Theory and Applications of Spectral Clustering

Jiayao (Emily) Li, Juliet Yang

December 2022

1 Background

Clustering refers to a series of techniques that are used in order to assign unlabeled data into different groups based on similarities between the data points. It has applications in a wide variety of fields, playing a large role in particular in the field of unsupervised machine learning. Spectral clustering is a form of clustering that relies on the use of eigenvalues and eigenvectors in order to reduce the dimensions of the data, so that standard clustering algorithms can then be applied. In this project, we implement the spectral clustering algorithm, first performing it on a dummy dataset, before catering our algorithm to analyze music genres.

1.1 Similiarity Graphs and Matrices

Given a set of n data points, one method of representation of those points is in the form of a graph. A graph is a data type that consists of vertices and edges between the vertices. Because clustering is mainly concerned with grouping similar data points, we use the data to construct a similarity graph, which is represented as the $n \times n$ A where $A_{ij} = 1, i \neq j$ if there is some degree of similarity between the ith data point and the jth data point. More specifically, the matrix A is an adjacency matrix representation of our similarity graph where

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between data point i and j} \\ 0 & \text{otherwise} \end{cases}$$

The existence of an edge between data point i and j indicates that a certain degree of similarity between the two points are met. For weighted graphs, 1 would be replaced with the weight of the edge. Note that our similarity graph is undirected, which implies that $A_{ij} = A_{ji}$ and we have no self edges, so $A_{ii} = 0$.

Similarity graphs can be derived in many ways. For the purposes of this project, we use the k-nearest neighbors search technique. The way that this technique works is that for each data point, its k-nearest neighbors are found (determined based on Euclidean distance), where k is specified by the user. An edge is then drawn from that data point to each of its k neighbors to create our similarity graph, which we represent in our adjacency matrix format as demonstrated above.

In addition to our adjacency matrix, we also have a matrix D called the degree matrix. Assuming we have n data points, this is the diagonal matrix

$$D = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}$$

where $D_{ii} = d_i$, in which d_i represents the degree of the ith data point. The degree of a data point of a vertex is defined to be the number of edges that are connected to it.

1.2 Graph Laplacian Matrix

Now that we have some form of representation for our data, we can use the adjacency and diagonal matrices we have described to construct the graph Laplacian matrix. The graph Laplacian matrix is another way to represent a graph and it is defined as follows:

$$L = D - A$$

where D and A are the degree and adjacency graphs of a set of data points.

The Laplacian has several properties, which are outlined below [2]:

1. For every $\vec{x} \in \mathbb{R}^n$,

   $$\vec{x}^T L \vec{x} = \frac{1}{2} \sum_{i,j=1}^{n} a_{ij}(x_i - x_j)^2$$

   We prove this as follows,

   $$\begin{aligned} \vec{x}^T L \vec{x} &= \vec{x}^T (D - A)\vec{x} \\ &= \vec{x}^T D\vec{x} - \vec{x}^T A\vec{x} \\ &= \sum_{i=1}^{n} d_i x_i^2 - \sum_{i,j=1}^{n} x_i a_{ij} x_j \\ &= \frac{1}{2}\left(2\sum_{i=1}^{n} d_i x_i^2 - 2\sum_{i,j=1}^{n} x_i a_{ij} x_j \right) \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} d_i x_i^2 - 2\sum_{i,j=1}^{n} x_i a_{ij} x_j + \sum_{j=1}^{n} d_j x_j^2 \right) \\ &= \frac{1}{2} \sum_{i,j=1}^{n} a_{ij}(x_i - x_j)^2 \end{aligned}$$

2. L has n real, non-negative eigenvalues and orthogonal eigenvectors

   D and A are symmetric, which implies that L is also symmetric, so the properties outlined are derived from the spectral theorem.

3. L is positive semi-definite with smallest eigenvector 0.

   We know that L is symmetric and from part 1, we note that $(x_i - x_j)^2 \geq 0$ and and that $a_{ij} \geq 0$ from our adjacency matrix representation, so $\vec{x}^T L \vec{x} \geq 0$, proving that L is positive semi-definite, which means that all n eigenvectors are ≥ 0.

Using these properties, we can then state the following about L. We can prove that the multiplicity m of the eigenvalue 0 for L corresponds to the number of connected components for the graph that L represents. A connected component refers to a set of vertices that are connected by path. In other words, for any two vertices that are contained in the same connected component, there exists a path between those vertices. A more detailed proof is outlined in [2], but the general gist is that if we assume that $\vec{x}$ is an eigenvector for L with eigenvalue 0, then we can use the equation in part 1 to derive that if two vertices are connected then $x_i = x_j$, which means that all vertices within the same connected component must have the same eigenvector $\vec{x}$. So, if there are m eigenvectors, there are m connected components

Then, for each eigenvalue close to zero, it follows that the data is close to having an additional connected component. So, when determining the number of clusters, the standard is to look for the first large gap between eigenvalues and cluster into k groups where k is the number of eigenvalues before the first gap.

1.3 K-Means

Once we have determined what our k value is, we take the first k eigenvectors $\vec{x}_1, \cdots, \vec{x}_k$ and put them in a matrix, which we will call E, as follows,

$$E = \left(\vec{x}_1 | \cdots | \vec{x}_k\right)$$

Then, we perform standard clustering algorithms on the rows of E. The reason why we take the first k eigenvectors is because it helps to reduce data that has many dimensions to a lower k-dimensional space, so that clustering is easier.

The clustering algorithm that we primarily use in this project is the k-means clustering algorithm. The way the algorithm works is that in order to sort n data points into k groups, it will arbitrarily choose k random data points to act as clustering centroid. Then, it will iterate through all n data points many times and assign each to the nearest clustering centroid. As the algorithm is iterating, the centroid will readjust its position to the one that is the average of all the points that have been assigned to it. The algorithm continues to iterate through all the points until the centroids reached a fixed state, which indicates that the point assignments have also become fixed.

2 Methods

2.1 Algorithms and Computation

In this section, we translate the theoretical ideas we discussed in the previous section into code format. We present a more generalized code version that can be adapted to analyze different datasets. Some of the code has been adapted to pseudocode for easier understanding. For the full code, refer to the appendix.

Assume we start off with n data points in $\mathbb{R}^m$ and assume we have put them in an $m \times n$ matrix called `data`. The first step is to create our similarity matrix through the use of k-nearest neighbors. As a quick note, the k-nearest neighbors function in Julia uses a tree of the points as well as the matrix of the points in order to perform the data search. We chose to use a kd tree as our representation because the NearestNeighbors package documentation specified that it works well with axis aligned metrics like Euclidean distance. The kd tree works by recursively splitting the points into groups. Also note that we used numNeighbors + 1 instead of numNeighbors for the knn function since for each node, the function also outputs the current node itself as a closest neighbor. Our code is presented as follows,

```
kdtrees = KDTrees(data)
# idx is a vector where the ith component is a vector for neighbors of
# data point i
idx, dists = knn(kdtree, data, numNeighbors + 1)
neighbors = (matrix representation of idx, self-neighbors are excluded)
```

Now that we have our neighbors, we can begin to construct our adjacency and diagonal matrices. For our adjacency matrix, we start off with a $n \times n$ matrix of zeroes and loop through the neighbors, updating the matrix with 1s for each neighboring pair.

```
adj = zeros(n, n)
for i in 1:n
  for j in 1:numNeighbors
    adj[i, neighbors[i, j]] = 1
    adj[neighbors[i, j], i] = 1
  end
end
```

For our diagonal matrix, we start off with a $n \times n$ matrix of zeros and set each diagonal entry to be the sum of the corresponding row of the adjacency matrix, since we represent edges with 1s.

```
degree = zeros(n, n)
for i in 1:n
  degree[i, i] = sum(adj[i,:])
end
```

Now, we have our Laplacian matrix.

```
laplacian = degree - adj
```

Then, we find the eigenvectors and eigenvalues of the matrix. Based off analysis of the eigenvalues, we can then specify the number of clusters we want and construct a matrix using that number of eigenvectors to perform k-means on the rows of A.

```
eigenval, eigenvec = eigen(laplacian)
A = eigenvec[:,1:numClusters]
km = kmeans(transpose(A), numClusters)
```

And we're done. Information about the cluster each point is assigned to is stored in `km.assignments`. For visualization purposes, the data can be represented in scatter plot form with data point coloring based off of the cluster assignments in `km.assignments`.

2.2 Coding and Algorithm Design Decisions

Here, we will discuss the following design decisions:

1. Similarity graph choice
2. Clustering algorithm choice

2.2.1 Similarity Graph Choice

For our similarity graph, we chose to use the k-nearest neighbors technique to construct the edges of our graph, which was detailed earlier. We chose to use the knn technique because of its simplicity, since the technique primarily relies on distances (in our case, Euclidean distances) to perform the search. The simplicity of knn also means that it can be used on many different types of data, since the data does not need to meet any starting assumptions in order to work. This works well when we want to find patterns in data that we don't know much about. Finally, the knn technique does not require any training period, which means that new data can be added or old data can be removed easily and the model can quickly adapt to each dataset.[5]

2.2.2 Clustering Algorithm Choice

There are many different clustering algorithms that can be used to group datasets. For this project, we chose to use the k-means clustering algorithm. Like the knn search technique, one of the advantages of the algorithm is its simplicity, both in its implementation as well as its ease of understanding. In addition, the k-means algorithm is a widely used clustering algorithm that works well with large datasets. Finally, the algorithm will always guarantee convergence, which refers to successful clustering of the data, making it adaptable to many different datasets. [6]

3 Experimentation

3.1 Datasets

We evaluated our model on two different datasets of various sizes. All of the samples and features were numerically represented.

The first dataset was a dummy dataset consisting of 100 randomly generated 2D data points in two clusters. This was used as proof of concept to ensure no major systematic error exists in our method. We generated these data points using Julia's MLJ and DataFrames package. We specified the number of neighbors in the KNN algorithm to be 10, which means that each data point will be connected to the 10 closest neighbors in our graph.

The second dataset was a subset of the GTZAN Music Genre Classification Dataset consisting of 1000 samples of various music belonging to 10 categories across 198 feature dimensions.

Each feature encoded a musical characteristic such as “tempo”, “chroma stft”, and “harmony mean”. The goal was to evaluate our model’s ability to cluster music by genre given these 198 characteristics in an unsupervised setting.

3.2 Data Preparation

Before applying our model to these datasets, we noted that certain values ranged numerically by a substantial amount depending on the feature dimension in the second and third datasets. And since an algorithm like K-means is sensitive to variances in input data, we applied a Z-Score Transformation to standardize the data points using the ZScoreTransform package provided by Julia in order to prevent the clustering dominated by features with a bigger scale. This normalization transformed our data points into corresponding standard scores by subtracting each value from the mean and scaling to the unit variance **zscoretransform**. We also explored other normalization methods such as a Unit Range Transform **unit range**, but Z-Score Transformation yielded more consistent clustering results.

4 Results

We present the following results of our experiment on all two datasets.

4.1 Dummy Dataset

4.1.1 Clustering

Plots of randomly generated 2-dimensional data points belonging to two clusters from before and after clustering are shown in Figures 1 and 2.

We also evaluated our model on randomly generated data clearly belonging to four clusters. Plots of the 2-dimensional data points from before and after clustering are shown in Figures 3 and 4.

As can be seen, our model competently clusters the data points into two and four corresponding clusters.

4.1.2 Eigenvalues of the Laplacian Matrix

We computed the eigenvalues and eigenvectors of the Laplacian matrix for the set of points generated for both two clusters and four clusters to verify our choice of k for the K-means algorithm. Figures 5, 6 illustrate plots of the first 15 eigenvalues.

For the 2-cluster plot, the first two eigenvalues are observed to be close to zero before the third eigenvalue spikes to around 4. According to the previously discussed theory, this implies that there are likely two connected components in the Laplacian graph. Since the ground truth number of clusters in the plot is indeed two, we can confirm that the value of k should indeed be two.

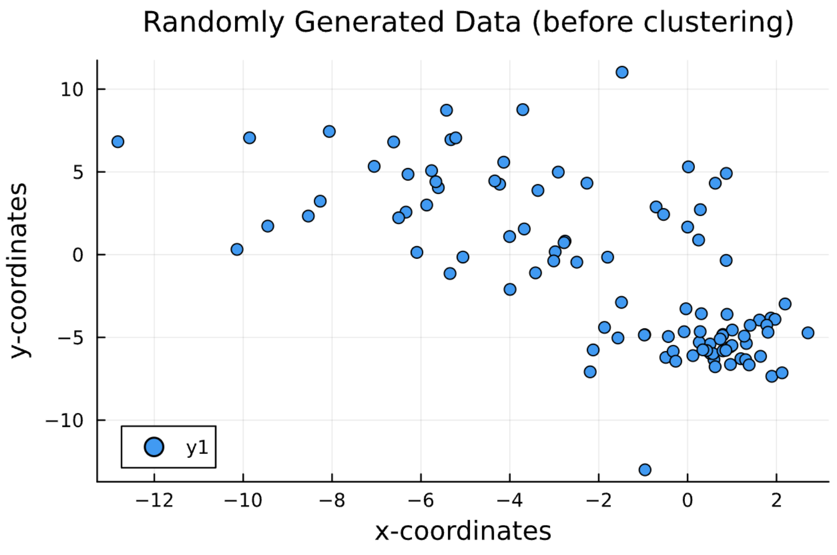

Figure 1: Visualization of 2-cluster 2D data points before clustering

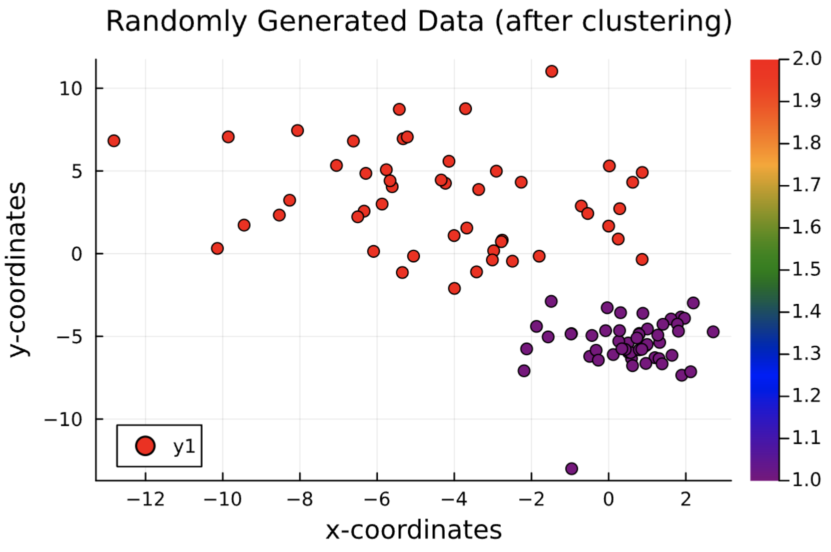

Figure 2: Visualization of 2-cluster 2D data points after clustering

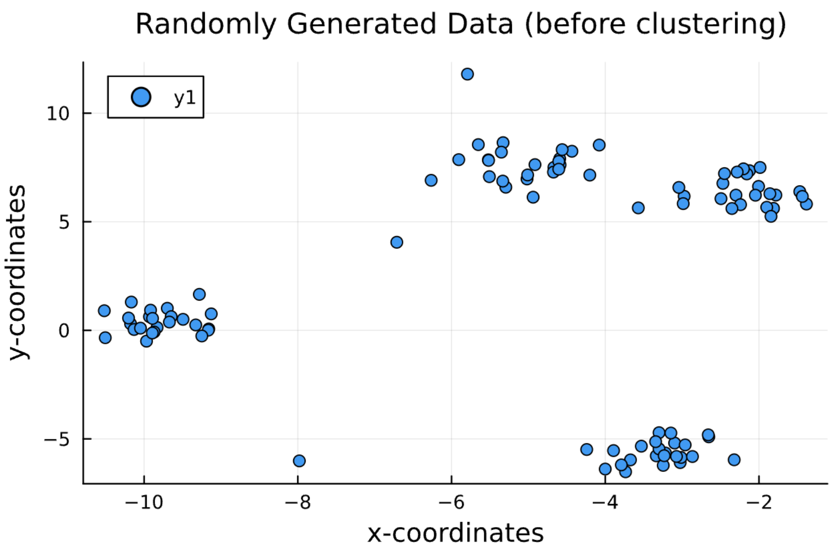

Figure 3: Visualization of 4-cluster 2D data points before clustering

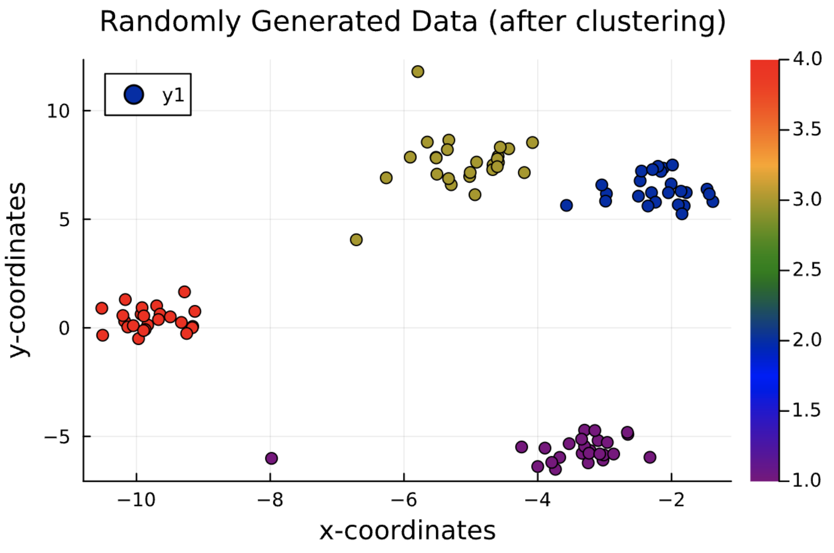

Figure 4: Visualization of 4-cluster 2D data points after clustering

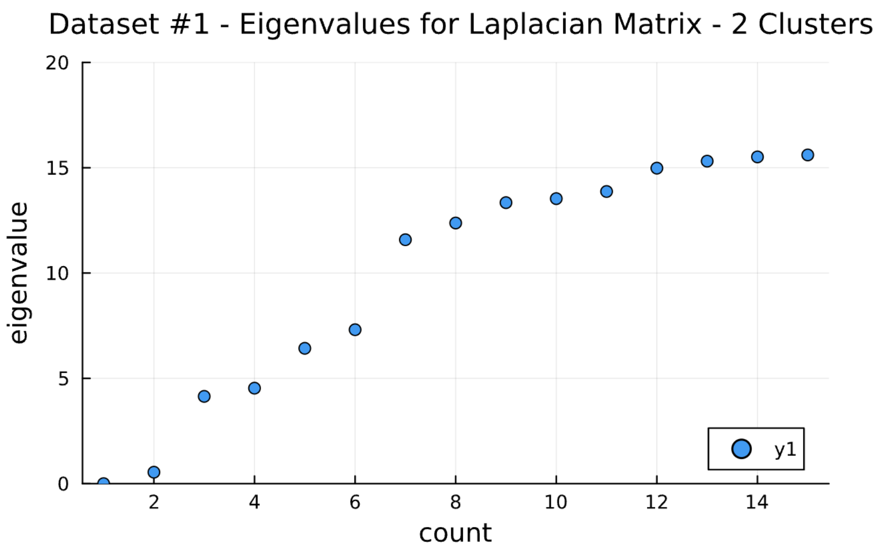

Figure 5: Visualization of eigenvalues for 2 clusters

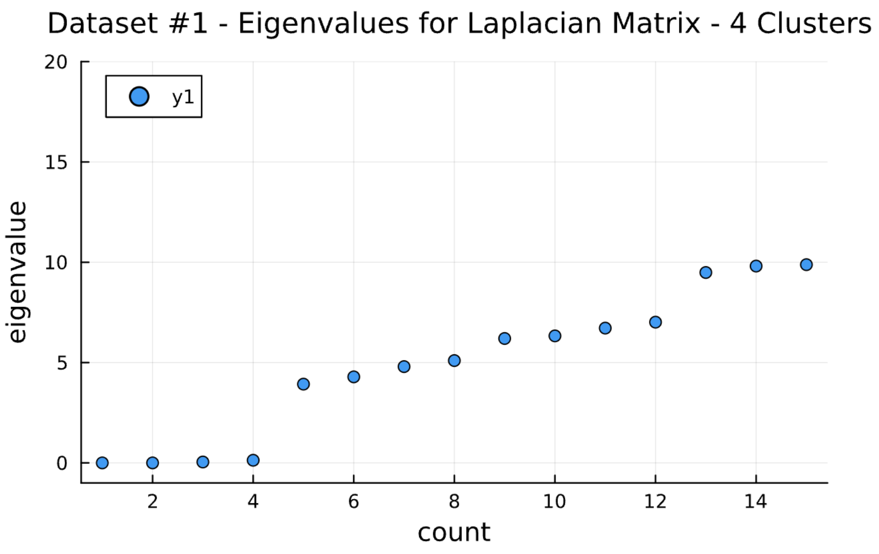

Figure 6: Visualization of eigenvalues for 4 clusters

We observed similar results for the 4-cluster plot, the first four eigenvalues are observed to be close to zero before the third eigenvalue spikes to around 4. This implies that there are likely four connected components in the Laplacian graph. Since the ground truth number of clusters in the plot is indeed four, we can confirm that the value of k should indeed be two. The observed results from both the 2-cluster and 4-cluster trials agree with our theoretical predictions.

4.2 Music Genre Classification Dataset

4.2.1 Clustering

Plots of the clustering results on the first dimension (tempo) of the music vs. the 31st dimension (roll-off mean) for all 1000 samples are shown in Figures 7 and 8. Figure 7 illustrates the clustering of the data with respect to the original scale while Figure 8 illustrates the clustering of the same data points with respect to the normalized scale based on the z-score of each point. Note that in music, the rolloff frequency denotes the approximate low bass and high treble limits in a frequency response curve. Higher rolloff frequencies are typically more prominent in types of music that involve distortion such rock and hip-hop, as opposed to jazz, blues, and classical music.

In each plot, the color of the clusters corresponds to the category it is clustered into based on a rainbow scale representing categories 1-10. The relatively clear clustering of the data points in both clusters demonstrates that there is a distinct correlation between the type of music, its tempo, and the rolloff frequency. In Figure 7, we note that the light blue cluster located near the top of the plot corresponds to the category of ”metal music” in our dataset. This indicates that our model is able to recognize that metal music generally tends to have higher mean roll-off frequencies. On the other hand, the dark green and dark blue cluster towards the bottom of the plot correspond to categories of ”blues” and ”country” music, both of which are music genres that involve lower rolloff frequencies.
It is also interesting to note that the tempo consistently varies among all ten categories of music. This is expected since in each of the ten music genres, music could be both fast and slow.

Figure 9 illustrates a plot of the tempo vs chroma feature in the music and provides a view of the clustering from another dimension. In a musical context, chroma-based features indicate the energy distribution among the twelve chromas to determine the harmonic progression of the underlying piece. This plot contains a clustering of categories that could clearly be distinguishable yet contain overlaps. This observation is expected since music from certain categories such as pop, classical, country often have the same if not similar progressions, yet music from other genres such as jazz has unique chord progressions and scales.

4.2.2 Eigenvalues of the Laplacian Matrix

We, again computed the eigenvalues and eigenvectors of the Laplacian matrix for the set of 1000 music data points to verify our choice of k for the K-means algorithm. Figure 10 illustrate plots of the first 15 eigenvalues.

In Figure 10, we observe that the first eigenvalue is the only eigenvalue that is very close to being 0. This indicates that there is only one connected component in the Laplacian graph.

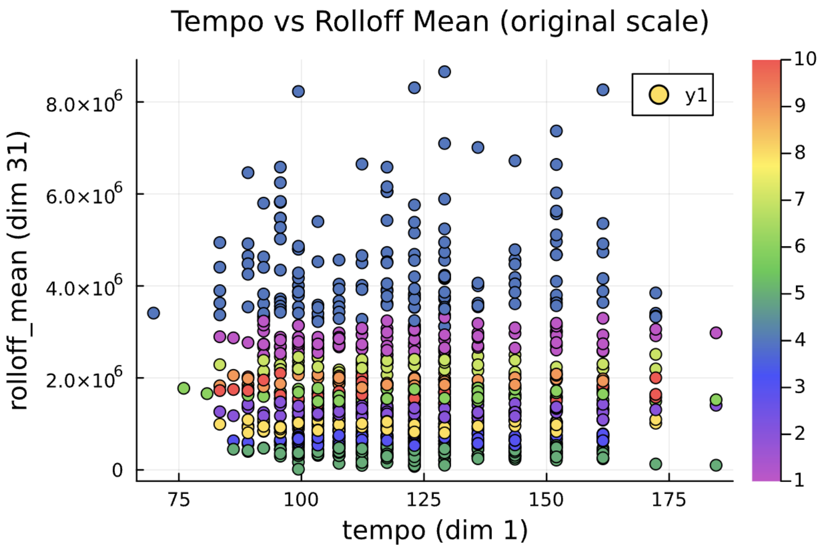

Figure 7: Tempo vs. Rolloff mean clustering of dataset 2 based on original scaling

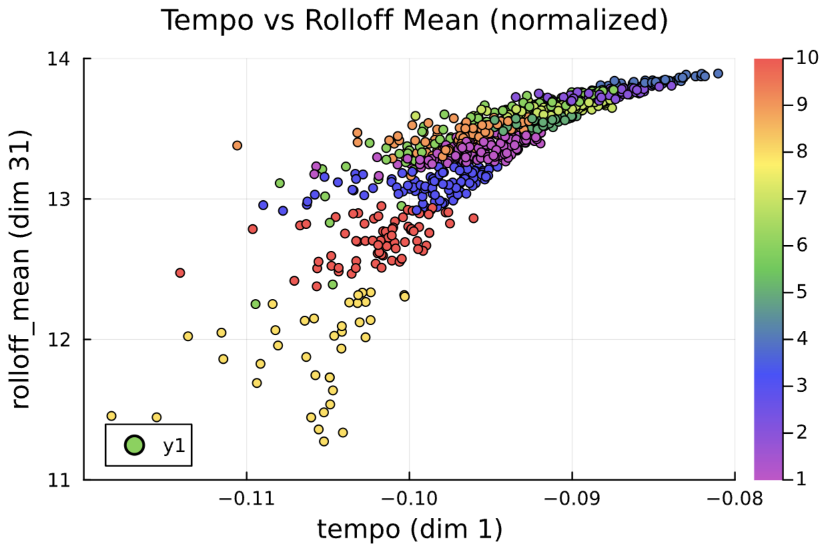

Figure 8: Tempo vs. Rolloff mean clustering of dataset 2 based on normalized scaling

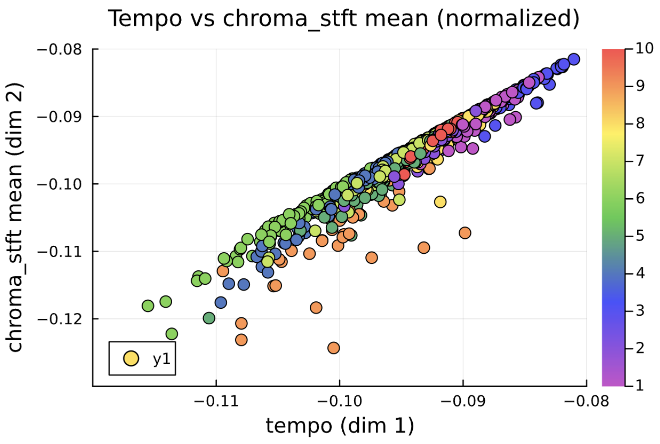

Figure 9: Tempo vs. chroma stft mean clustering of dataset 2 based on normalized scaling

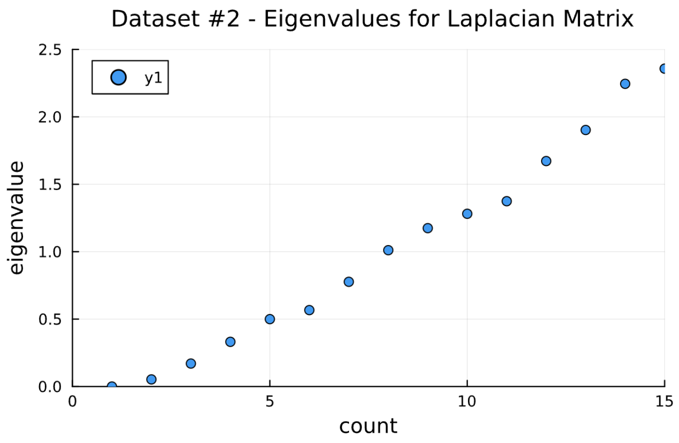

Figure 10: Visualization of eigenvalues for Music Genre Classification Dataset

After the first eigenvalue, the other eigenvalues gradually begin to increase but don't spike until the ten and the eleventh eigenvalue with a value of around 1.7. Since the ground truth number of clusters/categories of musical genres in this dataset is 10, it is very reasonable for the chosen value k to be 10.

5 References

5.1 General References:

[1] Fleshman, William. “Spectral Clustering.” Medium, Towards Data Science, 21 Feb. 2019, https://towardsdatascience.com/spectral-clustering-aba2640c0d5b.

[2] Von Luxburg, Ulrike. ”A tutorial on spectral clustering.” Statistics and computing 17.4 (2007): 395-416.

[3] Arora, Taaniya. “Graph Laplacian and Its Application in Machine Learning.” Medium, Towards Data Science, 23 Oct. 2022, https://towardsdatascience.com/graph-laplacian-and-its-application-in-machine-learning-7d9aab021d16.

[4] Lecture8-MassachusettsInstituteofTechnology.https://people.csail.mit.edu/dsontag/cou rses/ml14/slides/lecture8.pdf.

[5] Soni, Anuuz. “Advantages and Disadvantages of KNN.” Medium, Medium, 3 July 2020, https://medium.com/@anuuz.soni/advantages-and-disadvantages-of-knn-ee06599b9336.

[6] “K-Means Advantages and Disadvantages — Machine Learning — Google Developers.” Google, Google, https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages.

[7] “Roll-off Frequency.” SVS, https://www.svsound.com/blogs/glossary/roll-off-frequency.

5.2 Dataset and Package Links

https://www.kaggle.com/datasets/gabrielopecs/gtzan-modified-music-genre-classification?r esource=download

https://github.com/KristofferC/NearestNeighbors.jl

https://juliastats.org/StatsBase.jl/latest/transformations/

6 Appendix

Full source code: `https://github.com/emilyjiayaoli/spectral-clustering-project`

Julia code for dummy dataset:

```julia
numNeighbors = 10
numPoints = 100
numClusters = 4

# Randomly generates clustered dataset
using MLJ, DataFrames
X, y = make_blobs(numPoints, numClusters; centers=2, cluster_std=[1.0, 3.0])
dfBlobs = DataFrame(X)

# Extracts data from the dataframe
data = dfBlobs[:,1:2]
x1 = dfBlobs[:,1]
x2 = dfBlobs[:,2]


# Plots the randomly generated data
using Plots
scatter(x1, x2, xlims=(-15,15), ylims=(-15,15), aspect_ratio=:equal)

using LinearAlgebra
data = transpose([x1 x2])

# Performs knn search and puts neighbors into a matrix
using NearestNeighbors
kdtree = KDTree(data)
idx, dists = knn(kdtree, data, numNeighbors + 1, true)
idxMatrix = mapreduce(permutedims, vcat, idx)
neighbors = idxMatrix[:,2:numNeighbors + 1]

# Calculates adjacency matrix
adj = zeros(numPoints, numPoints)
for i in 1:numPoints
  for j in 1:numNeighbors
    adj[i, neighbors[i, j]] = 1
    adj[neighbors[i, j], i] = 1
  end
end

# Calculates degree matrix
degree = zeros(numPoints, numPoints)
for i in 1:numPoints
```

```
  degree[i, i] = sum(adj[i,:])
end

laplacian = degree - adj

# Extracts first the first k (numClusters) eigenvectors
eigenval, eigenvec = eigen(laplacian)
A = eigenvec[:,1:numClusters]

# Performs kmeans and plots data again, coloring based on cluster assignment
using RDatasets, Clustering
km = kmeans(transpose(A), numClusters)
scatter(x1, x2, marker_z=km.assignments, color=:rainbow, legend=false)
```

Julia code for dataset 2:

```
import Pkg
Pkg.add("DelimitedFiles")
Pkg.add("StatsBase")

# Loading Data
using DelimitedFiles
filepath = "./music_genre_data.csv"
data = readdlm(filepath, ',')
#/Users/emily/Desktop/21-241\ Linear/

# Data Info: 1000 data points, 199 dimensions of features

# removed the tile rows and columns, turn into float
data_num = Float64.(data[2:1000,2:198])

# Applied normalization
using StatsBase
using Random

#standardizing accross the columns, so dims=2
X_standard = standardize(ZScoreTransform, data_num, dims=2)

# First parameter options: UnitRangeTransform, ZScoreTransform

X_transposed = transpose(X_standard) # rows are dimensions, columns are datapoints
# X = X_transposed[:, shuffle(1:999)]

X = X_transposed[:, 1:999]

numNeighbors = 15
numPoints = 999
```

```
#numClusters = 10 # value based on the first gap in eigenvalues

using DataFrames
using NearestNeighbors
kdtree = KDTree(X)

idx, dists = knn(kdtree, X, numNeighbors + 1, true)

# Turns vector within vector into matrix
idxMatrix = mapreduce(permutedims, vcat, idx)


# Adjancency list (list of indexes for k nearest neighbors at each index):
neighbors = idxMatrix[:,2:numNeighbors + 1]


# Creating adjancency matrix
adj = zeros(numPoints, numPoints)

for i in 1:numPoints
    for j in 1:numNeighbors
      adj[i, neighbors[i, j]] = 1
      adj[neighbors[i, j], i] = 1
    end
  end


degree = zeros(numPoints, numPoints)

for i in 1:numPoints
    degree[i, i] = sum(adj[i,:])
  end

laplacian = degree - adj


using LinearAlgebra
eigenval, eigenvec = eigen(laplacian)


scatter(collect(1:999), eigenval,
  xlim=(0, 15), ylim=(0,2.5),
  xlabel="count", ylabel="eigenvalue",
  title="\n Dataset #2 - Eigenvalues for Laplacian Matrix",
  titlefont=font(12),
  bottom_margin = 10mm, top_margin = 5mm,
  left_margin = 10mm, right_margin = 10mm)

# Specify number of clusters
numClusters = 10

# Retrieve specified amount of eigenvectors
A = eigenvec[:,2:numClusters + 1]

# Perform k means
using RDatasets, Clustering
km = kmeans(transpose(A), numClusters)


# Plotting results.
using Plots
using Plots.PlotMeasures
s = 2.5
scatter(X[1, :], X[31, :],
      marker_z=km.assignments,
      color=:lightrainbow,
      legend=false, markersize=s)

s = 4
scatter(X[1, :], X[31, :],
            marker_z=km.assignments,
            color=:lightrainbow,
            legend=true, markersize=s,
            xlabel="tempo (dim 1)",
            ylabel="rolloff_mean (dim 31)",
            title="\n Tempo vs Rolloff Mean (original scale)" ,
            titlefont=font(12),
            bottom_margin = 10mm,
            top_margin = 5mm,
            left_margin = 10mm,
            right_margin = 10mm)


# Access results, clusters to verify and interpret clustering results

@assert nclusters(km) == numClusters # verify the number of clusters

a = assignments(km) # get the assignments of points to clusters
c = counts(km) # get the cluster sizes
M = km.centers # get the cluster centers
```