Spectral Clustering Eigenvectors and Outliers

Daniel Jiménez

Abstract

The task of dividing data into multiple groups has always been a challenge and there are many grouping algorithms that help us with this problem. Spectral clustering becomes important with data that other traditional clustering algorithms such as k-means cannot correctly classify, such as non-convex data sets. In this paper we review basic concepts of spectral grouping such as how eigenvectors work, explaining the eigengap heuristic for the selection of the number of groups and focusing on the eigenvectors for further processing in k-means analyzing outliers.

1 Introduction

Clustering is the task of dividing a set of data points into two or more groups. Data points that belong to the same group are more similar to other data points in the same group than to other groups. The goal is to assign unlabeled data to groups, where similar data points are assigned to the same group.

Spectral clustering is a clustering technique based on graph theory. The algorithm consists of transforming a data set into a graph and then perform cuts over it to split the graph into clusters with similar data points.

2 Other clustering algorithms

2.1 K-means clustering algorithm

K-means is one of the best known clustering algorithms [Ortega et al. [2019]]. It is easy to implement and performs well on convex data sets. The only input this algorithm needs is the number of k groups to divide the data set. Once the number of groups is selected, we add k initial centroids to random positions in the data set. These centroids will function as cluster centers. Next, we calculate the distances between each data point to each centroid and assign each data point to its closest centroid. After all data points have an assigned centroid, we calculate the mean of each group and move the centroid to this mean. We repeat the process of calculating the distances between all data points and centroids until all centroids remain fixed in the same position (the mean of all data points in all groups remains the same). The last step in the iteration is to take the variance of all clusters and save it. We repeat this process from the beginning selecting new random centroids. After max_iter iterations, the best solution is the solution whose centroids have the best variance distribution [Figure 1].

There are slight modifications of this algorithm such as k-means++ which is the one sklearn uses by default. In our paper we will use this version [Arthur and Vassilvitskii [2007]].

2.2 DBSCAN

DBSCAN (Density Based Spatial Clustering of Applications with Noise) is another clustering technique that is more closely related to spectral clustering [Ester et al. [1996]]. It needs two values as input, the radious limit of each point called eps and the $min\_samples$. Starting with a random point, create a circle (boundary) of radius eps around the point. If there are at least $min\_samples$ points within the boundary, then a group is created. This data point is now considered as a center

4th TUML Seminar (2021/22).

Figure 1: The data set is divided into three groups. Each centroid is drawn with a black triangle.

point. All data points that are inside the boundary are now edge points. For each of these edge points, we're going to create new boundaries, and all edge points that have at least $min\_samples$ data points inside are now considered center points. We repeat the same process for these new data points. After the cluster is created and no more data points are added, we restart the process on a data point that does not belong to any cluster. If this data point does not have at least $min\_samples$ within its limit, it will be a noise point and will not have a cluster. Go through this process until all points belong to a boundary or are noise points [Figure 2].

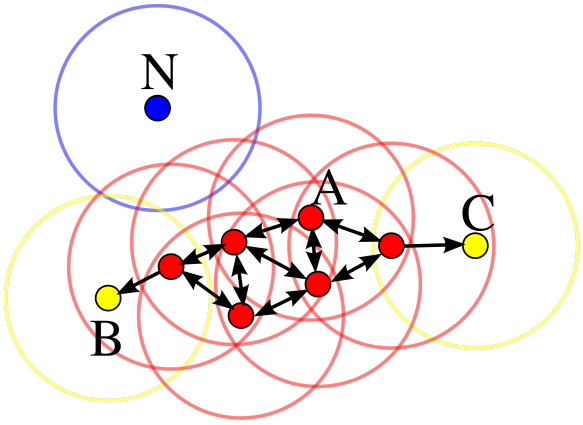

Figure 2: Consider $min\_samples = 4$. Red dots are considered center dots because there are at least other $min\_samples$ dots within their boundary. The yellow points (B and C) are considered edge points because they are on the boundary of a center point, but they do not have $min\_samples$ on their boundary. The blue point is a noise point, in its limit there are no $min\_samples$ or more data points. Image from Wikipedia.

DBSCAN, along with Spectral Clustering, works well in convex and non-convex sets because these algorithms do not make strong assumptions about the shape of the cluster. A convex set is a subset that intersects every line into a single line segment [Figure 3]. However, its performance decreases with clusters that have different density, since those with higher density will need a smaller eps in contrast to the lower density clusters.

3 Spectral Clustering

As we have seen in the previous section, we can divide our data set into groups using only our raw data set. However, the way spectral clustering works is slightly different:

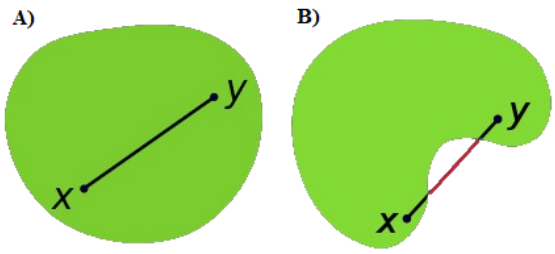

Figure 3: A) Convex set. B) Non-convex set. Image from Wikipedia.

1. Construct the graph Laplacian matrix of the data set using the weighted adjacency matrix and the degree matrix. The graph Laplacian matrix maintains the relationships between the data points in the data set.
2. Calculate eigenvectors and eigenvalues of the graph Laplacian matrix to create the new data set.
3. Once we have the new data set made up of k eigenvectors, we divide it into several groups using another clustering algorithm.

3.1 Similarity graph

The first part of the Spectral Clustering algorithm consists of obtaining the similarity graph. Consider a set of data points $x_1, ..., x_n$ and a function s, that maps for each pair of data points the similarity between them. The weighted adjacency matrix of the graph is the matrix $W = (w_{ij})_{i,j=1,...,n}$. $w_{ij} = 0$ means that there is no edge between the vertices v_i and v_j. As G is undirected $w_{ij} = w_{ji}$. The degree of a vertex v_i is defined as

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

The degree matrix D is the diagonal matrix with the degrees $d_1, ..., d_n$ on the diagonal.

For the similarity graph, each vertex v_i in the graph represents a data point x_i, and two vertices v_i and v_j are connected if the similarity s_{ij} between them is, for example, larger than a certain threshold. There are several popular constructions to transform a given set x_1,...,x_n of data points into a similarity graph.

- **The ϵ-neighborhood graph**: two vertices v_i and v_j are connected if the distance between the data points x_i and x_j is smaller than a certain threshold ϵ. Features in the data set should be scaled before computing the distances. The ϵ-neighborhood graph is usually considered as an unweighted graph.
- **The k-nearest neighbor graph**: two vertices v_i and v_j are connected if v_j is among the k-nearest neighbors of v_i. However, this works for a directed graph, but there is a problem when v_j is among the k-nearest neighbors of v_i but not the other way around. In this case we have two options: create an edge if one of both vertices is in the k-nearest neighbors of the other (k-nearest neighbor graph) or consider only an edge between them when only both vertices are in the k-nearest neighbors of the other vertex (mutual k-nearest neighbor graph). In both cases, after connecting the appropriate vertices we weight the edges by the similarity of their data points x_i and x_j.
- **The fully connected graph**: there are edges between all data points with positive similarity with each other, and we weight all edges by s_{ij}. As the graph should represent the local neighborhood relationships, this construction is only useful if the similarity function itself models local neighborhoods. An example for such a similarity function is the Gaussian similarity function $s(x_i, x_j) = exp(-||x_i - x_j||^2/2\sigma^2)$, where the parameter σ controls the width of the neighborhoods.

3.2 Graph Laplacians

After we have our similarity graph and thus the weighted adjacency matrix, we can construct the graph Laplacian matrix. We will use the unnormalized graph Laplacian matrix for our implementation of spectral clustering. All three are explained in more detail in von Luxburg [2007].

The unnormalized graph Laplacian matrix is defined as

$$L = D - W$$

It will have the degree of all vertices on the diagonal and the other positions will be filled with the negative weight of all edges of the graph.

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

1. *The smallest eigenvalue of L is 0.*
2. *For every vector $f \in R^n$ we have*

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

3. *L has n non-negative, real-valued eigenvalues 0 = $\lambda^{(1)} \leq \lambda^{(2)} \leq ... \leq \lambda^{(n)}$.*

Proof.
Part (1): The determinant of the graph Laplacian matrix L is 0, since each value of the diagonal is the negative sum of the rest values of the column. Due to the determinant is 0, the matrix becomes singular and every singular matrix has at least one 0 eigenvalue.

Part (2): Since the matrix has at least one eigenvalue, we can calculate its eigenvector using

$$Lf = \lambda f \rightarrow f^T L f = f^T \lambda f$$

Because the diagonal matrix has 0 in all positions except in the diagonal

$$f^T L f = f^T D f - f^T W f = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij} =$$

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

Part (3) is a direct consequence of Parts (1) - (2). □

Proposition 2. *(Number of connected clusters and the spectrum of L) Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected clusters $A^{(1)}$, ..., $A^{(k)}$ in the graph. The eigenspace of eigenvalue 0 is spanned by the eigenvectors of those clusters.*

Proof. The first case is with $k = 1$, which means that the graph is connected. Suppose f is an eigenvector with eigenvalue 0.

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

Since all weights w_{ij} are non-negative and $(f_i - f_j)^2$ is always positive, to make the function vanish we have to make the part $w_{ij}(f_i - f_j)^2$ equals 0, so, f_i has to be equal to f_j when w_{ij} is greater than 0, that is when two data points are connected by an edge. If the graph only has one cluster, i.e. all the data points belong to the same group, then we have a path between all the data points on the

graph. In this case, f has to be a constant vector (f_i has to be the same for all data points) to make the subtraction of f_i and f_j always 0.

Consider the general case of k clusters on the graph. We will assume that the data points are ordered according to the clusters they belong to and that there are no connections between data points of different clusters. In this case, the adjacency matrix W has the form of a diagonal block, it will have positive weights in the block of the cluster and 0 in the other blocks. The same for the graph Laplacian matrix L, which is compose by Laplacian blocks $L^{(i)}$.

In this general case, we know that the eigenvalues and eigenvectors of L are the union of the eigenvalues and eigenvectors of all $L^{(i)}$, padded with 0 at the positions of the other blocks. Therefore, each block $L^{(i)}$ will have an eigenvalue of 0 since the block itself is also a singular matrix, and f_i and f_j will be the same for all data points belonging to that block, so the eigenvector will have a constant value at all data points in the block and 0 at data points that do not belong to the block. The matrix L has as many eigenvalues 0 as connected clusters, and the corresponding eigenvectors are the indicator vectors of the connected clusters. □

We assume that eigenvectors are always sorted by their eigenvalue in ascending order, so the first eigenvalue will always be 0 and the first eigenvector is the eigenvector that corresponds to the first eigenvalue.

3.3 The "ideal" case

Let's see the above propositions in more detail. We are going to start with the "ideal case", in which all data points are completely separated in data space, i.e. data points belonging to one cluster have no edges connecting them to data points in another cluster. Consider that the data set contains the data points ordered by cluster and in total has k clusters. The first group of data points corresponds to cluster $k^{(1)}$, the second group of data points corresponds to cluster $k^{(2)}$, and so on. The adjacency matrix will be composed of blocks of clusters, in which the data points of these clusters will be connected and any other position will be 0, because the data points of one cluster have no connection with the data points of another other cluster. Each block of the adjacency matrix can be interpreted as an adjacency matrix of each group.

$$W = \begin{bmatrix} W^{(1)} & 0 & ... & 0 \\ 0 & W^{(2)} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & W^{(k)} \end{bmatrix}$$

If we compute the graph Laplacian matrix L we will obtain the same result due to the symmetry of the adjacency and degree matrices.

$$L = D - W = \begin{bmatrix} L^{(1)} & 0 & ... & 0 \\ 0 & L^{(2)} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & L^{(k)} \end{bmatrix}$$

To compute the eigenvalues and eigenvectors of L we have to solve the following equation: $Lf = \lambda f$. All the vectors that satisfy this equation would be eigenvectors of L and λ would be the scalar value of the eigenvector f.

Since L is block diagonal, its eigenvalues and eigenvectors are the union of the eigenvalues and eigenvectors of its blocks. Also, because L is singular, the blocks are also singular, so it means that they have also an eigenvalue 0. Consider the first block of $L^{(1)}$, and a vector $v^{(1)}$ for which $L^{(1)}v^{(1)} = 0$. $v^{(1)}$ is an eigenvector of $L^{(1)}$ with eigenvalue 0 since $L^{(1)}v^{(1)} = 0 = \lambda^{(1)}v^{(1)}$. Also, as we have seen before, to make the eigenvalue 0, all components of the eigenvector $v^{(1)}$ have to be constant to make $W_{ij}^{(1)}(v_i^{(1)} - v_j^{(1)})^2$ vanish. We can reproduce this process to the entire graph Laplacian matrix. Given a vector $v^{(i)}$ that satisfies $L^{(i)}v^{(i)} = 0$, we will have at least k eigenvalues 0

with an eigenvector associated to each one that will be padded with zeros in the positions that do not correspond to its block $L^{(i)}$.

$$\begin{bmatrix} L^{(1)} & 0 & ... & 0 \\ 0 & L^{(2)} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & L^{(k)} \end{bmatrix} \begin{bmatrix} v^{(1)} \\ 0 \\ ... \\ 0 \end{bmatrix} = 0 = \lambda^{(1)} f^{(1)} = 0 \begin{bmatrix} v^{(1)} \\ 0 \\ ... \\ 0 \end{bmatrix}$$

The number of zero-eigenvalues determines how many different clusters we have in our data set [Figure 4], and their associated eigenvectors will be the new data set to cluster. However, this is the ideal case, in which all clusters are completely separated from the other ones. When we work with real data all the clusters are not completely separated and eigenvalues and eigenvectors become less clear.

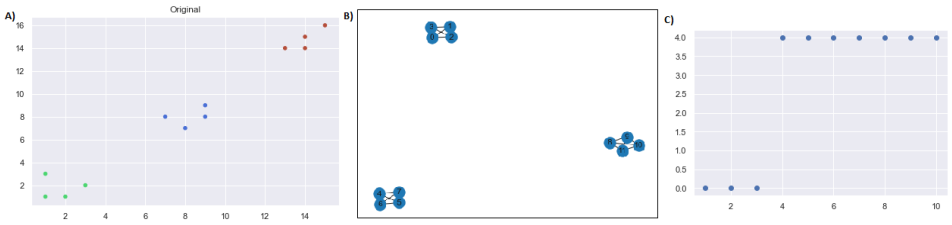

Figure 4: A) Data set in the plane. B) Similarity graph. C) Eigenvalues. As we can see the first three are 0.

When we connect the clusters [Figure 5], the eigenvalues of each cluster that were 0 are no longer 0 but close to 0. Furthermore, the eigenvectors associated with these eigenvalues will have a more or less constant part in the part of the vector that corresponds to the cluster block $L^{(i)}$ and smaller values in the parts that correspond to other blocks. We have to find other ways to know the value k, that is, how many clusters there are in the data set and the eigenvectors that represent them.

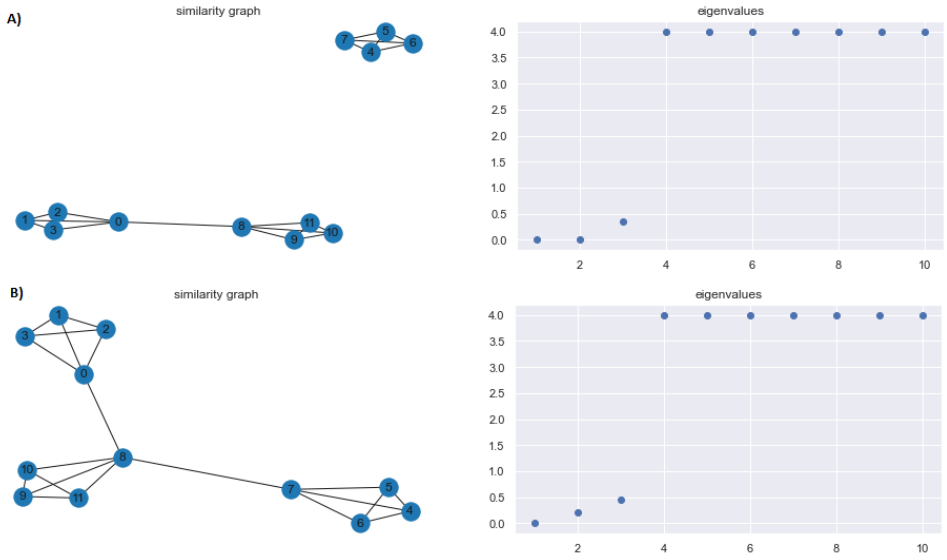

Figure 5: A) We connect the points 0-8. B) We connect the points 0-8 and 7-8.

3.4 Eigengap heuristic

Here we explain one tool which is particularly useful for Spectral clustering, which is called the eigengap heuristic. It is based on the eigenvalues and the goal is to know how many clusters there are

in the data set and how many eigenvectors to pick to create these clusters. We will start sorting the eigenvalues in ascending order. The first eigenvalue will be $\lambda^{(1)}$, which is always 0, and we will stop when the difference between $\lambda^{(k+1)} - \lambda^{(k)}$ becomes relatively larger. In the above example we can notify that the eigengap is produced between $\lambda^{(3)}$ and $\lambda^{(4)}$, so we will have three clusters and we will need the first three eigenvectors to calculate them.

However, the more noisy or overlapping the clusters are, the less effective is this heuristic. The eigengap heuristic usually works well if the data contains very well pronounced clusters, but in ambiguous cases it also returns ambiguous results.

Let's take a look at a more realistic example, with well-defined clusters. Since we have the correct groups for this data set, we are going to sort all the data points by group to better see how the eigenvectors are composed. Also, because the adjacency matrix is not perfect, i.e., there are connections between data points of one cluster to data points of other clusters, we are going to remove all edges that are not between the data points of the same group [Figure 6].

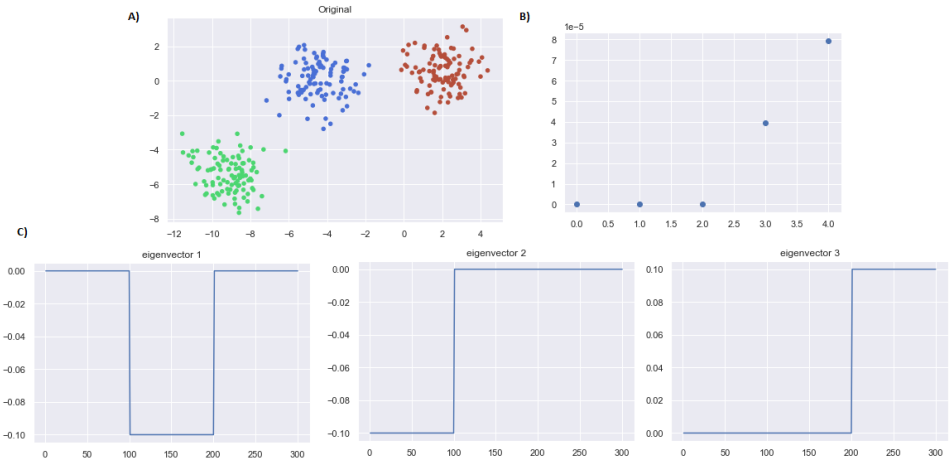

Figure 6: A) Data set of 300 data points. B) Eigenvalues. The first three eigenvalues are 0. C) First three eigenvectors.

If we compute the graph Laplacian matrix L of this example, it gives us the perfect blocks as in the small example of above. The first three eigenvalues are 0, so as we calculated before, the first three eigenvectors should have all zeros except in the Laplacian block of its cluster. The first eigenvector corresponds to $f^{(1)}$ and has zeros in all positions except for the data points of its block, in which has constant values -0.1 (it could be any other value).

$$\begin{bmatrix} L^{(1)} & 0 & 0 \\ 0 & L^{(2)} & 0 \\ 0 & 0 & L^{(3)} \end{bmatrix} \begin{bmatrix} 0 \\ \dots \\ 0 \\ -0.1 \\ \dots \\ -0.1 \\ 0 \\ \dots \\ 0 \end{bmatrix} = 0 = \lambda^{(1)} f^{(1)} = 0 \begin{bmatrix} 0 \\ v^{(1)} \\ 0 \end{bmatrix}$$

Each value of the eigenvector corresponds to one data point, so if we want to classify the data set into two clusters, the eigenvector 1 gives us information of how to split one cluster from the other two. We can made a cut in -0.05, and everything on top will form one cluster, and everything on bottom will from another. The data points from 0 to 99 will be on the same cluster as data points from 200 to 299. The other cluster will be composed of data points from 100 to 199.

But, in this case, we want to classify into three groups as we see with the eigengap heuristic. There are some approaches like the one explained in the paper Charles J. Alpert [1995] that say that the

more eigenvectors, the better it works, however, as we have tried, it does not work well in all data sets. In this case, we will continue to use the same number of eigenvectors as clusters. To do that, we create a new data set of points in which each column will be the selected eigenvectors, and we supply this data set to the k-means algorithm. The new data set for this example has shape (300, 3).

3.5 Clustering

The last step is to pool the new data set. To do this, we provide this data set as input to the k-means++ algorithm. Each data point in the output will correspond to the data point in the original data set.

4 Outliers in eigenvectors

This last example was the "ideal case", however, when there are edges between vertices of different clusters the eigenvectors become more unclear. The first eigenvector classifies all three groups well, however, since we have a point that is an outlier, if we compute the kmeans++ algorithm with only this eigenvector, it will give us a centroid only for that point and the other two centroids will be distributed among the three groups [Figure 7].

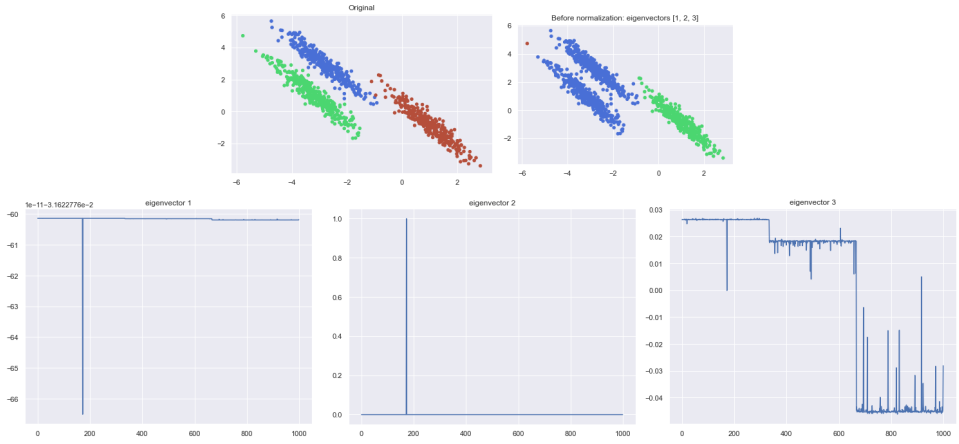

Figure 7: A) Data set of 1000 data points. B) The first eigenvalue is 0 and the next two are too close to 0. C) The first three eigenvectors.

Related literature [Yang and Huang [2008], Ina et al. [2017], Wang et al. [2016]] uses spectral clustering to find outliers in the main data set by using eigenvectors of the graph Laplacian matrix.

K-means++ varies from its original version in the way it selects the initial groups. The only input you need is the number of clusters, the same as the original version. First, we have to choose a centroid among all the data points with uniform probability. For each point, find the distance $d(x)$ between the point and the nearest centroid. In the first iteration there is only one centroid. Next, consider a random point to be a new centroid using a distributed weighted probability where the chance of picking a point is the probability proportional to the square of the distance to the nearest centroid $d(x)^2$. Repeat these steps until k groups have been selected. Now, we continue with the original k-means.

Consider the data set generated by the first eigenvector of some data set. We have n data points with only one feature, shape is $(n, 1)$, and there is only one outlier. The first centroid will be placed randomly with uniform probability among all data points, so the probability of creating the centroid in the outlier is $\frac{1}{n}$ which is too lower in a data set with a high value of n. We assume that the first centroid is around the mean of all data points without counting outliers. So the squared distance of each non-atypical data point will roughly be std^2. The second centroid is placed accordingly with the squared distance, so the probability of picking the outlier as a centroid is close to

$$\frac{(r \cdot std)^2}{(n-1)std^2 + (r \cdot std)^2}$$

where r is the number of times the outlier multiplies the standard deviation. The further the outlier is or as many outliers are in the eigenvector, the greater the probability. For this problem we propose the normalization of the eigenvectors. We consider an outlier a value that is at least twice the standard deviation of the data set. We calculate the maximum and minimum limit value of the data set taking into account the outliers and set these outliers to the maximum or minimum.

$$x_i = \begin{cases} mean(x) + 2 * std(x), & \text{if } x_i > mean(x) + 2 * std(x), \\ mean(x) - 2 * std(x), & \text{if } x_i < mean(x) - 2 * std(x), \\ x_i, & \text{otherwise} \end{cases} \tag{1}$$

However, since there are eigenvectors that have so larger outliers, we repeat this operation until no more outliers are found in the eigenvector, calculating in each iteration the mean and standard deviation of the new vector [Figure 8].

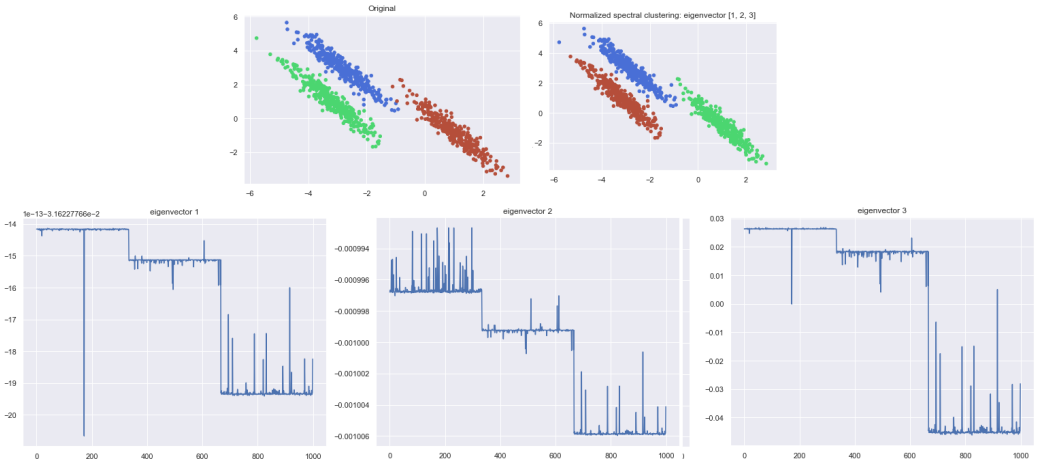

Figure 8: Normalized eigenvectors of the previous data set.

Once the eigenvectors are normalized, we run the k-means++ algorithm on the new data set of selected eigenvectors. We will have classified all the data points by the eigenvectors that correspond to the data points in the original data set.

5 Results

The algorithm has been tested on six different samples together with sklearn DBSCAN and sklearn spectral clustering. The number of groups is handwritten both in our algorithm and in the sklearn spectral group. The eps value for DBSCAN has been selected empirically and the one chosen in each sample (each sample has a different eps) is in the range of the most optimal eps for that sample. The $min\_samples$ value is set to 10 for DBSCAN. For our algorithm, the number of eigenvectors selected is the same as the number of groups in the sample. We have used the unnormalized graph Laplacian matrix and to correct for outliers in the eigenvectors the limit is set to twice the standard deviation. The graph Laplacian matrix has been constructed with the fully connected graph method and $\sigma = 0.1$. The value of σ has been selected empirically.

As we can see from the results [Figure 9], our spectral clustering algorithm performs slightly better on every sample (except the last one) than sklearn spectral clustering. Also, DBSCAN works quite well in most samples, but as mentioned in the introduction, its performance decreases when the clusters have different density as in the last sample.

6 Conclusion

We conclude that spectral clustering is an algorithm that does a great job on convex and non-convex sets, however it is highly dependent on the adjacency matrix.

Historically, k-means has been used as the common solver for all kinds of problems, but there are other algorithms that work much better depending on the data set. Based on our results, we recommend the use of DBSCAN for data sets with a similar density of data points throughout the entire data set, since that algorithm is easier to use and spectral clustering as an alternative when the data set differs in density.

References

David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

So-Zen Yao Charles J. Alpert. Spectral partitioning: The more eigenvectors, the better. In *32nd Design Automation Conference*, pages 195–200, 1995. doi: 10.1109/DAC.1995.250089.

F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

Takuro Ina, Atsushi Hashimoto, Masaaki Iiyama, Hidekazu Kasahara, Mikihiko Mori, and Michihiko Minoh. Outlier cluster formation in spectral clustering. *CoRR*, abs/1703.01028, 2017. URL http://arxiv.org/abs/1703.01028.

Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002. URL https://proceedings.neurips.cc/paper/2001/file/801272ee79cfde7fa5960571fee36b9b-Paper.pdf.

Joaquín Ortega, Nelva Almanza-Ortega, Andrea Vega-Villalobos, Rodolfo Pazos-Rangel, José Crispin Zavala-Diaz, and Alicia Martínez-Rebollar. *The K-Means Algorithm Evolution*. 04 2019. ISBN 978-1-83880-333-9. doi: 10.5772/intechopen.85447.

Ulrike von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007. URL http://arxiv.org/abs/0711.0189.

Yuan Wang, Xiaochun Wang, and Xia Wang. *A Spectral Clustering Based Outlier Detection Technique*, volume 9729, pages 15–27. 01 2016. ISBN 978-3-319-41919-0. doi: 10.1007/978-3-319-41920-6_2.

Peng Yang and Biao Huang. A spectral clustering algorithm for outlier detection. In *2008 International Seminar on Future Information Technology and Management Engineering*, pages 33–36, 2008. doi: 10.1109/FITME.2008.120.

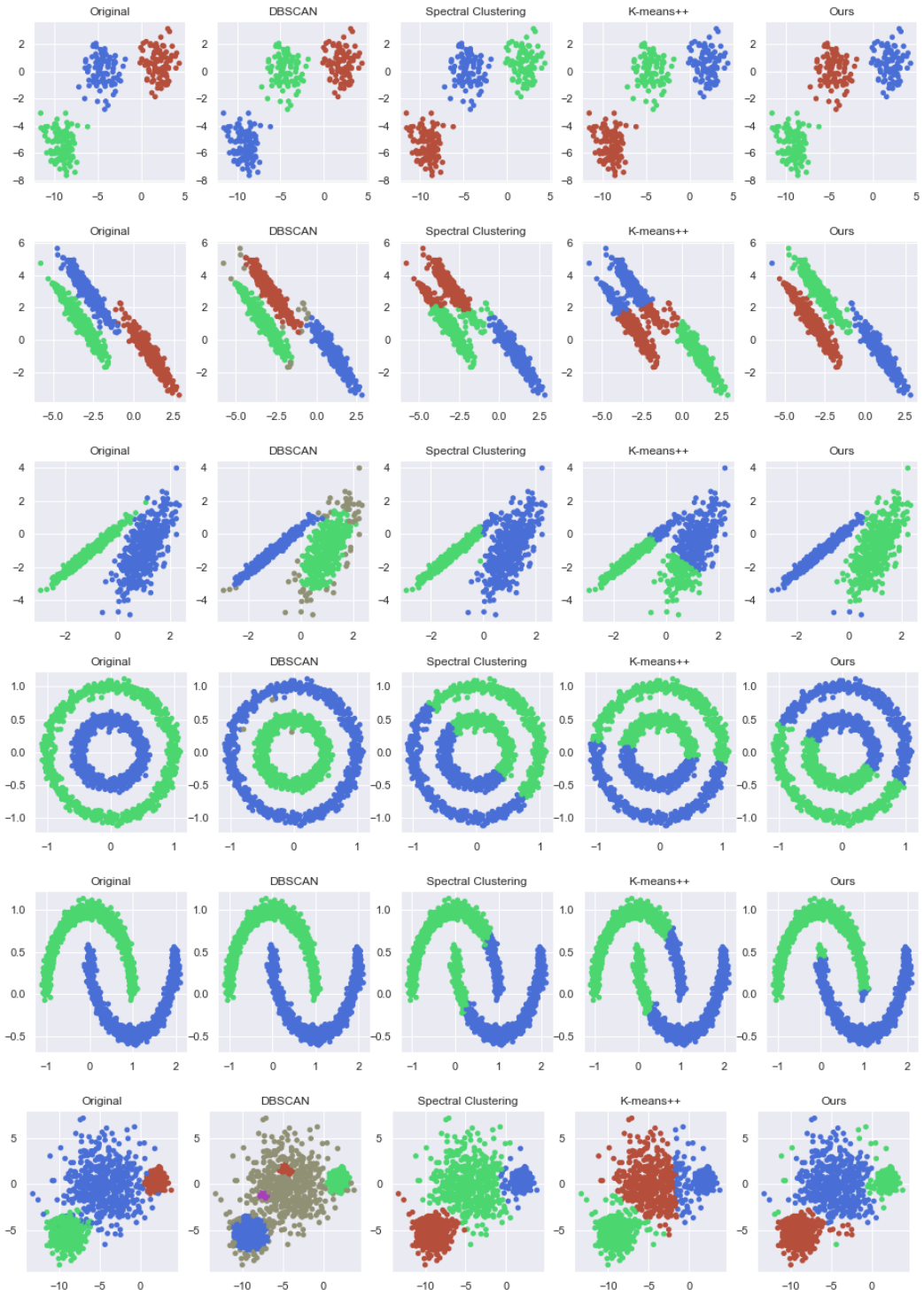

Figure 9: Results.