CS5691: Pattern Recognition and Machine Learning

Assignment 1

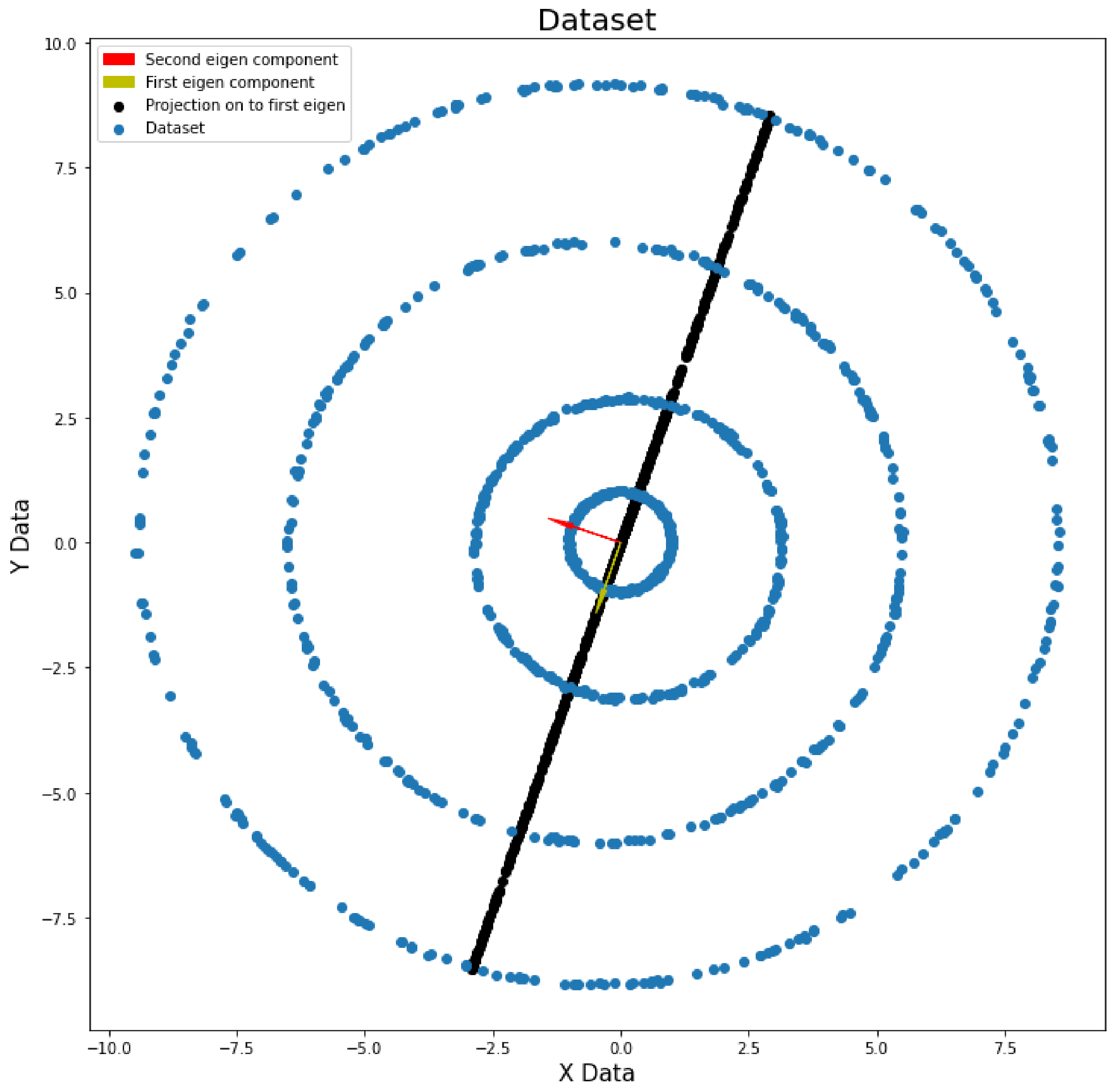
Course Instructor : Arun Rajkumar

Submitted by,
Muhammed Tharikh
cs22m058

Q1) You are given a data-set with 1000 data points each in R^2 .

i. *Write a piece of code to run the PCA algorithm on this data-set. How much of the variance in the data-set is explained by each of the principal components?*

On running normal PCA on the data-set I got below plot:



where blue dots represent the data, yellow vector (arrow) represent the direction of the first eigen component (eigenvector corresponding to highest eigenvalue), red vector (arrow) represent the second eigen component (eigenvector corresponding to second highest eigenvalue), and black dots represent the projection of data points onto the first eigen component (Basically the proxy of data points w.r.t first eigen component).

Variance of each eigen component is basically the eigenvalue:

$$\lambda_1 = \frac{1}{n}\sum_{i=1}^{n}(x_i^T w_1)^2$$

Where λ_1 represent the eigenvalue of first eigen component and the RHS represent the variance represented by w_1 (first eigen component), x_i represents a data point. So by this equation,

The variance corresponding to first component is (λ_1) : ***8565.95720122***

The variance corresponding to second component is (λ_2) : ***7244.80237467***

So the percent of variance in the data-set explained by the first eigen component is ***54.17802452885222 %*** and by the second eigenvector is ***45.82197547114778 %.***

***ii**. Study the effect of running PCA without centering the data-set. What are your observations? Does Centering help?*

The given data was almost centered as on seeing the mean of the data:

X 4.075000e-07 (mean along first feature)

Y 2.227000e-07 (mean along second feature)

As it can be seen that mean is very low (10^{-7}) on subtracting X and Y with their corresponding means, it doesn't change the dataset much, and eigenvalues, eigenvectors don't change (same as in the previous values). So centering doesn't help in this dataset, but if the dataset have not already been centered then centering helps as error minimization in centered data is variance maximization or in other words we can find the eigen components which passes through the origin, else we need to find a component which have been shifted by some value from the origin and needed much more sophisticated algorithms, so centering helps in using the already known algorithms (i.e find components which passes through origin).

***iii**. Write a piece of code to implement the Kernel PCA algorithm on this dataset. Use the following kernels :*

A. $k(x, y) = (1 + x^T y)^d$ *for d = {2, 3}*

B. $k(x, y) = exp^{\frac{-(x-y)^T(x-y)}{2\sigma^2}}$ *for σ = {0.1, 0.2, . . . , 1}*

Plot the projection of each point in the dataset onto the top-2 components for each kernel.
Use one plot for each kernel and in the case of (B), use a different plot for each value of σ.

Implemented Kernel PCA with both polynomial and radial basis and have obtained the plots as given below:

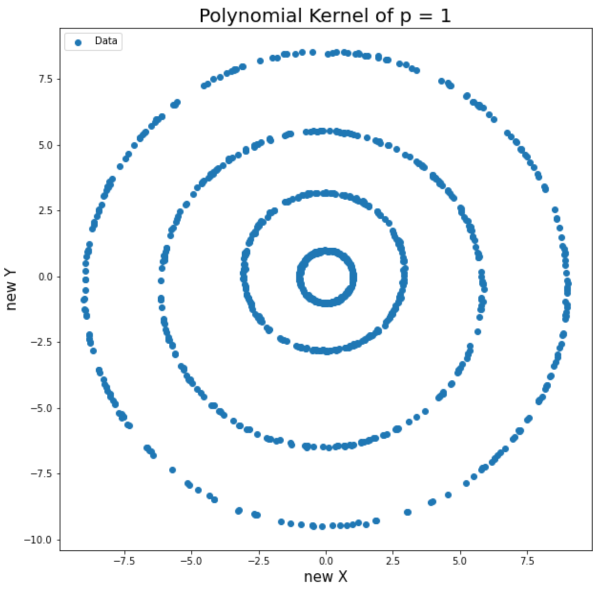


Fig 1. Polynomial kernel PCA (p=1)

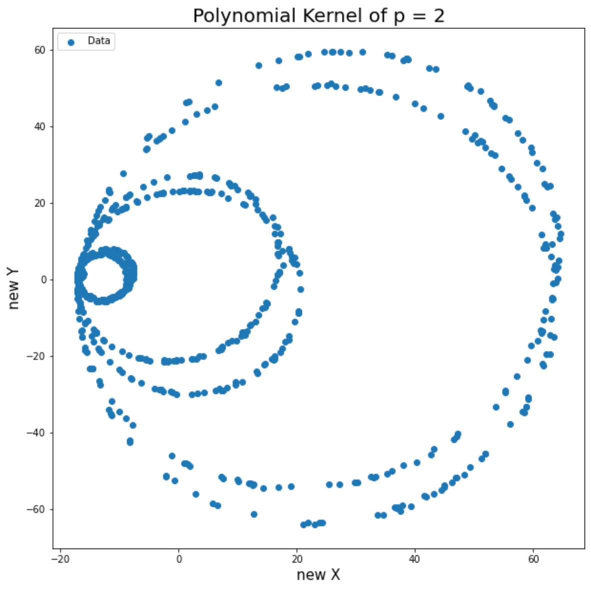


Fig 2. Polynomial kernel PCA (p=2)

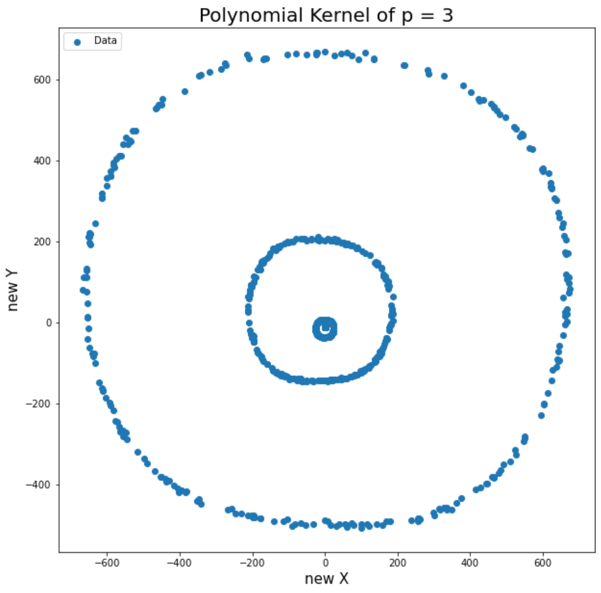


Fig 3. Polynomial kernel PCA (p=3)

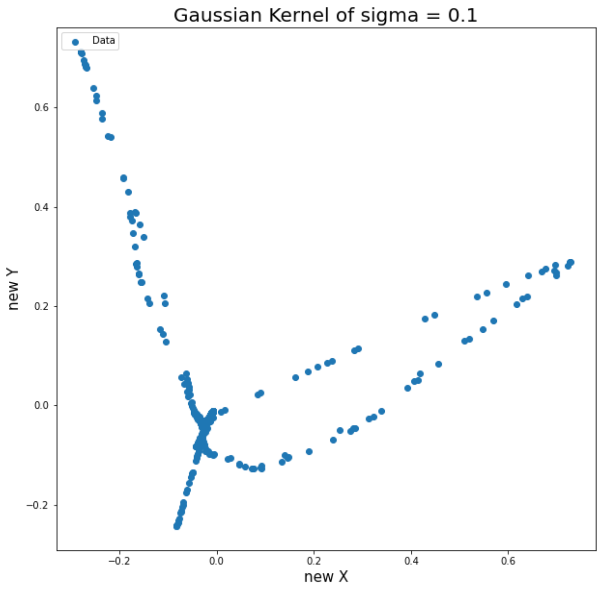


Fig 4. Gaussian kernel PCA (σ=0.1)

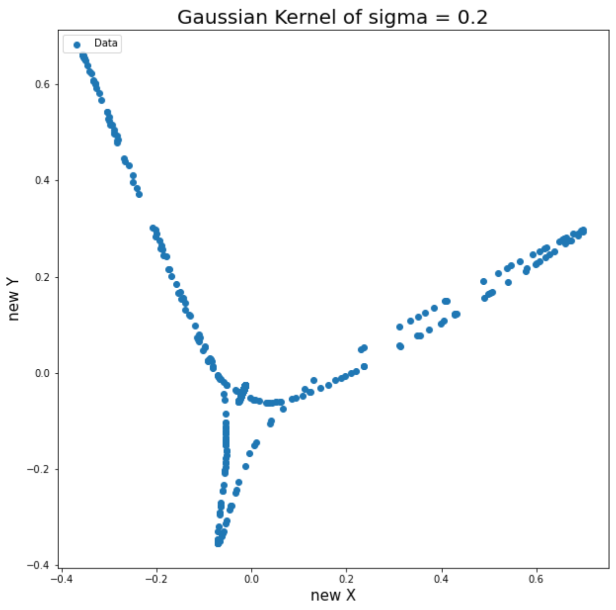


Fig 5. Gaussian kernel PCA (σ=0.2)

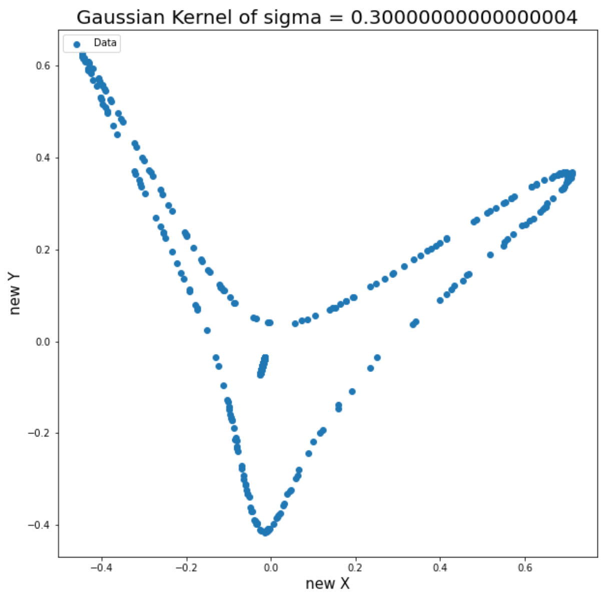


Fig 6. Gaussian kernel PCA (σ=0.3)

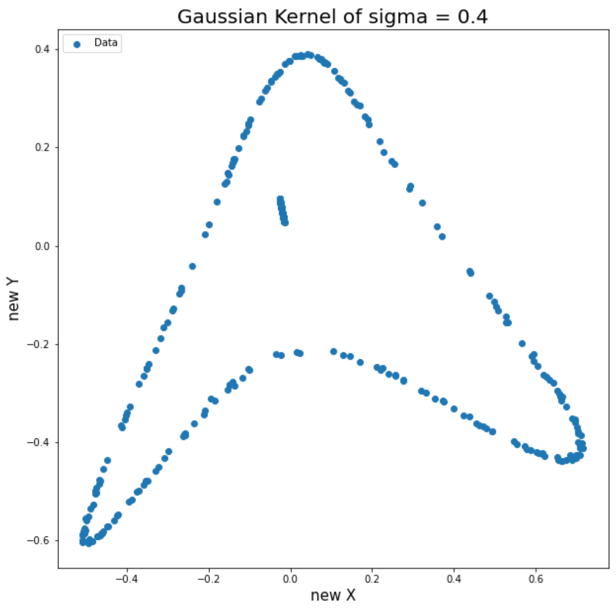


Fig 7. Gaussian kernel PCA (σ=0.4)

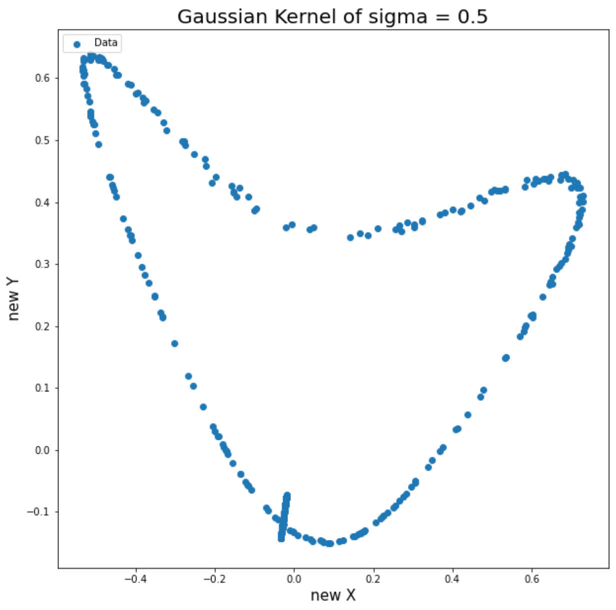


Fig 8. Gaussian kernel PCA (σ=0.5)

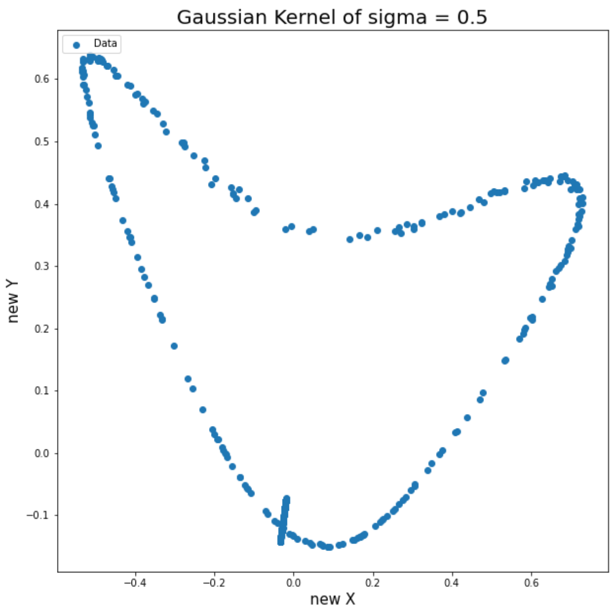


Fig 9. Gaussian kernel PCA (σ=0.5)

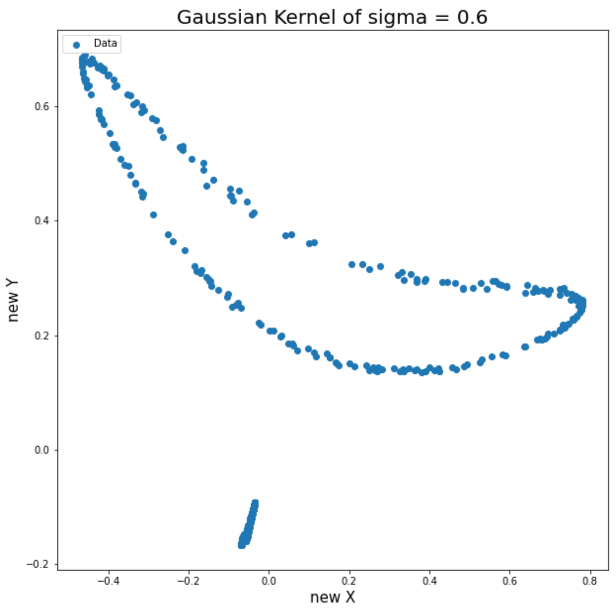


Fig 10. Gaussian kernel PCA (σ=0.6)

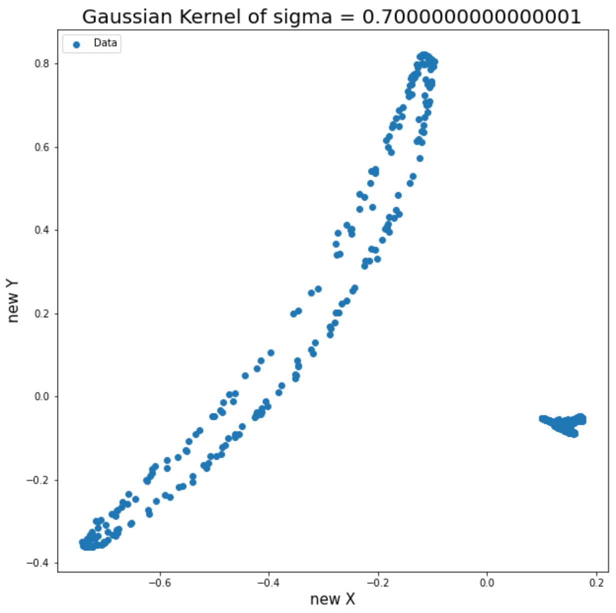


Fig 11. Gaussian kernel PCA (σ=0.7)

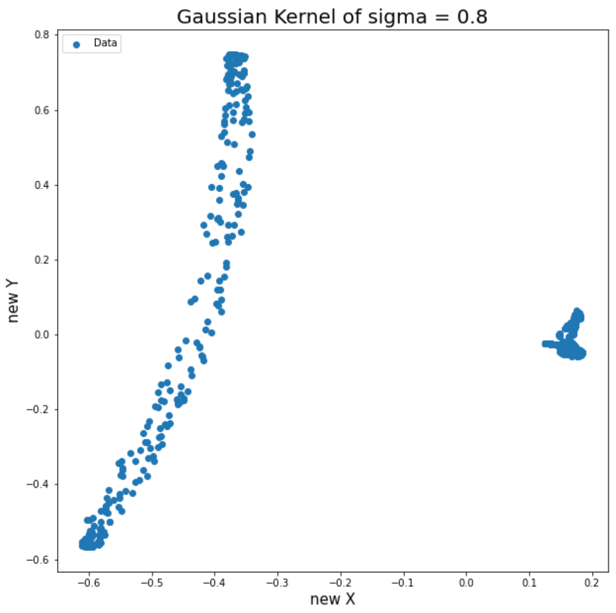
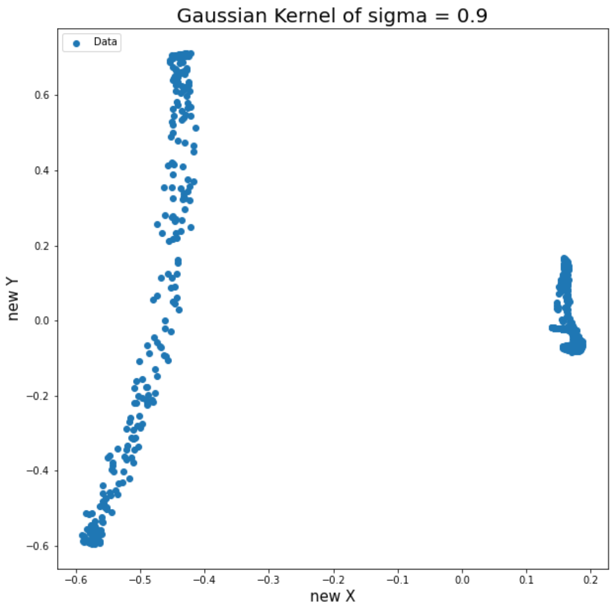


Fig 12. Gaussian kernel PCA (σ=0.8)



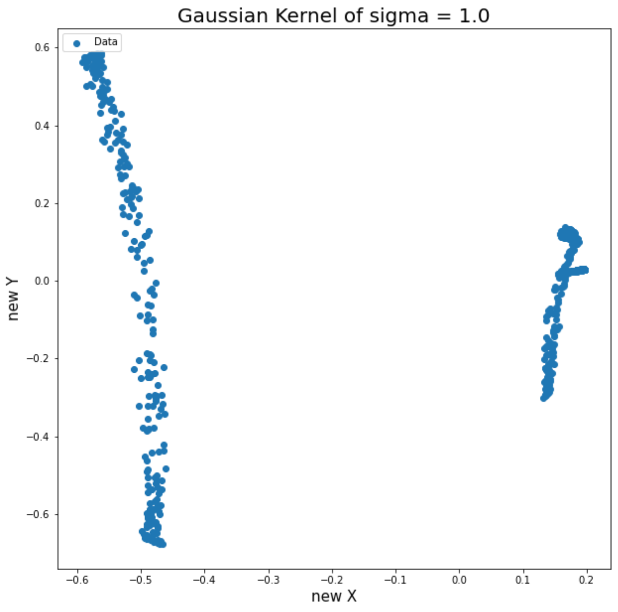


Fig 13. Gaussian kernel PCA (σ=0.9) Fig 14. Gaussian kernel PCA (σ=1)

Here new X is the projection of data onto the top eigen component by using the kernels and new Y is the projection of data onto the second highest eigen component.
Now when I use np.linalg.eigh for computing eigen value instead of np.linalg.eig then, I am getting a result which flips the above figures!, i.e eigenvectors are changing, so the directions are also changing

Polynomial kernels seem like it doesn't change the non-linearity of the dataset, i.e still forms like concentric circles! In the case of rbf, on high sigma values, it seems like it can be linearly separable (linearly approximated) and can easily be represented using small dimensions. Also for downstreaming tasks like clustering, rbf can be used as it can be linearly separable.

***iv**. Which Kernel do you think is best suited for this dataset and why?*

A Gaussian kernel (rbf) with $\sigma >= 0.6$ might be a good choice, as compared to other plots, it gives us linearly separable plots. So when pca is used before certain downstream tasks (like clustering, classification, etc) we can separate out the values in a linear fashion. Say we have a clustering job and on taking fig 14 (say), then we can cluster the data into 2 clusters easily after performing PCA on it.

In the polynomial kernel, data is not linearly separable (for p=2 it is very hard to separate it out and on p=3, it is similar to the dataset itself but has some changes in the number of concentric circles in the plot). In case of rbf with $\sigma <= 0.5$, linearly separability is less.

Q2) *You are given a data-set with 1000 data points each in R^2 .*

i. *Write a piece of code to run the algorithm studied in class for the K-means problem with $k = 4$. Try 5 different random initialization and plot the error function w.r.t iterations in each case. In each case, plot the clusters obtained in different colors.*

K-means have been implemented successfully with different initialization with k value fixed as 4.

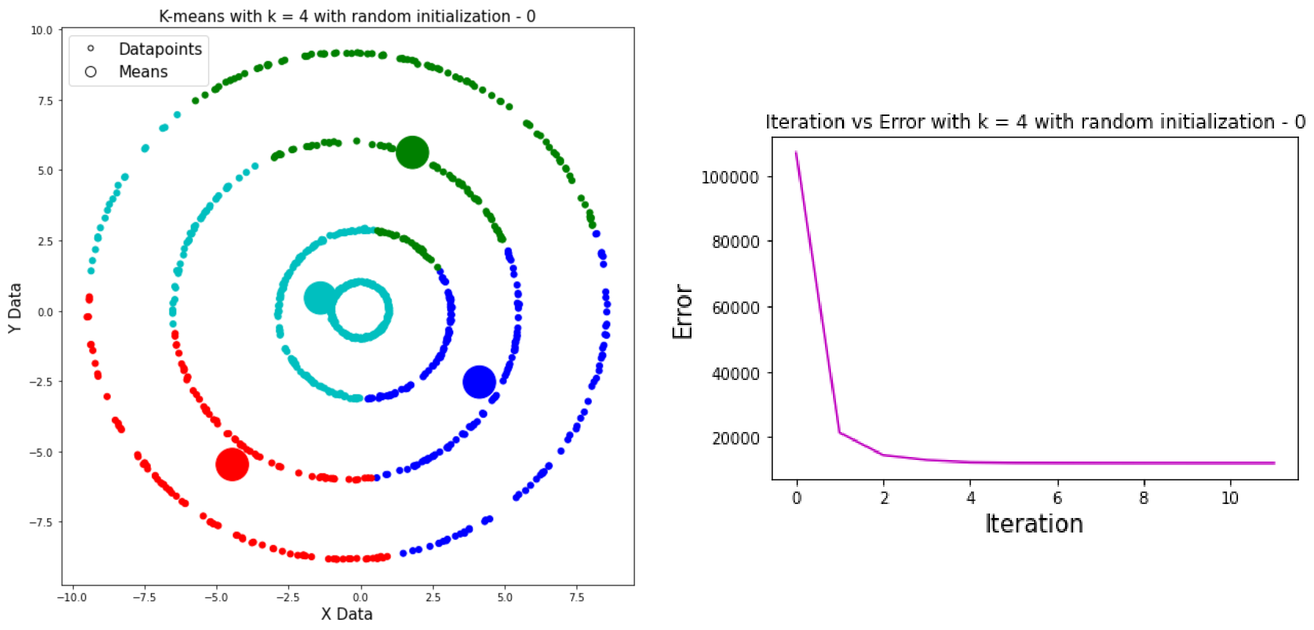
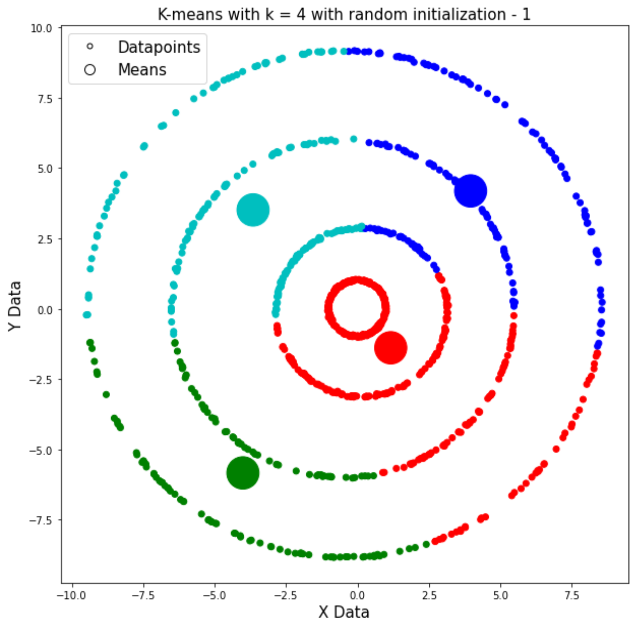


Fig 15. Random initialization 0



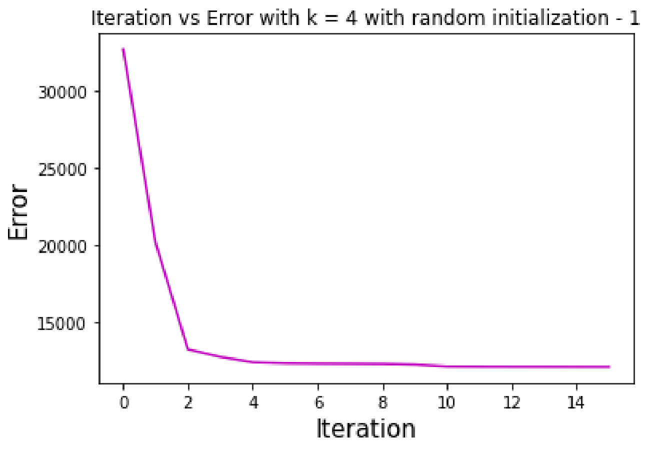
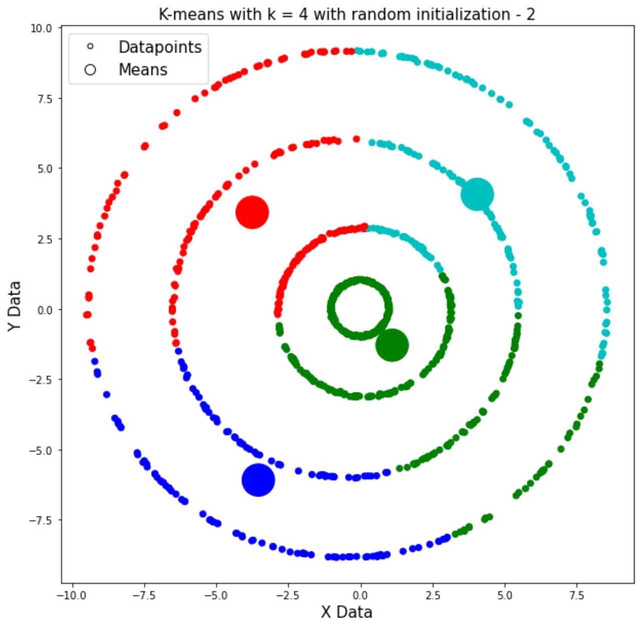


Fig 16. Random initialization 1



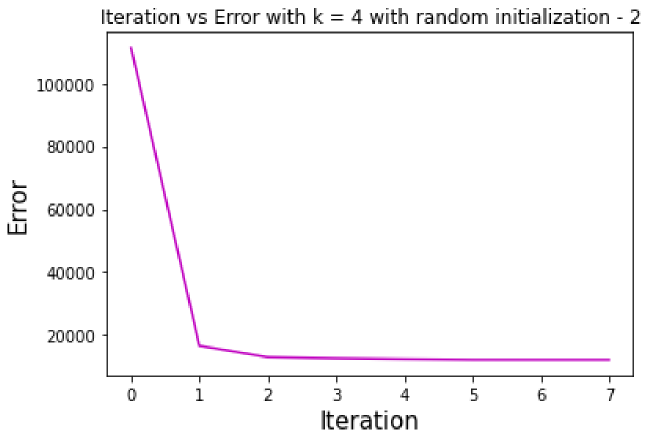
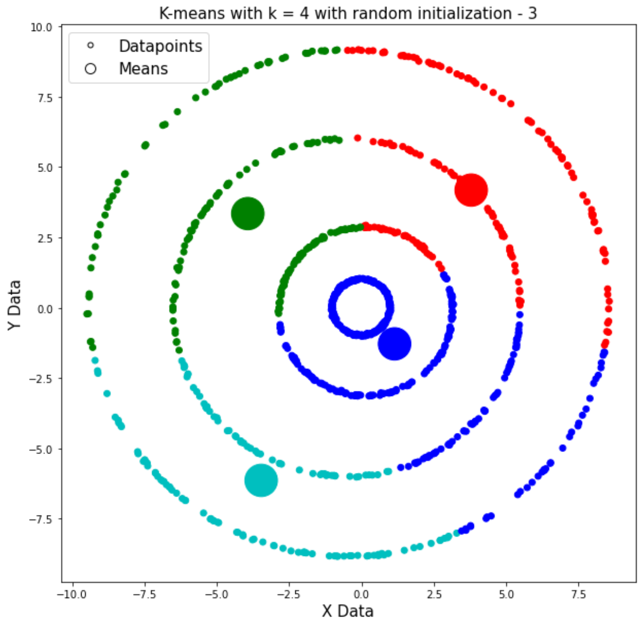


Fig 17. Random initialization 2



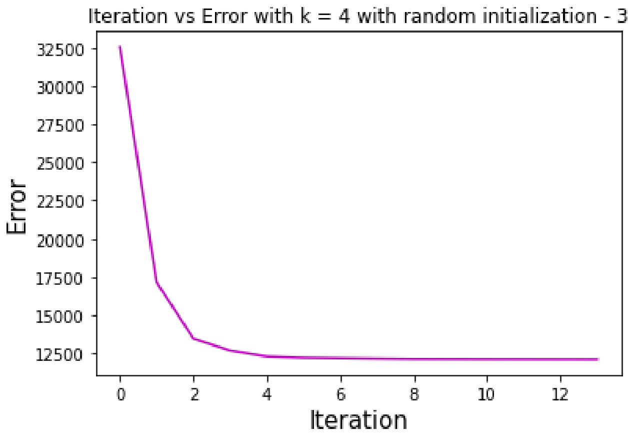
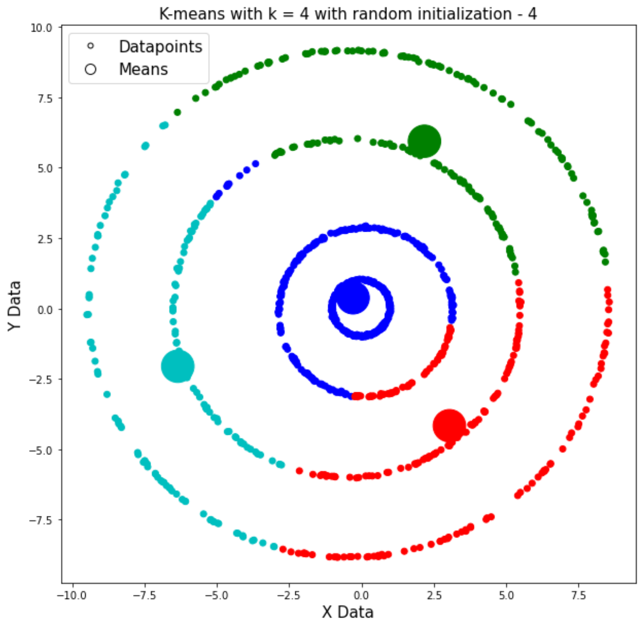


Fig 18. Random initialization 3



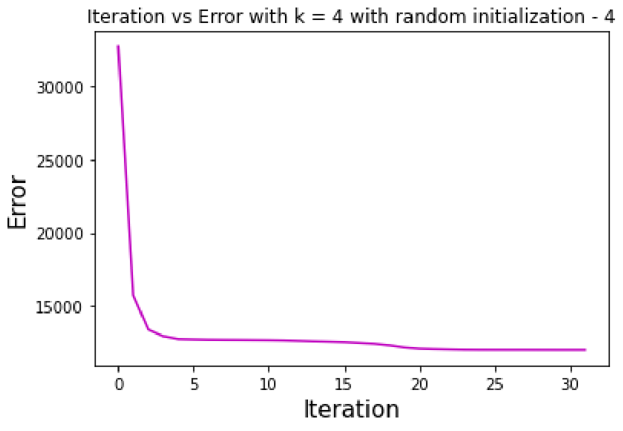


Fig 19. Random initialization 4

Here from fig 15,16,17,18,19 the left side represents the dataset (smaller circles) with their corresponding cluster assignment (same cluster have same color). The means are represented by bigger circles with color corresponding to its cluster.

Here based on the initialization values, cluster obtained varies (as it can be seen in the different figures obtained, where each cluster varies depending on the initial means), so k means depends on initialization of the means or the indicators.

Also on each iteration, the error decreases (here error is the distance of each data point from its corresponding mean), so it proves that k-means indeed converge (this is due to the fact that on each iteration, a datapoint changes its indicator only if the distance to that cluster's mean is less as compared to its own cluster mean and also on the fact that mean minimizes average of distances).

ii. *Fix a random initialization. For k = {2, 3, 4, 5}, obtain cluster centers according to the K-means algorithm using the fixed initialization. For each value of k, plot the Voronoi regions associated with each cluster center. (You can assume the minimum and maximum value in the data-set to be the range for each component of R^2).*

For fixed random initialization I used random seed to choose means randomly in a fixed manner, and obtained the initial cluster centers. After passing through the k-means algorithm, means of different clusters can be obtained, which is being used for finding voronoi regions.

Where voronoi is obtained by constructing grids of size 0.05 x 0.05 for values starting from -10 to 10 (as the data points lie between these points) in both x and y axis (basically splitting regions into uniform cells) and mapping each cell to each of the clusters.

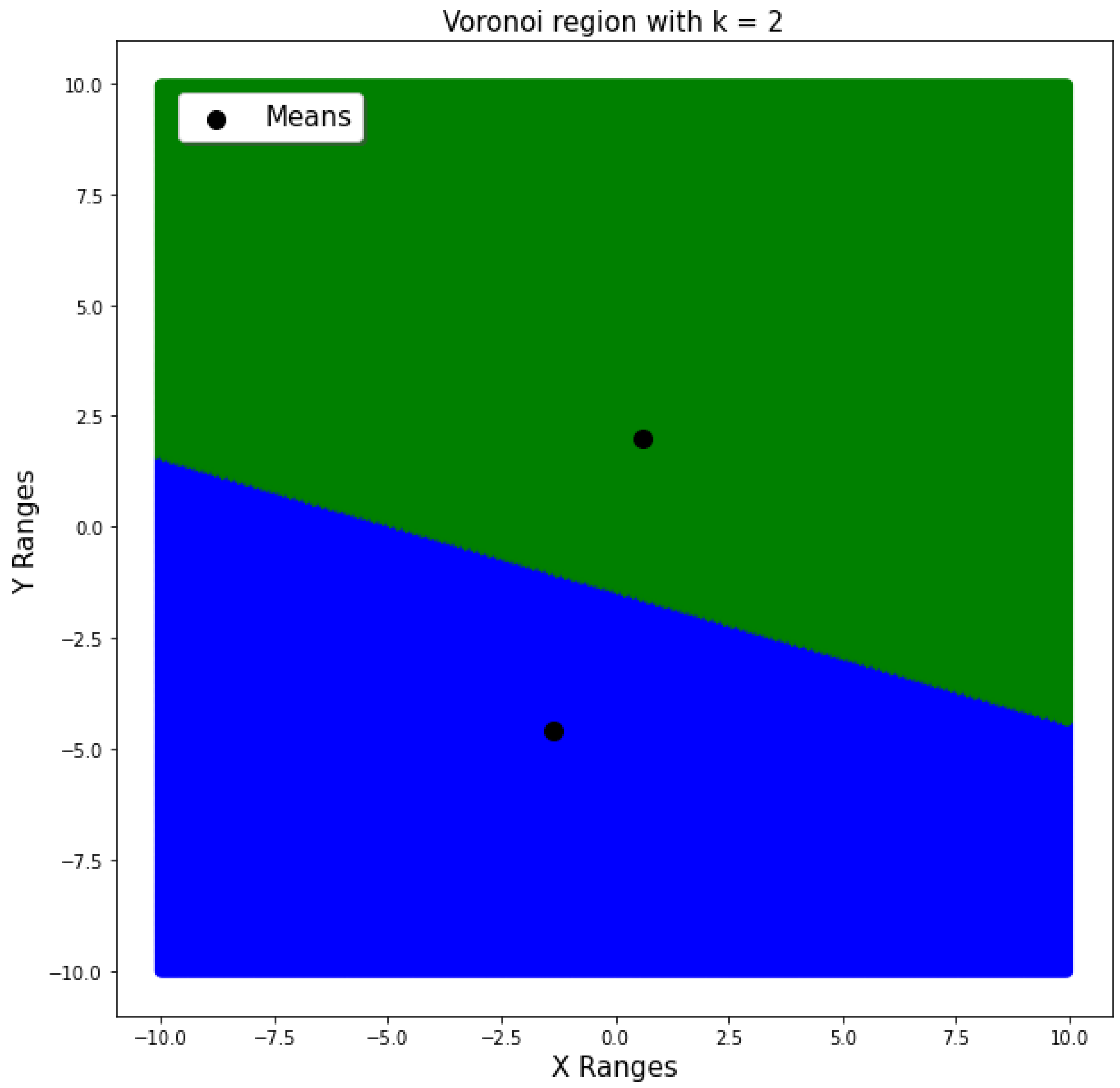


Fig 20. Voronoi region of k=2

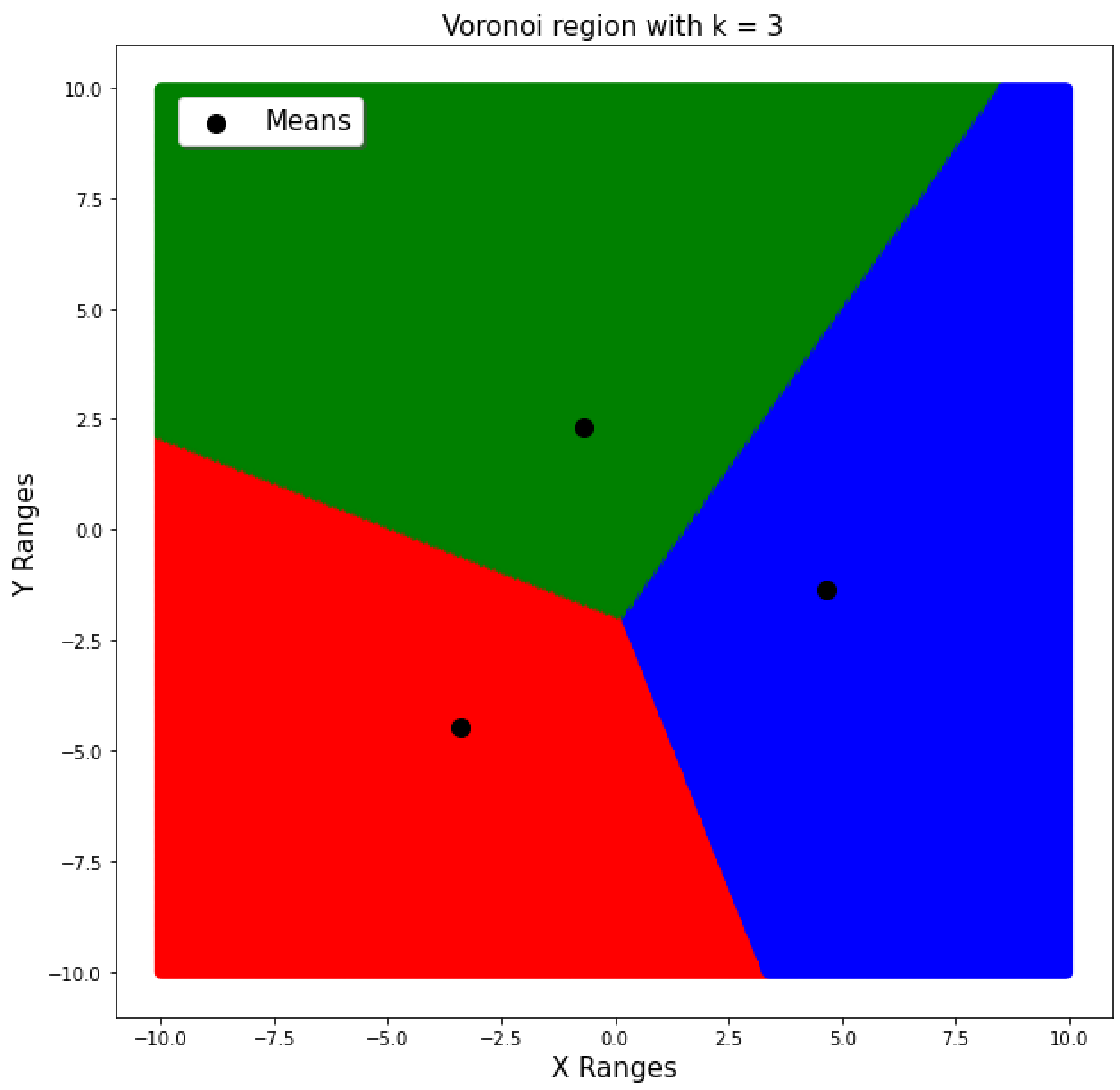


Fig 21. Voronoi region of k=3

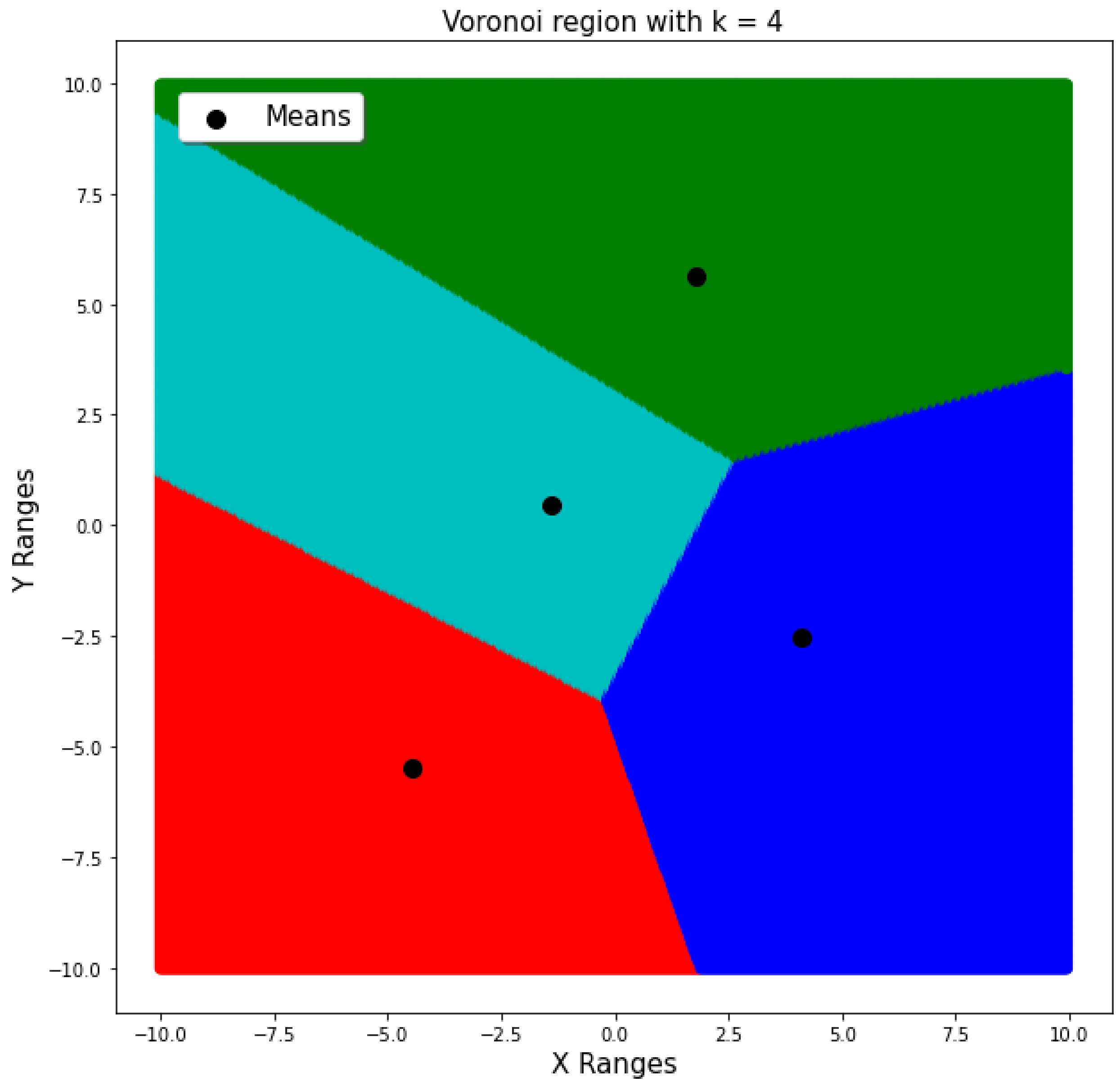


Fig 22. Voronoi region of k=4

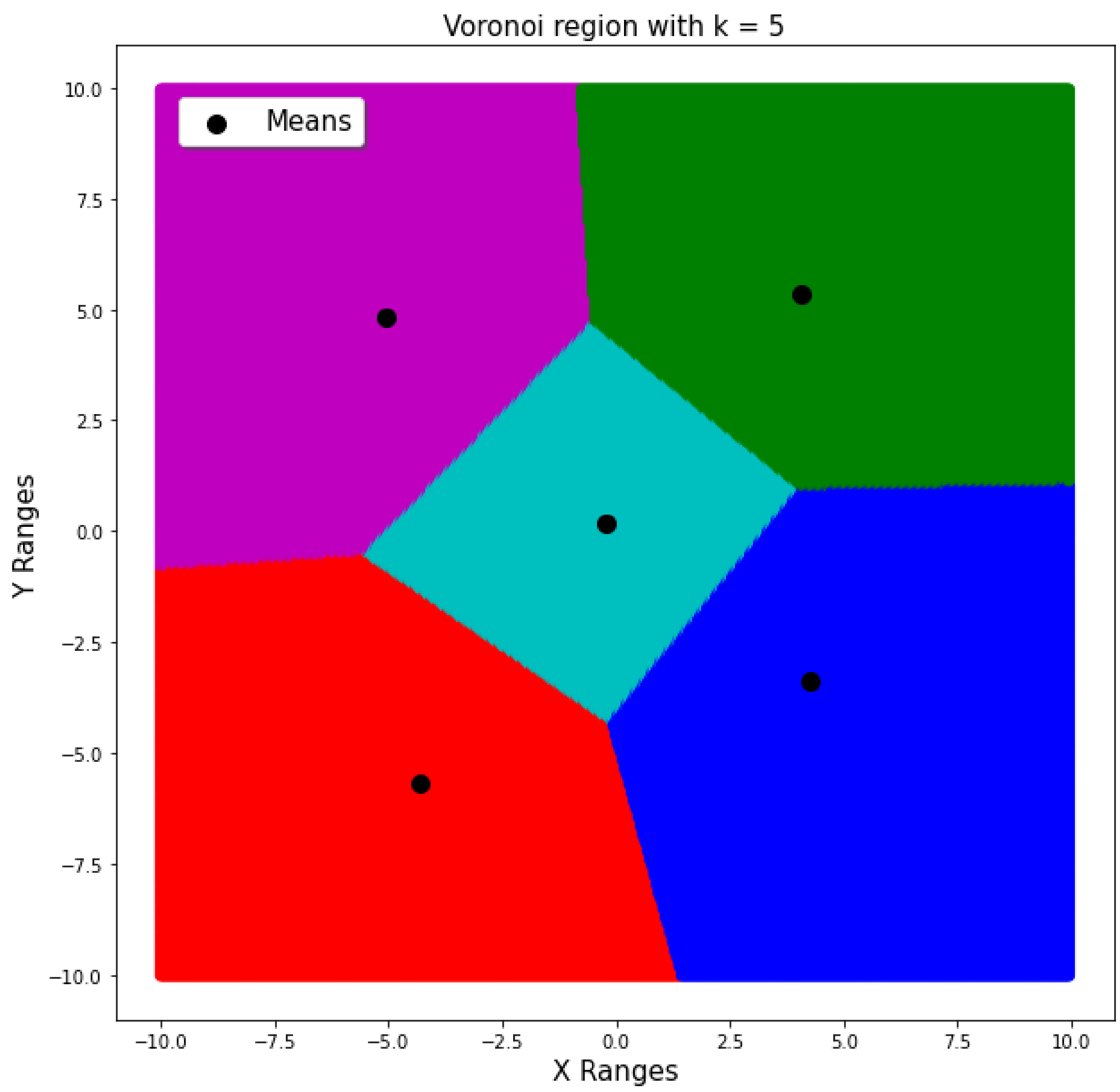


Fig 23. Voronoi region of k=5

Here black dot represents the mean of the corresponding cluster after applying the k-means algorithm, and color shows the voronoi region of the clusters (each color denotes voronoi region of its own cluster).

When k=2, 2 clusters are obtained and voronoi regions can be seen as two parts, similarly for k=3,4,5.

***iii**. Run the spectral clustering algorithm (spectral relaxation of K-means using Kernel-PCA) k = 4. Choose an appropriate kernel for this data-set and plot the clusters* *obtained in different colors. Explain your choice of kernel based on the output you obtain.*

I used polynomials (p=1,2,3) and rbf with σ = *{0.1 to 1}* and implemented the spectral clustering. The plot corresponding to this clustering is given below :

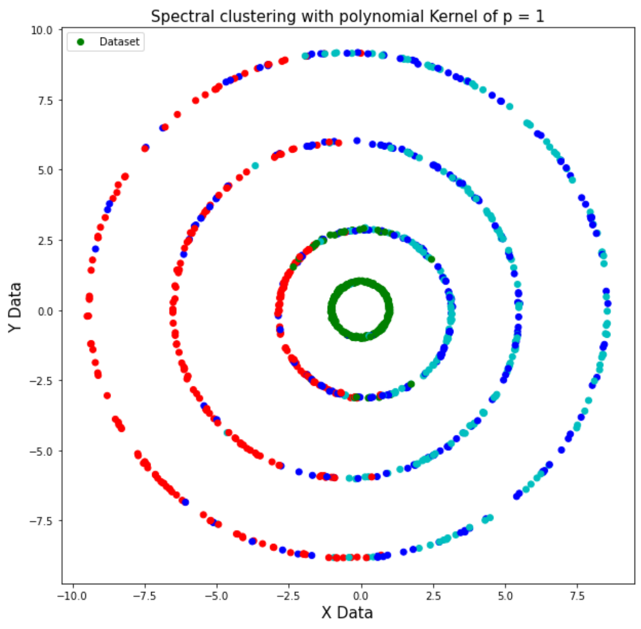


Fig 24 Spectral-polynomial with p=1

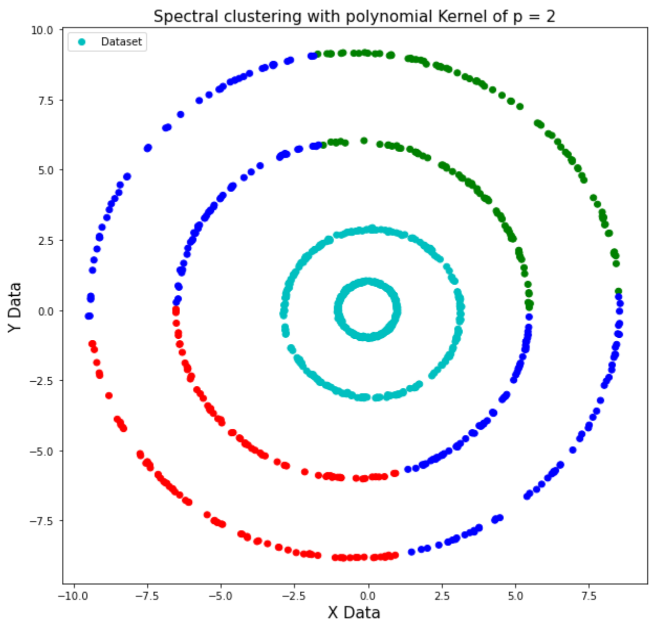


Fig 25 Spectral-polynomial with p=2

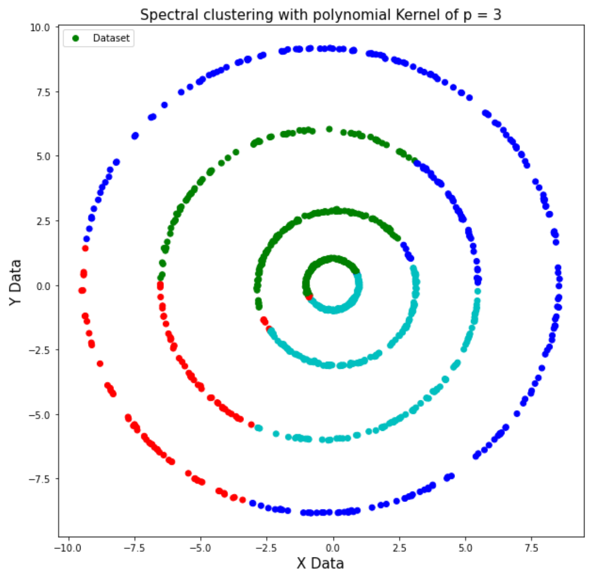


Fig 26 Spectral-polynomial with p=3

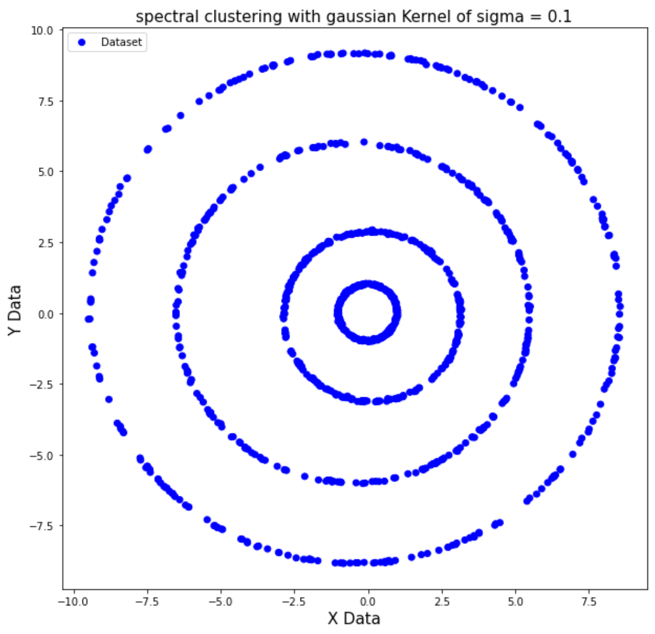


Fig 27 Spectral-rbf with σ=0.1

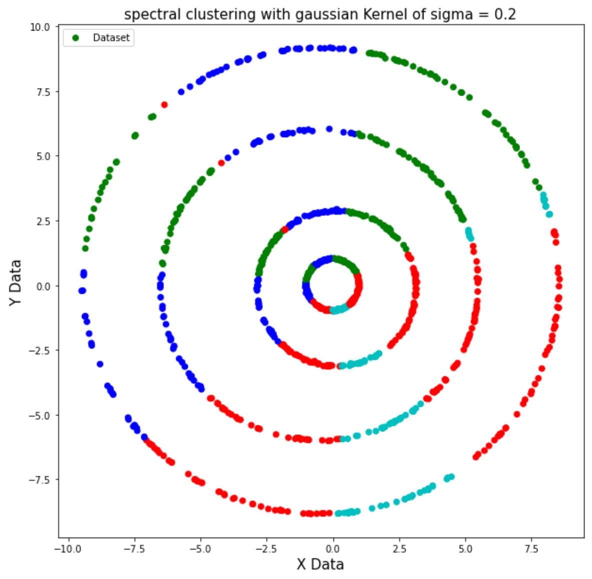


Fig 28 Spectral-rbf with σ=0.2

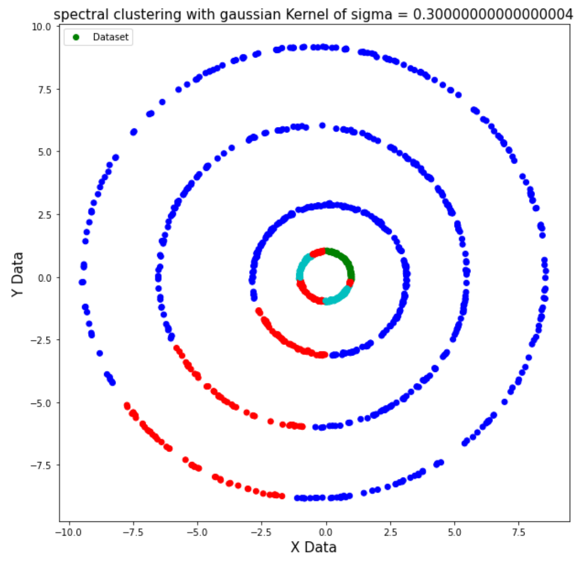


Fig 29 Spectral-rbf with σ=0.3

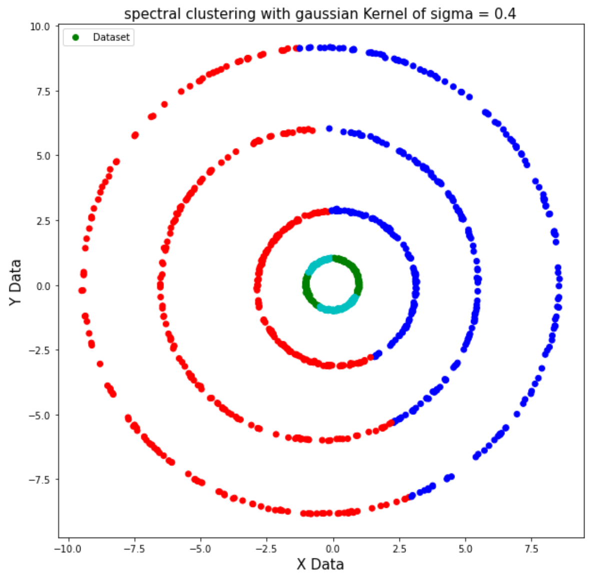


Fig 30 Spectral-rbf with σ=0.4

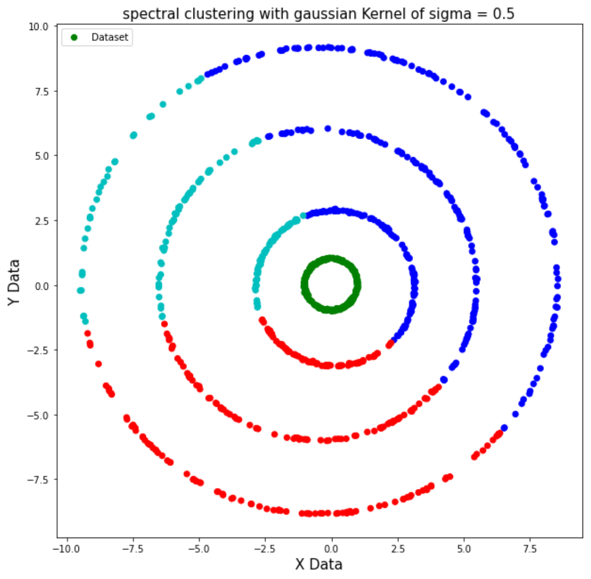


Fig 31 Spectral-rbf with σ=0.5

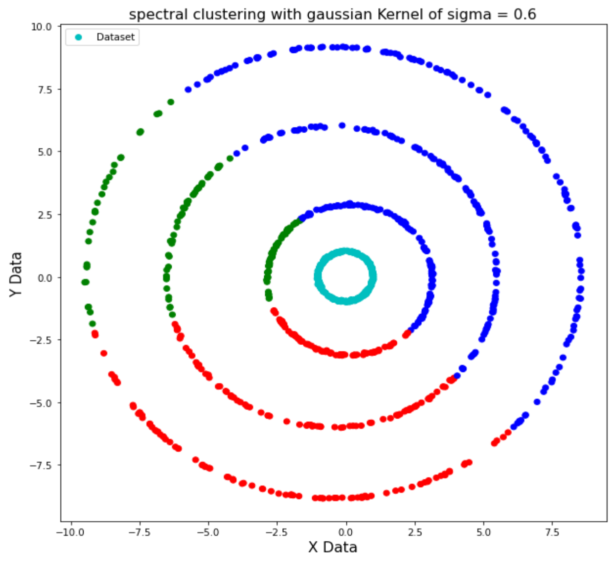


Fig 32 Spectral-rbf with σ=0.6

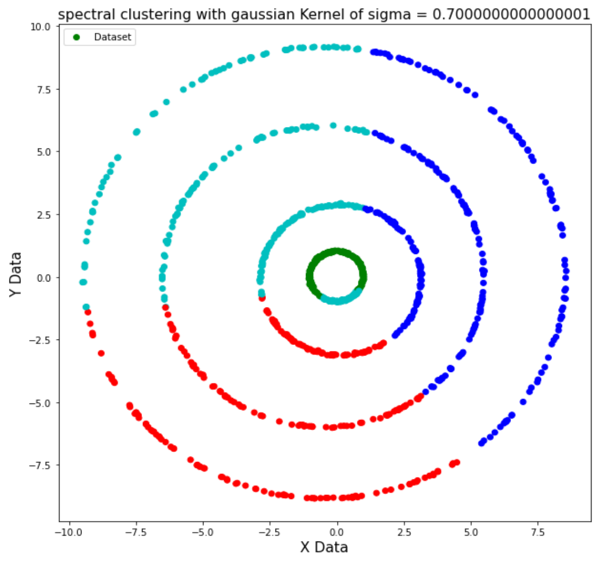


Fig 33 Spectral-rbf with σ=0.7

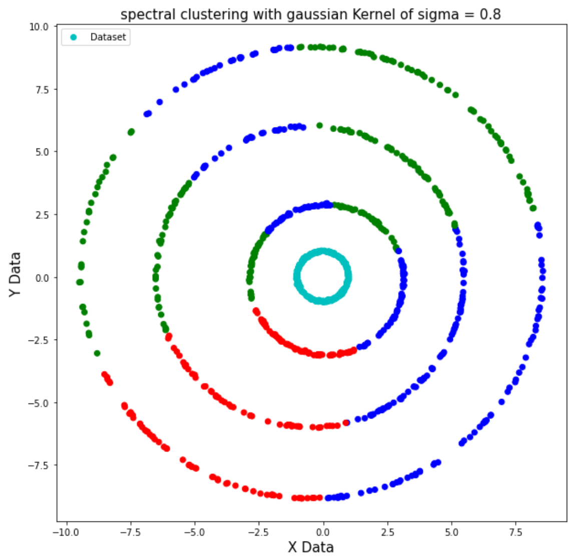


Fig 34 Spectral-rbf with σ=0.8

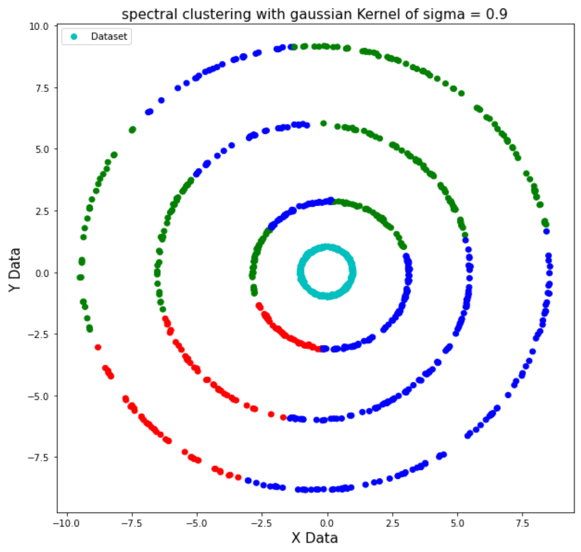


Fig 35 Spectral-rbf with σ=0.9

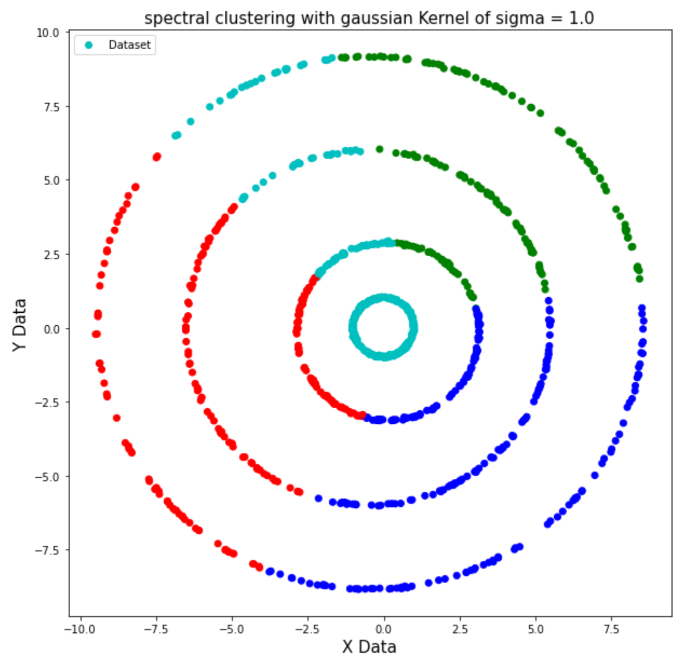


Fig 36 Spectral-rbf with σ=1

For spectral clustering using polynomial = 2 it gives good results, as it clusters inner two circles together, and outer circles are clustered like segments as can be seen in fig 25. For polynomial = 3, it doesn't give good results as compared to polynomial 2.

For radial basis, on σ=0.1, all data points are clustered as 1. And for σ *=0.2,0.3,0.4* it clustered as a kind of segment. For σ>=5 , the inner circle is clustered correctly, and the outer circle clustered like sectors or are grouped together, but with no unevenness. When σ is taken to be high value, say range between (2,20), clustering doesn’t change!, i.e, the cluster obtained from σ=10 is the same as the cluster obtained in σ=20.
Polynomial kernel with p=2, and a radial basis with σ>=0.5 can cluster better than the rest of the kernels!, but none of them can properly cluster the data points correctly(correct in the sense each concentric circle corresponds to each cluster).

Here I have taken top k eigenvectors to form the H matrix (here “k” is not same as the number of clusters, k is the number of eigenvectors used to construct the H matrix so that the row of H matrix will be the new data point) and k here is 8, I have tried to take top k eigenvectors where k is in range [3,16], but none of them clusters better as compared to k=8. k=6 also provided good results, but k=8 outperformed all!

I think, on going to higher dimension, a linear separation of all the concentric circle was not possible for the above kernels, so finding a proper kernel which can take the low dimensional data to high dimension where data can be separated linearly and then cluster them together and then mapping to low dimension- if that kernel is obtained, then it can be smoothly clustered together in a concentric way.

iv. *Instead of using the method suggested by spectral clustering to map eigenvectors to cluster assignments, use the following method: Assign data point i to cluster l whenever*

$$l = arg\,max_{j=1,\ldots,k} v_i^j$$

where $v^j \in R^n$ is the eigenvector of the Kernel matrix associated with the j-th largest eigenvalue. How does this mapping perform for this dataset?. Explain your insights.

I used polynomials (p=1,2,3) and rbf with σ = *{0.1 to 1}* and implemented the spectral clustering. The plot corresponding to this clustering is given below :

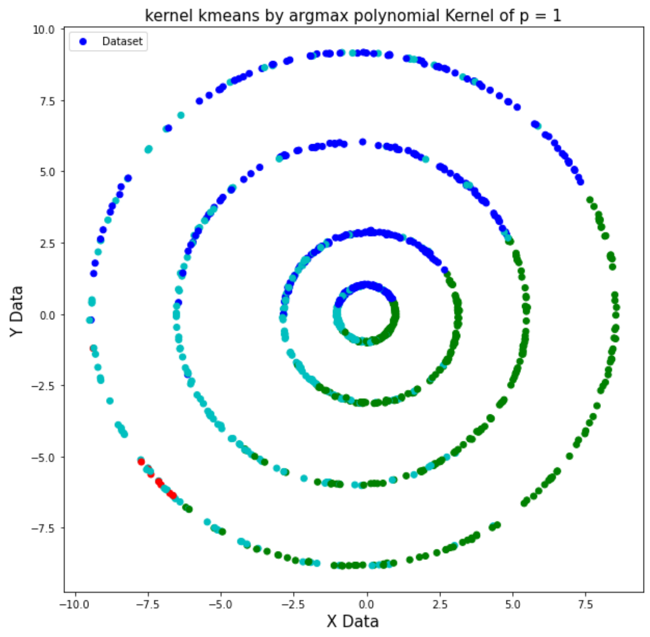


Fig . 37 poly kernel by argmax of poly = 1

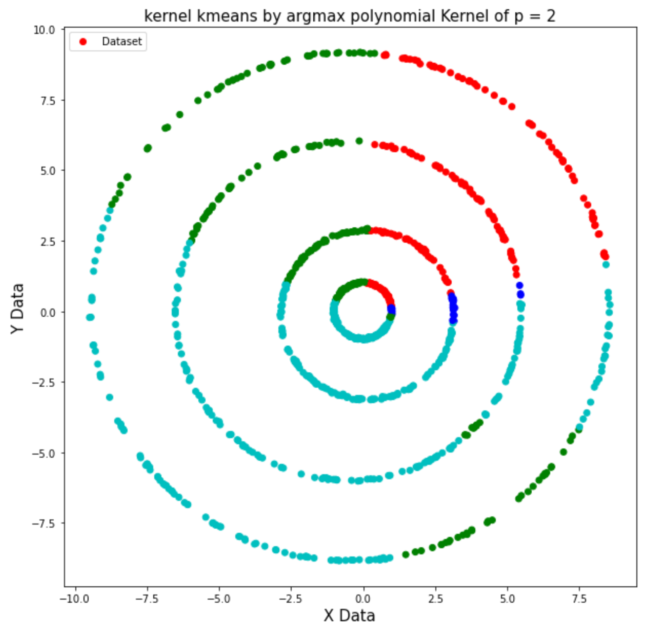


Fig . 38 polykernel by argmax of poly = 2

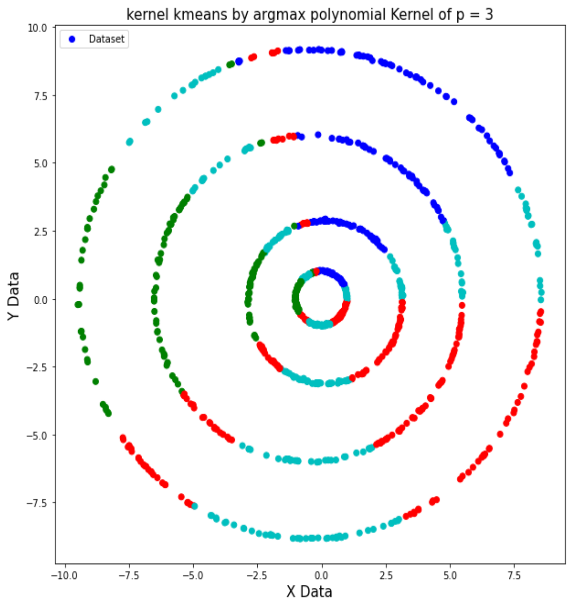


Fig . 39 poly kernel by argmax of poly = 3

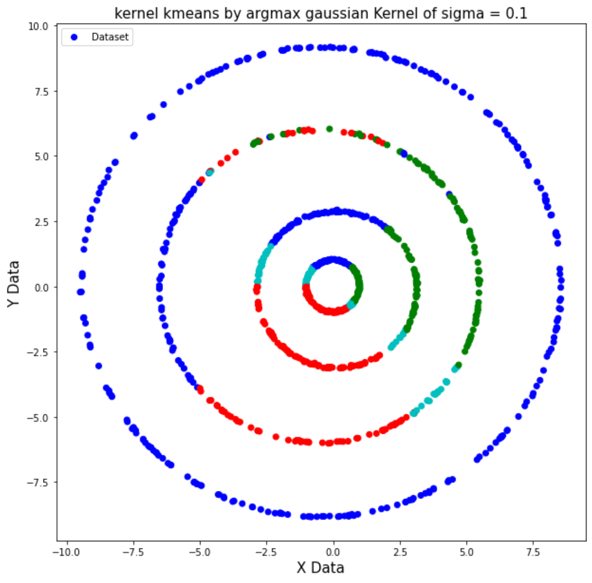


Fig . 40 rbf kernel by argmax of $\sigma = 0.1$

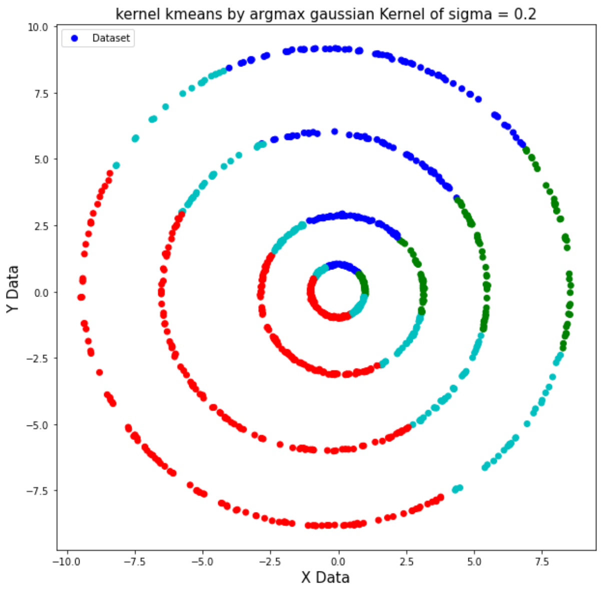


Fig . 41 rbf kernel by argmax of $\sigma = 0.2$

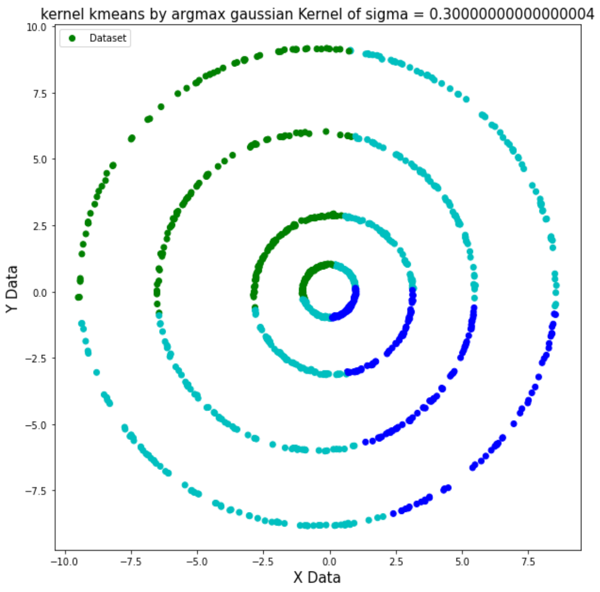


Fig . 42 rbf kernel by argmax of $\sigma = 0.3$

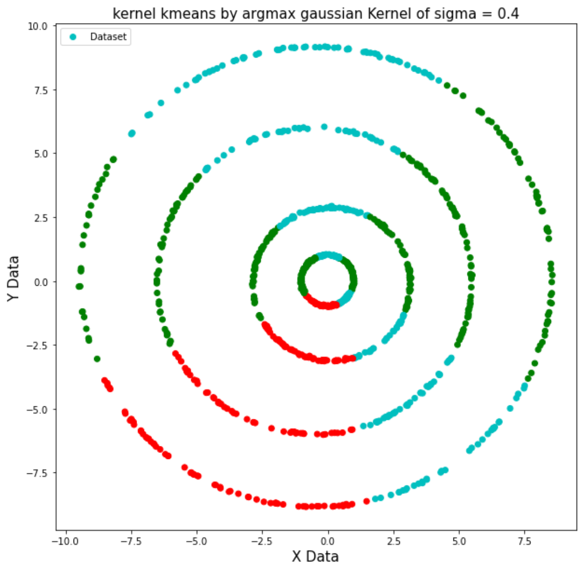


Fig 43. rbf kernel by argmax of $\sigma = 0.4$

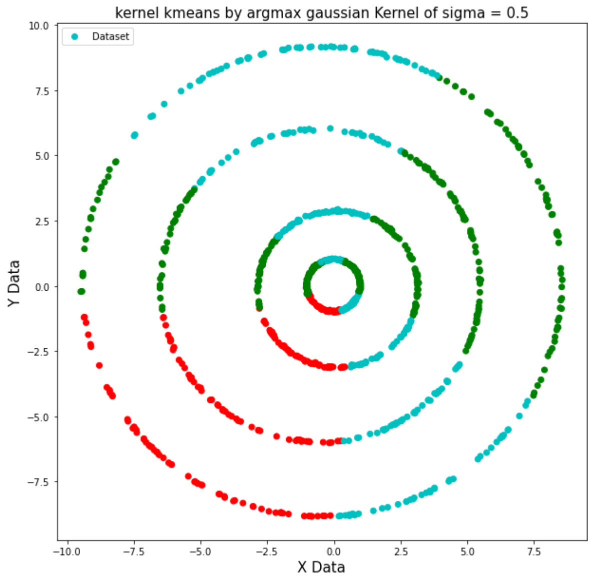


Fig 44. rbf kernel by argmax of $\sigma = 0.5$

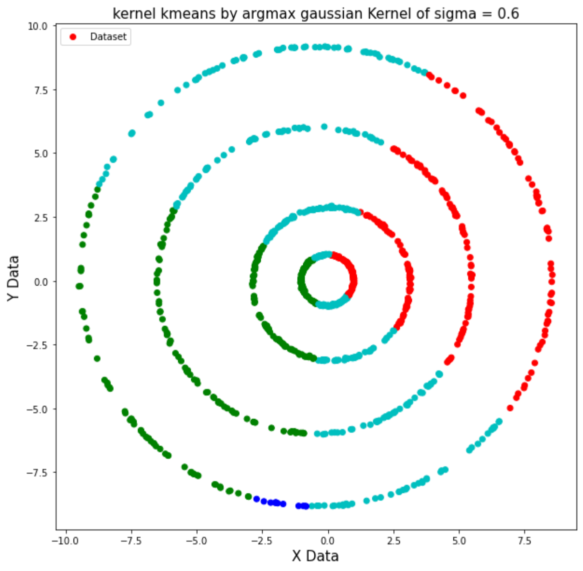


Fig . 45 rbf kernel by argmax of $\sigma = 0.6$

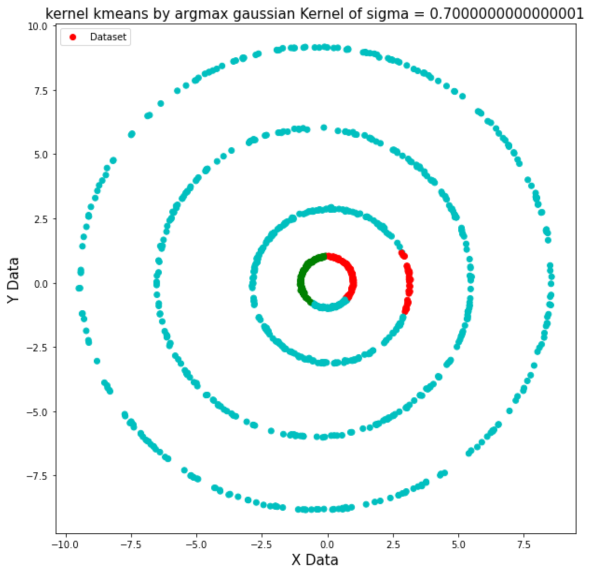


Fig . 46 rbf kernel by argmax of $\sigma = 0.7$

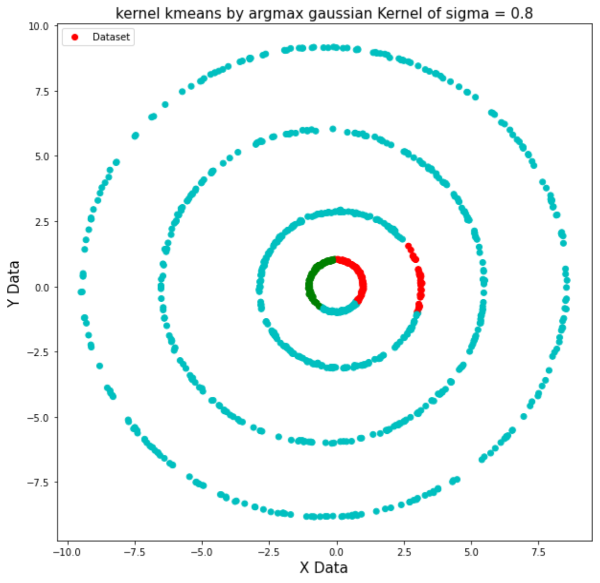


Fig . 47 rbf kernel by argmax of $\sigma = 0.8$

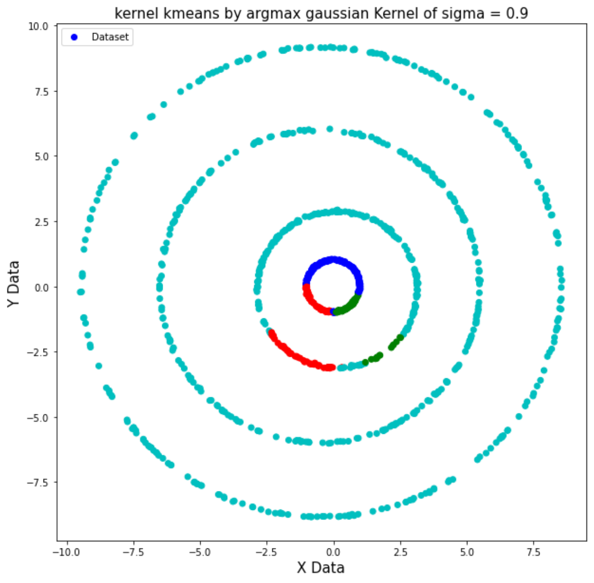


Fig . 48 rbf kernel by argmax of $\sigma = 0.9$

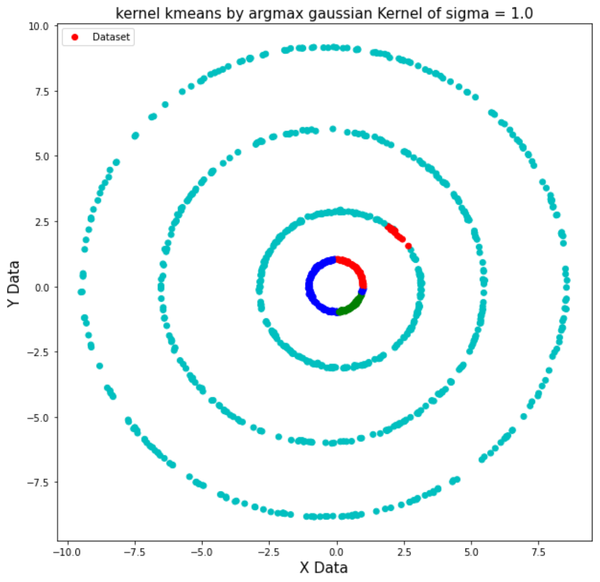


Fig . 49 rbf kernel by argmax of $\sigma = 1$

From taking the maximum assignment of rows of the H matrix as the indicator, we get the above figures, from the figures it can be seen that it doesn't perform that well.

Polynomial kernel is performing worse as compared to rbf as clustering is kind of random and also grouping together is not there. Rbf with $\sigma >= 0.2$ will start to group together clusters, each cluster belonging to one or more sectors and when $\sigma >= 0.7$, clustering is kind of constant, and so outer circles are grouped together and innermost circles of data points are shared by the rest of the clusters. But proper clustering is not obtained (proper in the sense each concentric circles corresponds to each cluster).