MIS ASSIGNMENT 6
Performance comparison of clustering using K-Means and Spectral Clustering

Submitted by:

Anirudh Edpuganti (CB.EN.U4AIE20005)

Onteddu Chaitanya Reddy (CB.EN.U4AIE20045)

Pillalamarri Akshaya (CB.EN.U4AIE20049)

Pingali Sathvika (CB.EN.U4AIE20050)

Under the supervision of

Prof.Dr.V.Sowmya Mam

DEPT. OF COMPUTATIONAL ENGINEERING AND NETWORKING

AMRITA SCHOOL OF ENGINEERING

MAY 2022

ABSTRACT

Keywords - Non Convex , Clustering , K Means , Spectral Clustering , Laplacian

Clustering can be defined as grouping of data on the basis of similar characteristics of subsets of data from a dataset ,which can be used for distinguishing data groups as clusters. In modern era, applications generate data that only leads to accumulation of more and more data. As the quantity of data grows, the need of algorithms that are capable of analyzing the data without any presumptions increases. So, the algorithms should be designed in such a way that they are robust, scalable and capable of handling big data and not susceptible to the complexities of the datasets.

There are many Clustering algorithms in data mining that are very efficient at grouping data. Major issues with clustering problem is that algorithmic time complexity increases exponentially with respect to dataset size. In real world scenario, the data may have multivariate attributes, missing values, noise, non - convex sets etc.

So, in the proposed report, the performance of K-Means on skewed dataset is compared with that of Spectral clustering.

Contents

CHAPTER 1: INTRODUCTION **2**

1.1 Overview 2

1.2 Problem Formulation 2

1.3 Non-convex dataset 3

1.4 Codes for Datasets creation 4

1.4.1 Concentric Circles 4

1.4.2 Moons dataset 5

1.4.3 Intersecting Circles 6

CHAPTER 2: Algorithms **7**

2.1 Spectral Clustering 7

2.1.1 Objectives 7

2.1.2 Working 7

2.1.3 Implementation 9

2.2 K-Means Algorithm 11

2.2.1 Objectives 11

2.2.2 Working 12

2.2.3 Implementation 12

2.2.4 Applying K means to the generated data sets 14

CHAPTER 3: Results & Inference **15**

3.1 Results 15

3.2 Comparison of Performance 16

3.2.1 K-Means 16

3.2.2 Spectral Clustering 16

3.3 Conclusion 16

3.4 Contributions 17

Chapter 1

INTRODUCTION

1.1 Overview

The main aim of this assignment is to get familiarised with the K-means and spectral clustering algorithms and the concept of non convex datasets.

We will start by generating the datasets which are non-convex in nature and then we evaluate the results of both the algorithms and compare their performances.

1.2 Problem Formulation

Initially the samples of non-convex data sets need to be generated similar to concentric circles, intersecting circles, moon shaped (banana shaped) data sets.These generated data sets should then be tested with the K means clustering algorithm and Spectral clustering. In the K means clustering algorithm, the cluster centroids play a major role in categorising the data points. The process of taking centroids, calculating mean followed by distinguishing data points until no further changes takes place in the position of the centroid. For Spectral clustering, calculation of adjacency matrix and laplacian comes into picture. The dimensions are reduced by extracting the significant eigen values and stacking the corresponding eigen vectors into the matrix. Then, the evaluation is done on this matrix.

1.3 Non-convex dataset

Convexity of a dataset is established by the Euclidean space spanned by the dataset. If the line segment joining any 2 points of the data set goes outside the space, then it is called as Non-convex data set. This can be understood by considering the below 2 figures.

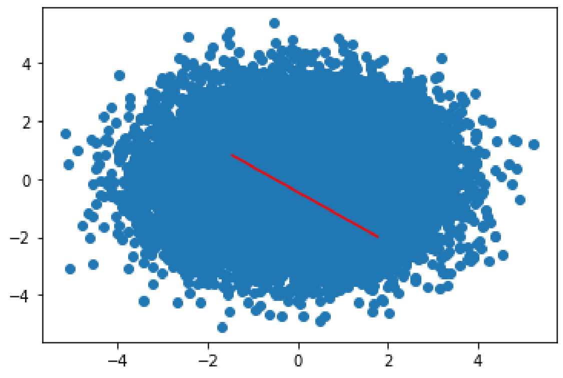


Figure 1.1: Convex Set

In Figure 1.1, the datapoints spanned in the euclidean space form a structure close to circle. All the datapoints here are inside this circular structure. So, the space inside the circle is considered to be the euclidean space. Now, 2 points can be considered randomly and can be joined. As shown in fig 1.1, 2 random points are joined with a red line. Whole line segment is lying inside the euclidean space and hence is defined as the convex set.

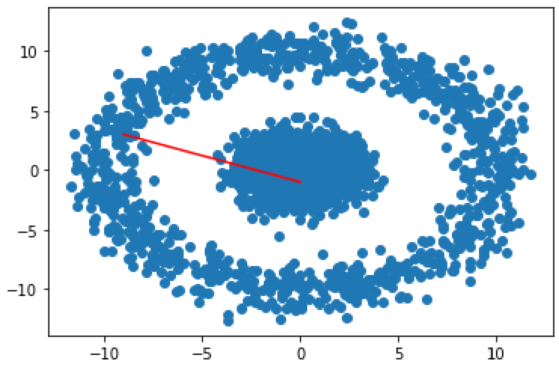


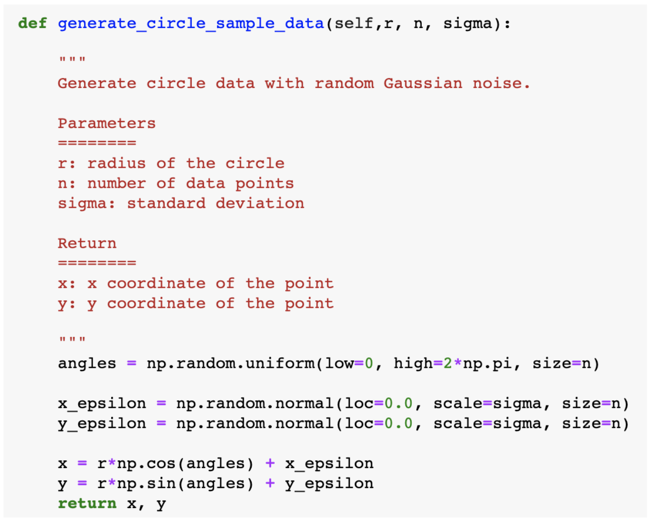
Figure 1.2: Non Convex Set

On the other hand, in the figure 1.2, the data points here also form a circular structure but it is clearly seen that the whole circle is not spanned. This seems more like a ring or in a 2D sense, this seems to be concentric circles. So, the area that is spanned by the data points, i.e , the blue region in fig 1.2 is the Euclidean space. Similar to the previous case, a line segment is drawn between 2 random points (in fig 1.2 with a red line) and the line segment is outside the euclidean space and hence considered as non-convex set.Important point to note is that, atleast 1 line segment coming out of the euclidean space can be considered as a non-convex set.

1.4 Codes for Datasets creation

1.4.1 Concentric Circles

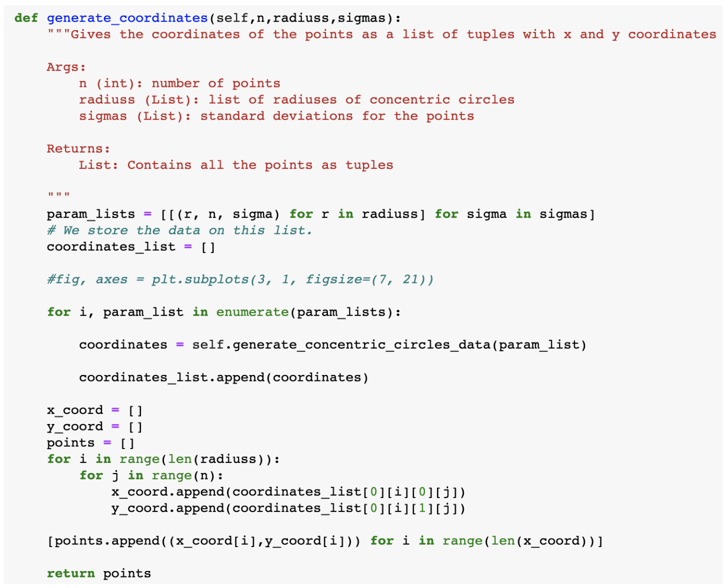
Concentric circles are generated as part of the non convex clusters. For generating concetric circles initially the data to form the circles is generated. Angles between 0 to 360 degrees are randomly generated.Then the epsilon values are randomly generated independently for x and y. Then the x and y coordinates of the circle are defined.Epsilon is added in order to produce deviation for generating concentric circles.The parameters required to get the final result are radius,number of data points for each specific radius and the value of sigma i.e deviation.



```python
def generate_circle_sample_data(self,r, n, sigma):

    """
    Generate circle data with random Gaussian noise.

    Parameters
    ========
    r: radius of the circle
    n: number of data points
    sigma: standard deviation

    Return
    ========
    x: x coordinate of the point
    y: y coordinate of the point

    """
    angles = np.random.uniform(low=0, high=2*np.pi, size=n)

    x_epsilon = np.random.normal(loc=0.0, scale=sigma, size=n)
    y_epsilon = np.random.normal(loc=0.0, scale=sigma, size=n)

    x = r*np.cos(angles) + x_epsilon
    y = r*np.sin(angles) + y_epsilon
    return x, y
```

Figure 1.3: Function defined to generate data

Then the data for the formation of concentric circles is generated.A list is created in order to append the values of coordinates of points for each specific radius.The generated concentric circles have coordinates which are appended in the form of list of tuples to the respective radius.In the nested for loop, for each specific radius, the code iterates n times(where n is the number of data points)for getting the coordinates of data points for each specific radius.Finally the x and y coordinates are appended in tuples and the points are returned.



```
def generate_coordinates(self,n,radiuss,sigmas):
    """Gives the coordinates of the points as a list of tuples with x and y coordinates

    Args:
        n (int): number of points
        radiuss (List): list of radiuses of concentric circles
        sigmas (List): standard deviations for the points

    Returns:
        List: Contains all the points as tuples

    """
    param_lists = [[(r, n, sigma) for r in radiuss] for sigma in sigmas]
    # We store the data on this list.
    coordinates_list = []

    #fig, axes = plt.subplots(3, 1, figsize=(7, 21))

    for i, param_list in enumerate(param_lists):

        coordinates = self.generate_concentric_circles_data(param_list)

        coordinates_list.append(coordinates)

    x_coord = []
    y_coord = []
    points = []
    for i in range(len(radiuss)):
        for j in range(n):
            x_coord.append(coordinates_list[0][i][0][j])
            y_coord.append(coordinates_list[0][i][1][j])

    [points.append((x_coord[i],y_coord[i])) for i in range(len(x_coord))]

    return points
```

Figure 1.4: Function defined to get coordinates

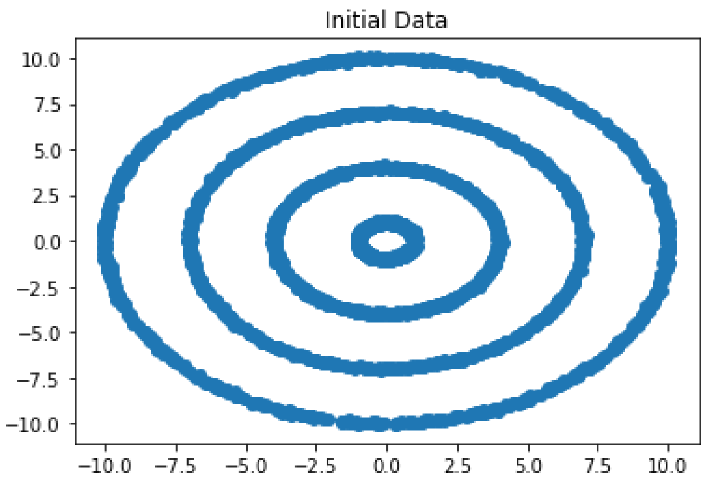
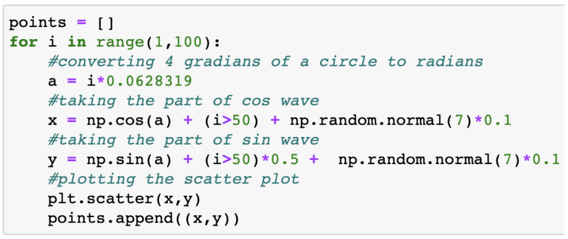


Figure 1.5: Intial data for concentric circles

1.4.2 Moons dataset

In order to create moons data set the upper part of the sin wave and lower part of the cos wave added with noise are generated.Initially the gradians of circle is converted to radians.Then the cos wave added with random noise is produced as one of the wave and sin wave added with random noise is generated as the next wave.Then the scatter plot is generated.This creates the moon data set.



```
points = []
for i in range(1,100):
    #converting 4 gradians of a circle to radians
    a = i*0.0628319
    #taking the part of cos wave
    x = np.cos(a) + (i>50) + np.random.normal(7)*0.1
    #taking the part of sin wave
    y = np.sin(a) + (i>50)*0.5 +  np.random.normal(7)*0.1
    #plotting the scatter plot
    plt.scatter(x,y)
    points.append((x,y))
```

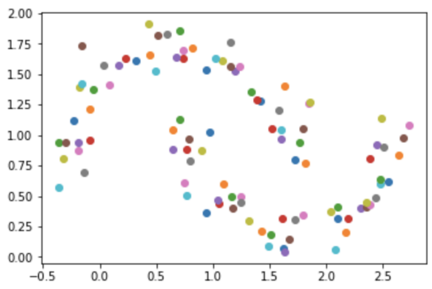
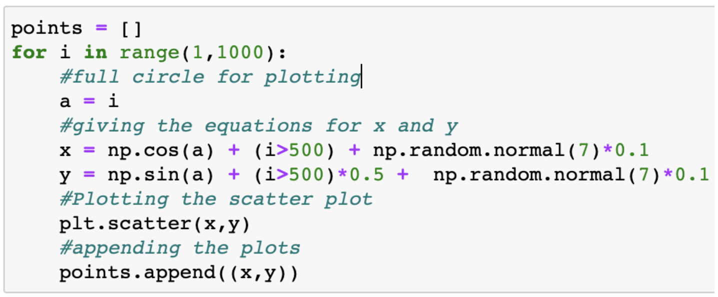


Figure 1.6: Moon datasets

1.4.3 Intersecting Circles

In order to create intersecting circles we follow the same procedure as above but produce more data points to fill up the other half portions.



```
points = []
for i in range(1,1000):
    #full circle for plotting
    a = i
    #giving the equations for x and y
    x = np.cos(a) + (i>500) + np.random.normal(7)*0.1
    y = np.sin(a) + (i>500)*0.5 +  np.random.normal(7)*0.1
    #Plotting the scatter plot
    plt.scatter(x,y)
    #appending the plots
    points.append((x,y))
```

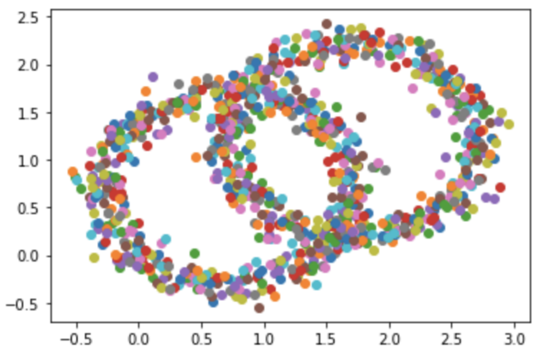


Figure 1.7: Intersecting Circles

Chapter 2

Algorithms

2.1 Spectral Clustering

2.1.1 Objectives

- It turns the clustering problem into a graph partitioning problem by treating each data point as a graph node.

2.1.2 Working

This problem involves 3 fundamental steps .

1. Adjacency matrix:

An adjacency matrix for the similarity graph is created which is represented by A. It is a square matrix with size $n \times n$. A_{ij} includes w_{ij} which is the weight of edge joining (i,j). It will be zero when there is no edge between (i,j).The basic understanding of adjacency matrix is that in an adjacency matrix, the row and column indices represent the nodes and the entries indicate whether there is an edge between the nodes or not.
The adjacency matrix can be built in the following manner:

K-Nearest Neighbours

A parameter k is set before.Then an edge is directed from u to v from two vertices u and v only if v is one of u's k-nearest neighbours.This results in the construction of a weighted and directed graph because it's not always the case that each u has v as one of its k-nearest

neighbours,and vice-versa.To make graph undirected the methods which are used are:
1.If either v or u is among the k-nearest neighbours of u OR v is among the k-nearest neighbours of v, direct an edge from u to v.
2.If v is among u's k-nearest neighbours AND u is among v's k-nearest neighbours, direct an edge from u to v and from v to u.

2. Projecting the data onto Lower dimensional space

There is a possibility that members of the same cluster may be far apart in the given dimension space, so the data is projected onto lower dimensional space.Thus the dimensional space is reduced so that the points will be closer.It is done by computing Graph Laplacian Matrix.
To compute it first we should find the degree of the node.
The degree is defined as:

$$d_i = \Sigma^n_{j=1|(i,j)\in E^{w_{ij}}}$$

Note that w_{ij} is the edge between the nodes i and j as defined earlier in adjacency matrix.
The degree matrix is defined as :
A Degree Matrix is a diagonal matrix in which the number of edges connecting to a node of the diagonal determines its degree. The degree of the nodes can alternatively be calculated by adding the total of each row in the adjacency matrix.

$$D_{ij} = \begin{cases} d_i, i = j \ \ \& \ \ 0, i \neq j \end{cases}$$

The Graph Laplacian Matrix is defined as:

$$L = D - A$$

For mathematical efficiency, this Matrix is then normalized. To minimize the dimensions, the eigenvalues and eigenvectors must first be determined. If there are k clusters, the first k eigenvalues and their corresponding eigen-vectors are taken and stacked into a matrix such that eigen-vectors are the columns of the matrix.

Eigenvalues of Graph Laplacian

When graph is completely disconnected then our eigenvalues are 0. As we add add edges ,some of our eigenvalues increase. Because there is only one connected component, the initial eigenvalue is 0. The values of the relevant eigenvector will always be constant.
First nonzero eigenvalue is called spectral gap.The spectral gap offers us an idea of the graph's density.
Second eigenvalue is called Fiedler value,and the Fiedler vector is the corresponding vector. The Fiedler value is an approximation of the minimum graph cut required to divide a graph into two linked components. Remember that the Fiedler value would be 0 if our graph already had two connected components. Each value in the Fiedler vector indicates which side of the cut each node belongs to.

Eigenvectors of Graph Laplacian

The matrix A can be thought of as a function that maps vectors to new vectors. When A is applied to most vectors, they will end up somewhere completely different, while eigenvectors will just change in magnitude. If you draw a line between the origin and the eigenvector, the eigenvector will still land on the line after the mapping. The magnitude with which the vector is scaled along the line is determined by λ.

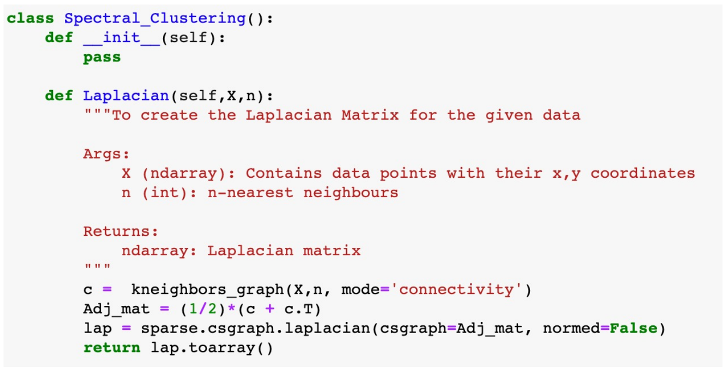
3. Clustering the data

This process mainly involves clustering the reduced data by using any traditional clustering technique – usually K-Means Clustering. First, each node is assigned a row of the normalized of the Graph Laplacian Matrix. Then the data is clustered using any traditional technique. To transform the clustering result, the node identifier is retained.

2.1.3 Implementation

We write all the functions inside a class. So, we start with creating a laplacian matrix for the non convex data generated. We pass the data and the value of k (for k nearest neighbours) and get the laplacian matrix for that data.

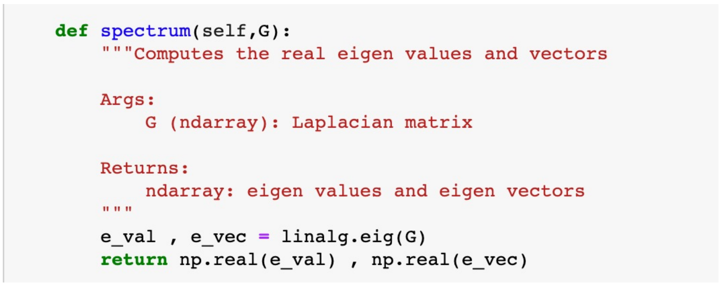
We then take this generated laplacian matrix and compute its eigen values and eigen



```python
class Spectral_Clustering():
    def __init__(self):
        pass

    def Laplacian(self,X,n):
        """To create the Laplacian Matrix for the given data

        Args:
            X (ndarray): Contains data points with their x,y coordinates
            n (int): n-nearest neighbours

        Returns:
            ndarray: Laplacian matrix
        """
        c =  kneighbors_graph(X,n, mode='connectivity')
        Adj_mat = (1/2)*(c + c.T)
        lap = sparse.csgraph.laplacian(csgraph=Adj_mat, normed=False)
        return lap.toarray()
```

Figure 2.1: Genrating Laplacian Matrix

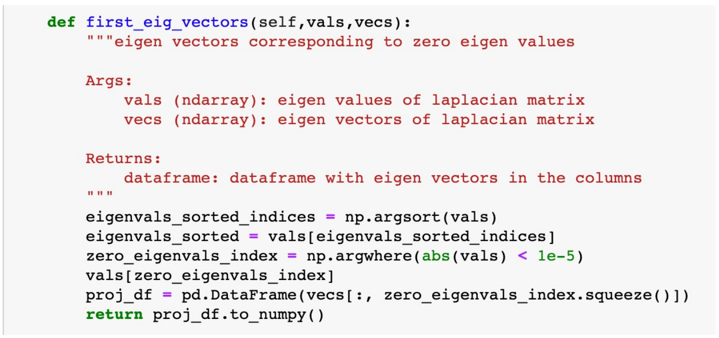
vectors inorder to implement the idea of projecting this data to lower dimensionality space. And hence, we return the eigen values and eigen vectors for the laplacian matrix



```python
    def spectrum(self,G):
        """Computes the real eigen values and vectors

        Args:
            G (ndarray): Laplacian matrix

        Returns:
            ndarray: eigen values and eigen vectors
        """
        e_val , e_vec = linalg.eig(G)
        return np.real(e_val) , np.real(e_vec)
```

Figure 2.2: Eigen values and Eigen vectors of Laplacian Matrix

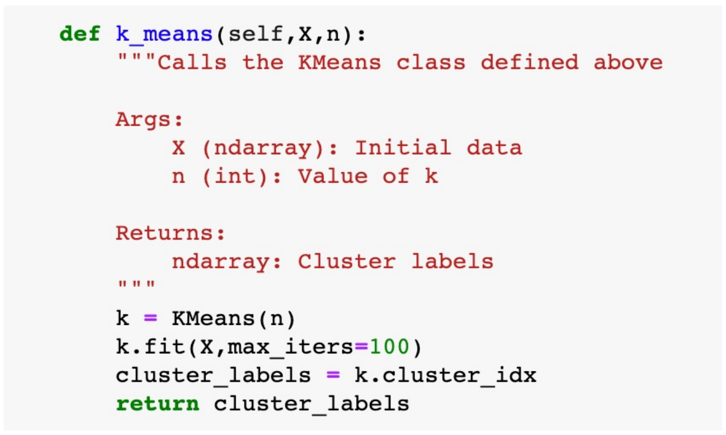
Then we consider the k eigen values where k corresponds to the number of clusters in the data. So, we sort the eigen values in the ascending order of theor value and consider the first k eigen values. Then we take the eigen vectors corresponding to these eigen values and stack them up in a matrix where the columns are the eigen vectors.



```python
    def first_eig_vectors(self,vals,vecs):
        """eigen vectors corresponding to zero eigen values

        Args:
            vals (ndarray): eigen values of laplacian matrix
            vecs (ndarray): eigen vectors of laplacian matrix

        Returns:
            dataframe: dataframe with eigen vectors in the columns
        """
        eigenvals_sorted_indices = np.argsort(vals)
        eigenvals_sorted = vals[eigenvals_sorted_indices]
        zero_eigenvals_index = np.argwhere(abs(vals) < 1e-5)
        vals[zero_eigenvals_index]
        proj_df = pd.DataFrame(vecs[:, zero_eigenvals_index.squeeze()])
        return proj_df.to_numpy()
```

Figure 2.3: Reduced dimension

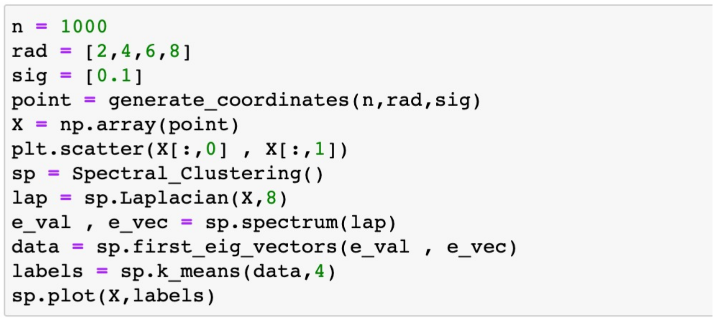
Then, we use the algorithm of K means to pass this reduced dimensional data and return the labels. Here, we use the K means implemented by us as another class.



```python
def k_means(self,X,n):
    """Calls the KMeans class defined above

    Args:
        X (ndarray): Initial data
        n (int): Value of k

    Returns:
        ndarray: Cluster labels
    """
    k = KMeans(n)
    k.fit(X,max_iters=100)
    cluster_labels = k.cluster_idx
    return cluster_labels
```

Figure 2.4: Passing data to K means

Finally, we take the labels produced by K means and the data projected to the lower dimensional space. We finally change this numpy data to a dataframe and then we scatter plot it to visualize.



```python
n = 1000
rad = [2,4,6,8]
sig = [0.1]
point = generate_coordinates(n,rad,sig)
X = np.array(point)
plt.scatter(X[:,0] , X[:,1])
sp = Spectral_Clustering()
lap = sp.Laplacian(X,8)
e_val , e_vec = sp.spectrum(lap)
data = sp.first_eig_vectors(e_val , e_vec)
labels = sp.k_means(data,4)
sp.plot(X,labels)
```

Figure 2.5: Passing data to our spectral clustering algorithm

Finally, all the results produced by the algorithm are shown in the results section.

2.2 K-Means Algorithm

2.2.1 Objectives

- K-Means is an unsupervised machine learning algorithm. It is an iterative algorithm that groups an unlabeled dataset into different clusters. We use this to cluster non convex data in this report.

2.2.2 Working

Consider an unlabeled dataset and assume some random k-number of points as cluster centroids. Then we start by calculating the euclidean distance between every datapoint and the cluster centroids and we assign each of the datapoint to the cluster whose cluster centroid is the nearest to the datapoint. Now we calculate the average of all the datapoints that belongs to a particular cluster and update the cluster centroid to that point. Finally we repeat the same process again until the euclidean distance between previous centroid and this centroid becomes 0. This results in no movement of of the centroids hence becoming the criterion to stop the iterations.

The best way to choose the optimal number of clusters is Elbow point.

Elbow point

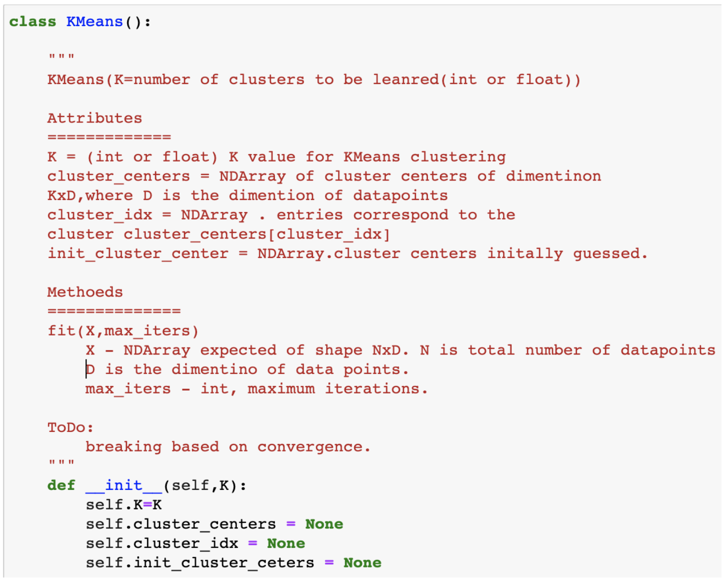
The elbow approach works by doing K-means clustering on the data set, where 'k' refers to the number of clusters.
The total of the squared distances between each member of the cluster and its centroid is known as the within cluster sum of squares (WCSS).

$$WCSS = \Sigma_{i=1}^{m}(x_i - c_i)^2$$

where x_i=data point and c_i=closest point to centroid
The WCSS is measured for each value of k.To find the optimal value of clusters, the elbow method follows the below steps.It executes the K-means clustering on a given dataset for different K values. For each value of K, calculates the WCSS value.Plots a curve between calculated WCSS values and the number of clusters K.The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

2.2.3 Implementation

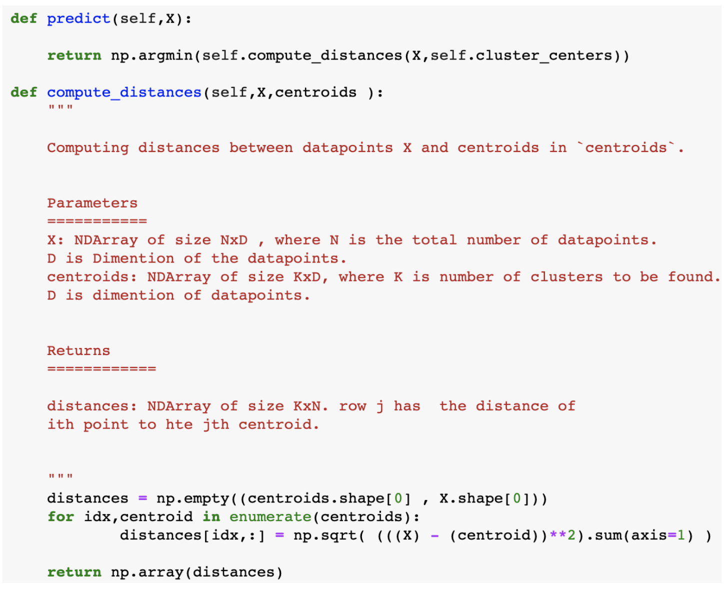
First all the necessary libraries are imported.Then the constructor is defined to give the value of k.The updated centroids every time are stored in cluster centres.Cluster indexes stores the cluster labels.Then the variables are initialised.



```python
class KMeans():

    """
    KMeans(K=number of clusters to be leanred(int or float))

    Attributes
    =============
    K = (int or float) K value for KMeans clustering
    cluster_centers = NDArray of cluster centers of dimentinon
    KxD,where D is the dimention of datapoints
    cluster_idx = NDArray . entries correspond to the
    cluster cluster_centers[cluster_idx]
    init_cluster_center = NDArray.cluster centers initally guessed.

    Methoeds
    ==============
    fit(X,max_iters)
        X - NDArray expected of shape NxD. N is total number of datapoints
        D is the dimentino of data points.
        max_iters - int, maximum iterations.

    ToDo:
        breaking based on convergence.
    """
    def __init__(self,K):
        self.K=K
        self.cluster_centers = None
        self.cluster_idx = None
        self.init_cluster_ceters = None
```

Figure 2.6: Constructor

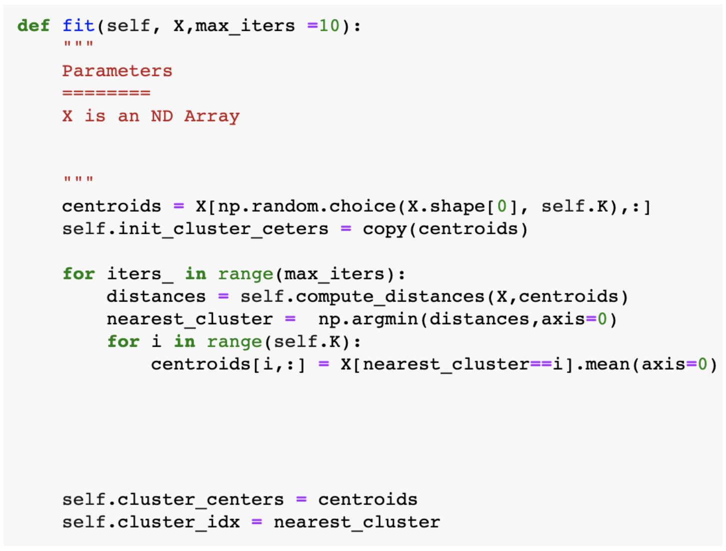
In order to compute distance from each data point to each of the randomly assumed centroids initially we have different distance metrics.Here euclidean distance is used for the calculation.In the predict function each data point is categorised into a particular label.The data point will be in the label that has minimum distance.



```python
def predict(self,X):

    return np.argmin(self.compute_distances(X,self.cluster_centers))

def compute_distances(self,X,centroids ):
    """

    Computing distances between datapoints X and centroids in `centroids`.


    Parameters
    ===========
    X: NDArray of size NxD , where N is the total number of datapoints.
    D is Dimention of the datapoints.
    centroids: NDArray of size KxD, where K is number of clusters to be found.
    D is dimention of datapoints.


    Returns
    ============

    distances: NDArray of size KxN. row j has  the distance of
    ith point to hte jth centroid.


    """
    distances = np.empty((centroids.shape[0] , X.shape[0]))
    for idx,centroid in enumerate(centroids):
            distances[idx,:] = np.sqrt( (((X) - (centroid))**2).sum(axis=1) )

    return np.array(distances)
```

Figure 2.7: predict and Computing distances

In the fit function we take random centres into consideration and copy them into initial clusters and calculates the distance for n iterations.Then the data point categorises into nearest centroid.

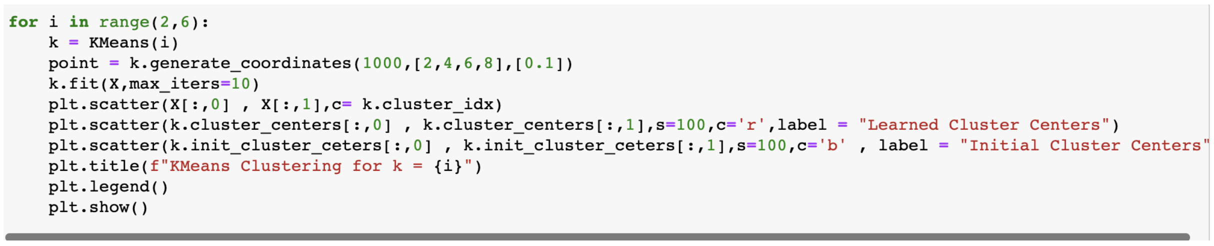


```python
def fit(self, X,max_iters =10):
    """
    Parameters
    ========
    X is an ND Array


    """
    centroids = X[np.random.choice(X.shape[0], self.K),:]
    self.init_cluster_ceters = copy(centroids)

    for iters_ in range(max_iters):
        distances = self.compute_distances(X,centroids)
        nearest_cluster =  np.argmin(distances,axis=0)
        for i in range(self.K):
            centroids[i,:] = X[nearest_cluster==i].mean(axis=0)




    self.cluster_centers = centroids
    self.cluster_idx = nearest_cluster
```

Figure 2.8: fitting the function

2.2.4 Applying K means to the generated data sets

Now the k means algorithm implemented is applied to the non convex clusters that are generated.



```python
for i in range(2,6):
    k = KMeans(i)
    point = k.generate_coordinates(1000,[2,4,6,8],[0.1])
    k.fit(X,max_iters=10)
    plt.scatter(X[:,0] , X[:,1],c= k.cluster_idx)
    plt.scatter(k.cluster_centers[:,0] , k.cluster_centers[:,1],s=100,c='r',label = "Learned Cluster Centers")
    plt.scatter(k.init_cluster_ceters[:,0] , k.init_cluster_ceters[:,1],s=100,c='b' , label = "Initial Cluster Centers"
    plt.title(f"KMeans Clustering for k = {i}")
    plt.legend()
    plt.show()
```

Figure 2.9: K means to the dataset

Chapter 3

Results & Inference

3.1 Results

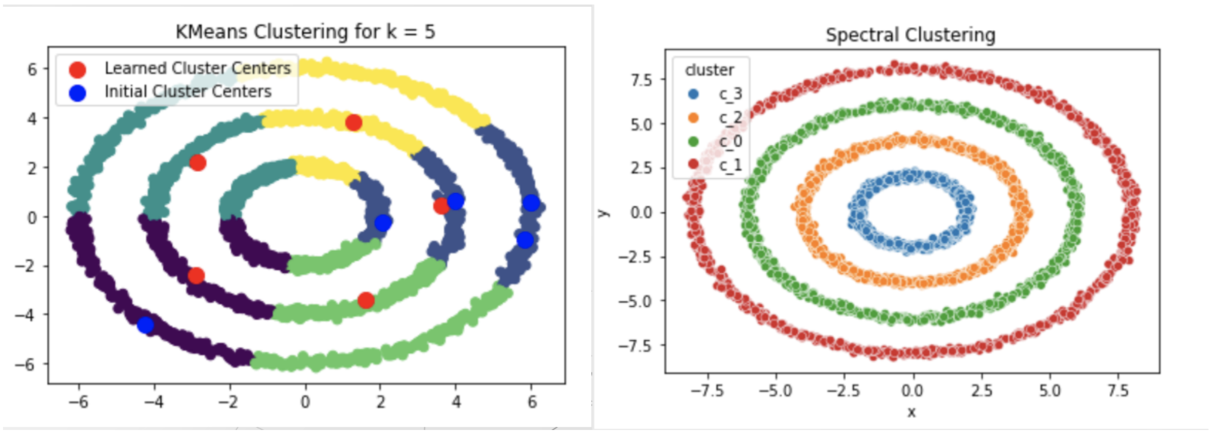


Figure 3.1: Clustering using K means vs Clustering using Spectral Clustering

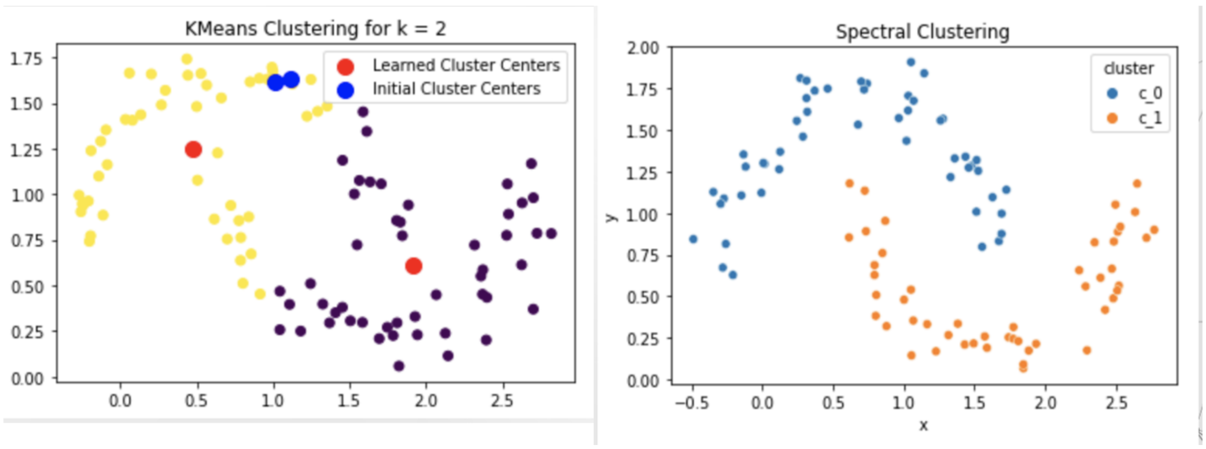


Figure 3.2: Clustering using K means vs Clustering using Spectral Clustering

3.2 Comparison of Performance

3.2.1 K-Means

Even when the true number of clusters K is known to the algorithm, K-means is failing to cluster the above data successfully. This is because K-means is a good data clustering algorithm for finding globular groups where all cluster members are close to each other (in the Euclidean sense).

3.2.2 Spectral Clustering

Graph-clustering approaches such as spectral clustering, on the other hand, do not cluster data points directly in their native data space but instead build a similarity matrix with the $(i, j)^{th}$row representing some similarity distance between the i^{th} and j^{th} data points in your dataset.

3.3 Conclusion

In some ways, spectral clustering is more general (and powerful) than K-means since spectral clustering is applicable whenever K-means is not (just use a simple Euclidean distance as the similarity measure).
However, the opposite is not true. When choosing one of these strategies over the other, there are some practical concerns to keep in mind. The input data matrix is factorized with K-means, whereas the Laplacian matrix is factorized with spectral clustering (a matrix derived from the similarity matrix).

3.4 Contributions

Anirudh Edpuganti - *Implementation of spectral clustering and Kmeans simulation*

Chaitanya Reddy - *Implementation of spectral clustering and Kmeans simulation*

Pillalamarri Akshaya - *Non convex datasets / Implemetation of Kmeans in python*

Pingali Sathvika - *Non convex datasets / Implemetation of Kmeans in python*