Spectral Clustering Tutorial

Gargee Jagtap, Sarah Saas, David Siamon, Nadir Siddiqui

DS 6030 | Fall 2022 | University of Virginia

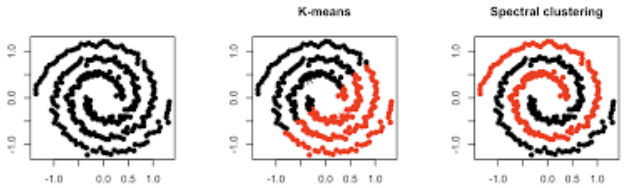
Table of Contents

Introduction

Clustering is a type of unsupervised learning algorithm that allows data points to be grouped into “clusters”. These clusters group data points into these clusters so that all data points in the cluster are similar in some way and dissimilar to the points in the other clusters. Clustering is useful because it has predictive power and objects can be grouped which is useful for naming them for communication. There are several common forms of clustering including; k-means, mean-shift, density-based spatial clustering of applications with noise

(DBSCAN), expectation maximum (EM), and spectral clustering. This tutorial will focus on spectral clustering.

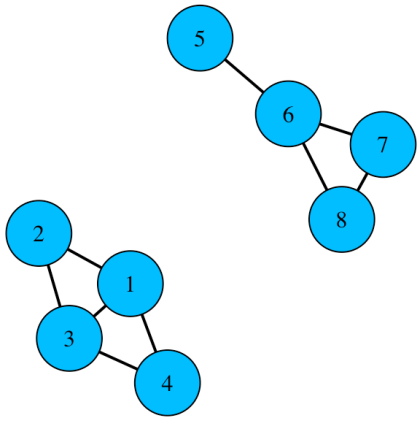
Spectral clustering is a popular clustering technique due to its ease of implementation and its ability to outperform other methods (such as the k-means algorithm). It makes no assumptions about the shape of the clusters and can use non-graphical data. Spectral clustering uses graph theory to use the edges to determine communities of nodes. This technique also differs from k-means and common algorithms because it uses connectivity instead of compactness like most algorithms. Compactness uses the distance between points to group them, meaning points closest together will be grouped. This often means that points are clustered around a geographic center.



When using connectivity, like in spectral clustering, points may be close together but not in the same cluster. This means that points can result in clusters that are shaped like spirals or wandering lines. On the other hand, since spectral clustering uses connectivity, it is computationally more expensive compared to other methods. Meaning this method should be used for small datasets with high dimensionality compared to large datasets with many observations. In addition to being computationally taxing, larger datasets also tend to decrease the accuracy of clustering in spectral clustering.

Mathematical Methods

We should define some basic graph theory terms. A graph $G = (V, E)$ consists of a set of vertices (nodes), and edges which connect the nodes. We can define our vertex set $V = \{v_1, \dots, v_n\}$, so each v_i represents a data point, or node $(\forall i \in 1, \dots, n)$. Let's look at the following graph as an example:



Our graph has 8 vertices and 9 edges. In this case, the edges are undirected and have no weights associated with them. Two nodes are adjacent if they are connected by an edge. The neighborhood of a vertex is the set of vertices that are adjacent to it. So, the neighborhood of vertex 1 is $\{2, 3, 4\}$, the neighborhood of vertex 7 is $\{6, 8\}$. The degree of a vertex is how many edges are incident (connected) to it. Since our graph is undirected, we can assume the in-degree equals the out-degree. In this example, the degree of each vertex in-order is: $\{3, 2, 3, 2, 1, 3, 2, 2\}$. We can construct a degree matrix where we have a row and column for each vertex, and the degree of each node is on the diagonal while the off-diagonals are zero. Another way to represent this graph is through an adjacency matrix. Adjacency matrices have rows and columns for each node and indicate if an edge is incident between two nodes (1 if yes, 0 if no). If our edges had weights, we would replace the 1 with the corresponding weight. The degree and adjacency matrix for this graph are as shown below, respectively:

$$\begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{1}$$

There appear to be two distinct sub-sections of the graph, these are called connected components. A connected component is a subgraph in which each pair of nodes is connected to another via a path (a finite sequence of edges which joins vertices). Since no node has an edge to itself, this graph is simply connected. Our graph contains two connected components: $\{1, 2, 3, 4\}$ and $\{5, 6, 7, 8\}$. When it comes to clustering, an immediate idea would be to define each connected component as its own cluster. However, what if our graph only consists of one connected component, i.e., the entire graph is connected? Or what if within each connected component, we want to define smaller, more defined clusters? Connected components are integral in spectral clustering, and we will expand on this further down.

Now we can begin our spectral clustering process. The algorithm can be outlined in four key steps:

I. Generate a similarity graph
II. Compute the Laplacian Matrix
III. Eigendecomposition
IV. Create clusters

I Similarity Graph

We start by computing a similarity graph $G = G(V, E)$. Let's assume our graph is undirected. We can represent this graph through an adjacency matrix composed of the similarities between nodes.

Computing the distance between two points is more intuitive for some problems than others. If we have a simple 2-D dataset with distance values on each axis, we can use the Euclidian distance formula $\sqrt{x^2 + y^2}$. However, for datasets with higher dimensionality, the distance between two points isn't as intuitive. Fortunately, the Euclidian distance formula has a dimension-generic version: $dist(x, y) = \sqrt{\sum_i (y_i - x_i)}$. We can apply this to numeric data of any dimension to compute the distance between two points.

We use the Euclidian distance formula when constructing our similarity matrix S. The formula for an entry in S is as follows:

$$s(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right) \tag{2}$$

Where x_i and x_j are the two data points, $||x_i - x_j||$ is their Euclidian distance, and sigma σ is a scaling constant to control for the width of the neighborhoods. For our example, since our edges aren't weighted, we will assume the similarity matrix is the same as the adjacency matrix.

II Laplacian Matrix

The next step is to compute the Laplacian Matrix, which we'll then perform eigendecomposition on. The motivation behind this is that the Laplacian reduces the dimensionality of our data, such that observations in the same cluster will be close to each other, while observations in different clusters will not.

The Laplacian matrix L is computed by taking the difference of the degree matrix D and the similarity matrix S. To calculate the degree $d(x)$ of our observations, we use the following formula:

$$d_i = \sum_{j|(i,j)\in E} w_{ij} \tag{3}$$

This formula differs from the one introduced above, because we are now dealing with weighted edges. Rather than simply computing the number of edges a node has, we now take the sum of the weights on all the edges. Our matrix D is the values of $d(x)$ along the diagonal, with all non-diagonal values set to 0.Once D is computed, we use it along with S: $L = D - S$.

The Laplacian matrix for our working example is given below:

$$\begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix} \tag{4}$$

III Eigendecomposition

Now we can obtain the eigenvectors and eigenvalues of the Laplacian matrix. Given our

matrix L, we can define an eigenvector x and its corresponding eigenvalue λ as any pair which satisfies the following:

$$Lx = \lambda x \tag{5}$$

This is known as eigendecomposition. For large matrices, these values are not computable by hand. That's why we use computers! Further below we'll see an example of eigendecomposition performed on a large matrix.

Once we have these, we can find the vectors associated with the k largest eigenvalues and form a new matrix from these. The number of eigenpairs that we want to keep is based on the number of clusters we want. If we don't know how many clusters we want, we sort our eigenvalues in non-decreasing order: $\lambda_1 \leq \lambda_2 \leq \ldots \lambda_n$. We then look for the initial point where the eigenvalues increase sharply, and only take only eigenvalues before that. The number of eigenvalues that are equal to zero is equal to the number of connected components in our graph. Therefore, the number of total eigenvectors we keep is equal to the number of clusters we want.

Let's go back to our working example. Keep in mind our graph consists of 8 vertices, therefore we will have 8 eigenvalues and 8 eigenvectors. The results of our eigendecomposition are shown below:

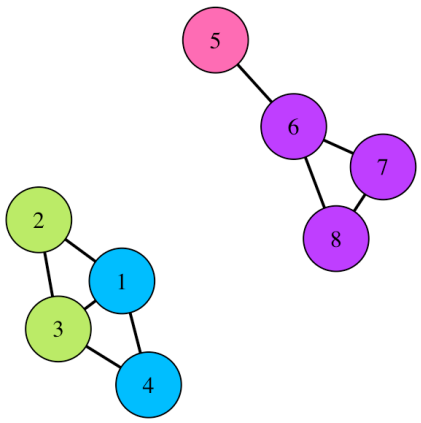
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & -2 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -2 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -3 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Each eigenvector is a column, corresponding to an eigenvalue. Our eigenvalues for our working example are: $\{0, 0, 1, 2, 3, 4, 4, 4\}$. Since our graph has two connected components, we have two eigenvalues equal to zero, the rest non-zero. If we wanted our graph to have k clusters $(k > 2)$, we would simply normalize our Laplacian matrix by $L_{norm} = D^{½} L D^{½}$ and redo the above steps.

IV Create clusters

Now the fun part! We have a set of eigenvectors, sorted by their corresponding eigenvalue in non-decreasing order. Suppose we've decided we want $k = 4$ clusters. We would then select the first four eigenvectors and put them through a clustering algorithm such as K-Means, which would output our cluster assignments! Below is the output when running K-means for the first four eigenvectors of our example: $\{0, 1, 1, 0, 3, 2, 2, 2\}$

And as a visual:



Note the graph has been subdivided into 4 sections, with nodes 1 and 3 arbitrarily assigned to one of their connected sections.

Spectral Clustering Coding Examples

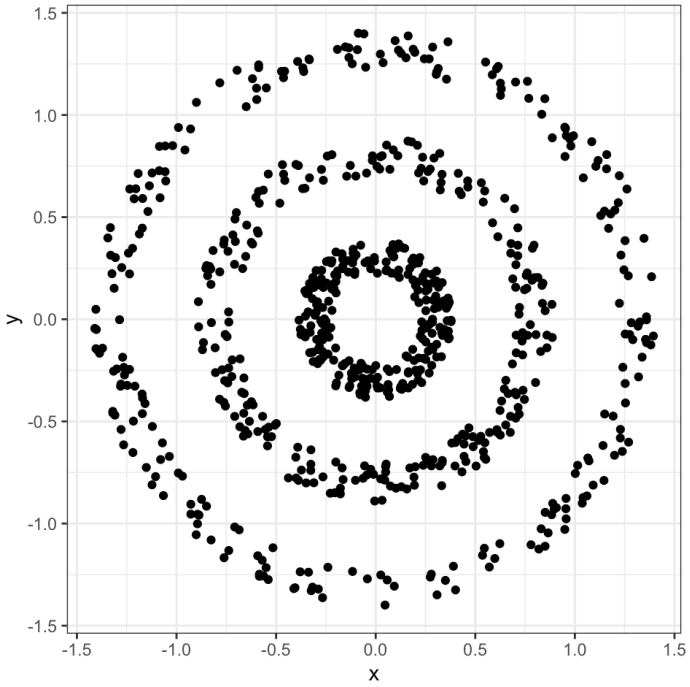
Using R

By this point we have laid the foundation for spectral clustering, but our working example wasn't really that complicated. Let's look at how to implement spectral clustering and compare the results to a k-means approach.

```
## Packages
library(tidyverse)
library(gridExtra)
library(ggplot2)
```

We first need to generate some data for clustering. We will generate three circular, uniform distributions and plot them:

```
# Generate circular uniform distributions of data
set.seed(1748)

## Set min and max parameters for each distribution
min_vals <- c(0.05, 0.5, 1.5)
max_vals <- c(0.15, 0.8, 2.0)
x <- c()
y <- c()

## Generate distributions using parameters
for(i in 1:3){
  r <- sqrt(runif(250, min = min_vals[i], max = max_vals[i]))
  p <- runif(250)
  theta <- p * 2 * pi
  x_temp <- r * cos(theta)
  y_temp <- r * sin(theta)
  x <- c(x, x_temp)
  y <- c(y, y_temp)
}

## Create data frame
data <- as.data.frame(cbind(x, y))

## Plot coordinates
ggplot(data=data, aes(x=x, y=y)) + geom_point()
```



Our goal is to cluster the data. By sight, there are clearly three rings representing the three distributions of data. We will cluster the data using k-means first:

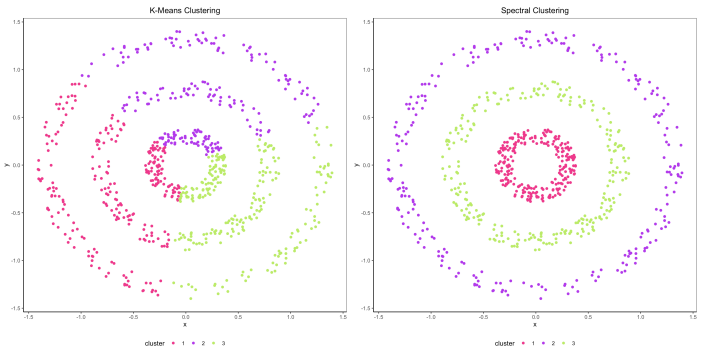
```
# Kmeans Application
set.seed(1748)

## Run Kmeans algorithm with 3 clusters
km <- kmeans(data, 3)


## Retrieve and store cluster assignments
kmeans_clusters <- km$cluster
kmeans_results <- cbind(data, cluster = as.factor(kmeans_clusters))
head(kmeans_results)
```

The output above represents the cluster each point is assigned to using k-means. We will now calculate the cluster assignments using spectral clustering:

```
# Spectral Clustering

## Create Similarity matrix of Euclidean distance between points
S <- as.matrix(dist(data))


## Create Degree matrix
D <- matrix(0, nrow=nrow(data), ncol = nrow(data)) # empty nxn matrix

for (i in 1:nrow(data)) {

      # Find top 10 nearest neighbors using Euclidean distance
      index <- order(S[i,])[2:11]

      # Assign value to neighbors
      D[i,][index] <- 1
}

# find mutual neighbors
D = D + t(D)
D[ D == 2 ] = 1

# find degrees of vertices
degrees = colSums(D)
n = nrow(D)


## Compute Laplacian matrix
# Since k > 2 clusters (3), we normalize the Laplacian matrix:
laplacian = ( diag(n) - diag(degrees^(-1/2)) %*% D %*% diag(degrees^(-1/2)) )


## Compute eigenvectors
eigenvectors = eigen(laplacian, symmetric = TRUE)
n = nrow(laplacian)
eigenvectors = eigenvectors$vectors[,(n - 2):(n - 1)]


set.seed(1748)
## Run Kmeans on eigenvectors
sc = kmeans(eigenvectors, 3)


## Pull clustering results
sc_results = cbind(data, cluster = as.factor(sc$cluster))
head(sc_results)
```

The output above represents the cluster each point is assigned to using spectral clustering. Let's now graph the data and use colors to represent the cluster assignment of both k-means and spectral clustering:

```
# Kmeans plot
kmeans_plot = ggplot(data = kmeans_results, aes(x=x, y=y, color = cluster)) +
  geom_point() +
  scale_color_manual(values = c('1' = "violetred2",
                                '2' ="darkorchid2",
                                '3' ="darkolivegreen2")) +
  ggtitle("K-Means Clustering") +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "blac
k"),
        plot.title = element_text(hjust = 0.5), legend.position="bottom")

# Spectral Clustering plot
sc_plot = ggplot(data = sc_results, aes(x=x, y=y, color = cluster)) +
  geom_point() +
  scale_color_manual(values = c('1' = "violetred2",
                                '2' ="darkorchid2",
                                '3' ="darkolivegreen2")) +
  ggtitle("Spectral Clustering") +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "blac
k"),
        plot.title = element_text(hjust = 0.5), legend.position="bottom")

# Arrange plots
grid.arrange(kmeans_plot, sc_plot, nrow=1)
```



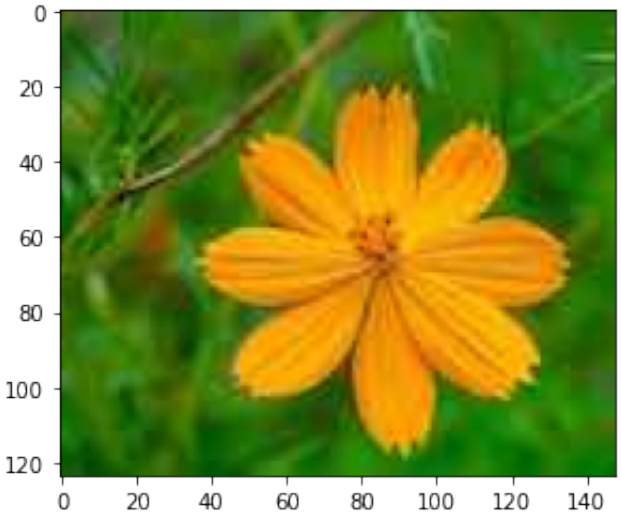
We have now learned how spectral clustering works and how to implement it with R. As an added bonus, we were able to visualize the advantage of using spectral clustering when compared to k-means clustering.

Using Python

We've also included an example of spectral clustering in Python for image segmentation:

In [26]:
```
import matplotlib.pyplot as plt
import matplotlib.image as img


flower = img.imread('yellowcopy.jpg')

# displaying the image
plt.imshow(flower)
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x7f8c37c39a60>
```



For this example we will be working with this image of a flower. Any image will work as long as it is small enough. For reference this image is around 100x80 pixels. This ran in a couple minutes. Images that are 500x500 are too large and will mostly likely not run due to the number of computations needed. This is why spectral clustering is better for smaller datasets or images. If you are following along with a different image, change the flower.jpg to match your file name of your image. Make sure the file is an image file (like jpg or png).

In [27]:
```
from sklearn.feature_extraction import image

matrix = image.img_to_graph(flower, mask=mask)
```

This chunk of code allows us to create a true or false (think 0 or 1 like the math walk-through) nested array based on our image. This is the array that will be used in the argument for turning our image into a graph object. This is needed in order to create the matrix for the spectral clustering.

In [28]:
```
matrix
```

Out[28]:
```
<50521x50521 sparse matrix of type '<class 'numpy.uint8'>'
    with 309691 stored elements in COOrdinate format>
```

In [ ]:
```
As you can see, just a 100x80 pixel image has produced a 28452x28452
matrix for spectral clustering. This is why large datasets are not good
for spectral clustering.
```

In [29]:
```
import numpy as numpy
```

In [ ]:
```
Since we are using numpy arrays and calculating matrices, we need to
import the numpy library.
```

In [30]:
```
matrix.data = np.exp(-graph.data / graph.data.std())
```

In [31]:
```
matrix.data
```

Out[31]:
```
array([ 1.34712325,  5.42874802,  1.30883241, ...,  4.02988222,
       11.48960838,  4.14777934])
```

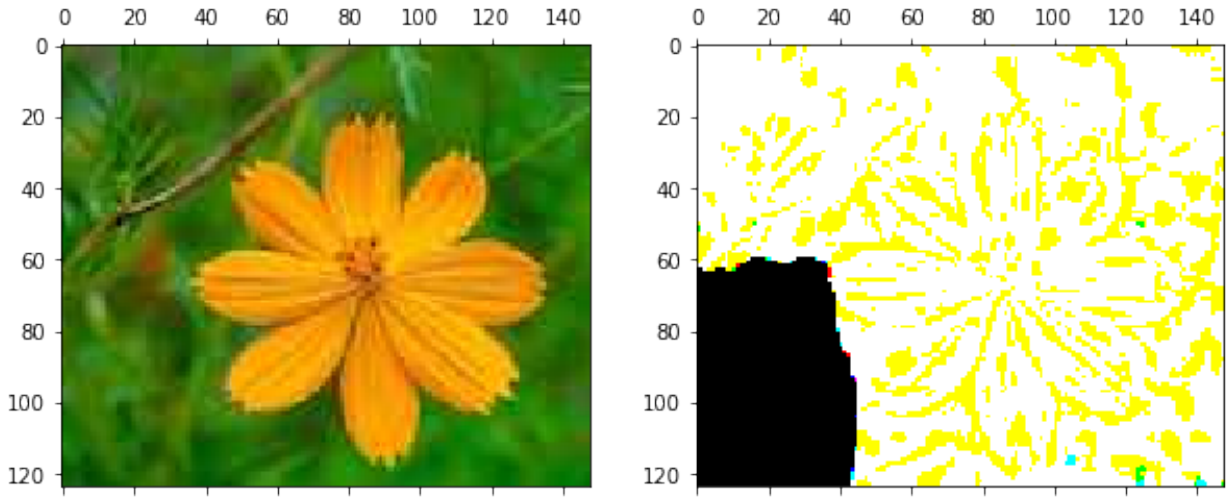
In [32]:
```
matrix.data.shape
```

Out[32]:
```
(309691,)
```

Like above, we create our similarity graph to allow us to compute our Laplacian. We then use eigen decomposition then K-Means to create our clusters.

In [34]:
```
from sklearn.cluster import spectral_clustering
import matplotlib.pyplot as plt

spectral = spectral_clustering(graph, n_clusters=5, eigen_solver="arpack"
sprectrum = np.full(mask.shape, -1.0)
sprectrum[mask] = spectral

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
axs[0].matshow(flower)
axs[1].matshow(sprectrum)

plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```



This is the result! It's important to note that most images are inherently noisy, making clustering a difficult problem. Nonetheless, we can see that spectral clustering has resulted in clear detection of the flower. We encourage readers to try this code out with other images!

Conclusion

We now have a basic understanding of the underlying graph theory and linear algebra necessary for spectral clustering. To summarize, we started our spectral clustering process by representing our data as a graph, where the vertices are data points the edges represent the similarities between the points. Next, we generated the Laplacian matrix using the similarities and degrees of each node, which we performed eigendecomposition on. The eigenvalues help determine the suggested number of clusters and the eigenvectors dictate the actual labels of the data points in each cluster.

Spectral clustering has become increasingly popular for several reasons. It makes no assumptions about the shape of the clusters, and its implementation is much easier than other similar methods. Compared to these methods, spectral clustering outperforms them in terms of accuracy. By using dimension reduction, it can correctly cluster observations that truly belong to the same cluster but may be farther in distance than observations in other clusters. In terms of speed, spectral clustering works best when datasets are sparser. Additionally, this technique offers a more flexible approach for finding clusters, useful for datasets which don't meet the requirements for other algorithms.

However, this does not mean that spectral clustering is without its limitations. It is more computationally complex compared to similar methods, the reason being how it uses connectivity. When there is more data, more eigendecomposition needs to be performed, sharply increasing the time complexity. In terms of scalability, the accuracy of spectral

clustering decreases on larger datasets. Furthermore, in the final step of the algorithm, K-means is used to determine the clusters. As a result, based on the choice of the initial centroids, the final clusters generated are not always the same. Replicating our results is not something we can accomplish using this method.

Applications

There are many applications of spectral clustering. EDA (exploratory data analysis), machine learning, and computer vision problems all benefit from the use of spectral clustering. Due to its self-tuning power, spectral clustering is useful in 3D image analysis. This is especially useful in the medical research community as well as the practice of medicine. One application of 3D image analysis includes identifying lower extremity skin wounds in the elderly using optical scans and then feeding those images to the algorithm for processing. Another medical application is in researching therapeutic responses to pancreatic cancers. Spectral clustering can be used to generate gene clusters from cell-line data and CRISPR-Cas9 data which has potential to identify better tissue targets for cancer therapies.

Outside of the medical field, image segmentation through spectral clustering can be used to footage. For example, an individual frame in a movie can be analyzed using spectral clustering which could then result in summarization. This could be useful for giving a visual summary of larger pieces of film.

References

https://bmcmedgenomics.biomedcentral.com/articles/10.1186/s12920-020-0681-6 (https://bmcmedgenomics.biomedcentral.com/articles/10.1186/s12920-020-0681-6)
https://datacarpentry.org/image-processing/aio/index.html (https://datacarpentry.org/image-processing/aio/index.html)
http://ijcsit.com/docs/Volume%206/vol6issue01/ijcsit2015060141.pdf (http://ijcsit.com/docs/Volume%206/vol6issue01/ijcsit2015060141.pdf)
https://pubmed.ncbi.nlm.nih.gov/27520612/ (https://pubmed.ncbi.nlm.nih.gov/27520612/)
https://rpubs.com/nurakawa/spectral-clustering (https://rpubs.com/nurakawa/spectral-clustering)
https://scholarship.claremont.edu/cgi/viewcontent.cgi?article=2735&context=cmc_theses (https://scholarship.claremont.edu/cgi/viewcontent.cgi?article=2735&context=cmc_theses)
https://scikit-learn.org/stable/auto_examples/cluster/plot_segmentation_toy.html (https://scikit-learn.org/stable/auto_examples/cluster/plot_segmentation_toy.html)
https://stackoverflow.com/questions/47689968/python-code-chunk-graphs-not-showing-up-in-r-markdown (https://stackoverflow.com/questions/47689968/python-code-chunk-graphs-not-showing-up-in-r-markdown)
https://stackoverflow.com/questions/6799105/spectral-clustering-image-segmentation-and-eigenvectors (https://stackoverflow.com/questions/6799105/spectral-clustering-image-segmentation-and-eigenvectors)

https://towardsdatascience.com/spectral-clustering-aba2640c0d5b
(https://towardsdatascience.com/spectral-clustering-aba2640c0d5b)
https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7
(https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7)
https://www.analyticsvidhya.com/blog/2021/05/what-why-and-how-of-spectral-clustering/
(https://www.analyticsvidhya.com/blog/2021/05/what-why-and-how-of-spectral-clustering/)
https://www.geeksforgeeks.org/working-with-images-in-python-using-matplotlib/
(https://www.geeksforgeeks.org/working-with-images-in-python-using-matplotlib/)
https://www.kaggle.com/datasets/joanpau/bike-rentals-study-uci
(https://www.kaggle.com/datasets/joanpau/bike-rentals-study-uci)