Spectrum: Fast Density-Aware Spectral Clustering for Single and Multi-Omic Data

Arthur Clarysse - 0616411
arthur.clarysse@vub.be
Vrije Universiteit Brussel
Brussels, Belgium

Guilaume Tibergyn - 0618147
guilaume.tibergyn@vub.be
Vrije Universiteit Brussel
Brussels, Belgium

Abstract

Clustering is a fundamental task in machine learning and bioinformatics. The goal is to group objects so that those in the same cluster are more similar to each other than to those in different clusters. In biomedical science, clustering is central to applications such as precision medicine, where patients are grouped based on genome-wide expression data, or identifying cell types where single-cell RNA-sequencing is used and cells are clustered based on their transcriptomes. However, clustering biomedical data poses several challenges: datasets can be massive, exhibit dense and globular cluster structures, or contain high-dimensional features. Furthermore, many clustering algorithms require the number of clusters to be specified a priori, which is often unknown in practice. In this paper, we implement Spectrum, a fast, density-aware spectral clustering algorithm designed for single- and multi-omic data, based on the method proposed by John et al.. Spectrum employs an adaptive density-aware kernel that integrates the Zelnik-Manor and Zhang kernels, enabling it to capture non-Gaussian cluster structures better. To address the unknown number of clusters, we incorporate a method to estimate the optimal number of clusters, k^*, based on the multimodality of eigenvectors. We compare this approach with the traditional eigengap method and demonstrate that the multimodality criterion yields superior performance on non-Gaussian datasets. We benchmark the Spectrum implementation against the scikit-learn spectral clustering algorithm, showing that Spectrum achieves more accurate and robust clustering, particularly for complex data distributions. Additionally, our Python implementation reproduces the results of the original R-based version, validating its correctness. Finally, we confirm that the choice of kernel significantly impacts performance and that the adaptive density-aware kernel offers notable advantages. Our implementation and experimental results are publicly available at our GitHub page[1].

1 Introduction

Precision medicine aims to tailor medical treatment to the individual characteristics of each patient [Hodson, 2016], [Kosorok and Laber, 2019]. However, before this tailor-made treatment can be applied, we must cluster the patients into subsets with similar characteristics, defined a patient's genome-wide expression data such as mRNA, miRNA, protein, or methylation data. In this paper, we will focus on the ability to cluster patients using both single- and multi-omic data, possibly combining both.

Another use case for clustering in biomedical science is single-cell RNA-sequencing analysis, where clustering methods group individual cells' transcriptomes (mRNA). This technique helps identify and characterise distinct cell types within a heterogeneous biological sample. Because of these diverse applications, it is important for clustering methods to handle both single-cell datasets and multi-omic data effectively.

Clustering is a common task in machine learning and bioinformatics. It is the process of grouping a set of objects so that objects in the same group (or cluster) are more similar to each other than those in other groups, without knowing the number of clusters in advance [Omran et al., 2007]. While clustering is a common task, finding the right algorithm for the job is not always easy. Previously, some single-omic clustering techniques have been developed, like Monte Carlo consensus clustering [John et al., 2018], CLEST [Dudoit and Fridlyand, 2002], PINSPlus [Nguyen et al., 2019], and more. While even more challenging, multi-omic clustering techniques have also been developed, like iClusterPlus [Shen et al., 2009], CIMLR [Ramazzotti et al., 2018], and others.

Analysing single-cell RNA-sequencing (scRNA-seq) data introduces additional complexities. ScRNA-seq enables unbiased, high-throughput, and high-resolution transcriptomic profiling of individual cells [John et al., 2020]. Consequently, scRNA-seq datasets typically contain significantly more data points, which frequently form dense, globular clusters. These characteristics make accurate clustering challenging, often requiring sophisticated algorithms and considerable computational resources.

For the implementation of one multi-purpose clustering technique to be successful, it must meet several requirements, such as being able to handle large datasets and cluster large globular clusters. In addition to the technical requirements, the algorithm must also be able to handle different types of data, such as single-omic and multi-omic data. Furthermore, the algorithm must also be computationally efficient for large datasets.

This paper will implement Spectrum [John et al., 2020], a fast density-aware spectral clustering algorithm for single and multi-omic data. This algorithm was originally implemented in R. We will implement it in Python and compare its performance with the original R implementation. We will also explore why some choices were made and why a special clustering algorithm was needed.

Another thing we will do is compare the proposed clustering algorithm with the Sklearn implementation of spectral clustering[2]. The spectral cluster algorithm will directly use the normalised and preprocessed data and will not require a similarity matrix. However, it will perform many similar steps to the Spectrum algorithm in the background, such as calculating the Laplacian matrix and the eigenvalues. We could thus expect similar results, but with a much faster implementation.

[1] https://github.com/ClarysseArthur/VUB-AMB-Spectrum-Public

[2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html

Spectral clustering refers to a family of clustering techniques that use the eigenvalues of a similarity matrix to reduce the data's dimensionality before clustering in fewer dimensions. It uses kernels to define the similarity between data points, then uses the Laplacian matrix's eigenvalues to find the clusters. Several of the kernels have been proposed, such as the Zelnik-Manor kernel [Zelnik-manor and Perona, 2004] and the Zhang density-aware kernel [Zhang et al., 2011]. We will combine both into the Adaptive Density Aware (ADA) kernel used in Spectrum.

Density-aware refers to the fact that the algorithm can account for local density to improve performance by amplifying intra-cluster similarity [Zhang et al., 2011]. Combining spectral clustering and density awareness allows us to cluster the data more efficiently and use the algorithm on larger, more globular datasets.

The data used in this paper is the same as the one used in the original paper. It is preprocessed and normalised and available on the Synapse repository syn1891155. The data consists of 3 datasets:

1. Single-omic data used for patient clustering.
2. Multi-omic data, also used for patient clustering.
3. Single-cell RNA-sequencing data used for cell clustering.

2 Material and Methods

Before we can explain the adaptive density-aware kernel, we must explain the Zelnik-Manor kernel and the Zhang kernel, which are its foundations. Combining both kernels will be key in creating a good similarity matrix to feed into the spectral clustering algorithm.

2.1 Zelnik-Manor Kernel

The Zelnik-Manor kernel is a self-tuning kernel designed to adapt to the scale of the data. Because of its self-tuning ability, it does not require tuning hyperparameters, which is a common, time-consuming task in clustering algorithms. The kernel is defined as follows [Zelnik-manor and Perona, 2004]:

$$A_{ij} = \exp\left(\frac{-d^2(s_i, s_j)}{\sigma_i \sigma_j}\right) \tag{1}$$

With

- $A \in \mathbb{R}^{N \times N}$ the similarity matrix of a set of point: $S = \{s_1, ..., s_N\}$
- $A_{ij} \in \mathbb{R}$ the similarity between two points s_i and s_j.
- N the number of data points in the dataset.
- s_i and s_j are two data points in the dataset from which we want to calculate the similarity.
- $d(s_i, s_j)$ the Euclidean distance between the two data points s_i and s_j.
- σ_x a local scaling parameter for the data point s_x, defined as the Euclidean distance to the P-th nearest neighbor of s_x. This free parameter typically does not need to be tuned; the default value is a low, close neighbour, allowing for the kernel's self-tuning.

2.2 Zhang Kernel

The Zhang kernel is a density-aware kernel that uses the local density of the data points to define their similarity. To do this, it uses the common neighbours of two data points, increasing the similarity between the points when there are more common neighbours. Using this local density allows the kernel to increase intra-cluster similarity. The kernel is defined as follows [Zhang et al., 2011]:

$$A_{ij} = \exp\left(\frac{-d^2(s_i, s_j)}{2\sigma^2(\mathrm{CNN}(s_i, s_j) + 1)}\right) \tag{2}$$

With, different from the Zelnik-Manor kernel:

- σ a global scaling parameter (fixed).
- $\mathrm{CNN}(s_i, s_j)$ the number of common neighbours of the two data points s_i and s_j inside a radius of ϵ around the two points.

2.3 Adaptive Density Aware Kernel

Now we will present the combined kernel proposed in the original paper. It combines the advantages of the Zelnik-Manor and Zhang kernels while not requiring the tuning of the hyperparameters. This will also be the starting point of the Spectrum algorithm.

$$A_{ij} = \exp\left(\frac{-d^2(s_i, s_j)}{\sigma_i \sigma_j(\mathrm{CNN}(s_i, s_j) + 1)}\right) \tag{3}$$

With, different from the Zelnik-Manor and Zhang kernels:

- $\mathrm{CNN}(s_i, s_j)$ the number of common neighbours in the set of S nearest neighbours of the two data points s_i and s_j. This differs from the Zhang kernel, where the common neighbours are calculated within a radius, thus not an exact number.

The original study used the following static parameters: P=3 and S=7. These parameters can be tuned; higher values prefer global structures, while lower values prefer local ones. Moreover, these values were proved in the original paper to generalise well.

2.4 Spectrum Algorithm

This first step of the algorithm is to create the similarity matrix using the adaptive density-aware kernel. The input for the kernel is a matrix, one for each dataset:

$$T = \{E_1, ..., E_n\} \tag{4}$$

With

- T the set of datasets
- $E_i \in \mathbb{R}^{N_i \times M_i}$ the i-th dataset
- N_i the number of points
- M_i the number of features.

To continue the algorithm, we can use a single dataset or combine multiple datasets into one. If we choose the latter, we can combine the datasets using the TPG (Tangent Principal Graph) method. Otherwise, we can skip equation 5 and start with the next step.

Combining Multi-View Data and TPG Diffusion This is an optional step, and can be skipped if we want to use a single dataset. The TPG step is inspired by the work of Shu and Latecki. We will use cross-view TPGs because they capture the global structure of the data while also handling noise and outliers. To combine the different datasets, we will calculate the cross-sectional TPG from each pair of graphs of the datasets. Computing the linear combination of the graphs creates a single graph, after which graph diffusion is performed to reveal the connections within the data. Spectrum uses

a slightly adapted version of the method Shu and Latecki proposed. The TPG is defined as follows for datasets $Y = \{y_1, ..., y_n\}$:

$$A = \sum_{i=1}^{n} y_i \tag{5}$$

With

- A the TPG
- y_i the i-th dataset
- n the number of datasets

Note that each dataset is weighted equally, so noisy data should be removed before this step.

In the original paper, the authors start with a non-flexible K value, after which they adapt the algorithm to use a flexible K value. We will directly implement the flexible K value and provide the pseudo code for the algorithm's most important parts. K represents the number of clusters, and is a hyperparameter that must be tuned.

***Spectral Clustering with Flexible** K*

1. We start by dynamically selecting the dataset's best kernel and P value. The procedure to calculate these values is presented in Algorithm 1. The output of this step is the optimal kernel A^* based on the optimal P value, use this kernel for the rest of the algorithm.
2. We compute the kNN graph from the similarity matrix A^*, after which we row-normalise the graph. Then, we perform diffusion iterations on the row-normalised graph. This step is optional and can be skipped. The pseudo-code for this step is shown in algorithm 3. The default values for Z and t are 10 and 4, respectively.
3. Now we can calculate the Laplacian matrix L from the similarity matrix A^*. The Laplacian matrix is defined as:
$$L = D^{-1/2}AD^{-1/2} \tag{6}$$
4. Find the eigenvectors $V = \{v_1, ..., v_N\}$ of L by performing an eigen-decomposition on the Laplacian matrix L.
5. Now, calculate for each eigenvector v_i the dip statistic Z_i. The dip statistic is a measure of the unimodality of the distribution of the eigenvector. Then calculate the differences between two consecutive dip statistics $d_i = Z_i - Z_{i+1}$, creating the vector $D = \{d_1, ...d_{N-1}\}$.
6. We will now find the optimal number of clusters K by executing Algorithm 2 on the vector D. The algorithm finds the optimal number of clusters by finding the last substantial multimodal gap. This algorithm is used because the more straightforward method of finding the maximum multimodality gap is prone to local minima. In the next step, use the output, k^*, as input for K in the clustering algorithm.
7. We can now cluster the data using the Gaussian Mixture Model (GMM) algorithm. First, some smaller steps must be performed:
   a. Create the matrix $X = [x_1, ..., x_{k^*}]$ by selecting the k^* most significant eigenvectors from the Laplacian matrix L, and stacking them as columns.
   b. Normalise by performing L2 normalisation on the matrix X.
   c. Use the GMM algorithm to cluster the data in X. In Python, this can be done using the `GaussianMixture` class from the `sklearn.mixture` module.

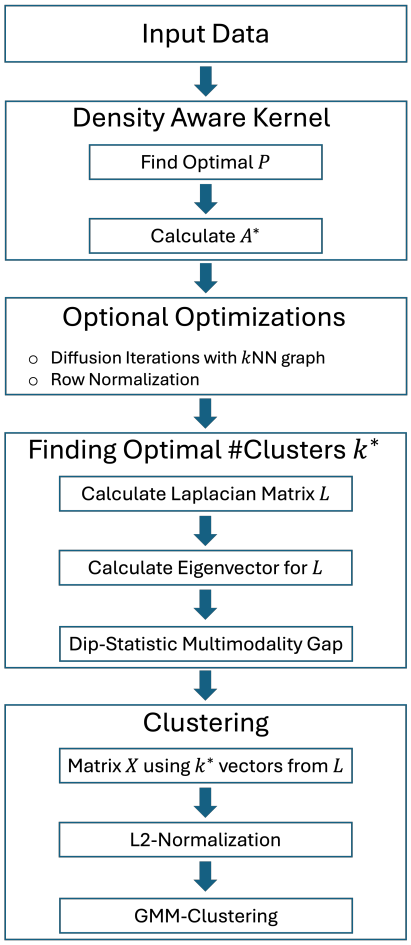

Figure 1: Flowchart visualization of the Spectrum algorithm.

2.5 Remarks & Choices of Spectrum

The traditional way to find the optimal number of clusters is to use the eigengap method. We first calculate the differences in eigenvalues, and then find the maximum difference. While this method is perfect for solving the problem for Gaussian distributions, it is not well suited for non-Gaussian distributions. This is visualised in Figures 3 and 4. The first figure shows that both methods identify the same number of clusters, which is to be expected. In the second figure, we can see that the traditional eigengap method identifies an arguably much worse number of clusters.

In the paper, we used a method where we first calculated the multimodality of the eigenvectors and then found the last point for which there was a substantial decrease in multimodality. We expect to find a significant drop in multimodality when testing the eigenvectors in a sorted order. To define the multimodality of an eigenvector, we use the dip statistic [Hartigan and Hartigan,

1985]. Looking at Figure 4, we can see that the multimodality of the eigenvectors is much more correct than the eigengap method. While estimating the same number of clusters as the eigengap method for Gaussian data. The multimodality method is better for finding the optimal number of clusters for non-Gaussian data.

2.6 Spectral Clustering

The spectral clustering algorithm is a common clustering algorithm that uses the eigenvalues of the Laplacian matrix to cluster the data. We will use the `sklearn.cluster.SpectralClustering` class to perform the clustering. An important part of the original paper that is not considered in this algorithm is how to find the optimal number of clusters k^*. Therefore, we will use the same method as the Spectrum algorithm to find the last substantial multimodality gap.

2.7 Datasets

We will use some of the datasets used in the original paper to test and evaluate the clustering algorithm. The complete dataset is a preprocessed and normalised combination of multi-omic data, RNA-seq data, and single-cell RNA-seq data from different sources. Naturally, these datasets are not all used for the same purpose, but the general goal is to create a clustering algorithm that can handle multi-omic and single-omic data.

TCGA Datasets - Patient Clustering The Cancer Genome Atlas Project (TCGA) is a cancer genomics program that has molecularly characterised over 20,000 primary cancer types. Since 2006, they have produced over 2.5 petabytes of genomic, epigenomic, transcriptomic, and proteomic data [noa, 2022]. This data is free to use for research purposes, and is available on the Genomic Data Commons (GDC) data portal[3]. The seven pre-normalised multi-omic datasets used in the original paper were downloaded from the Broad Institute[4], after which they were normalised using a $\log_2$ transformation. The last step they performed was to select the top 50% most variable features for each of the seven datasets.

The RNA-seq dataset was created using the same studies, data transformation and normalisation steps as the multi-omic datasets.

Single-Cell RNA-Sequencing Datasets - Cell Clustering The seven datasets used for this part were obtained from the Hemberg lab website[5] The same $\log_2$ transformation and normalisation steps were performed as on the TCGA dataset, and the top 100 most variable genes were selected.

Python Ready Datasets The preprocessed and normalised datasets used in the original paper are available on the Synapse repository syn18911550 [6]. However, these datasets are in R format and must be converted to a format better suited for Python. We converted the 21 datasets to a CSV format, and uploaded them to the GitHub repository of this project[7].

The datasets are constructed as follows: *location-x.csv* With *location* the name of the cancer location, and *x* the number of the dataset. In total, seven cancer locations were used, with three datasets per location.

The seven locations are:

1. Bladder cancer
2. Brain cancer
3. Breast cancer
4. Kidney cancer
5. Pheochromocytoma and Paraganglioma cancer
6. Skin cutaneous melanoma cancer
7. Thyroid cancer

The three datasets per location are:

1. multi-omic data
2. RNA-seq data
3. single cell RNA-seq data

Synthetic Datasets To test the algorithm, we also created some synthetic datasets. The first synthetic dataset is a Gaussian dataset with multiple clusters and around 200 data points. The dataset is generated using the `make_blobs` function from the `sklearn.datasets` module.

The second synthetic dataset is a non-Gaussian dataset with multiple clusters and around 200 data points. The dataset is generated using the `make_classification` function from the `sklearn.datasets` module. We did not choose to use the `make_moons` or similar generators, as it is not a good representation of the data we want to cluster. Figure 2 shows an example of the dataset.

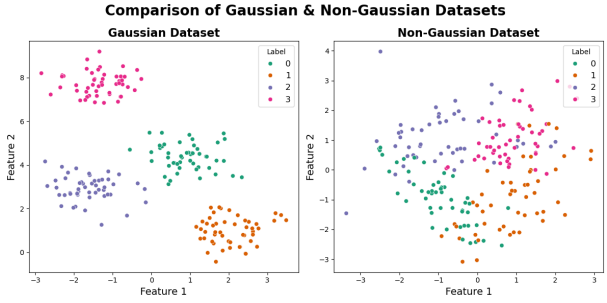

Figure 2: Two synthetic datasets with 5 clusters. Left: Gaussian dataset, right: non-Gaussian dataset.

3 Results

In this section, we will discuss the results of different aspects of the Python implementation of the Spectrum algorithm. We will start by evaluating k^* and comparing its performance to the original method of calculating the value using the eigengap method. After that, we will test the performance of the spectrum algorithm on the different synthetic datasets. Then we will compare the performance of the spectrum algorithm with the Sklearn implementation of spectral clustering. Lastly, we will compare the performance of the Python implementation with the original R implementation.

3.1 Finding The Optimal Number of Clusters

As explained in the methods chapter, the traditional eigengap method cannot find the optimal number of clusters for non-Gaussian

[3] https://portal.gdc.cancer.gov
[4] http://gdac.broadinstitute.org
[5] https://hemberg-lab.github.io/scRNA.seq.datasets
[6] https://synapse.org/Synapse:syn18911542/files
[7] https://github.com/ClarysseArthur/VUB-AMB-Spectrum-Public

data. We will compare the performance of the two methods on the synthetic datasets and show that the multimodality method is better suited for non-Gaussian data. Figure 2 shows the datasets used in this test. We used 200 datapoints divided into four clusters for both tests, and they are generated using the `make_blobs` and `make_classification` functions from the `sklearn.datasets` module.

The results of the Gaussian dataset show that both methods correctly identify the same number of clusters (green and orange lines in Figure 3). A clear peak is also visible in eigenvalues, suggesting that the eigengap method is suitable for finding the optimal number of clusters in Gaussian data.

However, the results of the non-Gaussian dataset show that the eigengap method fails to identify the correct number of clusters (green line in Figure 4). The multimodality method, on the other hand, correctly identifies the number of clusters (orange line). We can also clearly see why the eigengap method fails, as the eigenvalues do not show a clear peak. While much smaller than the eigenmaps, the multimodality of the eigenvectors on the fourth eigenvector does indeed peak.

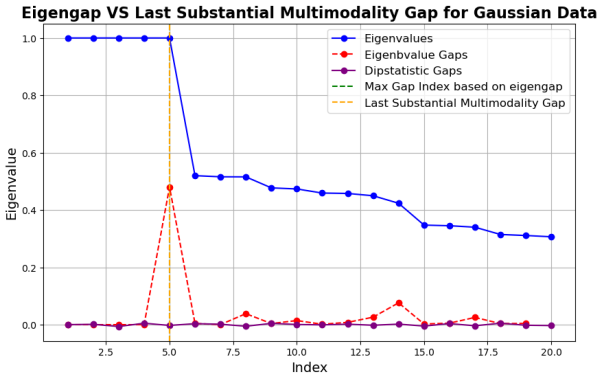

Figure 3: k^* for Gaussian data using eigengap and last substantial multimodality gap for clustering on the dataset showed in Figure 2 (left)

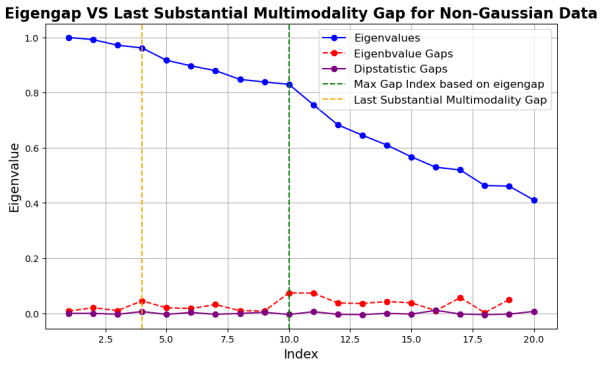

Figure 4: k^* for non-Gaussian data using eigengap and last substantial multimodality gap for clustering on the dataset showed in Figure 2 (right)

Looking at plots of one example does not give us a good idea of how accurate the k^* estimation is. Therefore, we will examine the relative difference between the known number of clusters and the calculated number of clusters for the two synthetic datasets. We will calculate this for 8 cluster amounts and 10 different datasets. The results will be compared using a t-test, and the null hypothesis will be: "*There is no significant difference in k^* quality between both methods*". We will use the relative distance define the quality of the k^* estimation. A lower values indicates a better estimation, and the relaitive distence is calculated as follows:

$$\text{Relative difference} = \frac{|k^* - k|}{k} \tag{7}$$

With

- k^* the calculated number of clusters.
- k the known number of clusters used to generate the clusters.

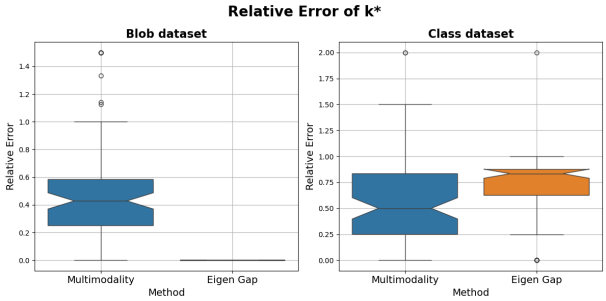

Figure 5: Relative difference between the known and calculated number of clusters for the two synthetic datasets. Left: Gaussian scores, right: non-Gaussian scores, note that two different scales are used. (Lower is better)

The p-value for both datasets was smaller than 0.05, so we can reject the null hypothesis. Resulting in a significant difference between the two methods. However, the direction of the difference depends on the dataset. For the Gaussian datasets, the eigengap method is much better than the multimodality method, almost always with an error of 0. Meanwhile, the multimodality method is much better for the non-Gaussian datasets than the eigengap method, resulting in an error of 0.3. The latter of the two results is more important than the former, as the data we want to predict is mostly non-Gaussian distributed.

Note that the predicted number of clusters is not always perfect; we can sometimes argue that it is maybe even better than the known number of clusters. But this is not that important; as long as the estimate lies within a reasonable range for the real data, it is good enough.

3.2 Synthetic Datasets

Now we will test the performance of the Spectrum algorithm on the synthetic datasets. We will use the same Gaussian and non-Gaussian datasets as in the previous section and analyse how well they are clustered compared to the original clusters. We will use the Adjusted Rand Index (ARI), which measures the similarity between

two clustering results, as a metric. We calculated the ARI using the `adjusted_rand_score` function from the `sklearn.metrics` module. The score is a value between -1 and 1, where one means that the two clusterings are identical, 0 means that they are random, and -1 means that they are entirely different, even less different than random clusters.

In Figure 6, we have plotted the resulting clusters of the Gaussian dataset; the ARI score is also shown. The ARI score is 0.97, which is very high and suggests that the clustering is almost perfect. The ARI score for the non-Gaussian dataset is 0.60, which is still reasonably good considering how much more difficult it is.

Note that the ARI score is not a perfect way to evaluate the clustering, as it is not always clear what the best clustering is, especially for non-Gaussian data, non-synthatic data.

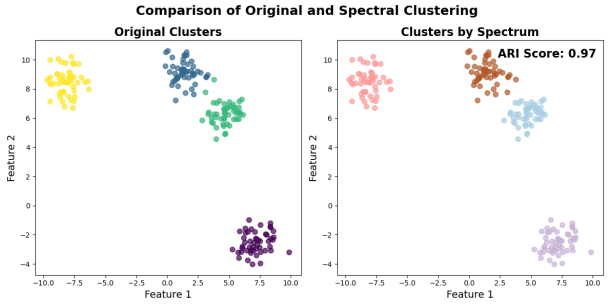

Figure 6: Clustering and ARI score for Gaussian data using the adaptive density aware kernel. Left: the original clusters, right: the clusters found by the algorithm.

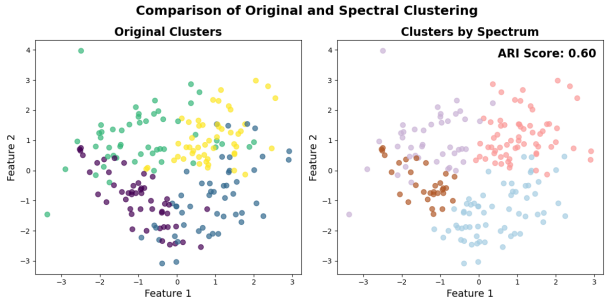

Figure 7: Clustering and ARI score for non-Gaussian data using the adaptive density aware kernel. Left: the original clusters, right: the clusters found by the algorithm.

3.3 Spectrum vs Sklearn SpectralClustering

In this section, we will compare the performance of the Spectrum algorithm with the Sklearn implementation of spectral clustering. We will do this in two ways: first, we will examine the performance of both algorithms on different datasets, and then we will examine the runtimes to see if there is a difference in performance.

Synthetic Datasets The first test we performed was to compare the performance of the spectral clustering algorithm with the Spectrum algorithm on the synthetic datasets. The datasets used are the same as the ones used for the previous section; the results of one test instance (the same one from the previous section) are shown in Figure 8. We have run both algorithms on 10 different datasets with eight different amounts of clusters for both distributions, totalling 160 clustering results. After that, we can compare both algorithms' ARI scores using the t-test and see if there is a significant difference.

The null hypothesis for both distribution types is: "*There is no significant difference in ARI score between both algorithms*". For this test, the number of classes used to generate the datasets was used as the value for k^*; this way, the results were not confounded by the quality of k^*. When we compared both algorithms on Gaussian data, we found that the p-value was 0.27, so we cannot reject the null hypothesis. So, we can assume that the two algorithms probably have no significant difference. However, when comparing the algorithms on non-gaussian data, the p-value was smaller than 0.05, so we can reject the null hypothesis. Meaning that there is a significant difference between the two algorithms. This conclusion is supported by the test results presented in Figure 9. We can see that the Spectrum algorithm's ARI scores are much higher than those of the Sklearn implementation. Note that the boxplots of the ARI scores for Gaussian data are not plotted. This is because all the scores were 1, and only a few outliers of 0.99 were present, and thus nothing useful could be shown.

Real Datasets It is challenging to compare the performance of the two algorithms on real datasets, as there is no ground truth available. We can, however, compare the runtimes of both algorithms and see if there is a significant difference.

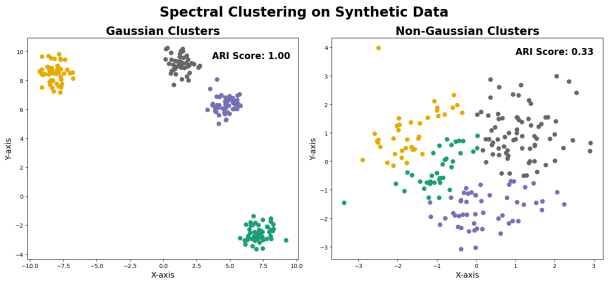

Figure 8: Clustering and ARI score for Gaussian and non-Gaussian data on the same dataset as section 3.2. Left: Gaussian data, right: non-Gaussian data

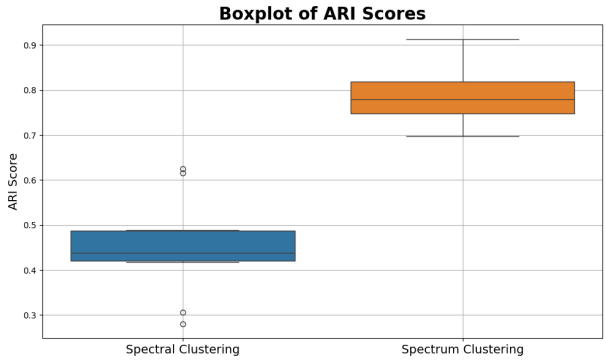

Figure 9: Boxplots of the ARI score for spectral clustering and the Spectrum algorithm on the non-Gaussian dataset. (Higher is better)

Runtime Behavior We will use the same Gaussian dataset as in the previous sections to test the two algorithms' runtime behaviour. We will calculate the runtimes of both algorithms on instances of 100 datapoints, with 2 to 10 classes, using 10 different datasets for each class amount. This totals to 80 tests for each algorithm. These results are then compared using a t-test, and the null hypothesis is: "*There is no significant difference in runtime between both algorithms*".

The test's p-value was smaller than 0.05, which means that we can reject the null hypothesis. Concluding that there is a significant difference in runtime between the two algorithms. The runtimes of the two algorithms are shown in Figure 10. We can confirm that the Spectrum algorithm takes much longer to find the clusters than the Sklearn implementation. Spectrum takes around 9 seconds, while the Sklearn implementation takes around 0.02 seconds.

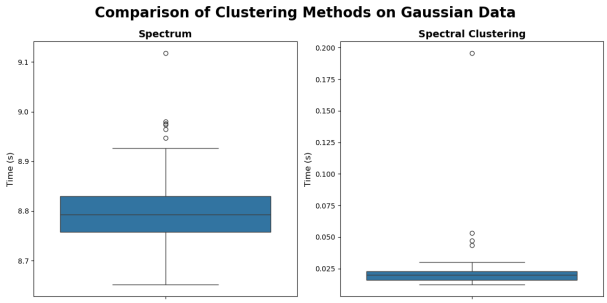

Figure 10: Runtime of the two algorithms on the Gaussian dataset. Left: Spectrum algorithm runtime, right: Sklearn implementation runtime. Note that two scales are used; the lower is better.

3.4 Python vs R

While it is impossible to directly compare the performance of the Spectrum algorithm in both languages, we can visually compare the results of the two algorithms. Figures 11 and 12 show the results of the two implementations on two of the three brain datasets. The results are very similar, and the two algorithms can find the same clusters. This proves that the Python implementation of the Spectrum algorithm is indeed good and that it can find correct clusters.

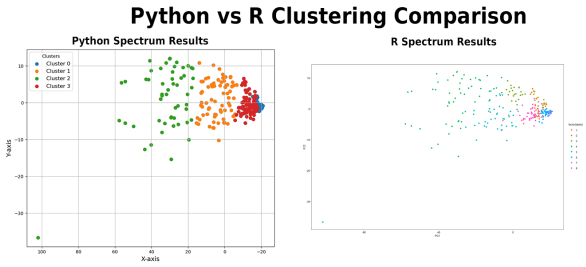

Figure 11: Visual comparison of clustering results between the Python and R implementations of the Spectrum algorithm on Brain 2. Left: Python implementation, right: R implementation.

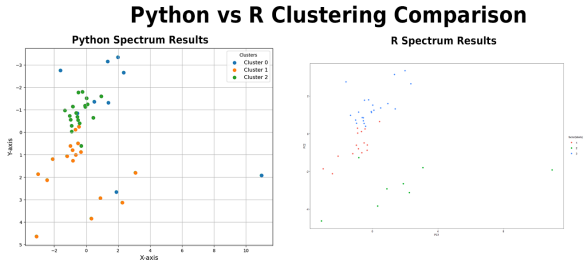

Figure 12: Visual comparison of clustering results between the Python and R implementations of the Spectrum algorithm on Brain 3. Left: Python implementation, right: R implementation.

Runtime comparison, however, is possible. We can compare the runtimes of the two implementations on the same dataset and see if there is a significant difference. Due to the limitations of the R implementation, we can only compare the runtimes of the two implementation on the three brain datasets. The runtimes are displayed in Table 1.

We can see that the runtimes of the two implementations are very different. The algorithm in R runs significantly faster than the Python implementation, which can be due to several reasons. One is that Python uses generic matrix structures and operations to perform the calculations, while R uses a much more optimised version of the same operations, and makes advantages of the sparsified A^* matrix.

	Brain 1	Brain 2	Brain 3
R	1223	0.85	0.16
Python	/	129	1.96

Table 1: Comparison across Brain metrics for R and Python in seconds, lower is better.

3.5 Kernels

The last thing we will compare is the performance difference between the two kernels on which the adaptive kernel is based and the adaptive kernel itself. We will do this by comparing the ARI scores of the two kernels on the synthetic datasets. We will calculate the score on the same synthetic non-Gaussian dataset as in the previous sections. The test results are collected on 10 datasets, each with eight different cluster amounts, totalling 80 datasets. The null hypothesis for the three comparisons is: "*There is no significant difference in ARI score between the two kernels*". To test this, we have performed a Mann-Whitney U test on the ARI scores of the two kernels. This test is chosen because the data is skewed. All p-values are larger than 0.05, so we cannot reject the null hypothesis. Suggesting that there is probably no significant difference between the two kernels. The results are shown in Figure 13, where we can confirm this conclusion.

Note that the ARI score is again not a perfect way to evaluate the clustering performance; however, it is a simple way to get an idea of the performance.

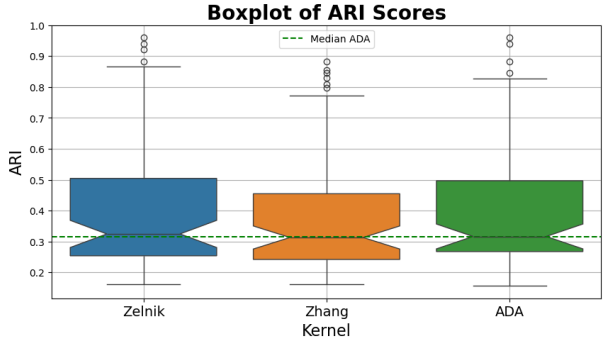

Figure 13: Boxplots of the ARI score for the three kernels on the non-Gaussian dataset. (Higher is better)

4 Discussion

In this study, we have presented a Python implementation of the Spectrum algorithm. This novel spectral clustering framework leverages a multimodality-based estimate of the optimal number of clusters and an Adaptive Density Aware (ADA) kernel to capture non-Gaussian structures. We have shown that the algorithm can find the optimal number of clusters (k^*) for non-Gaussian data based on the multimodality of the eigenvectors. Multiple tests were performed to evaluate the predicted value for k^*, and we found that the multimodality method is much better than the eigangap method for non-Gaussian data.

We have also shown that the Spectrum algorithm can very well cluster non-Gaussian data and that its performance is significantly better than that of the Scikit-learn implementation of spectral clustering. The three kernels were also compared, resulting in the surprising conclusion that there is no significant difference between them. Future work should explore the differences between the kernels in more detail and examine whether there are any differences in performance on real datasets.

An important difference was found when comparing the runtimes of Spectrum and Scikit-learn spectral clustering. The runtimes of the two algorithms are very different, with the Sklearn implementation being much faster than the Spectrum algorithm. Although this slow performance is acceptable for moderate-sized datasets, omics studies routinely involve thousands of samples and features. Furthermore, the original implementation in R runs significantly faster than the Python implementation. Future work could focus on optimising the Python implementation and determining whether its performance can be improved until it reaches the speed of the R implementation.

Fortunately, the Python implementation does produce the same results as the R implementation. When we compared the clustering results of the two implementations, they were very similar. This is a good indication that the Python implementation of the Spectrum algorithm is indeed good and that it can find correct clusters.

In summary, we have shown that the Spectrum algorithm is a powerful tool for clustering non-Gaussian data and that it can find the optimal number of clusters based on the multimodality of the eigenvectors. We have also shown that the results are very similar to the original implementation. However, the runtimes can be improved by optimising the Python implementation.

References

(2022). The Cancer Genome Atlas Program (TCGA) - NCI. Archive Location: nciglobal,ncienterprise.

Dudoit, S. and Fridlyand, J. (2002). A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology*, 3(7):research0036.1.

Hartigan, J. A. and Hartigan, P. M. (1985). The Dip Test of Unimodality. *The Annals of Statistics*, 13(1):70–84. Publisher: Institute of Mathematical Statistics.

Hodson, R. (2016). Precision medicine. *Nature*, 537(7619):S49–S49. Publisher: Nature Publishing Group.

John, C. R., Watson, D., Barnes, M. R., Pitzalis, C., and Lewis, M. J. (2020). Spectrum: fast density-aware spectral clustering for single and multi-omic data. *Bioinformatics (Oxford, England)*, 36(4):1159–1166.

John, C. R., Watson, D., Lewis, M., Russ, D., Goldmann, K., Ehrenstein, M., Pitzalis, C., and Barnes, M. (2018). M3C: A Monte Carlo reference-based consensus clustering algorithm. Pages: 377002 Section: New Results.

Kosorok, M. R. and Laber, E. B. (2019). Precision Medicine. *Annual Review of Statistics and Its Application*, 6(Volume 6, 2019):263–286. Publisher: Annual Reviews.

Nguyen, H., Shrestha, S., Draghici, S., and Nguyen, T. (2019). PINSPlus: a tool for tumor subtype discovery in integrated genomic data. *Bioinformatics (Oxford, England)*, 35(16):2843–2846.

Omran, M. G., Engelbrecht, A. P., and Salman, A. (2007). An overview of clustering methods. *Intelligent Data Analysis*, 11(6):583–605. Publisher: SAGE Publications.

Ramazzotti, D., Lal, A., Wang, B., Batzoglou, S., and Sidow, A. (2018). Multi-omic tumor data reveal diversity of molecular mechanisms that correlate with survival. *Nature Communications*, 9(1):4453. Publisher: Nature Publishing Group.

Shen, R., Olshen, A. B., and Ladanyi, M. (2009). Integrative clustering of multiple genomic data types using a joint latent variable model with application to breast and lung cancer subtype analysis. *Bioinformatics*, 25(22):2906–2912.

Shu, L. and Latecki, L. J. (2016). Integration of Single-view Graphs with Diffusion of Tensor Product Graphs for Multi-view Spectral Clustering. In *Asian Conference on Machine Learning*, pages 362–377. PMLR. ISSN: 1938-7228.

Zelnik-manor, L. and Perona, P. (2004). Self-Tuning Spectral Clustering. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press.

Zhang, X., Li, J., and Yu, H. (2011). Local density adaptive similarity measurement for spectral clustering. *Pattern Recognition Letters*, 32(2):352–358.

A Algorithm Pseudo-Code

Algorithm 1 Adaptive Kernel Selection via Dip Test

Input: Dataset with N samples
Output: `best_A_star`
Initialize: `Kernels`, `D_min`, `A_stars` $\leftarrow [\,]$, `max_K` $\leftarrow N$

for $p = 1$ to 10 **do**
    `A_star` $\leftarrow$ `calculate_ADA_SM(dataset,` p`)`
    Append `A_star` to `A_stars`
    $L \leftarrow$ `calculate_Laplacian(A_star,` N`)`
    $(\lambda, V) \leftarrow$ `eig(`L`)` ▷ Eigen-decomposition

    Initialize $Z_p \leftarrow$ `zeros(`N`)`
    for $i = 1$ to N **do**
        $Z_p[i] \leftarrow$ `dipstat(`$V[:, i]$`)`
    end for

    Initialize $D_p \leftarrow$ `zeros(`$N - 1$`)`
    for $i = 1$ to `max_K` $- 2$ **do**
        $D_p[i] \leftarrow Z_p[i+1] - Z_p[i+2]$
    end for

    Append $\min(D_p)$ to `D_min`
end for

`best_p` $\leftarrow$ argmin(`D_min`) ▷ Index of minimum value
`best_A_star` $\leftarrow$ `A_stars[best_p]`

Algorithm 2 Finding the Last Substantial Multimodality Gap

Input: D_k, `max_K`
Output: $k^\star$
Initialize: `counter` $\leftarrow 1$, $d_{\min} \leftarrow D_k[1]$, $f \leftarrow 2$, $c_{\max} \leftarrow 7$

for $i = 1$ to `max_K` $- 2$ **do**
    if $\frac{d_{\min}}{f} > D_k[i]$ **then**
        $d_{\min} \leftarrow D_k[i]$
    end if

    if `counter` $> c_{\max}$ **then**
        break
    end if

    `counter` $\leftarrow$ `counter` $+ 1$
end for

$k^\star \leftarrow$ `index of` $d_{\min}$ `in` $D_k - 2$

Algorithm 3 Sparse Similarity Matrix Construction and Diffusion

Input: Dataset with N samples, A^*
Output: Updated A^*
$D \leftarrow$ `calculate_distance_matrix(dataset,` N`)`
$Z \leftarrow 10$
Initialize `sparse_A_star` $\leftarrow$ `zeros(`N, N`)`

for $i = 1$ to N **do**
    `closest_j` $\leftarrow Z + 1$ smallest values in $D[i]$
    `closest_j_index` $\leftarrow$ corresponding indices, excluding i

    for $j = 1$ to N **do**
        if $j \in$ `closest_j_index` **then**
            `sparse_A_star`$[i, j] \leftarrow A^*[i, j]$
        end if
    end for
end for

Row-normalize `sparse_A_star` so each row sums to 1
$I \leftarrow$ identity matrix of size N
$Q \leftarrow$ `sparse_A_star`

for $t = 1$ to 4 **do** ▷ Diffusion iterations
    $Q \leftarrow$ `sparse_A_star` $\cdot\, Q \cdot$ `sparse_A_star`$^\top + I$
end for

$A^* \leftarrow Q^\top$