Spectral Clustering

Computational linear algebra for large scale problems

02TWYSM

Fatemeh Ahmadvand

s301384

2022-2023

spectral-clustering

February 9, 2023

Computational Linear Algebra for Large Scale Problems

Spectral Clustering (HW_SC)

Fatemeh Ahmadvand

Student Number: s301384

Email: fatemeh.ahmadvand@studenti.polito.it

1 Introduction

Clustering is a popular unsupervised learning technique. The grouping is such that points in a cluster are similar to each other, and less similar to points in other clusters.There are two broad approaches for clustering:

1. Compactness — Points that lie close to each other fall in the same cluster and are compact around the cluster center. The closeness can be measured by the distance between the observations. E.g.: **K-Means Clustering**

2. Connectivity — Points that are connected or immediately next to each other are put in the same cluster. Even if the distance between 2 points is less, if they are not connected, they are not clustered together. **Spectral clustering** is a technique that follows this approach.

2 Project Description

To perform a spectral clustering I need three main steps:

step (1)-Create a similarity graph between our N objects to cluster. In order to represent the data, I first construct an undirected graph $G = (V, E)$ with vertex sets $V = v_1, ..., v_n$. This can be represented by an adjacency matrix which has the similarity between each vertex as its elements . In the similarity graph each vertex $v_i \in V$ represents a data point X_i. An edge between two vertices v_i and v_j exists if the similarity s_{ij} between the corresponding data points X_i and X_j is either positive or larger than a certain threshold (minimum similarity value ϵ for a connection to take place between two data points). I assume that $s_{ij} = s_{ji}$ and that the edge connecting v_i and v_j is weighted by s_{ij} . Consequently, it turns out that the similarity graph is undirected. The weighted adjacency matrix is defined as $W_{ij} = s_{ij}$, if $i = j$ and $W_{ij} = 0$, if $i = j$. One way to compute a similarity graph is to create *Fully connected graph*. I simply connect all points and weight all edges based on their similarity s_{ij} and apply similarity functions such as the Gaussian similarity function to model the local neighborhood relationships in this graph.

$$s_{i,j} = exp\left(\frac{||X_i, X_j||^2}{2\sigma^2}\right)$$

the parameter σ controls the width of the neighborhoods. ***(Question 1)***

step (2)-Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object. I want to transform the space so that when two points are close together, they always belong to the same cluster, and when they are far away, they belong to distinct clusters. My observations must be projected into a low-dimensional space. I calculate the Graph Laplacian matrices for this.This is the main tool for spectral clustering.

$$L = D - W$$

where L is Laplacian matrix, D is the degree matrix and W is the adjacency matrix of the graph.The Laplacian's diagonal is the degree of our nodes, and the off diagonal is the negative edge weights. This is the representation I am after for performing spectral clustering.***(Question 2)***

To determine how many connected components there are in the similarity graph, I compute the cosine similarity.***(Question 3)***

$$CosineSimilarity := cos(\theta) = \frac{a_i.b_i}{\|a_i\|.\|b_i\|}$$

The goal of computing the Graph Laplacian L is to find eigenvalues and eigenvectors for it so that the data points can be embedded in a low-dimensional space. So in this step, I can go ahead and calculate eigenvalues.

$$L\lambda = \lambda v$$

where v is the eigenvector of L corresponding to eigenvalue λ. Thus I get eigenvalues {$ _1, _2, …, _n $} , where $0 = _1 _2 … _n $ (*Spectrum of the Laplacian*) and eigenvectors $\{v_1, v_2, ..., v_n\}$.

The eigenvalues of the Graph Laplacian can then be used to find the best number of clusters, and the eigenvectors can be used to find the actual cluster labels.***(Question 4)***

step (3)-Run k-means on these features to separate objects into k classes.

The final three questions, which are used to implement the three stages of K-means, are as follows: - creating a matrix $U \in \mathbb{R}^{N\times M}$ with these vectors as columns by computing the M eigenvectors $u_1, ..., u_M \in \mathbb{R}^N$ that correspond to the M smallest eigenvalues of the Laplacian matrix.***(Question 5)***

- For $i = 1, ..., N$ let $y_i \in \mathbb{R}^M$ be the vector corresponding to the i-th row of U. Clustering the points y_i , $i = 1, ..., N$ in $\mathbb{R}^M$ with the k-means algorithm into clusters $C_1, ..., C_M$.***(Question 6)***
- Assigning the original points in X to the same clusters as their corresponding rows in U and construct the clusters $A_1, ..., A_M$, with $A_i = x_j : y_j \in C_i$.***(Question 7)***

After applying thses three k-means steps, I plot the clusters of points with different colors.***(Question 8)***

I use two methodes for clustering, first I apply *k-means* and then *Spectral Clustering* ***(Question 9)***

As you can see in the K-means plots, the dataset clustered into 3 clusters perfectly.

I have two datasets named *Circle.csv* and *Spiral.csv* contain totaling N points that I will be using. The x-values and y-values of the points are listed in two columns in the Circle dataset, and the index of the correct cluster is listed in the third column of the Spiral dataset.

I do the nine tasks for *Cricle* dataset and *Spiral* dataset with different values of $K = 10, 20, 40$, $\sigma = 1$ seperately as below.

3 Import library

First of all, I start with the importation of the modules.

[1]:
```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import seaborn as sns
from sklearn.cluster import SpectralClustering, KMeans
from sklearn.metrics import pairwise_distances
from matplotlib import pyplot as plt
```

For the entire notebook, I define the decimal number up to three decimal places.

[2]:
```
float_formatter = lambda x: "%.3f" % x
np.set_printoptions(formatter={'float_kind':float_formatter})
#from sklearn.datasets.samples_generator import make_circles
sns.set()
```

4 load the *Circle* Dataset (CD)

[3]:
```
df=pd.read_csv('D:/HW2/Circle.csv',names=["a", "b"], encoding='latin-1')
df
```

[3]:

	a	b
0	2.7147	0.81472
1	2.8054	0.94572
2	2.0253	0.20682
3	2.8096	1.03310
4	2.5257	0.79188
..	…	…
895	5.8968	-4.19650
896	3.5603	-8.78640
897	7.0303	-8.00040

```
898   5.4921 -5.49120
899   8.3419 -6.80190

[900 rows x 2 columns]
```

4.1 $k = 10$ in Circle dataset

Question1 ($k = 10$, CD)

[4]:
```
#construct the k-nearest neighborhood similarity graph and its adjacency matrix
from sklearn.neighbors import radius_neighbors_graph
W = radius_neighbors_graph(df,1,mode='distance', metric='minkowski', p=10,␣
 ↪metric_params=None, include_self=False) # p=k=10
W = W.toarray()
print("The adjacency matrix is: ")
print(W)
```

```
The adjacency matrix is:
[[0.000 0.131 0.707 … 0.000 0.000 0.000]
 [0.131 0.000 0.817 … 0.000 0.000 0.000]
 [0.707 0.817 0.000 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 0.000]]
```

[5]:
```
#Constructing the degree matrix ( )
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[32.094 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 28.044 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 28.075 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 15.605 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 48.593 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 6.244]]
```

Question2 ($k = 10$, CD)

[6]:
```
from scipy.sparse import csgraph

# with the function below we can get the Laplacian Matrix(L=D-W) from the␣
 ↪adjacency matrix we have from previous section
L = csgraph.laplacian(W, normed=False)

print('The Laplacian Matrix L is: ')
print(L)
```

```
print('The first 5 values of L are: ')
print(L[:5,:5])
```

```
The Laplacian Matrix L is:
[[32.094 -0.131 -0.707 … -0.000 -0.000 -0.000]
 [-0.131 28.044 -0.817 … -0.000 -0.000 -0.000]
 [-0.707 -0.817 28.075 … -0.000 -0.000 -0.000]
 …
 [-0.000 -0.000 -0.000 … 15.605 -0.000 -0.000]
 [-0.000 -0.000 -0.000 … -0.000 48.593 -0.000]
 [-0.000 -0.000 -0.000 … -0.000 -0.000 6.244]]
The first 5 values of L are:
[[32.094 -0.131 -0.707 -0.218 -0.189]
 [-0.131 28.044 -0.817 -0.087 -0.280]
 [-0.707 -0.817 28.075 -0.866 -0.596]
 [-0.218 -0.087 -0.866 27.103 -0.289]
 [-0.189 -0.280 -0.596 -0.289 33.225]]
```

Question3 ($k = 10$, **CD**)

[7]:
```
#compute the cosine similarity to determine how many connected components there␣
 ↪are in the similarity graph
# import required libraries
from numpy.linalg import norm

# compute  similarity
cosine = np.dot(df['a'],df['b'])/(norm(df['a'],)*norm(df['b']))
print("Cosine Similarity is : ", cosine)
```

```
Cosine Similarity is :  -0.7272658115521661
```

Question4 ($k = 10$, **CD**)

[8]:
```
e, v = np.linalg.eig(L)
# eigenvalues
print('eigenvalues:')
print(e)
# normalised eigenvectors
print('M eigenvectors:')
print(v)
```

```
eigenvalues:
[76.405 74.645 73.200 69.831 69.681 69.190 68.663 68.192 67.823 67.570
 67.187 66.872 66.493 66.518 66.202 65.596 65.629 65.055 64.794 64.638
 64.514 64.032 63.601 63.452 63.124 62.725 62.804 62.347 62.183 62.021
 61.866 61.380 61.245 61.069 60.888 60.728 60.407 60.315 60.063 59.538
 59.118 59.459 57.638 58.916 58.725 58.736 58.521 58.360 58.381 57.395
 57.121 57.046 56.703 55.935 55.231 55.740 55.639 54.714 54.400 54.217
```

53.984 0.852 53.561 53.290 53.393 0.008 53.049 52.819 52.630 52.490
52.326 52.437 52.026 51.901 0.000 -0.000 47.110 51.620 51.564 51.507
48.130 51.379 50.667 50.711 51.185 51.033 51.059 50.356 50.210 49.219
49.648 50.950 50.276 49.518 48.520 48.901 48.841 48.652 49.460 48.214
0.154 0.214 0.435 0.810 0.994 1.127 1.393 1.558 1.732 2.004 1.829 1.777
2.126 2.602 3.031 3.386 47.564 47.625 47.408 47.014 47.309 45.721 46.791
46.667 46.618 46.391 46.356 44.823 45.474 45.723 45.736 45.068 44.996
44.906 44.778 44.680 44.629 40.021 44.500 44.372 44.362 43.091 40.895
40.981 44.277 44.078 43.783 43.612 42.639 44.085 42.078 43.135 44.022
41.926 41.456 41.469 44.064 42.880 42.707 42.329 42.487 42.575 41.931
42.047 41.339 41.566 41.531 3.512 3.629 3.667 3.969 4.719 4.744 4.946
5.122 5.278 5.559 5.747 5.999 6.225 6.430 41.109 40.936 40.884 40.065
40.564 39.850 40.407 40.393 39.508 39.648 39.949 39.823 39.798 39.279
39.671 36.545 36.735 39.560 36.907 39.253 38.931 38.697 38.823 36.945
38.563 39.082 37.299 37.294 37.730 37.846 38.178 38.110 38.465 37.599
37.519 38.895 38.812 38.395 38.227 37.875 38.475 37.777 38.129 37.595
37.514 38.446 6.962 6.990 7.490 7.557 7.852 7.875 7.972 8.333 8.431 8.767
37.269 37.229 36.998 36.439 36.923 36.260 36.125 36.173 36.747 36.607
35.969 35.927 35.859 36.361 36.218 35.748 35.614 35.481 35.543 36.022
35.292 35.331 35.092 35.005 34.965 33.945 34.913 34.114 34.013 34.226
34.816 34.752 34.689 34.603 34.402 34.359 34.422 34.558 34.541 34.525
35.547 35.145 34.944 34.772 9.133 9.334 9.341 9.470 9.606 9.628 9.773
9.884 9.942 9.919 9.964 9.952 10.054 10.094 10.165 34.326 33.849 33.866
34.093 33.685 33.649 33.752 33.588 33.952 33.442 33.382 33.357 33.303
32.347 32.407 32.651 32.567 33.174 33.122 33.073 32.774 32.795 32.451
32.862 32.886 32.490 33.010 32.956 33.753 33.231 33.460 33.448 33.042
32.980 10.363 10.379 10.389 10.439 10.539 10.600 10.643 10.652 10.705
10.837 10.878 10.903 10.928 10.996 10.962 32.263 32.236 32.324 32.171
32.046 32.167 31.961 31.909 31.980 31.856 31.269 31.805 31.364 31.744
31.339 31.691 31.455 31.500 31.551 31.576 31.616 31.634 32.511 32.412
32.300 31.993 31.769 11.017 11.042 1.619 11.182 11.195 11.323 11.237
11.285 11.263 11.396 11.424 11.431 11.486 11.481 21.532 22.255 22.111
21.824 22.466 22.551 21.729 22.648 22.042 21.647 21.590 31.250 31.224
31.191 30.999 31.161 31.144 31.095 31.058 30.873 30.719 31.069 30.750
30.631 30.820 30.966 30.672 30.933 30.559 31.442 30.964 11.557 11.568
11.613 11.630 11.616 11.658 11.680 11.702 22.393 22.903 23.004 21.688
23.171 23.365 22.936 23.579 20.650 20.588 21.238 21.075 23.544 23.147
20.403 21.226 20.974 21.119 21.012 20.148 21.020 20.200 20.230 30.523
20.877 30.504 30.471 30.404 30.615 30.363 23.411 20.315 30.311 30.309
30.537 30.425 11.743 11.775 11.805 11.844 11.950 11.944 11.862 11.920
11.906 11.874 11.886 19.390 19.555 20.065 19.726 19.742 19.872 19.836
23.570 19.887 23.752 24.377 24.154 24.107 24.036 23.844 23.835 23.918
30.280 30.203 30.103 30.167 29.886 29.954 29.984 30.070 24.020 23.867
29.867 29.905 30.287 30.176 29.930 30.065 30.176 12.019 11.984 11.996
19.320 19.816 24.458 24.217 29.817 29.948 12.053 24.650 12.075 12.094
12.142 12.167 12.176 19.116 19.242 19.241 19.169 19.080 18.971 12.220
18.919 29.750 29.737 29.688 29.648 29.808 12.275 12.294 12.326 12.350
12.436 12.372 12.416 12.393 12.384 18.853 18.732 18.446 18.578 18.881

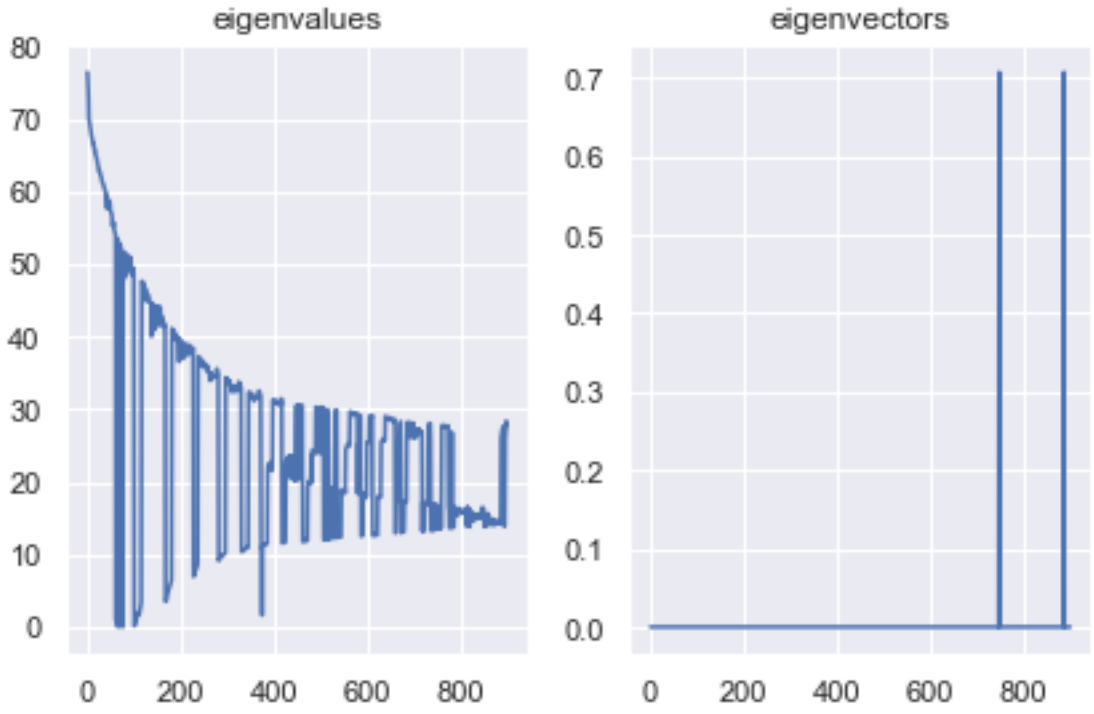
```
 18.777 18.502 18.792 18.507 24.450 24.606 24.720 25.028 25.071 24.971
 25.121 25.225 24.968 29.612 29.640 29.572 29.580 29.527 29.492 29.488
 29.224 29.146 29.369 29.250 29.233 29.444 29.336 29.299 29.320 29.393
 18.353 25.261 25.326 25.093 29.107 29.095 29.078 12.509 12.541 12.567
 12.581 18.361 18.278 18.200 18.178 18.286 17.719 18.121 18.104 18.002
 25.418 25.470 25.443 25.532 25.550 29.054 29.038 29.004 28.955 28.964
 28.910 29.139 12.589 12.603 12.669 12.696 12.637 12.724 12.756 12.861
 12.803 12.769 12.638 17.669 17.994 17.804 17.888 17.954 17.833 17.942
 25.333 25.615 25.667 25.686 25.873 25.821 25.783 25.941 25.737 28.991
 28.744 28.722 28.780 28.849 28.810 28.835 28.829 28.653 28.625 28.544
 28.815 28.638 28.414 28.377 28.356 28.393 28.505 28.530 28.515 28.606
 12.947 12.956 12.987 17.694 17.522 17.581 17.626 17.440 25.979 26.217
 26.162 28.295 28.265 12.965 13.024 13.043 13.103 13.115 13.164 13.135
 17.381 17.301 17.201 17.231 17.255 26.012 28.188 28.225 26.291 26.359
 26.346 28.121 26.441 26.107 26.239 26.496 28.088 28.033 27.958 28.057
 26.074 26.551 26.693 26.623 26.077 27.074 27.125 27.258 26.746 26.970
 27.223 26.911 27.187 26.591 26.798 26.826 26.884 26.864 13.200 17.157
 13.237 13.257 13.284 13.246 17.024 17.115 17.128 17.079 16.992 17.059
 16.976 27.313 27.832 27.936 27.925 28.052 25.753 13.276 16.931 16.902
 13.314 13.309 13.327 13.365 16.887 13.407 13.449 16.826 13.602 13.591
 13.470 13.486 13.547 13.502 16.739 16.636 16.657 16.721 13.514 16.705
 27.401 27.790 27.787 27.757 27.466 27.557 27.658 27.723 27.502 27.522
 27.576 27.443 27.684 27.705 16.538 16.512 13.653 13.688 13.629 13.619
 16.684 26.848 26.420 13.718 13.738 13.761 13.775 16.495 16.459 16.393
 16.360 15.324 15.343 15.397 15.390 15.456 16.326 16.309 16.030 16.067
 16.118 16.145 15.949 15.720 15.681 15.884 15.956 15.830 15.846 15.790
 15.755 16.183 16.168 13.889 13.872 16.572 13.967 13.858 16.242 13.810
 14.006 15.908 14.791 14.832 14.848 15.771 14.910 14.664 15.500 14.682
 15.022 15.077 15.054 15.526 14.964 15.251 15.146 15.202 15.175 15.288
 14.951 15.641 15.598 15.571 15.126 15.111 15.244 16.490 16.377 16.211
 16.256 13.807 15.536 13.842 14.033 14.090 14.107 14.659 15.613 14.279
 14.724 14.058 14.265 14.354 14.392 14.233 14.224 14.949 14.041 14.431
 14.445 14.587 14.140 14.047 14.719 14.343 14.176 14.198 14.511 14.492
 14.481 14.538 14.566 14.178 14.413 14.546 14.558 26.082 26.589 27.206
 27.357 27.527 27.628 16.278 13.841 14.185 26.059 26.894 28.348 27.891
 27.742]
M eigenvectors:
[[-0.000 -0.000 -0.000 … 0.000 -0.000 -0.000]
 [-0.000 0.000 0.000 … 0.000 0.000 -0.000]
 [0.000 -0.000 -0.000 … 0.000 0.000 -0.000]
 …
 [-0.004 0.001 0.004 … 0.020 -0.001 0.007]
 [0.050 0.050 0.037 … 0.011 0.018 0.022]
 [0.000 0.001 0.001 … -0.006 -0.001 -0.002]]
```

Question5 ($k = 10$, CD)

[9]:
```
# construct the matrix U
u=v+L
print("the U matrix is: ")
print(u)
```

```
the U matrix is:
[[32.094 -0.131 -0.707 … 0.000 -0.000 -0.000]
 [-0.131 28.044 -0.817 … 0.000 0.000 -0.000]
 [-0.707 -0.817 28.075 … 0.000 0.000 -0.000]
 …
 [-0.004 0.001 0.004 … 15.624 -0.001 0.007]
 [0.050 0.050 0.037 … 0.011 48.611 0.022]
 [0.000 0.001 0.001 … -0.006 -0.001 6.242]]
```

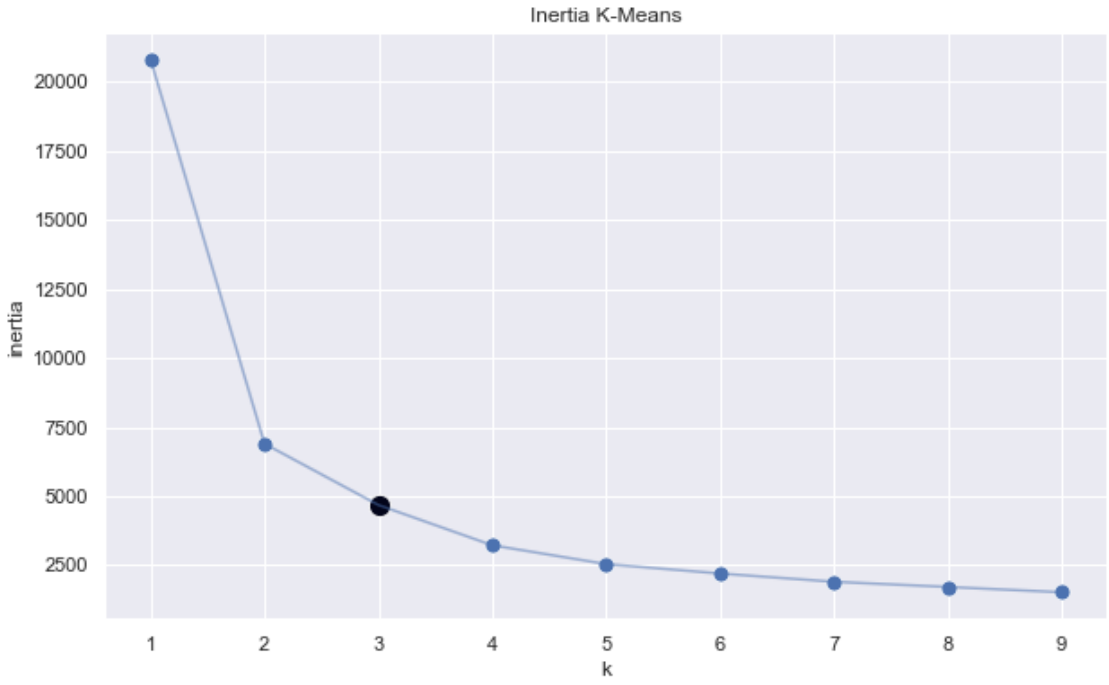
[10]:
```
# eigenvalues and eigen vector plot
fig = plt.figure()
ax1 = plt.subplot(121)
plt.plot(e)
ax1.title.set_text('eigenvalues')
i = np.where(e < 10e-6)[0]
ax2 = plt.subplot(122)
plt.plot(v[:, i[0]])
ax2.title.set_text('eigenvectors')

fig.tight_layout()
plt.show()
```



[11]:
```
df=pd.read_csv('D:/HW2/Circle.csv',names=["a", "b"], encoding='latin-1')
```

I run the k-means algorithm on the circle dataset to try to get some clusters. I select the number of cluster using the elbow method by considering the inertia (sum of squared distances of samples to their closest cluster center) as a function of the number of clusters.

[12]:
```
from sklearn.cluster import KMeans
import seaborn as sns
inertias = []

k_candidates = range(1, 10)

for k in k_candidates:
    k_means = KMeans(random_state=42, n_clusters=k)
    k_means.fit(df)
    inertias.append(k_means.inertia_)

fig, ax = plt.subplots(figsize=(10, 6))
sns.scatterplot(x=k_candidates, y = inertias, s=80, ax=ax)
sns.scatterplot(x=[k_candidates[2]], y = [inertias[2]], c=[3], s=150, ax=ax)
sns.lineplot(x=k_candidates, y = inertias, alpha=0.5, ax=ax)
ax.set(title='Inertia K-Means', ylabel='inertia', xlabel='k');
```



From this plot we see that $K = 3$ is a good choice. Now I get the clusters.

Question 6,7,8,9 ($k = 10$, CD)

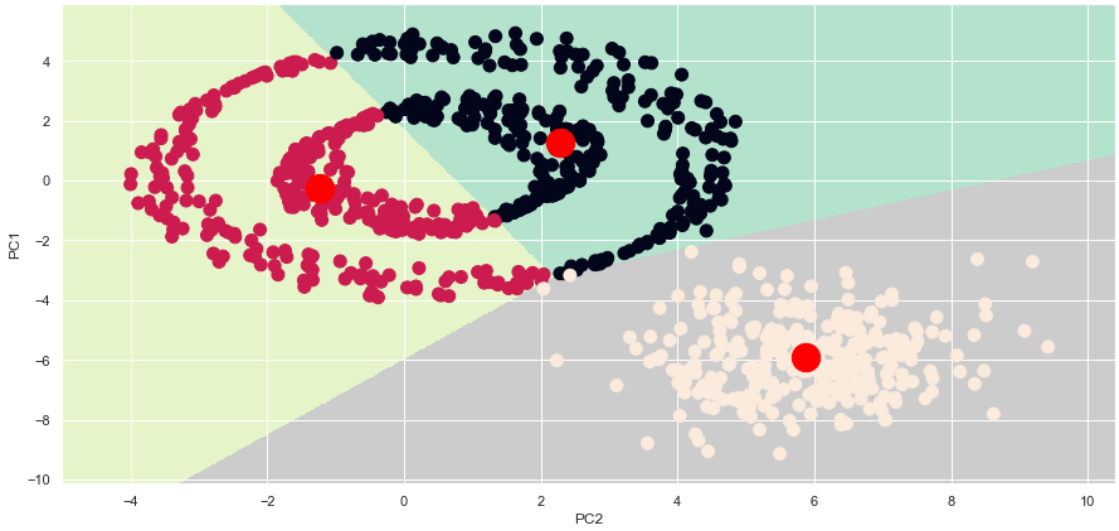
[13]:
```
#Compute the clusters for the Circle dataset with K-means method
algorithm = (KMeans(n_clusters = 3,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 50  , algorithm='elkan') )
algorithm.fit(df)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

[14]:
```
X1 = df[['a','b']].iloc[: , :].values
```

[15]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\base.py:451: UserWarning: X does not have valid feature names,
but KMeans was fitted with feature names
  "X does not have valid feature names, but"
```

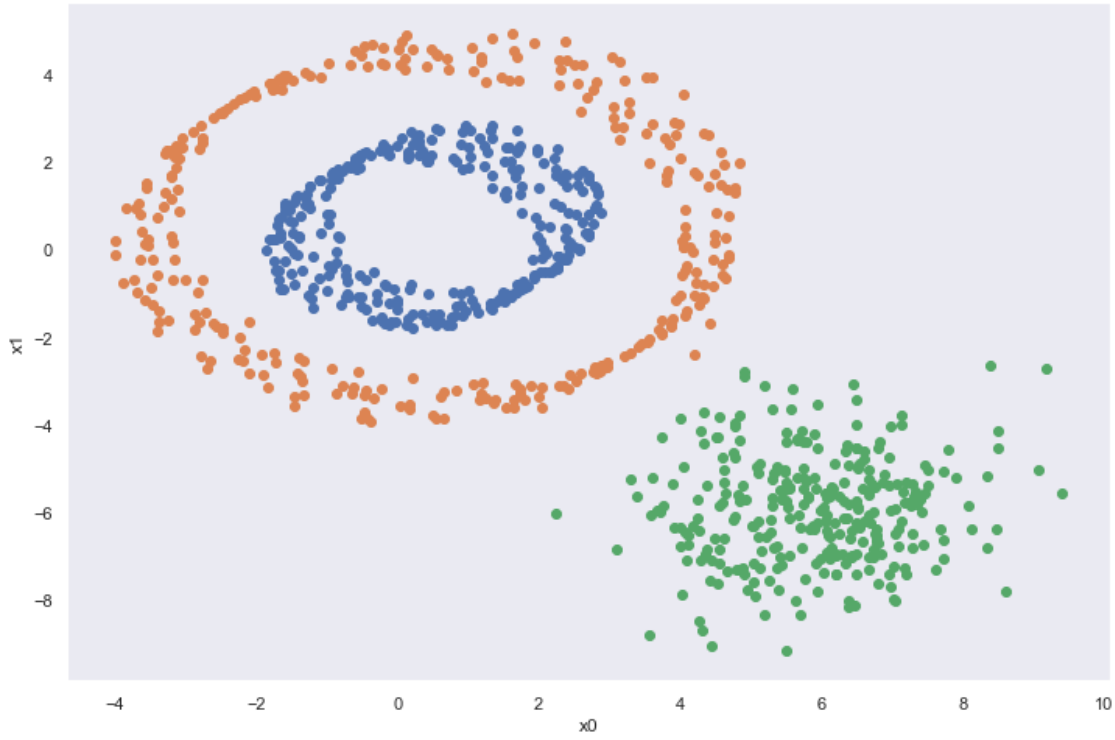
[16]:
```
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'a', y = 'b', data = df, c = labels1, s = 100)
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 500, c = 'red'␣
 ↪, alpha = 1)
plt.ylabel('PC1') , plt.xlabel('PC2')
plt.show()
```



[17]:
```
#Compute the clusters for the Circle dataset with clustering method
from sklearn.cluster import SpectralClustering
sine_dataset = pd.DataFrame(df)

## Fitting spectral clustering with 3 clusters
spectral_clustering = SpectralClustering(n_clusters = 3,
                                         affinity = 'nearest_neighbors',
                                         n_neighbors = 10).fit(df)

## Appending cluster to the sine dataset
sine_dataset['spectral_clusters'] = spectral_clustering.labels_


## Visualizing the data with cluster
colors = ['lightblue', 'orange']

plt.figure(figsize = (12, 8))
plt.xlabel('x0')
plt.ylabel('x1')
plt.grid()

for c in sine_dataset['spectral_clusters'].unique():

        temp = sine_dataset[sine_dataset['spectral_clusters'] == c]
        plt.scatter(temp['a'], temp['b'])
```

```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\manifold\_spectral_embedding.py:261: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
```

"Graph is not fully connected, spectral embedding may not work as expected."



4.2 $k = 20$ in Circle dataset

Question1 ($k = 20$, CD)

[18]:
```
#construct the k-nearest neighborhood similarity graph and its adjacency matrix
from sklearn.neighbors import radius_neighbors_graph
W = radius_neighbors_graph(df,1,mode='distance', metric='minkowski', p=20,␣
 ↪metric_params=None, include_self=False) # k=p=20
W = W.toarray()
print("The adjacency matrix is: ")
W
```

The adjacency matrix is:

[18]:
```
array([[0.000, 0.131, 0.692, …, 0.000, 0.000, 0.000],
       [0.131, 0.000, 0.792, …, 0.000, 0.000, 0.000],
       [0.692, 0.792, 0.000, …, 0.000, 0.000, 0.000],
       …,
       [0.000, 0.000, 0.000, …, 0.000, 0.000, 0.000],
       [0.000, 0.000, 0.000, …, 0.000, 0.000, 0.000],
       [0.000, 0.000, 0.000, …, 0.000, 0.000, 0.000]])
```

[19]:
```
#Constructing the degree matrix ( )
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[32.030 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 26.968 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 27.967 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 15.527 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 50.316 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 6.224]]
```

Question2 ($k = 20$, **CD**)

[20]:
```
from scipy.sparse import csgraph

# with the function below we can get the Laplacian Matrix(L=D-W) from the␣
 ↪adjacency matrix we have from previous section
L = csgraph.laplacian(W, normed=False)

print('The Laplacian Matrix L is: ')
print(L)

print('The first 5 values of L are: ')
print(L[:5,:5])
```

```
The Laplacian Matrix L is:
[[32.030 -0.131 -0.692 … -0.000 -0.000 -0.000]
 [-0.131 26.968 -0.792 … -0.000 -0.000 -0.000]
 [-0.692 -0.792 27.967 … -0.000 -0.000 -0.000]
 …
 [-0.000 -0.000 -0.000 … 15.527 -0.000 -0.000]
 [-0.000 -0.000 -0.000 … -0.000 50.316 -0.000]
 [-0.000 -0.000 -0.000 … -0.000 -0.000 6.224]]
The first 5 values of L are:
[[32.030 -0.131 -0.692 -0.218 -0.189]
 [-0.131 26.968 -0.792 -0.087 -0.280]
 [-0.692 -0.792 27.967 -0.839 -0.586]
 [-0.218 -0.087 -0.839 25.038 -0.284]
 [-0.189 -0.280 -0.586 -0.284 33.121]]
```

Question3 ($k = 20$, **CD**)

[21]:
```
#compute the cosine similarity to determine how many connected components there␣
 ↪are in the similarity graph
# import required libraries
from numpy.linalg import norm
```

```
# compute  similarity
cosine = np.dot(df['a'],df['b'])/(norm(df['a'],)*norm(df['b']))
print(" Similarity:", cosine)
```

```
Similarity: -0.7272658115521661
```

Question4 ($k = 20$, **CD**)

[22]:
```
e, v = np.linalg.eig(L)
# eigenvalues
print('eigenvalues:')
print(e)
# normalised eigenvectors
print('M eigenvectors:')
print(v)
```

```
eigenvalues:
[-0.000 2.129 2.257 7.401 8.816 48.232 47.265 14.992 45.819 16.325 44.724
 44.050 19.854 20.071 43.048 42.721 42.261 41.982 41.435 41.380 20.324
 20.844 41.105 40.885 21.136 40.491 21.315 40.162 21.571 40.035 21.464
 21.652 21.991 39.671 22.383 22.448 39.576 39.318 39.235 22.761 38.932
 38.918 38.641 38.609 38.316 38.225 38.138 37.911 37.755 37.515 37.579
 37.369 37.155 36.886 36.763 36.623 36.459 36.376 36.265 36.327 36.186
 36.009 35.956 34.511 35.916 35.865 35.693 35.401 35.571 35.510 35.756
 35.244 34.795 34.714 34.644 35.107 35.172 35.049 35.023 34.967 34.945
 23.300 23.283 23.853 23.924 24.080 24.326 24.370 24.532 24.690 24.745
 24.871 25.033 25.217 25.275 25.308 34.294 34.381 34.404 34.147 34.025
 33.896 33.840 33.777 33.708 33.665 33.656 33.590 33.405 33.475 33.513
 33.345 33.160 33.261 33.216 32.990 32.926 32.801 32.889 32.753 32.561
 32.623 32.667 32.484 31.892 32.395 32.378 32.199 32.232 32.451 31.986
 32.008 32.013 32.132 32.071 32.101 25.370 25.465 25.424 25.571 25.601
 25.673 25.772 25.895 25.943 25.914 26.014 26.065 26.125 26.152 26.201
 26.226 26.305 26.341 26.323 31.772 31.728 31.711 31.651 31.577 31.485
 31.568 31.452 31.394 31.375 31.316 31.251 31.206 31.138 31.061 31.076
 30.942 30.906 31.032 30.861 30.853 30.674 30.812 30.748 30.737 30.699
 26.369 26.452 26.511 26.542 26.606 26.582 26.668 26.662 26.719 26.710
 30.596 30.576 26.871 30.506 26.902 30.462 30.411 30.346 30.310 27.023
 30.264 27.051 27.068 27.082 27.118 30.256 30.224 30.201 30.169 27.304
 27.178 30.127 27.218 27.149 30.092 27.395 30.078 29.925 29.985 29.965
 30.030 30.032 27.204 29.667 29.939 29.750 29.803 29.809 29.820 29.785
 27.459 27.484 29.623 29.573 29.558 29.602 29.513 27.553 27.528 29.498
 29.483 27.516 27.609 29.442 29.452 27.584 27.893 27.689 27.626 29.400
 27.927 27.974 28.013 27.751 27.696 27.783 27.997 29.371 27.716 29.317
 29.348 28.098 28.125 28.071 29.291 29.258 28.249 28.196 29.219 29.182
 29.202 28.292 28.635 28.720 27.800 28.764 29.094 29.071 28.339 29.035
 28.451 28.186 28.850 28.703 29.000 27.788 28.962 28.516 28.529 28.926
 28.872 28.881 28.399 28.380 28.936 28.596 28.548 28.349 28.484 0.195
 0.223 0.696 0.976 1.640 1.990 2.867 3.429 -0.000 4.773 5.197 5.991 6.985
```

7.483 7.829 8.358 0.000 0.000 25.912 9.052 24.077 9.484 9.553 23.360
9.742 22.582 22.221 9.847 9.894 9.925 10.009 10.063 21.827 21.586 21.414
10.287 10.345 10.384 10.413 21.143 10.482 21.055 20.734 20.597 10.607
10.650 10.575 20.405 20.459 20.107 19.721 19.764 19.938 19.883 19.871
19.654 19.511 19.447 19.357 19.275 19.141 19.043 19.001 18.867 18.829
18.795 18.739 18.618 18.498 18.441 18.352 18.241 17.188 18.178 17.269
17.298 18.054 18.033 17.692 17.349 17.847 17.516 17.886 17.817 17.923
17.417 17.380 17.982 17.997 17.529 17.570 10.791 10.838 10.905 10.851
10.855 10.935 11.062 11.076 11.113 11.156 11.218 11.260 11.204 11.249
17.133 16.337 17.088 16.939 16.426 16.449 17.021 16.579 16.526 16.844
16.623 17.011 17.007 16.482 16.730 16.760 16.652 16.687 16.808 16.777
16.803 11.364 11.421 11.438 11.448 11.520 11.529 11.546 11.611 11.603
15.846 15.869 15.892 16.160 16.119 16.207 16.258 15.926 16.285 15.993
15.933 16.081 16.052 16.066 16.236 16.374 11.669 11.738 11.770 11.812
11.827 11.860 11.846 11.882 15.380 15.406 15.745 15.452 15.470 15.793
15.551 15.692 15.649 15.526 15.493 15.687 15.600 15.606 11.877 11.920
12.028 11.968 11.956 12.052 12.108 12.141 12.160 12.203 11.961 12.249
15.358 15.335 15.301 15.217 15.264 15.250 15.082 15.121 15.168 15.013
14.995 15.089 15.319 14.976 14.991 12.327 12.277 12.287 12.374 12.410
12.426 12.523 12.296 12.548 12.579 12.433 12.650 12.784 12.747 14.932
14.918 14.869 12.701 12.844 14.829 14.764 14.839 14.739 14.606 14.616
14.713 14.672 14.696 12.916 12.906 14.545 14.466 14.446 14.502 14.555
14.344 14.417 14.400 14.406 12.354 12.597 12.666 12.711 14.426 12.949
12.863 12.966 13.003 14.281 13.056 13.097 14.242 13.131 13.027 14.223
13.071 13.162 13.229 13.236 13.186 13.022 14.193 13.292 13.286 13.202
13.382 13.341 14.180 14.159 13.411 13.895 14.112 14.141 13.871 13.979
13.962 14.081 14.093 14.041 14.072 13.810 14.000 13.454 13.850 13.787
13.480 13.722 14.018 13.674 13.692 13.644 13.591 13.562 13.531 13.763
13.494 13.600 14.024 13.843 13.705 13.519 13.513 13.625 13.504 0.788
14.170 76.970 74.962 73.582 70.437 70.264 69.737 69.281 68.658 1.100
1.446 2.012 1.807 2.366 3.164 68.441 68.056 67.960 3.610 3.768 67.529
4.401 66.718 66.564 4.729 4.899 66.316 66.171 5.736 5.199 5.032 65.823
62.080 65.460 6.772 7.982 62.486 64.848 63.655 62.613 63.424 6.451 8.438
64.103 64.426 62.778 7.298 63.037 64.786 9.144 65.173 64.279 6.382 9.614
9.876 9.937 10.302 10.402 12.308 11.576 12.041 11.851 10.778 10.663
62.682 55.938 61.424 61.259 61.773 56.783 57.055 60.636 60.899 60.346
57.311 57.614 57.956 59.705 60.125 59.068 58.672 58.856 58.327 59.467
58.273 59.340 59.379 58.203 1.619 12.775 13.065 13.344 13.795 52.698
52.787 56.201 53.090 54.351 54.539 54.765 55.426 55.505 55.152 53.268
53.607 53.377 54.111 55.637 53.495 15.971 15.812 15.074 15.342 13.945
14.085 14.635 50.347 50.780 51.035 52.118 51.274 51.821 52.471 51.994
51.583 51.142 51.343 51.733 16.678 16.938 16.255 16.109 17.756 17.930
18.472 17.824 48.613 48.736 49.592 49.811 49.344 49.149 49.204 50.452
50.096 50.083 49.264 18.841 19.084 19.454 18.967 20.893 20.261 19.897
20.121 19.975 46.264 47.766 48.122 46.856 47.334 46.542 47.041 47.403
46.701 47.012 21.112 21.315 21.573 21.762 21.931 22.305 23.101 23.128
23.458 23.626 24.026 23.846 45.804 45.960 45.586 45.311 45.212 44.892
44.737 44.601 44.487 44.697 44.211 44.383 43.105 43.271 44.404 44.133

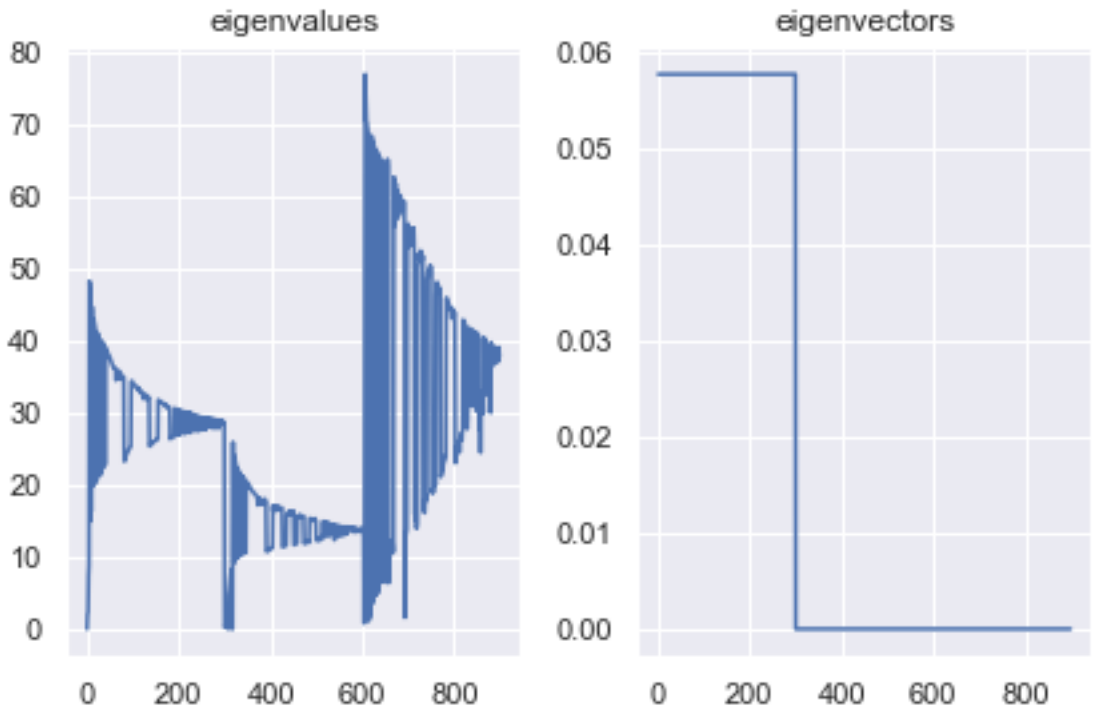
```
 43.952 43.728 43.767 43.855 23.035 24.220 25.033 24.727 24.395 25.632
 25.427 24.845 25.994 24.542 26.374 26.657 27.048 26.865 26.143 27.360
 27.512 42.884 42.717 42.495 42.557 42.276 28.481 29.158 28.702 28.232
 27.774 29.561 41.814 41.772 41.676 31.464 35.735 34.914 31.600 35.299
 41.528 30.905 34.452 30.969 32.084 41.273 32.565 41.101 40.862 41.069
 33.425 30.232 33.032 40.954 40.588 40.505 27.863 28.790 32.776 24.483
 30.552 30.482 30.031 30.109 32.245 29.933 40.414 32.334 35.105 40.159
 40.019 39.503 33.921 34.210 34.033 33.247 36.111 36.169 36.606 33.727
 34.136 29.992 39.756 36.324 39.225 38.903 37.561 36.800 39.156 37.851
 36.997 38.340 37.025 38.227 38.505 37.708 38.157 37.091 39.083 38.684
 37.764]
M eigenvectors:
[[0.058 0.053 -0.063 … 0.000 0.000 0.000]
 [0.058 0.052 -0.067 … 0.000 0.000 0.000]
 [0.058 0.071 -0.042 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 0.007 0.003 0.011]
 [0.000 0.000 0.000 … -0.002 -0.003 -0.027]
 [0.000 0.000 0.000 … -0.004 0.000 -0.001]]
```

Question5 ($k = 20$, CD)

[23]:
```
# construct the matrix U
u=v+L
print("the U matrix is: ")
print(u)
```

```
the U matrix is:
[[32.088 -0.078 -0.755 … 0.000 0.000 0.000]
 [-0.073 27.020 -0.859 … 0.000 0.000 0.000]
 [-0.634 -0.721 27.925 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 15.534 0.003 0.011]
 [0.000 0.000 0.000 … -0.002 50.313 -0.027]
 [0.000 0.000 0.000 … -0.004 0.000 6.223]]
```

[24]:
```
# eigenvalues and eigen vector plot
fig = plt.figure()
ax1 = plt.subplot(121)
plt.plot(e)
ax1.title.set_text('eigenvalues')
i = np.where(e < 10e-6)[0]
ax2 = plt.subplot(122)
plt.plot(v[:, i[0]])
ax2.title.set_text('eigenvectors')
fig.tight_layout()
plt.show()
```



[25]:
```
df=pd.read_csv('D:/HW2/Circle.csv',names=["a", "b"], encoding='latin-1')
```

Question 6,7,8,9 ($k = 20$, **CD**)

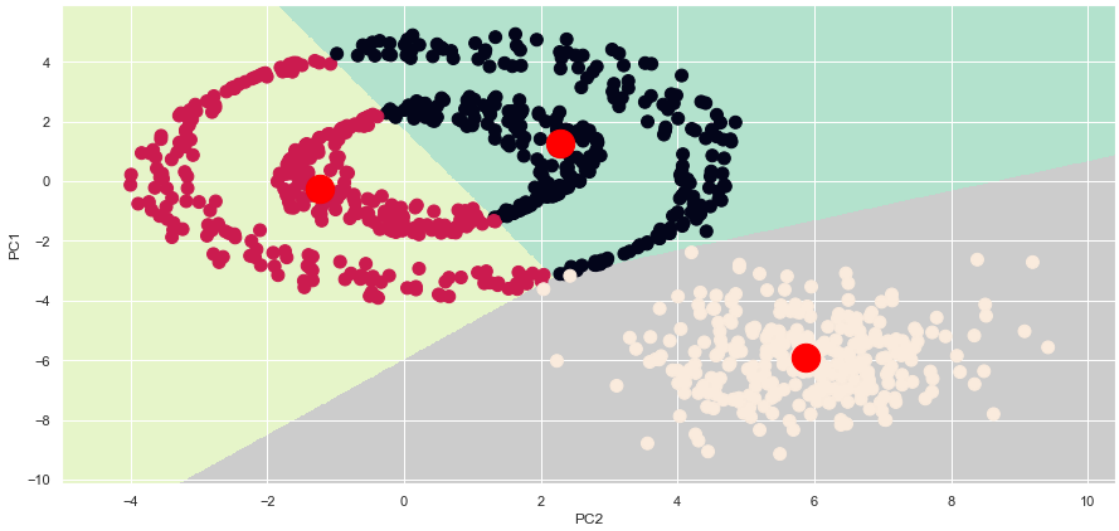
[26]:
```
#Compute the clusters for the Circle dataset with K-means method
algorithm = (KMeans(n_clusters = 3,init='k-means++', n_init = 20 ,max_iter=1000,
                    tol=0.0001,  random_state= 50  , algorithm='elkan') )
algorithm.fit(df)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

[27]:
```
X1 = df[['a','b']].iloc[: , :].values
```

[28]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

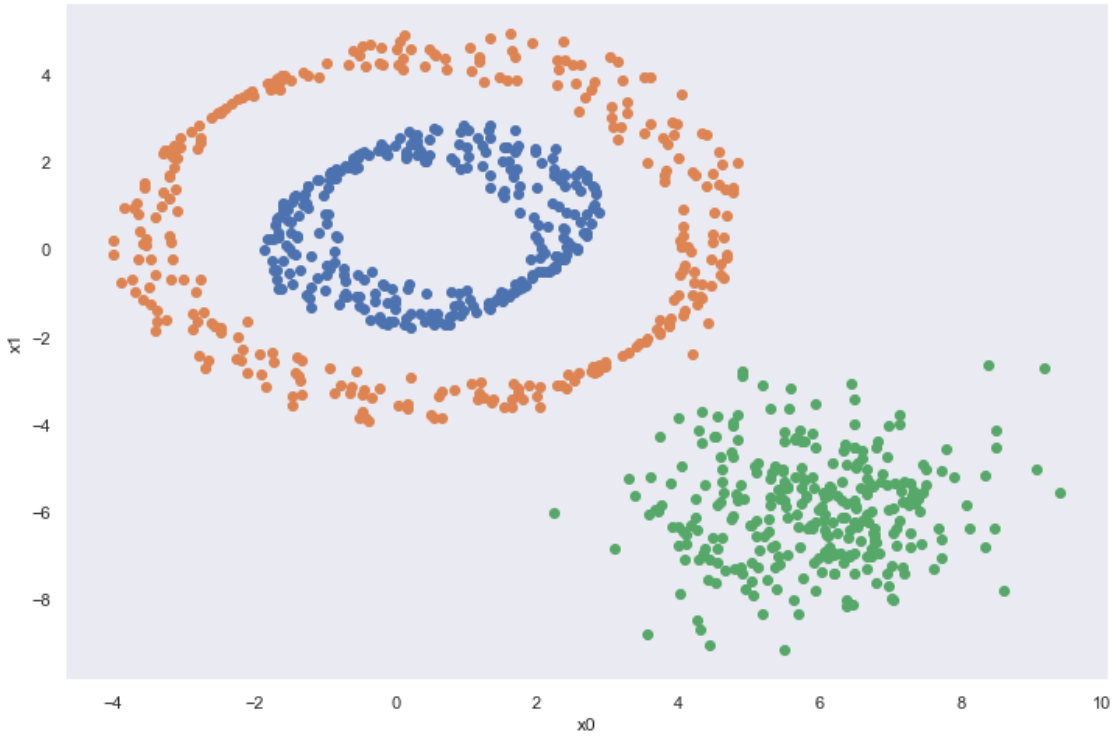
```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\base.py:451: UserWarning: X does not have valid feature names,
but KMeans was fitted with feature names
  "X does not have valid feature names, but"
```

[29]:
```
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'a', y = 'b', data = df, c = labels1, s = 100)
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 500, c = 'red'␣
 ↪, alpha = 1)
plt.ylabel('PC1') , plt.xlabel('PC2')
plt.show()
```



[30]:
```
#Compute the clusters for the Circle dataset with clustering method
from sklearn.cluster import SpectralClustering
sine_dataset = pd.DataFrame(df)

## Fitting spectral clustering with 3 clusters
spectral_clustering = SpectralClustering(n_clusters = 3,
                                         affinity = 'nearest_neighbors',
                                         n_neighbors = 20).fit(df)

## Appending cluster to the sine dataset
sine_dataset['spectral_clusters'] = spectral_clustering.labels_


## Visualizing the data with cluster
colors = ['lightblue', 'orange']
```

```
plt.figure(figsize = (12, 8))
plt.xlabel('x0')
plt.ylabel('x1')
plt.grid()

for c in sine_dataset['spectral_clusters'].unique():

        temp = sine_dataset[sine_dataset['spectral_clusters'] == c]
        plt.scatter(temp['a'], temp['b'])
```



4.3 $k = 40$ in Circle dataset

Question1 ($k = 40$, CD)

[31]:
```
#construct the k-nearest neighborhood similarity graph and its adjacency matrix
from sklearn.neighbors import radius_neighbors_graph
W = radius_neighbors_graph(df,1,mode='distance', metric='minkowski', p=40,␣
 ↪metric_params=None, include_self=False) # k=p=40
W = W.toarray()
print("The adjacency matrix is: ")
W
```

```
The adjacency matrix is:
```

[31]:
```
array([[0.000, 0.131, 0.690, …, 0.000, 0.000, 0.000],
       [0.131, 0.000, 0.782, …, 0.000, 0.000, 0.000],
       [0.690, 0.782, 0.000, …, 0.000, 0.000, 0.000],
       …,
       [0.000, 0.000, 0.000, …, 0.000, 0.000, 0.000],
       [0.000, 0.000, 0.000, …, 0.000, 0.000, 0.000],
       [0.000, 0.000, 0.000, …, 0.000, 0.000, 0.000]])
```

[32]:
```
#Constructing the degree matrix ( )
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[32.019 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 26.952 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 27.940 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 15.506 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 50.238 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 6.221]]
```

Question2 ($k = 40$, CD)

[33]:
```
from scipy.sparse import csgraph

# with the function below we can get the Laplacian Matrix(L=D-W) from the␣
 ↪adjacency matrix we have from previous section
L = csgraph.laplacian(W, normed=False)

print('The Laplacian Matrix L is: ')
print(L)

print('The first 5 values of L are: ')
print(L[:5,:5])
```

```
The Laplacian Matrix L is:
[[32.019 -0.131 -0.690 … -0.000 -0.000 -0.000]
 [-0.131 26.952 -0.782 … -0.000 -0.000 -0.000]
 [-0.690 -0.782 27.940 … -0.000 -0.000 -0.000]
 …
 [-0.000 -0.000 -0.000 … 15.506 -0.000 -0.000]
 [-0.000 -0.000 -0.000 … -0.000 50.238 -0.000]
 [-0.000 -0.000 -0.000 … -0.000 -0.000 6.221]]
The first 5 values of L are:
[[32.019 -0.131 -0.690 -0.218 -0.189]
 [-0.131 26.952 -0.782 -0.087 -0.280]
 [-0.690 -0.782 27.940 -0.829 -0.585]
 [-0.218 -0.087 -0.829 25.015 -0.284]
 [-0.189 -0.280 -0.585 -0.284 33.092]]
```

Question3 ($k = 40$, **CD**)

[34]:
```
#compute the cosine similarity to determine how many connected components there␣
 ↪are in the similarity graph
# import required libraries
from numpy.linalg import norm

# compute  similarity
cosine = np.dot(df['a'],df['b'])/(norm(df['a'],)*norm(df['b']))
print(" Similarity:", cosine)
```

```
Similarity: -0.7272658115521661
```

Question4 ($k = 40$, **CD**)

[35]:
```
e, v = np.linalg.eig(L)
# eigenvalues
print('eigenvalues:')
print(e)
# normalised eigenvectors
print('M eigenvectors:')
print(v)
```

```
eigenvalues:
[-0.000 2.140 2.264 7.413 8.867 14.996 16.384 48.293 47.294 45.837 44.689
 44.060 19.827 20.033 20.317 20.842 21.134 21.298 21.456 21.540 21.639
 21.973 43.104 42.763 42.288 41.978 41.489 41.339 41.125 40.859 40.571
 40.168 40.028 39.670 39.609 39.446 39.274 39.043 38.890 38.699 38.607
 38.353 38.263 38.177 37.986 37.886 37.703 37.522 37.538 37.196 36.951
 36.916 36.728 36.638 36.611 36.532 36.352 36.301 36.252 35.437 35.509
 36.113 36.023 35.938 35.849 35.803 35.632 35.646 22.380 22.437 22.745
 23.274 23.382 23.814 23.984 24.041 24.312 24.381 24.533 24.712 24.681
 24.868 25.037 35.248 35.170 35.117 35.048 35.034 34.869 34.917 34.681
 34.659 34.549 34.540 34.200 34.383 34.340 34.125 33.958 33.892 33.854
 33.569 33.619 33.647 33.701 33.688 33.392 33.441 33.528 32.152 33.212
 33.236 33.319 32.268 33.044 32.772 32.914 32.580 33.163 32.933 32.508
 32.680 32.672 32.389 32.364 32.424 32.430 25.186 25.219 25.278 25.353
 25.401 25.472 25.515 25.610 25.699 25.875 25.881 25.893 25.939 26.024
 26.076 26.095 26.161 26.192 26.211 26.239 32.096 32.045 32.080 31.997
 31.978 31.942 31.881 31.796 31.817 31.733 31.701 31.645 31.533 31.527
 31.364 31.451 31.437 31.281 31.267 31.197 30.865 31.122 31.071 30.824
 30.999 31.227 30.795 31.038 30.963 26.288 26.305 26.339 26.425 26.520
 26.477 26.569 26.468 26.610 26.641 26.723 26.671 30.727 30.713 30.658
 30.603 26.840 30.540 30.487 30.463 26.876 30.363 30.396 30.323 30.293
 30.238 30.232 30.202 27.005 30.136 27.199 27.155 27.021 30.118 27.299
 30.087 27.103 27.092 30.033 29.977 30.066 27.055 29.950 29.888 30.008
 27.050 29.857 29.941 29.773 27.070 27.419 27.443 29.815 29.793 27.317
 27.459 29.761 29.717 29.680 29.636 27.514 27.521 27.555 27.739 27.793
 27.587 29.573 27.843 27.628 29.534 29.527 29.502 27.880 27.606 27.663
```

29.466 27.658 27.657 29.423 27.967 27.954 28.007 27.995 28.626 28.027
28.568 29.042 28.671 28.152 28.798 28.890 28.928 28.974 29.410 29.078
29.256 29.135 28.997 28.734 29.322 29.269 28.748 29.206 28.161 29.332
29.360 29.178 28.088 29.369 28.761 28.499 28.239 28.309 28.276 28.351
29.147 28.475 28.415 28.252 28.291 28.449 28.365 28.473 29.691 0.196
0.224 0.699 0.982 1.649 2.005 0.000 2.873 3.462 4.803 5.227 6.031 7.034
7.501 7.831 8.470 -0.000 0.000 9.048 26.003 9.517 9.635 9.768 9.843 9.897
9.929 10.108 10.055 10.278 10.379 24.083 23.340 22.660 22.208 21.805
21.749 21.629 21.287 21.155 20.718 20.583 20.448 20.416 20.085 20.015
18.131 19.908 19.818 19.781 19.734 19.628 19.525 19.463 19.376 19.267
18.321 19.175 19.098 18.418 18.478 18.536 18.991 18.672 18.917 18.839
18.767 18.782 10.335 10.402 10.483 10.605 10.575 10.688 10.806 10.863
10.841 10.924 10.954 11.048 11.078 11.136 11.174 18.232 18.032 18.014
17.927 17.961 17.877 17.827 17.839 17.685 17.632 17.549 17.404 17.369
17.504 17.326 17.517 16.593 17.268 17.170 17.135 16.651 17.095 16.760
16.711 16.955 16.843 16.915 17.018 16.680 16.999 16.795 16.901 16.818
11.205 11.236 11.248 11.350 11.407 11.427 11.445 11.515 11.477 11.548
11.614 11.598 11.568 16.056 16.617 16.124 16.109 16.179 16.529 16.303
16.259 16.351 16.100 16.366 16.218 16.483 16.445 16.471 16.415 16.226
11.687 11.729 11.768 11.820 11.835 11.917 11.884 11.854 11.845 15.521
15.997 15.953 15.638 15.878 15.759 15.698 15.924 15.560 15.591 15.719
15.833 15.802 15.665 15.841 11.956 11.909 12.029 12.054 12.015 11.999
12.117 12.165 12.194 12.219 12.327 12.265 15.483 15.452 15.469 15.198
15.404 15.370 15.352 15.253 15.269 15.309 15.304 15.313 15.088 15.100
15.151 15.219 12.411 12.383 12.384 12.299 12.291 12.493 12.284 12.517
12.563 12.545 12.607 12.586 12.655 15.055 15.016 12.775 12.694 12.692
14.901 14.931 14.984 14.978 14.966 14.862 14.815 14.826 14.754 14.961
14.727 12.853 12.863 12.896 13.042 14.699 14.677 14.638 12.989 14.595
12.933 13.011 14.613 14.534 14.566 14.457 12.904 14.483 14.520 12.712
13.068 13.088 13.122 12.952 12.962 14.386 14.421 14.416 13.171 13.190
13.222 13.230 13.291 14.368 14.335 14.298 13.274 13.342 13.307 13.375
13.370 13.421 14.234 14.214 13.452 13.483 13.742 13.760 13.797 14.189
14.169 13.938 13.503 13.683 13.963 13.659 13.634 14.102 14.134 13.860
13.988 14.161 13.839 13.521 13.617 14.048 14.030 13.877 13.832 14.153
13.553 14.074 13.876 14.009 13.528 13.571 13.573 13.596 14.011 14.095
0.780 77.055 75.311 73.842 70.640 70.368 69.777 69.604 69.176 68.795
68.427 68.699 67.800 65.863 66.129 66.679 66.992 66.867 65.595 64.378
65.013 65.266 64.252 64.820 64.757 1.114 1.441 1.807 2.397 3.167 2.357
3.606 4.080 7.992 5.739 4.618 4.743 5.191 4.900 6.764 7.295 6.449 5.029
6.375 8.460 9.145 9.625 9.860 9.923 10.285 10.391 11.580 10.793 12.038
11.842 10.646 63.698 63.400 62.990 62.977 62.556 62.738 62.683 57.009
57.266 61.993 57.587 61.557 58.028 61.379 60.997 60.330 60.043 58.829
62.227 59.419 58.294 60.747 60.726 59.722 58.428 59.156 59.066 58.209
59.807 12.305 12.759 13.328 13.067 53.082 53.383 53.489 54.446 54.566
56.247 56.842 56.014 55.153 55.089 55.721 54.181 53.715 53.608 55.663
55.866 16.020 15.799 13.784 15.091 15.336 14.703 14.075 13.922 50.765
51.093 51.153 51.388 52.887 52.306 51.653 51.897 52.737 52.027 51.991
52.449 53.244 1.619 16.196 16.611 16.816 16.948 18.527 17.741 17.918

```
 17.817 48.748 49.120 49.840 49.378 49.614 50.136 49.192 50.334 50.035
 49.217 50.453 18.814 19.006 19.218 19.518 19.927 20.336 20.293 20.099
 46.267 47.943 48.576 46.600 48.140 47.658 46.820 47.477 46.909 47.153
 47.082 20.993 21.382 21.521 21.769 21.570 22.288 21.988 23.091 23.190
 23.284 23.667 23.816 25.726 46.053 46.141 45.795 45.396 45.301 44.861
 44.721 44.196 44.566 44.358 44.451 44.646 43.346 44.643 44.099 43.984
 43.729 43.745 43.818 23.945 25.181 24.897 24.806 24.288 24.197 24.427
 24.499 25.983 25.440 26.372 26.676 26.851 43.248 24.534 27.022 42.846
 26.107 27.330 42.725 27.477 42.646 42.515 42.270 28.473 29.181 28.678
 42.135 35.817 35.357 27.855 34.941 29.531 34.478 41.900 41.809 41.718
 31.453 41.163 28.773 30.845 31.271 30.649 31.119 31.938 30.274 41.094
 41.521 41.595 40.887 32.765 27.742 30.515 40.844 33.111 33.007 32.221
 32.566 28.223 30.009 29.970 29.982 32.303 33.667 33.468 40.685 33.857
 34.201 36.118 36.174 36.330 36.805 40.601 40.506 40.277 39.296 38.104
 37.304 38.802 37.589 38.588 39.712 33.990 39.872 36.944 39.115 38.655
 37.734 35.179 37.777 36.969 40.048 39.947 37.094 34.145 34.023 39.069
 38.282]
M eigenvectors:
[[-0.058 -0.056 -0.060 … 0.000 0.000 0.000]
 [-0.058 -0.056 -0.065 … 0.000 0.000 0.000]
 [-0.058 -0.073 -0.039 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 0.005 0.000 -0.000]
 [0.000 0.000 0.000 … 0.003 0.000 -0.007]
 [0.000 0.000 0.000 … -0.003 -0.000 0.000]]
```

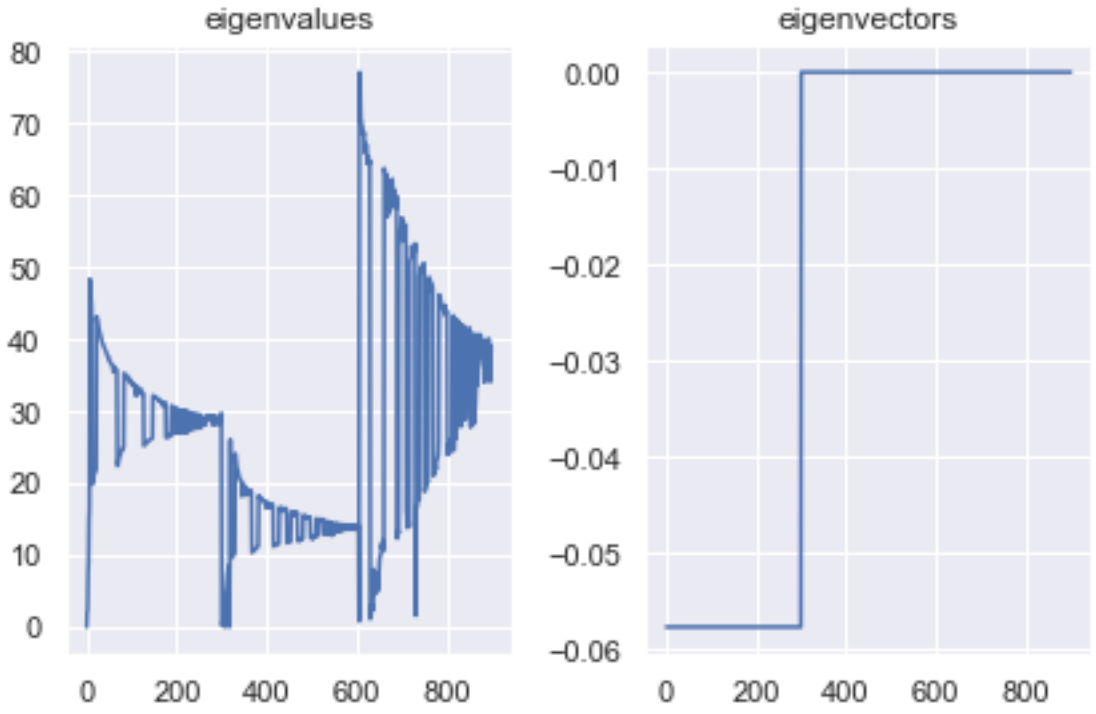
Question5 ($k = 40$, CD)

[36]:
```
# construct the matrix U
u=v+L
print("the U matrix is: ")
print(u)
```

```
the U matrix is:
[[31.961 -0.187 -0.750 … 0.000 0.000 0.000]
 [-0.189 26.897 -0.847 … 0.000 0.000 0.000]
 [-0.747 -0.855 27.901 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 15.511 0.000 -0.000]
 [0.000 0.000 0.000 … 0.003 50.238 -0.007]
 [0.000 0.000 0.000 … -0.003 -0.000 6.221]]
```

[37]:
```
fig = plt.figure()
ax1 = plt.subplot(121)
plt.plot(e)
ax1.title.set_text('eigenvalues')
i = np.where(e < 10e-6)[0]
ax2 = plt.subplot(122)
```

```
plt.plot(v[:, i[0]])
ax2.title.set_text('eigenvectors')
fig.tight_layout()
plt.show()
```



[38]:
```
df=pd.read_csv('D:/HW2/Circle.csv',names=["a", "b"], encoding='latin-1')
```

Question 6,7,8,9 ($k = 40$, CD)

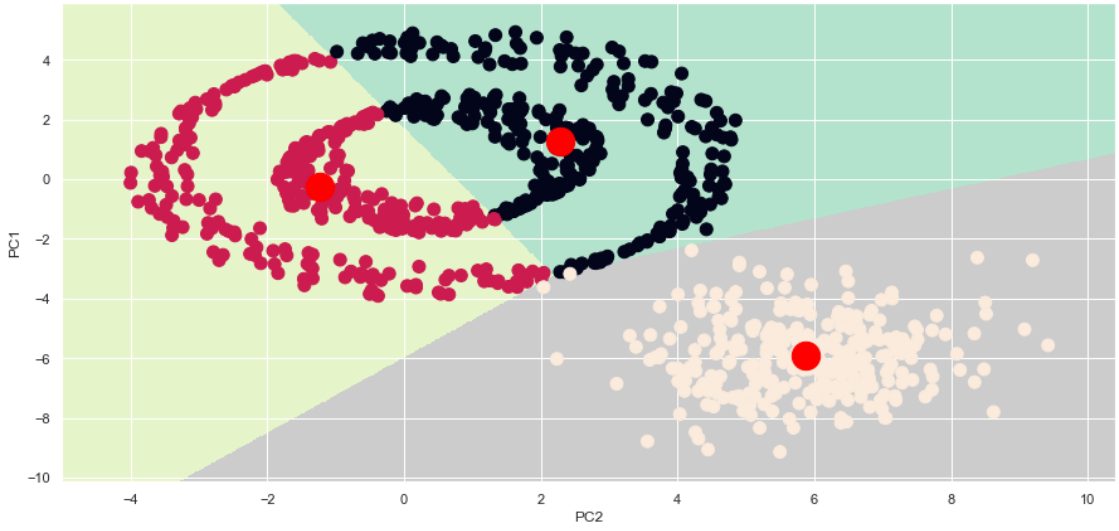
[39]:
```
#Compute the clusters for the Circle dataset with K-means method
algorithm = (KMeans(n_clusters = 3,init='k-means++', n_init = 40 ,max_iter=100,
                    tol=0.0001,  random_state= 50  , algorithm='elkan') )
algorithm.fit(df)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

[40]:
```
X1 = df[['a','b']].iloc[: , :].values
```

[41]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

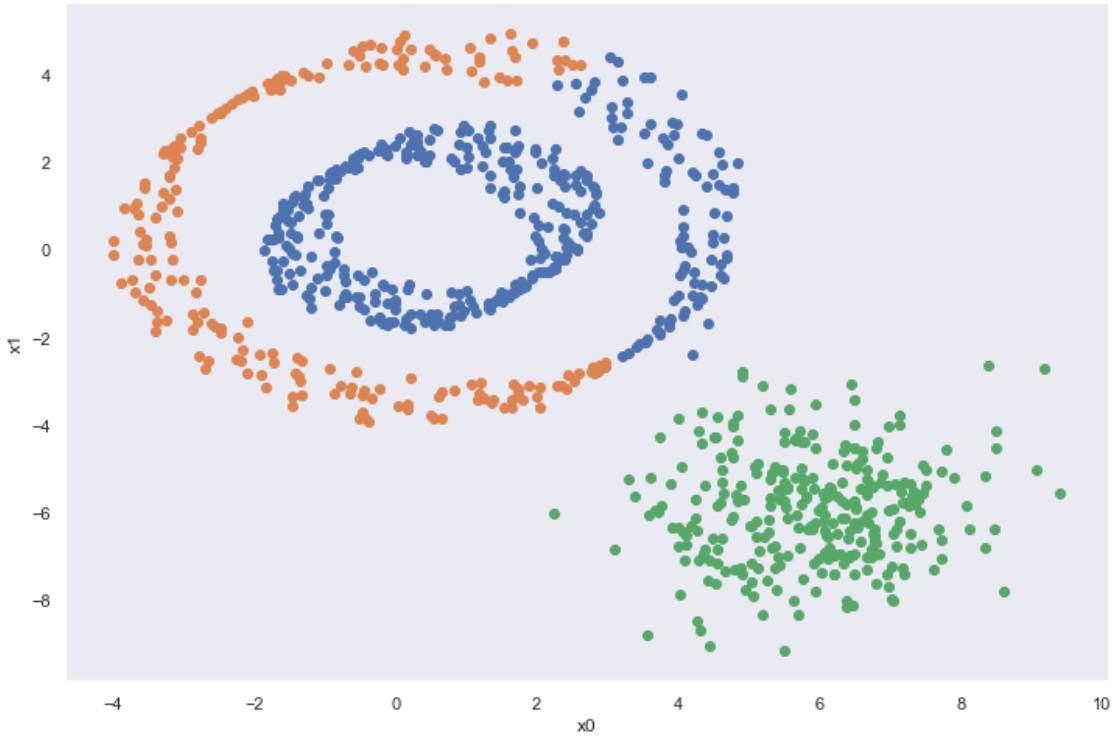
```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\base.py:451: UserWarning: X does not have valid feature names,
but KMeans was fitted with feature names
  "X does not have valid feature names, but"
```

[42]:
```
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'a', y = 'b', data = df, c = labels1, s = 100)
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 500, c = 'red'␣
 ↪, alpha = 1)
plt.ylabel('PC1') , plt.xlabel('PC2')
plt.show()
```



[43]:
```
#Compute the clusters for the Circle dataset with clustering method
from sklearn.cluster import SpectralClustering
sine_dataset = pd.DataFrame(df)

## Fitting spectral clustering with 3 clusters
spectral_clustering = SpectralClustering(n_clusters = 3,
                                         affinity = 'nearest_neighbors',
                                         n_neighbors = 40).fit(df)

## Appending cluster to the sine dataset
sine_dataset['spectral_clusters'] = spectral_clustering.labels_
```

```
## Visualizing the data with cluster
colors = ['lightblue', 'orange']

plt.figure(figsize = (12, 8))
plt.xlabel('x0')
plt.ylabel('x1')
plt.grid()

for c in sine_dataset['spectral_clusters'].unique():

        temp = sine_dataset[sine_dataset['spectral_clusters'] == c]
        plt.scatter(temp['a'], temp['b'])
```



5 load the Spiral Dataset(SD)

[44]:
```
df=pd.read_csv('D:/HW2/Spiral.csv',names=["a", "b", "cluster"],␣
↪encoding='latin-1')
df
```

[44]:

	a	b	cluster
0	31.95	7.95	3
1	31.15	7.30	3
2	30.45	6.65	3
3	29.70	6.00	3
4	28.90	5.55	3
..	…	…	…
307	15.75	13.85	2
308	15.65	14.05	2
309	15.65	14.25	2
310	15.65	14.50	2
311	15.65	14.60	2

[312 rows x 3 columns]

[45]:
```
dz=df.drop(['cluster'], axis=1)
```

5.1 $k = 10$ in Spiral dataset

Question1 ($k = 10$ SD)

[46]:
```
#construct the k-nearest neighborhood similarity graph and its adjacency matrix
from sklearn.neighbors import radius_neighbors_graph
W = radius_neighbors_graph(dz,1,mode='distance', metric='minkowski', p=10,␣
 ↪metric_params=None, include_self=False) #k=p=10
W = W.toarray()
print("The adjacency matrix is: ")
W
```

The adjacency matrix is:

[46]:
```
array([[0.000, 0.810, 0.000, …, 0.000, 0.000, 0.000],
       [0.810, 0.000, 0.728, …, 0.000, 0.000, 0.000],
       [0.000, 0.728, 0.000, …, 0.000, 0.000, 0.000],
       …,
       [0.000, 0.000, 0.000, …, 0.000, 0.250, 0.350],
       [0.000, 0.000, 0.000, …, 0.250, 0.000, 0.100],
       [0.000, 0.000, 0.000, …, 0.350, 0.100, 0.000]])
```

[47]:
```
#Constructing the degree matrix ( )
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[0.810 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 1.537 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 1.494 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 3.400 0.000 0.000]
```

```
 [0.000 0.000 0.000 … 0.000 2.300 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 2.700]]
```

Question2 ($k = 10$ **,SD)**

[48]:
```
from scipy.sparse import csgraph

# with the function below we can get the Laplacian Matrix(L=D-W) from the␣
 ↪adjacency matrix we have from previous section
L = csgraph.laplacian(W, normed=False)

print('The Laplacian Matrix L is: ')
print(L)

print('The first 5 values of L are: ')
print(L[:5,:5])
```

```
The Laplacian Matrix L is:
[[0.810 -0.810 -0.000 … -0.000 -0.000 -0.000]
 [-0.810 1.537 -0.728 … -0.000 -0.000 -0.000]
 [-0.000 -0.728 1.494 … -0.000 -0.000 -0.000]
 …
 [-0.000 -0.000 -0.000 … 3.400 -0.250 -0.350]
 [-0.000 -0.000 -0.000 … -0.250 2.300 -0.100]
 [-0.000 -0.000 -0.000 … -0.350 -0.100 2.700]]
The first 5 values of L are:
[[0.810 -0.810 -0.000 -0.000 -0.000]
 [-0.810 1.537 -0.728 -0.000 -0.000]
 [-0.000 -0.728 1.494 -0.766 -0.000]
 [-0.000 -0.000 -0.766 1.567 -0.800]
 [-0.000 -0.000 -0.000 -0.800 1.651]]
```

Question3 ($k = 10$ **, SD)**

[49]:
```
#compute the cosine similarity to determine how many connected components there␣
 ↪are in the similarity graph
# import required libraries
from numpy.linalg import norm

# compute  similarity
cosine = np.dot(df['a'],df['b'])/(norm(df['a'],)*norm(df['b']))
print(" Similarity:", cosine)
```

```
 Similarity: 0.8575754202341677
```

Question4 ($k = 10$ **, SD)**

[50]:
```
e, v = np.linalg.eig(L)
# eigenvalues
```

```
print('eigenvalues:')
print(e)
# normalised eigenvectors
print('M eigenvectors:')
print(v)
```

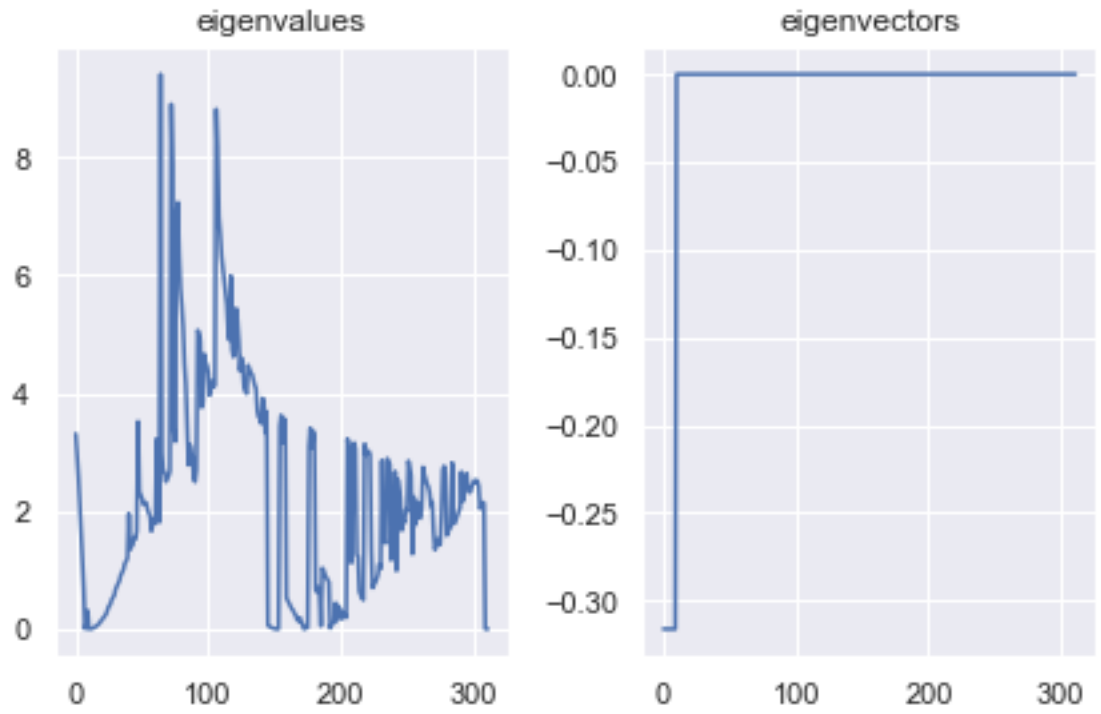
```
eigenvalues:
[3.310 2.954 2.589 2.131 1.628 1.126 0.680 -0.000 0.081 0.312 0.000 0.001
 0.005 0.013 0.030 0.038 0.057 0.077 0.101 0.139 0.160 0.198 0.223 0.261
 0.332 0.393 0.411 0.494 0.530 0.569 0.672 0.720 0.757 0.824 0.918 0.958
 1.012 1.115 1.146 1.210 1.956 1.347 1.400 1.427 1.557 1.530 1.657 3.520
 2.330 2.272 2.214 2.113 2.146 2.141 2.042 1.965 1.926 1.661 1.860 1.778
 1.790 3.222 2.435 1.808 9.409 2.986 2.668 2.658 2.499 2.557 2.594 2.679
 8.900 7.845 3.403 3.179 7.042 7.236 6.348 5.768 5.487 5.127 4.496 4.163
 3.397 2.784 3.129 3.054 2.925 2.556 2.497 2.690 5.065 5.011 4.892 3.757
 3.844 4.663 4.520 4.469 4.349 3.966 4.079 4.207 4.103 4.140 8.815 8.176
 6.991 6.694 6.327 6.116 5.872 5.731 5.502 5.050 4.900 5.987 4.738 4.615
 4.659 5.436 5.242 4.887 4.377 4.593 4.439 4.093 4.024 3.992 4.472 4.403
 4.349 4.317 4.226 4.135 4.079 3.701 3.590 3.582 3.485 3.913 3.821 3.325
 3.688 0.062 0.043 0.032 0.021 0.012 0.004 0.001 -0.000 -0.000 3.440 3.624
 3.165 3.534 3.555 0.526 0.478 0.424 0.393 0.341 0.315 0.265 0.230 0.155
 0.141 0.194 0.107 0.092 0.001 0.006 0.033 0.021 3.097 3.405 3.080 3.358
 3.252 0.667 0.626 0.710 0.575 0.051 1.032 0.997 0.916 0.877 0.837 0.793
 0.013 0.074 0.095 0.114 0.441 0.140 0.392 0.359 0.319 0.171 0.266 0.225
 0.195 3.222 3.070 1.217 1.119 2.872 3.156 3.056 1.264 1.225 0.636 0.576
 0.539 0.493 3.134 2.960 3.000 3.020 2.958 1.349 0.699 0.752 0.790 0.846
 0.920 1.099 1.021 2.857 1.559 1.482 1.455 2.891 2.821 2.789 1.170 1.632
 1.583 2.670 0.989 2.534 2.392 2.222 1.670 1.840 1.805 2.025 1.992 2.849
 2.734 2.599 1.271 2.243 1.762 2.159 2.142 1.907 1.877 2.042 2.743 2.623
 2.535 2.461 2.410 2.313 2.085 2.153 1.651 1.344 1.398 1.525 1.470 1.431
 1.905 2.708 2.761 2.634 1.587 1.651 1.699 1.726 2.816 1.786 1.794 1.858
 1.933 1.987 2.060 2.665 2.164 2.269 2.600 2.638 2.369 2.315 2.331 2.429
 2.474 2.522 2.500 2.516 2.420 2.042 2.085 2.150 2.135 0.000 0.000 0.000]
M eigenvectors:
[[-0.015 -0.090 -0.230 … 0.000 0.000 0.000]
 [0.046 0.237 0.505 … 0.000 0.000 0.000]
 [-0.096 -0.362 -0.474 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 0.000]]
```

Question5 ($k = 10$, SD)

[51]:
```
# construct the matrix U
u=v+L
print("the U matrix is: ")
print(u)
```

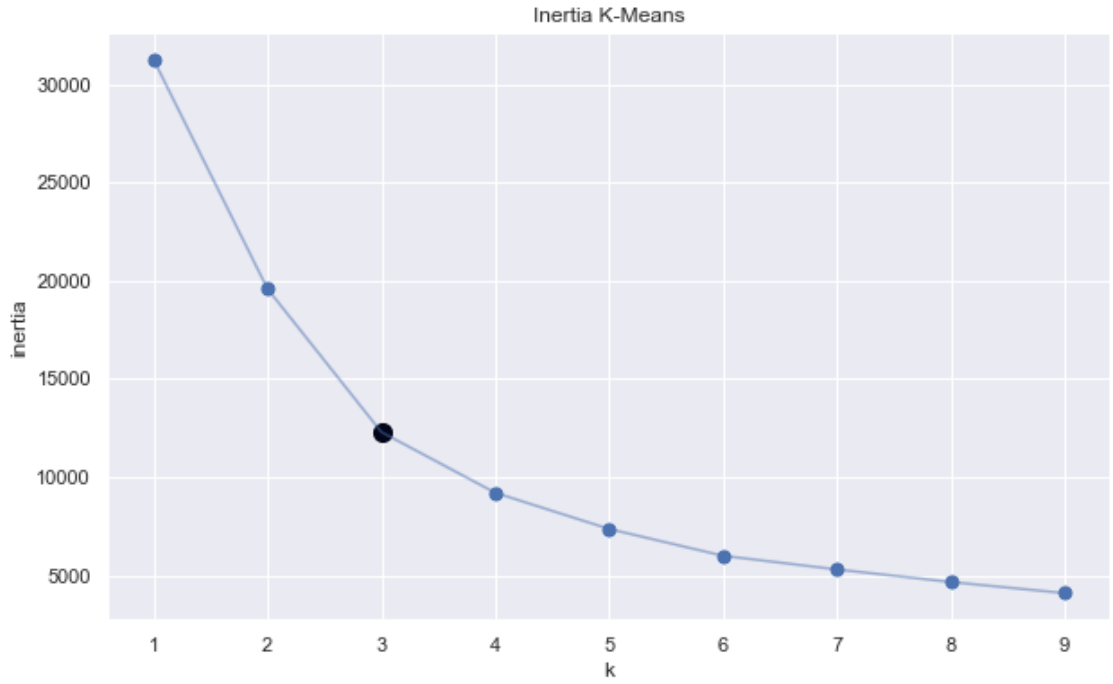
```
the U matrix is:
[[0.795 -0.899 -0.230 … 0.000 0.000 0.000]
 [-0.763 1.775 -0.223 … 0.000 0.000 0.000]
 [-0.096 -1.090 1.020 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 3.400 -0.250 -0.350]
 [0.000 0.000 0.000 … -0.250 2.300 -0.100]
 [0.000 0.000 0.000 … -0.350 -0.100 2.700]]
```

[52]:
```
fig = plt.figure()
ax1 = plt.subplot(121)
plt.plot(e)
ax1.title.set_text('eigenvalues')
i = np.where(e < 10e-6)[0]
ax2 = plt.subplot(122)
plt.plot(v[:, i[0]])
ax2.title.set_text('eigenvectors')
fig.tight_layout()
plt.show()
```



I run the k-means algorithm on the Spiral dataset to try to get some clusters. I select the number of cluster using the **Elbow method** by considering the inertia (sum of squared distances of samples to their closest cluster center) as a function of the number of clusters.

[53]:
```
from sklearn.cluster import KMeans
import seaborn as sns
inertias = []

k_candidates = range(1, 10)

for k in k_candidates:
    k_means = KMeans(random_state=42, n_clusters=k)
    k_means.fit(dz)
    inertias.append(k_means.inertia_)

fig, ax = plt.subplots(figsize=(10, 6))
sns.scatterplot(x=k_candidates, y = inertias, s=80, ax=ax)
sns.scatterplot(x=[k_candidates[2]], y = [inertias[2]], c=[3], s=150, ax=ax)
sns.lineplot(x=k_candidates, y = inertias, alpha=0.5, ax=ax)
ax.set(title='Inertia K-Means', ylabel='inertia', xlabel='k');
```



From this plot we see that $K = 3$ is a good choice. Now I get the clusters.

Question 6,7,8,9 ($k = 10$, SD)

[54]:
```
#Compute the clusters for the Spiral dataset with K-means method
algorithm = (KMeans(n_clusters = 3,init='k-means++', n_init = 10 ,max_iter=100,
                    tol=0.0001,  random_state= 50  , algorithm='elkan') )
algorithm.fit(dz)
```

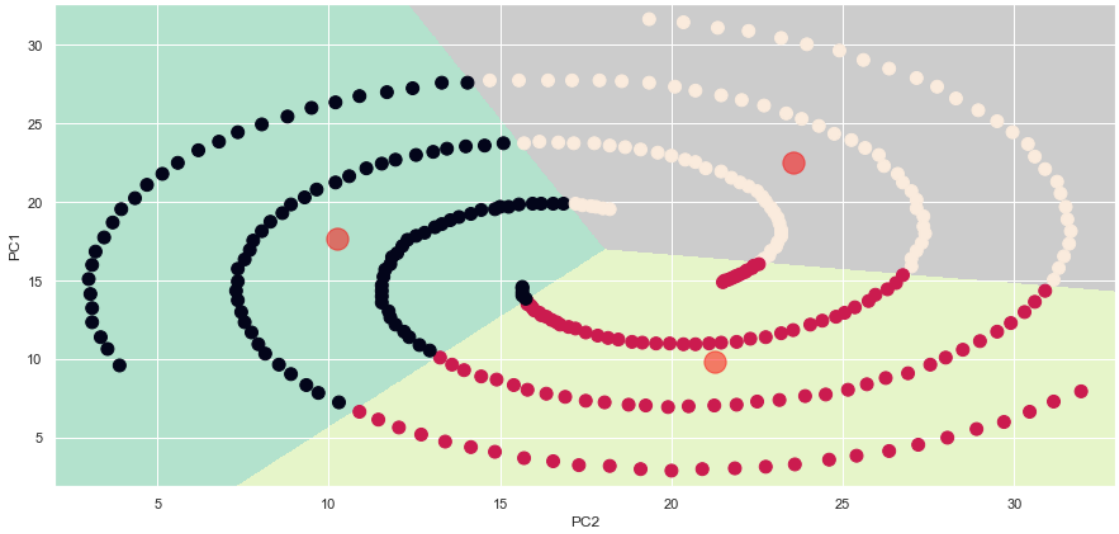
```
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

[55]:
```
X1 = dz[['a','b']].iloc[: , :].values
```

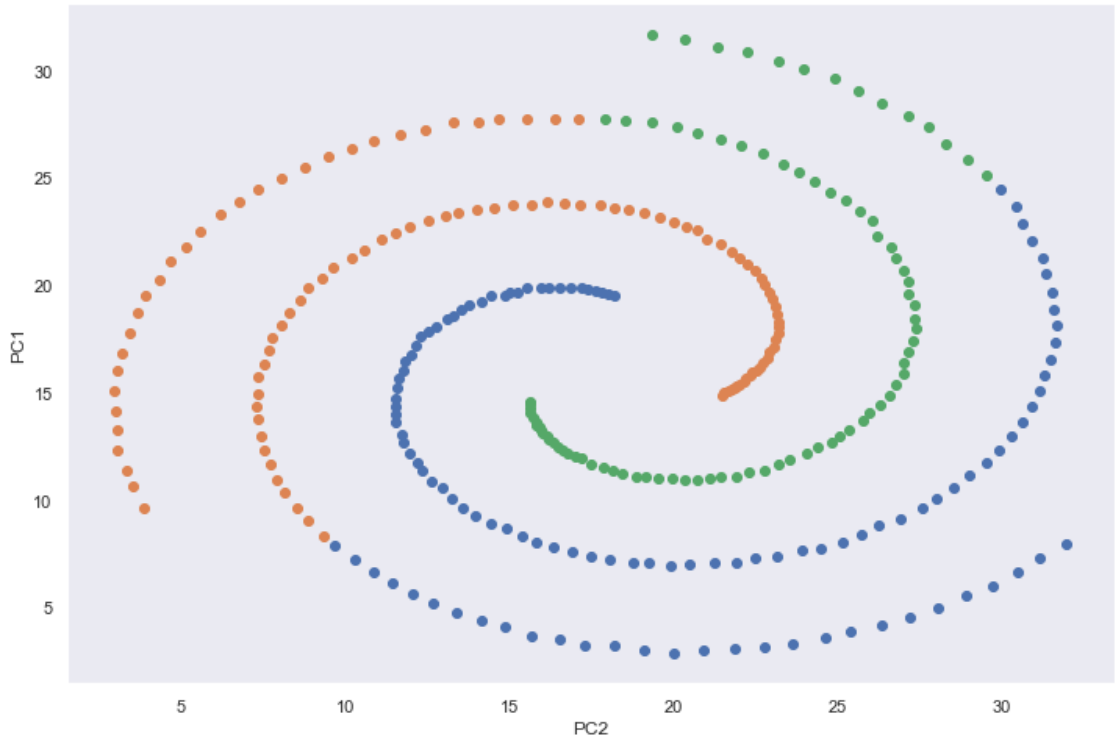
[56]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\base.py:451: UserWarning: X does not have valid feature names,
but KMeans was fitted with feature names
  "X does not have valid feature names, but"
```

[57]:
```
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'a', y = 'b', data = dz, c = labels1, s = 100)
#plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 500, c =␣
 ↪'red' , alpha = 1)
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c =␣
 ↪'red' , alpha = 0.5)
plt.ylabel('PC1') , plt.xlabel('PC2')
plt.show()
```



[58]:
```
#Compute the clusters for the Spiral dataset with clustering method
from sklearn.cluster import SpectralClustering
sine_dataset = pd.DataFrame(dz)

## Fitting spectral clustering with 3 clusters
spectral_clustering = SpectralClustering(n_clusters = 3,
                                         affinity = 'nearest_neighbors',
                                         n_neighbors = 10).fit(dz)

## Appending cluster to the sine dataset
sine_dataset['spectral_clusters'] = spectral_clustering.labels_


## Visualizing the data with cluster
colors = ['lightblue', 'orange']

plt.figure(figsize = (12, 8))
plt.xlabel('PC2')
plt.ylabel('PC1')
plt.grid()

for c in sine_dataset['spectral_clusters'].unique():

        temp = sine_dataset[sine_dataset['spectral_clusters'] == c]
        plt.scatter(temp['a'], temp['b'])
```



5.2 $k = 20$ in Spiral dataset

Question1 ($k = 20$, SD)

[59]:
```
#construct the k-nearest neighborhood similarity graph and its adjacency matrix
from sklearn.neighbors import radius_neighbors_graph
W = radius_neighbors_graph(dz,1,mode='distance', metric='minkowski', p=20,␣
 ↪metric_params=None, include_self=False) # k=p=20
W = W.toarray()
print("The adjacency matrix is: ")
W
```

```
The adjacency matrix is:
```

[59]:
```
array([[0.000, 0.801, 0.000, …, 0.000, 0.000, 0.000],
       [0.801, 0.000, 0.707, …, 0.000, 0.000, 0.000],
       [0.000, 0.707, 0.000, …, 0.000, 0.000, 0.000],
       …,
       [0.000, 0.000, 0.000, …, 0.000, 0.250, 0.350],
       [0.000, 0.000, 0.000, …, 0.250, 0.000, 0.100],
       [0.000, 0.000, 0.000, …, 0.350, 0.100, 0.000]])
```

[60]:
```
#Constructing the degree matrix ( )
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[0.801 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 1.508 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 1.459 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 3.400 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 3.300 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 2.700]]
```

Question2 ($k = 20$, **SD**)

[61]:
```
from scipy.sparse import csgraph

# with the function below we can get the Laplacian Matrix(L=D-W) from the␣
 ↪adjacency matrix we have from previous section
L = csgraph.laplacian(W, normed=False)

print('The Laplacian Matrix L is: ')
print(L)

print('The first 5 values of L are: ')
print(L[:5,:5])
```

```
The Laplacian Matrix L is:
[[0.801 -0.801 -0.000 … -0.000 -0.000 -0.000]
 [-0.801 1.508 -0.707 … -0.000 -0.000 -0.000]
 [-0.000 -0.707 1.459 … -0.000 -0.000 -0.000]
 …
 [-0.000 -0.000 -0.000 … 3.400 -0.250 -0.350]
 [-0.000 -0.000 -0.000 … -0.250 3.300 -0.100]
 [-0.000 -0.000 -0.000 … -0.350 -0.100 2.700]]
The first 5 values of L are:
[[0.801 -0.801 -0.000 -0.000 -0.000]
 [-0.801 1.508 -0.707 -0.000 -0.000]
 [-0.000 -0.707 1.459 -0.752 -0.000]
 [-0.000 -0.000 -0.752 1.552 -0.800]
 [-0.000 -0.000 -0.000 -0.800 1.650]]
```

Question3 ($k = 20$, **SD**)

[62]:
```
#compute the cosine similarity to determine how many connected components there␣
 ↪are in the similarity graph
# import required libraries
from numpy.linalg import norm
```

```
# compute  similarity
cosine = np.dot(df['a'],df['b'])/(norm(df['a'],)*norm(df['b']))
print(" Similarity:", cosine)
```

```
 Similarity: 0.8575754202341677
```

Question4 ($k = 20$, **SD)**

[63]:
```
e, v = np.linalg.eig(L)
# eigenvalues
print('eigenvalues:')
print(e)
# normalised eigenvectors
print('M eigenvectors:')
print(v)
```

```
eigenvalues:
[3.307 2.936 2.556 2.118 1.618 1.120 0.675 0.000 0.080 0.309 3.520 3.222
 2.985 2.667 2.492 2.232 2.078 1.888 1.626 1.367 1.136 0.899 0.683 0.495
 0.319 0.000 0.021 0.081 0.182 9.302 8.799 7.756 0.000 0.003 0.011 0.033
 0.047 0.087 0.117 0.164 0.217 0.254 0.345 0.484 0.521 0.578 0.664 0.791
 0.872 0.946 1.109 1.271 1.332 1.390 1.507 1.575 2.400 2.268 1.695 1.711
 2.175 1.832 1.852 1.993 1.953 2.113 2.126 1.790 6.968 7.116 6.257 5.736
 5.460 5.085 5.027 4.410 2.553 2.657 2.563 3.390 3.406 3.332 3.151 2.666
 3.108 3.053 2.978 4.928 2.482 2.533 2.909 2.677 2.706 4.798 4.575 4.440
 4.413 4.302 3.738 3.826 4.065 4.164 3.959 4.090 4.082 4.017 8.871 8.121
 6.946 6.656 6.283 6.164 6.012 5.834 5.576 5.154 5.986 4.976 5.718 4.715
 5.221 4.698 4.597 4.530 4.909 4.089 3.984 3.692 4.030 4.686 4.348 4.024
 4.543 4.378 4.384 4.314 4.233 4.163 4.109 3.978 3.938 3.820 3.794 3.595
 3.486 3.570 3.398 3.316 3.621 3.576 3.184 3.456 3.112 3.334 0.102 0.241
 0.163 0.201 0.367 0.415 0.922 0.503 0.653 0.603 3.555 3.265 3.247 3.046
 3.151 2.996 1.238 1.115 0.889 0.792 0.536 0.018 0.003 0.054 0.032 0.393
 0.369 0.298 0.108 0.240 0.175 0.207 3.129 2.898 1.309 0.916 0.711 0.657
 0.591 0.132 0.216 3.358 1.456 0.977 0.843 0.760 0.765 0.460 0.460 0.051
 0.053 0.081 2.810 3.042 2.979 2.946 1.650 1.610 1.539 1.560 1.207 1.171
 1.140 1.143 0.019 0.009 0.001 0.031 2.659 1.703 2.884 2.854 2.545 2.291
 1.373 1.385 2.812 1.473 2.731 2.593 2.758 1.529 1.841 1.828 2.367 2.363
 1.983 1.912 2.089 2.039 2.168 2.151 -0.000 -0.000 1.560 2.667 2.662 0.008
 0.033 1.600 2.731 2.620 2.550 2.599 2.609 2.418 2.422 2.473 2.482 2.281
 2.266 2.219 2.222 1.713 1.726 2.009 2.026 1.917 1.854 1.823 1.762 1.797
 -0.000 0.000 0.126 0.071 2.368 2.088 2.129 0.194 0.284 0.378 0.485 0.608
 0.733 1.950 2.149 3.055 0.846 1.010 3.043 1.149 2.904 2.851 1.313 1.459
 2.723 1.618 2.491 2.405 1.760 1.924 2.060 2.165 2.611 0.000 0.000 0.000]
M eigenvectors:
[[-0.013 -0.078 0.230 … 0.000 0.000 0.000]
 [0.040 0.208 -0.505 … 0.000 0.000 0.000]
 [-0.086 -0.333 0.488 … 0.000 0.000 0.000]
 …
```

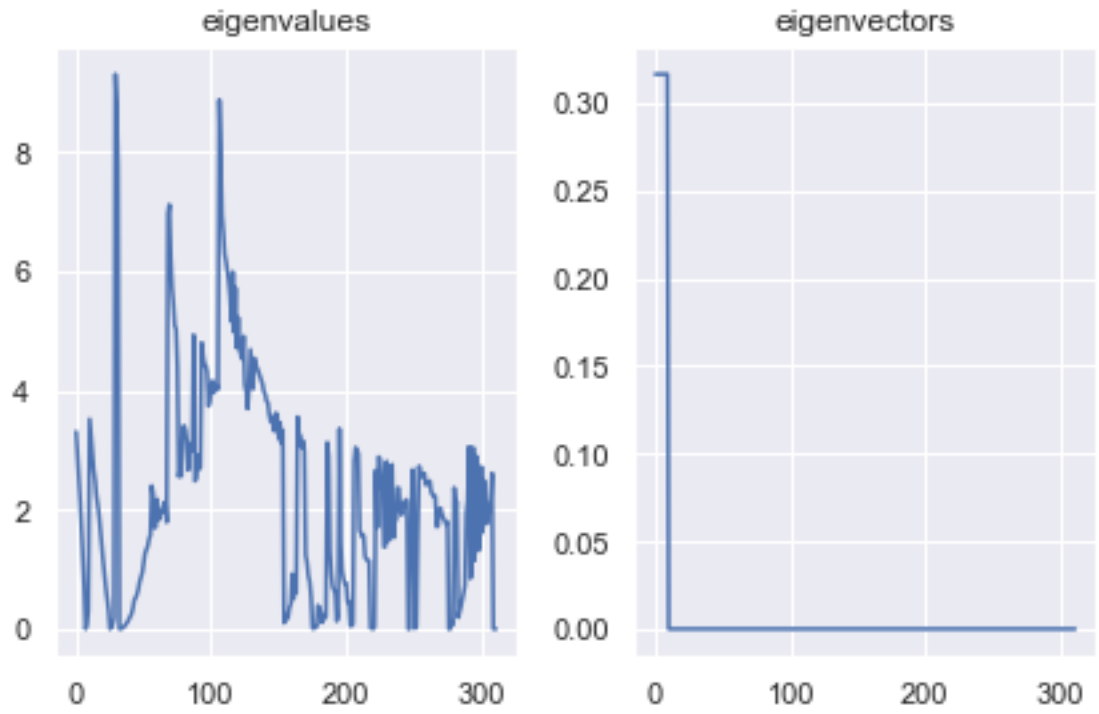
```
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 0.000]]
```

Question5 ($k = 20$, **SD**)

[64]:
```
# construct the matrix U
u=v+L
print("the U matrix is: ")
print(u)
```

```
the U matrix is:
[[0.788 -0.879 0.230 … 0.000 0.000 0.000]
 [-0.761 1.716 -1.212 … 0.000 0.000 0.000]
 [-0.086 -1.040 1.947 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 3.400 -0.250 -0.350]
 [0.000 0.000 0.000 … -0.250 3.300 -0.100]
 [0.000 0.000 0.000 … -0.350 -0.100 2.700]]
```

[65]:
```
fig = plt.figure()
ax1 = plt.subplot(121)
plt.plot(e)
ax1.title.set_text('eigenvalues')
i = np.where(e < 10e-6)[0]
ax2 = plt.subplot(122)
plt.plot(v[:, i[0]])
ax2.title.set_text('eigenvectors')
fig.tight_layout()
plt.show()
```



[66]:
```
df=pd.read_csv('D:/HW2/Spiral.csv',names=["a", "b", "cluster"],␣
 ↪encoding='latin-1')
dz=df.drop(['cluster'], axis=1)
```

Question 6,7,8,9 ($k = 20$, SD)

[67]:
```
#Compute the clusters for the Spiral dataset with K-means method
algorithm = (KMeans(n_clusters = 3,init='k-means++', n_init = 20 ,max_iter=100,
                    tol=0.0001,  random_state= 50  , algorithm='elkan') )
algorithm.fit(dz)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

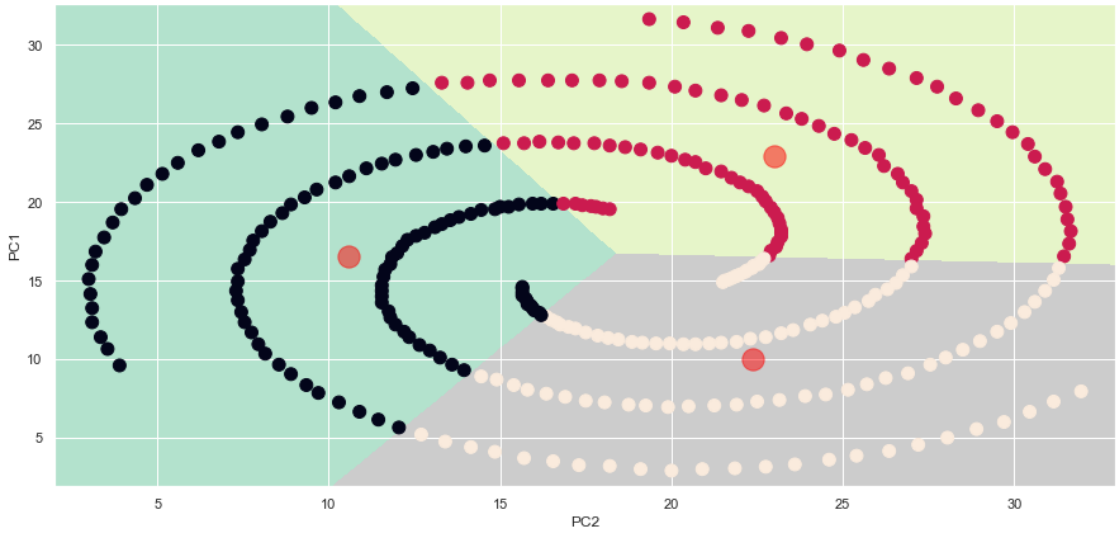
[68]:
```
X1 = dz[['a','b']].iloc[: , :].values
```

[69]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\base.py:451: UserWarning: X does not have valid feature names,
but KMeans was fitted with feature names
```

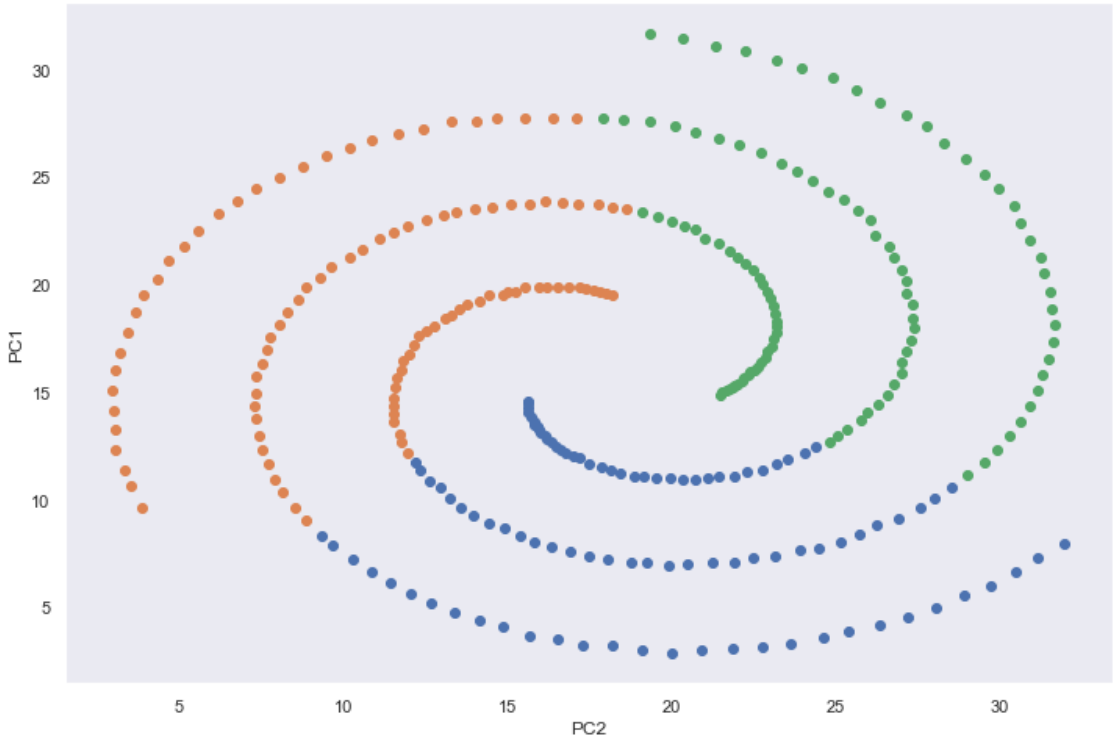
```
"X does not have valid feature names, but"
```

[70]:
```
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'a', y = 'b', data = dz, c = labels1, s = 100)
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c =␣
 ↪'red' , alpha = 0.5)
plt.ylabel('PC1') , plt.xlabel('PC2')
plt.show()
```



[71]:
```
#Compute the clusters for the Circle dataset with clustering method
from sklearn.cluster import SpectralClustering
sine_dataset = pd.DataFrame(dz)

## Fitting spectral clustering with 3 clusters
spectral_clustering = SpectralClustering(n_clusters = 3,
                                         affinity = 'nearest_neighbors',
                                         n_neighbors = 20).fit(dz)

## Appending cluster to the sine dataset
sine_dataset['spectral_clusters'] = spectral_clustering.labels_


## Visualizing the data with cluster
```

```
colors = ['lightblue', 'orange']

plt.figure(figsize = (12, 8))
plt.xlabel('PC2')
plt.ylabel('PC1')
plt.grid()

for c in sine_dataset['spectral_clusters'].unique():

        temp = sine_dataset[sine_dataset['spectral_clusters'] == c]
        plt.scatter(temp['a'], temp['b'])
```



5.3 $k = 40$ in Spiral dataset

Question1 ($k = 40$, SD)

[72]:
```
#construct the k-nearest neighborhood similarity graph and its adjacency matrix
from sklearn.neighbors import radius_neighbors_graph
W = radius_neighbors_graph(dz,1,mode='distance', metric='minkowski', p=40,␣
 ↪metric_params=None, include_self=False) #k=p=40
W = W.toarray()
print("The adjacency matrix is: ")
W
```

The adjacency matrix is:

[72]:
```
array([[0.000, 0.800, 0.000, …, 0.000, 0.000, 0.000],
       [0.800, 0.000, 0.701, …, 0.000, 0.000, 0.000],
       [0.000, 0.701, 0.000, …, 0.000, 0.000, 0.000],
       …,
       [0.000, 0.000, 0.000, …, 0.000, 0.250, 0.350],
       [0.000, 0.000, 0.000, …, 0.250, 0.000, 0.100],
       [0.000, 0.000, 0.000, …, 0.350, 0.100, 0.000]])
```

[73]:
```
#Constructing the degree matrix ( )
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[0.800 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 1.501 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 1.451 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 4.400 0.000 0.000]
 [0.000 0.000 0.000 … 0.000 3.300 0.000]
 [0.000 0.000 0.000 … 0.000 0.000 2.700]]
```

Question2 ($k = 40$, SD)

[74]:
```
from scipy.sparse import csgraph

# with the function below we can get the Laplacian Matrix(L=D-W) from the␣
 ↪adjacency matrix we have from previous section
L = csgraph.laplacian(W, normed=False)

print('The Laplacian Matrix L is: ')
print(L)

print('The first 5 values of L are: ')
print(L[:5,:5])
```

```
The Laplacian Matrix L is:
[[0.800 -0.800 -0.000 … -0.000 -0.000 -0.000]
 [-0.800 1.501 -0.701 … -0.000 -0.000 -0.000]
 [-0.000 -0.701 1.451 … -0.000 -0.000 -0.000]
 …
 [-0.000 -0.000 -0.000 … 4.400 -0.250 -0.350]
 [-0.000 -0.000 -0.000 … -0.250 3.300 -0.100]
 [-0.000 -0.000 -0.000 … -0.350 -0.100 2.700]]
The first 5 values of L are:
[[0.800 -0.800 -0.000 -0.000 -0.000]
 [-0.800 1.501 -0.701 -0.000 -0.000]
 [-0.000 -0.701 1.451 -0.750 -0.000]
```

```
[-0.000 -0.000 -0.750 1.550 -0.800]
[-0.000 -0.000 -0.000 -0.800 1.650]]
```

Question3 ($k = 40$, **SD**)

[75]:
```
#compute the cosine similarity to determine how many connected components there␣
 ↪are in the similarity graph
# import required libraries
from numpy.linalg import norm

# compute  similarity
cosine = np.dot(df['a'],df['b'])/(norm(df['a'],)*norm(df['b']))
print(" Similarity:", cosine)
```

```
Similarity: 0.8575754202341677
```

Question4 ($k = 40$, **SD**)

[76]:
```
e, v = np.linalg.eig(L)
# eigenvalues
print('eigenvalues:')
print(e)
# normalised eigenvectors
print('M eigenvectors:')
print(v)
```

```
eigenvalues:
[3.578 3.427 3.258 3.111 2.968 2.818 2.625 2.523 2.377 2.227 2.107 2.002
 1.839 1.654 1.505 1.373 1.191 1.045 0.905 0.750 0.627 0.498 0.392 0.292
 0.000 0.009 0.034 0.075 0.132 0.205 4.385 4.027 0.000 0.006 0.023 0.051
 0.110 0.146 0.230 0.257 0.382 0.523 0.589 0.728 0.854 0.989 3.403 3.383
 1.172 3.142 3.033 1.282 1.384 1.435 1.590 1.616 1.812 1.869 1.979 2.044
 2.140 2.260 2.369 2.654 2.663 2.546 2.554 2.537 9.280 8.769 0.000 0.061
 0.262 0.627 7.735 1.170 7.079 6.955 1.558 1.775 6.223 5.867 2.172 5.588
 5.412 2.462 2.581 2.763 2.800 2.995 3.213 3.345 3.450 3.646 5.027 4.939
 3.801 4.822 4.747 3.991 4.050 4.549 4.229 4.300 4.349 4.389 3.542 3.096
 3.133 2.999 2.862 2.791 2.686 2.617 2.486 2.402 2.303 2.117 1.969 1.814
 1.706 1.574 1.451 1.320 1.182 1.014 0.871 0.758 0.636 0.519 0.415 0.324
 -0.000 0.007 0.027 0.062 0.108 0.169 0.239 4.353 4.149 3.664 3.476 2.994
 0.000 0.007 0.035 0.089 0.168 0.260 0.332 0.459 0.602 2.965 0.758 0.950
 1.046 2.599 2.472 2.449 2.295 2.187 2.143 1.990 1.932 1.846 1.202 1.660
 1.580 1.508 1.409 0.000 0.048 0.193 0.447 6.319 5.965 0.774 5.334 1.165
 1.494 4.916 4.782 1.729 1.807 4.612 1.997 4.364 4.160 4.162 4.032 3.989
 3.811 3.636 2.172 3.529 3.428 2.344 2.419 2.443 3.236 3.089 3.001 2.601
 2.729 2.763 2.836 8.884 8.349 7.041 6.680 6.584 6.301 6.116 5.863 5.634
 5.203 5.110 4.958 4.826 4.556 4.461 4.159 4.007 3.984 3.628 3.548 3.450
 4.324 3.978 3.270 3.555 3.070 2.933 2.886 3.358 2.646 2.450 2.191 1.800
 1.613 1.403 2.060 1.045 0.630 1.010 0.732 0.607 0.283 0.485 0.377 0.049
 0.284 0.194 0.126 0.071 0.033 0.008 -0.000 0.000 -0.000 3.055 2.904 2.850
```

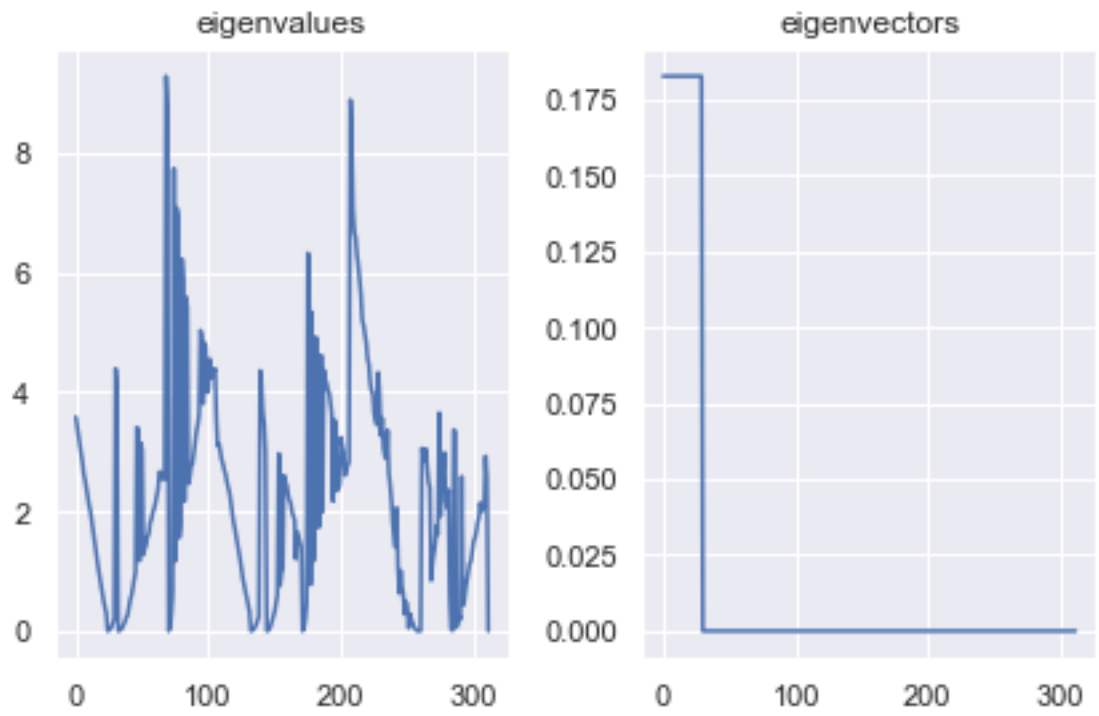
```
 3.043 2.722 2.490 2.404 0.845 1.148 1.312 1.458 1.759 1.617 3.649 1.923
 2.609 2.731 2.985 2.163 2.060 2.362 0.576 0.031 0.006 3.365 3.334 0.060
 0.124 0.252 0.196 2.584 0.430 0.515 0.720 0.850 0.971 1.129 1.171 1.335
 1.504 1.558 1.662 1.818 1.899 2.145 1.995 2.084 2.075 2.924 2.567 0.000]
M eigenvectors:
[[-0.001 -0.003 -0.011 … 0.000 0.000 0.000]
 [0.002 0.011 0.033 … 0.000 0.000 0.000]
 [-0.006 -0.028 -0.071 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 0.000 0.000 0.000]
 [0.000 0.000 0.000 … -0.000 -0.000 0.000]
 [0.000 0.000 0.000 … -0.000 -0.000 0.000]]
```

Question5 ($k = 40$ SD)

[77]:
```
# construct the matrix U
u=v+L
print("the U matrix is: ")
print(u)
```

```
the U matrix is:
[[0.799 -0.804 -0.011 … 0.000 0.000 0.000]
 [-0.798 1.512 -0.667 … 0.000 0.000 0.000]
 [-0.006 -0.728 1.380 … 0.000 0.000 0.000]
 …
 [0.000 0.000 0.000 … 4.400 -0.250 -0.350]
 [0.000 0.000 0.000 … -0.250 3.300 -0.100]
 [0.000 0.000 0.000 … -0.350 -0.100 2.700]]
```

[78]:
```
fig = plt.figure()
ax1 = plt.subplot(121)
plt.plot(e)
ax1.title.set_text('eigenvalues')
i = np.where(e < 10e-6)[0]
ax2 = plt.subplot(122)
plt.plot(v[:, i[0]])
ax2.title.set_text('eigenvectors')
fig.tight_layout()
plt.show()
```



[79]:
```
df=pd.read_csv('D:/HW2/Spiral.csv',names=["a", "b", "cluster"],␣
 ↪encoding='latin-1')
dz=df.drop(['cluster'], axis=1)
```

Question 6,7,8,9 ($k = 40$, SD)

[80]:
```
#Compute the clusters for the Spiral dataset with K-means method
algorithm = (KMeans(n_clusters = 3,init='k-means++', n_init = 40 ,max_iter=100,
                    tol=0.0001,  random_state= 50  , algorithm='elkan') )
algorithm.fit(dz)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

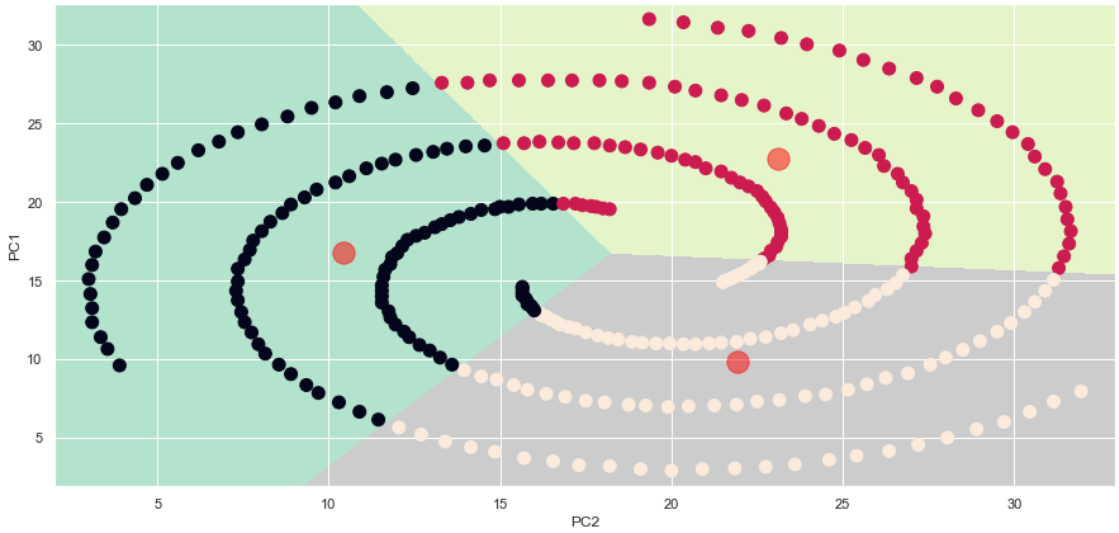
[81]:
```
X1 = dz[['a','b']].iloc[: , :].values
```

[82]:
```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
c:\users\sadaf\appdata\local\programs\python\python37\lib\site-
packages\sklearn\base.py:451: UserWarning: X does not have valid feature names,
but KMeans was fitted with feature names
```

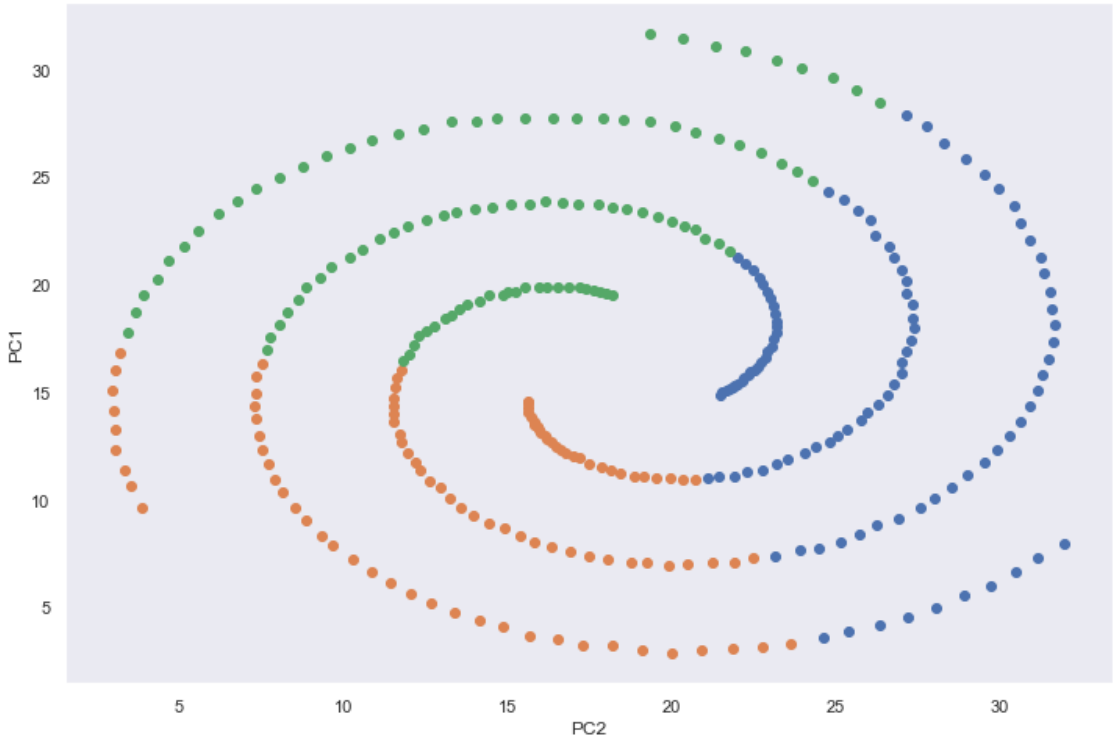
```
"X does not have valid feature names, but"
```

[83]:
```
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'a', y = 'b', data = dz, c = labels1, s = 100)
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c =␣
 ↪'red' , alpha = 0.5)
plt.ylabel('PC1') , plt.xlabel('PC2')
plt.show()
```



[84]:
```
#Compute the clusters for the Spiral dataset with clustering method
from sklearn.cluster import SpectralClustering
sine_dataset = pd.DataFrame(dz)

## Fitting spectral clustering with 3 clusters
spectral_clustering = SpectralClustering(n_clusters = 3,
                                         affinity = 'nearest_neighbors',
                                         n_neighbors = 40).fit(dz)

## Appending cluster to the sine dataset
sine_dataset['spectral_clusters'] = spectral_clustering.labels_


## Visualizing the data with cluster
```

```
colors = ['lightblue', 'orange']

plt.figure(figsize = (12, 8))
plt.xlabel('PC2')
plt.ylabel('PC1')
plt.grid()

for c in sine_dataset['spectral_clusters'].unique():

        temp = sine_dataset[sine_dataset['spectral_clusters'] == c]
        plt.scatter(temp['a'], temp['b'])
```



6 Conclution

In a nutshell, The ideal number of clusters employing the Elbow method was generally found to be three. By adjusting k, the clustering algorithm in the Circle dataset experiences barely perceptible variations in the K-means algorithm. While K=10 and K=20 show similar changes in the Spectral Clustering algorithm, the result is not very surprising since k-means generates convex clusters there are few differences between the two. However, K=40 made a clustering error and mistook some of the orange spots for blue. In the Spectral Clustering algorithm of the Spiral dataset, K=10 shows poorly grouped points, K=20 shows a small error in the green cluster, and K=40 shows perfectly clustered points. In the k-means, there is no clustering by altering k, and nearly all of the points are correctly categorized.

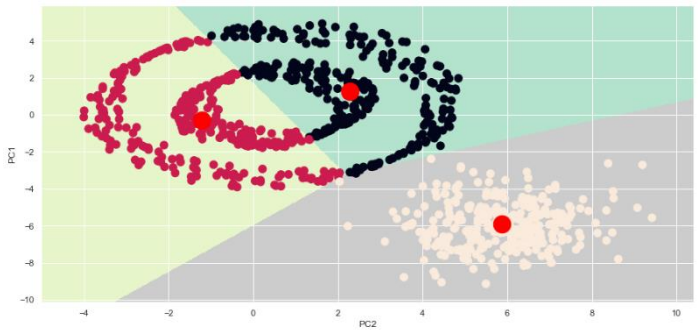
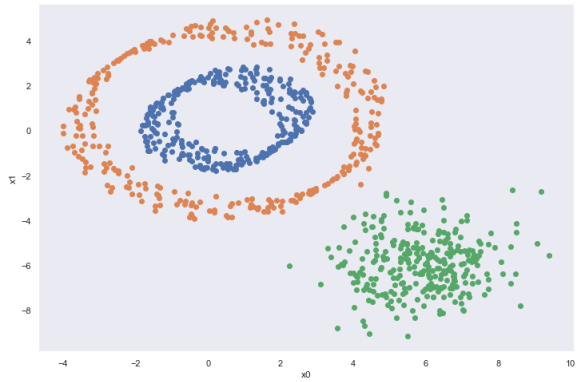
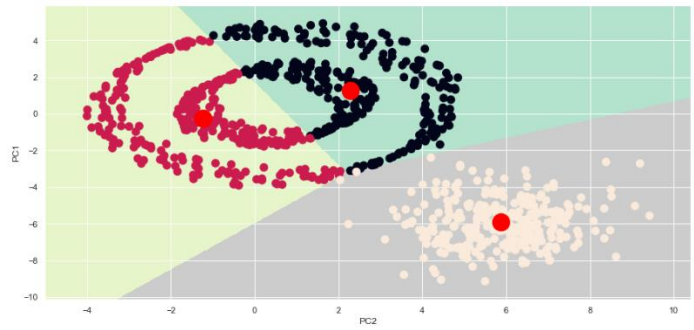
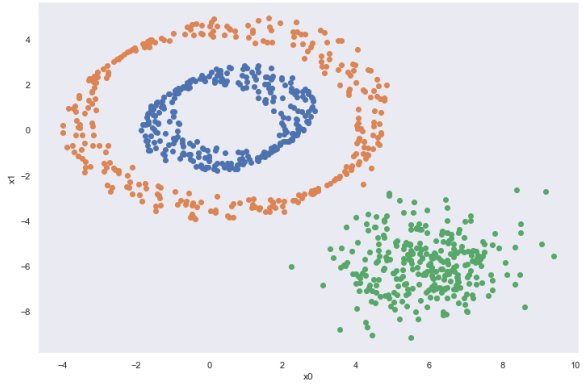
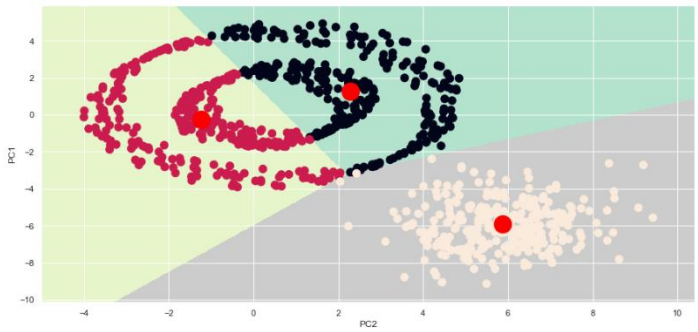
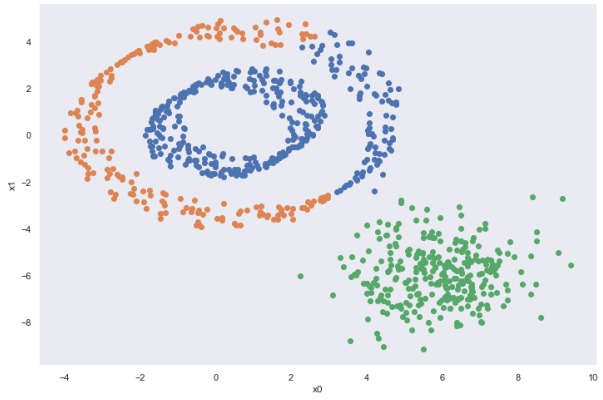
7 Future Prospect

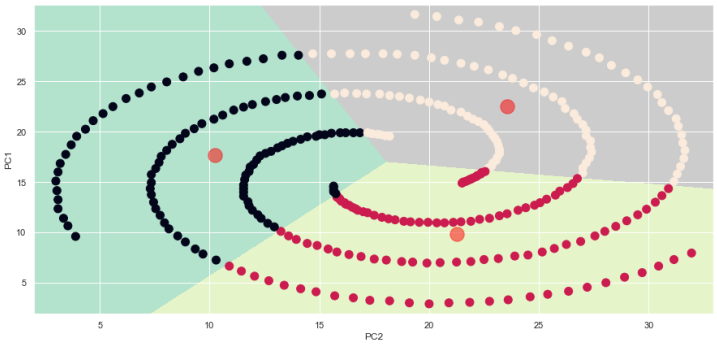
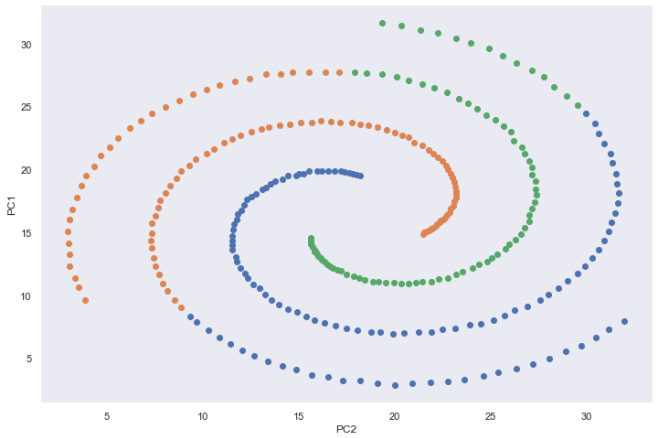
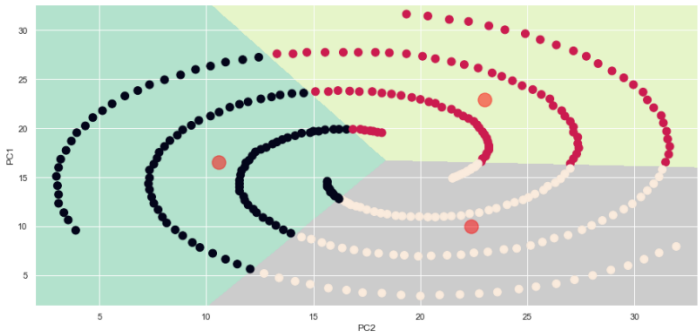
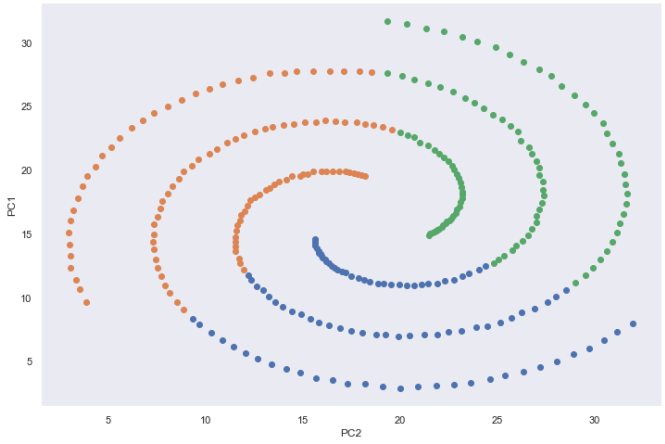
improving K-means clustering with a density-based clustering algorithm like DBSCAN

K-Means clustering may cluster loosely related observations together. Every observation becomes a part of some cluster eventually, even if the observations are scattered far away in the vector space. Since clusters depend on the mean value of cluster elements, each data point plays a role in forming the clusters. A slight change in data points might affect the clustering outcome. This problem is greatly reduced in DBSCAN due to the way clusters are formed. This is usually not a big problem unless we come across some odd shape data.

Another challenge with k-means is that we need to specify the number of clusters ("k") in order to use it. Much of the time, we won't know what a reasonable k value is a priori.

What's nice about DBSCAN is that we don't have to specify the number of clusters to use it. All we need is a function to calculate the distance between values and some guidance for what amount of distance is considered "close". DBSCAN also produces more reasonable results than k-means across a variety of different distributions.

Circle Dataset	
K-means algorithm	Spectral clustering algorithm
K=10	K=10
	
K=20	K=20
	
K=40	K=40
	

Spiral Dataset	
K-means algorithm	Spectral clustering algorithm
K=10	K=10
	
K=20	K=20
	
K=40	K=40