[i, j] = 0$, otherwise

If G is an undirected graph, then

$A[i, j] = A[j, i]$ for all i, j.

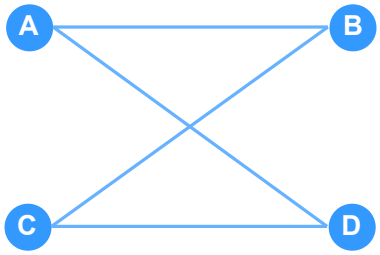
It is important to note that the diagonal elements of the adjacency matrix A are always 0 for an undirected graph since there are no self-loops in an undirected graph. For a directed graph, the diagonal elements can be either 0 or 1, depending on whether self-loops are allowed or not.

If G is a weighted graph, then

$A[i, j] = w[i, j]$, where $w[i, j]$ is the weight of the edge from

vertex i to vertex j.

Adjacency matrices are commonly used to represent graphs in computer algorithms because they allow for efficient access to information about the edges and vertices of the graph. For example, to find all the neighbors of a vertex in the graph, we simply need to look at the corresponding row or column in the adjacency matrix. (Figure 2.8) shows an example of an adjacency matrix for an undirected graph G with 4 vertices labeled A, B, C, and D and the following edges: AB, BC, CD, and DA.



Graph G

	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	1
D	1	0	1	0

Adjacency Matrix

Figure 2.8: An example graph G with 4 vertices and its corresponding (4x4) adjacency matrix based on the edge connections of the graph G.

Adjacency list - An adjacency list is a data structure used to represent a graph as a collection of linked lists, where each linked list corresponds to a vertex in the graph and contains the vertices adjacent to it. In other words, in an adjacency list, each vertex in the graph has a list of its neighboring vertices. Each vertex in the list represents an adjacent vertex and can contain additional information, such as the weight or label of the edge connecting the vertices.

The adjacency list representation is particularly useful for sparse graphs where the number of edges is much smaller than the number of vertices, as it allows for a more efficient use of memory compared to the adjacency matrix representation.

(Figure 2.9) shows an example of an adjacency list representation for an undirected graph with 4 vertices labeled A, B, C, and D and the following edges: AB, AC, BC, CD, and DA.

In this example, the adjacency list for vertex A contains the adjacent vertices B, C, and D. Similarly, the adjacency list for vertex B contains the adjacent vertices A and C. The adjacency list representation provides an efficient way to iterate over the neighbors of each vertex, making it a popular choice for graph algorithms. Note that for a directed graph, the adjacency list representation would contain separate lists for the incoming and outgoing edges for each vertex. Also, for weighted graphs, the adjacency list would store the weight of each edge in addition to the adjacent vertex.

Edge list - An edge list is a simple way to represent a graph by listing all the edges in the graph as pairs of vertices or triples of vertices and edge weights. In an

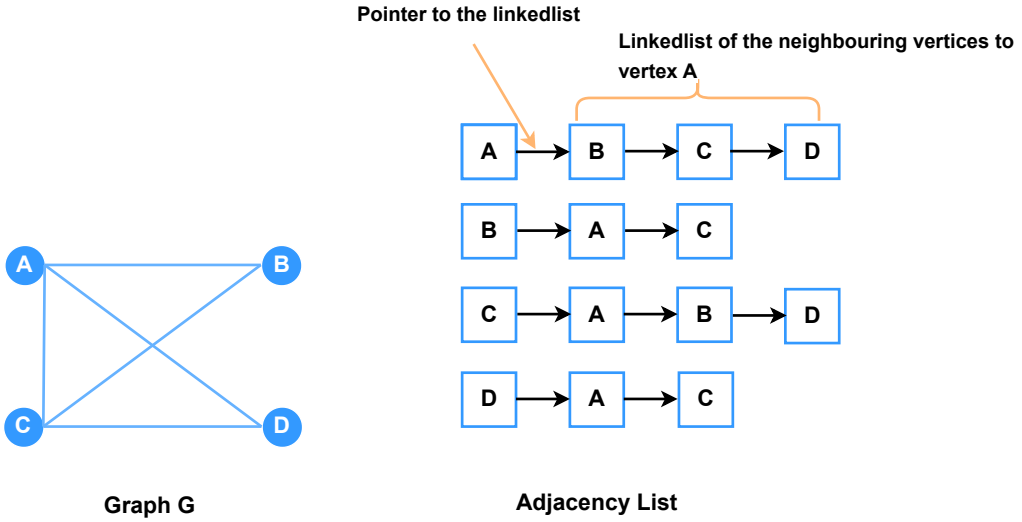


Figure 2.9: An example graph G with 4 vertices and its corresponding adjacency list based on the 5 edge connections of the graph G. The arrow from A to B in the adjacency list is the pointer to the linkedlist and subsequently the vertices B, C, and D are linkedlist to vertex A.

edge list, each edge in the graph is represented by a tuple or triple, where the first two elements indicate the vertices connected by the edge, and the third element (if present) indicates the weight or label of the edge as shown in (Figure 2.10). A tuple is a sequence of two values, such as (A, B), where A and B are the endpoints of an edge in the graph. A triple, on the other hand, is a sequence of three values, such as (A, B, W), where A and B are the endpoints of an edge and W is the weight of the edge.

(Figure 2.10b) shows an example of an edge list for an undirected and unweighted graph (Figure 2.10a) without the weights. In this example, each line represents an edge between two vertices.

Note that for weighted graphs (Figure 2.10a), the edge list representation would include a third element for each edge indicating the weight or label of the edge, as shown in (Figure 2.10c). In this example, each line represents an edge between two vertices and the weight of the edge.

The edge list representation is compact and easy to understand, making it a popular choice for small graphs or for storing graph data in a text file. However, the edge list representation does not provide direct access to information about the vertices or their neighbors. In order to determine the neighbors of a vertex, one must iterate over all the edges in the list to find those that involve the given vertex. For example, to find all the neighbors of vertex A, we need to scan through the edge list and identify all the edges that have A as one of the endpoints. This can be less efficient than the adjacency list representation, which provides direct access to the neighbors of each vertex.

Incidence matrix - An incidence matrix is a matrix representation of a graph where each row corresponds to a vertex and each column corresponds to an edge. For a graph with n vertices and m edges, the incidence matrix is an n x m matrix.

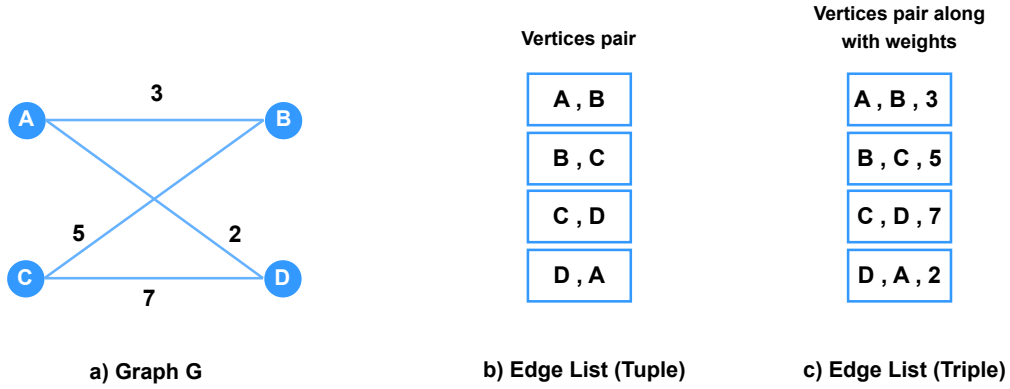


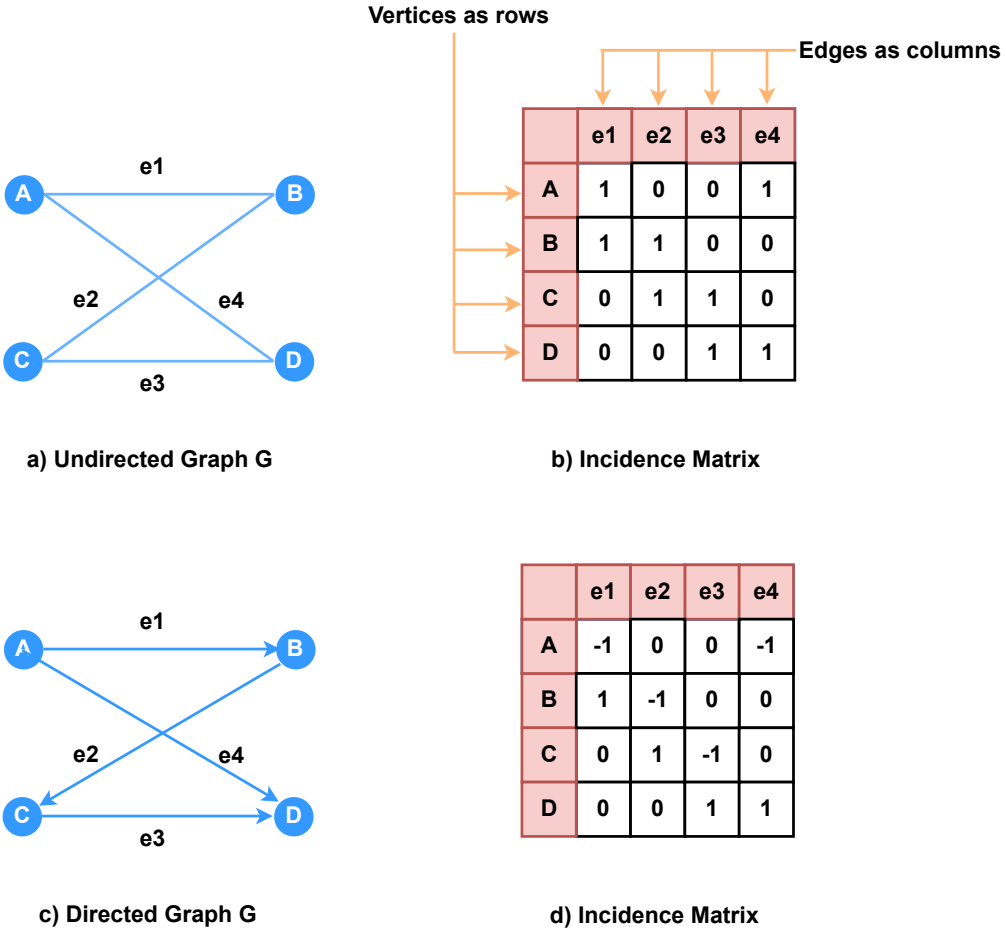
Figure 2.10: (a) An example undirected, weighted graph G, (b) Edge List with only vertices pair also called a tuple, and (c) Edge List with vertices pair and weights also called a triple

In an undirected graph, the incidence matrix is a binary matrix. The entry in row i and column j is 1 if vertex i is incident to edge j, and 0 otherwise. For example, let's consider the undirected graph in (Figure 2.11a), vertex A is incident to edges e1, and e4, so the entries in the first row of the incidence matrix shown in (Figure 2.11b) are 1, 1 respectively for e1 and e4.

In a directed graph, the incidence matrix can have entries of -1, 0, and 1 [Gross and Yellen, 2005]. The sign of the entry depends on whether the vertex is connected to the tail or head of the edge. If the vertex is connected to the tail of an edge then the entry in the incidence matrix is -1, if the vertex is connected to head of an edge the entry in the matrix is 1, and 0 when an edge is not incident on a vertex. For example, consider the directed graph in (Figure 2.11c). The vertex A is connected to vertices B and D with two edges e1 and e4 with their tails respectively. Thus, the entry in the incidence matrix shown in (Figure 2.11d) is -1 and -1 for edges e1 and e4 alongside vertex A. However, vertex B is connected to vertices A and C with two edges e1 and e2 with their head and tail respectively. Thus, the entry for vertex B is 1 and -1 for edges e1 and e2 respectively. It is important to note that many authors use the opposite sign convention.

The incidence matrix can be used to represent graphs in computer algorithms and to perform various graph operations. However, it is not always the most efficient representation, especially for sparse graphs, where the number of edges is much smaller than the number of possible edges. In those cases, alternative representations such as adjacency matrices or adjacency lists may be more efficient.

Each of these representations has its own advantages and disadvantages, depending on the application and the properties of the graph. For example, adjacency matrices are efficient for testing whether two vertices are adjacent (adjacency test), but can be memory-intensive for large graphs. However, adjacency lists, on the other hand, are memory-efficient but can be slower for adjacency tests. Edge lists are compact, but can be slow in searching for specific vertices. Incidence matrices are useful for certain algorithms, such as network flow algorithms, but can be slow for adjacency test



	e1	e2	e3	e4
A	1	0	0	1
B	1	1	0	0
C	0	1	1	0
D	0	0	1	1

	e1	e2	e3	e4
A	-1	0	0	-1
B	1	-1	0	0
C	0	1	-1	0
D	0	0	1	1

Figure 2.11: (a) is an undirected graph G with 4 vertices and 4 edges and (b) is its corresponding 4x4 incidence matrix, where vertices and edges are represented as rows and columns respectively. In the case of an undirected graph the entry 1 in the incidence matrix indicates connection of an edge to a vertex and 0 otherwise. (c) is a directed graph G and (d) is its corresponding incidence matrix. In this incidence matrix, -1 indicates an edge leaving a vertex, 1 indicates an edge entering a vertex, and 0 indicates an edge is not incident to a vertex.

[Sedgewick and Wayne, 2011]. A comprehensive overview about graph representations is presented by [Goodrich et al., 2014].

2.1.3 Graph operations

Graph operations are algorithms or procedures that manipulate a graph or extract information from it. There are several types of graph operations, including basic operations such as adding and removing vertices and edges, updating vertex/edge properties, and traversing the graph, furthermore there are advanced operations such as finding shortest paths, minimum spanning trees, and maximum flows.

The basic graph operations are:

Add vertex

Add edge

Remove vertex

Remove edge

Update vertex/edge properties

Traversal - This operation visits all the vertices in the graph in a systematic way, such as depth-first search (DFS) or breadth-first search (BFS) algorithms. DFS and BFS are two fundamental graph traversal algorithms used to visit all the vertices and edges of a graph.

Depth-First Search (DFS) [Sedgewick and Wayne, 2011] is a graph traversal algorithm that explores as far as possible along each branch before backtracking. The algorithm starts at a vertex, visits all its adjacent vertices, and recursively explores the vertices that have not been visited until all vertices in the graph have been visited. DFS can be implemented using a stack or by recursion. The traversal order for DFS of the graph in (Figure 2.12) is 1, 2, 4, 5, 3, 6, 7. Starting from vertex 1, the DFS algorithm will visit vertex 2, then vertex 4, then backtrack to vertex 2 and visit vertex 5, then backtrack to vertex 1 and visit vertex 3, then visit vertex 6, backtrack to vertex 3, visit vertex 7, and finally backtrack to vertex 1.

Breadth-First Search (BFS) [Cormen et al., 2009a] is a graph traversal algorithm that explores the graph level by level. It starts at a vertex and visits all of its adjacent vertices. Then, it moves to the next level of vertices and visits all of their adjacent vertices. The process continues until all vertices have been visited. BFS can be implemented using a queue. The traversal order for BFS of the graph in (Figure 2.12) is 1, 2, 3, 4, 5, 6, 7. Starting from vertex 1, the BFS algorithm will visit vertex 1, then visit vertices 2 and 3, then visit vertices 4, 5, 6, and 7 in that order.

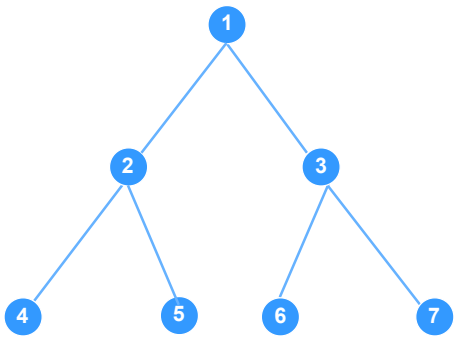


Figure 2.12: An example graph with DFS (Depth-First Search) traversal order 1, 2, 4, 5, 3, 6, 7 and BFS (Breadth-First Search) traversal order 1, 2, 3, 4, 5, 6, 7.

The advanced graph operations are:

Shortest path - The concept of a ”shortest path” between two vertices in a graph is typically associated with weighted graphs, where the edges of the graph have numerical weights that represent the cost or distance between the vertices. In this context, the shortest path problem involves finding the path between two vertices that has the smallest sum of edge weights.

However, it is possible to extend the concept of shortest path to unweighted graphs as well. In an unweighted graph, all the edges have the same weight, and the shortest path between two vertices is simply the path with the smallest number of edges. There are many algorithms for solving the shortest path problem, but two of the most commonly used algorithms are Dijkstra's algorithm [Dijkstra, 2022] and the Bellman-Ford algorithm [Bellman, 1958].

Minimum spanning tree - A minimum spanning tree is a tree that connects all the vertices of a given undirected graph with the minimum possible total edge weight. In other words, it is a tree that spans all the vertices of the graph while minimizing the sum of the weights of the edges in the tree. Overall, the minimum spanning tree operation is useful for finding a subgraph that represents the ”cheapest” way to connect all the vertices of a graph. Kruskal's [Kruskal, 1956] and Prim's [Prim, 1957] algorithms provide two ways to efficiently find such a subgraph.

If there are no weights associated with the edges of a graph, then the problem of finding a minimum spanning tree reduces to the problem of finding a minimum spanning tree with respect to the number of edges. In other words, the goal is to find a subgraph that is a tree and connects all the vertices of the graph with the minimum possible number of edges. One algorithm that can be used to solve this problem is the breadth-first search (BFS) algorithm [Cormen et al., 2009a].

Maximum flow - In a network flow graph, there is a directed graph with a source vertex (a vertex that generates flow) and a sink vertex (a vertex that receives flow). Each edge in the graph has a capacity that determines how much flow can pass through that edge. The goal of the maximum flow operation is to find the maximum amount of flow that can be sent from the source vertex to the sink vertex without exceeding the capacity of any edge. The Ford-Fulkerson algorithm [Ford and Fulkerson, 1956] and the Edmonds-Karp algorithm [Edmonds and Karp, 1972] are two common algorithms used to solve this problem.

Topological sorting - Topological sorting is a common graph operation that involves ordering the vertices in a directed acyclic graph (DAG) in such a way that for every directed edge uv in the graph, vertex u comes before vertex v in the ordering. In other words, the ordering is such that all the edges in the graph point forward, from earlier vertices to later vertices.

To perform a topological sort of a DAG, there are several algorithms that can be used, including depth-first search (DFS) [84] and breadth-first search (BFS) [24]. The basic idea is to start at a vertex with no incoming edges (i.e., a vertex with in-degree 0) and add it to the ordering. Then, remove that vertex and all its outgoing edges from the graph and repeat the process until all vertices have been added to the ordering.

It is important to note that topological sorting is only defined for directed acyclic graphs (DAGs), since cyclic graphs have no valid ordering that satisfies the requirement of all edges pointing forward. If the graph contains cycles, it is not possible to perform a topological sort.

Connectivity - Connectivity is a fundamental graph operation that involves determining whether a given graph is connected or disconnected. A graph is said to be connected if there exists a path between any two vertices in the graph. If a graph is not connected, it is said to be disconnected and can be broken down into two or more connected components.

There are several algorithms that can be used to determine the connectivity of a graph, including depth-first search (DFS) and breadth-first search (BFS). The basic idea behind these algorithms is to start at a vertex and explore all the vertices that can be reached from that vertex using edges in the graph. If all vertices can be reached, the graph is connected; otherwise, the graph is disconnected and can be broken down into connected components. Finding the connected components of a graph involves identifying all the subgraphs of the graph that are connected. This can be done using similar algorithms to those used to determine the connectivity of the graph, such as DFS or BFS.

These operations are essential for solving many problems in graph theory and are widely used in various applications such as network routing [Kurose and Ross, 2017], social network analysis [Newman, 2018], and image segmentation [Jain, 1989]. Graph algorithms including various graph operations are discussed in [Brandes, 2005], [Cormen et al., 2009a], [Goodrich et al., 2014].

2.2 Spectral Graph Theory

Spectral graph theory [Biggs, 1993] is a branch of mathematics that explores the properties of graphs using linear algebra [Strang, 2012] and spectral theory [Aupetit, 2012]. In spectral graph theory, ”spectral” refers to the use of the eigenvalues and eigenvectors of certain matrices associated with a graph [Chung, 1997]. These matrices are often called ”spectral matrices”. The most commonly used spectral matrix in spectral graph theory is the Laplacian matrix. Other spectral matrices used in spectral graph theory include the normalized Laplacian matrix, and the adjacency matrix itself. By analyzing the eigenvalues and eigenvectors of these matrices, spectral graph theory provides insights into various aspects of graph structure, connectivity, and behavior. In this subsection, we introduce and familiarize readers with the Laplacian matrix. We will discuss in detail the different graph Laplacian and their properties in section 2.3.3.

To derive the Laplacian matrix L, we need to compute a separate matrix called the degree matrix D from the adjacency matrix A of a graph. The degree matrix is a diagonal matrix that sums up the number of edges on every node in the graph. The degree matrix is computed by summing each row of the adjacency matrix and then placing the sum or the degree in the corresponding position of the row number. We have already seen the computation of an adjacency matrix in section 2.1.2. Once we have computed both the adjacency matrix and the degree matrix of a graph we then compute the Laplacian matrix which is defined as:

$$L = D - A \tag{2.1}$$

where L is the Laplacian matrix, D is the degree matrix and A is the adjacency matrix of a graph.

We present an example in (Figure 2.13) to visualize the various steps involved in deriving the Laplacian matrix from a graph. (Figure 2.13a) shows an example graph G with six nodes and eight edges and the (Figure 2.13b) shows the adjacency matrix constructed from the graph G based on the connectivity of nodes and edges. (Figure 2.13c) shows the degree matrix computed from the adjacency matrix of the graph G, and (Figure 2.13d) shows the unnormalized Laplacian matrix computed by subtracting the adjacency matrix from the degree matrix.

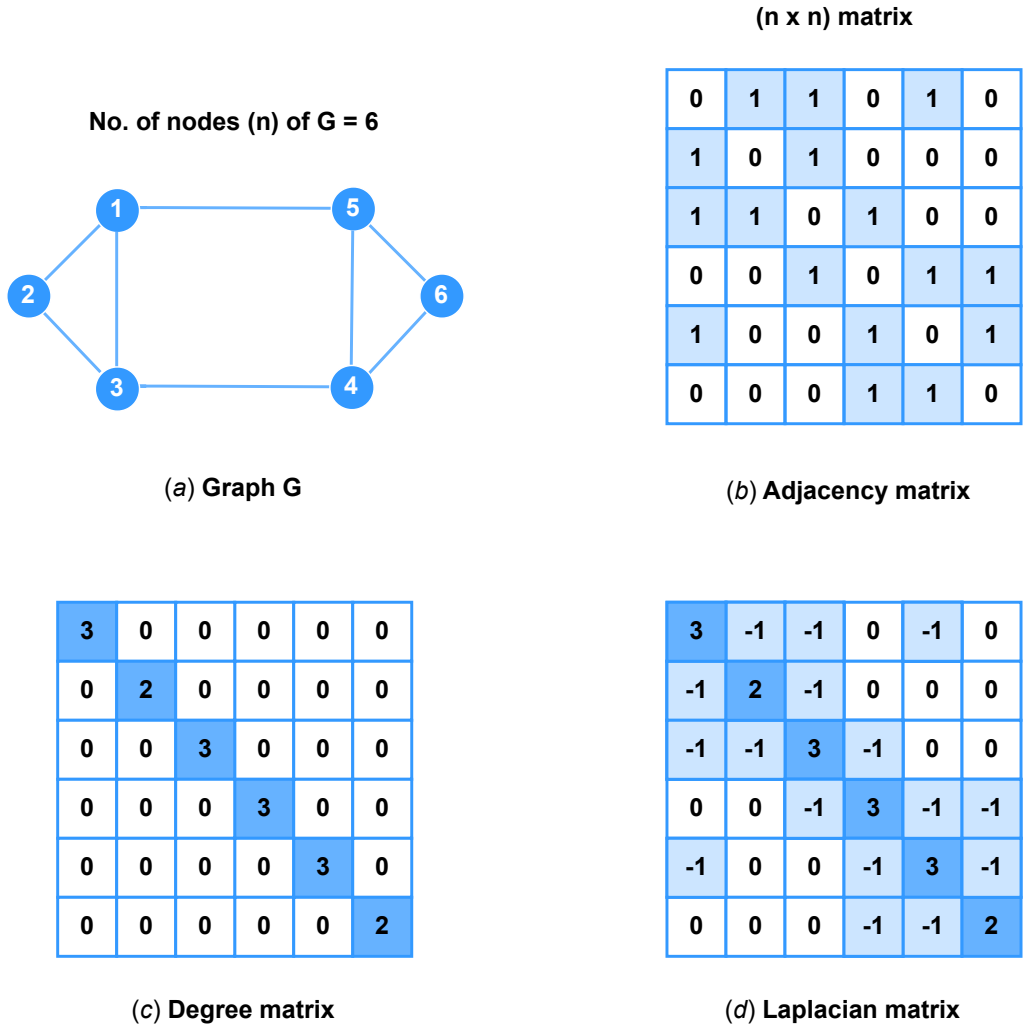


Figure 2.13: Steps involved in computing the Laplacain matrix from a graph. (a) An example connected and undirected graph G. (b) Adjacency matrix A based on the connection of vertices and edges of G. (c) degree matrix D computed from the adjacency matrix, and (d) Laplacian matrix L computed as, $L = D - A$.

[2]http://snap.stanford.edu/class/cs224w-2019/slides/05-spectral.pdf

The spectral bit of the Laplacian matrix focuses on analyzing the eigenvalues and eigenvectors of a graph as it can provide insights into the structural properties of the graph. For example, the second smallest eigenvalue of the Laplacian matrix, known as the algebraic connectivity or the Fiedler value, provides valuable insights into the connectivity and structural properties of a graph, facilitating the understanding of network behavior, partitioning, synchronization, and resilience [Mohar et al., 1991] [Mohar, 1997].

The (Figure 2.14) shows the eigen representation(i.e. the eigenvalues and eigenvectors) of the Laplacian matrix L in (Figure 2.13d). In this example the Laplacian matrix for a 6-vertex graph is a symmetric matrix of size 6×6, denoted as L. To describe the eigen representation of a 6x6 Laplacian matrix, we consider the eigenvalues and eigenvectors associated with the matrix. The Laplacian matrix has six eigenvalues, denoted as $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$, and λ_6. These eigenvalues are real and non-negative, with the smallest eigenvalue being zero [Von Luxburg, 2007]. For each eigenvalue, there is a corresponding eigenvector. In this case of a 6×6 Laplacian matrix, there are six eigenvectors, denoted as u_1, u_2, u_3, u_4, u_5, and u_6. These eigenvectors are real-valued and mutually orthogonal. In the example, each eigenvector contains 6 values. The eigenvectors are 6-dimensional vectors corresponding to the 6 vertices of the graph represented by the Laplacian matrix. Each value in the eigenvector represents the contribution or weight of the corresponding vertex in the particular eigenvector. So, in this specific case, the eigenvectors are 6-dimensional vectors, with each vector having 6 values.

The eigenvectors associated with the smallest non-zero eigenvalues ($\lambda_2, \lambda_3, \lambda_4, \lambda_5$, and λ_6) capture information about the graph's connectivity, community structure, and partitioning [Mohar et al., 1991]. In particular, the second smallest eigenvalue (λ_2), known as the Fiedler value, and its corresponding eigenvector (u_2), called the Fiedler vector, are of special interest [Chung, 1997]. The Fiedler vector provides insights into graph partitioning and can be used to divide this graph into two parts based on the sign of its entries. The eigenvector associated with the zero eigenvalue (λ_1) is called the constant vector $\mathbb{1}$ [Von Luxburg, 2007]. It is a special eigenvector representing the uniform distribution of values across the vertices of the graph.

Spectral graph theory has numerous applications across various disciplines. Spectral graph theory is extensively used in graph partitioning and clustering algorithms [Chung, 1997], image segmentation [Boykov and Funka-Lea, 2006], community detection in social networks [Fortunato, 2010], network analysis and visualization [Seary and Richards, 2003], graph embeddings and dimensionality reduction [Yan et al., 2006], spectral clustering [Von Luxburg, 2007] and many more.

2.3 Spectral clustering

Spectral clustering is a popular technique in data analysis and machine learning that leverages the principles of spectral graph theory. It is used to partition data points into clusters based on their similarity, considering the underlying structure of the data represented as a graph. The work of [Von Luxburg, 2007] provides an introduction to spectral clustering. In this section, we are going to see in detail the various steps involved in spectral clustering and its challenges.

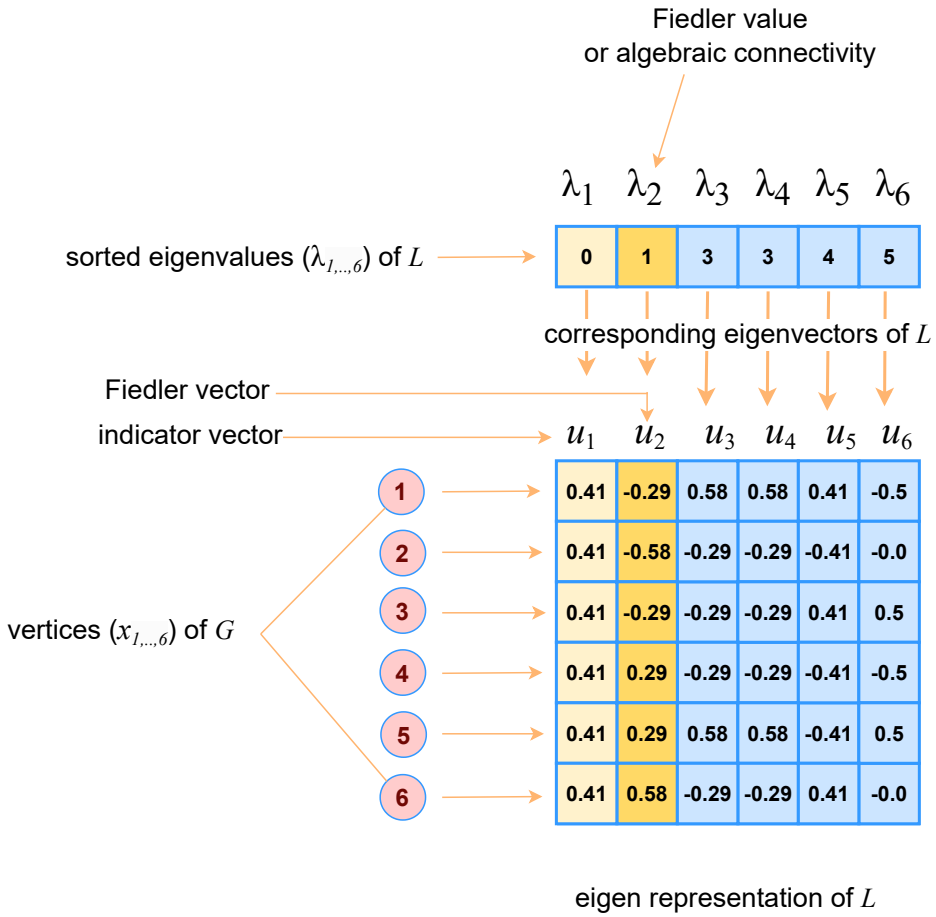


Figure 2.14: Eigen representation of the Laplacian matrix L of size 6×6 of the graph G in (Figure 2.13). The figure shows the six eigenvalues $(\lambda_1, ..., \lambda_6)$ sorted in ascending order along with its corresponding eigenvectors$(u_1, ..., u_6)$. The eigenvectors are 6-dimensional vectors corresponding to the 6 vertices of the graph represented by the Laplacian matrix. See text for more details.

2.3.1 Similarity measures

In spectral clustering, the first step is to define a similarity measure or similarity function between data points. The similarity measure quantifies how similar or related two data points are, and it plays a fundamental role in constructing the similarity graph. Common similarity measures include Gaussian kernel similarity, cosine similarity, Euclidean distance, and others. The choice of a similarity measure depends on the nature of the data and the specific problem at hand. For example, in a dataset of points in a high-dimensional space, the Euclidean distance between two points can be used as a similarity measure. Alternatively, in a text dataset, the cosine similarity between two document vectors may be employed.

Gaussian kernel - The Gaussian kernel, also known as the radial basis function (RBF) kernel, is defined by the Gaussian distribution function, which assigns a similarity value to each pair of data points based on their Euclidean distance. Thereby, it quantifies the similarity between data points by assigning higher values to data points that are closer together and lower values to data points that are further apart. The

similarity W_{ij} between data points i and j can be calculated using the formula for the Gaussian kernel function, given as:

$$W_{ij} = exp(\frac{-||x_i - x_j||^2}{2\sigma^2}) \tag{2.2}$$

where x_i and x_j are the feature vectors of data points i and j, $||x_i - x_j||$ represents the Euclidean distance between the data points i and j, and σ^2 is the variance parameter of the Gaussian kernel which determines the extent to which nearby data points influence each other(i.e. σ controls the width of the neighborhoods) [Von Luxburg, 2007].

The width of the Gaussian kernel is determined by the value of σ. A smaller value of σ results in a narrow kernel, where only nearby points have significant similarities. Conversely, a larger value of σ leads to a wider kernel, where points further apart still exhibit some level of similarity. The kernel enables modeling of non-linear relationships and enables the clustering algorithm to consider the density and proximity of data points [Bishop and Nasrabadi, 2006].

Cosine similarity - Cosine similarity is commonly used when the data points represent documents or text, where the feature vectors represent the word frequencies or some other kind of word representations. The cosine similarity measures the cosine of the angle between two feature vectors and is suitable for capturing the semantic similarity between documents [Salton, 1983]. The cosine similarity between data points i and j is given by:

$$cosine\ similarity = S_c(x_i, x_j) := cos(\theta) = \frac{(x_i \cdot x_j)}{(||x_i|| * ||x_j||)} \tag{2.3}$$

where $S_c(x_i, x_j)$represents the cosine similarity between two feature vectors, x_i and x_j. The numerator of the cosine similarity formula, $(x_i \cdot x_j)$, represents the dot product of vectors x_i and x_j. The dot product calculates the sum of the products of the corresponding elements of the two vectors. The denominator $((||x_i|| * ||x_j||))$, represents the product of the Euclidean norms (or magnitudes) of the two vectors, denoted as $||x_i||$ and $||x_j||$. By dividing the dot product by the product of the vector magnitudes, the cosine similarity formula normalizes the similarity measure, allowing comparisons between vectors of different lengths.

The cosine similarity ranges from -1 to 1, where a value of 1 indicates that the vectors are identical, 0 indicates no similarity, and -1 indicates complete dissimilarity.

Graph-based similarity measures - These measures are based on the concept of graph theory and leverage the connectivity of data points to determine their similarities. The adjacency matrix section 2.1.2 and the Laplacian matrix section 2.2 of a graph are two commonly used graph-based similarity measures in spectral clustering.

2.3.2 Different similarity graph construction

Once the similarity measure is defined, the next step is to construct a similarity graph based on the pairwise similarities between data points. The similarity graph is

represented as an adjacency matrix, where each entry $A[i, j]$ indicates the similarity between data points i and j. Here, we will discuss some of the different techniques used to construct a similarity graph in spectral clustering [Von Luxburg, 2007].

Epsilon-neighborhood graph - In an ϵ-neighborhood graph each pair of data points (x_i, x_j) is connected if their distance is below the given threshold ϵ. In other words, an edge is formed between x_i and x_j if $||x_i - x_j|| \leq \epsilon$, where $||x_i - x_j||$ denotes the distance metric. (Figure 2.15c) shows an example of an ϵ-neighborhood graph with $\epsilon = 0.5$.

k-nearest neighbors graph - For each data point x_i, the k-nearest neighbors graph connects it to its k nearest neighbors based on the calculated pairwise distances. This involves finding the k data points with the shortest distances to x_i. An edge is then formed between x_i and each of its k nearest neighbors, representing the local connectivity in the graph. A variation of the k-nearest neighbors graph is the mutual k-nearest neighbors graph. To construct the mutual k-NN graph, a mutual nearest neighbor criterion is applied. This means that two data points x_i and x_j are considered mutual neighbors if both are among the k nearest neighbors of each other. In other words, x_i is a mutual neighbor of x_j if x_j is one of the k nearest neighbors of x_i, and vice versa. (Figure 2.15b) shows an example of k-nearest neighbors graph with k = 4.

Fully connected graph The fully connected graph is the simplest similarity graph construction technique, where every pair of data points is connected with an edge, resulting in a complete graph. In this graph, all data points are considered similar to each other, and there are no constraints on the edge connections based on distances or thresholds. However, the similarity between pairs of data points is distinguished by weighting the edges using a similarity measure. (Figure 2.15d) shows an example of a fully connected graph constructed using the Gaussian kernel with $\sigma = 0.5$. Constructing a fully connected graph can be computationally expensive, especially for large datasets.

2.3.3 Graph Laplacian choices and their properties

After constructing the similarity matrix using one of the similarity graph construction techniques, we need to convert the similarity matrix to a Laplacian matrix. The Laplacian matrix is the primary tool for spectral clustering, because of its many useful properties. There are different Laplacian matrices [Von Luxburg, 2007], and we will discuss their properties respectively. When we talk about the eigenvector of a matrix here, the constant vector $\mathbb{1}$ and a multiple $a.\mathbb{1}$ for some $a \neq 0$ are considered as the same eigenvectors. This means that if a matrix has an eigenvalue λ with a corresponding eigenvector u, then any scalar multiple of u (such as $a.u$ for some nonzero scalar a) is also considered as an eigenvector associated with the same eigenvalue λ. Essentially, the eigenvectors are considered equivalent as long as they are scalar multiples of each other. Eigenvalues will always be ordered increasingly, respecting multiplicities. Meaning that when listing the eigenvalues of a matrix in ascending order, any eigenvalue that has multiplicity (repeated multiple times) will be listed accordingly with its respective multiplicity count. For example, if an eigenvalue λ occurs three times, it will appear three times in the ordered list of

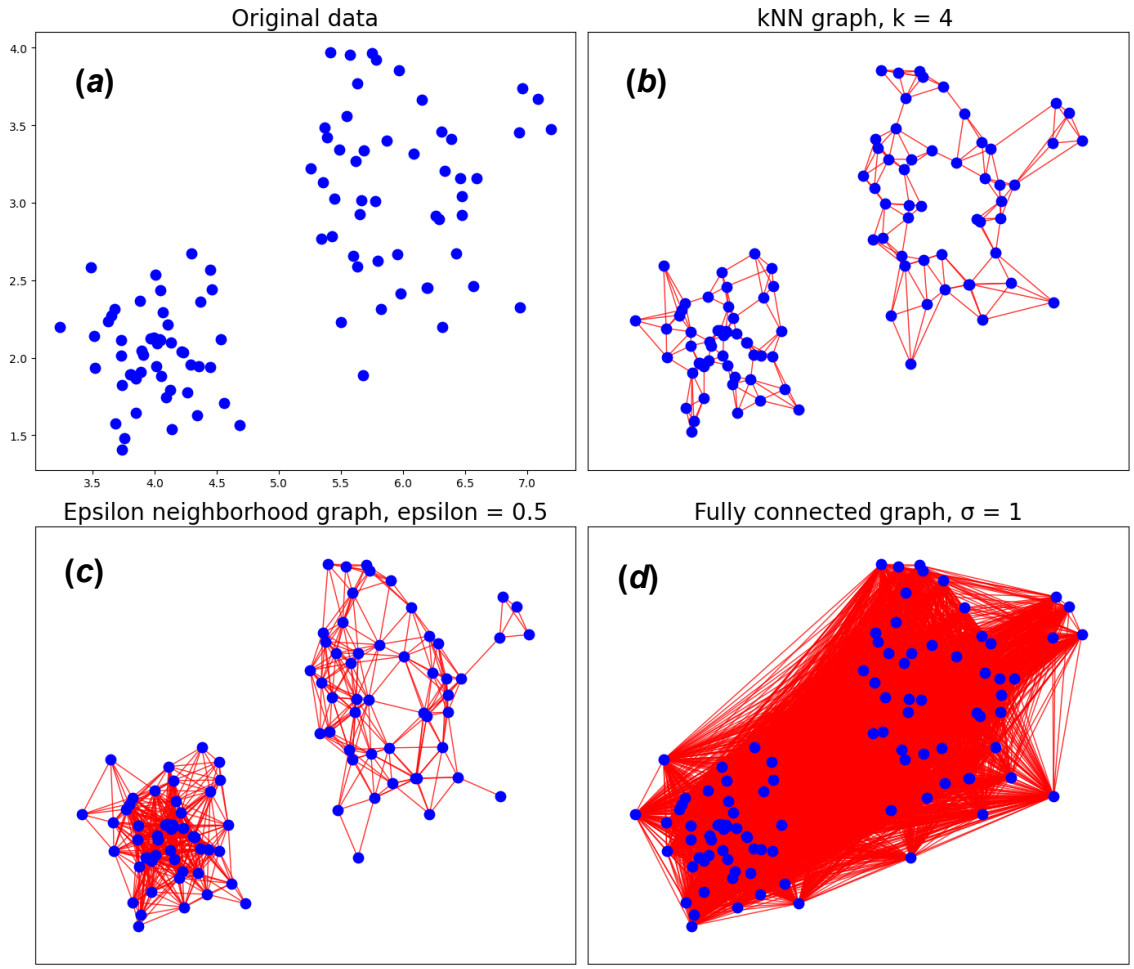


Figure 2.15: Different similarity graph construction from a sample dataset. (a) Sample dataset with 100 data points that are randomly distributed, which can be visually partitioned into two groups (bottom left and the top right). The parameters for the different graph construction techniques are carefully chosen to replicate the partitioning. (b) kNN graph constructed from the sample with k = 4, (c) ϵ-neighborhood graph constructed from the sample with $\epsilon = 0.5$, and (d) the fully connected graph with $\sigma = 1$. See (Appendix A.3) for graphs with different parameters.

eigenvalues. By the first k eigenvectors, we refer to the eigenvectors corresponding to the k-smallest eigenvalues. The first k eigenvectors often represent the most informative and significant eigenvectors in spectral clustering.

The unnormalized graph Laplacian - We refer the readers to section (2.2) for the derivation and definition of the unnormalized Laplacian matrix. An overview of many graph Laplacian properties can be found in the work of [Mohar et al., 1991; Mohar, 1997]. The following proposition summarizes the most important facts needed for spectral clustering [Von Luxburg, 2007].

Proposition 1 (Properties of L) *The matrix L satisfies the following properties:*

1. For every vector $f \in \mathbb{R}^n$ *we have*

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

2. L is symmetric and positive semi-definite.

3. The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector $\mathbb{1}$*.*

4. L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$*.*

Proposition 2 (Number of connected components and the spectrum of L**)** *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected components* $A_1, ..., A_k$ *in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors* $\mathbb{1}A_1, ..., \mathbb{1}A_k$ *of those components.*

The proof of these propositions and explanations can be found in (Appendix A.1.1). The unnormalized Graph Laplacian, despite its simplicity, has proven to be effective in capturing the local structure and connectivity of graphs in spectral clustering. However, it does not consider the scale or normalization factors of the graph, which may affect the performance in certain scenarios. The normalization factors refer to the scaling or normalization of the Graph Laplacian to consider the scale and magnitude of the graph's data or weights. When using the unnormalized Graph Laplacian, the original values of the graph's edge weights or vertex degrees are directly used in the Laplacian computation without any scaling. Therefore, other variants of graph Laplacians, such as the normalized Laplacians, have been developed to address these limitations.

The normalized graph Laplacians - The normalized graph Laplacians [Chung, 1997] are a variant of the graph Laplacian matrix, which is derived from the adjacency matrix and degree matrix of a graph. The normalized graph Laplacian can be computed in different ways, depending on the specific normalization scheme used. Two commonly employed normalization methods are random walk normalization and symmetric normalization [Von Luxburg, 2007].

1. Random walk normalized Laplacian: It incorporates the normalization factor based on the concept of random walks on the graph. The random walk normalized Laplacian is denoted as L_{rw} and is defined as:

$$L_{rw} := D^{-1}L = I - D^{-1}W \tag{2.4}$$

where L_{rw} is the random walk normalized Laplacian matrix, D is the degree matrix, L is the unnormalized Laplacian matrix, I is the identity matrix, and W is the weighted adjacency matrix of a graph. In the random walk normalized Laplacian, the edge weights of the graph are normalized by the corresponding node degrees. The motivation behind this normalization is to interpret the graph as representing a random walk process on the graph.

2. Symmetric normalized Laplacian: It is a symmetric matrix denoted as L_{sym} and is defined as:

$$L_{sym} := D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2} \tag{2.5}$$

where L_{sym} is the symmetric normalized Laplacian, D is the degree matrix, W is the adjacency matrix, L is the unnormalized graph Laplacian matrix, I is the identity matrix, and $D^{-1/2}$ denotes the matrix square root of the inverse of the degree matrix of a graph. In the symmetric normalized Laplacian, the edge weights of the graph are scaled by the inverse square root of the corresponding node degrees. This normalization is designed to ensure that the resulting Laplacian matrix is symmetric.

The following propositions summarize the properties of the normalized Laplacians L_{rw} and L_{sym} [Von Luxburg, 2007].

Proposition 3 (Properties of L_{sym} and L_{rw}) *The normalized Laplacians satisfy the following properties:*

1. For every $f \in \mathbb{R}^n$ we have

$$f' L_{sym} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2 .$$

2. λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2}u$.

3. λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ and u solve the generalized eigenproblem $Lu = \lambda Du$.

4. 0 is an eigenvalue of L_{rw} with the constant one vector $\mathbb{1}$ as eigenvector. 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\mathbb{1}$.

5. L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq ... \leq \lambda_n$.

The multiplicity of the eigenvalue 0 of the normalized graph Laplacian is related to the number of connected components, as it is for the unnormalized graph Laplacian:

Proposition 4 (Number of connected components and spectra of L_{sym} and L_{rw}) *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of both L_{rw} and L_{sym} equals the number of connected components $A_1, ..., A_k$ in the graph. For L_{rw}, the eigenspace of 0 is spanned by the indicator vectors $\mathbb{1}_{A_i}$ of those components. For L_{sym}, the eigenspace of 0 is spanned by the vectors $D^{1/2}\mathbb{1}_{A_i}$.*

The proof of these propositions and explanations can be found in (A.1.2). The main difference between these two normalization processes is in the scaling of the edge weights in the graph. The random walk normalized Laplacian normalizes each edge weight by the degree of its corresponding source vertex. On the other hand, the symmetric normalized Laplacian scales each edge weight by the product of the square root of the degrees of both the source and target vertices.

2.3.4 Eigendecomposition

Eigendecomposition, also known as diagonalization, is a fundamental matrix decomposition technique that breaks down a square matrix into its constituent eigenvalues

and eigenvectors. An eigenvalue represents a scalar that quantifies how the corresponding eigenvector is stretched or shrunk by the linear transformation [Lang, 2012] defined by the matrix. The eigenvectors are the directions along which the linear transformation only stretches or shrinks the vectors without changing their direction. For spectral clustering, we perform two steps in the eigendecomposition phase:

1. Compute eigenvalues and eigenvectors of the Laplacian matrix: The first step involved in eigendecomposition is to compute the eigenvalues and eigenvectors of the Laplacian matrix (L). The Laplacian matrix captures the pairwise relationships between data points and reflects the underlying structure of the data. Eigenvalues and eigenvectors are calculated by solving the eigenvalue problem for the Laplacian matrix. The eigenvalues represent the spectral decomposition of the Laplacian matrix, while the eigenvectors correspond to the directions associated with each eigenvalue. The eigenvalues are arranged in ascending order, denoted as $\lambda_1, \lambda_2, ..., \lambda_n$, where n is the number of data points. The eigenvectors are represented as, $u_1, u_2, ..., u_n$, with each eigenvector corresponding to an eigenvalue. Mathematically, the eigenvalue-eigenvector pairs of the Laplacian matrix are defined as follows:

Given a graph with Laplacian matrix L, an eigenvector u and its corresponding eigenvalue λ satisfy the equation:

$$Lu = \lambda u \tag{2.6}$$

Eigenvectors - Eigenvectors represent different partitions or clusters of nodes in the graph. Each eigenvector corresponds to a specific eigenvalue and provides information about the connectivity and relationships between nodes. The values in an eigenvector indicate the membership or assignment of each node to a particular cluster.

Eigenvalues - Eigenvalues quantify the significance of the corresponding eigenvectors. They indicate the amount of variation explained by each eigenvector and can be used to determine the number of clusters or partitions in the graph. Eigenvalues associated with smaller magnitudes represent cohesive clusters, while larger eigenvalues indicate less structured or more homogeneous regions. The first few eigenvalues in ascending order, often referred to as the dominant eigenvalues, are particularly important as they capture the most significant structure and can guide the selection of the number of clusters in spectral clustering.

2. Map data points to lower-dimensional representation: Once the eigenvalues and eigenvectors are obtained, the next step is to map each data point to a lower-dimensional representation based on one or more eigenvectors. The number of eigenvectors chosen determines the dimensionality of the lower-dimensional space and also can be representative of the choice of number of clusters in the dataset. This lower-dimensional representation captures the important structural information of the data, which can then be used for clustering.

2.3.5 Clustering

The final step in spectral clustering is the actual clustering of the data points based on the lower-dimensional representation. After computing the eigenvalues

and eigenvectors of the Laplacian matrix and mapping the data points to the lower-dimensional space, the next step is to apply a clustering algorithm to group the data points into distinct clusters.

Clustering is a fundamental technique in data analysis and machine learning that aims to group similar data points together based on their intrinsic characteristics. It is an unsupervised learning method, meaning that it does not rely on prior class labels or target variables. Clustering algorithms organize data into clusters or groups such that the data points within each cluster are more similar to each other than to those in other clusters. The goal is to discover hidden patterns, structures, or relationships in the data without any prior knowledge.

Clustering algorithms can be broadly categorized into partitioning-based (e.g., k-mean), hierarchical (e.g., Agglomerative Hierarchical Clustering), density-based (e.g., DBSCAN), and model-based approaches (e.g., Gaussian Mixture Models). Each algorithm has its own strengths and weaknesses and is suitable for different types of data and clustering objectives. (Figure 2.16) shows a comparison of various clustering algorithms that are applied to different toy datasets (a through f) with pre-defined shapes in each row of the image. In the (Figure 2.16), we see that all datasets give an impression of visually well separated clusters except for the dataset (f) in the last row where the data is homogeneous, which leads to poor cluster performance.

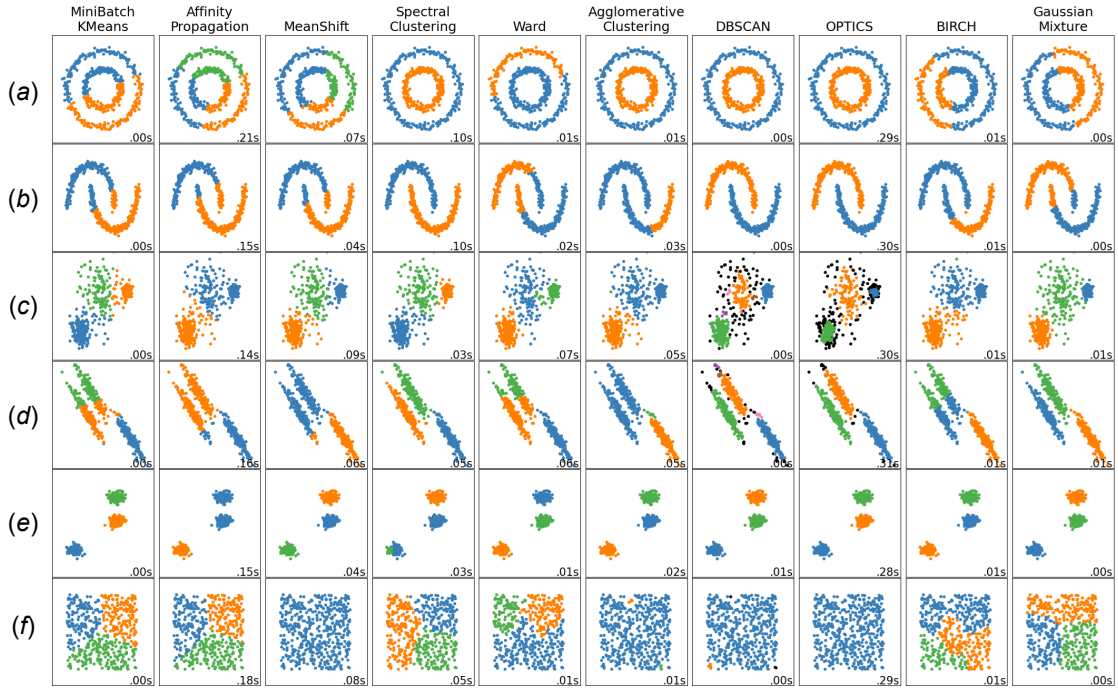


Figure 2.16: Comparison of different clustering algorithms on different toy datasets with pre-defined shapes in each row (a) through (f).

Clustering is widely used in various domains, including pattern recognition [Duda et al., 2001], image analysis [Szeliski, 2022], document classification [Mogotsi, 2010], customer segmentation [Jain et al., 1999], anomaly detection [Chandola et al., 2009],

[3]https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

and recommendation systems [Ricci et al., 2010]. Clustering can help in understanding the underlying data distribution, making predictions, and supporting decision-making processes. It is important to note that clustering is an exploratory technique, and the interpretation of the resulting clusters relies on domain knowledge and context. It is also essential to understand the limitations of clustering algorithms, such as sensitivity to initialization, sensitivity to noise and outliers, and difficulty in handling high-dimensional data.

The specific clustering algorithm used in the final step of spectral clustering can vary depending on the application and requirements. One common choice is the k-means algorithm, which partitions the data points into k clusters based on their proximity in the lower-dimensional space. Other clustering algorithms, such as DBSCAN, hierarchical clustering, and many more, can also be employed to perform the clustering step. In this thesis, we use k-means clustering as the third step for grouping data points that are represented in low-dimensional space through eigendecomposition to form clusters.

K-means clustering: K-means clustering is one of the most widely used unsupervised machine learning algorithms for partitioning data into groups or clusters based on similarity. It is a simple and intuitive algorithm that aims to find K centroids representing the cluster centers and assigns each data point to its nearest centroid. K-means clustering aims to minimize the within-cluster sum of squares, also known as the distortion function. The algorithm iteratively refines the clusters by minimizing the distances between the data points and their assigned centroids. Once convergence is reached, each data point belongs to a specific cluster.

K-means clustering algorithm:

Algorithm 1 K-means clustering algorithm

Require: K, number of clusters; D, a data set of N points.
Ensure: A set of K clusters.

1. Initialization.
2. **repeat**
3. **for** each point p in D **do**
4. find the nearest center and assign p to the corresponding cluster.
5. **end for**
6. update clusters by calculating new centers using mean of the members.
7. **until** stop-iteration criteria satisfied
8. **return** clustering results.

Given a dataset $D$with N data points and the desired number of clusters K, the k-means clustering algorithm proceeds as follows:

Initialization: Randomly initialize K cluster centroids in the feature space. These centroids serve as the initial representatives of the clusters.
Assignment: For each data point, calculate its distance to each centroid using a

[4]https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8 _425

distance metric, typically Euclidean distance. Assign each data point to the cluster with the nearest centroid.
Update: Recalculate the centroids of each cluster by computing the mean of all the data points assigned to that cluster. This step ensures that the centroids are representative of the cluster members.
Iteration: Repeat the steps of assignment and update until convergence, which occurs when the assignments and centroid updates no longer change significantly, or a maximum number of iterations is reached.
Final Result: The result of the k-means clustering algorithm is K clusters, each containing data points that are most similar to each other based on their distance to the cluster centroid. (Figure. 2.17) is a pictorial representation of different transitional steps of the k-means clustering algorithm until the centroids converge.

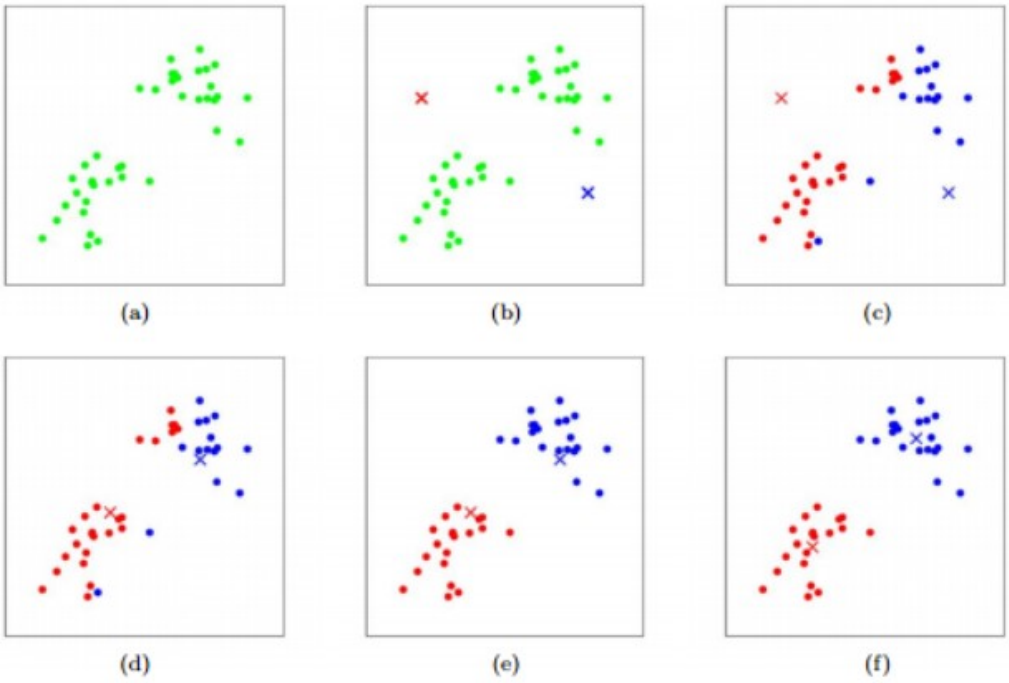


Figure 2.17: K-means algorithm. Training examples are shown as dots, and cluster centroids are shown as crosses. (a) Original dataset. (b) Random initial cluster centroids. (c-f) Illustration of running two iterations of k-means. In each iteration, there is an assignment of each training example to the closest cluster centroid (shown by "painting" the training examples the same color as the cluster centroid to which it is assigned); then each cluster centroid is moved to the mean of the points assigned to it.

Advantages of k-means: The k-means algorithm has several advantages that contribute to its popularity and widespread use in various applications:

- **Simplicity and availability:** K-means is a simple and intuitive algorithm that is easy to understand and implement. It does not require complex mathematical computations or extensive parameter tuning. K-means is implemented in various

[5]https://stanford.edu/ cpiech/cs221/handouts/kmeans.html

software libraries and packages, making it readily available and easy to use. It is widely supported in popular programming languages such as Python (scikit-learn), R, and MATLAB.

- **Efficiency and parallelizability:** The algorithm is computationally efficient and scales well with large datasets. It has a linear time complexity with respect to the number of data points, making it suitable for handling big data scenarios. The k-means algorithm can be parallelized, allowing for faster execution on multi-core or distributed computing systems. This enables efficient clustering of large-scale datasets.
- **Effectiveness:** Despite its simplicity, k-means often produces satisfactory results in many real-world scenarios. It can effectively partition data into clusters based on similarity, revealing underlying patterns and structures in the data.
- **Interpretability:** The resulting clusters and centroids in k-means are easily interpretable. Each cluster represents a distinct group of data points, and the centroids provide meaningful representations of the cluster centers.
- **Versatility:** K-means can be applied to various types of data, including numerical, continuous, categorical, and mixed data. Thus, making it a versatile clustering algorithm.

Limitation of k-means: While the k-means algorithm has many advantages, it also has several limitations that should be considered when applying it to clustering tasks:

- **Predefined number of clusters:** One of the main limitations of k-means is that it requires the number of clusters (k) to be specified in advance. However, determining the optimal number of clusters is often a challenging task and may require prior knowledge or cross validation.
- **Sensitive to initial centroid positions:** The performance of k-means can be highly influenced by the initial positions of the centroids. It is possible for the algorithm to converge to suboptimal solutions if the initial centroids are poorly chosen. Multiple runs with different initializations may be required to mitigate this issue.
- **Assumption of spherical clusters and similar sizes:** K-means assumes that the clusters have a spherical shape and similar sizes. However, in real-world data, the clusters may have complex shapes and different variances. As a result, k-means may not perform well on datasets with irregular or overlapping clusters.
- **Sensitivity to outliers:** K-means is sensitive to outliers, which are data points that deviate significantly from the majority of the data. Outliers can distort the cluster centroids and affect the clustering results. One way to deal with outliers is to remove them from the dataset before running the k-means algorithm, or we could use variations of k-means, such as the outlier-robust k-means clustering [Olukanmi and Twala, 2017], that is designed to be less affected by outliers.

2.4 Challenges of spectral clustering

Spectral clustering is a powerful unsupervised machine learning technique that can effectively cluster complex and non-linear data. However, spectral clustering also

presents some challenges that can impact its performance and practical applicability. In this subsection, we discuss some of the main challenges of spectral clustering.

2.4.1 Choice of similarity measure

Spectral clustering relies on constructing an affinity matrix if there is no initial graph structure that captures the pairwise similarities between data points. The choice of similarity measure determines how these similarities are calculated. Different similarity measures capture different aspects of the data, and selecting an appropriate measure is essential for obtaining meaningful clusters. There is no universal similarity measure that works optimally for all scenarios. The choice of similarity measure depends on the nature of the data, the underlying cluster structure, and the desired clustering outcome. It requires careful consideration and domain knowledge. For example, Euclidean distance works well when the data points are represented in a continuous feature space and are relatively dense. However, it can be sensitive to outliers and might not perform optimally in high-dimensional spaces (curse of dimensionality). The Gaussian kernel can be effective in capturing non-linear relationships between data points, and is more robust to outliers compared to the Euclidean distance. However, selecting an appropriate width (variance) parameter for the Gaussian kernel is critical. Cosine similarity is particularly suitable for high-dimensional and sparse data, such as text data in document clustering. It is commonly used in text mining and information retrieval tasks.

2.4.2 Scalability

Scalability is a critical challenge when applying spectral clustering to large-scale datasets. The computation of eigenvectors for the affinity matrix requires $O(n^3)$ operations, where n represents the number of data points. As the dataset size increases, the computational requirements and memory usage of spectral clustering grow significantly, making traditional algorithms impractical for large datasets [Mahoney and Drineas, 2009].

The high computational complexity arises from the computation of the affinity matrix, eigendecomposition, and final clustering step. Additionally, storing the affinity matrix for large datasets can be memory-intensive, requiring memory-efficient techniques like sparse matrix representations or approximation methods [Mahoney and Drineas, 2009].

To address scalability challenges, various techniques have been proposed. Approximation methods, subsampling, and parallel computing aim to reduce computational and memory requirements while preserving clustering performance [Mahoney and Drineas, 2009].

[Yang et al., 2012] recognize the scalability limitations of traditional spectral clustering algorithms for large datasets and propose a scalable algorithm based on random graph embedding to approximate the original data graph and enable efficient spectral clustering.

2.4.3 Which is the preferred Graph Laplacian?

The choice of the preferred Graph Laplacian presents challenges in spectral clustering, as it directly influences the clustering performance and interpretability of results [Chung, 1997]. The choice of Laplacian is influenced by the degree distribution of the similarity graph. When the graph exhibits a regular and uniform degree distribution, the Laplacians, whether unnormalized or normalized, yield similar results. However, if the graph's degree distribution is highly diverse, the Laplacians differ significantly, leading to distinct clustering outcomes [Von Luxburg, 2007].

To choose a specific type of Graph Laplacian, certain considerations can be made. The unnormalized Laplacian, introduced by Chung [Chung, 1997], is the most straightforward definition and incorporates node degrees in its computation. However, it lacks normalization factors, which might affect the scaling of eigenvalues and eigenvectors. There are several compelling arguments in favor of using normalized spectral clustering over unnormalized spectral clustering. First, the normalized Laplacian accounts for the varying degrees of vertices in the graph, making it more robust in handling datasets with non-uniform degree distributions [Von Luxburg, 2007]. Second, the normalized Laplacian ensures that the eigenvalues and eigenvectors are independent of the graph size, providing a more stable and consistent representation of the data structure [Von Luxburg, 2007].

Within the normalized case as described by [Von Luxburg, 2007], using the eigenvectors of the random walk Laplacian (L_{rw} according to [Shi and Malik, 2000]) is recommended over those of the symmetric normalized Laplacian (L_{sym} according to [Ng et al., 2002]). The rationale behind this preference lies in the fact that the eigenvectors of L_{rw} represent cluster indicator vectors $\mathbb{1}A_i$, while the eigenvectors of L_{sym} are further multiplied with $D^{1/2}$, which could introduce unintended distortions or artifacts in the clustering results. Moreover, opting for L_{sym} does not offer any computational advantages compared to L_{rw}.

Ultimately, the choice of the preferred Graph Laplacian depends on the specific characteristics of the data, the similarity measure used in constructing the graph, and the desired clustering outcome. The appropriate Laplacian should be selected to best capture the underlying structure of the data and optimize the clustering performance.

2.4.4 Sensitivity to hyperparameters

Spectral clustering encounters challenges due to the sensitivity of hyperparameters, including the scaling parameter(e.g., ϵ-neighborhood graph), the number of eigenvectors (eigenvectors chosen from the eigendecomposition determines the dimensionality of the reduced space), the clustering algorithm (e.g., k-means) [Chung, 1997], and the choice of optimal number of clusters k. The selection of these hyperparameters is crucial for the clustering performance, but determining their optimal values can be difficult, as no universal guidelines exist for their selection. Instead, the appropriate parameter values depend on the dataset's characteristics, the desired clustering outcome, and the specific spectral clustering algorithm used. This lack of universal guidelines makes hyperparameter selection a challenging task that often requires sensitivity analysis Chung [1997].

To address these challenges and improve the robustness of the clustering results, several approaches have been proposed. Cross-validation is a widely used technique to estimate the model or algorithm's performance on unseen data, allowing the assessment of clustering quality for different hyperparameter settings [Arlot and Celisse, 2010]. Grid search and optimization algorithms systematically explore a range of hyperparameter values to find the combination that yields the best clustering performance. Optimization algorithms, such as gradient-based methods or evolutionary algorithms, can automatically search for optimal hyperparameter values based on specific optimization criteria [Bergstra and Bengio, 2012]. Furthermore, robust clustering techniques aim to minimize dependence on hyperparameters by incorporating additional mechanisms to handle uncertainty or variability in the data. These methods often integrate robust statistical techniques or Bayesian frameworks, reducing sensitivity to hyperparameters [Huang et al., 2005].

Various techniques have been proposed for estimating the optimal number of clusters. The elbow method involves plotting clustering performance against the number of clusters and identifying the "elbow" point, indicating a significant decrease in improvement [Thorndike, 1953]. Silhouette analysis evaluates clustering quality based on cohesion and separation, calculating a silhouette coefficient for each data point [Rousseeuw, 1987]. Stability-based approaches assess a clustering solution's robustness by perturbing data or algorithm parameters, identifying the number of clusters that yield stable results across perturbations [Ben-Hur and Guyon, 2003]. Gap statistics, spectral gap or eigengap heuristics, and information criteria have also been suggested to estimate the optimal number of clusters [Von Luxburg, 2007].

To address the challenge of the choice of clustering algorithm, researchers have proposed using various post-processing techniques to refine the initial cluster assignments obtained from spectral decomposition. These techniques, such as density-based clustering, hierarchical clustering, or consensus clustering, can better capture the inherent structure and resolve ambiguity in the cluster assignments [Von Luxburg et al., 2010]. Consensus clustering, for example, combines multiple clusterings obtained from different runs or variations of spectral clustering to improve stability and reliability [Monti et al., 2003]. Density-based approaches, like DBSCAN, can be employed as post-processing steps to handle overlapping clusters and improve cluster assignments [Sander et al., 1998].

2.4.5 Interpretability

Interpretability is an essential aspect of clustering algorithms, allowing users to comprehend and make sense of the clustering results. However, interpreting the clusters produced by spectral clustering can be challenging due to several reasons [1]. First, spectral clustering operates in a transformed space defined by the eigenvectors obtained from the spectral decomposition. The resulting eigenvectors may not have a direct correspondence with the original features of the data, making it difficult to understand the precise meaning or relevance of each eigenvector in the context of the underlying data [Von Luxburg, 2007].

Another challenge is the subjectivity in interpretation. The interpretability of spectral clustering often involves assigning semantic meaning to the clusters based on patterns

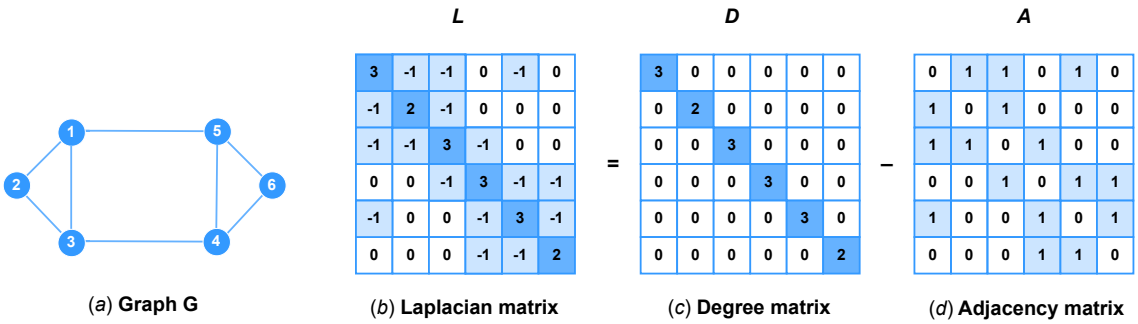
observed in the transformed space. However, different interpretations of the same clustering result may arise, leading to potential ambiguity or disagreement in the interpretation process [Von Luxburg, 2007].

To enhance the interpretability of the clustering results, several techniques have been proposed. Dimensionality reduction techniques, such as PCA (Principal Component Analysis) [Jolliffe, 2002] or t-SNE (t-Distributed Stochastic Neighbor Embedding) [Van der Maaten and Hinton, 2008], can be employed to reduce the dimensionality of the transformed space and visualize the clusters in a lower-dimensional space [1]. Cluster profiling aims to characterize and describe the clusters in a more interpretable manner by analyzing the attributes or features of data points within each cluster [Jain et al., 1999]. Involving domain experts in the interpretation process can also significantly enhance interpretability by providing domain-specific knowledge, validating the clustering results, and contributing insights that align with the context of the data [Caruana et al., 2015].

2.5 A use case for spectral clustering

In this section, we present a use case for spectral clustering. We will walk through an example of spectral clustering to demonstrate its steps and concepts. To maintain continuity and clarity, we will use the same example graph introduced earlier in (Figure 2.13a).

Laplacian matrix construction: In the first step of spectral clustering also called preprocessing, we construct a Laplacian matrix from a set of data points by a constructing a similarity graph using some similarity measure. In our example, we assume that the graph G has already been precomputed, as shown in (Figure 2.18a). As discussed in section 2.2, we need to construct various matrices to perform spectral clustering. To begin, we employ a binary similarity measure to create the weighted adjacency matrix (Figure 2.18d). Next, we proceed to construct the unnormalized Laplacian matrix (Figure 2.18b). This is achieved by subtracting the adjacency matrix from the degree matrix (Figure. 2.18c).



(a) Graph G (b) Laplacian matrix (c) Degree matrix (d) Adjacency matrix

Figure 2.18: Steps involved in computing the Laplacian matrix from a graph. (a) An example connected and undirected graph G. (b) Laplacian matrix L computed as, $L = D - A$. (c) degree matrix D computed from the adjacency matrix, and (d) Adjacency matrix A constructed using a binary similarity measure based on the connection of vertices and edges of G.

Eigendecomposition: The second step of spectral clustering involves eigendecomposition of the Laplacian matrix, which is performed in two stages. In the first stage, we compute the eigenvalues and eigenvectors of the Laplacian matrix, as illustrated in (Figure 2.19a). These eigenvalues and eigenvectors capture essential patterns and relationships within the data.

In the second stage of eigendecomposition, we transform the eigen representation of the Laplacian matrix into a lower-dimensional representation denoted as U, as shown in (Figure 2.19b). Here, we have chosen to reduce the dimensionality from 6 to 2 for this specific use case. The reason behind this choice is that the Fiedler vector, derived from the eigen representation, exhibits remarkable discriminative power. It can effectively partition the graph into distinct groups based on the signs of its values. The Fiedler vector serves as a valuable indicator of the underlying clusters in the data. By observing its behavior in a two-dimensional space, we can clearly distinguish the data points and establish meaningful partitions without the need to consider the remaining dimensions. This decision enhances the interpretability and simplicity of the clustering process.

Note that in the lower-dimensional representation, the similarity matrix of the original graph $S \in \mathbb{R}^{6\times6}$ is transformed to $U \in \mathbb{R}^{6\times2}$ and the data points $(x_i)_{i=1,\dots,6} \in \mathbb{R}^6$ is transformed to $(y_i)_{i=1,\dots,6} \in \mathbb{R}^2$. See algorithm (2) for unnormalized Laplacian.

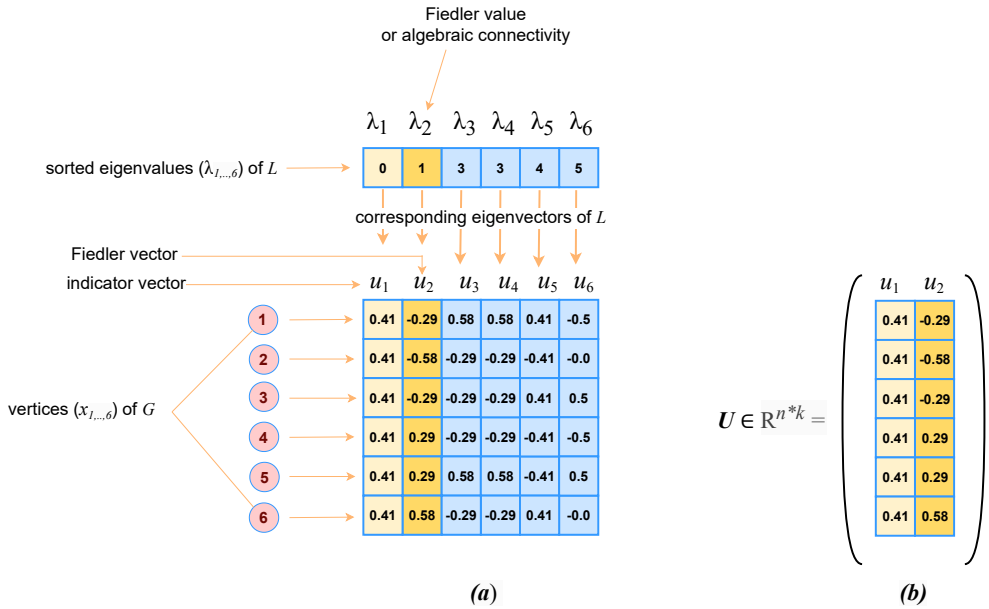


Figure 2.19: Eigen representation of the Laplacian matrix L of size 6×6 of the graph G. (a) shows the sorted eigenvalues and their corresponding eigenvectors of the Laplacian matrix, indicating the most important eigenvalues and eigenvectors of the graph. (b) shows the lower -dimensional representation of L as U.

Clustering: The third and final step of spectral clustering is to cluster the graph using a suitable clustering algorithm within the low-dimensional representation. In this example and within the scope of this thesis, we opt to employ the k-means clustering algorithm for this purpose.

Upon transforming the data into the reduced space, we find that the first dimension consists of an indicator vector with constant values, while the second dimension contains the Fiedler vector, which holds significant discriminatory power, as demonstrated in (Figure 2.20a).

With the reduced data representation now comprising two dimensions, we proceed with k-means clustering. This algorithm effectively partitions the data points into clusters by separating the values of the Fiedler vector into two groups (A and B) based on their signs, as depicted in (Figure 2.20b).

By leveraging the Fiedler vector's capacity to distinctly categorize data points, we successfully obtain two well-defined clusters in the reduced space. The k-means algorithm efficiently handles the clustering task, and the separation of values based on the Fiedler vector facilitates the formation of meaningful and cohesive clusters.

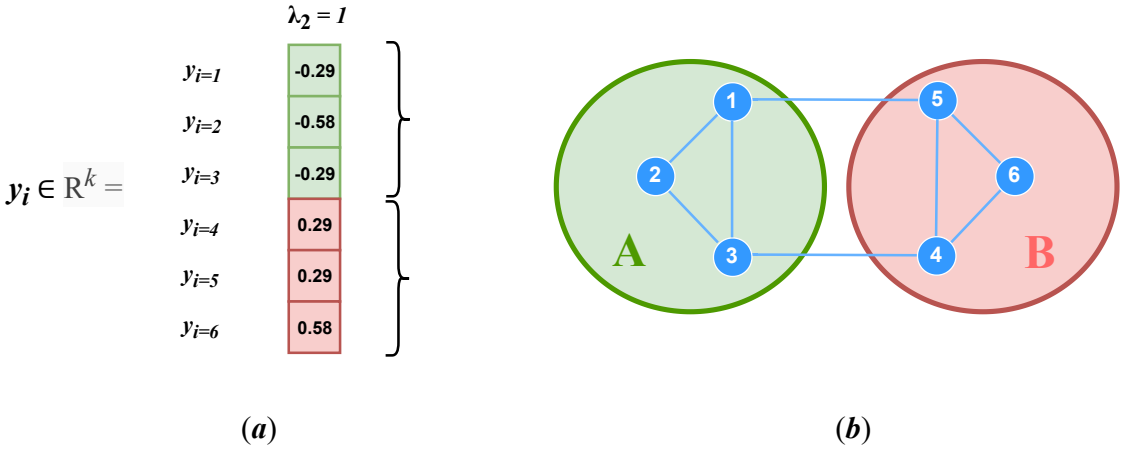


Figure 2.20: (a) Depicts the ability of the second smallest eigenvalue λ_2 and its corresponding eigenvector, the Fiedler vector in identifying clusters in the lower-dimensional representation. (b) Shows the clusters of G based on the signs of the values of the Fiedler vector.

2.6 Metric

In this section, we utilize a range of metrics to calculate distances and evaluate the efficacy of our spectral clustering framework. We begin by introducing diverse distance metrics for pairwise distance computations. In our framework, we use distance metric for constructing the adjacency matrix and different similarity graphs for constructing the weighted adjacency matrix (or similarity matrix). We then discuss clustering evaluation metrics, starting with internal cluster measures and subsequently exploring external measures.

2.6.1 Distance metric

To facilitate the foundation of spectral clustering, we judiciously employ a diverse set of distance metrics. These metrics, encapsulate distinct aspects of data relationships, thereby ensuring adaptability to the inherent characteristics of diverse datasets

[Jain and Dubes, 1988] [Xu and Wunsch, 2005]. The distance metrics used in our framework are as follows:

Euclidean distance: Euclidean distance between two points u and v in a multi-dimensional space is the straight-line distance between them [Xu and Wunsch, 2005]:

$$euclidean\_distance(u, v) = \sqrt{\sum_i (v_i - u_i)^2} \tag{2.7}$$

v_i and u_i are the i-th componenets of vectors v and u, respectively.

Squared Euclidean distance: Squared Euclidean distance is the square of the Euclidean distance between two points [Everitt et al., 2011]:

$$squared\_euclidean\_distance(u, v) = \sum_i (v_i - u_i)^2 \tag{2.8}$$

Cosine distance: Cosine distance between two vectors u and v is calculated as the cosine of the angle between them, representing their normalized similarity [Schutze et al., 2008]:

$$cosine\_distance(u, v) = 1 - \frac{u \cdot v}{||u|| \cdot ||v||} \tag{2.9}$$

where, $\cdot$ denotes the dot product between vectors u and v. $||u||$ and $||v||$ are the Euclidean norms of vectors u and v, which represent the lengths of vectors u and v in the Euclidean space respectively.

Bray-Curtis distance: Bray-Curtis distance between two vectors u and v is calculated as the sum of absolute differences divided by the sum of their absolute values [Bray and Curtis, 1957]:

$$bray\_curtis\_distance(u, v) = \frac{\sum_i |u_i - v_i|}{\sum_i |u_i| + \sum_i |v_i|} \tag{2.10}$$

Canberra distance: Canberra distance between two vectors u and v is calculated as the sum of absolute differences between corresponding elements, normalized by the sum of their absolute values [Lance and Williams, 1966]:

$$canberra\_distance(u, v) = \frac{\sum_i |u_i - v_i|}{\sum_i (|u_i| + |v_i|)} \tag{2.11}$$

Correlation distance: Correlation distance between two vectors u and v is calculated based on their Pearson correlation coefficient [Kaufman and Rousseeuw, 2009]:

$$correlation\_distance(u, v) = 1 - \frac{cov(u, v)}{std(u) \cdot std(v)} \tag{2.12}$$

where, $cov(u, v)$ is the covariance between vectors u and v. $std(u)$ and $std(v)$ are the standard deviations of vectors u and v, respectively.

The formula for covariance is:

$$cov(u, v) = \frac{1}{n-1} \sum_{i=1}^{n} (u_i - \bar{u})(v_i - \bar{v}) \tag{2.13}$$

Where n is the number of elements in the vectors. u_i and v_i are the elements of vectors u and v at index i, respectively. $\bar{u}$ and $\bar{v}$ are the means (average) of vectors u and v, respectively. This formula calculates the average of the product of the deviations of each element from the mean of their respective vectors. The division by $(n-1)$ is used for unbiased estimation.

The formula for the standard deviation is:

$$std(u) = \sqrt{\frac{\sum_{i=1}^{n} (u_i - \bar{u})^2}{n-1}} \tag{2.14}$$

City block distance (Manhattan distance): The city block distance, also known as the Manhattan distance [Jain and Dubes, 1988], between two points u and v in multi-dimensional space is calculated as the sum of the absolute differences between their corresponding coordinates:

$$city\_block\_distance(u, v) = \sum_{i} |v_i - u_i| \tag{2.15}$$

2.6.2 Internal cluster validity indices

To assess the clustering quality without relying on any external ground truth, we have selected three widely used internal measures, the Silhouette Score [Rousseeuw, 1987], Davies-Bouldin (DB) Score [Davies and Bouldin, 1979], and Calinski-Harabasz (CH) Score [Caliński and Harabasz, 1974].

Silhouette Score: The Silhouette Score computes the cohesion and separation of clusters by considering the average distance of each data point to its own cluster and the average distance to the nearest neighboring cluster. The Silhouette Score has a range of values from -1 to 1. A higher Silhouette Score indicates better-defined and well-separated clusters [Rousseeuw, 1987]. For each data point i, let:

$a(i)$ be the average distance of data point i to all other points within the same cluster. $b(i)$ be the minimum average distance of data point i to all points in any other cluster. The silhouette score for data point i is given by:

$$\text{Silhouette}(i) = \frac{b(i) - a(i)}{\max(b(i), a(i))} \tag{2.16}$$

The overall silhouette score is the average of all the individual silhouette scores across all data points.

Davies-Bouldin (DB) Score: The DB Score evaluates the compactness and separation of clusters by measuring the average similarity between each cluster and its most similar cluster. The DB Score is a relative measure with no fixed range. It ranges

from a minimum of 0, and there is no predefined upper limit. Lower values indicate improved clustering performance, with values near zero indicating well-separated clusters [Davies and Bouldin, 1979].

The DB score is calculated as:

$$\text{DB Score} = \frac{1}{n}\sum[\max R_k] \tag{2.17}$$

where n be the number of clusters. C_k be the kth cluster. M_k be the centroid of cluster C_k. S_k be the average distance of all points in cluster C_k to the centroid M_k. R_k be the maximum of $(S_i + S_j)$ for all i and j, where i and j are different clusters.

Calinski-Harabasz (CH) Score: The CH Score assesses the ratio of between-cluster variance to within-cluster variance.Similar to the DB Score, the CH Score is also a relative measure and does not have a specific fixed range. It evaluates the compactness and separation of clusters, with higher CH Scores indicating better-defined clusters and a higher ratio of inter-cluster distance to intra-cluster distance [Caliński and Harabasz, 1974]. Let:

B be the between-cluster variance. W be the within-cluster variance. k be the number of clusters. N be the total number of data points. The CH score is given by:

$$\text{CH Score} = \frac{B}{W} \cdot \frac{N-k}{k-1} \tag{2.18}$$

Where, $B = \sum[n_k \cdot d(M_k, M)^2]$. $W = \sum[\sum[d(x, M_k)^2]]$, for each data point x in cluster C_k. n_k = number of data points in cluster C_k. M_k = centroid of cluster C_k. M = overall centroid (mean) of all data points in the dataset. $d(A, B)$ = Euclidean distance between points A and B.

2.6.3 External cluster validity indices

In addition to internal evaluation metrics, we have employed external evaluation measures to compare the clustering results to the external ground truth, making them suitable for datasets with known class labels. The selected external measures include Accuracy [Schutze et al., 2008], Normalized Mutual Information (NMI) [Strehl and Ghosh, 2003], Adjusted Rand Index (ARI), Rand Index (RI) [Hubert and Arabie, 1985], Homogeneity and Completeness [Rosenberg and Hirschberg, 2007].

Accuracy: Accuracy quantifies the ratio of data points correctly classified to the total number of data points, thereby gauging the agreement between the clustering outcomes and the actual class labels [Schutze et al., 2008].

$$\text{Accuracy} = \frac{\textit{Number of correctly classified data points}}{\textit{Total number of data points}} \tag{2.19}$$

Normalized Mutual Information (NMI): The NMI measures the mutual information between the clustering results and the true class labels, normalized by the entropy of the clustering outcomes and the entropy of the true class labels. NMI is bounded between 0 and 1, where higher values denote enhanced clustering performance and

a more robust alignment between the clustering and the actual labels [Strehl and Ghosh, 2003].

$$\text{NMI} = \frac{2 \cdot I(C, G)}{H(C) + H(G)} \tag{2.20}$$

Where, $H(C)$ stands for the entropy of the clustering outcome C. $H(G)$ represents the entropy of the true class labels G. $I(C, G)$ signifies the mutual information between the clustering outcome C and the true class labels G.

Entropy in the context of clustering evaluation, is used to measure the amount of disorder or impurity within a cluster.

The formula for entropy is given by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.21}$$

Where $H(X)$ is the entropy of the random variable X. $p(x_i)$ is the probability of observing the outcome x_i.

Rand Index (RI): The Rand Index (RI) quantifies the agreement between a clustering result and the true class labels by comparing pairs of data points. It evaluates the proportion of pairs of data points that share the same cluster assignment in both the clustering result and the true class labels (agreement) or are in different clusters in both (agreement) [Hubert and Arabie, 1985]. The RI ranges from 0 to 1, where a higher value indicates greater agreement between the clustering result and the true class labels. The formula for RI is:

$$\text{RI} = \frac{\text{Number of agreeing pairs}}{\text{Total number of pairs}} \tag{2.22}$$

Where the Number of agreeing pairs $(a+d)$ represents pairs of data points that are either in the same cluster (a) in both the clustering result and the true class labels (agreement) or in different clusters(d) in both (agreement). The Total number of pairs $(a + b + c + d)$ represents all possible pairs of data points.

Adjusted Rand Index (ARI): The Adjusted Rand Index (ARI) is a modification of the Rand Index that adjusts for chance greement. It accounts for the expected agreement that could occur by random chance [Hubert and Arabie, 1985]. The formula for ARI is :

$$\text{ARI} = \frac{RI - Expected\,RI}{max(RI_{Max} - Expected\,RI)} \tag{2.23}$$

Where, RI represents the Rand Index. $ExpectedRI$ is the expected Rand Index under random agreement. RI_{Max} is the maximum possible Rand Index value. The ARI ranges from -1 to 1, where:
- ARI = 1, indicates perfect agreement.
- ARI = 0, suggests clustering results no better than random.
- ARI $<$ 0, indicates clustering results worse than random.

The formula for RI and $ExpectedRI$ is given by:

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.24}$$

$$ExpectedRI = \frac{(TP + FP) \cdot (TP + FN)}{TP + TN + FP + FN} \tag{2.25}$$

Where TP = True Positives (number of pairs that are in the same cluster in both the clustering result and the true class labels). TN = True Negatives (number of pairs that are in different clusters in both the clustering result and the true class labels). FP = False Positives (number of pairs that are in the same cluster in the clustering result but in different clusters in the true class labels). FN = False Negatives (number of pairs that are in different clusters in the clustering result, but in the same cluster in the true class labels).

Homogeneity: Homogeneity assesses how well clusters align with individual true classes. It is calculated using conditional entropy:

$$H = 1 - \frac{H(C|K)}{H(C)} \tag{2.26}$$

where $H(C|K)$ = conditional entropy of true class labels given cluster assignments. $H(C)$ = entropy of true class labels.

The conditional entropy $H(C|K)$ measures the uncertainty of true class labels given cluster assignments. It quantifies how well the cluster assignments explain the true class labels. The formula for conditional entropy is:

$$H(C|K) = -\sum_i \sum_j p(c_i, k_j) \log \frac{p(c_i, k_j)}{p(k_j)} \tag{2.27}$$

Where $p(c_i, k_j)$ represents the joint probability of true class c_i and cluster assignment k_j. $p(k_j)$ is the probability of cluster assignment k_j. The sums run over all possible true class labels c_i and cluster assignments k_j.

Homogeneity ranges from 0 to 1, with higher values indicating better class consistency within clusters [Rosenberg and Hirschberg, 2007].

Completeness: Completeness measures if all data points of a true class are correctly grouped in a single cluster. It is calculated using conditional entropy:

$$C = 1 - \frac{H(K|C)}{H(K)} \tag{2.28}$$

Where $H(K|C)$ = conditional entropy of cluster assignments given true class labels. $H(K)$ = entropy of cluster assignments

Completeness ranges from 0 to 1, with higher values indicating better capture of true classes by clusters [Rosenberg and Hirschberg, 2007].

3. Related Work

Spectral clustering has gained significant attention in machine learning [Bishop and Nasrabadi, 2006] and data mining [Tan et al., 2016] research due to its versatility and theoretical foundation rooted in spectral analysis. In this chapter, we explore the related work on spectral clustering from three perspectives that coincide with the research questions and the scope of this thesis:

- In Section 3.1, we provide a comprehensive overview of recent findings and evaluations in the field of spectral clustering, focusing on three key aspects: the construction of similarity matrices, the formation of Laplacian matrices, and the selection of eigenvectors.
- In Section 3.2, we discuss the research conducted for comparing the performance of spectral clustering to k-means clustering.
- In Section 3.3, we discuss research conducted to evaluate spectral clustering performance.

3.1 Research in spectral clustering development

In the research direction of spectral clusering, various authors have proposed distinctive methodologies and conducted comprehensive assessments, unveiling valuable insights into their respective advantages and limitations [Jia et al., 2014]. This section delves into the research conducted on crucial aspects of spectral clustering, focusing on three key areas.

Constructing similarity matrix

Researchers propose novel methods for constructing similarity matrices to enhance spectral clustering performance:

[Zhang et al., 2011] introduced the Common-Near-Neighbor (CNN) method, a local density adaptive similarity measure. Utilizing local density to scale the Gaussian kernel function, CNN amplifies intra-cluster similarity, resulting in a clearer block

diagonal affinity matrix. In this matrix, the groups of datapoints are represented as distinct blocks along the diagonal, separated by lower similarity values, which are indicative of the boundary regions between groups.

[Wang et al., 2011] proposed Spectral Multi-Manifold Clustering (SMMC), effective when similarity values between points of different clusters are relatively low. In spectral clustering, a manifold denotes a curved or distorted space within higher dimensions, while "low-dimensional" implies efficient representation with fewer dimensions than the original space. SMMC assumes data lie on or near multiple smooth low-dimensional manifolds, some separated while others intersect. Local geometric information is leveraged to construct a suitable similarity matrix.

[Zhang and You, 2011] proposed a random walk-based approach to process the Gaussian kernel similarity matrix, incorporating neighbor relations to enhance the similarity matrix and better describe the data distribution. In the absence of neighbors, [Zhang and You, 2011]'s random walk-based approach could face challenges in enhancing the similarity matrix and accurately representing the data distribution.

[Li and Guo, 2012] introduced a new similarity matrix generation method based on the neighbor relation propagation principle, increasing the similarity of point pairs belonging to the same cluster and effectively detecting the underlying data structure.

Creating the Laplacian matrix

Once the similarity matrix is constructed, the next step involves creating the Laplacian matrix, which is crucial for various graph cut methods used in spectral clustering [Luo et al., 2010]. The Laplacian matrix plays a significant role in the performance of spectral clustering algorithms, and its selection depends on the clustering conditions (see section 2.3.3).

In recent research, the application of the p-Laplacian, a natural nonlinear extension of the graph Laplacian, to two-class cases has garnered attention ([Luo et al., 2010]). The authors undertake a comprehensive eigenvector analysis of the p-Laplacian, resulting in a naturally derived representation that captures the global arrangement of data points, rendering it well-suited for navigating the intricacies of complex data. In stark contrast to earlier methods reliant on greedy search strategies, their innovative approach employs efficient gradient descent optimization.

[Yang et al., 2010] propose the clustering algorithm LDMGI (Local Discriminant Models and Global Integration), which learns a new Laplacian matrix by combining manifold structure and local discriminant information. LDMGI constructs a local clique for each data point and evaluates the clustering performance within these cliques using local discriminant models. A unified objective function globally integrates the local models of all the cliques, making LDMGI more robust to algorithmic parameter selection and more suitable for real image clustering applications.

Selection of eigenvectors

Eigenvector selection is crucial in spectral clustering, as not all eigenvectors are informative for clustering. [Xiang and Gong, 2008] introduce "eigenvector relevance" to identify only relevant ones for clustering. Their method accurately estimates the cluster number and reveals natural grouping even with sparse and noisy data.

[Zhao et al., 2010] proposed an eigenvector selection method based on entropy ranking (ESBER). They rank eigenvectors by their importance on clustering and select the most relevant ones.

Rebagliati and Verri challenge the NJW algorithm's working hypothesis [Rebagliati and Verri, 2011]. They suggest a weaker version, using a k-dimensional subspace of the first m (m > k) eigenvectors for optimal partition.

3.2 Comparison of Spectral clustering and k-means clustering

Somashekara and Manjunatha [Somashekara and Manjunatha, 2014] conducted a comprehensive comparison based on eleven different clustering validity indices (both internal and external) for various clusters for the iris dataset. Their findings demonstrated the superior performance of spectral clustering over k-means clustering. They employed the Jordan approach (symmetric normalized Laplacian and Gaussian kernel) for spectral clustering. However, they did not employ different graph construction techniques nor different Laplacians for conduction spectral clustering. Unlike their approach, in our study, we also compare spectral clustering and k-means clustering using the iris and cORA dataset, but with six different cluster validity indices (three internal and three external) for three clusters.

3.3 Evaluation of spectral clustering performance

In order to reevaluate the performance of our spectral clustering framework, we draw upon the insights provided by Somashekara et al. ([Somashekara and Manjunatha, 2014]). In this instance, our comparative analysis focuses on the Iris dataset, assessing the performance of the two approaches across six distinct cluster evaluation measures. While Somashekara et al. ([Somashekara and Manjunatha, 2014]) utilized the Jordan approach for spectral clustering, we extend the evaluation by employing nine different combinations of similarity graphs (kNN, ϵ-neighborhood graph, and fully connected graph) and graph Laplacian (L, L_{rw}, andL_{sym}) methods within our framework. This allows us to conduct a comprehensive comparison between their approach and our framework, spanning a spectrum of clustering strategies and enhancing the depth of analysis.

4. Methodology

In this chapter, we present the methodological framework designed to address the research questions posed in this thesis. Another objective of our study is to comprehensively explore the flexibility and performance of our proposed spectral clustering framework. The following sections detail the methodologies devised to investigate each of the three research questions. We begin in section 4.1 where we describe our dataset, preprocess and analyse them. Section 4.2 describes the methods involved in creation and customization of our framework. In section 4.3 we discuss the methods for choosing various optimal hyperparameters. We then describe how we compare spectral clustering and k-Means clustering in section 4.4. Followed by comparing spectral comparing performance with other spectral clustering approach in section 4.5. In section 4.6 and section 4.7 we show the spectral clustering hyperparameters and environment of experiments respectively.

4.1 Datasets

This section of the thesis focuses on the two key datasets that form the foundation of the research. Each dataset is examined through two main phases, data preprocessing and data analysis. We begin by introducing the Iris dataset in subsection 4.1.1, and the CORA dataset is discussed in subsection 4.1.2.

4.1.1 Iris

In this section, we present the Iris dataset, a classic benchmark dataset in the realm of machine learning and pattern recognition [Bishop and Nasrabadi, 2006]. The Iris dataset comprises measurements of iris flowers' sepal length, sepal width, petal length, and petal width in centimeters. It consists of 150 samples, with each sample representing one iris flower. The (Figure 4.1) describes that there are three classes of Iris flower which have the same amount of samples in each class. Over time, Iris dataset has evolved into a foundational benchmark dataset for numerous classification and clustering algorithms due to its simplicity and well-balanced composition.

Data preprocessing

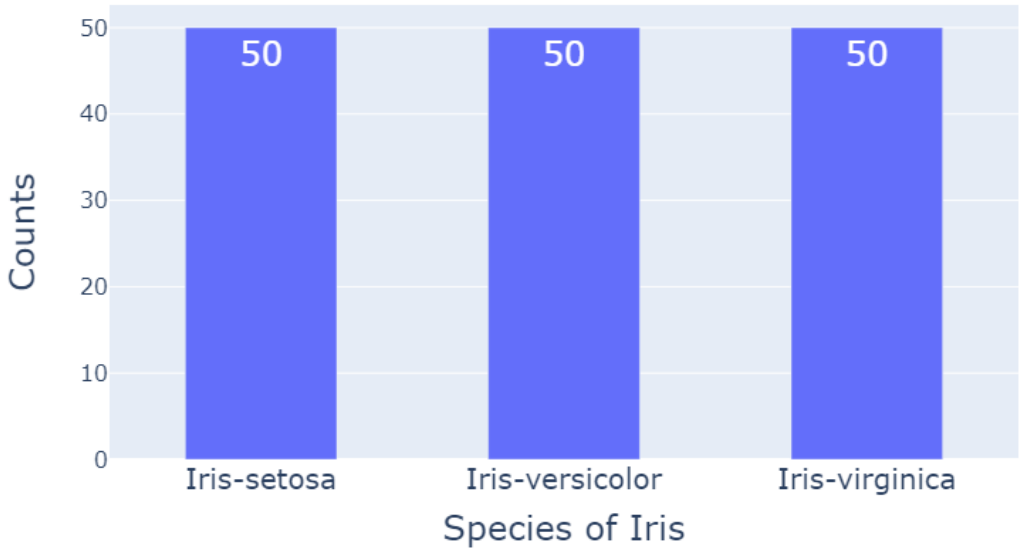


Figure 4.1: Distribution of Iris flower types, showing the count of each flower species in the dataset.

Initially, we delve into the various preprocessing techniques applied to refine the Iris dataset. The preprocessing phase encompasses the handling of missing values, encoding of categorical data, and normalization.

Removal of missing values: The Iris dataset, reveals no traces of missing values.

Encoding categorical data: Within the Iris dataset, categorical class labels corresponding to "species" are translated into numerical values via label encoding.

Normalization: Data were not normalized, as the Iris dataset exhibits a well-balanced distribution of attributes. This strategic choice ensures that the distinct attributes of the dataset are maintained in their original scale, thereby preserving the integrity of the datasets in the spectral clustering process.

Data analysis

We depict the Iris data using a scatter plot in two reduced components via Principal Component Analysis (PCA), as illustrated in Figure 4.2. We detail the descriptive statistical analysis of Iris data, including count, mean, standard deviation, and range calculations for features such as sepal length, sepal width, petal length, and petal width as shown in (Table 4.1). We also conduct exploratory data visualization (Figure 4.3), plotting Kernel Density Estimate (KDE)) and scatter plots to visualize feature distributions and potential relationships among species.

4.1.2 CORA

In this section, we introduce the CORA dataset, a widely used benchmark dataset in the field of academic literature-based research. The CORA dataset was compiled by

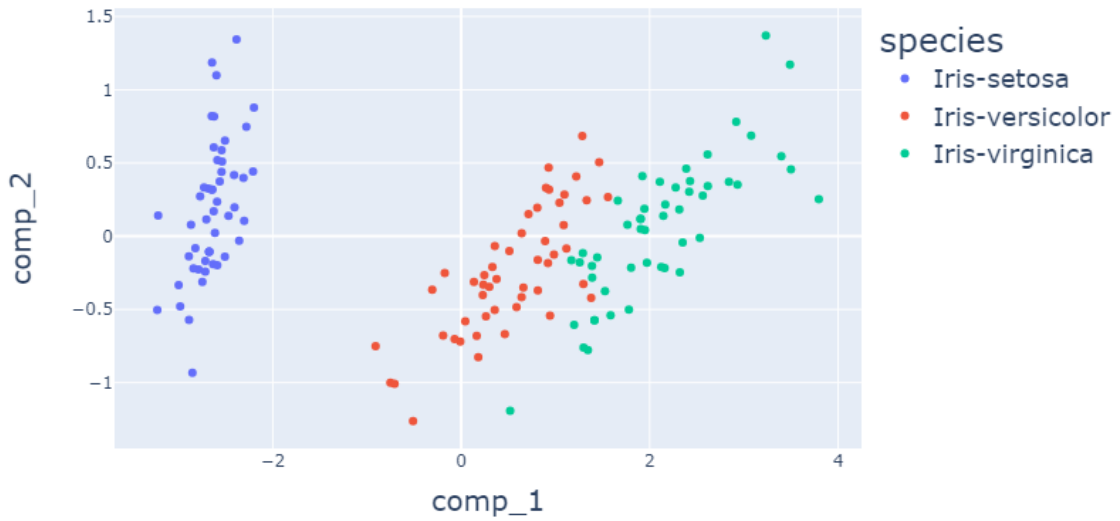


Figure 4.2: Visualization of the Iris dataset using PCA with a scatter plot showing two principal components

	sepal length	sepal width	petal length	petal width
count	**150**	**150**	**150**	**150**
mean	**5.84**	**3.05**	**3.76**	**1.2**
std	**0.83**	**0.43**	**1.76**	**0.76**
min	**4.3**	**2**	**1**	**0.1**
25%	**5.1**	**2.8**	**1.6**	**0.3**
50%	**5.8**	**3**	**4.35**	**1.3**
75%	**6.4**	**3.3**	**5.1**	**1.8**
max	**7.9**	**4.4**	**6.9**	**2.5**

Table 4.1: Statistical Analysis of Iris Dataset: Summary of descriptive statistics including count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum values for the four features of the Iris dataset

Andrew McCallum, and was first introduced in their paper [McCallum et al., 2000]. The dataset was created to facilitate research in the area of semi-supervised learning [Chapelle et al., 2009] and text classification [Sebastiani, 2002].

The version of the CORA dataset we use in our work, comprises 2708 scientific papers that have been categorized into seven distinct classes as shown in (Figure 4.4). Additionally, the dataset contains 5429 edges linking papers based on citations. Each individual publication in the dataset is represented by a binary word vector, where the presence or absence of each word from a dictionary of 1433 unique words is indicated by 1 or 0, respectively. Notably, the dataset portrays an inherent class imbalance, wherein certain classes may possess a considerably higher number of instances than others.

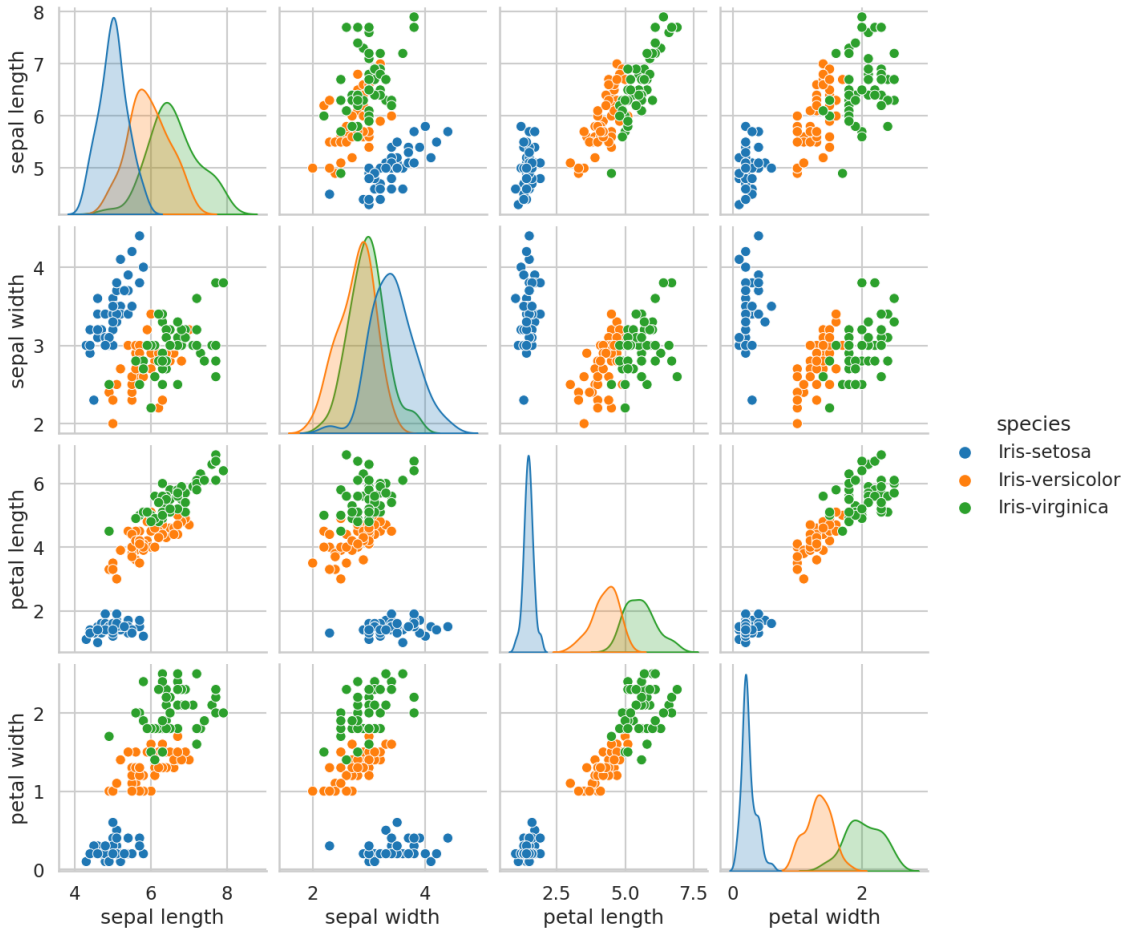


Figure 4.3: Visualization of the pairwise relationships of Iris flower features, categorized by their respective species. Along the diagonal, Kernel Density Estimation (KDE) plots provide a visual representation of the feature distribution for each species. Each point in the scatter plot corresponds to an individual Iris flower, with its position and color indicating its specific species. The visualization provides insights into feature correlations and variations across the different Iris species.

In the CORA dataset, the existence of edges introduces a directed graph structure, which is a common feature in citation networks. However, for the seamless operation of our spectral clustering framework, it is imperative that the graph representation remains undirected. Consequently, we deliberately disregard both the inherent directionality of the edges in the citation network and the edges themselves. Our primary objective centers around leveraging the node features within the CORA dataset for the purpose of spectral clustering, with the aim of effectively clustering similar publications.

Data preprocessing

Removal of missing values: The chosen version of the CORA dataset, is also devoid of any missing values.

Encoding categorical data: Within the CORA dataset, the categorical class labels ”subject” is encoded utilizing label encoding.

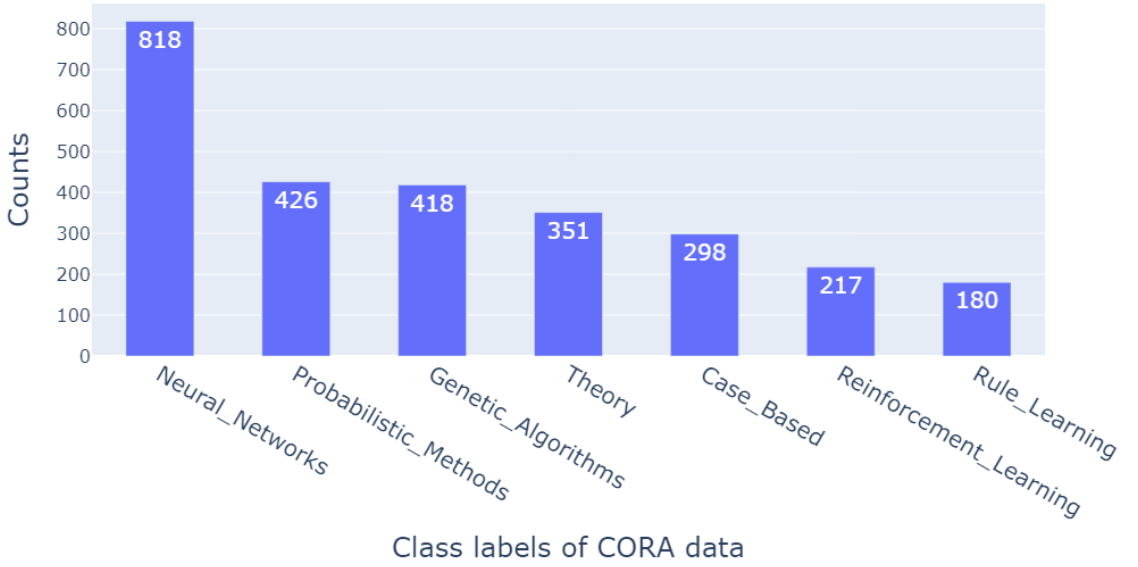


Figure 4.4: Distribution of CORA publications, showing the count of each publication across the 7 classes.

Normalization: Data were not normalized, as the CORA dataset features binary word vectors (represented as 1s and 0s).

Data analysis

For the CORA dataset, given the dataset's high dimensionality, we undertake an initial dimensionality reduction step for enhanced interpretability. We employ PCA to transform the data into a two-dimensional representation, as illustrated in (Figure 4.5). This visualization aids in intuitively grasping the inherent structure of the data.

While the full statistical analysis of all dimensions of the dataset would be cumbersome to depict comprehensively, we instead employ data visualization techniques. Distribution plots, box plots, and probability plots are generated using two principal components, providing insights into the feature distributions of the dataset, as shown in (Figure 4.6).

4.2 (RQ1): Creation and customizability of the spectral clustering framework

Our spectral clustering function takes input in the form of a numpy array X, where X is of datatype `numpy.ndarray` and has a shape of $[n\_samples, n\_samples]$ if the `adj` parameter is set to `True`. Alternatively, when `adj` is `False`, users can provide a `numpy.ndarray` with shape $[n\_samples\_a, n\_features]$, and the adjacency matrix is precomputed using the specified distance metric. The distance metric is governed by the `metric` parameter and utilizes the `scipy.spatial.distance.pdist()` function.

The type of similarity graph used is defined by the string parameter `sim_graph`, which has a datatype of `str`. The options for `sim_graph` include `'fully_connect'`,

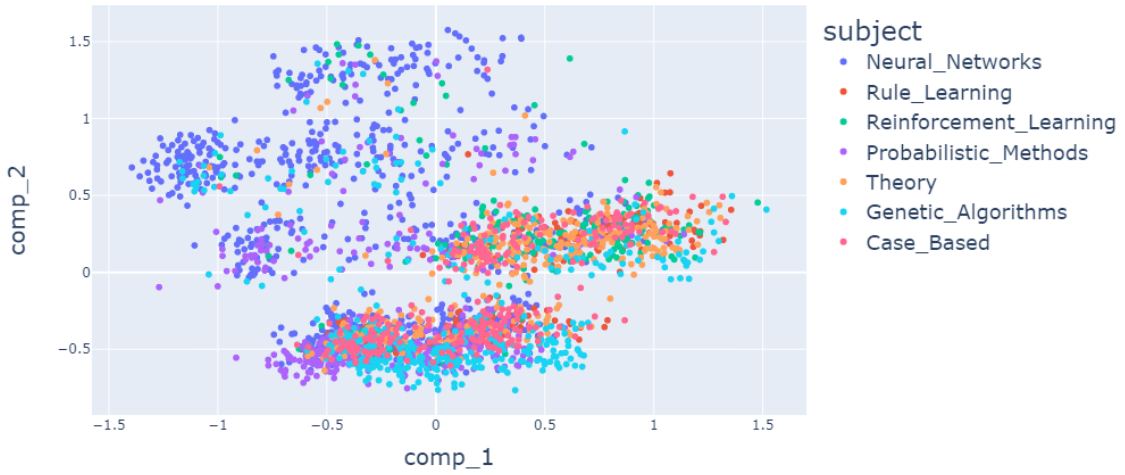


Figure 4.5: Scatter plot of PCA visualization for CORA dataset by subject: Each data point represents a scientific publication, projected onto two principal components through PCA. The scatter plot is color-coded with seven distinct colors, each representing a different subject of publication within the CORA dataset.

`'eps_neighbor'`, and `'knn'`. The parameter `sigma`, of datatype `float`, is relevant when constructing a fully connected graph using the Gaussian (RBF) kernel. For k-Nearest Neighbor graphs, the `knn` parameter, of datatype `int`, sets the number of neighbors. The `epsi` parameter, of datatype `float`, controls connections in an epsilon neighborhood graph.

Furthermore, users can customize the graph Laplacian normalization using the integer parameter `normalized`, which has a datatype of `int`. The choices include `1` for Random Walk normalized, `2` for Symmetric normalized, and other integers corresponding to Unnormalized versions.

The output of the function is an instance of the `sklearn.cluster` class, which includes the following attributes: `cluster_centers_`, an array with shape [$n\_clusters$, $n\_features$] containing the coordinates of cluster centers in K-Means; `labels_`, an array containing the labels of each data point; `inertia_`, a float representing the sum of squared distances of samples to their closest cluster center in K-Means; and `n_iter_`, an integer indicating the number of iterations run in K-means. Additionally, for reproducibility of results, the attribute `random_state=0` is available to set the random seed.

This spectral clustering framework is implemented using libraries like NumPy, SciPy, and scikit-learn, bettering user control and enabling tailored analysis of different similarity graphs and graph Laplacians for improved clustering performance.

4.3 Choosing hyperparameters

In our comprehensive hyperparameter analysis, we systematically explore the sensitivity of Spectral Clustering on both the Iris dataset and the Cora dataset, utilizing ground truth information.

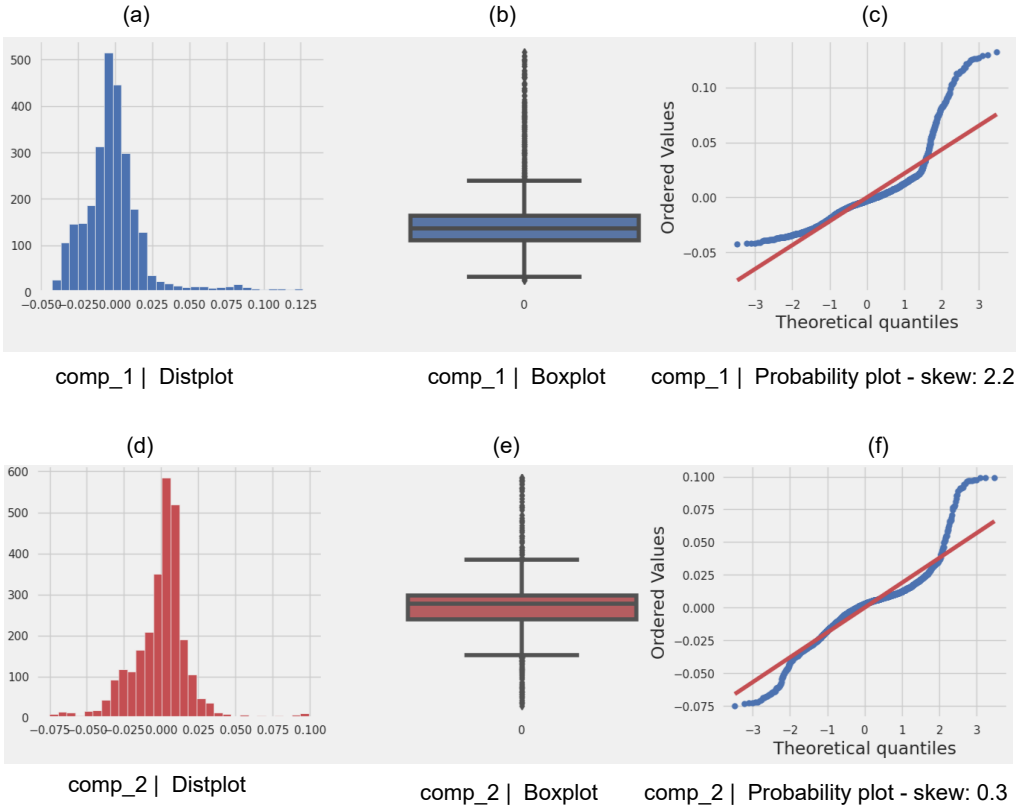


Figure 4.6: Visualization of Two Principal Components of CORA Data: Panels (a), (b), and (c) depict the distribution plot, box plot, and probability plot for Component 1, which exhibits a skew of 2.2 in the probability plot—indicating a moderate departure from a perfectly symmetric distribution. Panels (d), (e), and (f) showcase the distribution plot, box plot, and probability plot for Component 2, which displays a minor skew of 0.3 in the probability plot, implying a slight deviation from symmetry.

4.3.1 Choosing a distance function

In this section, we perform sensitivity analysis aimed at selecting an optimal distance function, a critical aspect of our spectral clustering framework. To this end, we explore the impact of different distance functions on the quality of clustering results for both the Iris and CORA datasets. The goal is to identify a distance metric that maximizes the effectiveness of our framework, keeping in mind the use case and datasets. We begin by considering seven distinct distance functions as discussed in section 2.6.1 for calculating pairwise distances between data points to construct the adjacency matrix. This comprehensive evaluation is performed manually, leveraging the ground truth information available for both the Iris and CORA datasets.

In our spectral clustering approach applied to both the Iris and CORA datasets, we had the freedom to choose from nine different combinations of similarity graph types and Laplacian matrices. However, we opted for specific configurations, not with the intention of identifying the best clustering performance, but rather to observe consistent outcomes across various distance metrics and configurations.

For the Iris dataset, we selected a k-Nearest Neighbors (kNN) graph with k=6 and the random walk normalized Laplacian (L_{rw}). This choice of kNN graph aims to capture local neighborhood relationships within the Iris dataset, emphasizing the proximity of nearby data points while mitigating noise influence. The choice of L_{rw} adheres to the recommendation of [Von Luxburg, 2007] for L_{rw} usage due to its aptness in reflecting balanced data connectivity. The decision to form m=3 clusters was guided by the known ground truth of the Iris dataset. We assess the performance of each distance metric through spectral clustering as shown in (Figure 4.7) and (Figure 4.8) using three internal cluster validity indices (discussed in Section 2.6.2) and three external cluster validity indices (discussed in Section 2.6.3) respectively.

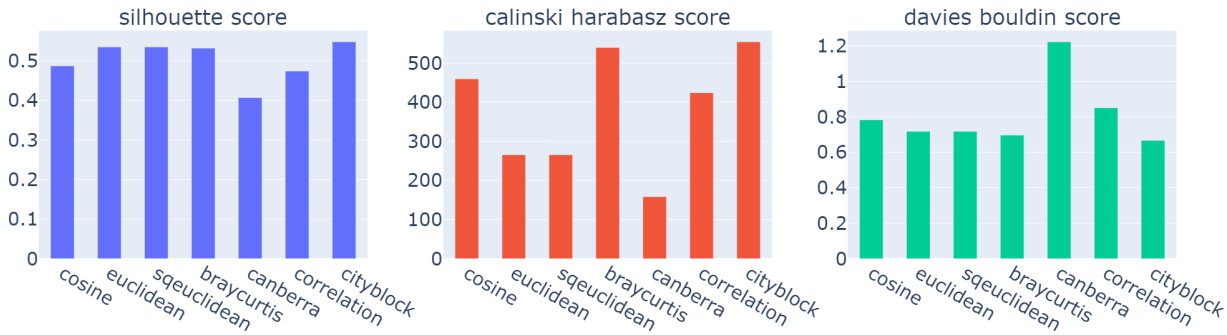


Figure 4.7: Comparison of different distance functions with internal cluster measures of Iris data. (Spectral Clustering: kNN=6, L_{rw}, and no. of clusters $k = 3$)

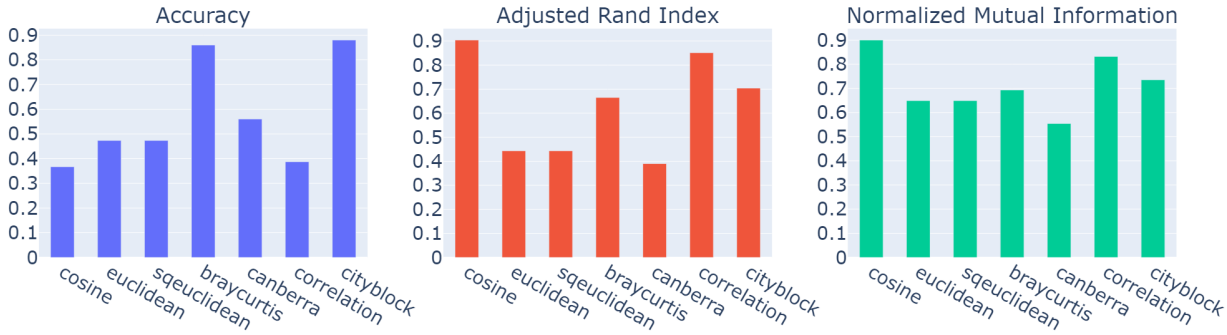


Figure 4.8: Comparison of different distance functions with external cluster measures of Iris data. (Spectral Clustering: kNN=6, L_{rw}, and no. of clusters $k = 3$)

For the CORA dataset, in a similar vein, a kNN graph with k=23 in tandem with L_{rw} was applied. This configuration acknowledges the intricate relationships in high-dimensional data, particularly prominent in academic paper citation networks. The choice of m=7 clusters aimed to effectively encapsulate the diverse academic paper topics (labels of the dataset). This enables us to assess the performance of each distance metric through spectral clustering, gauging their effectiveness via three internal cluster validity indices (Figure 4.9) and three external cluster validity indices (Figure 4.10).

Since the cosine distance shows good consistency for both Iris and CORA datasets and achieved the best performance for some evaluation metrics, we chose the cosine distance as distance metric for all following experiments.

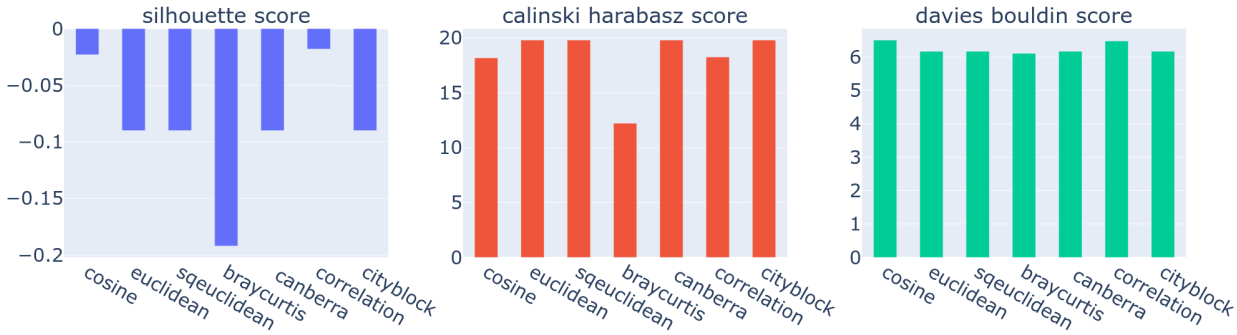


Figure 4.9: Comparison of different distance functions with internal cluster measures of CORA data. (Spectral Clustering: kNN=23, L_{rw}, and no. of clusters $k = 7$)

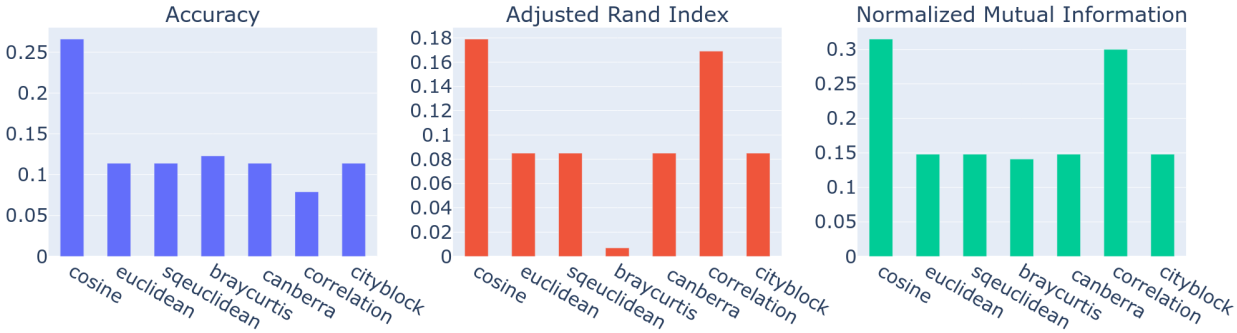


Figure 4.10: Comparison of different distance functions with external cluster measures of CORA data. (Spectral Clustering: kNN=23, L_{rw}, and no. of clusters $k = 7$)

4.3.2 Choosing hyperparameters for similarity graphs

To comprehensively gauge the performance of different Laplacian types, namely the unnormalized (L), random walk normalized (L_{rw}), and symmetric normalized (L_{sym}) Laplacians, we pair each of them with three diverse types of similarity graphs. These similarity graph variants include the k-Nearest Neighbors (kNN) graph, epsilon neighborhood graph, and fully connected graph. Each combination of Laplacian type and similarity graph forms a unique configuration, resulting in a total of nine configurations.

Our methodology involves systematically tuning the hyperparameters associated with each configuration. We consider factors such as the number of nearest neighbors 'k' for kNN graphs, the threshold 'ϵ' for epsilon neighborhood graphs, and the standard deviation 'σ' for Gaussian kernels in fully connected graphs. By carefully varying these hyperparameters, we evaluate the performance of Spectral Clustering for each configuration and Laplacian type across both the Iris and CORA datasets.

To comprehensively assess the performance of these configurations, we use six external cluster evaluation metrics. Rand Index, Adjusted Rand Index, Homogeneity Score, Completeness Score, V-Measure Score (or Normalized Mutual Information), and Accuracy. Notably, these metrics capitalize on our prior knowledge of the datasets' ground truth, ensuring a meaningful evaluation process. This cohesive approach provides an intuitive and holistic view of performance, enabling effective comparison and selection of optimal hyperparameters. Table 4.2 shows the selection of the best hyperparameters of nine different combinations of Laplacian types and similarity graphs of both the Iris and the CORA dataset. The exhaustive evaluation of all hyperparameters is plotted in Appendix A.4. These optimal settings lay the foundation for our subsequent evaluation of spectral clustering performance on both the Iris and Cora dataset. This data-driven approach ensures that the chosen hyperparameters align with the true nature of the datasets, enhancing the reliability of our subsequent analyses.

Combination	Laplacian Type	Similarity Graph	Best Hyperparameters	
			Iris	CORA
1	Unnormalized (L)	k-NN	k = 6	k = 23
2	Unnormalized (L)	Epsilon Neighbor	$\epsilon = 0.01$	$\epsilon = 1$
3	Unnormalized (L)	Fully Connected	$\sigma = 0.01$	$\sigma = 1$
4	Normalized (L_{rw})	k-NN	k = 6	k = 23
5	Normalized (L_{rw})	Epsilon Neighbor	$\epsilon = 0.01$	$\epsilon = 0.8$
6	Normalized (L_{rw})	Fully Connected	$\sigma = 0.01$	$\sigma = 0.4$
7	Normalized (L_{sym})	k-NN	k = 6	k = 22
8	Normalized (L_{sym})	Epsilon Neighbor	$\epsilon = 0.01$	$\epsilon = 0.8$
9	Normalized (L_{sym})	Fully Connected	$\sigma = 0.01$	$\sigma = 0.1$

Table 4.2: List of the best hyperparameters for similarity graphs used in all experiments: This table presents the best hyperparameters identified through a comprehensive sensitivity analysis for Spectral Clustering on two distinct datasets, Iris and CORA. The table showcases nine different combinations of Laplacian types and similarity graphs, along with their corresponding optimal hyperparameters. The optimal values for each combination are listed for both the Iris and CORA datasets, providing insights into the ideal settings for successful Spectral Clustering. We refer to Appendix A.4 for a comprehensive evaluation of all values of evaluated hyperparameters.

4.3.3 Choosing the optimal number of clusters

Our methodology for finding optimal k clusters involves exploring three distinct techniques, the elbow method, internal measures, and the eigengap heuristics. Irrespective of the availability of the ground truth of the dataset in use, one can employ these techniques for evaluating the best k. For demonstration, we use the Iris dataset to illustrate the application of each technique.

Elbow method: The elbow method as shown in (Figure 4.11) is employed to determine the optimal number of clusters for k-means clustering. The within-cluster sum of squares (WCSS) also called inertia is calculated across a range of potential cluster

counts (`no_of_clusters`). The k-means clustering is executed using the `KMeans` class from the scikit-learn library with 'k-means++' initialization and a maximum of 500 iterations (`max_iter = 500`) for stability. The resulting WCSS values are stored in the list `wcss`. The inertia values are visualized using a Plotly Express line plot.

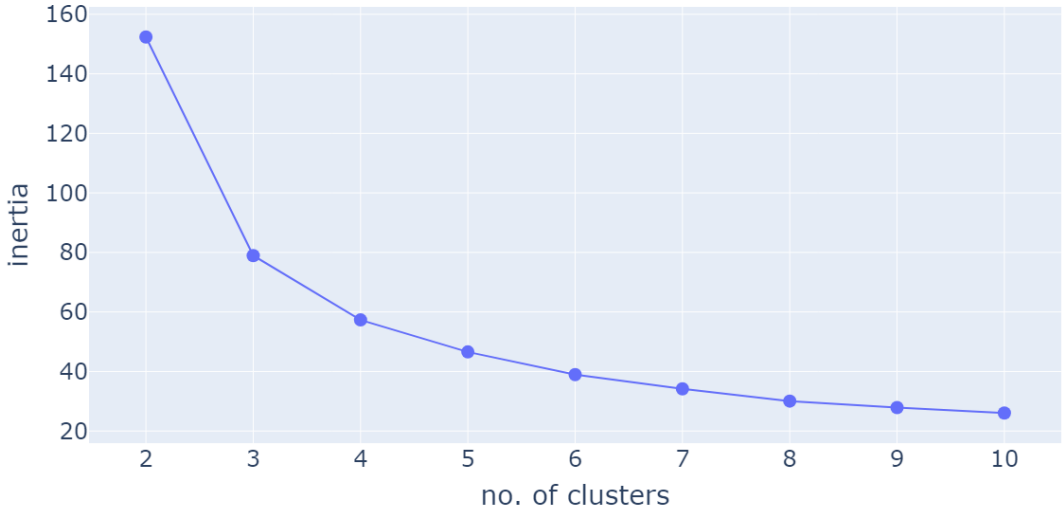


Figure 4.11: Elbow method: the line plot illustrates the Elbow Method applied to the Iris dataset. The x-axis represents the number of clusters, while the y-axis depicts the inertia. A distinct "elbow" point is evident at cluster three, indicating the optimal number of clusters for the Iris data

The plot assists in identifying the "elbow point," indicating an appropriate balance between minimizing intra-cluster variance and preventing overfitting.

Internal measures: To assess the quality of the clusters produced by the spectral clustering algorithm across varying cluster counts (`no_of_clusters`), three internal validity indices are employed as shown in (Figure 4.12) (silhouette score, Calinski-Harabasz score, and Davies-Bouldin score). These indices provide insights into the compactness and separation of clusters, the overall dispersion, and the inter-cluster similarity, respectively. The analysis is visualized using the Plotly library.

For each internal validity index, scatter plots are generated, where the x-axis corresponds to the number of clusters and the y-axis represents the calculated index score. All the indices are calculated using the scikit-learn library. The silhouette score quantifies how similar an object is to its own cluster compared to other clusters. The Calinski-Harabasz score measures the ratio of between-cluster variance to within-cluster variance. The Davies-Bouldin score evaluates the average similarity between each cluster and its most similar cluster. By observing the behavior of these indices across various cluster counts, we can determine the optimal number of clusters for spectral clustering, ensuring the most meaningful and accurate partitioning of the data.

Eigengap heuristics: To illustrate the eigengap heuristics for finding optimal k suggested by [Von Luxburg, 2007], a combination of the NumPy and Plotly Express

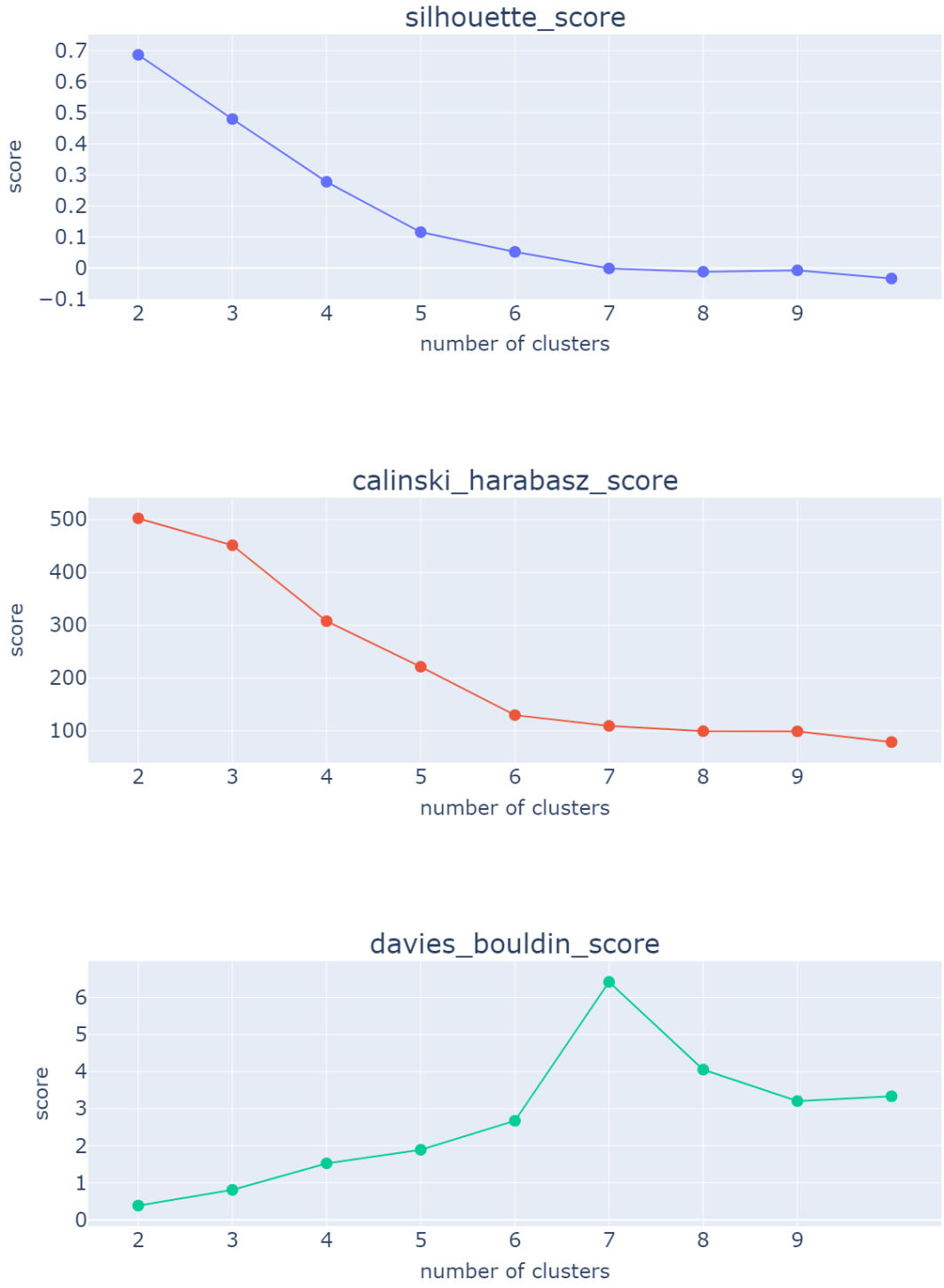


Figure 4.12: Internal measures to identify optimal number of clusters : the three-subplot image shows internal cluster scores (Silhouette Score, Calinski-Harabasz Score, and Davies-Bouldin Score) across a range of cluster numbers for the Iris dataset. The analysis aids in determining the optimal number of clusters for the data.

libraries are utilized. The process begins by generating an array of integers from 1 to 15, representing the indices of the eigenvalues to be analyzed. These eigenvalues are obtained from the eigen decomposition of the Laplacian matrix (see 2.3.4) and are subsequently sorted and made real for consistency.

The scatter plot as shown in (Figure 4.13) is constructed using Plotly Express, where the x-axis represents the indices of the eigenvalues, and the y-axis displays the corresponding eigenvalues themselves. The color-coding of the points is aligned with the magnitude of the eigenvalues, utilizing a color scale that ranges from low to high values for better visualization.

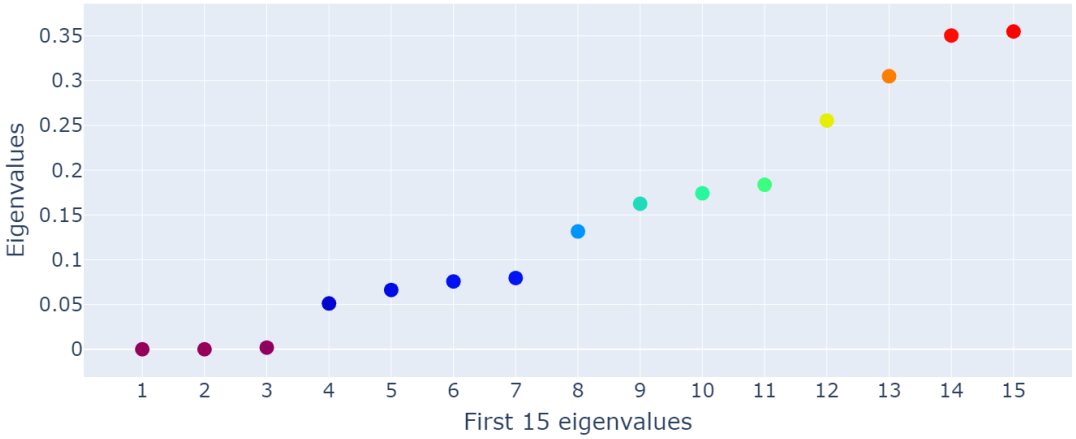


Figure 4.13: Eigengap heuristics for the Iris Dataset. The plot displays the eigenvalues of the graph Laplacian matrix computed from the Iris dataset. The gap, observed between the third and fourth eigenvalues, indicates the presence of three distinct clusters in the dataset. The first three eigenvalues are very close to zero, this further suggests the presence of three connected components within the Iris dataset which are clusters of Iris.

The significant gap between the eigenvalues 3 and 4 indicates that the Iris dataset ideally comprises three clusters. Notably, the ideal case of k completely disconnected clusters is characterized by an eigenvalue of 0 having a multiplicity of k. In the Iris dataset, the near-zero values of the first three eigenvalues signify that the data exhibits characteristics close to this ideal scenario, suggesting the presence of three clusters.

In our case, we leverage the availability of the ground truth of the Iris dataset. However, it is crucial to note that the efficacy of these methodologies can vary across different datasets, yielding different optimal solutions. Combining multiple techniques is essential to ensure consensus among methods to identifying the optimal number of clusters.

The resulting plot provides insights into the appropriate number of clusters for the dataset, aiding in informed decision-making during clustering analysis.

4.4 (RQ2): Comparative analysis of spectral clustering with k-Means clustering

The Research Question (RQ2) centers around a comparative analysis between our spectral clustering framework and k-Means clustering. Our methodology for addressing this question is characterized by a systematic sequence of steps. We finetuned hyperparameters within our spectral clustering framework, aligning with the procedure previously outlined in section 4.2. Subsequently, leveraging these optimized configurations, we proceed to perform spectral clustering on our datasets. For the k-Means execution, the scikit-learn library's `KMeans` class is used to execute k-Means clustering. The parameter `n_clusters` denotes the number of clusters to form. Initialization is performed using the 'k-means++' method to enhance convergence. The algorithm is run with a maximum of 500 iterations (`max_iter = 500`), over 10 different centroid seeds (`n_init = 10`) for result stability. The `random_state` parameter is set to 0 for reproducibility. The `fit` function is then applied to the dataframe `df`, resulting in cluster assignments and centroids refined iteratively for accurate clustering outcomes.

To evaluate the results, we employ six distinct cluster evaluation measures—three internal measures (Silhouette score, Calinski-Harabasz index, and Davies-Bouldin index) and three external measures (Accuracy, Adjusted Rand Index, and Normalized Mutual Information). This comprehensive analysis enables us to objectively compare the clustering outcomes of our spectral clustering framework against those of the k-Means approach.

4.5 (RQ3): Comparative analysis of spectral clustering approaches

In order to address Research Question (RQ3) regarding the comparative analysis of our framework's spectral clustering approach with another spectral clustering method executed by [Somashekara and Manjunatha, 2014], we adopt a systematic methodology guided by the results obtained in response to Research Question (RQ2).

As established in (RQ2), we conducted an in-depth comparison of our spectral clustering framework against k-Means clustering on the Iris and CORA datasets. To extend this analysis, we leverage a similar methodology to execute our spectral clustering framework, but for the comparative analysis, we substitute the k-Means clustering outcomes with those obtained using the approach presented by [Somashekara and Manjunatha, 2014] for the Iris dataset. This substitution allows us to assess the performance of our framework against an alternative spectral clustering technique.

To perform spectral clustering using our framework, we follow the same procedure outlined in Section 4.2, carefully fine-tuning hyperparameters for optimized results. Additionally, we execute the spectral clustering approach by [Somashekara and Manjunatha, 2014] with their reported performance values to maintain consistency.

For evaluating and comparing the outcomes of these clustering approaches, we employ six distinct cluster evaluation measures. These measures include three

internal measures (Silhouette score, Calinski-Harabasz index, and Davies-Bouldin index) and three external measures (Accuracy, Adjusted Rand Index, and Normalized Mutual Information). By quantitatively assessing these metrics for both approaches, we can objectively compare the effectiveness of the two spectral clustering techniques.

In visualizing the comparative analysis results, we utilize the Plotly library to create bar plots for each of the evaluation measures. The plots are organized into three rows, with the measures for Accuracy, Adjusted Rand Index, and Normalized Mutual Information represented in each respective row. Each bar plot presents the performance of our spectral clustering framework and the [Somashekara and Manjunatha, 2014]'s approach side by side, facilitating a direct visual comparison.

4.6 Hyperparameters

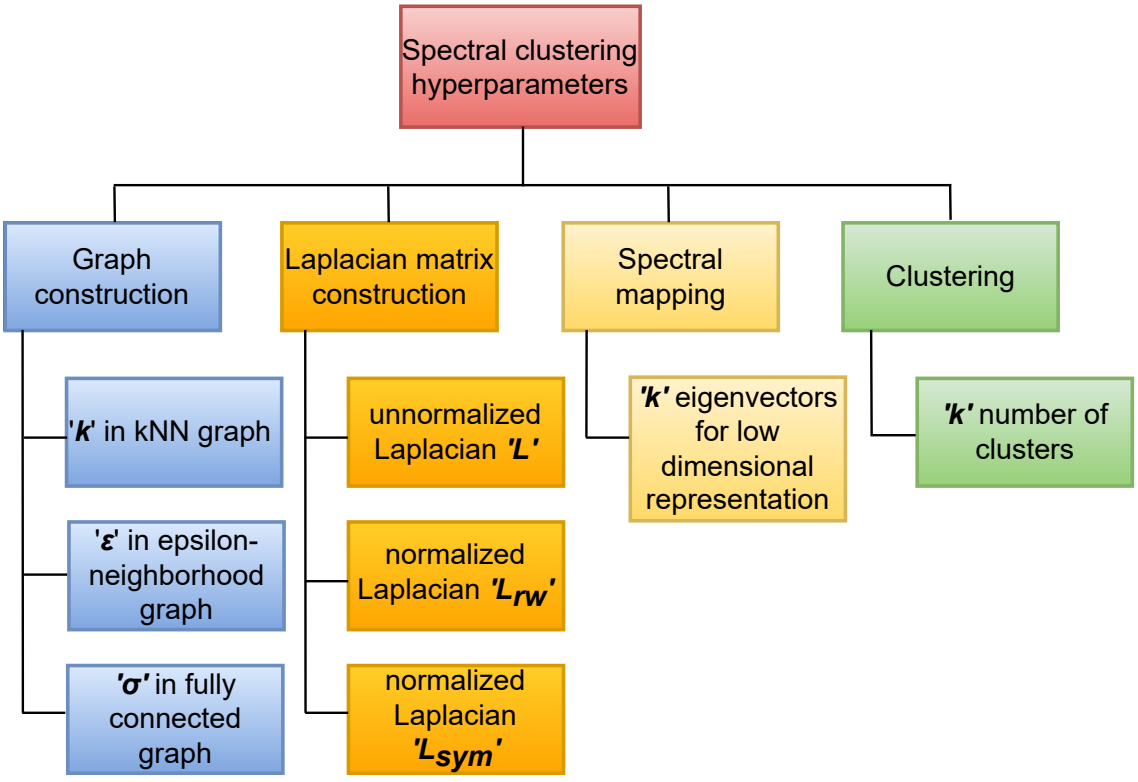


Figure 4.14: Spectral clustering hyperparameters

4.7 Environment for experiments

In this section, we provide an overview of the experimental setup under which we deployed our experiments. It is imperative to highlight that the hardware configurations outlined below are distinct for two key platforms, the university's Linux server and the Google Linux server designed to support Google Colaboratory executions.

- **Operating System** Ubuntu 22.04.2 LTS
- **Processor** Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz, 2vCPU Intel(R) Xeon(R) CPU @ 2.20GHz
- **Memory** 251 GB RAM, 13GB RAM
- **Programming Languages** Python (Version 3.10.12)
- **Programming Tools** Visual studio code (Version 1.81.0), Google Colaboratory
- **Libraries** Refer Table 4.3

Library	***Version***	***Library***	***Version***
matplotlib	3.7.1	scipy	1.10.1
numpy	1.23.5	seaborn	0.12.2
pandas	1.5.3	sklearn	1.2.2
plotly	5.15.0	-	-

Table 4.3: Libraries used for experiments.

5. Framework

This chapter is dedicated to introducing our spectral clustering framework, which is critical for answering research question 1 (RQ1). This chapter aims to offer a comprehensive overview of the distinct stages and steps constituting the spectral clustering process at a higher level.

The spectral clustering framework, as illustrated in (Figure 5.1), encompasses four pivotal stages denoted as (a), (b), (c), and (d), revealing a systematic approach to extract meaningful clusters from input data.

At the outset, the framework's initiation stage (a) involves introducing a *.csv* file as the primary data source.

Central to our framework is the preprocessing stage (b). In this stage, a series of vital steps are performed. The first step, (b1), calculates pairwise distances between data points using a designated distance metric as discussed in section 2.6.1, contributing to the construction of the pairwise distance matrix. In section 4.3.1, we provide a thorough explanation of choosing the best distance function. The subsequent step, (b2), results in the formation of the weighted adjacency matrix (or similarity matrix) through three distinct similarity graph construction techniques (kNN graph, ϵ-neighborhood graph, and fully connected graph) as discussed in section 2.3.2. Accompanying this, (b3) constructs the degree matrix as discussed in section 2.2. Finally, (b4) constructs three variants of the Laplacian matrix (L, L_{rw} and L_{sym}) as discussed in section 2.3.3, pivotal for subsequent spectral analysis.

Afterwards, the created Laplacian matrices are eigen decomposed in the next stage (c). Step (c1) employs an eigen solver to acquire the eigenvectors and eigenvalues of each Laplacian matrix. Subsequently, step (c2) involves the selection of k eigenvectors, leading to the creation of a lower-dimensional representation of the original data via spectral mapping (see section 2.3.4), providing insight into intricate clusters and communities.

Finally, in the clustering stage (d), the first step, (d1), leverages the widely-adopted k-means clustering algorithm (see section 2.3.5) to partition the reduced data representation into distinct clusters based on proximity within the lower-dimensional

space. The last step, (d2), assigns data points to discrete clusters based on the outcomes of the clustering process (see section 2.3.5), leading to the formation of groups.

Notably, the framework offers nine unique outputs, each resulting from distinct combinations of similarity graphs and graph Laplacians, which are compared to identify the most appropriate cluster results as detailed in section 4.3.2.

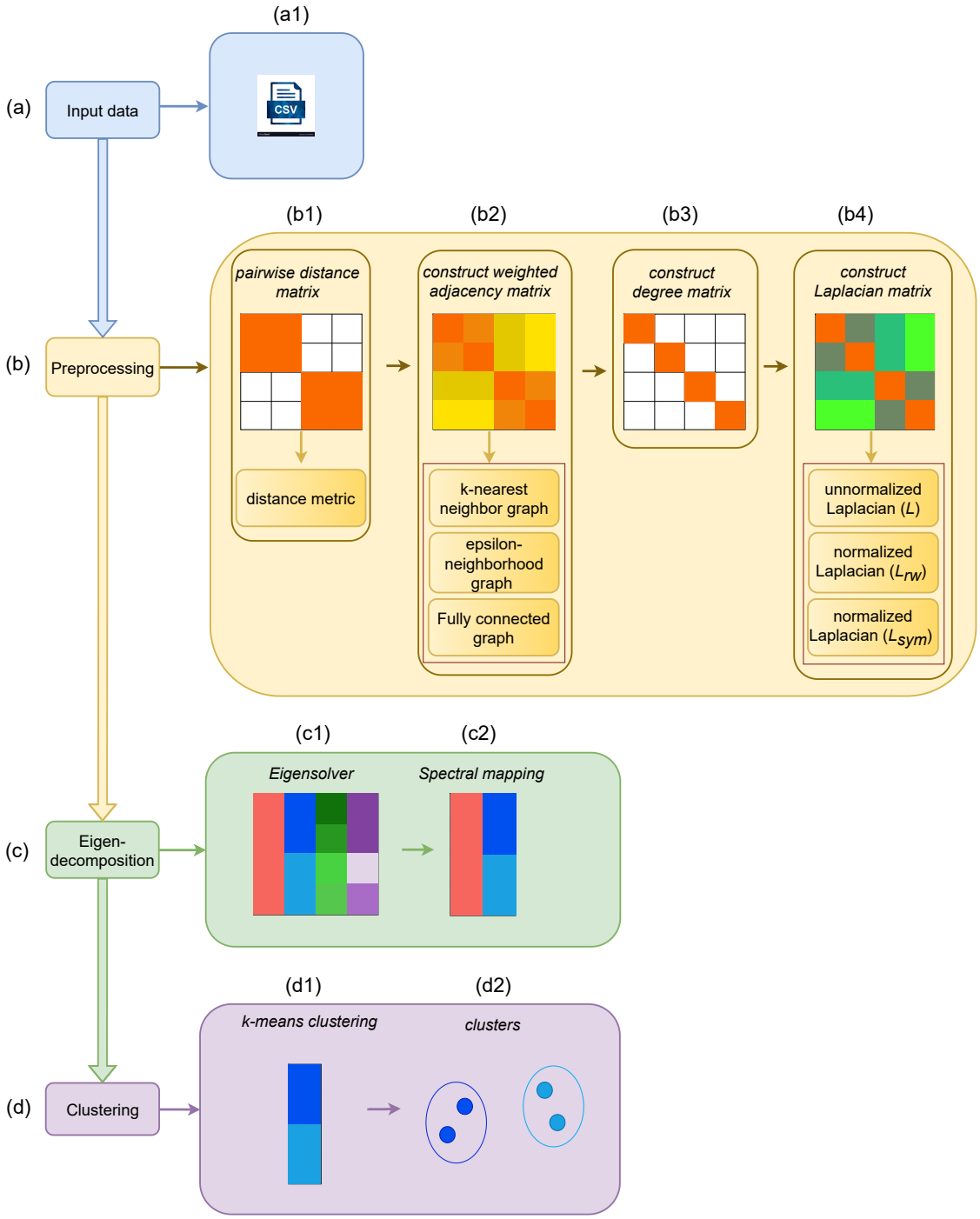


Figure 5.1: A high-level view of spectral clustering framework: The framework is depicted in four stages: (a) introduces the initial .csv input file. Preprocessing (b) involves (b1) pairwise distance calculation for the adjacency matrix, (b2) weighted adjacency matrix (or similarity matrix) construction using three similarity graph techniques, (b3) degree matrix creation, and (b4) Laplacian matrix construction with three Laplacian variants. Eigendecomposition (c) includes (c1) eigen solver for eigenvectors and eigenvalues and (c2) selection of k eigenvectors for lower-dimensional representation. Clustering (d) involves (d1) k-Means clustering in reduced space and (d2) assignment of data points to clusters. The framework generates nine diverse outputs, each resulting from varying selections (kNN graph, ϵ-neighborhood graph, and fully connected graph) of similarity graphs and graph Laplacians (L, L_{rw} and L_{sym}), and subsequently, these nine outputs are systematically compared to determine the optimal spectral clustering result.

6. Results and discussion

In this chapter, we are presenting and discussing the outcomes derived from addressing the three research questions formulated in Section 1. This chapter is divided into three sections, each section corresponds to one research question and focuses on its results and discussion.

6.1 (RQ1) Creation and customizability of our spectral clustering framework

To address the first research question (RQ1), we engage in a comprehensive comparison between our spectral clustering framework and the spectral clustering implementation offered by scikit-learn [Pedregosa et al., 2011]. Our investigation centers on evaluating the flexibility of our framework in enabling users to customize spectral clustering across diverse similarity graphs and graph Laplacians.

I successfully implemented a new framework for user controlled spectral clustering to address the limitations of the spectral clustering choices provided by scikit-learn. The core of our framework's adaptability lies in its provision of multiple choices throughout the spectral clustering process. To initiate spectral clustering, our framework allows for the construction of a pairwise distance matrix by calculating pairwise distances between data points, employing seven distinct distance metrics (cosine, Euclidean, sqeuclidean, Braycurtis, Canberra, correlation, and cityblock). This fulfills the Criteria (i) stated in Chapter 1. Following the construction of the pairwise distance matrix, our framework introduces three options (kNN graph, ϵneighborhood graph, and fully connected graph) for creating the weighted adjacency matrix or similarity matrix. The diversity in our framework arises from the utilization of three different similarity graphs. A greater variety in the choice of similarity graph fulfills Criteria (ii) stated in Chapter 1. Another pivotal component is the generation of the Laplacian matrix. Our framework allows users to choose from three variants of the Laplacian matrix, the unnormalized Laplacian (L), random walk normalized Laplacian (L_{rw}), and symmetric normalized Laplacian ((L_{sym}). The different choices of graph Laplacian fulfills criteria (iii) stated in Chapter 1. As a result of these

combined choices, our framework yields nine distinctive outputs for spectral clustering, i.e., all combinations of three different similarity graphs and three different Laplacian variants.

6.2 (RQ2) Performance comparison of spectral clustering and k-Means clustering

In this section, we conduct a comprehensive comparison between spectral clustering and k-Means clustering for both the Iris (section 6.2.1) and CORA (section 6.2.2) dataset. We evaluate their performances using a range of external and internal cluster measures to address (RQ2).

6.2.1 Comparison of Spectral clustering and k-Means clustering performance using Iris dataset

In this section, we present the outcomes of our comparison between spectral clustering and k-Means clustering on the Iris dataset to address (RQ2). We utilized our optimized hyperparameters (section 4.3) to conduct spectral clustering using our framework. The obtained results were then compared with those of k-Means clustering using six different cluster measures, encompassing both external and internal evaluations. Table 6.1 shows the evaluation of these cluster measures for both k-Means and spectral clustering.

Method	External measures			Internal measures		
	Accuracy	ARI	NMI	Sil. score	CH score	DB score
k-Means	0.893	0.730	0.758	**0.553**	**560.400**	0.662
$SC\_L_{rw}\_kNN$	0.366	**0.903**	**0.899**	0.486	459.641	0.782
$SC\_L_{rw}\_EN$	0.906	0.759	0.695	0.430	387.551	0.900
$SC\_L_{rw}\_FC$	0.366	**0.903**	**0.899**	0.486	459.641	0.782
$SC\_L_{sym}\_kNN$	0.366	**0.903**	**0.899**	0.486	459.641	0.782
$SC\_L_{sym}\_EN$	**0.946**	0.850	0.830	0.479	451.074	0.808
$SC\_L_{sym}\_FC$	0.366	**0.903**	**0.899**	0.486	459.641	0.782
SC_L_kNN	0.366	**0.903**	**0.899**	0.486	459.641	0.782
SC_L_EN	0.0.660	0.558	0.720	0.552	252.563	**0.377**
SC_L_FC	0.0.660	0.558	0.720	0.552	252.563	**0.377**

Table 6.1: Comparison of k-Means and Spectral Clustering Performance: the table compares our framework with k-Means clustering for Iris dataset. The best values across all the cluster validity measures and the various methods are highlighted in bold. We use the naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph for our framework.

External cluster measures for Iris dataset: Our spectral clustering framework consistently surpasses k-Means clustering in terms of external cluster measures (Accuracy, ARI, NMI) when applied to the Iris dataset (see Figure 6.1). Notably, the

majority of our ARI and NMI results outperform those of k-means. Additionally, in a specific case involving the symmetric normalized Laplacian (L_{sym}) in conjunction with the epsilon neighborhood graph, our approach achieves a remarkable accuracy of 94.7%, representing a significant 5.4% enhancement over k-Means. Moreover, our framework demonstrates a noteworthy 12.1% increase in the ARI value, resulting in a value of 0.851, and a substantial 9.4% improvement in the NMI value, which reaches 0.831.

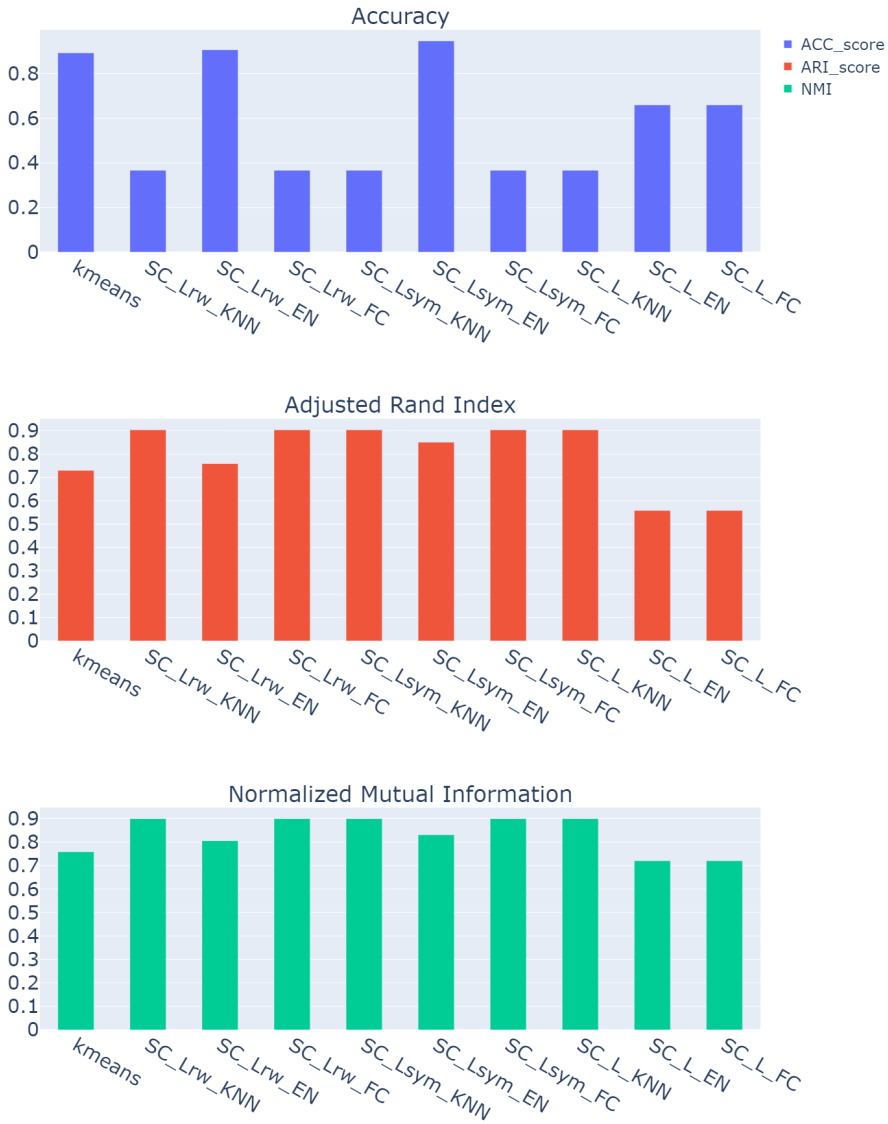


Figure 6.1: k-Means clustering and Spectral clustering performance comparison: the figure compares external cluster validity measures (Accuracy, Adjusted Rand Index, and Normalized Mutual Information) for the Iris dataset. Subplots contrast the k-Means approach with nine spectral clustering configurations. We use the naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph for our framework.

These differences in performance can be attributed to the fact that spectral clustering leverages graph spectral properties to uncover complex data structures, allowing it to capture inherent relationships and non-linearities that k-Means might overlook. Spectral clustering's ability to consider the underlying geometry of the data aids in forming more accurate clusters that align with the true class labels. The symmetric normalized Laplacian and epsilon neighborhood graph combination further enhances this capability by enabling the framework to detect subtle variations and connections within the data distribution, resulting in improved cluster assignment accuracy.

In contrast, k-Means relies on simple geometric centroids and may struggle with non-linear or irregularly shaped clusters. It doesn't take into account the intricate relationships between data points that spectral clustering captures through the graph representation. Consequently, k-Means might produce less accurate clusters, leading to lower values in external cluster measures. The observed improvements in Accuracy, ARI, and NMI metrics with our spectral clustering framework underline its effectiveness in identifying meaningful clusters within the Iris dataset.

Internal Cluster Measures for Iris Dataset: When comparing our spectral clustering framework with k-Means clustering, we observe differences in performance across internal cluster measures. k-Means clustering demonstrates better results in these measures (see Figure 6.2). For instance, the silhouette score for k-Means is 0.55, while for spectral clustering it is 0.48. Similarly, the Calinski-Harabasz (CH) score is higher for k-Means at 560.4 compared to 451.07 for spectral clustering. Additionally, the Davies-Bouldin (DB) score is lower for k-Means at 0.66 compared to 0.80 for spectral clustering.

These differences can be attributed to the nature of the algorithms. k-Means relies on geometric centroids to form clusters, which can lead to well-defined spherical clusters that align well with the Euclidean distance metric used in silhouette and CH scores. On the other hand, spectral clustering considers the underlying graph structure and relationships between data points. It can capture non-linear and irregularly shaped clusters, leading to a lower silhouette score and CH score but higher DB score due to the emphasis on separation between clusters.

In summary, while k-Means performs better in internal cluster measures due to its simplicity and sensitivity to spherical clusters, spectral clustering excels in external cluster measures by capturing complex data structures and forming clusters that align well with true class labels. The differences in performance highlight the trade-offs between the two approaches and the strengths they bring to cluster analysis.

Visualization of clustering results of Iris data using PCA: For the Iris dataset, we begin by utilizing Principal Component Analysis (PCA) to reduce the dimensionality of the data to two components. Total explained variance of 2 components of the Iris data is 97.8% (component1: 92.4%, component2: 5.3%). This enables visual representation of the original dataset, k-Means clustering, and spectral clustering using our framework.

In (Figure 6.3a), the scatter plot visualizes the original Iris dataset. Subsequently, in (Figure 6.3b), we observe the scatter plot resulting from k-Means clustering. Here, it is evident that k-Means employs a linear decision boundary, attempting to divide

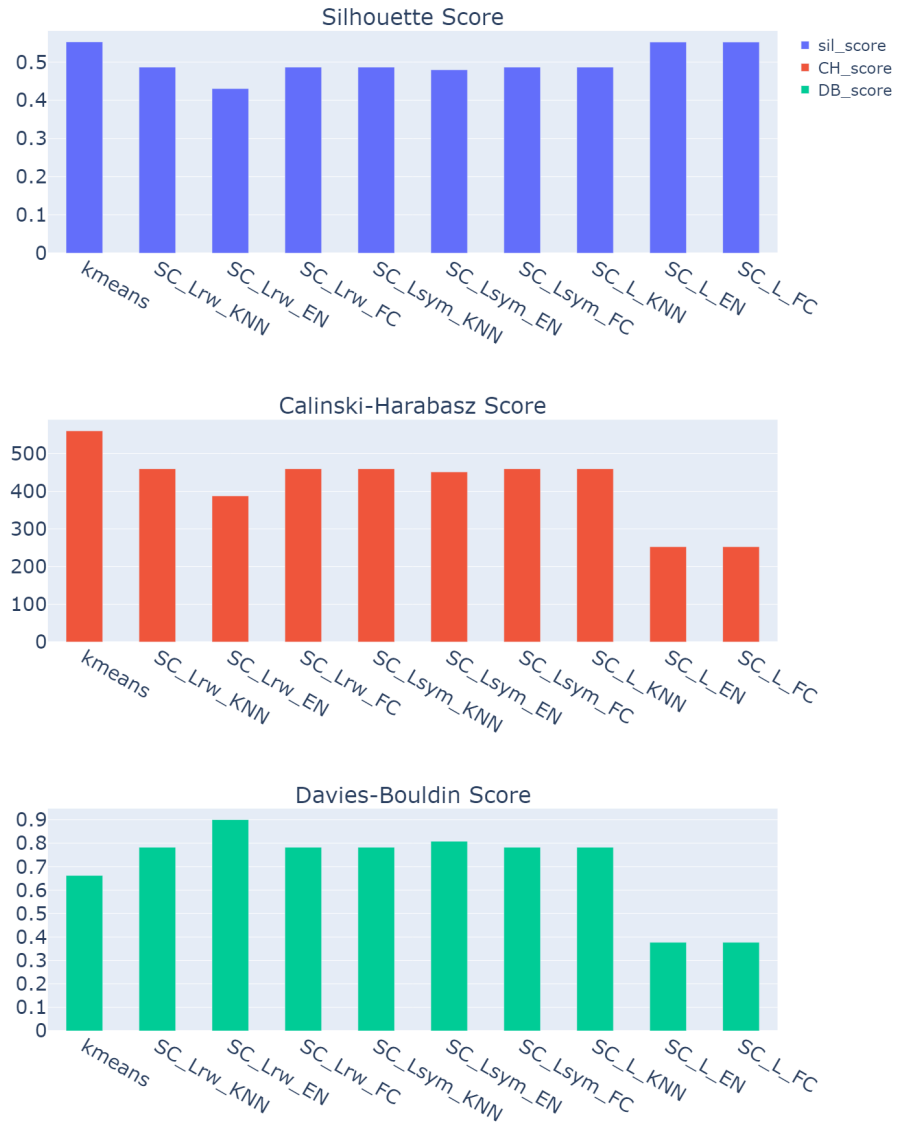


Figure 6.2: k-Means clustering and Spectral clustering performance comparison: the figure compares internal cluster validity measures (Silhouette Score, Calinski-Harabasz Score, and Davies-Bouldin Score) for the Iris dataset. Subplots contrast the k-Means approach with nine spectral clustering configurations, with naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph.

clusters 2 and 3 with a straight line. In contrast, (Figure 6.3c) depicts the scatter plot from spectral clustering using our framework. Notably, spectral clustering attempts to mimic the original dataset's structure in a nonlinear fashion, resulting in a better representation of the clusters' natural divisions.

This visualization validates our claim of spectral clustering's superiority over k-Means in accurately capturing the underlying cluster structure. The intuitive insight derived from this visualization reinforces our argument for placing more emphasis on external

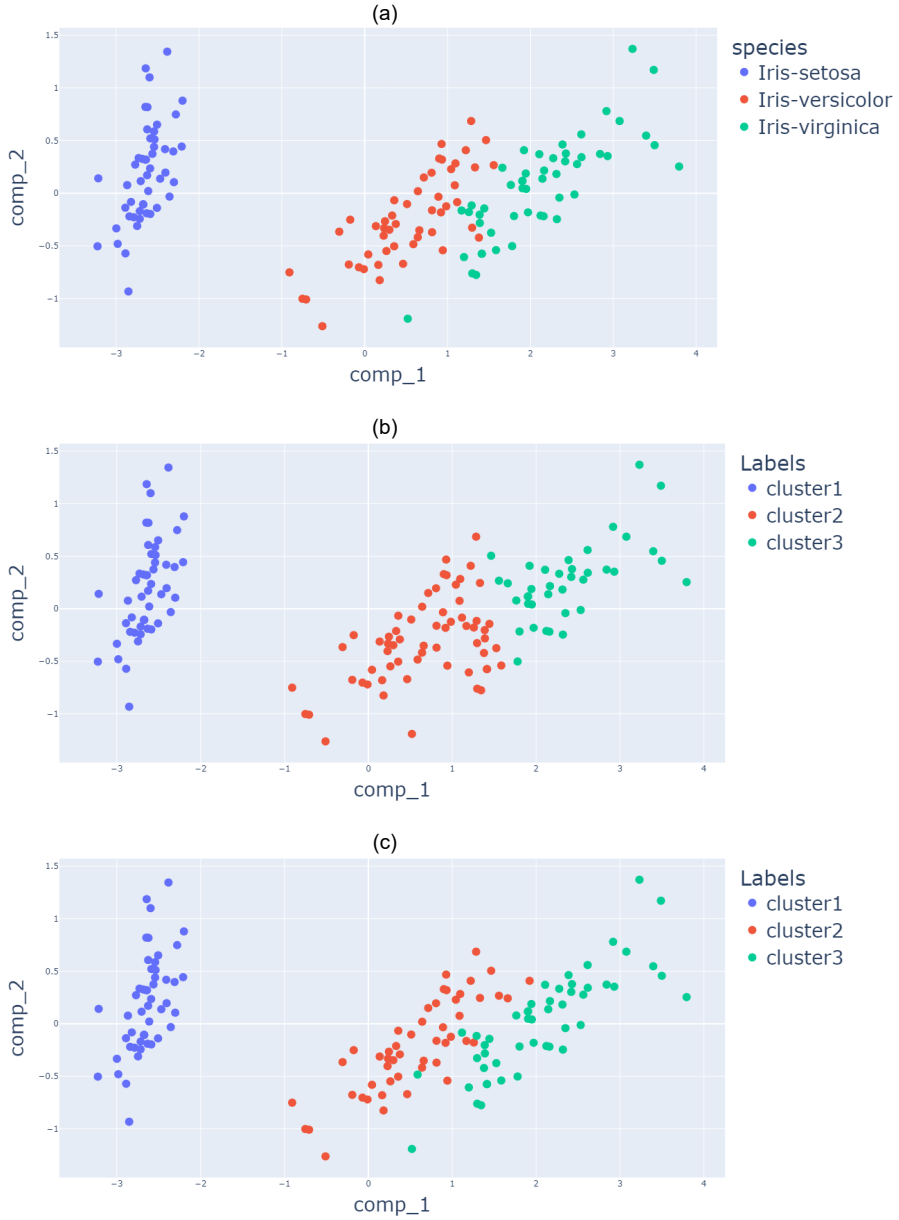


Figure 6.3: PCA visualization of Iris data: image(a) - reduced Iris data, image(b) - projection of Iris data after applying k-Means clustering, and image(c) - projection of Iris data after applying spectral clustering.

cluster measures, especially in the case of the Iris dataset. This is because spectral clustering more closely mirrors the ground truth, resulting in higher accuracy of clustering.

6.2.2 Comparison of Spectral clustering and k-Means clustering performance using CORA dataset

Now shifting our focus to the CORA dataset, we employed the same methodology to compare the outcomes of spectral clustering and k-Means clustering. Table 6.2 shows the evaluation of these cluster measures for both k-Means and spectral clustering.

Method	External measures			Internal measures		
	Accuracy	ARI	NMI	Sil. score	CH score	DB score
k-Means	0.221	0.066	0.102	**0.001**	20.692	6.704
$SC\_L_{rw}\_kNN$	**0.266**	0.179	0.315	-0.023	18.185	6.510
$SC\_L_{rw}\_EN$	0.104	0.122	0.201	-0.055	19.372	6.896
$SC\_L_{rw}\_FC$	0.171	0.095	0.158	0.000	**22.068**	6.443
$SC\_L_{sym}\_kNN$	0.222	**0.236**	**0.321**	-0.018	18.274	6.941
$SC\_L_{sym}\_EN$	0.195	0.176	0.262	-0.003	14.211	9.402
$SC\_L_{sym}\_FC$	0.224	0.125	0.155	-0.018	20.997	7.699
SC_L_kNN	**0.266**	0.179	0.315	-0.023	18.185	6.510
SC_L_EN	0.107	-0.000	0.008	-0.195	0.900	6.746
SC_L_FC	0.110	-o.001	0.003	-0.246	0.301	**2.106**

Table 6.2: Comparison of k-Means and Spectral Clustering Performance: the table compares our framework with k-Means clustering for CORA dataset. The best values across all the cluster validity measures and the various methods are highlighted in bold. We use the naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph for our framework.

External Cluster Measures for CORA Dataset: When comparing our spectral clustering framework with k-Means clustering on the CORA dataset, we observe significant differences in their performance across external cluster measures (see Figure 6.4). Our spectral clustering framework outperforms k-Means in terms of two out of three measures. The accuracy values for both clustering techniques were comparable, with spectral clustering achieving 22.2% accuracy and k-Means achieving only 22.1%. This slight difference can be attributed to the nature of the dataset and the challenges in accurately classifying the nodes in a network. However, when we examine the Adjusted Rand Index (ARI), our spectral clustering framework outperforms with a value of 0.23, significantly surpassing k-Means' ARI of 0.06. This substantial difference indicates that our framework is better at capturing the similarity between true and predicted cluster assignments. Similarly, the Normalized Mutual Information (NMI) values highlight the superiority of our framework. Spectral clustering achieves an NMI of 32.1%, whereas k-Means achieved only 10.2%. This difference underscores the framework's ability to identify clusters that align well with true class labels.

Internal Cluster Measures for CORA Dataset: When comparing our spectral clustering framework with k-Means clustering on the CORA dataset, the performance differences are observed mainly in the internal cluster measures is shown in (Figure 6.5). For the silhouette score, k-Means achieved a slightly higher value of 0.001 compared to spectral clustering's -0.018. This difference indicates that k-Means was able to create clusters with slightly better separation between them. In terms of the Calinski-Harabasz (CH) score, k-Means also demonstrated a slightly better result. The CH score for spectral clustering was 18.27, whereas k-Means achieved a higher score of 20.69. This suggests that k-Means was able to achieve better cluster compactness and separation. The Davies-Bouldin (DB) score, which evaluates

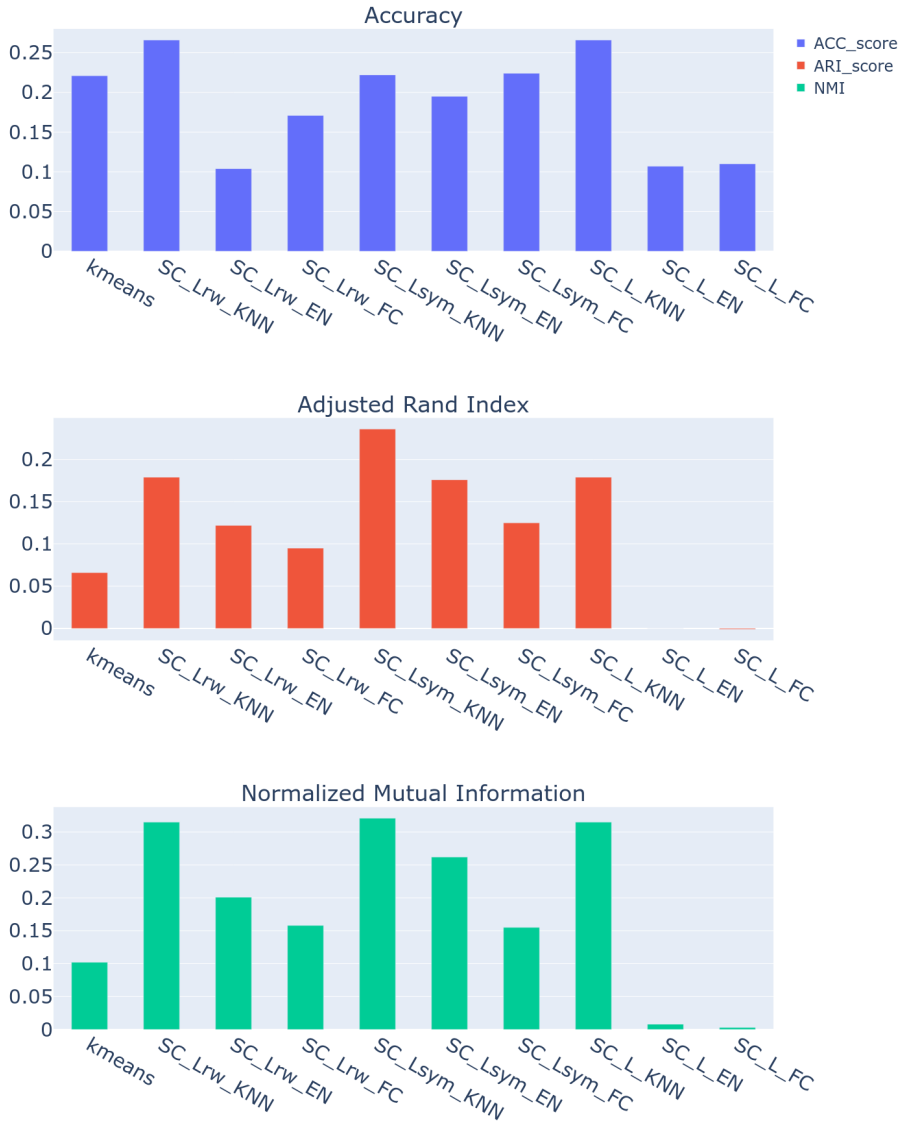


Figure 6.4: k-Means clustering and Spectral clustering performance comparison: the figure compares external cluster validity measures (Accuracy, Adjusted Rand Index, and Normalized Mutual Information) for the CORA dataset. Subplots contrast the k-Means approach with nine spectral clustering configurations. We use the naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph for our framework.

the average similarity between clusters, showed minor differences. The DB score for spectral clustering was 6.94, while k-Means achieved a slightly lower value of 6.70. This implies that k-Means managed to create clusters with slightly better intra-cluster similarity.

In summary, our spectral clustering framework shows enhanced clustering performance on the CORA dataset with external measures, as evidenced by superior ARI and

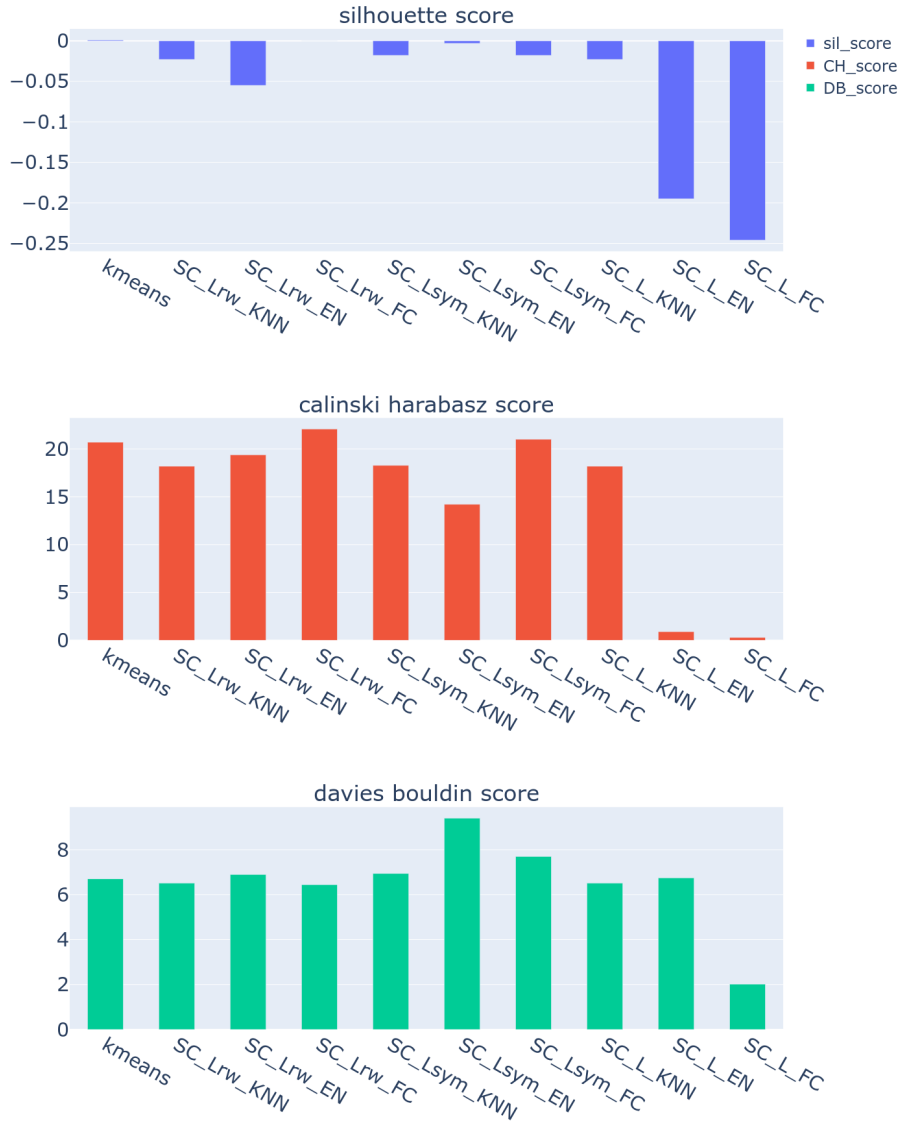


Figure 6.5: k-Means clustering and Spectral clustering performance comparison: the figure compares internal cluster validity measures (Silhouette Score, Calinski-Harabasz Score, and Davies-Bouldin Score) for the CORA dataset. Subplots contrast the k-Means approach with nine spectral clustering configurations, with naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph.

NMI values. While accuracy remains comparable due to the nature of the dataset. However, k-Means exhibited slightly better results in internal cluster measures on the CORA dataset, these differences are relatively minor. These variations in scores highlight the nuanced differences in cluster structures achieved by the two methods.

Visualization of clustering results of CORA data using PCA Turning our attention to the CORA dataset, which is high-dimensional, we again utilize PCA to project the data into two dimensions for visualization purposes. Figure 6.6a provides a scatter

plot of the reduced dataset. Subsequently, (Figure 6.6b) presents the visualization of k-Means clustering results, and (Figure 6.6c) shows the scatter plot for spectral clustering using our framework.

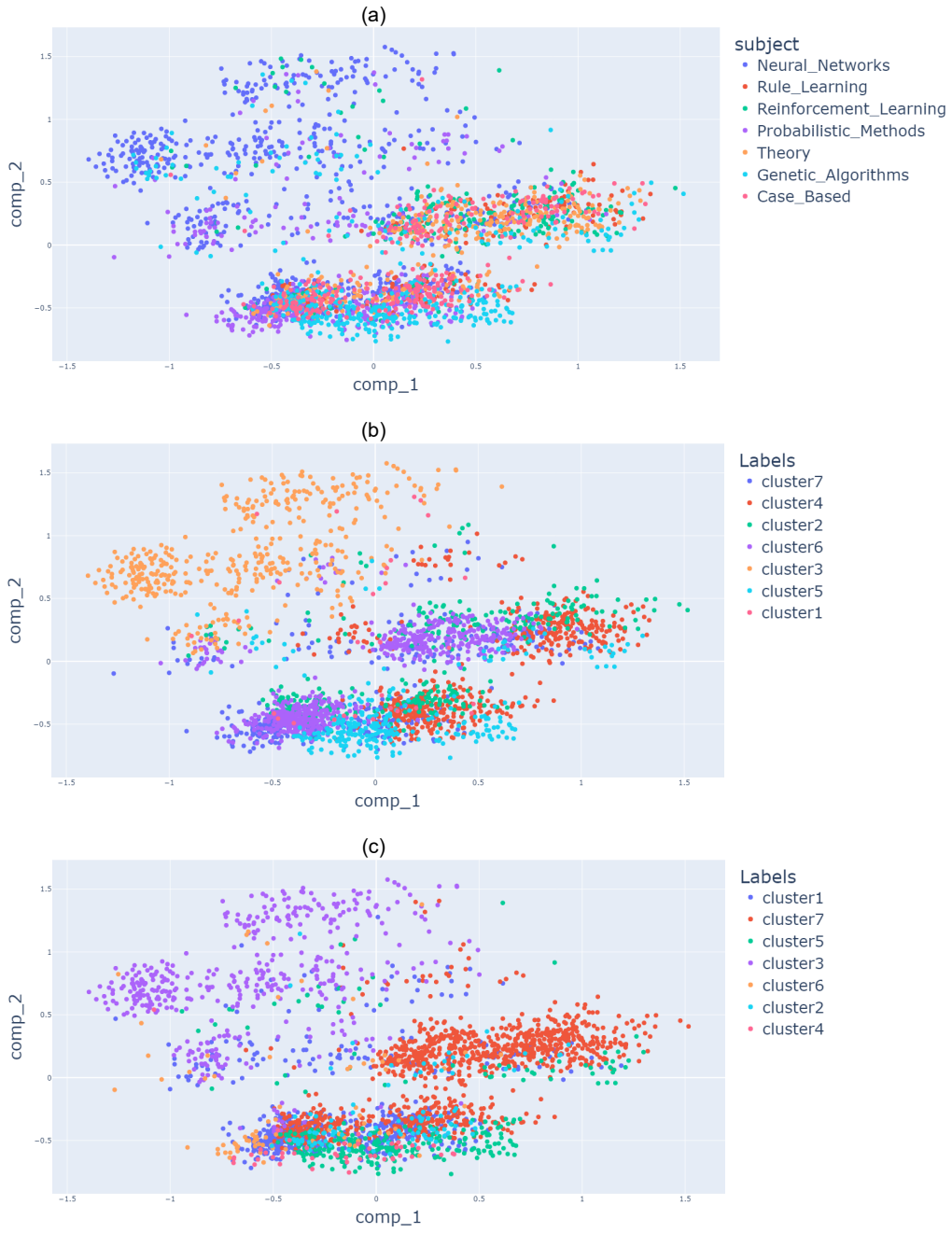


Figure 6.6: PCA visualization of CORA data: image(a) - reduced CORA data, image(b) - projection of CORA data after applying k-Means clustering, and image(c) - projection of CORA data after applying spectral clustering.

Upon examining Figure 6.6b, it is evident that k-Means clustering struggles with clusters that have overlapping and closely situated data points. For instance, while cluster 3 (Neural networks) is well identified, the other clusters exhibit mixed and

blended colors due to overlapping points. However, (Figure 6.6c) highlights a slightly improved visualization of spectral clustering. This method also distinguishes cluster 3 (Neural network) but apart from that it is quite hard to tell apart the other clusters. The unclear visualization for both k-Means and spectral clustering can be contributed by the high dimensionality of the CORA data and a total explained variance of only 3.4% for two principal components, which also explains the low performance across measures in both dataset.

Insights into visualization of clustering results The visual intuition derived from these scatter plots aligns with the quantitative results. It demonstrates how spectral clustering, when combined with our framework, captures intricate cluster patterns more accurately than k-Means, which relies on linear decision boundaries [Bishop and Nasrabadi, 2006]. These visual insights support our rationale for prioritizing external cluster measures over internal measures in this case. However, for both clustering techniques applied on CORA dataset, it is hard to observe clear and distinct clusters. The somewhat close resemblance of spectral clustering to the original distribution underscores its effectiveness, and this alignment with ground truth is observed due to the external validation.

The visual analysis of clustering results through scatter plots offers intuitive confirmation of spectral clustering's superior performance and justifies our emphasis on external cluster measures, particularly when ground truth labels are available. It is essential to recognize that such visual validation complements numerical analyses and supports the comprehensive understanding of clustering outcomes.

6.2.3 Insight into graph Laplacian and similarity graph choices

An additional advantage of our spectral clustering framework lies in its ability to provide insight into the optimal Laplacian and similarity graph choices. For the Iris dataset, the symmetric normalized Laplacian (L_{sym}) in combination with the epsilon neighborhood graph emerges as a promising choice after observing all the performance values as shown in Table 6.1. Notably, both the normalized Laplacians (L_{sym} and L_{rw}) demonstrated improved performance over k-Means (Figure 6.1 for external measures and figure 6.2 for internal measures), particularly in terms of ARI and NMI, reinforcing their utility for spectral clustering in this context. As our framework provides results from multiple Laplacian variants, one can choose the clustering results that are most appropriate for the desired use case.

For the CORA dataset, the symmetric normalized Laplacian (L_{sym}) with the k-Nearest Neighbors (kNN) graph yielded the best results. Furthermore, both normalized Laplacians demonstrated enhanced performance compared to k-Means, indicating their effectiveness in capturing the underlying patterns in the data. Interestingly, all three Laplacian options in combination with the kNN graph outperformed k-Means in terms of external cluster measures (Figure 6.4), suggesting the efficacy of kNN graph in representing the relationships within the CORA dataset

6.3 (RQ3): Comparing our spectral clustering framework with other spectral clustering approach:

To validate the performance of our framework, we conduct a comparison with the work of [Somashekara and Manjunatha, 2014], utilizing the Iris dataset as shown in (Table 6.3). We employ six distinct cluster validity measures, encompassing both internal and external evaluation metrics, for evaluating the performances of both spectral clustering approaches.

Our approach consistently surpasses the Jordan approach in terms of multiple cluster validity measures. Notably, our approach exhibits superior performance for both Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) across all nine combinations (refer to Figure 6.7). Furthermore, in the context of internal measures, our approach consistently achieves improved results (see Figure 6.8). It's worth highlighting that our approach consistently outperforms the Jordan approach across all six measures.

Method	External measures			Internal measures		
	Accuracy	ARI	NMI	Sil. score	CH score	DB score
Jordan	0.793	0.554	0.600	0.478	417.24	0.683
$SC\_L_{rw}\_kNN$	0.366	**0.903**	**0.899**	0.486	**459.641**	0.782
$SC\_L_{rw}\_EN$	0.906	0.759	0.695	0.430	387.551	0.900
$SC\_L_{rw}\_FC$	0.366	**0.903**	**0.899**	0.486	**459.641**	0.782
$SC\_L_{sym}\_kNN$	0.366	**0.903**	**0.899**	0.486	**459.641**	0.782
$SC\_L_{sym}\_EN$	**0.946**	0.850	0.830	0.479	451.074	0.808
$SC\_L_{sym}\_FC$	0.366	**0.903**	**0.899**	0.486	**459.641**	0.782
SC_L_kNN	0.366	**0.903**	**0.899**	0.486	**459.641**	0.782
SC_L_EN	0.0.660	0.558	0.720	**0.552**	252.563	**0.377**
SC_L_FC	0.0.660	0.558	0.720	**0.552**	252.563	**0.377**

Table 6.3: Comparison of Spectral Clustering Performance: the table compares our framework with spectral clustering by Jordan approach suggested by [Somashekara and Manjunatha, 2014] for Iris dataset. The best values across all the cluster validity measures and the various methods are highlighted in bold. We use the naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph for our framework.

In a specific instance, focusing on the ($SC\_L\_sym\_EN$) combination, our approach showcases remarkable enhancements. This includes an impressive 15.33% increase in Accuracy, a substantial 53.92% enhancement in ARI, and a notable 38.39% improvement in NMI. Moreover, for the internal measures, our approach demonstrates a similar performance in silhouette score with a very slight betterment of about 0.17% and a significant betterment of approximately 8.07% in the Calinski-Harabasz (CH)

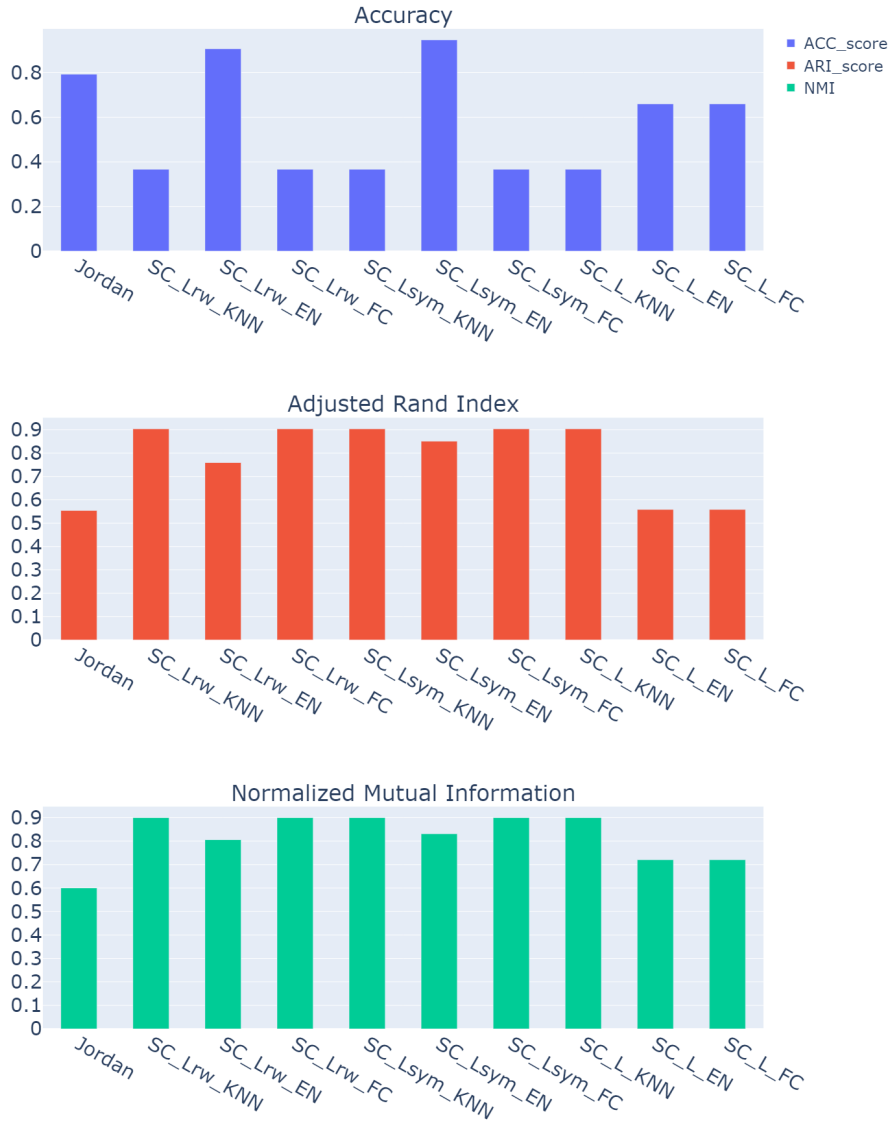


Figure 6.7: Spectral clustering performance comparison between Jordan approach and our framework: the figure compares external cluster validity measures (Accuracy, Adjusted Rand Index, and Normalized Mutual Information) for the Iris dataset. Subplots contrast the Jordan approach by [Somashekara and Manjunatha, 2014] with nine spectral clustering configurations, with naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph.

score. However, it's important to acknowledge that in terms of the Davies-Bouldin (DB) score, the approach proposed by [Somashekara and Manjunatha, 2014] exhibits slightly better performance. Their method achieves a DB score of 0.68 compared to our approach's DB score of 0.80. This suggests that their method's results are marginally more favorable in terms of the specific measure of cluster similarity indicated by the DB score.

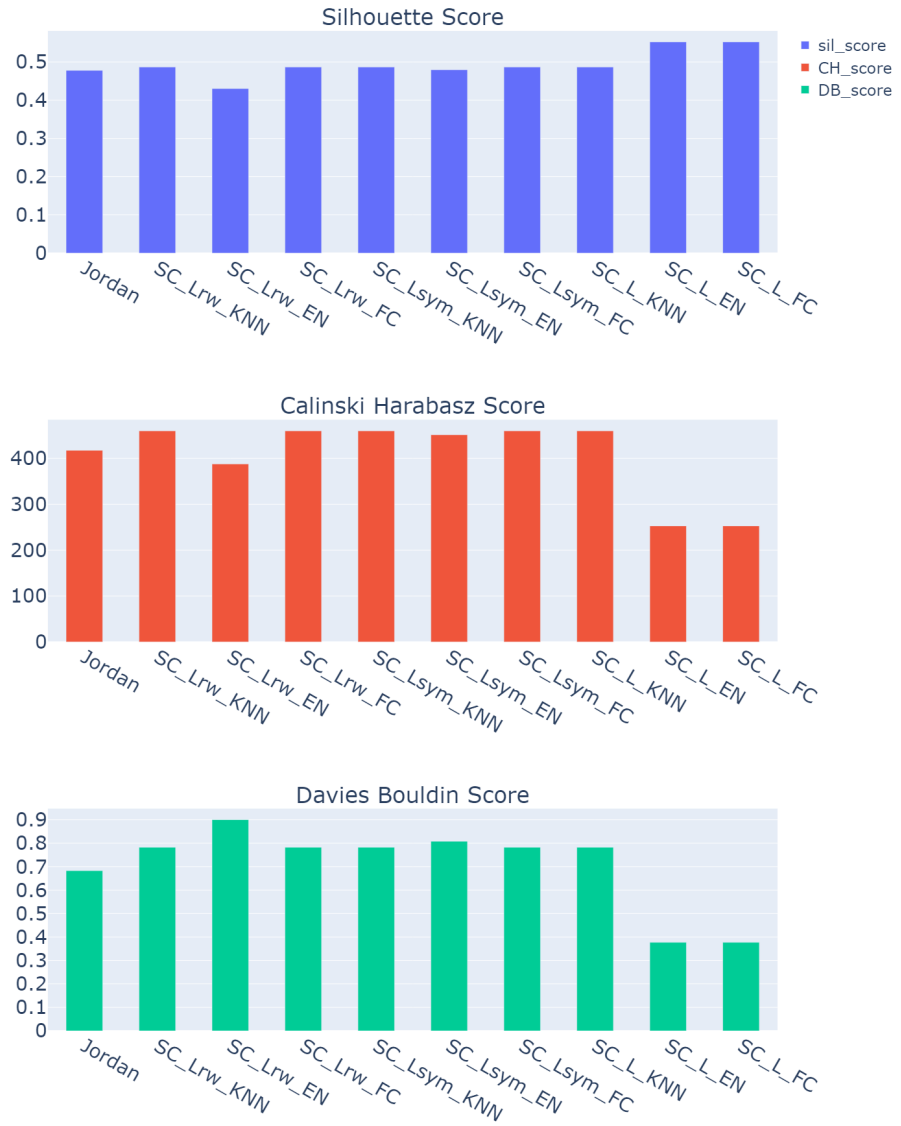


Figure 6.8: Spectral clustering performance comparison between Jordan approach and our framework: the figure compares internal cluster validity measures (Silhouette Score, Calinski-Harabasz Score, and Davies-Bouldin Score) for the Iris dataset. Subplots contrast the Jordan approach by [Somashekara and Manjunatha, 2014] with nine spectral clustering configurations, with naming convention, SC-spectral clustering, L_{rw}-random walk normalized Laplacian, L_{sym}-symmetric normalized Laplacian, L-unnormalized Laplacian, KNN-k nearest neighbor graph, EN-epsilon neighborhood graph, and FC-fully connected graph.

In summary, our approach consistently showcases its ability to yield superior clustering outcomes across a range of combinations and the majority of the evaluated metrics. This success can be attributed to the fine-tuning of hyperparameters associated with different similarity graph options, as well as the flexibility to choose diverse graph Laplacians. Notably, the approach presented by [Somashekara and Manjunatha, 2014] does not provide explicit details about the hyperparameters used in their evaluation, which could potentially contribute to the differences in performance observed.

7. Conclusion and future work

In this chapter, we summarize our findings and contributions in section 7.1 and the possibilities of extending our framework in section 7.2.

7.1 Conclusion

Through the implementation of our framework, we extend the possibilities of spectral clustering. In conclusion, this thesis has successfully addressed the research questions posed, demonstrating the flexibility and adaptability of the proposed "User Controlled Spectral Clustering" framework across various aspects of spectral clustering analysis. We have addressed this thesis by answering the three research questions stated in the beginning.

The investigation of our first research question (RQ1) centered around the customization capability of our framework, has yielded clear and affirmative results. We have demonstrated how our framework surpasses the customization options offered by scikit-learn's spectral clustering. Specifically, we have elaborated on our framework's ability to address the limitations associated with distance metric selection, similarity graph construction, and graph Laplacian choices that are currently offered by the standard scikit-learn spectral clustering implementation. We addressed these limitations by fulfilling the three criteria stated in the beginning. We fulfill criteria(i) by enabling the construction of a pairwise distance matrix by calculating pairwise distances between data points, employing seven distinct distance metrics (cosine, Euclidean, sqeuclidean, Braycurtis, Canberra, correlation, and cityblock). We fulfill criteria (ii) by introducing in our framework three options (kNN graph, ϵneighborhood graph, and fully connected graph) for creating a similarity matrix. Our framework allows users to choose from three variants of the Laplacian matrix, the unnormalized Laplacian (L), random walk normalized Laplacian (L_{rw}), and symmetric normalized Laplacian ((L_{sym}). The different choices of graph Laplacian fulfills criteria (iii).

To address (RQ2) we compared the performance of our spectral clustering framework against k-Means clustering for both Iris and CORA datasets. Our framework consistently outperforms the k-Means clustering method in terms of external cluster

measures for both the Iris and CORA dataset. Notably, our framework achieves an accuracy of 94.7%, demonstrating a substantial 5.4% improvement over k-Means. Additionally, the Adjusted Rand Index (ARI) shows a remarkable 12.1% increase, reaching 0.851, while the Normalized Mutual Information (NMI) value improves by 9.4% to 0.831. In the context of the CORA dataset, our spectral clustering framework, specifically for the ARI and NMI values demonstrated superior performance. The ARI value for our framework reaches 0.23, displaying a substantial improvement compared to k-Means' 0.06. Similarly, the NMI value for spectral clustering is 32.1%, significantly surpassing k-Means' 10.2%.

By comparing the performance of our spectral clustering framework with existing spectral clustering approach (by [Somashekara and Manjunatha, 2014]), we have answered (RQ3). Our framework has consistently demonstrated substantial enhancements across multiple cluster validity measures, both external and internal. Notably, the combination of the symmetric normalized Laplacian(L_{sym}) with the epsilon neighborhood graph exhibits significant improvements, including a 15.33% increase in Accuracy, a remarkable 53.92% increase in ARI, and a notable 38.39% increase in NMI. The internal measures also show improvements, with a substantial 8.07% rise in the Calinski-Harabasz (CH) score and a similar silhouette score with a minor gain of 0.17%.

7.2 Future work

The completion of this thesis opens the door to a multitude of future possibilities for extending and enhancing the proposed "User Controlled Spectral Clustering" framework. While numerous avenues for exploration exist, several ideas stand out as immediate and promising directions for future endeavors.

1. Incorporation of Graph Input and Utilizing Graph Databases: One notable extension involves expanding the framework's input capabilities beyond the current reliance on CSV files. This could entail graphs with edges that are not solely based on some kind of similarity. Additionally, the framework could be extended to integrated with graph databases such as Neo4j, providing a scalable and efficient storage solution for complex graph data. This advancement would allow the framework to handle graph data formats and not just CSV files, enhancing its applicability to a wider range of real-world scenarios such as social networks, recommendations engines, and healthcare to name a few.

2. Automation of Hyperparameter Tuning: Given the multitude of hyperparameters (e.g. distance metric, similarity graph choices, hyperparameters within specific similarity graph such as σ, ϵ, and number of nearest neighbor n, graph Laplacian choices, number of clusters k) involved in the framework's customization, a natural progression would be the development of an automated hyperparameter tuning mechanism. This would alleviate the burden on users, especially those with limited domain knowledge, by dynamically selecting optimal hyperparameter settings. Machine learning techniques such as grid search, random search, or Bayesian optimization could be integrated to efficiently explore the hyperparameter space and arrive at the most suitable configuration, thus assisting the clustering process.

3. Extension to Multiple Clustering Techniques: While the framework currently offers the k-Means algorithm as the final clustering step, an intriguing future direction involves expanding this aspect. The framework could be enhanced to support a variety of clustering algorithms, e.g. hierarchical clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise.), and more. This expansion would make the framework a versatile platform where users can choose from a repertoire of clustering techniques, tailoring their approach to the specific characteristics of their data and research goals.

4. Development of an Online Tool with Graph Database Backend: As the demand for user-friendly and accessible data analysis tools continues to grow, the proposed framework could evolve into a web application. By providing an user-interface, where users could seamlessly apply spectral clustering by uploading their data to the web application and configure their choices of hyperparameters using drop down menus.

While there exists a myriad of possible extensions for the ”User Controlled Spectral Clustering” framework, the aforementioned ideas offer exciting and feasible directions for future research. By embracing graph inputs, automating hyperparameter tuning, expanding clustering technique options, and potentially transforming the framework into an online tool, this thesis can be extended to further impact the field of data analysis and spectral clustering.

Appendix

A.1 Graph Laplacians and their properties

The various graph laplacian properties/propositions and their proofs in this section are a direct reference from Von Luxburg [2007].

A.1.1 Unnormalized graph Laplacian

The unnormalized graph Laplacian matrix in the work of Von Luxburg [2007] is defined as:

$$L = D - W$$

Proposition 1 (Properties of L) *The matrix L satisfies the following properties:*

1. For every vector $f \in \mathbb{R}^n$ we have

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

2. L is symmetric and positive semi-definite.

3. The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector $\mathbb{1}$.

4. L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$.

Proof.
Part (1): By the definition of d_i,

$$f'Lf = f'Df - f'Wf = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij}$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} d_j f_j^2\right) = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2.$$

Part (2): The symmetry of L follows directly from the symmetry of W and D. The positive semidefiniteness is a direct consequence of Part (1), which shows that $f'Lf \geq 0$ for all $f \in \mathbb{R}^n$.

Part (3): Obvious.

Part (4) is a direct consequence of Parts (1) - (3).

Note that the unnormalized graph Laplacian does not depend on the diagonal elements of the adjacency matrix W. Each adjacency matrix which coincides with W on all off-diagonal positions leads to the same unnormalized graph Laplacian L. In particular, self-edges in a graph do not change the corresponding graph Laplacian.

Proposition 2 (Number of connected components and the spectrum of L) *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}A_1, ..., \mathbb{1}A_k$ of those components.*

Proof. We start with the case $k = 1$, that is the graph is connected. Assume that f is an eigenvector with eigenvalue 0. Then we know that

$$0 = f'Lf = \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2.$$

As the weights w_{ij} are non-negative, this sum can only vanish if all terms $w_{ij}(f_i - f_j)^2$ vanish. Thus, if two vertices v_i and v_j are connected (i.e., $w_{ij} > 0$), then f_i needs to equal f_j . With this argument, we can see that f needs to be constant for all vertices which can be connected by a path in the graph. Moreover, as all vertices of a connected component in an undirected graph can be connected by a path, f needs to be constant on the whole connected component. In a graph consisting of only one connected component, we thus only have the constant one vector $\mathbb{1}$ as eigenvector with eigenvalue 0, which obviously is the indicator vector of the connected component.

Now consider the case of k connected components. Without loss of generality, we assume that the vertices are ordered according to the connected components they belong to. In this case, the adjacency matrix W has a block diagonal form, and the same is true for the matrix L:

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}$$

Note that each of the blocks L_i is a proper graph Laplacian on its own, namely, the Laplacian corresponding to the subgraph of the i-th connected component. As it is the case for all block diagonal matrices, we know that the spectrum of L is given by the union of the spectra of L_i, and the corresponding eigenvectors of L are the eigenvectors of L_i, filled with 0 at the positions of the other blocks. As each L_i is a graph Laplacian of a connected graph, we know that every L_i has eigenvalue 0 with multiplicity 1, and the corresponding eigenvector is the constant one vector on the i-th connected component. Thus, the matrix L has as many eigenvalues 0 as there are connected components, and the corresponding eigenvectors are the indicator vectors of the connected components.

A.1.2 The normalized graph Laplacians

There are two matrices which are called normalized graph Laplacians. Both matrices are closely related to each other and are defined in the work of Von Luxburg [2007] as:

$$L_{sym} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$
$$L_{rw} := D^{-1} L = I - D^{-1} W$$

We denote the first matrix by L_{sym} as it is a symmetric matrix, and the second one by L_{rw} as it is closely related to a random walk. In the following, we summarize several properties of L_{sym} and L_{rw}. The standard reference for normalized graph Laplacians is Chung [1997].

Proposition 3 (Properties of L_{sym} and L_{rw}) *The normalized Laplacians satisfy the following properties:*

1. For every $f \in \mathbb{R}^n$ we have

$$f' L_{sym} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2 .$$

2. λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2} u$.

3. λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ and u solve the generalized eigenproblem $Lu = \lambda Du$.

4. 0 is an eigenvalue of L_{rw} with the constant one vector $\mathbb{1}$ as eigenvector. 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\mathbb{1}$.

5. L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq ... \leq \lambda_n$.

Proof.

Part (1) can be proved similarly to Part (1) of Proposition 1.

Part (2) can be seen immediately by multiplying the eigenvalue equation $L_{sym} w = \lambda w$ with $D^{-1/2}$ from the left and substituting $u = D^{-1/2} w$.

Part (3) follows directly by multiplying the eigenvalue equation $L_{rw} u = \lambda u$ with D from the left.

Part (4): The first statement is obvious as $L_{rw}\mathbb{1} = 0$, the second statement follows from (2).

Part (5): The statement about L_{sym} follows from (1), and then the statement about L_{rw} follows from (2).

As it is the case for the unnormalized graph Laplacian, the multiplicity of the eigenvalue 0 of the normalized graph Laplacian is related to the number of connected components:

Proposition 4 (Number of connected components and spectra of L_{sym} and L_{rw}) *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of both L_{rw} and L_{sym} equals the number of connected components $A_1, ..., A_k$ in the graph. For L_{rw}, the eigenspace of 0 is spanned by the indicator vectors $\mathbb{1}_{A_i}$ of those components. For L_{sym}, the eigenspace of 0 is spanned by the vectors $D^{1/2}\mathbb{1}_{A_i}$.*

Proof. The proof is analogous to the one of Proposition 2, using Proposition 3.

A.2 Algorithms for spectral clustering

Below are three spectral clustering algorithms for the different graph Laplacians, as highlighted in the work of [Von Luxburg, 2007]. The assumptions are that our data consists of n data points, $x_1, ..., x_n$ which can be arbitrary objects. By using some similarity function we measure their similarities $s_{ij} = s(x_i, x_j)$, and $S = (s_{ij})i, j = 1...n.$ represents the corresponding similarity matrix.

It should be noted that the normalized spectral clustering algorithm (L_{rw}) employs the generalized eigenvectors of L, which correspond to the eigenvectors of the matrix L_{rw} according to Proposition 3 (A.1.2). As a result, the algorithm is known as normalized spectral clustering because it works with eigenvectors of the normalized Laplacian L_{rw}.

The normalized spectral clustering algorithm (L_{sym}) also employs a normalized Laplacian, but this time with the matrix L_{sym} rather than L_{rw}. This algorithm requires an additional row normalization step that the other algorithms do not.

Except for the fact that they use three different graph Laplacians, the three algorithms described above appear to be very similar. The main trick in all three algorithms is to change the representation of the abstract data points x_i to points $y_i \in \mathbb{R}^k$. This change in representation is useful because of the properties of the graph Laplacians. Afterward, clustering algorithms like k-means can detect clusters faster due to reduced dimensions and easier due to enhanced separation of clusters in the low dimensional representation [Jordan and Weiss, 2002].

Algorithm 2 Unnormalized spectral clustering $(L = D - W)$

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.
• Construct a similarity graph by one of the ways described in 2.3.2. Let W be its weighted adjacency matrix.
• Compute the unnormalized Laplacian L.
• **Compute the first k eigenvectors $u_1, ..., u_k$ of L.**
• Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, ..., u_k$ as columns.
• For $i = 1, ..., n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
• Cluster the points $(y_i)_{i=1,...,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, ..., C_k$.
Output: Clusters $A_1, ..., A_k$ with $A_i = \{j | y_j \in C_i\}$.

Algorithm 3 Normalized spectral clustering according to [Shi and Malik, 2000]

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.
• Construct a similarity graph by one of the ways described in 2.3.2. Let W be its weighted adjacency matrix.
• Compute the unnormalized Laplacian L.
• **Compute the first k generalized eigenvectors $u_1, ..., u_k$ of the generalized eigenproblem $Lu = \lambda Du$.**
• Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, ..., u_k$ as columns.
• For $i = 1, ..., n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
• Cluster the points $(y_i)_{i=1,...,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, ..., C_k$.
Output: Clusters $A_1, ..., A_k$ with $A_i = \{j | y_j \in C_i\}$.

Algorithm 4 Normalized spectral clustering according to [Jordan and Weiss, 2002]

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.
• Construct a similarity graph by one of the ways described in 2.3.2. Let W be its weighted adjacency matrix.
• Compute the normalized Laplacian L_{sym}.
• **Compute the first k eigenvectors $u_1, ..., u_k$ of L_{sym}.**
• Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, ..., u_k$ as columns.
• **Form the matrix $T \in \mathbb{R}^{n \times k}$ from U by normalizing the rows to norm 1,** that is set $t_{ij} = u_{ij} / (\sum_k u_{ik}^2)^{1/2}$.
• For $i = 1, ..., n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of T.
• Cluster the points $(y_i)_{i=1,...,n}$ with the k-means algorithm into clusters $C_1, ..., C_k$.
Output: Clusters $A_1, ..., A_k$ with $A_i = \{j | y_j \in C_i\}$.

A.3 Different similarity graph construction

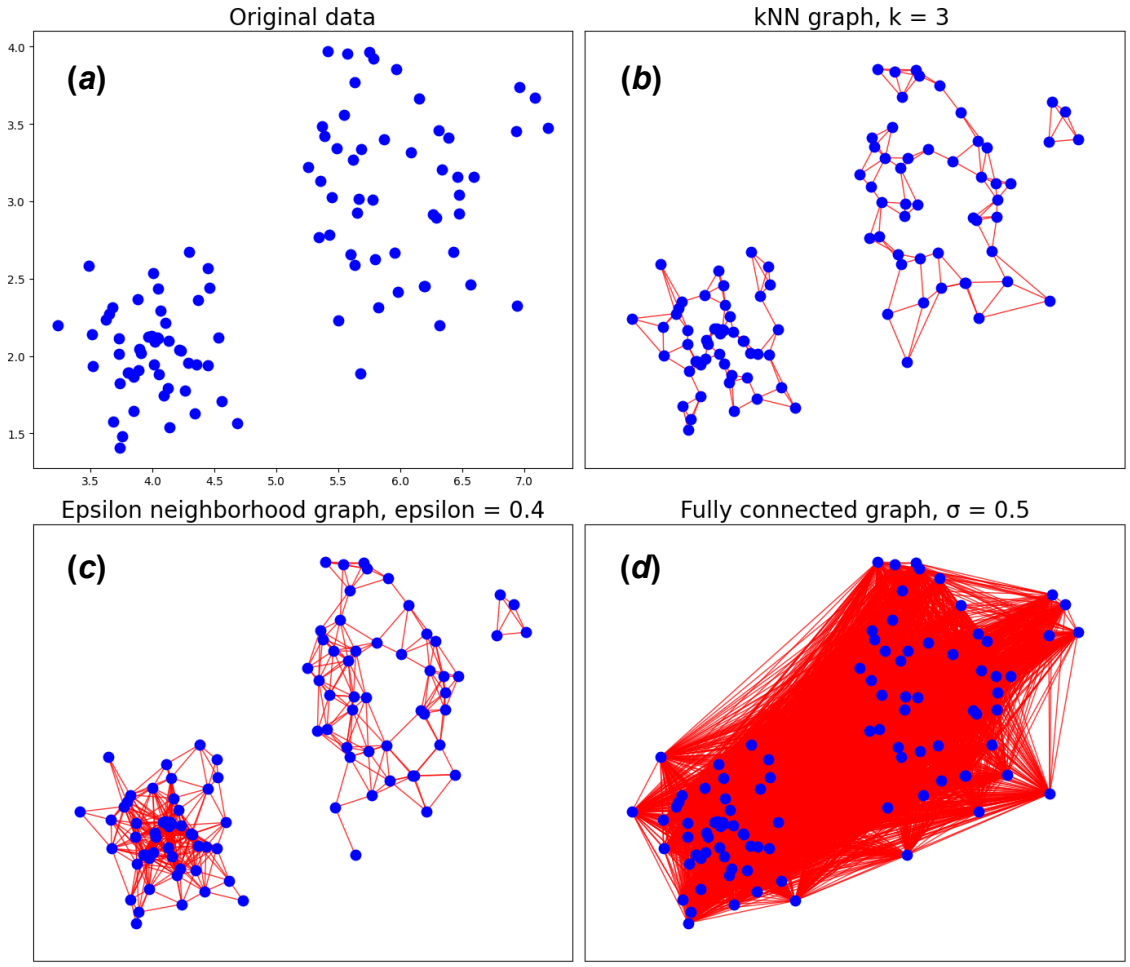


Figure A.1: Different similarity graph construction from a sample dataset. (a) Sample dataset with 100 data points that are randomly distributed, which can be visually partitioned into two groups (bottom left and the top right). The parameters for the different graph construction techniques are undesired and therefore it does not replicate the partitioning. (b) kNN graph constructed from the sample with k = 3, (c) ϵ-neighborhood graph constructed from the sample with $\epsilon = 0.4$, and (d) the fully connected graph with $\sigma = 0.5$.

A.4 Sensitivity analysis

All these metrics are aligned on a shared evaluation scale ranging from 0 to 1 (except for ARI, which ranges from -1 to 1).

Iris sensitivity analysis:

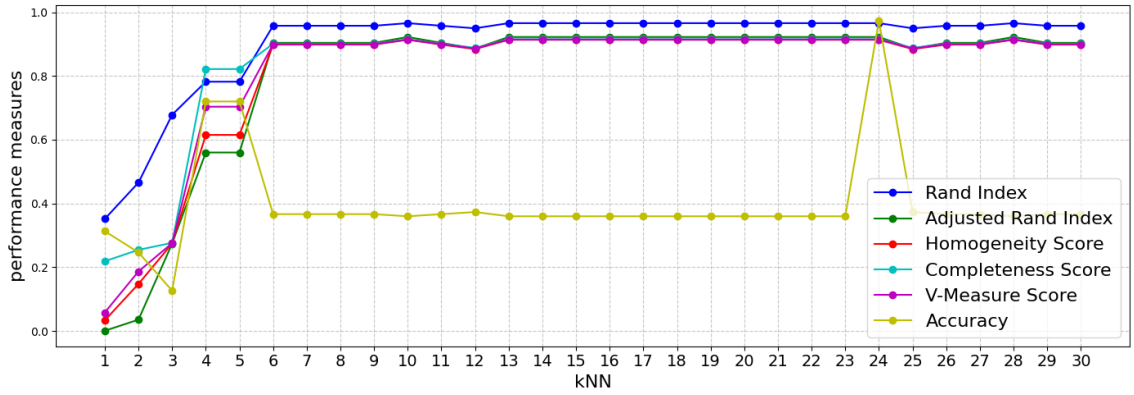


Figure A.2: Performance of Spectral Clustering on Iris ($k = 3$) using L and k nearest neighbor graph with varying kNN, optimal at 6.

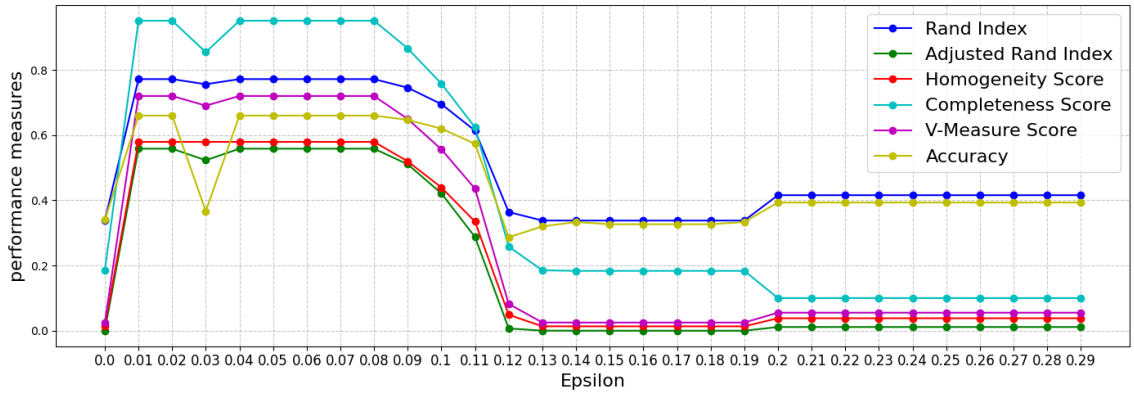


Figure A.3: Performance of Spectral Clustering on Iris ($k = 3$) using L and ϵ-neighborhood graph with varying ϵ, optimal at 0.01.

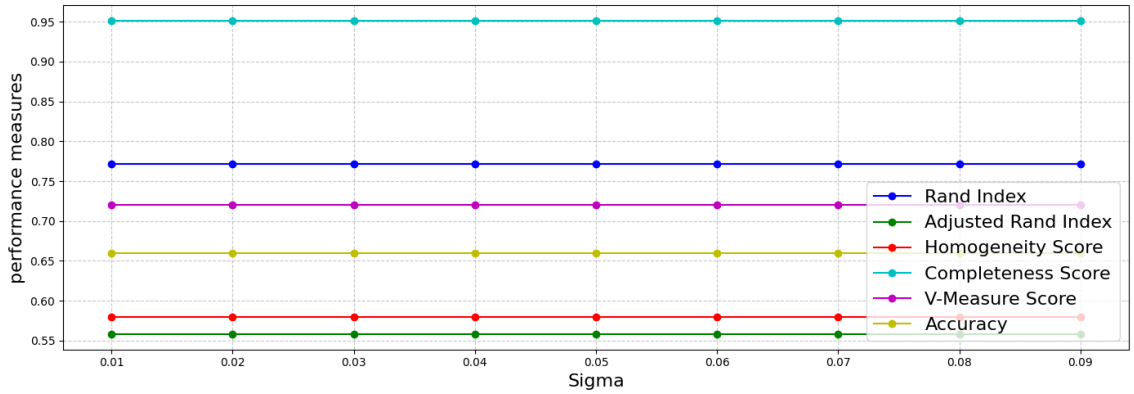


Figure A.4: Performance of Spectral Clustering on Iris ($k = 3$) using L and fully connected graph with varying σ, optimal at 0.01.

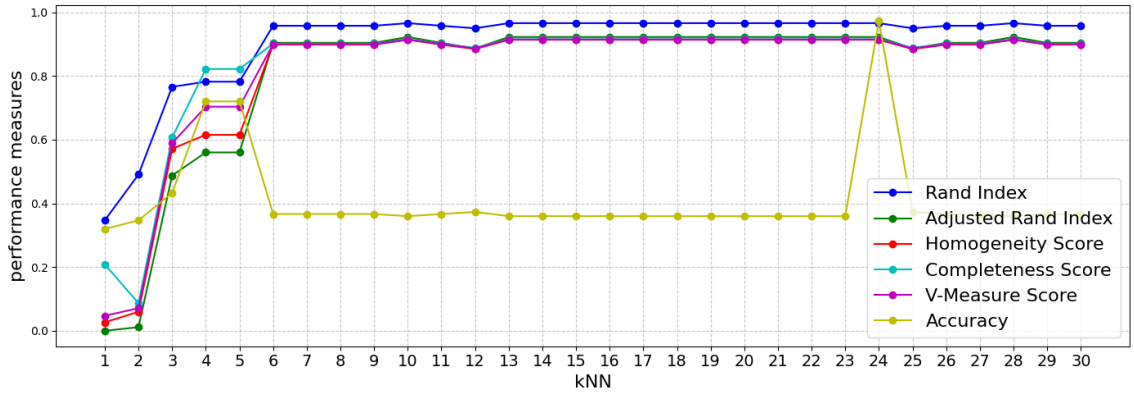


Figure A.5: Performance of Spectral Clustering on Iris ($k = 3$) using L_{rw} and k nearest neighbor graph with varying kNN, optimal at 6.

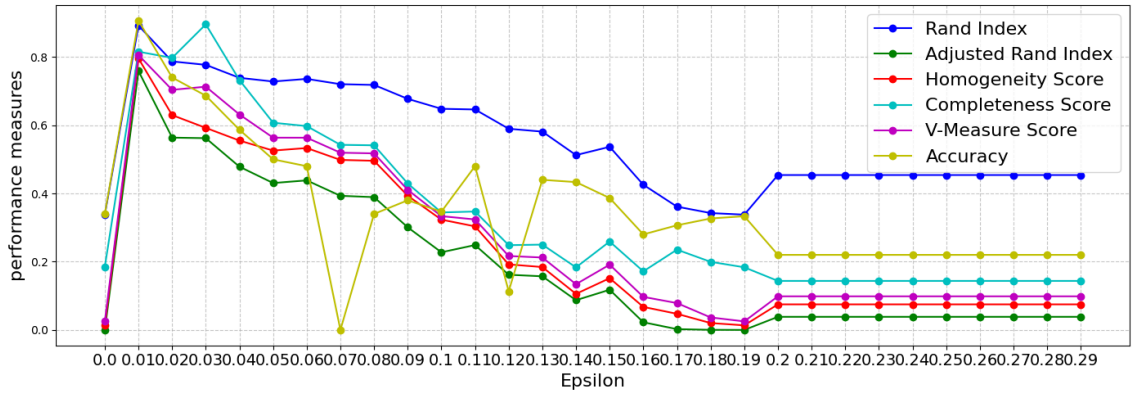


Figure A.6: Performance of Spectral Clustering on Iris ($k = 3$) using L_{rw} and ϵ-neighborhood graph with varying ϵ, optimal at 0.01.

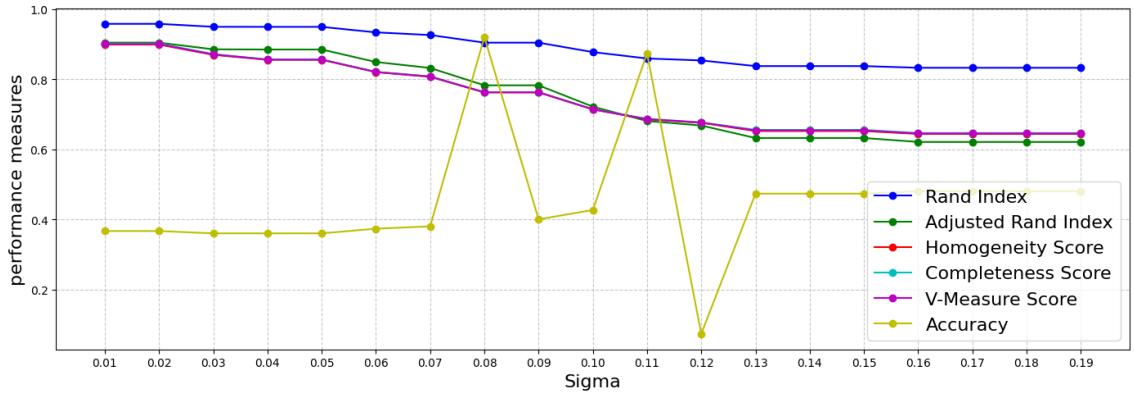


Figure A.7: Performance of Spectral Clustering on Iris ($k = 3$) using L_{rw} and fully connected graph with varying σ, optimal at 0.01.

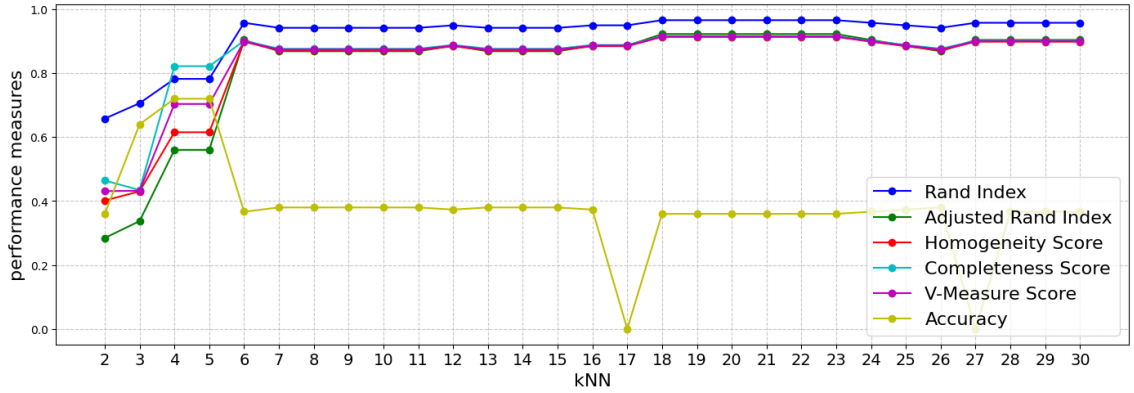


Figure A.8: Performance of Spectral Clustering on Iris ($k = 3$) using L_{sym} and k nearest neighbor graph with varying kNN, optimal at 6.

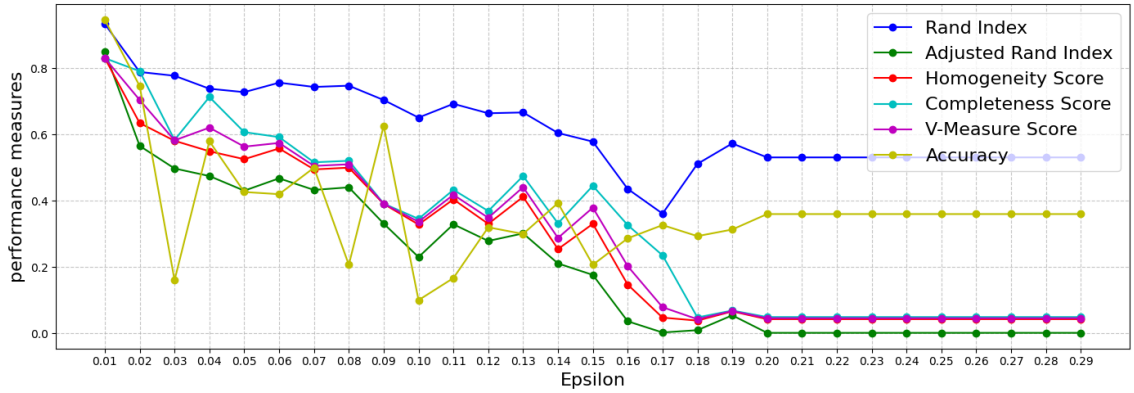


Figure A.9: Performance of Spectral Clustering on Iris ($k = 3$) using L_{sym} and ϵ-neighborhood graph with varying ϵ, optimal at 0.01.

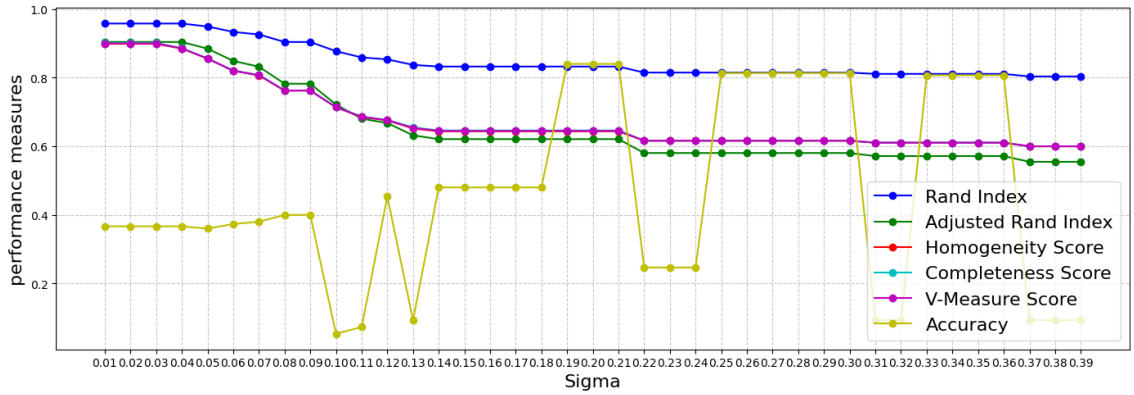


Figure A.10: Performance of Spectral Clustering on Iris ($k = 3$) using L_{sym} and fully connected graph with varying σ, optimal at 0.01.

CORA sensitivity analysis:

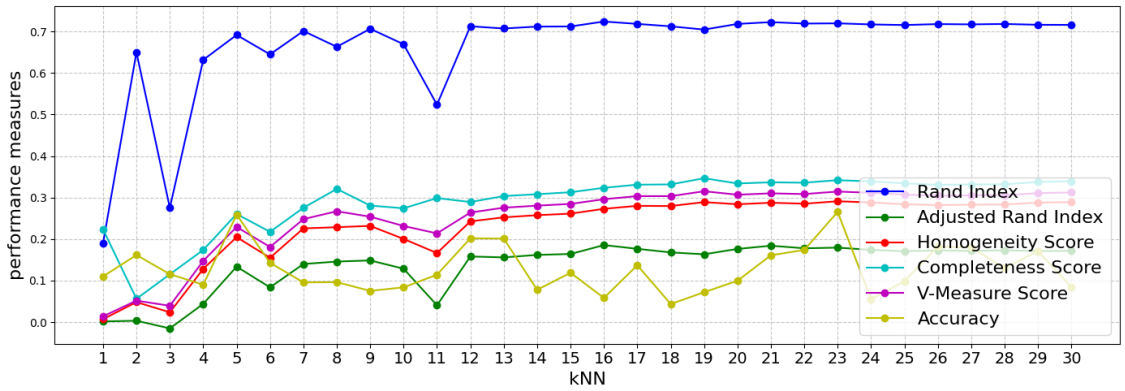


Figure A.11: Performance of Spectral Clustering on CORA ($k = 7$) using L and k nearest neighbor graph with varying kNN, optimal at 23.

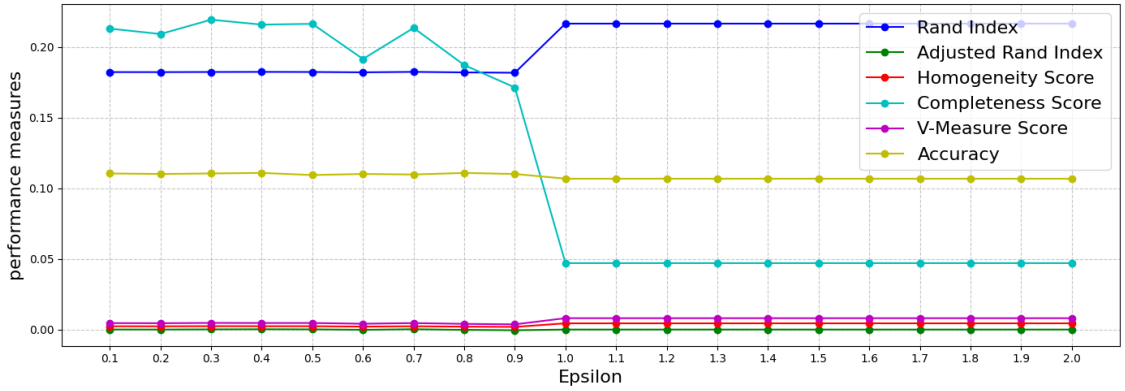


Figure A.12: Performance of Spectral Clustering on CORA ($k = 7$) using L and ϵ-neighborhood graph with varying ϵ, optimal at 1.

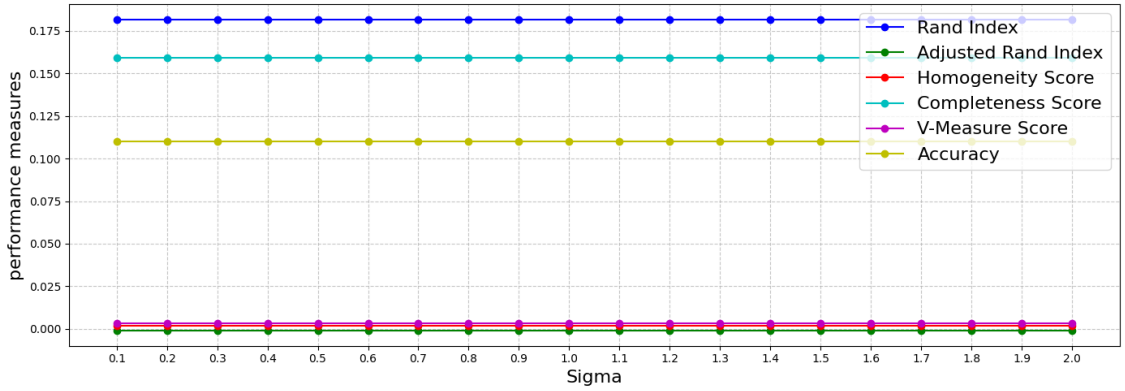


Figure A.13: Performance of Spectral Clustering on CORA ($k = 7$) using L and fully connected graph with varying σ, optimal at 1.

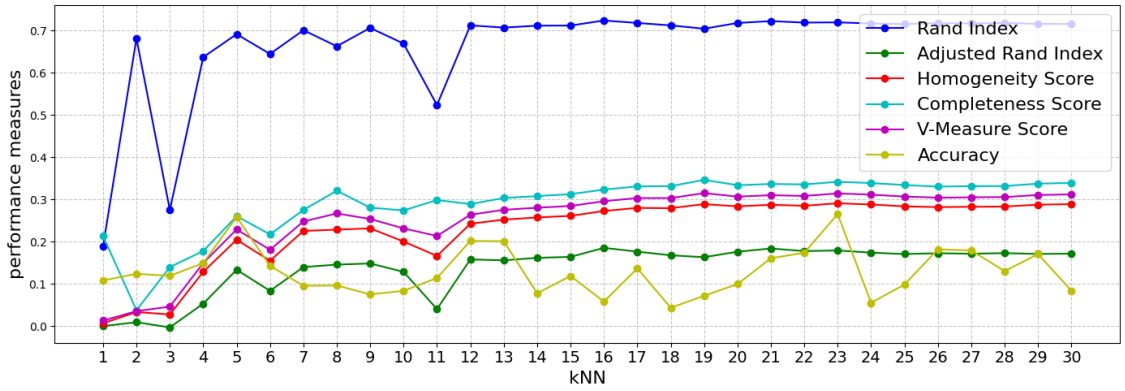


Figure A.14: Performance of Spectral Clustering on CORA ($k = 7$) using L_{rw} and k nearest neighbor graph with varying kNN, optimal at 23.

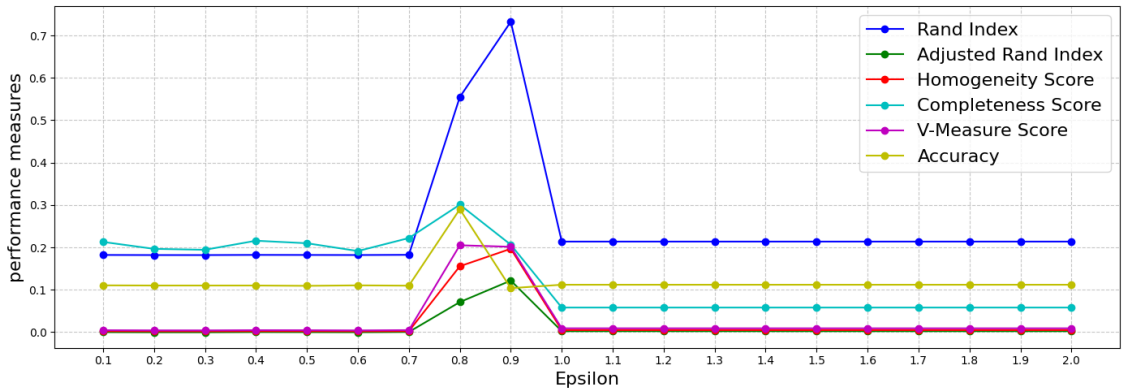


Figure A.15: Performance of Spectral Clustering on CORA ($k = 7$) using L_{rw} and ϵ-neighborhood graph with varying ϵ, optimal at 0.8.

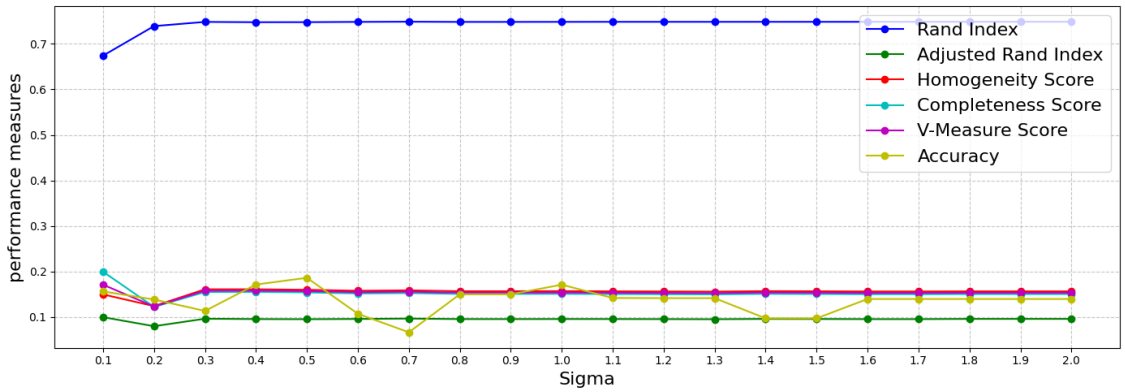


Figure A.16: Performance of Spectral Clustering on CORA ($k = 7$) using L_{rw} and fully connected graph with varying σ, optimal at 0.4.

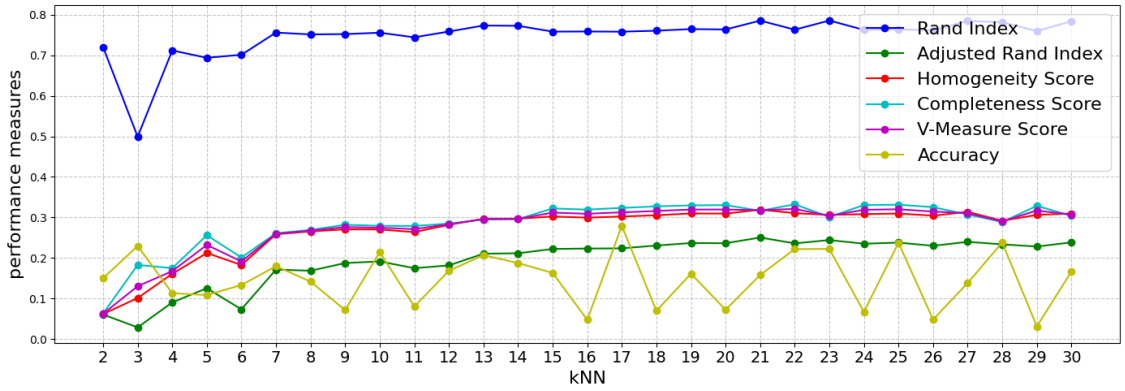


Figure A.17: Performance of Spectral Clustering on CORA ($k = 7$) using L_{sym} and k nearest neighbor graph with varying kNN, optimal at 22.

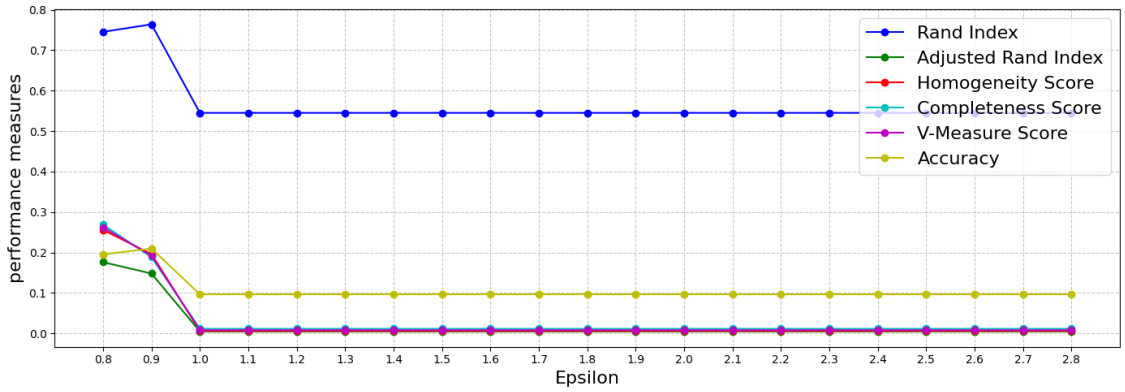


Figure A.18: Performance of Spectral Clustering on CORA ($k = 7$) using L_{sym} and ϵ-neighborhood graph with varying ϵ, optimal at 0.8.

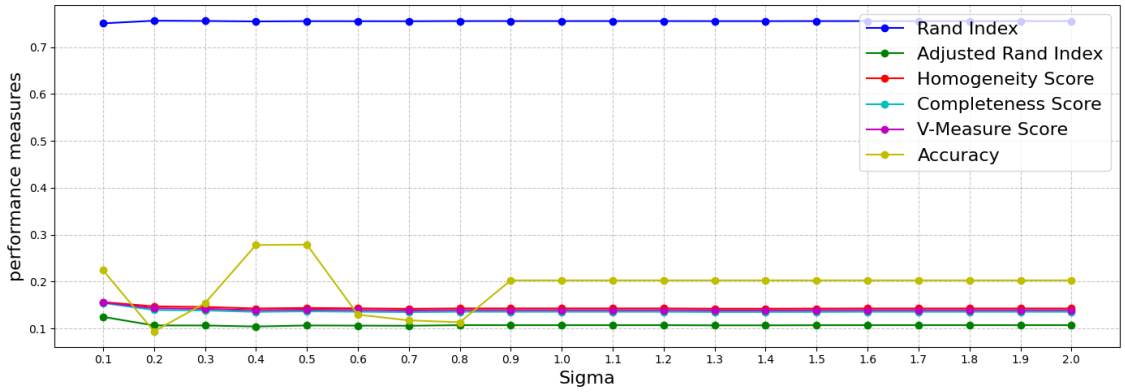


Figure A.19: Performance of Spectral Clustering on CORA ($k = 7$) using L_{sym} and fully connected graph with varying σ, optimal at 0.1.

Bibliography

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005. (cited on Page 1)

Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2010. (cited on Page 1)

Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. 2010. (cited on Page 35)

Bernard Aupetit. *A primer on spectral theory*. Springer Science & Business Media, 2012. (cited on Page 19)

AL Barabási. Network science. cambridge university press, cambridge, 2016. (cited on Page 10)

Albert-Laszlo Barabasi and Zoltan N Oltvai. Network biology: understanding the cell's functional organization. *Nature reviews genetics*, 5(2):101–113, 2004. (cited on Page 1)

Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1): 87–90, 1958. (cited on Page 18)

Asa Ben-Hur and Isabelle Guyon. Detecting stable clusters using principal component analysis. *Functional Genomics: Methods and Protocols*, pages 159–182, 2003. (cited on Page 35)

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012. (cited on Page 35)

Norman Biggs. *Algebraic graph theory*. Number 67. Cambridge university press, 1993. (cited on Page 19)

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006. (cited on Page 1, 23, 45, 49, and 79)

Katy Börner, Weixia Huang, Micah Linnemeier, Russell Duhon, Patrick Phillips, Nianli Ma, Angela Zoss, Hanning Guo, and Mark Price. Rete-netzwerk-red: analyzing and visualizing scholarly networks using the network workbench tool. *Scientometrics*, 83(3):863–876, 2010. (cited on Page 9)

Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient nd image segmentation. *International journal of computer vision*, 70(2):109–131, 2006. (cited on Page 21)

Ulrik Brandes. *Network analysis: methodological foundations*, volume 3418. Springer Science & Business Media, 2005. (cited on Page 19)

J Roger Bray and John T Curtis. An ordination of the upland forest communities of southern wisconsin. *Ecological monographs*, 27(4):326–349, 1957. (cited on Page 39)

Markus Brede. Book review: Networks-an introduction by mark ej newman. *Artificial Life*, 18(2):241–242, 2012. (cited on Page 9)

Leo Breiman. *Classification and regression trees*. Routledge, 2017. (cited on Page 1)

AE Brouwer and WH Haemers. Spectra of graphs. 2012. (cited on Page 9)

Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974. (cited on Page 40 and 41)

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1721–1730, 2015. (cited on Page 36)

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009. (cited on Page 29)

Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009. (cited on Page 51)

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997. (cited on Page 19, 21, 26, 34, and 89)

T Cormen, C Leiserson, R Rivest, and Clifford Stein. Book: introduction to algorithms, 2009a. (cited on Page 17, 18, and 19)

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Computational geometry. *Introduction to Algorithms, 3rd ed.; The MIT Press: Cambridge, MA, USA*, pages 1022–1027, 2009b. (cited on Page 7)

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995. (cited on Page 1)

David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979. (cited on Page 40 and 41)

Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022. (cited on Page 18)

Richard C Dorf and James A Svoboda. Circuit theorems. In *Introduction to Electric Circuits*, pages 162–207. Wiley, 2010. (cited on Page 8)

Richard O Duda, Peter E Hart, and David G Stork. Pattern classification, john willey & sons. *Inc., second edition edition*, 2001. (cited on Page 29)

Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972. (cited on Page 18)

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996. (cited on Page 2)

Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster analysis*. John Wiley & Sons, 2011. (cited on Page 39)

Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. (cited on Page 18)

Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010. (cited on Page 21)

Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2001. (cited on Page 9)

Michael T Goodrich, Roberto Tamassia, and Michael H Goldwasser. *Data structures and algorithms in Java*. John wiley & sons, 2014. (cited on Page 16 and 19)

Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. CRC press, 2005. (cited on Page 8 and 15)

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009. (cited on Page 1)

Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006. (cited on Page 9)

Joshua Zhexue Huang, Michael K Ng, Hongqiang Rong, and Zichen Li. Automated variable weighting in k-means type clustering. *IEEE transactions on pattern analysis and machine intelligence*, 27(5):657–668, 2005. (cited on Page 35)

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2:193–218, 1985. (cited on Page 41 and 42)

Anil K Jain. *Fundamentals of digital image processing*. Prentice-Hall, Inc., 1989. (cited on Page 19)

Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988. (cited on Page 2, 39, and 40)

Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999. (cited on Page 29 and 36)

Hongjie Jia, Shifei Ding, Xinzheng Xu, and Ru Nie. The latest research progress on spectral clustering. *Neural Computing and Applications*, 24:1477–1486, 2014. (cited on Page 45)

Ian T Jolliffe. *Principal component analysis for special types of data*. Springer, 2002. (cited on Page 36)

Michael I Jordan and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems: Proceedings of the 2001 Conference*, volume 14, page 849. MIT Press, 2002. (cited on Page 90 and 91)

Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009. (cited on Page 39)

Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956. (cited on Page 18)

James Kurose and Keith Ross. Computer networking: A top-down approach, global edition, 2017. (cited on Page 19)

Godfrey N Lance and William T Williams. Computer programs for hierarchical polythetic classification ("similarity analyses"). *The Computer Journal*, 9(1):60–64, 1966. (cited on Page 39)

Serge Lang. *Introduction to linear algebra*. Springer Science & Business Media, 2012. (cited on Page 28)

Xin-Ye Li and Li-jie Guo. Constructing affinity matrix in spectral clustering based on neighbor propagation. *Neurocomputing*, 97:125–130, 2012. (cited on Page 46)

Dijun Luo, Heng Huang, Chris Ding, and Feiping Nie. On the eigenvectors of p-laplacian. *Machine Learning*, 81:37–51, 2010. (cited on Page 46)

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967. (cited on Page 2)

Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009. (cited on Page 33)

N Gregory Mankiw. *Principles of economics*. Cengage Learning, 2014. (cited on Page 6)

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000. (cited on Page 51)

IC Mogotsi. Christopher d. manning, prabhakar raghavan, and hinrich schütze: Introduction to information retrieval: Cambridge university press, cambridge, england, 2008, 482 pp, isbn: 978-0-521-86571-5, 2010. (cited on Page 29)

Bojan Mohar. *Some applications of Laplace eigenvalues of graphs*. Springer, 1997. (cited on Page 21 and 25)

Bojan Mohar, Y Alavi, G Chartrand, and OR Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2(871-898):12, 1991. (cited on Page 21 and 25)

Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine learning*, 52:91–118, 2003. (cited on Page 35)

DS Moore, GP McCabe, and BA Craig. Introduction to the practice of statistics ninth edition, 2017. (cited on Page 6)

Mark Newman. *Networks*. Oxford university press, 2018. (cited on Page 19)

Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004. (cited on Page 1)

Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In Thomas Dietterich, Steve Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2002. (cited on Page 2 and 34)

Daiheng Ni. *Traffic flow theory: Characteristics, experimental methods, and numerical techniques*. Butterworth-Heinemann, 2015. (cited on Page 7)

JW Nilsson. Electric circuits author: James w. *Nilsson, Susan Riedel, Publisher: Prentice Hall Pages: 816 Published*, 20, 2014. (cited on Page 7)

Peter O Olukanmi and Bhekisipho Twala. K-means-sharp: modified centroid update for outlier-robust k-means clustering. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, pages 14–19. IEEE, 2017. (cited on Page 32)

Mercedes Pascual and Jennifer A Dunne. *Ecological networks: linking structure to dynamics in food webs*. Oxford University Press, 2006. (cited on Page 7)

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (cited on Page 2 and 69)

Larry L Peterson and Bruce S Davie. Computer networks: A systems approach, 2011. (cited on Page 7)

Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957. (cited on Page 18)

Nicola Rebagliati and Alessandro Verri. Spectral clustering with more than k eigenvectors. *Neurocomputing*, 74(9):1391–1401, 2011. (cited on Page 47)

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2010. (cited on Page 30)

Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007. (cited on Page 41 and 43)

Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987. (cited on Page 35 and 40)

Gerard Salton. Introduction to modern information retrieval. *McGraw-Hill*, 1983. (cited on Page 23)

Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2:169–194, 1998. (cited on Page 35)

Hinrich Schutze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge University Press, 2008. (cited on Page 39 and 41)

Andrew J Seary and William D Richards. *Spectral methods for analyzing and visualizing networks: an introduction*. na, 2003. (cited on Page 21)

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002. (cited on Page 51)

Robert Sedgewick and Kevin Wayne. Algorithms (4th edn). *Google Scholar Google Scholar Digital Library Digital Library*, 2011. (cited on Page 5, 10, 16, 17, and 18)

Shi. Multiclass spectral clustering. In *Proceedings ninth IEEE international conference on computer vision*, pages 313–319. IEEE, 2003. (cited on Page 2)

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000. (cited on Page 34 and 91)

MT Somashekara and D Manjunatha. Performance evaluation of spectral clustering algorithm using various clustering validity indices. *International Journal of Electronics Communication and Computer Engineering*, 5(6):1274–1276, 2014. (cited on Page xiii, xviii, 47, 62, 63, 80, 81, 82, and 84)

Gilbert Strang. *Linear algebra and its applications*. 2012. (cited on Page 19)

Alexander Strehl and Joydeep Ghosh. Cluster ensembles–a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3(3), 2003. (cited on Page 41 and 42)

Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022. (cited on Page 29)

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016. (cited on Page 45)

Robert Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953. (cited on Page 35)

Richard J Trudeau. *Introduction to graph theory*. Courier Corporation, 2013. (cited on Page 12)

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. (cited on Page 36)

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17: 395–416, 2007. (cited on Page 2, 21, 23, 24, 25, 26, 27, 34, 35, 36, 56, 59, 87, 89, and 90)

Ulrike Von Luxburg et al. Clustering stability: an overview. *Foundations and Trends® in Machine Learning*, 2(3):235–274, 2010. (cited on Page 35)

Yong Wang, Yuan Jiang, Yi Wu, and Zhi-Hua Zhou. Spectral clustering on multiple manifolds. *IEEE Transactions on Neural Networks*, 22(7):1149–1161, 2011. (cited on Page 46)

Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001. (cited on Page 6)

Johan C Winterwerp and Walther GM Van Kesteren. *Introduction to the physics of cohesive sediment dynamics in the marine environment*. Elsevier, 2004. (cited on Page 11)

Tao Xiang and Shaogang Gong. Spectral clustering with eigenvector selection. *Pattern Recognition*, 41(3):1012–1029, 2008. (cited on Page 46)

Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005. (cited on Page 39)

Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 29(1): 40–51, 2006. (cited on Page 21)

Yi Yang, Dong Xu, Feiping Nie, Shuicheng Yan, and Yueting Zhuang. Image clustering using local discriminant models and global integration. *IEEE Transactions on Image Processing*, 19(10):2761–2773, 2010. (cited on Page 46)

Zheng Yang, Yi Zhao, and Nasser M Nasrabadi. Scalable spectral clustering using random graph embedding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2169–2183, 2012. (cited on Page 33)

Xianchao Zhang and Quanzeng You. An improved spectral clustering algorithm based on random walk. *Frontiers of Computer Science in China*, 5:268–278, 2011. (cited on Page 46)

Xianchao Zhang, Jingwei Li, and Hong Yu. Local density adaptive similarity measurement for spectral clustering. *Pattern Recognition Letters*, 32(2):352–358, 2011. (cited on Page 45)

Feng Zhao, Licheng Jiao, Hanqiang Liu, Xinbo Gao, and Maoguo Gong. Spectral clustering with eigenvector selection based on entropy ranking. *Neurocomputing*, 73(10-12):1704–1717, 2010. (cited on Page 47)

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

Magdeburg, 23^{rd} August 2023