Lecture 21

Spectral Clustering

Spectral Clustering is a clustering technique that uses the eigenvalues (spectrum) of a similarity matrix (derived from the data) to perform dimensionality reduction before applying a standard clustering algorithm like K-means. Unlike traditional clustering methods that operate directly on the data points, spectral clustering makes use of the structure of the data as captured in a graph, where nodes represent data points and edges represent the similarity between them.

How Spectral Clustering Works

1. *Construct a Similarity Graph*:
    - Represent the data as a graph. Each data point is a node, and edges between nodes represent the similarity between them.
    - The similarity can be computed using different methods, such as the Gaussian (RBF) kernel or nearest neighbors.

2. *Compute the Laplacian Matrix*:
    - From the similarity graph, compute the graph Laplacian matrix. There are different types of Laplacians (e.g., unnormalized, normalized).
    - The Laplacian matrix encapsulates the structure of the graph, including connections between data points.

3. *Eigen Decomposition*:
    - Compute the eigenvalues and eigenvectors of the Laplacian matrix.
    - Select the first k eigenvectors (where k is the desired number of clusters) to form a new representation of the data. This step is akin to dimensionality reduction.

4. *Clustering in the Reduced Space*:
    - Treat the rows of the matrix of selected eigenvectors as new data points in a lower-dimensional space.
    - Apply a standard clustering algorithm like K-means to these new data points to obtain the final clusters.

5. *Assign Clusters*: The resulting clusters from the clustering algorithm are used as the final clusters for the original data points.

Advantages of Spectral Clustering

- *Handles Non-Convex Clusters*: Unlike K-means, which assumes convex cluster shapes, spectral clustering can handle more complex and non-convex structures.
- *Incorporates Graph-Based Structure*: It can leverage graph-based relationships, making it more flexible and applicable in situations where other clustering methods may struggle.
- *Flexibility with Similarity Measures*: Different types of similarity measures can be employed, making it adaptable to various kinds of data.

Disadvantages of Spectral Clustering

- *Computationally Expensive*: The eigen decomposition step, which involves computing eigenvectors and eigenvalues, can be computationally intensive, especially for large datasets.

- *Choice of Similarity Measure and Parameters*: The results of spectral clustering can be sensitive to the choice of similarity measure and the parameters used to construct the graph (e.g., the value of sigma in the Gaussian kernel or the number of nearest neighbors).
- *Number of Clusters (k) Must Be Pre-Specified*: Similar to K-means, spectral clustering requires specifying the number of clusters beforehand.

Applications of Spectral Clustering

- *Image Segmentation*: Commonly used in computer vision for segmenting images into different regions.
- *Graph Partitioning*: Applied in areas where the data can naturally be represented as a graph, such as social networks or biological networks.
- *Data Mining*: Useful in finding patterns and structures in high-dimensional data.

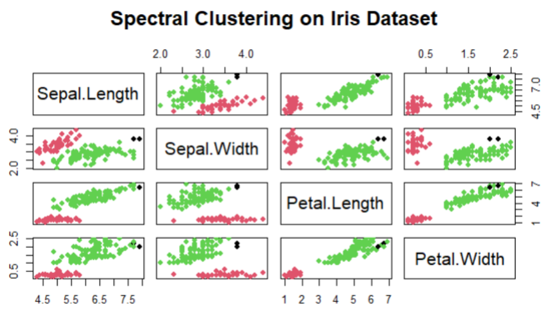
Example Algorithms

1. *Normalized Cut (NCut):* A popular form of spectral clustering used in image segmentation. It partitions a graph based on minimizing a normalized measure of the graph's cut.
2. *Shi-Malik Algorithm*: A specific algorithm used for image segmentation based on spectral clustering.

Let's look at an example of spectral clustering using an example on the iris dataset.

```
# Load necessary library
library(kernlab)

# Load iris dataset (using only numeric variables)
data(iris)
iris_data <- iris[, -5]  # Remove the species column for clustering

# Check the dimensions of the data to ensure correctness
print(dim(iris_data))  # Should return 150 rows and 4 columns

# Apply spectral clustering
# Ensure that centers parameter is set correctly
sc <- specc(as.matrix(iris_data), centers = 3)

# Check the length of the clustering result
length(sc)  # This should return 150, not 4

# Check the length of the true labels
length(iris$Species)  # Should be 150

# If the lengths match, create the table
if(length(sc) == length(iris$Species)) {
  clustering_results <- table(sc, iris$Species)
  print(clustering_results)
} else {
  stop("The lengths of the clustering results and true labels do not match.")
}
```

```
# Assuming all above steps are correct
# Apply spectral clustering again
sc <- specc(as.matrix(iris_data), centers = 3)

# Check the output
print(length(sc))  # Should be 150

# Generate confusion matrix if lengths are correct
if (length(sc) == 150) {
  clustering_results <- table(sc, iris$Species)
  print(clustering_results)
} else {
  print("The lengths of the clustering results and the true labels do not match.")
}

# Visualization
plot(iris_data, col = as.factor(sc), pch = 19, main = "Spectral Clustering on Iris Dataset")
```



What we see happening here is that spectral clustering did less well on this dataset than did k-means in the previous lecture. This could be because the clusters in the iris dataset are actually convex and work well with k-means, while spectral clustering is designed to work best with non-convex sets. Particularly when you are doing data exploration, you don't know *a priori* what the clusters look like, which is one of the reasons we test multiple algorithms before settling on a favorite. It will depend on the data.

Let's look more closely at what is going on behind the scenes.

Construct a Similarity Matrix

A common choice is the Gaussian (RBF) kernel: $S_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$

Where $\|x_i - x_j\|^2$ is the Euclidean distance between points x_i and x_j, and σ is a parameter that controls the width of the Gaussian kernel.

```
# Load the iris dataset, excluding the species column
data(iris)
iris_data <- iris[, -5]  # Use only the numeric columns

# Set a sigma value for the Gaussian kernel
```

```
sigma <- 1

# Construct the similarity matrix using Gaussian kernel
n <- nrow(iris_data)
similarity_matrix <- matrix(0, n, n)
for (i in 1:n) {
  for (j in 1:n) {
    similarity_matrix[i, j] <- exp(-sum((iris_data[i, ] - iris_data[j, ])^2) / (2 * sigma^2))
  }
}

# View the similarity matrix
head(similarity_matrix)
```

Compute the Laplacian Matrix
The Laplacian matrix L is computed as:

$$L = D - S$$

Where D is the degree matrix (a diagonal matrix where each entry is the sum of the corresponding row in the similarity matrix S).

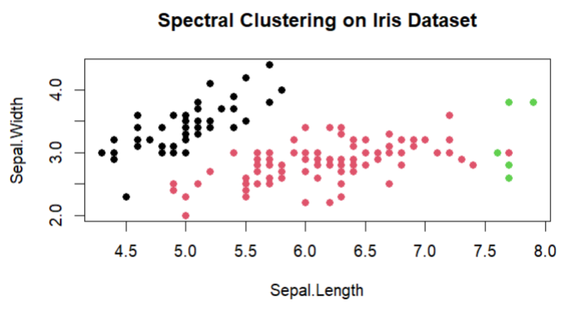
```
# Compute the degree matrix
degree_matrix <- diag(rowSums(similarity_matrix))

# Compute the Laplacian matrix
laplacian_matrix <- degree_matrix - similarity_matrix

# View the Laplacian matrix
head(laplacian_matrix)
```

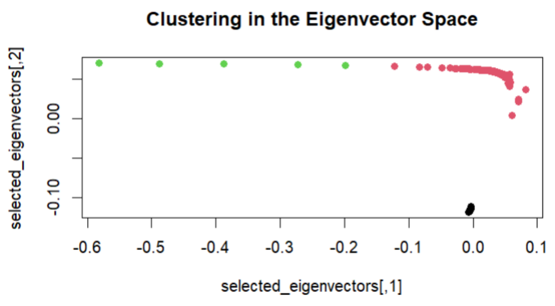
Compute the Eigenvectors of the Laplacian
The next step is to compute the eigenvectors and eigenvalues of the Laplacian matrix. We'll use the eigenvectors corresponding to the smallest non-zero eigenvalues for clustering.

```
# Compute eigenvalues and eigenvectors
eigen_decomp <- eigen(laplacian_matrix)
eigenvalues <- eigen_decomp$values
eigenvectors <- eigen_decomp$vectors

# Select the eigenvectors corresponding to the smallest non-zero eigenvalues
# Let's select the first 2 or 3 eigenvectors, depending on the clusters you want
selected_eigenvectors <- eigenvectors[, (n-2):(n-1)]  # Selects the last 2 eigenvectors

# View the selected eigenvectors
head(selected_eigenvectors)
```

Apply k-Means to the Embedded Data
Finally, we use k-means clustering on the rows of the matrix formed by the selected eigenvectors to cluster the original data points.

```
# Apply k-means clustering to the selected eigenvectors
set.seed(123)  # For reproducibility
k <- 3  # Number of clusters
clusters <- kmeans(selected_eigenvectors, centers = k)$cluster

# Compare the resulting clusters with the true species labels
table(clusters, iris$Species)

# Visualize the clusters
plot(iris_data[, 1:2], col = clusters, pch = 19,
    main = "Spectral Clustering on Iris Dataset")

# Plot the rows of the selected eigenvectors
plot(selected_eigenvectors, col = clusters, pch = 19,
    main = "Clustering in the Eigenvector Space")
```





Let’s look at some of the customizations that are possible with spectral clustering if you need to improve the results.

1. Similarity Matrix Customizations

- *Choice of Kernel/Distance Function*: While the Gaussian (RBF) kernel is common, other kernel functions can be used depending on the nature of the data. For example:
    - Linear Kernel: Suitable for linearly separable data.
    - Polynomial Kernel: Can capture more complex structures.
    - Cosine Similarity: Useful for text data or where angles between points are meaningful.
    - Manhattan Distance: An alternative to Euclidean distance, especially in high-dimensional spaces.

- *Parameter Tuning (e.g., σ in Gaussian Kernel)*: The choice of parameters like σ in the RBF kernel can significantly impact the performance of spectral clustering. Cross-validation or heuristic methods (e.g., median heuristic) are often used to tune these parameters.

- *k-Nearest Neighbors (k-NN) Graph*: Instead of using a fully connected graph, you can build a k-NN graph where each point is only connected to its k nearest neighbors. This can reduce computational cost and make the method more robust to outliers.

- *Sparse Graphs*: In large datasets, constructing a sparse similarity matrix (e.g., using only the top connections) can significantly reduce memory and computational requirements.

2. Choice of Laplacian Matrix

- *Unnormalized Laplacian*: $L = D - S$, where D is the degree matrix and S is the similarity matrix.

- *Normalized Laplacians*:
    - Symmetric Normalized Laplacian: $L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$. This version normalizes the Laplacian to remove the influence of node degree, which can be helpful when node degrees vary significantly.
    - Random Walk Normalized Laplacian: $L_{rw} = D^{-1}L$. This variant normalizes differently and is interpreted as a random walk on a graph.

- *Choice of Eigenvectors*: Instead of using just the smallest non-zero eigenvectors, you can explore methods like spectral gap analysis to determine the optimal number of eigenvectors to use.

3. Dimensionality Reduction Techniques

- *PCA Before Spectral Clustering*: Applying PCA to the data before spectral clustering can help reduce dimensionality and computational cost, especially in high-dimensional spaces.

- *Laplacian Eigenmaps*: This technique involves using the eigenvectors corresponding to the smallest non-zero eigenvalues of the normalized Laplacian to embed data into a lower-dimensional space before clustering.

4. Clustering Algorithm

- *Different Clustering Algorithms*: While k-means is the most commonly used clustering algorithm after the eigen-decomposition step, other clustering methods can also be applied, such as:
    - *Gaussian Mixture Models (GMM):* This allows for a probabilistic clustering approach.
    - *Agglomerative Hierarchical Clustering*: This can be used when the number of clusters is not known in advance.

- *Multiple Runs and Consensus Clustering*: Running k-means multiple times with different initializations and using consensus clustering (e.g., majority voting across runs) can help reduce sensitivity to initialization.

5. Handling Multi-Scale Data

- *Multi-Scale Clustering*: If your data has different structures at different scales, you can apply spectral clustering at multiple scales (varying the kernel parameter σ\sigmaσ) and then combine the results.

6. Post-Processing

- *Outlier Detection*: Spectral clustering may be sensitive to outliers. After clustering, you can identify and possibly remove outliers based on their distance from cluster centroids or by examining their behavior in the eigenvector space.

- *Cluster Refinement*: Sometimes, it's beneficial to refine clusters after the initial clustering, such as by applying further k-means clustering within each cluster or by merging/splitting clusters based on additional criteria.

7. Dealing with Large Datasets

- *Nyström Method*: This method approximates the eigenvectors for large datasets by sampling a subset of the data, computing the eigenvectors on this subset, and then extending them to the full dataset.

- *Landmark Spectral Clustering*: Another approach for large datasets is to use a subset of the data (landmarks) to approximate the Laplacian and its eigenvectors.

Visualization Customizations

- *Embedding Visualization*: After clustering, visualize the data in the lower-dimensional space spanned by the eigenvectors. This can help you understand the structure of the clusters.

- *Graph Visualization*: Visualizing the similarity graph or the Laplacian matrix can provide insights into the connectivity and structure of the data.

Resources:

1. https://www.geeksforgeeks.org/spectral-clustering-using-r/
2. https://rpubs.com/gargeejagtap/SpectralClustering
3. https://cran.r-project.org/web/packages/Spectrum/vignettes/Spectrum_vignette.pdf
4. https://search.r-project.org/CRAN/refmans/kernlab/html/specc.html
5. https://eranraviv.com/understanding-spectral-clustering/