Università della Svizzera italiana

Faculty of Informatics

Bachelor Thesis

June 5, 2020

A study of Spectral Clustering techniques for machine learning applications

Lazar Najdenov

Abstract

Clustering is considered one of the most important unsupervised learning method used to find hidden patterns and group data with similar characteristics. In particular, spectral clustering has become in recent years one of the most used techniques for clustering the data in its different properties and shapes, due to its simplicity, computational efficiency and accuracy. Our study focuses on finding meaningful communities in graphs emerging from machine learning applications. To achieve this, we initially experiment with different similarity graphs configurations, constructed from the datasets, and select the one that best describes the datasets under question. Subsequently, we present different graph Laplacian, and choose the one that gives the optimal spectral clustering results based on a variety of evaluation metrics. Our results are based on the implementation available in our Github repository `https://github.com/LazarNajdenov/Bachelor-Thesis`.

Keywords: spectral clustering; graph Laplacian; eigen-decomposition;

Advisor:
Prof. Olaf Schenk
Co-advisor:
Dimosthenis Pasadakis

Approved by the advisor on Date:

Contents

1 Introduction

Clustering is considered the most important unsupervised learning method used to find hidden patterns or grouping data in exploratory data analysis [5] and data mining applications [24] . It has a large number of applications spread across various domains like recommendation engines [16], market segmentation [19], social network analysis [29], search result grouping [22], medical imaging [15], image segmentation [21]. The main purpose of clustering is to divide the dataset into natural groups (clusters), with data points in the same group being similar (intra-cluster similarity), whereas data points in different groups dissimilar (inter-cluster dissimilarity).

Traditional clustering methods, such as k-means algorithm, are easy to implement and give good quality results when the clusters are well separated. However they lack the ability to handle complex data structures, as they do not take into account densities of the graph, but rely only on topological features. Therefore they tend to fail on high-dimensional spaces.

To overcome the limit of traditional approaches, another clustering method based on algebraic graph theory has become increasingly popular due to its simple implementation and promising computational performance in many clustering problems: spectral clustering. In general it can be applied in particular cases where traditional methods fail, for example non-convex data sets [32].

In spectral clustering, the problem of data clustering transforms into the problem of graph partitioning. The first is constructing an undirected weighted graph, called adjacency matrix, with each point in the dataset being a vertex and the similarity value between any two points being the weight of the edge connecting the two vertices. Subsequently, form the associated Laplacian matrix and map each point to a lower-dimensional representation, based on one or more eigenvectors, depending on the number of clusters we need. Finally, these eigenvectors (spectral coordinates) are clustered using the k-means algorithm on this new representation.

Spectral clustering is considered a powerful method with strong theoretical background to cluster data. It has been applied successfully in various scientific domains, such as data analysis [5], speech separation[2], video indexing [20], multi-valued function recognition [10], or image processing [21].

This report is structured as follows: in Appendix A, we delineate the general development of the project and the main challenges; in section 2, we give the background knowledge about the spectral clustering method; section 3, is the evaluation part of the project, where we set up the techniques that will be used in the testing phase and comment on the results obtained; lastly, in section 4, we summarize the entire study, and talk about possible future extensions. We refer mostly to the von Luxburg [26] paper, the "Modern algorithms of cluster analysis [27]" book, and references therein for a detailed introduction to various aspects of spectral clustering.

For the remainder of this text our mathematical notation is the following: a vector, which in our case will be a data point in a (high-dimensional) state space (or feature space),is denoted as a lower case letter in bold, for example $\mathbf{s}$; a set, which in our case is a dataset of n data points, is denoted as an upper case letter, for example $S = \{\mathbf{s}_1, \dots, \mathbf{s}_n\}$; a matrix, is denoted as an upper case letter in bold, for example $\mathbf{W} \in \mathbb{R}^{m \times n}$, with w_{ij} being an element of the matrix $\mathbf{W}$ at row i and column j; a scalar multiplier is denoted as a lower case greek letter.

2 Spectral clustering background

2.1 Graph notation

Let $\Gamma = (V, E)$ be an undirected and connected graph. Here

$$V = \{\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_n}\}$$

is the set of nodes, and E is set of m edges

$$e_{ij} = \{\mathbf{v}_i, \mathbf{v}_j\}$$

connecting the vertices. If the graph is weighted, then it's adjacency matrix is denoted as $\mathbf{W} \in \mathbb{R}^{\mathrm{n \times n}}$

$$\mathbf{W} = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \dots & w_{nn} \end{bmatrix},$$

and its weights are in the form

$$\mathbf{W}(\mathbf{v}_i, \mathbf{v}_j) = \begin{cases} s(\mathbf{v}_i, \mathbf{v}_j) & \text{if } \{\mathbf{v}_i, \mathbf{v}_j\} \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $s(v_i, v_j)$ is a function that computes the weight of the edges connecting the vertices, and since Γ is undirected, it means that W is symmetric, therefore $w_{ij} = w_{ji}$; in the next sections we will discuss about different ways to compute the weights of a graph.
The sum of $W's$ i-th row, denoted d_i, that is defined as

$$d_i = \sum_{j=1}^{n} w_{ij} \tag{2}$$

is the degree of the node v_i, and the diagonal matrix $\mathbf{D} \in \mathbb{R}^{\mathrm{n \times n}}$

$$\mathbf{D} = \begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_n \end{bmatrix}$$

whose diagonal (i, i)-th element is the degree d_i, is said to be the degree matrix, which is defined as

$$\mathbf{D} = \operatorname{diag}(d_1, d_2, ..., d_n) \tag{3}$$

In order to compute the size of any subset $X \subset V$, we have two different metrics, very useful when dealing with balanced cut criteria: $|X|$ as the number of vertices belonging to X, called also cardinality, and the volume of X, which measures the size of X as the sum of the degrees of the vertices belonging to the set:

$$\operatorname{vol}(X) = \sum_{\mathbf{v}_i \in X} d_i. \tag{4}$$

2.2 Clustering objectives

The main objective of spectral clustering (SC) is to find a partition of the graph such that the edges between different groups have very low weights, and the edges within a group have high weights. This corresponds to a partitioning with high within-cluster similarity and low inter-cluster similarity. The resulting graph representing these results is called the similarity graph. The clusters should be balanced in the sense that the size/density of the clusters should not differ too much. Given a dataset with a set of points in a feature space (a space where each point cannot be represented spatially) and a similarity measure, the data can be transformed into a weighted, undirected graph Γ, where the vertices V represent the points in the feature space and the positive edge weights W encode the similarity of pairs of points. A clustering of the points is then equivalent to a partition of V into subsets $\{C_1, C_2.., C_k\}$, where every C_i is a cluster.

2.2.1 Balanced graph cut criteria

SC treats the problem of clustering as a graph partitioning problem and constructs an undirected weighted graph **W**, called also adjacency matrix, with each point in the dataset being a vertex and the similarity value between any two points being the weight of the edge connecting the two vertices. The main goal of this approach is to evenly distribute the number of nodes while minimizing weight of cut edges, that is the sum of weights of the removed edges, which basically means solving the minimum cut (min-cut) problem. Therefore the clustering problem is equivalent to choosing a partition $C_1, C_2, ..., C_k$ for a given number k of subsets of a graph which minimizes a specific objective function, in this case the min-cut approach

$$\min \text{cut}(C_1, C_2, ..., C_k) = \frac{1}{2} \sum_{i=1}^{k} \text{cut}(C_i, \overline{C_i}), \tag{5}$$

where 1/2 is employed for not counting the edges twice within the cut and

$$\text{cut}(C_i, \overline{C_i}) = \sum_{i \in C, j \in \overline{C}} w_{ij} \tag{6}$$

represents the weight of the edges connecting the nodes belonging to the cluster C_i, and the complement $\overline{C_i}$, that is the set of all the other clusters. In general,

$$C_i + \overline{C_i} = \text{the entire graph.} \tag{7}$$

This sort of cutting criteria in practice often does not provide a good partition. The issue is that in many cases, the answer of min-cut simply separates one individual vertex from the rest of the graph i.e. a really small sub-graph is cut away, therefore our partitioned graph is not balanced. To avoid this disadvantage, we utilize slightly different criteria, that enforce clusters with a reasonably large number of points. The general form of these criteria is the following:

$$F(C_1, C_2, ..., C_k) = \sum_{j=1}^{k} \frac{\text{cut}(C_j, \overline{C_j})}{f(C_j)}, \tag{8}$$

where $\text{cut}(C_j, \overline{C_j})$ is the sum of weights of the removed edges (cut edges) between cluster C_j and its complement $\overline{C_j}$, F is the kind of balanced cut we want to achieve, and f is a function applied to the C_i cluster to measure its size.
There are several criteria following this general rule, but we will consider in our study two of them: RatioCut (RCut) and the NormalizedCut (NCut).

The RCut introduces the size of clusters, which reduces the possibility of over-split, indeed the size of a subset C_i of a graph is measured by its number of vertices $|C_i|$:

$$\text{RCut}(C_1, C_2, ..., C_k) = \sum_{i=1}^{k} \frac{\text{cut}(C_i, \overline{C_i})}{|C_i|}, \tag{9}$$

and the main disadvantage of this method is that it only focuses on reducing the similarity between clusters. The NCut instead, takes into consideration both inter-cluster connections and intra-cluster connections, with the size of a subset C_i measured by the weights of its edges $vol(C_i)$:

$$\text{NCut}(C_1, C_2, ..., C_k) = \sum_{i=1}^{k} \frac{\text{cut}(C_i, \overline{C_i})}{\text{vol}(C_i)}. \tag{10}$$

Also we note that the minimum of the above functions are achieved if, when dealing with RCut all $|C_i|$ coincide, whereas with NCut is achieved if all vol(C_i) coincide [26]. Relaxing NCut ends up in normalized spectral clustering, whereas relaxing RCut leads to unnormalized spectral clustering. We will see these variants of spectral clustering in section 2.2.3. Another important point is that if clusters are well separated, both the cut criteria give very similar and accurate results, whereas if clusters are marginally separated, NCut provides more accurate results [32]. The optimal clustering results can be obtained by minimizing the objective function of the above graph cut methods. However, it is known that finding the global optimum of all these balanced graph cut criteria is an NP-hard optimization problem, due to its discrete nature. Hence, with the assistance of spectral methods, the original problem can be solved in polynomial time by relaxing the original discrete optimization problem to a real domain, and then using some heuristic approach to re-convert it to a discrete solution. Specifically, the utilization of the eigenvectors of the graph Laplacian matrix approximates the balanced graph cut, which will be covered in the next subsections.

2.2.2 Similarity and connectivity

In order to apply correctly spectral clustering we firstly have to preprocess our data that is, let X be a set of n data items, the relationships between the pairs of data items are represented by a weighted undirected graph $\Gamma = (V, E, W)$, where $V = \{\mathbf{v}_1, ..., \mathbf{v}_n\}$ is the set of nodes representing the data items, E is the set of edges and $\mathbf{W}$ is a generalized adjacency matrix. Hence our main goal in the preprocessing part is to create the adjacency matrix $\mathbf{W}$ satisfying (1). It has to be noted that, if in the adjacency matrix the number of connected components is higher than the number of clusters we need, then spectral clustering will return the connected components as clusters.

Thus, the first step for creating $\mathbf{W}$ is to model the local neighborhood relationships between the data points, meaning that we have to create the symmetric connectivity matrix $\mathbf{G}$ where we connect data points based on a particular method, which is usually considered as an unweighted graph: the first is the ϵ-neighborhood graph where we connect all points whose pairwise distances are smaller than ϵ, called also Euclidean distance defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{y}_i)^2} = ||\mathbf{x} - \mathbf{y}||, \tag{11}$$

where $\mathbf{x}$ and $\mathbf{y}$ being two generic vectors.

The choice of ϵ is really important, as we have to choose it such that the resulting graph is safely connected. If we have data on different scales, that is the distances between data points are different in different regions of the space, it might happen that some points will not have any connection. Therefore, the resulting graph

will not be connected. A particular technique used for choosing ϵ is to take the value of the longest edge of the minimal spanning tree generated from a fully connected graph.
Another technique that defines the way nodes are related is the K-nearest neighbor(kNN) graph where we connect vertex $\mathbf{v}_i$ and $\mathbf{v}_j$ with an undirected edge if $\mathbf{v_i}$ is among the k-nearest neighbors of $\mathbf{v}_j$ or if $\mathbf{v}_j$ is among the k-nearest neighbors of $\mathbf{v}_i$, where the term nearest means the k vertices whose Euclidean distance is the smallest from a particular vertex. In the resulting graph $\mathbf{G}$, each vertex is connected to at least k vertices. Here, the choice of the k neighbors in order to have a safely connected graph depends on its size. For small graphs, trial and error attempts, until it is safely connected; for big graphs, k has to be $k \approx log(n)$, with n being the number of points of the dataset. See [26] for more details regarding the selection of k. In cases where the ϵ-neighborhood was not able, the kNN, on the other hand, can connect data points that are in a low-density with points in a high density, and also can break into several disconnected components if there are high density regions which are reasonably far away from each other.
Lastly, the Mutual k-nearest neighbor (mkNN) graph is a variant of the kNN, where vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ are connected if both $\mathbf{v}_i$ is among the k-nearest neighbors of $\mathbf{v}_j$ and $\mathbf{v}_j$ is among the k-nearest neighbors of $\mathbf{v}_i$. The mkNN works better when there are clusters of different densities, indeed it tends to connect points within regions of constant density, but does not connect regions of different densities with each other. So the mkNN graph can be considered as an intermediate version between the $\epsilon-$neighborhood graph and the kNN [26]. We can infer that the mkNN graph, for the same parameter k, connects less nodes from the kNN. This means that k for the same graph has to be larger for the mkNN than the standard kNN.
The next step is defining a similarity function on the data, which guarantees that the local neighborhoods promoted by this similarity function are meaningful. Hence we create the similarity matrix $\mathbf{S}$, where we compute the degrees of similarity (weights) between every pair of vertices, and study four types of functions with their respective benefits and drawbacks.
The first we consider is the Gaussian similarity function where data points live in the Euclidean Space $\mathbb{R}^{\mathrm{d}}$, which is the most suitable and the similarity between every pair of data point. It is computed as:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right). \tag{12}$$

exp(), with $||\mathbf{x}_i - \mathbf{x}_j||$ being the Euclidean distance between the two data points, and σ controlling the size of the neighborhood as the ϵ value in the ϵ-neighborhood graph. Thus with a fixed parameter σ, the similarity between two items is only a function of their Euclidean distance and not adaptive to their surroundings, therefore this procedure is inefficient in proper reflecting the distribution of complex data, like multi-scale data set [32].
The local scale similarity function is based on handling multi-scale data where the Gaussian similarity mostly fails. Here, instead of selecting a single scaling parameter σ, we compute a local scaling parameter σ_i for each data point $\mathbf{x}_i$, indeed the distance from $\mathbf{x}_i$ to $\mathbf{x}_j$ as seen by $\mathbf{x}_i$ is $\frac{||\mathbf{x}_i - \mathbf{x}_j||}{\sigma_i}$ whereas from $\mathbf{x}_j$ is $\frac{||\mathbf{x}_j - \mathbf{x}_i||}{\sigma_j}$. The local scale similarity function between every pair of data point is defined as:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma_i \cdot \sigma_j}\right). \tag{13}$$

Here, σ_i stands for the distance between i-th data point and its k-th nearest neighbor:

$$\sigma_i = ||\mathbf{x}_i - \mathbf{x}_k||,$$

with $\mathbf{x}_k$ being the k'th neighbor of $\mathbf{x}_i$. However, this local distance aware similarity cannot reveal the properties of real clusters and fails on many real world data sets [32]. More details are given in [31].

The max similarity function is computed between every pair of data points as:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \max\{\mathbf{x}_i(j), \mathbf{x}_j(i)\} \quad \text{where} \;\; \mathbf{x}_i(j) = \exp\left(-\frac{4 * ||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma_i^2}\right); \tag{14}$$

such that the resulting similarity value is given by one of the two points whose euclidean distance with the k-th neighbor, given by σ_i, is bigger. More details are given in the [3].

The last similarity function we consider is the the Common-near-neighbor(CNN), which reflects the local density between two data points, instead of the other similarities mentioned above. The main idea is that if two points are distributed in the same cluster, they are in the same region which has a relatively high density, that is, two points fall into the same cluster because there are many points between them, and the similarity measure between two data points is defined as follows:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2 \cdot (CNN(\mathbf{x}_i, \mathbf{x}_j) + 1)}\right), \tag{15}$$

where CNN$(\mathbf{x}_i, \mathbf{x}_j)$ is the function that counts the number of points in the joint region of the ϵ-neighborhoods around points $\mathbf{x}_i$ and $\mathbf{x}_j$, with ϵ being the radius of the sphere region denoting the ϵ-neighborhoods of a given point, and the $+1$ as a lower bound whenever the CNN's result is equal to zero. Thus, with the scaled σ parameter, the new similarity is adaptive to local density. It has an effect of amplifying intra-cluster similarity, meaning that the bigger the number of CNNs between the two points, the bigger will be the similarity of the two, making theoretically the adjacency matrix much more block diagonal in order to identify correctly the clusters. The main issue with this approach is the selection of ϵ, indeed we mostly choose this value based on some heuristics. More details about this similarity can be found in [32].

When the connectivity and the similarity matrices are computed for a given dataset the corresponding adjacency matrix of the graph reads:

$$\mathbf{W} = \mathbf{S} \odot \mathbf{G}, \tag{16}$$

where the $\odot$ operator performs element-wise multiplication, resulting in a sparse matrix, which contains the similarity measures only where $\mathbf{G}$ contains elements, therefore only when there is an edge connecting two nodes satisfying (1).

In order to show the different behavior of the weighted edge distributions from the adjacency matrices computed, the we consider the Outlier dataset [14]:

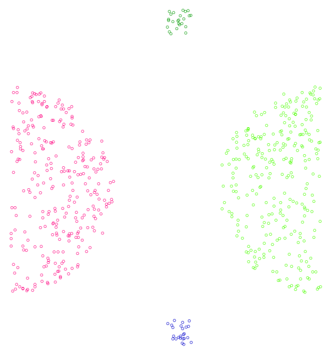
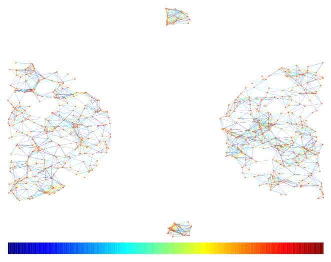
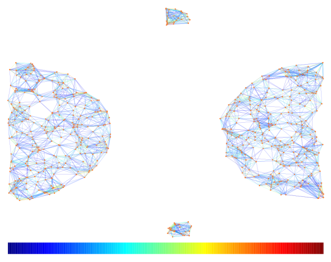


Figure 1. Scatter plot of Outliers dataset.

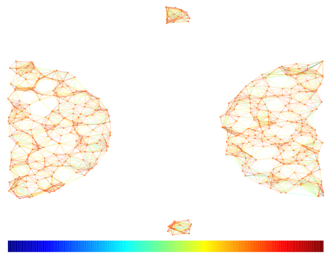
Notice that in this case we do not consider the fact that $\mathbf{G}$ is connected, as we want to emphasize on the way the different similarity functions behave on the dataset.



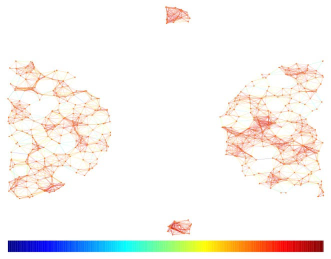
(a) Gaussian similarity with epsilon-neighborhood.



(b) Max similarity with kNN connectivity.



(c) Local scaled similarity with kNN connectivity.



(d) CNN with epsilon-neighborhood.

Figure 2. Figure 2a represents the weighted edges using the Gaussian similarity function, Figure 2b the max similarity, Figure 2c the local scale similarity, and Figure 2d the CNN similarity. An hotter edge color means that the similarity between the two points is high, whereas a cooler edge color represents a lower one.

2.2.3 The graph Laplacian

The choice of graph Laplacian matrix plays a critical role in spectral clustering. Through the spectrum of the Laplacian the spectral clustering process is able to split graphs generated from a dataset. In the study we cover three types of Laplacian: unnormalized Laplacian, and the normalized Laplacian, which itself has two variants called symmetric and Random-walk Laplacian.
By assuming that $\Gamma = (V, E, W)$ is an undirected, weighted graph with adjacency matrix $\mathbf{W}$, the unnormalized Laplacian is defined as follows:

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{17}$$

where $\mathbf{D}$ is the degree matrix and $\mathbf{W}$ the adjacency matrix, and it satisfies the following properties:

1. $\mathbf{L}$ is symmetric, since follows directly from the symmetry of $\mathbf{W}$ and $\mathbf{D}$, meaning that its eigenvalues are real and its eigenvectors are real and orthogonal.

2. The smallest eigenvalue of $\mathbf{L}$ is 0, the corresponding eigenvector is the constant one vector(the vector of all ones) $\mathbf{e} = [1, ..., 1]^T$ for which it is true

$$\mathbf{L} \cdot \mathbf{v} = \lambda \cdot \mathbf{v} \rightarrow \mathbf{L} \cdot \mathbf{e} = 0 \cdot \mathbf{e} = 0$$

3. $\mathbf{L}$ is positive semi-definite, since its eigenvalues, that are also called the spectrum of $\mathbf{L}$, are non-negative:

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

indeed for every vector $\mathbf{x} \in \mathbb{R}^{\mathrm{d}}$ the function that maps $f : V \rightarrow \mathbb{R}$, is given by the following quadratic form:

$$\mathbf{x^T L x} = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (\mathbf{x}_i - \mathbf{x}_j)^2 \geq 0$$

where $\mathbf{x^T L x} = 0$ only when $\mathbf{x}$ is the constant vector one with eigenvalue 0.

4. The multiplicity k of the eigenvalue 0 of $\mathbf{L}$ is equal to the number of connected components in our graph Γ.

A deeper understanding of these properties can be found in [26]. Notice that when Γ is connected, i.e. we have one connected component, then $\lambda_2 \neq 0$, which is called algebraic connectivity, whose magnitude indicates how tightly connected are the nodes in the graph, thus effectively measuring the connectivity of the graph.
The normalized symmetric Laplacian instead is defined as follows:

$$\mathbf{L_{sym}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}, \tag{18}$$

where $\mathbf{L}$ is the unnormalized Laplacian, $\mathbf{I}$ is the identity matrix, $\mathbf{W}$ is adjacency matrix and $\mathbf{D}$ is the degree matrix which is diagonal, therefore its reciprocal square root $\mathbf{D}^{-\frac{1}{2}}$ is just the diagonal matrix whose diagonal entries are the reciprocals of the positive square roots of the diagonal entries of $\mathbf{D}$.
The other normalized Laplacian, the Random-walk Laplacian, is instead defined as follows:

$$\mathbf{L_{rw}} = \mathbf{I} - \mathbf{P} \quad \text{where} \quad \mathbf{P} = \mathbf{D}^{-\beta}\mathbf{W} \tag{19}$$

where $\mathbf{P}$ is the transition probability matrix, which re-weights the edges of Γ such that the degree of each node is equal to 1. The power factor β is a varying parameter whose value affects significantly on the clustering results [12]. We will see afterwards in 3.2 how much the different values of this factor affects the spectral clustering results for different datasets. Notice also that from the non-symmetric diffusion matrix $\mathbf{P}$, that is row-stochastic $\sum_{j=1}^{m} p_{ij} = 1$ whose entries p_{ij} can be viewed as the probability of moving from node $\mathbf{v_i}$ to node $\mathbf{v_j}$, the resulting Random-walk Laplacian will also be non-symmetric. Both the normalized Laplacian satisfies the following properties:

1. Both matrices $\mathbf{L_{sym}}$ and $\mathbf{L_{rw}}$ are positive semi-definite, indeed their eigenvalues(the spectrum) are non-negative:

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

   in fact for every vector $\mathbf{x} \in \mathbb{R}^{\mathrm{d}}$ the function that maps $f : V \to \mathbb{R}$, is given by the following quadratic form:

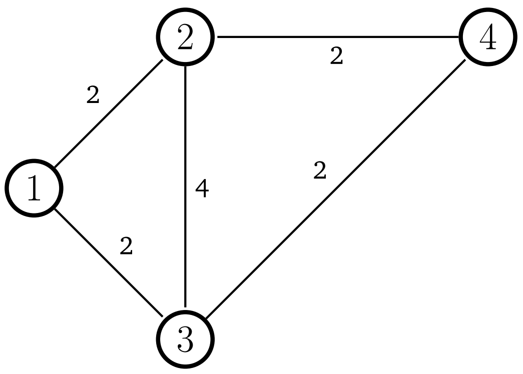
$$\mathbf{x^T L_{sym} x} \geq 0$$

2. $\mathbf{L_{sym}}$ is symmetric, whereas $\mathbf{L_{rw}}$ is not.

3. The smallest eigenvalue of both is 0, and the corresponding eigenvector of eigenvalue 0 for $\mathbf{L_{sym}}$ is $\mathbf{D}^{1/2}\mathbb{1}$ with $\mathbb{1}$ being the constant vector one, whereas for $\mathbf{L_{rw}}$ is just the constant vector one.

4. As in the unnormalized version, the multiplicity k of the eigenvalue 0 of the normalized graph Laplacian is related to the number of connected components.

Once more we refer to [26] for a detailed analysis of these properties.
Let us consider an undirected, weighted graph $\Gamma = (V, E, W)$ and compute the different Laplacian derived from it:



The relative adjacency **W** and degree **D** matrices are the following:

$$\mathbf{W} = \begin{bmatrix} 0 & 2 & 2 & 0 \\ 2 & 0 & 4 & 2 \\ 2 & 4 & 0 & 2 \\ 0 & 2 & 2 & 0 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Hence the unnormalized Laplacian matrix is given by:

$$\mathbf{L} = \mathbf{D} - \mathbf{W} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} - \begin{bmatrix} 0 & 2 & 2 & 0 \\ 2 & 0 & 4 & 2 \\ 2 & 4 & 0 & 2 \\ 0 & 2 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & -2 & -2 & 0 \\ -2 & 8 & -4 & -2 \\ -2 & -4 & 8 & -2 \\ 0 & -2 & -2 & 4 \end{bmatrix}$$

Then the normalized symmetric Laplacian matrix is of the form:

$$\mathbf{L_{sym}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{2\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} 4 & -2 & -2 & 0 \\ -2 & 8 & -4 & -2 \\ -2 & -4 & 8 & -2 \\ 0 & -2 & -2 & 4 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{2\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 0 \\ -\frac{1}{2\sqrt{2}} & 1 & -\frac{1}{2} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{2\sqrt{2}} & -\frac{1}{2} & 1 & -\frac{1}{2\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 1 \end{bmatrix}$$

And the normalized Random-walk Laplacian is:

$$\mathbf{L_{rw}} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W} = \mathbf{I} - \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{8} & 0 & 0 \\ 0 & 0 & \frac{1}{8} & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \cdot \begin{bmatrix} 0 & 2 & 2 & 0 \\ 2 & 0 & 4 & 2 \\ 2 & 4 & 0 & 2 \\ 0 & 2 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{4} & 1 & -\frac{1}{2} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{2} & 1 & -\frac{1}{4} \\ 0 & -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}$$

By recalling 2.2.1, the min-cut criteria does not always give good results, therefore we want our solutions to be balanced, and this can be achieved by using RCut or NCut. But adding balancing conditions makes our problem NP-hard, therefore the spectral clustering process makes use of the Laplacian's eigenvectors in order to approximate the graph balanced cut, that it is relaxing the original discrete optimization problem and solve it in polynomial time. Based on the Fiedler vector, that is the eigenvector corresponding to the second eigenvalue λ_2, we can divide our graph in two parts, or based on multiple k main eigenvectors, we can divide

the graph into k parts. This is basically the process of solving the eigenvalue problem for a given matrix, that is in our case the Laplacian.
We demonstrate hereby how the approximation of the RCut between two clusters work, which is the simplest one; the main goal is to solve the following minimization problem:

$$\min_{C\subset V} RCut(C,\overline{C}) = \min_{C,\overline{C}} \frac{1}{2}\left(\frac{\text{cut}(C,\overline{C})}{|C|} + \frac{\text{cut}(\overline{C},C)}{|\overline{C}|}\right) \tag{20}$$

where we consider only two clusters, C and its complement $\overline{C}$. We are looking for a vector $\mathbf{x}$ such that minimizing the $RCut(C,\overline{C})$ is the same as minimizing $\mathbf{x^TLx}$, that is the unnormalized graph Laplacian, subject to some constraints. Hence given the subset $C \subset V$, where V is the set of the nodes of our initial graph, we define the vector $\mathbf{x} = (x_1, ..., x_n)^T \in \mathbb{R}^n$ with the following entries:

$$x_i = \begin{cases} \sqrt{\frac{|\overline{C}|}{|C|}} \;\; if \;\; v_i \in C \\ -\sqrt{\frac{|C|}{|\overline{C}|}} \;\; if \;\; v_i \in \overline{C} \end{cases} \tag{21}$$

We can now express our objective function in terms of the graph Laplacian:

$$\mathbf{x^TLx} = \mathbf{x^TDx} - \mathbf{x^TWx} = \sum_{i=1}^{n} d_i x_i^2 - \sum_{i,j=1}^{n} x_i x_j w_{ij} = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 =$$

$$= \frac{1}{2}\sum_{i\in C, j\in\overline{C}} w_{ij}\left(\sqrt{\frac{|\overline{C}|}{|C|}} + \sqrt{\frac{|C|}{|\overline{C}|}}\right)^2 + \frac{1}{2}\sum_{i\in\overline{C}, j\in C} w_{ij}\left(-\sqrt{\frac{|\overline{C}|}{|C|}} - \sqrt{\frac{|C|}{|\overline{C}|}}\right)^2 =$$

$$= \left(\frac{|\overline{C}|}{|C|} + \frac{|C|}{|\overline{C}|} + 2\right)\cdot\left(\frac{1}{2}\sum_{i\in C, j\in\overline{C}} w_{ij} + \frac{1}{2}\sum_{i\in\overline{C}, j\in C} w_{ij}\right) =$$

$$= \left(\frac{|\overline{C}| + |C|}{|C|} + \frac{|C| + |\overline{C}|}{|\overline{C}|}\right)\cdot \text{cut}(C,\overline{C}) = \text{cut}(C,\overline{C})\cdot\left(\frac{1}{|C|} + \frac{1}{|\overline{C}|}\right)\cdot(|C| + |\overline{C}|) =$$

$$|V|\cdot \text{RCut}(C,\overline{C}) = \mathbf{x^TLx} \tag{22}$$

Thus, minimizing the RCut is the same as minimizing $\mathbf{x^TLx}$ subject to the following constraints:

1. $\sum_{i=1}^{n} x_i = 0 \rightarrow x \perp \mathbb{1}$, i.e. x is orthogonal to the constant vector one
2. $||\mathbf{x}||^2 = n = |V|$, that is x measures the cardinality of the graph.
3. x_i takes discrete values, as stated in (21).

This is still a discrete optimization problem, due to the entries x_i of the solution vector $\mathbf{x}$ which is therefore still NP-hard. Hence a solution to obtain a relaxed optimization problem is achieved by allowing x_i to attain arbitrary values in $\mathbb{R}$, therefore our problem assumes the following form:

$$\min_{\mathbf{x}\in\mathbb{R}} \mathbf{x^TLx} \quad \text{subject to } \mathbf{x} \perp \mathbb{1}, ||\mathbf{x}||^2 = n \tag{23}$$

The Rayleigh-Ritz (RR) theorem indicates that the solution of this minimization problem is given by the Fiedler eigenvector, the eigenvector corresponding to the 2nd smallest eigenvalue of **L**, since the smallest eigenvalue of **L**, which is positive-definite, is 0 corresponding to the constant vector of ones. But we have to transform approximate solution back to its discrete setting, in order to obtain a partition of our graph. Following an heuristic approach, the spectral clustering algorithm makes the coordinates x_i as points in $\mathbb{R}$, and cluster them into two groups A and $\overline{A}$ with a flat clustering algorithm which will be k-means in our study, due to its simplicity and efficiency. The clustering results are then associated with the underlying data points:

$$\begin{cases} v_i \in C \ \ if \ \ x_i \in A \\ v_i \in \overline{C} \ \ if \ \ x_i \in \overline{A} \end{cases}$$

This shows that we can approximate RCut by using the second eigenvector of L when dealing with two clusters, which indeed translates into the fact that the min-cut problem can be solved by an eigen-decomposition of the Laplacian matrix in polynomial time.
Following this proof it is possible to use similar arguments to show that solving $minRCut$ for $k > 2$ corresponds to finding the first k eigenvectors of L, and also for the $minNCut$, in which case we deal with a normalized Laplacian.

2.2.4 **Final clustering of the eigenvectors**

Classical flat clustering algorithms, such as k-means, cluster the dataset based only on the euclidean distances between data points (on their topological features) and do not take into account the different densities of each cluster. Hence such methods tend to fail on much more complex datasets. On the other hand, spectral clustering overcomes these difficulties by mapping each point to a lower-dimensional representation based on one or more eigenvectors of the derived Laplacian, depending on the number of clusters it needs. Then it partitions the dataset by applying the k-means algorithm on this new lower-dimensional representation, which is faster to compute, leading into an approximate solution to the layout problem and giving much more visual information, since the clusters are easier to detect.
Let us consider the finite element mesh `3elt`, originating from aeronautics applications. We visualize its nodal coordinates and its eigenvector coordinates where we locate the vertex i at position $\mathbf{x_i} = (\mathbf{v_2}(i), \mathbf{v_3}(i))$, with $\mathbf{v_2}, \mathbf{v_3}$ being the eigenvectors corresponding to the 2nd and 3rd smallest eigenvalues of the Laplacian:

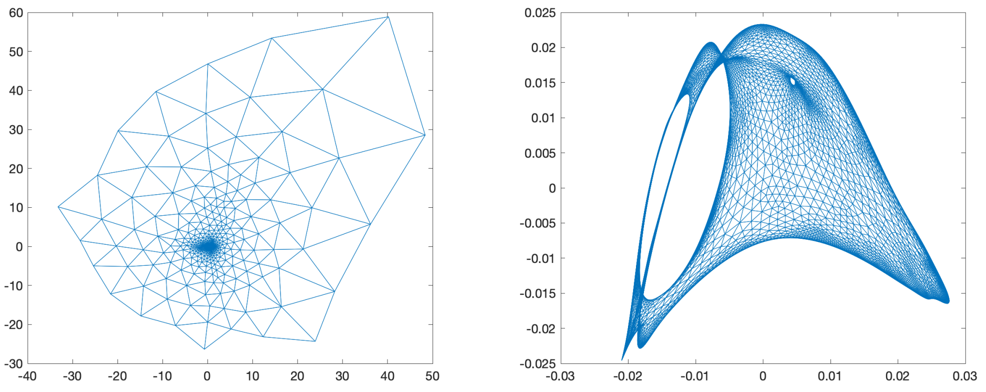


Figure 3. The figure on the left is the nodal coordinates representation, whereas the one to the right, the eigenvector coordinates representation.

One can see, if we apply directly k-means on the nodal coordinates, it is much more difficult to obtain good results, due to regions of varying density. Whereas, applying on the new space represented by the eigenvectors, the clusters are more separated. Therefore k-means will perform better since the cluster-properties in

the data are enhanced.
In the remainder of this subsection we provide the full spectral clustering pseudocode depending on the different Laplacian, based mostly on [26]; here, as input, we consider our dataset to have n data points with d features, and k being the number of clusters we want to separate:

Algorithm 1 Unnormalized spectral clustering

```
Input: Data points $\mathbf{Pts} \in \mathbb{R}^{n \times d}$ and $k$ as the number of cluster we want
Output: Clusters;
1: Construct the connectivity matrix $\mathbf{G} \in \mathbb{R}^{n \times n}$ and the similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$;
2: Compute the adjacency sparse matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ given by $\mathbf{W} = \mathbf{S} \odot \mathbf{G}$;
3: Compute the unnormalized Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$;
4: Compute the first $k$ eigenvectors $\mathbf{x_1}, ..., \mathbf{x_k}$, corresponding to the k smallest eigenvalues of $\mathbf{L}$, by solving the eigenvalue problem $\mathbf{Lx} = \lambda \mathbf{x}$;
5: Let $\mathbf{X} \in \mathbb{R}^{n \times k}$ be the matrix having as column vectors $\mathbf{x_1}, ..., \mathbf{x_k}$
       $$\mathbf{X} = \begin{bmatrix} x_{11} & \dots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nk} \end{bmatrix}$$
6: For $i = 1, ..., n$ let $\mathbf{y_i} \in \mathbb{R}^k$ be the vector corresponding to the i-th row of $\mathbf{X}$;
7: Cluster the $n$ rows $\mathbf{y_i}$ with the flat k-means algorithm into Clusters $C_1, ..., C_k$;
```

Algorithm 2 Normalized spectral clustering with symmetric Laplacian

```
Input: Data points $\mathbf{Pts} \in \mathbb{R}^{n \times d}$ and $k$ as the number of cluster we want
Output: Clusters;
1: Construct the connectivity matrix $\mathbf{G} \in \mathbb{R}^{n \times n}$ and the similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$;
2: Compute the adjacency sparse matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ given by $\mathbf{W} = \mathbf{S} \odot \mathbf{G}$;
3: Compute the unnormalized Laplacian $\mathbf{L}$ and therefore the relative normalized symmetric Laplacian $\mathbf{L_{sym}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$;
4: Compute the first $k$ eigenvectors $\mathbf{x_1}, ..., \mathbf{x_k}$, corresponding to the k smallest eigenvalues of $\mathbf{L_{sym}}$, by solving the eigenvalue problem $\mathbf{L_{sym}x} = \lambda \mathbf{x}$;
5: Let $\mathbf{X} \in \mathbb{R}^{n \times k}$ be the matrix having as column vectors $\mathbf{x_1}, ..., \mathbf{x_k}$;
6: For $i = 1, ..., n$, cluster the $n$ rows $\mathbf{y_i} \in \mathbb{R}^k$, that are the vectors corresponding to the i-th row of $\mathbf{X}$, with the flat k-means algorithm into Clusters $C_1, ..., C_k$;
```

Algorithm 3 Normalized spectral clustering with symmetric Random-walk Laplacian

```
Input: Data points $\mathbf{Pts} \in \mathbb{R}^{n \times d}$ and $k$ as the number of cluster we want
Output: Clusters;
1: Construct the connectivity matrix $\mathbf{G} \in \mathbb{R}^{n \times n}$ and the similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$;
2: Compute the adjacency sparse matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ given by $\mathbf{W} = \mathbf{S} \odot \mathbf{G}$;
3: Compute the unnormalized Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$ and therefore the relative normalized Random-walk
   Laplacian $\mathbf{L_{rw}} = \mathbf{I} - \mathbf{P}$ where $\mathbf{P} = \mathbf{D}^{-\beta}\mathbf{W}$ is the transition probability matrix(diffusion matrix);
4: Compute the first $k$ eigenvectors $\mathbf{x_1}, ..., \mathbf{x_k}$, corresponding to the k smallest eigenvalues of $\mathbf{L_{rw}}$, by solving
   the eigenvalue problem $\mathbf{L_{rw}}\mathbf{x} = \lambda\mathbf{x}$;
5: Let $\mathbf{X} \in \mathbb{R}^{n \times k}$ be the matrix having as column vectors $\mathbf{x_1}, ..., \mathbf{x_k}$;
6: For $i = 1, ..., n$, cluster the $n$ rows $\mathbf{y_i} \in \mathbb{R}^{k}$, that are the vectors corresponding to the i-th row of $\mathbf{X}$, with the
   flat k-means algorithm into Clusters $C_1, ..., C_k$;
```

3 Optimal spectral clustering for machine learning applications

We now consider the application of an optimal spectral clustering configuration in machine learning tasks. One of the critical elements for choosing the best spectral clustering setup is the selection of appropriate metrics in order to evaluate the quality of the results.
Initially, we consider some artificial datasets, with known labels, and evaluate on them the effectiveness of different similarity graph configurations, as outlined in 2.2.2. Details regarding the parameter selection for these configurations are offered in Section 3.1.1. The metrics we use to evaluate the quality of the clustering results is the clustering accuracy (ACC), and the ratiocut (RCut, (9)) between the different results. The accuracy is defined following [17]:

$$\mathrm{ACC} = \frac{\sum_{i=1}^{n} \delta(l_i, \mathrm{map}(c_i))}{n}, \tag{24}$$

where n is the number of points of the dataset, l_i is the true class label and c_i the inferred cluster label of x_i, which is the clustering result. The delta function $\delta(x, y)$ is defined as: $\delta(x, y) = 1$, is $x = y$, otherwise $\delta(x, y) = 0$. The mapping function $\mathrm{map}(\cdot)$ assigns the true label on the inferred label with the most frequent hits within the cluster. Therefore, $\mathrm{ACC} \in [0, 1]$ is a metric indicating the predicting accuracy of a clustering algorithm, with values $\mathrm{ACC} \approx 1$ corresponding to a perfect clustering result.
We seek the configuration that gives the highest accuracy, and the best minimization of Rcut as defined in (20). Subsequently, we choose the best graph Laplacian configuration, which will be part of the optimal spectral clustering. In particular, during this phase we evaluate the results on two more metrics along with ACC and RCut. The normalized cut (Ncut, see (10)) criteria, and the modularity (MOD), whose maximization is one of the most widely used methods for community or cluster detection [12], and it is computed following Newman-Girvan's proposal [18]:

$$\mathrm{MOD} = \sum_i \left(d_{ii} - \left(\sum_j d_{ij} \right)^2 \right), \tag{25}$$

which takes a symmetric matrix of clusters and each element d_{ij} represents the degree of the edges that link nodes between clusters i and j; each d_{ii} represents the degree of the edges linking nodes within cluster i.

3.1 Graph creation

During the first phase of testing, we evaluate the datasets on the ACC and RCut by using three different graph constructions, i.e creating (16):

1. *Eps-Gauss* configuration; as **G** we use the epsilon-neighborhood connectivity function introduced in section 2.2.2, and as **S** we use the Gaussian similarity function defined in (12).

2. *kNN-Max* configuration; as **G** we use the kNN connectivity function introduced in section 2.2.2, and as **S** we use the Max similarity function defined in (14).

3. *Eps-Cnn* configuration; as **G** we use the epsilon-neighborhood connectivity function, and as **S** we use the CNN similarity function defined in (15).

3.1.1 Parameter selection

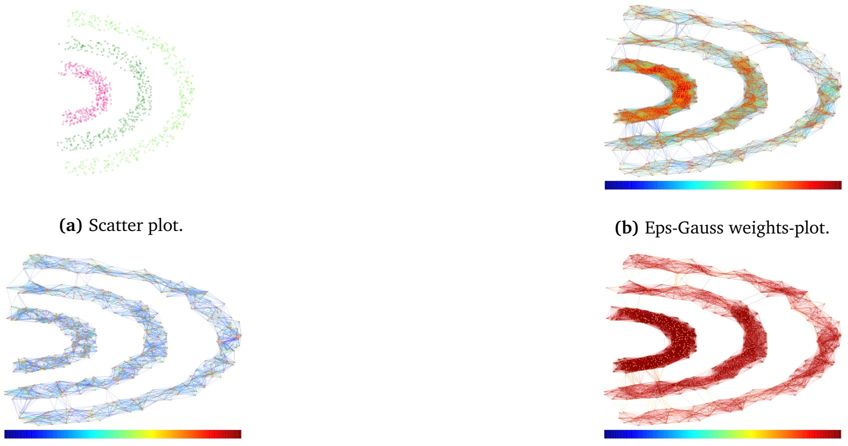
In order to ensure the connectivity of the graph constructed from the datasets, we decide to take the values of ϵ for the Eps-Gauss and Eps-Cnn configurations following heuristics used in [26]:

$$\epsilon = \log(n). \tag{26}$$

The choice of k, that is the number of neighbors in the kNN graph similarity, is based on trial and error attempts by increasing the k value until connectivity is guaranteed [27]. The effective choice of these parameters remains an open research problem [26]. In what follows, we measure the performance and accuracy of the above mentioned graph construction techniques in a series of artificial datasets, introduced in [8].

3.1.2 Generated three-kernels dataset

The first artificial dataset we test is the generated three-kernels dataset in Figure 4, a modified and slightly complex version of the two-kernels in [14]. The three-kernels dataset and the weight plots are presented in Figure 4:


(a) Scatter plot.

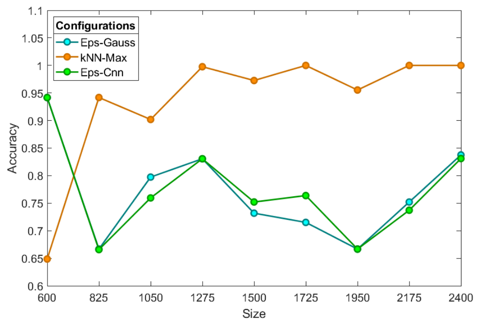
(b) Eps-Gauss weights-plot.

(c) kNN-Max weights-plot.

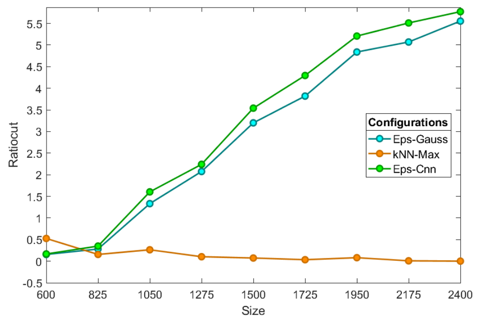
(d) Eps-Cnn weights-plot.

Figure 4. Figure 4a shows the shape of the dataset, whereas Figure 4b, Figure 4c, and Figure 4d, the edge weights of the different adjacency matrices using the three configurations in 3.1. The edge weights are mapped to a colormap, represented by the bar plot below each figure, with values ranging from 0 (blue color) to 1 (red color).

It is worth noting that at Figure 4b and Figure 4d, whose connectivity function is the same, through the use of the CNN similarity measure (15) the weights of the edges in the same density kernel are increased from the first one which uses the Gaussian similarity (12).
In order to evaluate the optimal configuration for the given dataset in an efficient way, we test the three-kernels by progressively increasing the problem size for the three different strategies. Nine runs on the dataset have been made, with the number of points ranging from 600 to 2400 with a step size of 225. At every iteration we compute the adjacency matrices for the three configurations. Moreover, by running a total of 10 k-means runs on the K smallest eigenvectors of the unnormalized Laplacian we compare the clustering results on the average of both the ACC (24) and the Rcut (20) for each of them, shown respectively in Figure 5a and Figure 5b.
In order to guarantee the connectivity, we set epsilon following (26) when dealing with the ϵ-neighborhood, whereas the number of neighbors k for the kNN case is set to $k = 20$; we take this choice based on the common k value that guarantees connectivity for the smallest and biggest problem size in our evaluation. Our results are summarized in Figure 5:



(a) ACC for an increasing problem size.

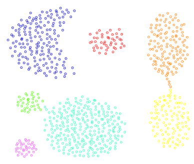


(b) RCut for an increasing problem size.

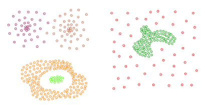
Figure 5. Results in Figure 5a and Figure 5b, show that the kNN-Max configuration performs better. On the average, the ACC is consistently greater than the other two configurations, and the value of the Rcut is better minimized following (20).

3.1.3 **Artificial datasets**

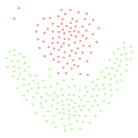
We proceed by comparing the performance of ACC and RCut on a series of publicly available artificial datasets, commonly used in the machine learning community (see Figure 6). We follow the same evaluation procedure



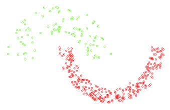
(a) Aggregation [1]



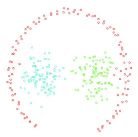
(b) Compound [30]



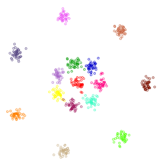
(c) Flame [9]



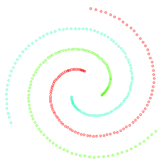
(d) Jain [13]



(e) Pathbased [4]



(f) R15 [6]



(g) Spirals [6]

Figure 6. Scatter plots of the artificial datasets [8] used in the numerical experiment.

applied in section 3.1.2. Since in this phase each dataset is already generated, we can not change the problem size. To guarantee the connectivity, we set once more epsilon following (26) when dealing with the ϵ-neighborhood. However, the heuristic choice of epsilon did not generate meaningful results, therefore it involved a process of trial and errors for obtaining the optimal epsilon value.
The k for the kNN case also involves trial-and-error attempts for the different problems. In particular, we are interested in finding the minimum k in kNN, in order to get a sparse connected graph that permits quicker and more accurate computations. These procedure results in k value set to $k = 8$ for the Spiral dataset, $k = 20$ for the Compound, Flame, Jain, Pathbased problems, and $k = 40$ for the Aggregation and R15 ones. The results are presented in Table 1 and Table 2. The first showing the ACC for each problem with the three

configurations, and the RCut for the second:

Table 1. For each row, on the left hand side of the central column, we have the problem tested, with its number of nodes and clusters. On the right hand side we have the ACC (24) computed on the different graph configurations introduced in section 3.1. Each measure gives an ACC value $\in [0, 1]$, where $ACC \approx 1$ is a perfect clustering result.

Problem	Nodes	Clusters	Eps-Gauss	kNN-Max	Eps-Cnn
Aggregation	788	7	0.9909	0.9673	0.9949
Compound	399	6	0.8381	0.8005	0.8135
Flame	240	2	0.6417	0.8258	0.6417
Jain	373	2	0.8123	0.7694	0.8123
Pathbased	300	3	0.7467	0.7843	0.7467
R15	600	15	0.5133	0.9722	0.4980
Spirals	312	3	0.3756	0.7673	0.3788

Table 2. For each row, on the left hand side of the central column, we have the problem tested, with its number of nodes and clusters. On the right hand side we have the RCut (9) computed on the different graph configurations introduced in section 3.1.

Problem	Nodes	Clusters	Eps-Gauss	kNN-Max	Eps-Cnn
Aggregation	788	7	19.9625	6.1911	16.4231
Compound	399	6	22.9116	4.0713	20.6457
Flame	240	2	8.6180	0.6552	8.9985
Jain	373	2	0.6052	0.2311	0.6341
Pathbased	300	3	2.1485	1.4283	2.2456
R15	600	15	293.5543	12.1285	303.6924
Spirals	312	3	8.8360	0.1629	8.9356

As in the case of the three-kernels dataset (see Figure 5a and Figure 5b), we notice in Table 1 that the kNN-Max outperforms in terms of ACC on the other two configurations, and in Table 2 it dominates in terms of Rcut minimizations. In the 58% of the cases, the ACC with kNN-Max is higher than the other two, and in the 100% of the cases, the RCut with kNN-Max is smaller. The percentages given, are computed by counting the best configuration in every dataset. Moreover, if two entries are equal, then every configuration gets half of the point gained, and a third if three entries are the same. In general, we can define c_k as the number of times configuration k has been the best one.

$$c_k = \sum_{i=1}^{23} \frac{\iota(y_{i_k} = \hat{y}_i)}{m_i}, \tag{27}$$

where y_{i_k} is the score we get on dataset i with configuration k, $\hat{y}_i$ is the maximum score for dataset i, that is

$$\hat{y}_i = \max_{k \in \{1,\ldots,K\}} y_{i_k}, \tag{28}$$

m_i is the number of configurations giving a scores which is equal to $\hat{y}_i$ on dataset i, and $\iota(y_{i_k} = \hat{y}_i)$ is an indicator variable that equals to 1 if $y_{i_k} = \hat{y}_i$ and 0 otherwise.

In particular, one can notice that, when a dataset has clusters with high-variation of distances among them (e.g the R15 [6]), that is the distances between data points are different in different regions of the space [26], the construction of the similarity graph using the ϵ-neighborhood technique fails in terms of RCut. This is due to the fact that, for preserving the connectivity of the graph, one needs an high epsilon value, which will

hinder the clustering process, because it will result into a fully connected graph, hence the adjacency matrix will not have a clear block diagonal structure.
Therefore, it seems reasonable to use as first choice the kNN-Max configuration, since is apparent from the result that it is the most robust configuration.

3.2 Evaluating the performance of different graph Laplacians

With the results obtained in sections 3.1.2 and 3.1.3, the best spectral clustering configuration is the kNN-Max (see section 3.1), since it outperforms on average the other two. Therefore, we utilize it as a graph construction method in order to evaluate the performance of the different graph Laplacian configurations. We measure the effectiveness of each graph Laplacian variant based on the values of accuracy (24), Ratio cut (9), Normalized cut (10) and modularity (25). As stated in 2.2.3, one can distinguish three different Laplacian forms:

1. *Unnormalized Laplacian* (UL), defined in (17).
2. *Normalized Symmetric Laplacian* (NSL), defined in (18).
3. *Normalized Random Walk Laplacian* (RWL), defined in (19).

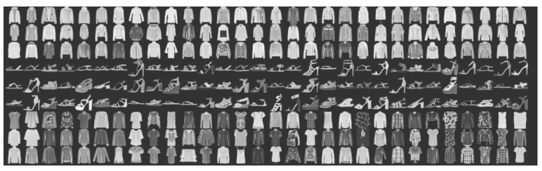
For RWL the beta factor equals to $\beta = 1$, and corresponds to the regular spectral analysis as outlined in [12]. Hence, we distinguish in our evaluation another type of Laplacian that we call Random Walk Beta Laplacian (RWBL), for which we will include the best resulting beta for each dataset in terms of modularity, for $\beta \in [1.1, 1.9]$ with step $\beta = \beta + \Delta\beta$, with $\Delta\beta = 0.1$. We know, from [26], that finding the eigenvectors of the unnormalized Laplacian, corresponds to a minimization of the RCut, and finding the eigenvectors of the normalized Laplacian, leads to a minimization of the NCut. Therefore, for a fair comparison, to determine the best RWBL configuration, and compare it with the other setups, we select modularity in order to determine the best beta β value for each dataset.

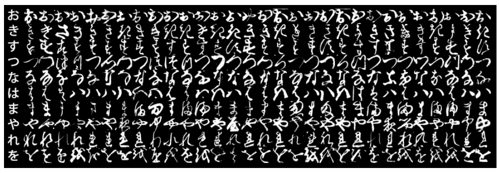
3.2.1 Laplacian Selection

In the final phase of testing, we run our numerical experiments on a total of 23 datasets. We consider once more the shape sets from [8], and moreover we evaluate real-life datasets used in machine learning studies from the OpenML [1] database [25]. These datasets have ground-truth labels, and describe a variety of clustering tasks. The objective is to group accurately different objects in classes according to their labels. Some examples of these datasets are depicted in Figure 7.
Initially, we compute the adjacency matrix generated using the kNN-Max's configuration (see section 3.1) for all the datasets, with different values of k in order to guarantee a connected graph. Subsequently, we measure the ACC, the RCut, the Ncut, and the MOD for the various Laplacians.
We organize the datasets in three sets:

1. Artificial [8] test case (ATS) .
2. Small ($<$ 5000 nodes) openML test case (SOTS).
3. Big ($>$ 10000 nodes) openML test case (BOTS).

[1] https://www.openml.org/


(a) Fashion-MNIST


(b) KMNIST

(c) USPS

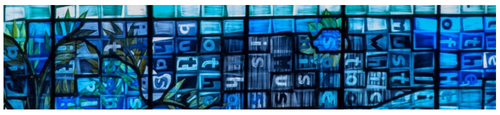
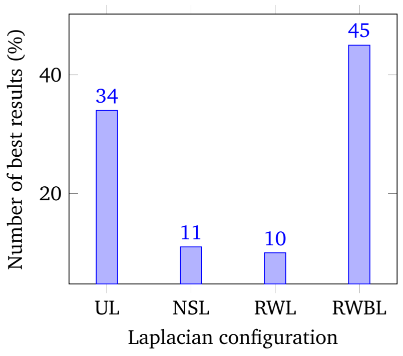
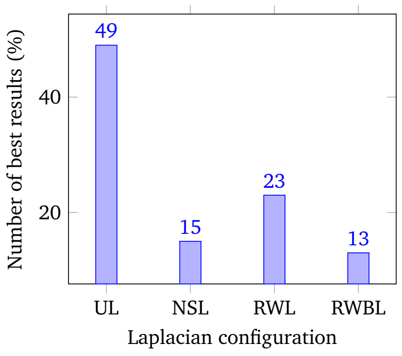

(d) Glass Identification

Figure 7. Some examples of the datasets used in machine learning applications. Figure 7a represents the Fashion-MNIST, that is a dataset of Zalando's article images [28].; Figure 7b, represents the the Kuzushiji-MNIST (KMNIST) dataset, which includes characters in Hiranaga, based on pre-processed images of characters from 35 books from the 18th century [23]; Figure 7c, shows the USPS dataset, which refers to numeric data obtained from the scanning of handwritten digits from envelopes by the U.S. Postal Service [11]; Figure 7d, represents the Glass Identification database, whose study of classification of types of glass was motivated by criminology's investigation [7].

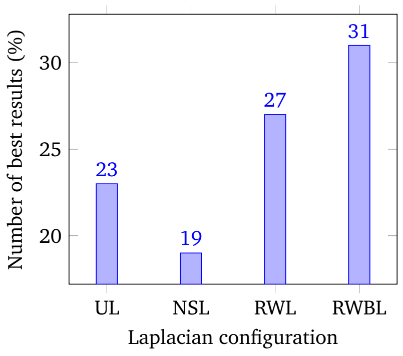
The tables in Appendix B show the ACC Table 3, Rcut Table 4, NCut Table 5 and MOD Table 6 for the different test cases respectively. In Figure 8 we summarize our results with a comparative study between the different Laplacian configurations for each of the metrics considered.



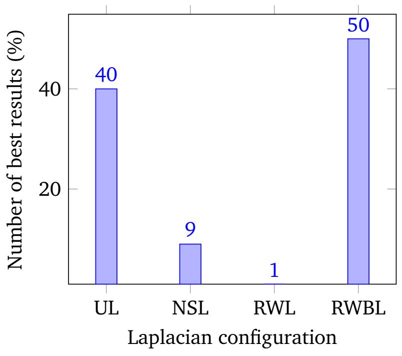
(a) Accuracy (ACC)



(b) Ratio cut (RCut)



(c) Normalized cut (NCut)



(d) Modularity (MOD)

Figure 8. The percentage of occurrences that every Laplacian configuration performed the best, for each considered metric, over the 23 datasets. Figure 8a shows the best configuration with respect to ACC, based on the results in Table 3; Figure 8b, shows the best configuration with respect to RCut, based on the results in Table 4; Figure 8c, shows the best configuration with respect to NCut, based on the results in Table 5; Figure 8d, the best configuration with respect to MOD, based on the results in Table 6.

One can notice that, in Figure 8a the ACC in the 45% of the cases with the RWBL is the highest. From Figure 8b and Figure 8c we can confirm that normalized spectral clustering leads to a minimization of the Ratio cut, while its normalized counterpart to a minimization of the Normalized cut [26]. Indeed, in the 49% of the cases, the UL configuration gives better RCut minimization, whereas the RWBL and RWL in the 31% and 27% respectively gives NCut smaller values. From Figure 8d, we can confirm that the beta factor plays an important role in the spectral clustering results [12]. In 50% of the cases, the modularity computed for each dataset using the RWBL is the highest. Notice also that we compute the percentages following (27) when dealing with the best similarity graph configuration.
From these results we can infer that the Random Walk Beta Laplacian with a varying β factor (19), with the kNN-Max (see section 3.1) similarity graph, performs the best on average. It is the spectral clustering configuration which outperforms the rest in terms of accuracy and modularity maximization, while at the same time minimizes efficiently the Normalized cut, for datasets of different shapes and properties.

4 Conclusion and future work

In this work, we have been focusing on conducting experiments in a wide variety of graphs emerging from artificial and real-life machine learning datasets. We conduct a study on artificial graphs with a varying size and shape (see sections 3.1.2 and 3.1.3) in order to choose the best graph creation routine, and observe that the similarity graph construction is crucial to the performance of spectral clustering (see section 3.1). Moreover, the selection of good parameters (see section 3.1.1) to tune the clustering process cumbersome and requires a heuristics approach together with trial and error attempts in order to guarantee the connectivity and sparsity of the graph. The type of graph Laplacian (see section 3.2) selected is identified as the key point of the spectral clustering routine. Numerical experimental on both artificial datasets and datasets emerging from machine learning applications show that the proposed RWBL, with the kNN-Max graph configuration, outperforms UL, NSL, RWL. Our study is based on various evaluation metrics (see section 3), in an effort to present a thorough and detailed comparison.
For future work, we want to:

1. Cluster datasets without labels with the best setup that we found, i.e the RWBL with the kNN-Max graph configuration.

2. Find meaningful communities in complex networks, such as Social and Protein-Protein interaction Networks. Particularly attractive is the application in Power Law Graphs,

3. The CNN (15) similarity function, is suggested as the best approach in [32], and this was not justified in our research. Therefore, we want find the scenarios in which CNN would be the optimal graph construction method.

4. Transfer the project in a more performant programming language, i.e C, where we can optimize freely the code.

A The project in a nutshell

A.1 Project requirements and challenges

The main goal in this study is finding meaningful communities in machine learning applications. In order to achieve this, we select the best graph constructions, based on connectivity and similarity functions for capturing the real similarities and differences of the nodes, by using spectral clustering techniques. Then we select appropriate quality metrics for the final results.
However, the unique nature of spectral clustering posed also some challenges in achieving the desired results. These are the issues we will face in our study:

1. Finding the most suitable similarity and connectivity functions, since capturing the true similarities and differences between data points for different datasets is quite challenging.

2. Finding the best graph Laplacian to work on, since we have no general rule on choosing which Laplacian matrix is the most appropriate.

A.2 Technologies used

In order to apply these spectral clustering techniques, we have to choose firstly the programming language which enable us to work easily with the different datasets. During our study, we need to visualize the results in a fashionable way, retrieve features of dataset easily, and performs matrix/vector operations efficiently for our algorithms. Therefore the most obvious choice is Matlab, because it is an optimized language for technical computing. It integrates computation, visualization (for example high-level commands for two-dimensional and three-dimensional data visualization,), and programming in an easy-to-use environment. It includes facilities for managing the variables in our work-space and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling, i.e testing efficiency of algorithms.

A.3 Timeline

The first thing we did during the first meeting was planning the entire workflow for the project. Thereby we decided to divide the workload in weeks:

1. *Week 1 - 3*: Study of different similarity and connectivity functions between data points and implementation on Matlab. Start the report.

2. *Week 3 - 5*: Study of different graph Laplacian and creation of them from adjacency matrix based on similarity and connectivity, and report writing.

3. *Week 5 - 6*: Selecting appropriate metrics to evaluate the final result, and report writing.

4. *Week 6 - 8*: Final clustering step on the spectral coordinate space, and report writing.

5. *Week 8 - 10*: Application to machine learning datasets, and report writing.

6. *Week 10 - 12*: Optimize the code and finalizing the report.

B Results

Table 3. Each block of the table shows the ACC results for problems of different test cases, and for each row, on the left hand side of the table, we have the name of the problem tested, with its number of nodes and clusters. On the right hand side we have the ACC (24) computed on the different Laplacian configurations introduced in section 3.2. Each measure gives an value $\in [0,1]$, with $ACC \approx 1$ is a perfect spectral clustering result.

ACC							
Test case	**Problem**	**Nodes**	**Clusters**	**UL**	**NSL**	**RWL**	**RWBL**
ATS	Aggregation	788	7	0.9797	0.9886	0.9797	0.9530 ($\beta = 1.1$)
	Compound	399	6	0.8296	0.8321	0.8296	0.8321 ($\beta = 1.5$)
	Flame	240	2	0.8208	0.8125	0.8125	0.8125 ($\beta = 1.1$)
	Jain	373	2	0.7694	0.7748	0.7694	0.7694 ($\beta = 1.9$)
	Pathbased	300	3	0.7833	0.7800	0.7800	0.7833 ($\beta = 1.3$)
	R15	600	15	0.9917	0.9933	0.9900	0.9950 ($\beta = 1.2$)
	Spirals	312	3	0.7660	0.7724	0.7660	0.8526 ($\beta = 1.6$)
SOTS	Binaryad	1404	36	0.5036	0.4067	0.4772	0.4067 ($\beta = 1.2$)
	Ecoli	336	8	0.8304	0.8274	0.8244	0.8244 ($\beta = 1.1$)
	Glass	214	6	0.5841	0.5888	0.6028	0.5935 ($\beta = 1.3$)
	Iris	150	3	0.9067	0.9000	0.9067	0.9067 ($\beta = 1.2$)
	Mice	1077	8	0.4308	0.3909	0.4067	0.4717 ($\beta = 1.1$)
	Olivetti	400	40	0.6125	0.6550	0.6275	0.6775 ($\beta = 1.3$)
	Plants	1600	100	0.4831	0.4625	0.4913	0.4850 ($\beta = 1.4$)
	Spectro	531	48	0.5706	0.5631	0.5593	0.5593 ($\beta = 1.1$)
	UMIST	575	20	0.6104	0.5739	0.5913	0.5983 ($\beta = 1.7$)
	Vehicle	846	4	0.4586	0.4740	0.4740	0.4811 ($\beta = 1.4$)
	Yeast	1484	12	0.5364	0.5256	0.5391	0.5512 ($\beta = 1.8$)
BOTS	Fashion	10000	10	0.6009	0.5964	0.5994	0.5984 ($\beta = 1.1$)
	Kmnist	10000	10	0.4044	0.4056	0.4179	0.4298 ($\beta = 1.6$)
	MNIST	10000	10	0.6859	0.6885	0.6877	0.7040 ($\beta = 1.1$)
	Pendigits	10992	10	0.7962	0.8006	0.8004	0.8237 ($\beta = 1.6$)
	USPS	11000	10	0.6327	0.6039	0.6085	0.5975 ($\beta = 1.6$)

Table 4. Each block of the table shows the RCut results for problems of different test cases, and for each row, on the left hand side of the table, we have the name of the problem tested, with its number of nodes and clusters. On the right hand side we have the RCut (9) computed on the different Laplacian configurations introduced in section 3.2.

RCut							
Test case	**Problem**	**Nodes**	**Clusters**	**UL**	**NSL**	**RWL**	**RWBL**
ATS	Aggregation	788	7	5.2148	4.1757	5.2148	6.4479 ($\beta = 1.1$)
	Compound	399	6	3.2400	3.1847	3.2091	3.1567 ($\beta = 1.5$)
	Flame	240	2	0.6469	0.6435	0.6435	0.6435 ($\beta = 1.1$)
	Jain	373	2	0.2311	0.2575	0.2311	0.2311 ($\beta = 1.9$)
	Pathbased	300	3	1.4228	1.4597	1.4404	1.4228 ($\beta = 1.3$)
	R15	600	15	7.3916	7.5810	8.0550	7.0439 ($\beta = 1.2$)
	Spirals	312	3	0.1467	0.1554	0.1467	0.2047 ($\beta = 1.6$)
SOTS	Binaryad	1404	36	4.0359	5.6116	4.4036	5.0669 ($\beta = 1.2$)
	Ecoli	336	8	0.8089	0.9332	0.8249	0.8116 ($\beta = 1.1$)
	Glass	214	6	0.4677	0.4866	0.4291	0.4292 ($\beta = 1.3$)
	Iris	150	3	0.6987	0.6246	0.6987	0.6987 ($\beta = 1.2$)
	Mice	1077	8	0.1037	0.1884	0.0872	0.1169 ($\beta = 1.1$)
	Olivetti	400	40	6.4453	7.7264	7.0043	6.9223 ($\beta = 1.3$)
	Plants	1600	100	18.9317	22.4580	19.5665	21.3464 ($\beta = 1.4$)
	Spectro	531	48	10.2185	12.2537	11.7630	11.9475 ($\beta = 1.1$)
	UMIST	575	20	1.4178	1.3557	1.3131	1.4466 ($\beta = 1.7$)
	Vehicle	846	4	0.0781	0.0515	0.0445	0.0789 ($\beta = 1.4$)
	Yeast	1484	12	7.0115	7.2757	7.1817	8.2145 ($\beta = 1.8$)
BOTS	Fashion	10000	10	0.2478	0.2938	0.2507	0.2616 ($\beta = 1.1$)
	Kmnist	10000	10	0.3402	0.4240	0.3429	0.3944 ($\beta = 1.6$)
	MNIST	10000	10	8.4066	8.6070	8.4855	9.2056 ($\beta = 1.1$)
	Pendigits	10992	10	0.1228	0.0933	0.1013	0.1048 ($\beta = 1.6$)
	USPS	11000	10	0.7982	0.9132	0.8667	0.8789 ($\beta = 1.6$)

Table 5. Each block of the table shows the NCut results for problems of different test cases, and for each row, on the left hand side of the table, we have the name of the problem tested, with its number of nodes and clusters. On the right hand side we have the NCut (10) computed on the different Laplacian configurations introduced in section 3.2.

NCut							
Test case	**Problem**	**Nodes**	**Clusters**	**UL**	**NSL**	**RWL**	**RWBL**
ATS	Aggregation	788	7	0.2182	0.1742	0.2182	0.3063 ($\beta = 1.1$)
	Compound	399	6	0.3377	0.3317	0.3343	0.3287 ($\beta = 1.5$)
	Flame	240	2	0.0699	0.0695	0.0695	0.0695 ($\beta = 1.1$)
	Jain	373	2	0.0234	0.0261	0.0234	0.0234 ($\beta = 1.9$)
	Pathbased	300	3	0.1365	0.1392	0.1380	0.1365 ($\beta = 1.3$)
	R15	600	15	0.3128	0.3212	0.3400	0.2980 ($\beta = 1.2$)
	Spirals	312	3	0.0374	0.0396	0.0374	0.0524 ($\beta = 1.6$)
SOTS	Binaryad	1404	36	9.1447	7.9539	8.6083	7.8923 ($\beta = 1.2$)
	Ecoli	336	8	0.6403	0.7100	0.6463	0.6362 ($\beta = 1.1$)
	Glass	214	6	0.3488	0.2791	0.2599	0.2597 ($\beta = 1.3$)
	Iris	150	3	0.1008	0.0897	0.1008	0.1008 ($\beta = 1.2$)
	Mice	1077	8	0.0607	0.1072	0.0498	0.0696 ($\beta = 1.1$)
	Olivetti	400	40	11.6204	10.1723	9.3445	9.2189 ($\beta = 1.3$)
	Plants	1600	100	23.0591	20.2876	19.6536	20.9721 ($\beta = 1.4$)
	Spectro	531	48	16.1534	14.3321	14.0712	15.0996 ($\beta = 1.1$)
	UMIST	575	20	1.0506	0.9195	0.9221	0.9876 ($\beta = 1.7$)
	Vehicle	846	4	0.0796	0.0468	0.0394	0.0810 ($\beta = 1.4$)
	Yeast	1484	12	2.0827	2.1336	2.1289	2.4186 ($\beta = 1.8$)
BOTS	Fashion	10000	10	0.5154	0.6078	0.5214	0.5234 ($\beta = 1.1$)
	Kmnist	10000	10	0.7894	0.9676	0.7873	0.9181 ($\beta = 1.6$)
	MNIST	10000	10	0.9004	0.9208	0.9084	0.9848 ($\beta = 1.1$)
	Pendigits	10992	10	0.0578	0.0438	0.0483	0.0505 ($\beta = 1.6$)
	USPS	11000	10	0.7319	0.8180	0.7853	0.7946 ($\beta = 1.6$)

Table 6. Each block of the table shows the MOD results for problems of different test cases, and for each row, on the left hand side of the table, we have the name of the problem tested, with its number of nodes and clusters.. On the right hand side we have the MOD (25) computed on the different Laplacian configurations introduced in section 3.2. Each measure gives a modularity value MOD $\in [0, 1]$, where MOD ≈ 1 is the highest modularity.

MOD							
Test case	**Problem**	**Nodes**	**Clusters**	**UL**	**NSL**	**RWL**	**RWBL**
ATS	Aggregation	788	7	0.7843	0.7843	0.7843	0.8017 ($\beta = 1.1$)
	Compound	399	6	0.7410	0.7418	0.7415	0.7423 ($\beta = 1.5$)
	Flame	240	2	0.4632	0.4623	0.4623	0.4623 ($\beta = 1.1$)
	Jain	373	2	0.4883	0.4868	0.4883	0.4883 ($\beta = 1.9$)
	Pathbased	300	3	0.5901	0.5858	0.5885	0.5901 ($\beta = 1.3$)
	R15	600	15	0.9133	0.9129	0.9116	0.9143 ($\beta = 1.2$)
	Spirals	312	3	0.5808	0.5845	0.5808	0.6490 ($\beta = 1.6$)
SOTS	Binaryad	1404	36	0.7314	0.6873	0.7187	0.7336 ($\beta = 1.2$)
	Ecoli	336	8	0.7756	0.7692	0.7740	0.7755 ($\beta = 1.1$)
	Glass	214	6	0.7540	0.7146	0.7240	0.7243 ($\beta = 1.3$)
	Iris	150	3	0.6229	0.6252	0.6229	0.6229 ($\beta = 1.2$)
	Mice	1077	8	0.7549	0.6990	0.7163	0.7558 ($\beta = 1.1$)
	Olivetti	400	40	0.7332	0.7136	0.7235	0.7369 ($\beta = 1.3$)
	Plants	1600	100	0.8141	0.7784	0.8132	0.8089 ($\beta = 1.4$)
	Spectro	531	48	0.7522	0.7108	0.7200	0.7208 ($\beta = 1.1$)
	UMIST	575	20	0.8860	0.8918	0.8896	0.8852 ($\beta = 1.7$)
	Vehicle	846	4	0.6985	0.6564	0.6586	0.6681 ($\beta = 1.4$)
	Yeast	1484	12	0.6991	0.6912	0.6950	0.6940 ($\beta = 1.8$)
BOTS	Fashion	10000	10	0.8230	0.8152	0.8227	0.8230 ($\beta = 1.1$)
	Kmnist	10000	10	0.7107	0.7181	0.7179	0.7342 ($\beta = 1.6$)
	MNIST	10000	10	0.7851	0.7900	0.7912	0.7947 ($\beta = 1.1$)
	Pendigits	10992	10	0.8594	0.8640	0.8632	0.8671 ($\beta = 1.6$)
	USPS	11000	10	0.7897	0.7816	0.7849	0.7828 ($\beta = 1.6$)

References

[1] H. M. A. Gionis and P. Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2007.

[2] F. Bach and M. Jordan. *Spectral Clustering for Speech Separation*. 2009.

[3] T. Buhler and M. Hein. Spectral clustering based on the graph p-laplacian. *ICML '09*, 2009.

[4] H. Chang and D. Yeung. Robust path-based spectral clustering. *Pattern Recognition*, 2008.

[5] Y. Chen, X. Li, J. Liu, G. Xu, and Z. Ying. Exploratory item classification via spectral graph clustering. *Applied Psychological Measurement*, 2017.

[6] M. R. C.J. Veenman and E. Backer. A maximum variance cluster algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2002.

[7] I. W. Evett and E. J. Spiehler. Rule induction in forensic science. *Central Research Establishment. Home Office Forensic Science Service*, 1987.

[8] P. Franti and S. Sieranoja. K-means properties on six clustering benchmark datasets. *Applied Intelligence, 48 (12), 4743-4759,*, 2018.

[9] L. Fu and E. Medico. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. *BMC bioinformatics*, 2007.

[10] Z. Huang, Z. Zhang, and G. Yu. Multivalued function recognition based on spectral clustering. *Journal of Physics: Conference Series*, 2020.

[11] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[12] K. Inoue, W. Li, and H. Kurata. Diffusion model based spectral clustering for protein-protein interaction networks. *PloS one*, 2010.

[13] A. Jain and M. Law. Data clustering: A user's dilemma. *Lecture Notes in Computer Science*, 2005.

[14] J. Kools. 6 functions for generating artificial datasets. `https://www.mathworks.com/matlabcentral/fileexchange/41459-6-functions-for-generating-artificial-datasets`, 2013.

[15] C.-T. Kuo, P. Walker, O. Carmichael, and I. Davidson. Spectral clustering for medical imaging. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2015.

[16] X. Li and Z. Wang. A new recommendation algorithm combined with spectral clustering and transfer learning. *Cluster Computing*, 2019.

[17] D. Luo, H. Huang, C. Ding, and F. Nie. On the eigenvectors of p-laplacian. *Machine Learning*, 2010.

[18] N. ME and G. M. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys 69*, 2004.

[19] S. Ramasubbareddy, T. Srinivas, K. Govinda, and S. Manivannan. *Comparative Study of Clustering Techniques in Market Segmentation*, pages 117–125. 03 2020.

[20] S. Renukadevi and S. Murugappan. Spectral cluster based temporal feature extraction and b-tree indexing for video retrieval. *Journal of Theoretical and Applied Information Technology*, 2017.

[21] J. Sourati, D. Brooks, J. Dy, and D. Erdogmus. Constrained spectral clustering for image segmentation. *IEEE International Workshop on Machine Learning for Signal Processing*, 2012.

[22] J. Szymanski and T. Dziubich. Spectral clustering wikipedia keyword-based search results. *Frontiers in Robotics and AI*, 2017.

[23] A. K. A. L. e. a. Tarin Clanuwat, Mikel Bober-Irizar. Deep learning for classical japanese literature. *Computer Vision and Pattern Recognition*, 2018.

[24] S. Trivedi, Z. A. Pardos, N. T. Heffernan, and G. N. Sarkozy. Spectral clustering in educational data mining. *EDM 2011 - Proceedings of the 4th International Conference on Educational Data Mining*, 2011.

[25] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[26] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 2007.

[27] S. T. Wierzchon and M. A. Klopotek. *Modern Algorithms of Cluster Analysis*. Springer, 2017.

[28] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[29] Q. Yang, N. Yang, T. Browning, B. Jiang, and T. Yao. Clustering product development project organization from the perspective of social network analysis. *IEEE Transactions on Engineering Management*, 2019.

[30] C. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 1971.

[31] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. *Advances in Neural Information Processing Systems*, 2004.

[32] X. Zhang, J. Li, and H. Yu. Local density adaptive similarity measurement for spectral clustering. *Pattern Recognition Letters*, 2011.