Pattern Recognition and Machine Learning

Assignment Report

submitted by

Karthikeya P (CS22B026)

Under the supervision

of

Dr.Arun Rajkumar

Assistant Professor

Computer Science and Engineering

Indian Institute of Technology Madras

Jan - May 2024

Contents

List of Figures

1 Question 1

1.1 Part i

The below image shows all the 784 Principal Components of the training data. The image at the position $[i, j](0-indexed)$ is the $(28i+j)^{th}$ Principal Component. Each row has 28 images, Observe that the images in the 0^{th} row are different from each other showing that they explain large amounts of data in different orthogonal directions. From the 5^{th} row all the images look similar. This tells that some number less that 28*5 = 140 components can explain most of the data.

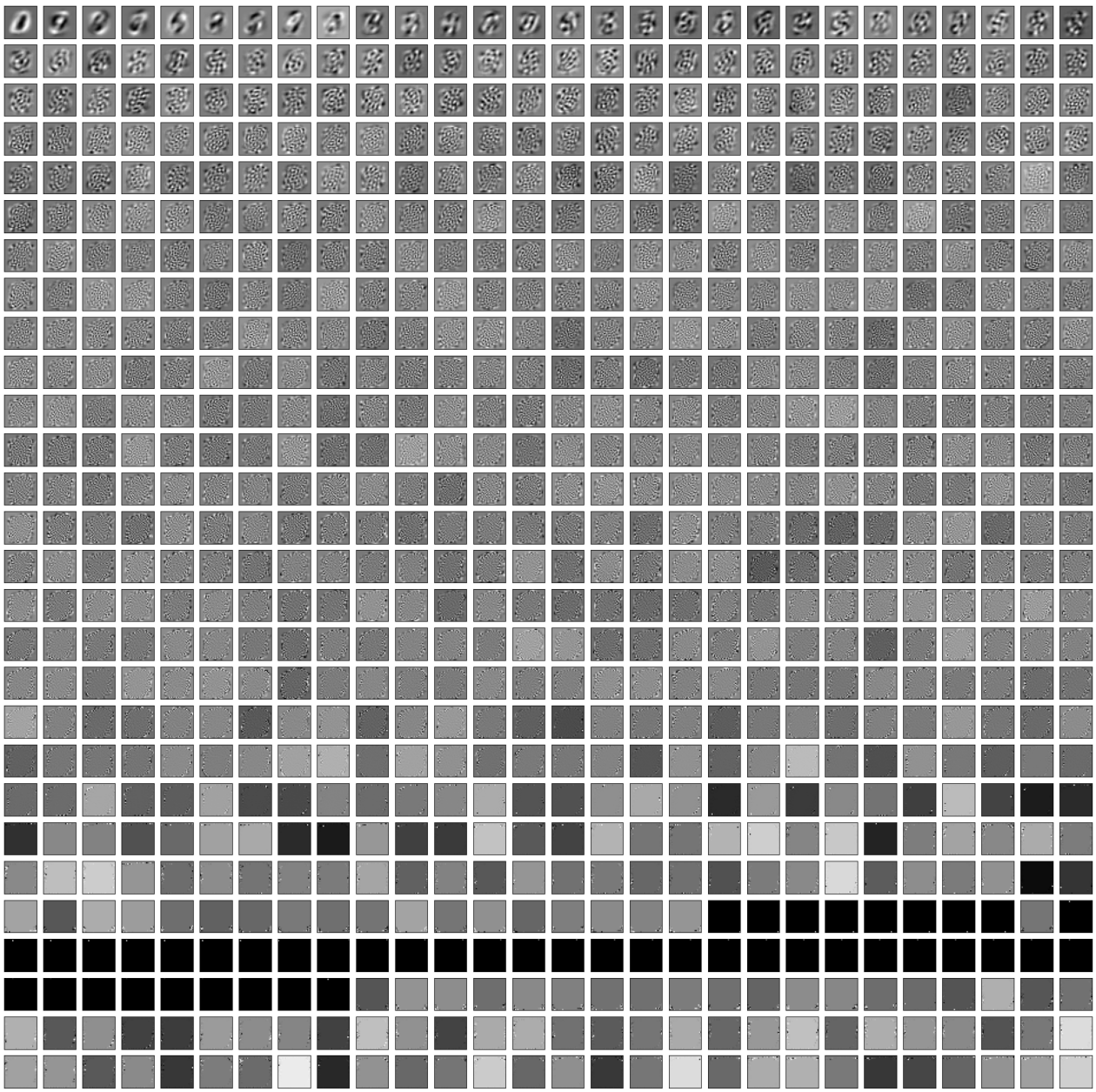


Figure 1: Principal Components

Taking the "mnist" dataset and ran the PCA algorithm on the dataset containing of 100 images of each digit. Here the size of the dataset is $n = 1000$ and each image is 28×28 pixel image, after flattening each image corresponds to a $d = 784$ dimensional vector. The data is stored in a $d \times n$ matrix named as *labelled_training_data* in which each data point corresponds to a image. The data is then centered by subtracting the mean of each feature to obtain the *centered_training_data*.

Compute the $covariance\_matrix = (centered\_training\_data)^T.(centered\_training\_matrix)$. Then find the eigen_values and corresponding eigen_vectors of the covariance_matrix and sort them in descending order of magnitude. Since the covariance_matrix is positive semi-definite all of its eigen values must be positive. If any of the obtained eigen_values are negative. Add a small ϵ to all the diagonal elements in order to avoid any floating point errors.

Each of the eigen_values explains the variance in proportional to its magnitude. Calculate the sum of all eigen values given by $eigen\_sum = \Sigma_i eigen\_values[i]$. Using this we can calculate the percentage variance explained by each of the directions or eigen_vectors which are the Principal Components .Calculating the cummulative percentage of variance explained we observed that 130 of the 784 Principal Components are sufficient to explain 95 percent of the variance.

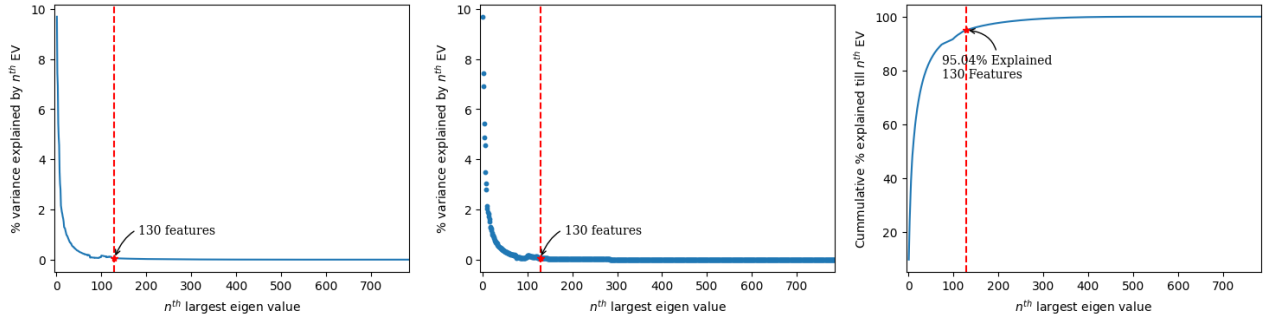


Figure 2: % variance and cummulative % variance explained by n^{th} Principal Components

1.2 Part ii

Use the top eigen_vectors which explain most of the variance of the data. For that calculate the length of the projections along these top Principal Components. To reconstruct the image add the projections along the top Principal Components . The length of projections are stored in a matrix $stored\_data = np.matmul(centered\_training\_data.T, eigen\_vectors)$. The projections of all the images are stored in a matrix projections which is given by

```
projections = []
# The outer for loop is to iterate through all images
# The inner for loop we can adjust j to get only the projections along the
# top Principal Components of interest
for i in range(10*num_images):
    for j in range(784):
        projections.append(np.array([stored_data[i, j]*eigen_vectors[:, j]]))
# List to numpy array
projections = np.array(projections)
```

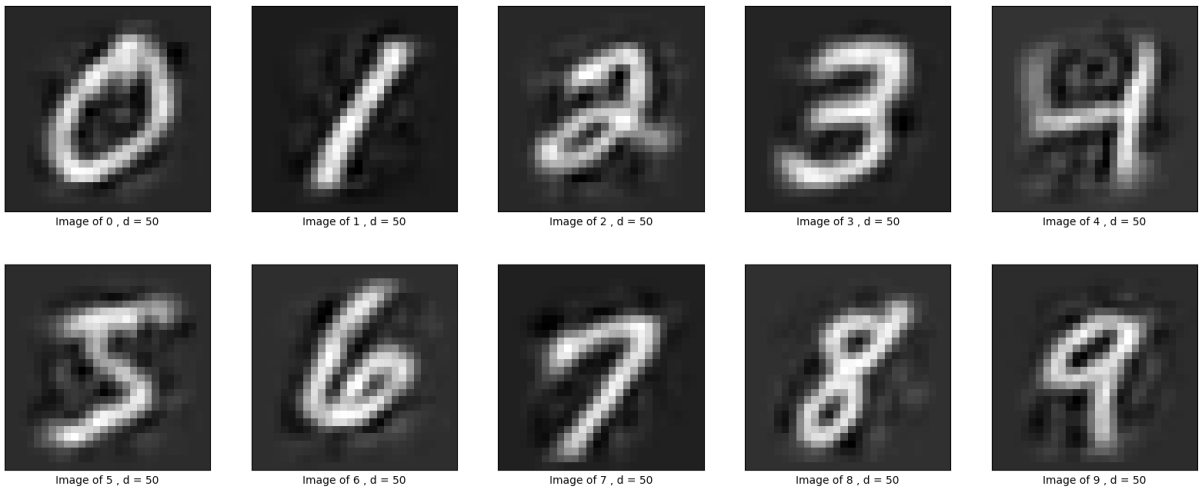


Figure 3: Reconstructed with top 50 Principal Components

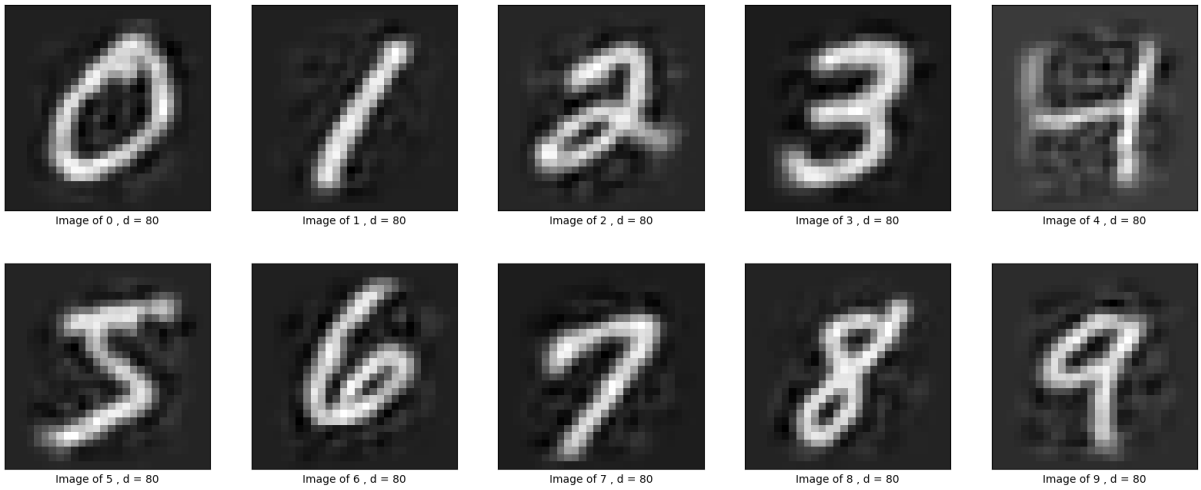


Figure 4: Reconstructed with top 80 Principal Components

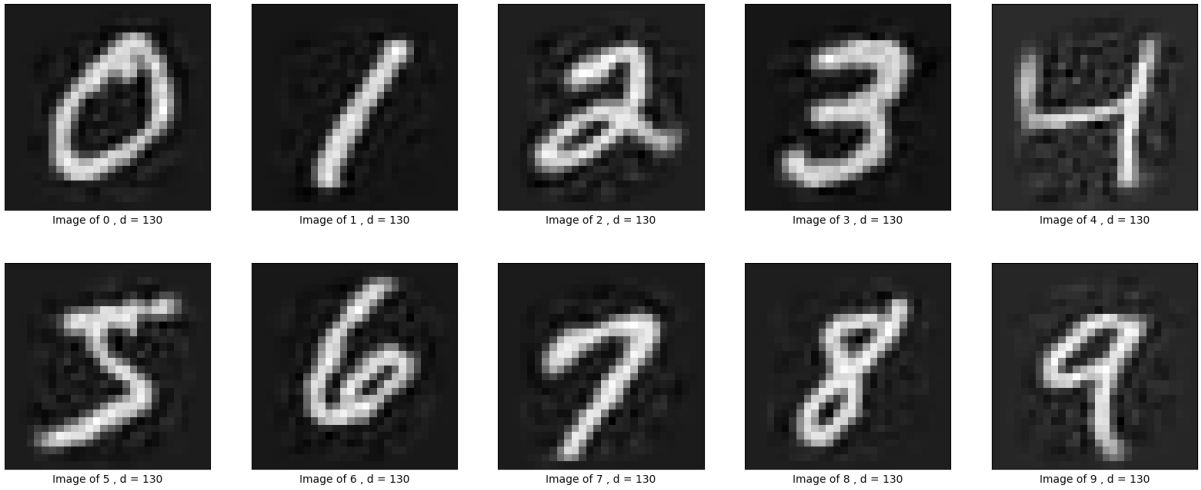


Figure 5: Reconstructed with top 130 Principal Components

Using 130 Principal Components can reconstruct 95% of data, for a downstream task of identifying digits, 90 Principal Components should fairly do the job. Since, using only 50 components one could identify the digits and 130 explains more than 95% of data.

1.3 Part iii - Kernel PCA

1.3.1 Polynominal Kernel

Kernelise the training data with different polynomial kernels $\kappa(x, y) = (1 + x^T y)^d, d = \{2, 3, 4\}$ using the *kernelise* function.

```
def kernelise(labelled_data, d):
    kernelised_data = np.zeros((10*num_images, 10*num_images))
    for i in range(10*num_images):
        for j in range(10*num_images):
            kernelised_data[i, j] = (1+np.dot(labelled_data[:, i],
            labelled_data[:, j]))**d
    return kernelised_data
```

Then center this kernel data using the below *center_kernel* function.

```
def center_kernel(k_d):
    one_n = np.ones((10*num_images, 10*num_images))/(10*num_images)
    centered_kernelised_data = np.array(k_d - np.dot(one_n, k_d)
    - np.dot(k_d, one_n) + np.dot(np.dot(one_n, k_d), one_n))
    return centered_kernelised_data
```

Calculate the *eigen_values* of these kernel matrix and plot the percentage variance explained by these principal components for different polynomial kernels are as follows.

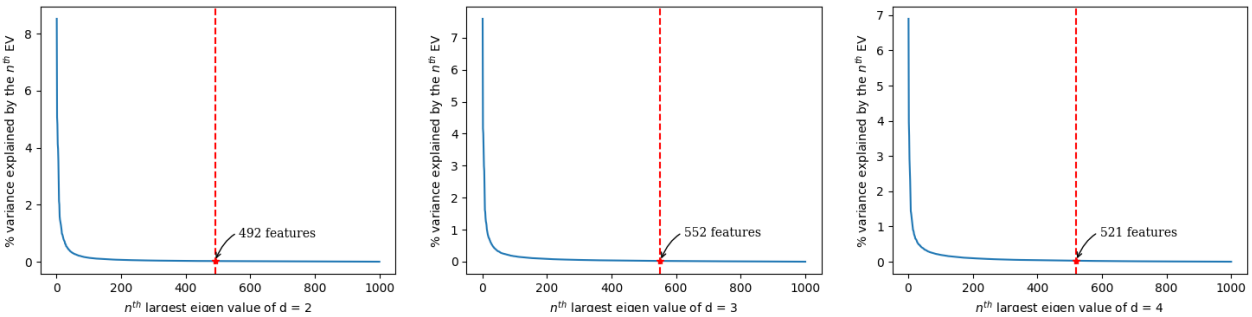


Figure 6: % variance explained by n^{th} Principal Components of different Polynomial Kernels

Then calculate the values of $\phi(x)^T \omega_i$ using the kernel and then plotting these values of each points of ω_1 against ω_2 for different Polynomial Kernels.

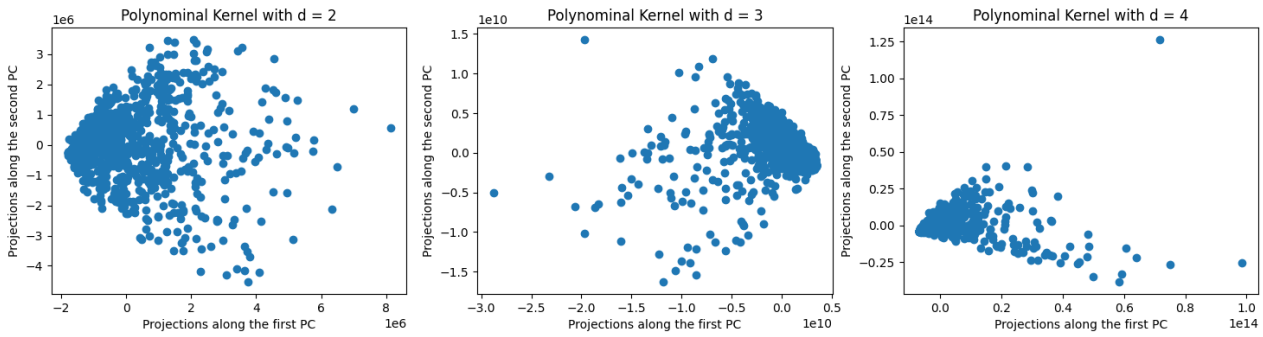


Figure 7: ω_1 vs ω_2 projections for differnet Polynomial Kernels

1.3.2 Radial Kernel

Kernelise the training data with radial kernel $\kappa(x, y) = exp\frac{-(x-y)^T(x-y)}{2\sigma^2}$ using different values of σ using the *kernelise* function.

```
def kernelise(labelled_data, d):
'''The kernelise function kernelises the array sent to it'''
kernelised_data = np.zeros((10*num_images, 10*num_images))
for i in range(10*num_images):
    for j in range(10*num_images):
        kernelised_data[i, j] = np.exp((-(np.dot(labelled_data[:, i] -
        labelled_data[:, j], labelled_data[:, i] -labelled_data[:,j]))/(2*(d**2))))
return kernelised_data
```

Then center this kernel data using the below *center_kernel* function.

```
def center_kernel(k_d):
    one_n = np.ones((10*num_images, 10*num_images))/(10*num_images)
    centered_kernelised_data = np.array(k_d - np.dot(one_n, k_d)
    - np.dot(k_d, one_n) + np.dot(np.dot(one_n, k_d), one_n))
    return centered_kernelised_data
```

Calculate the *eigen_values* of these kernel matrix and plot the percentage variance explained by these principal components for different values of σ are as follows.

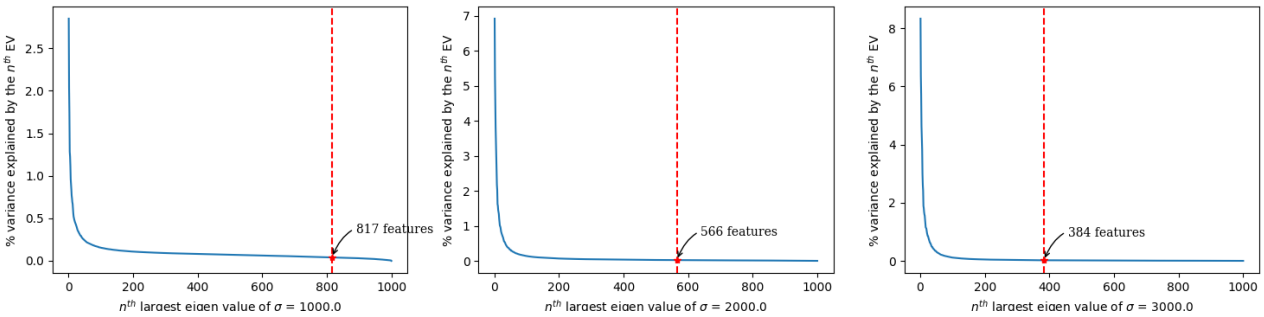


Figure 8: % variance explained by n^{th} Principal Components of Radial Kernels for different σ

Then calculate the values of $\phi(x)^T\omega_i$ using the kernel and then plotting these values of each points of ω_1 against ω_2 for radial kernels for different values of σ. The below plots are for $\sigma = \{1000, 2000, 3000\}$

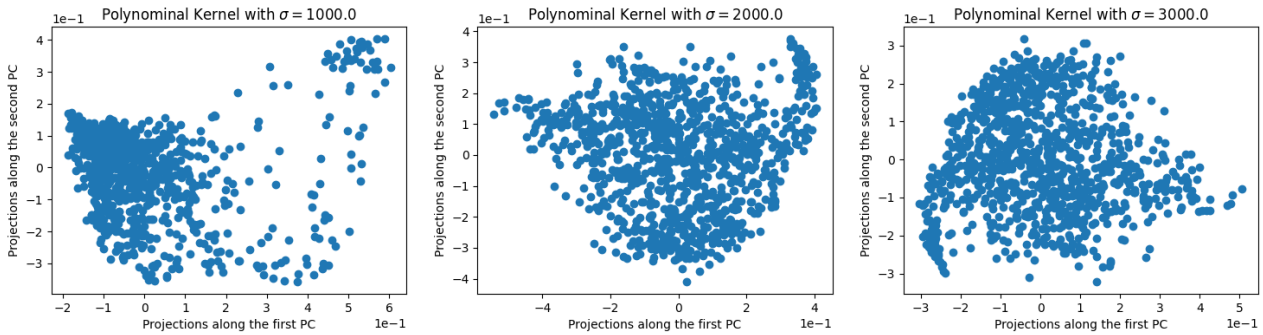


Figure 9: ω_1 vs ω_2 projections for Radial Kernels with different σ

1.4 Part iv

For this dataset I feel not kernelising would be a better choice. The reasons for this are

- For data which is not kernelised we needed 130 Principal Components to explain 95% variance.
- For data which is kernelised using the polynomial kernel we needed $492, 552, 521$ Principal Components for power $= 2, 3, 4$ respectively.(refer to 6)
- For data which is kernelised using the radial kernel we needed $817, 566, 384$ Principal Components for $\sigma = 1000, 2000, 3000$.

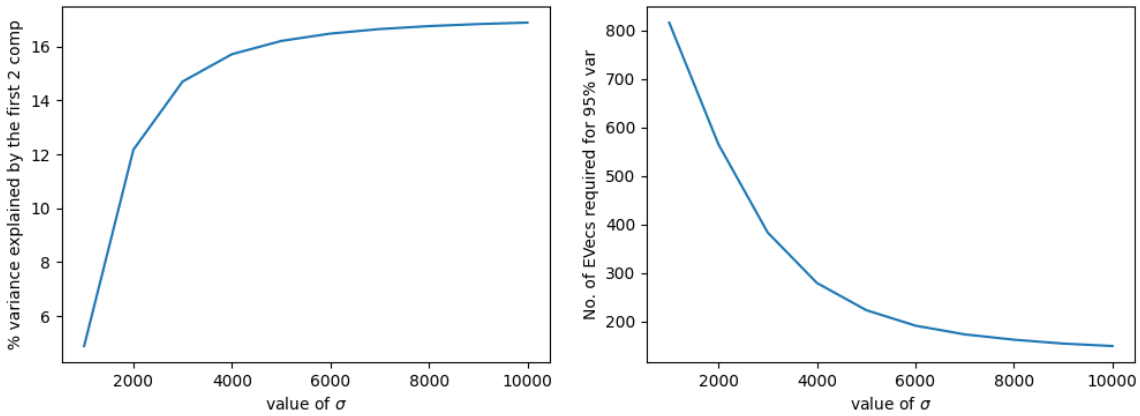


Figure 10: % Var explained and # of Components used for 95% variance

- From the above plots, one can see that as the value of σ increases, number of components used to explain 95% of the variance is decreasing.
- But that doesn't mean that we can increase to any large value cause then the kernel will make all the elements same.
- So if we want to use radial kernel we need to use some optimal σ which doesn't make all of them same after kernelisation and at the same time decreases number of principal components used.
- 3000-4000 is one such range and among the given kernels radial_kernel with $\sigma = 3000 - 4000$ is better compared to others as it uses only 384 or less components and doesn't affect the importance of data much compared to 492 components used for polynomial kernel.
- But if I need to choose, I prefer to use the identity kernel as it is using only 130 Principal Components where as other kernels are using more principal components.

2 Question 2

2.1 Part i

Taking the "cm_dataset.csv" into the dataset. First separate the x and y values and plot these values on xy-plane.

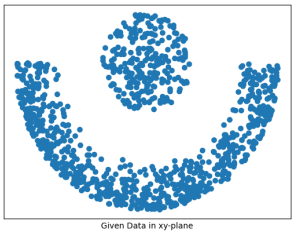


Figure 11: cm_dataset plot

The goal is to get 2 clusters from the data. Applying K-means directly for 5 random initializations taking $k = 2$. The below figure shows the plots before and after K-means and how the error is changing with the iterations for 20 iterations.

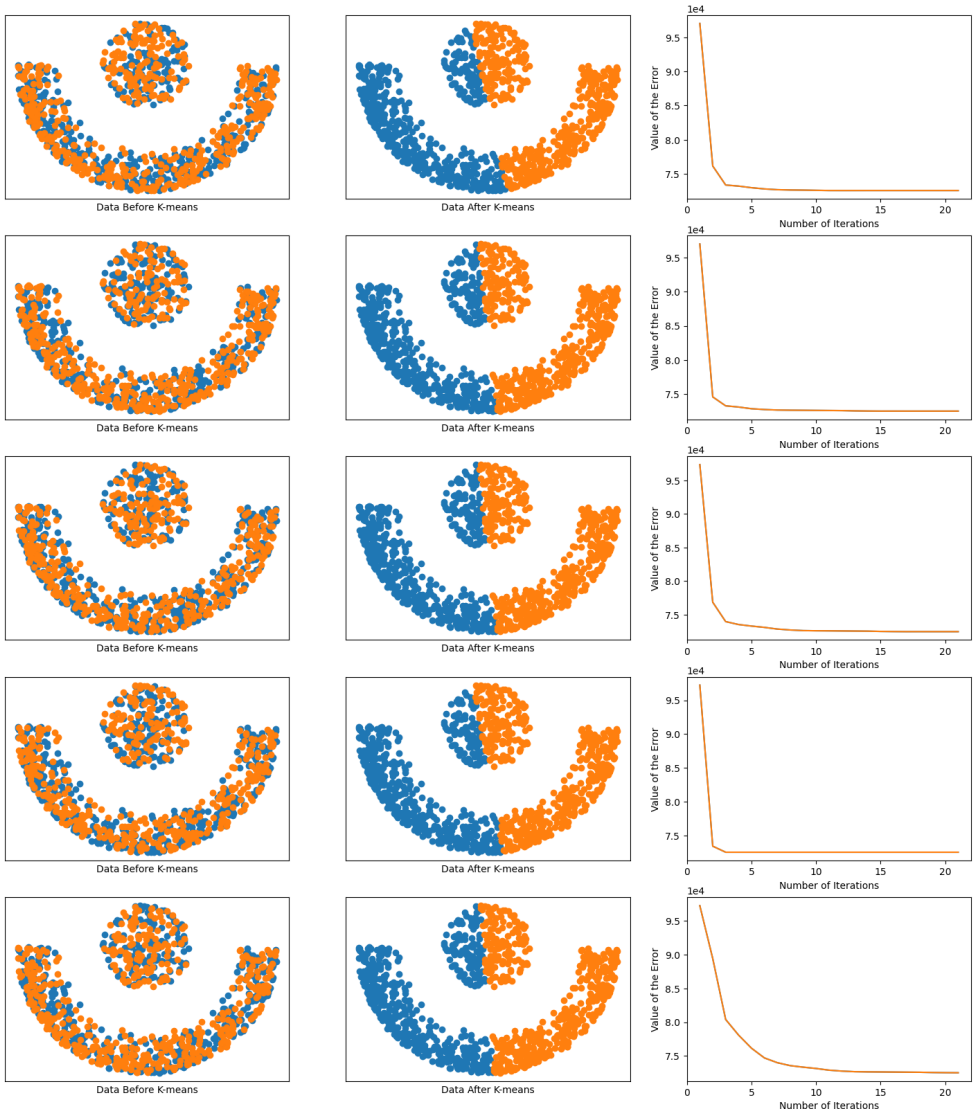


Figure 12: Data Before and After K-means and Error Values

It is interesting to look how the position of Means of each Voronoi Region moves with iterations.

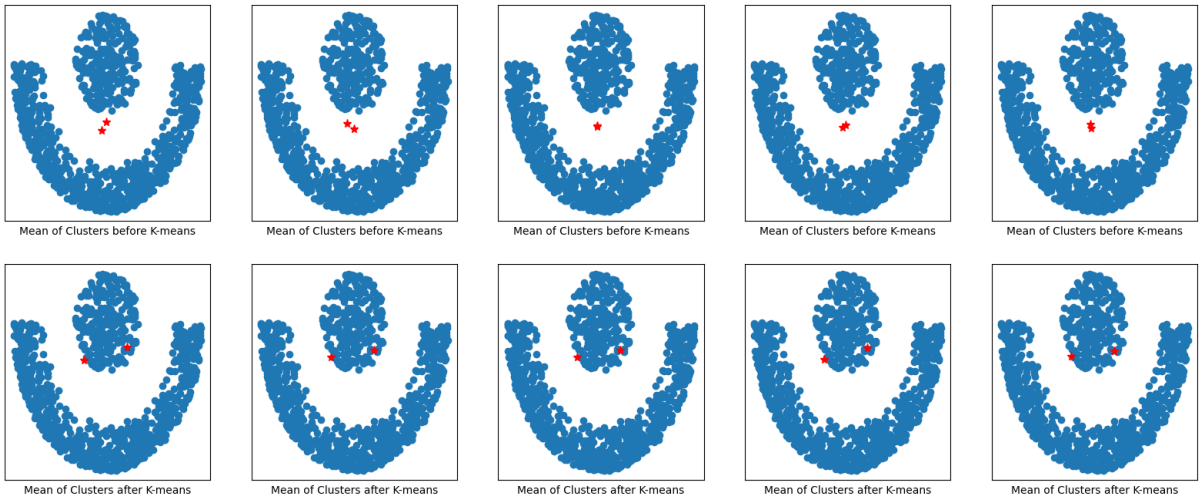


Figure 13: Initial and Final Positions of Means for different Initialisations

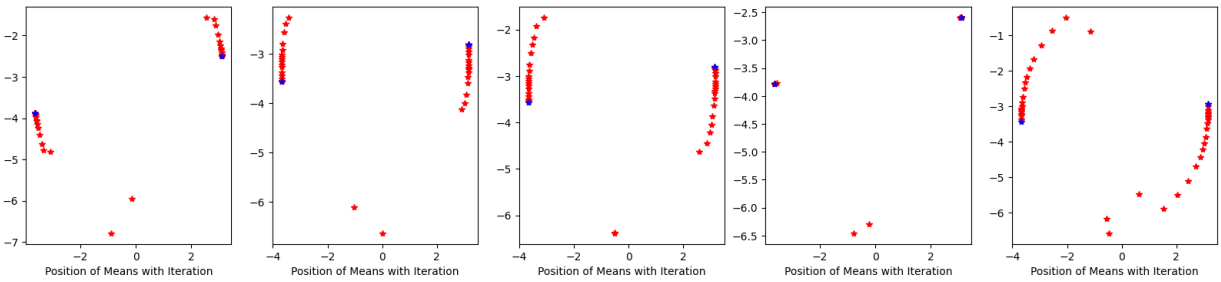


Figure 14: Movement of Means with Iterations for different Initialisations

On careful observation we can see that the above 3 images are closely related. Few observations are given below.

- Observe the value of the error values (Fig 12) after 20 iterations, all these are approaching the same value close to 73000. Now observe the position of final means (Fig 13) they are all nearly at the same positions.
- The dip in the error value (Fig 12) after the first iteration is huge but after that, the error is changing slowly. This can also be observed in Figure 14 there are 2 stars far from remaining stars, they are the initial means, this separation indicates huge shift in the position of means leading to huge decrease in error value.
- After that as the shift is not that drastic the error function is also not showing drastic dips. Observe that the stars are becoming dense towards the blue stars (final position), indicating less movement.
- The error value (Fig 12) in 4^{th} random initialisation is almost same from the 3^{rd} iteration indicating not much change in position of means which can be seen in Fig 14 also.
- The dip in error value (Fig 12) in the 5^{th} initialisation after 1^{st} iteration is less compared to other initialisations. This can also be seen in Fig 14, the shift of one of the initial 2-means is less compared to other initialisations.

2.2 Part ii

The goal is to see how do Vornoi Regions look for different values of k in the K-means algorithm. Here the plots shown below are for $k = 2, 3, 4, 5$ with random initialisations.

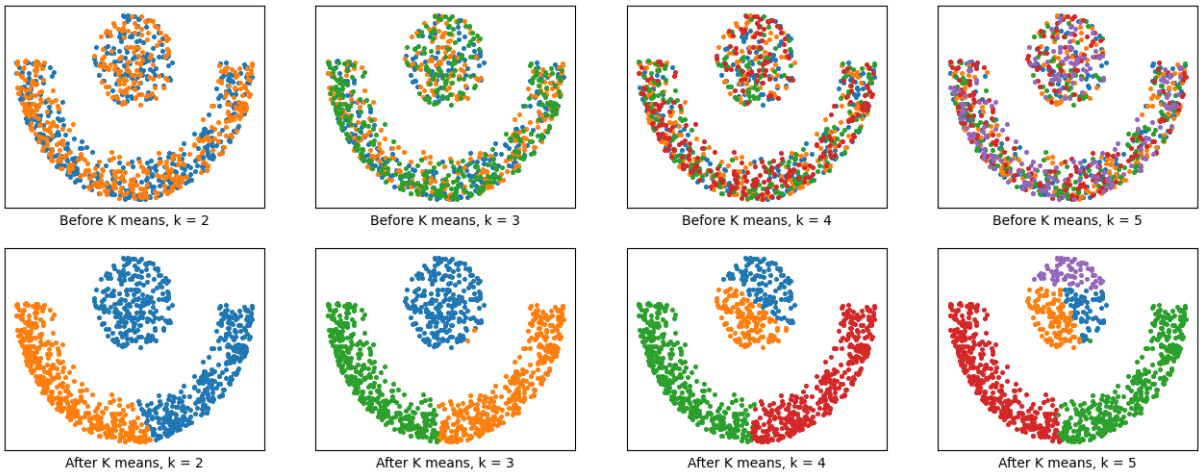


Figure 15: Voronoi Regions for $k = 2, 3, 4, 5$

An important thing to keep in mind is that even though few of the Clusters appear to have non-linear boundaries, this doesn't mean that the Voronoi Regions also have non-linear boundaries. That is because of the kind data we are having.

Similar plots as earlier to see the movement of means at each iteration are generated.

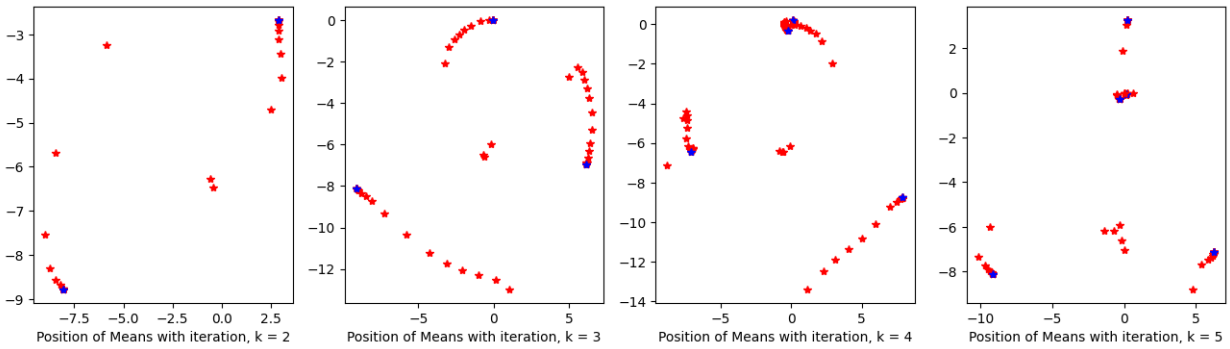


Figure 16: Movement of Means for $k = 2, 3, 4, 5$ with iterations

K-means algorithm clusters the data into Voronoi-Regions , that is the reason why the clustering is not as expected. The Voronoi-Regions are separated by the perpendicular bisector of the final means which are straight lines, so that clusters with curvilinear boundaries can't be clustered directly. To resolve this, take the data into some higher dimension using a Kernel and apply K-means. This will cluster the given data into required clusters.

2.3 Part iii

To kernelise the given data and apply Lloyd's Algorithm in higher dimension I tried different kernels with k = 2 with and without normalisation. The below image is for radial kernel with $\sigma = 3.5$

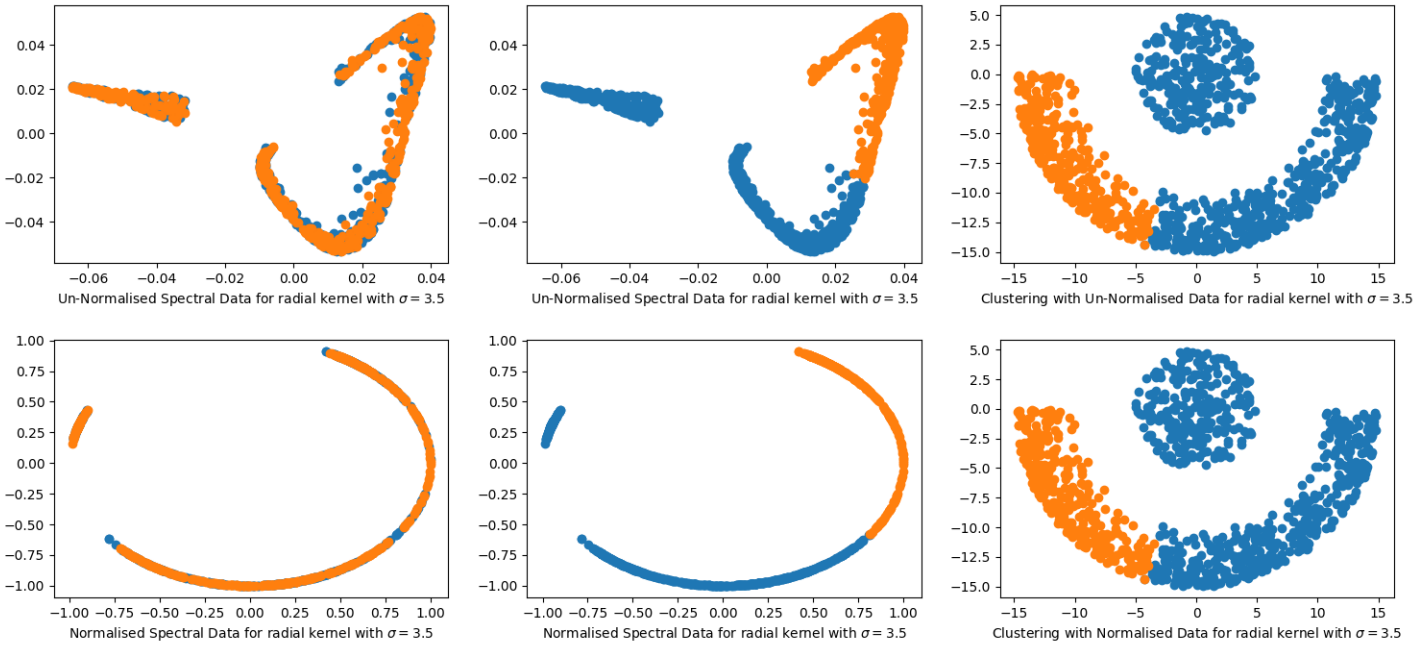


Figure 17: Spectral Clustering with Radial Kernel $\sigma = 3.5$

The below image is for polynomial kernel with $d = 2$

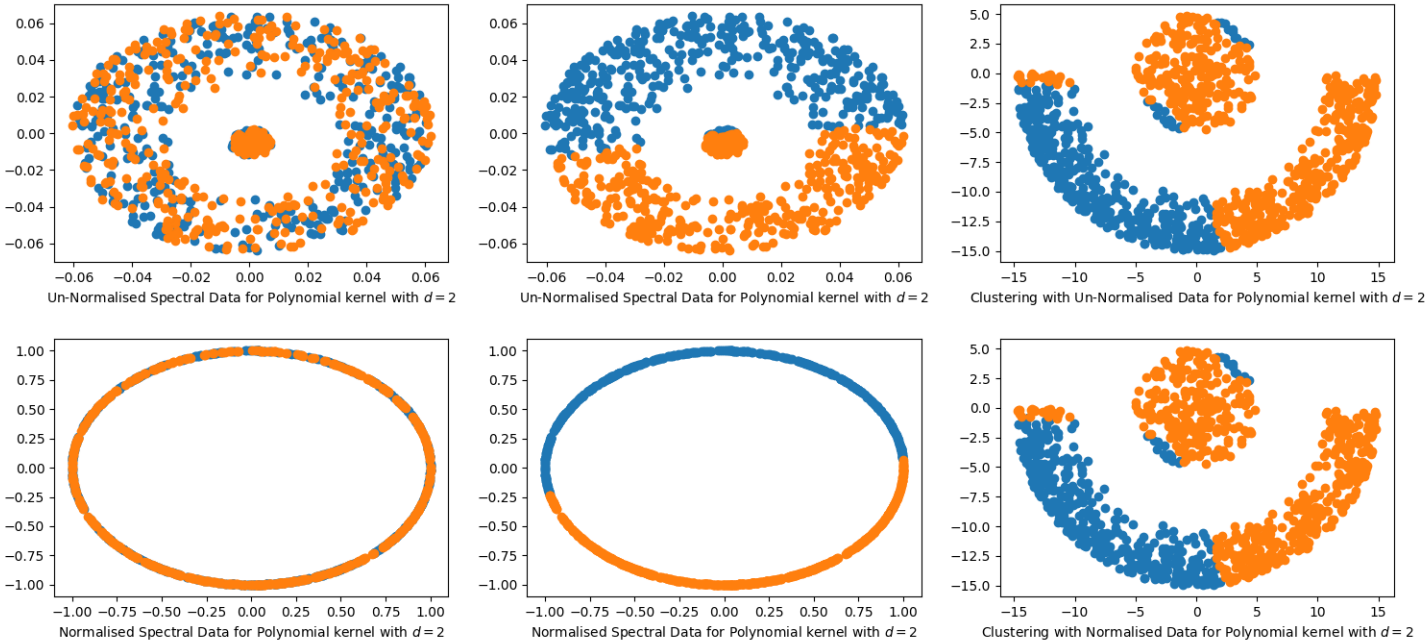


Figure 18: Spectral Clustering with Polynomial Kernel $d = 2$

The below image is for polynomial kernel with $d = 3$

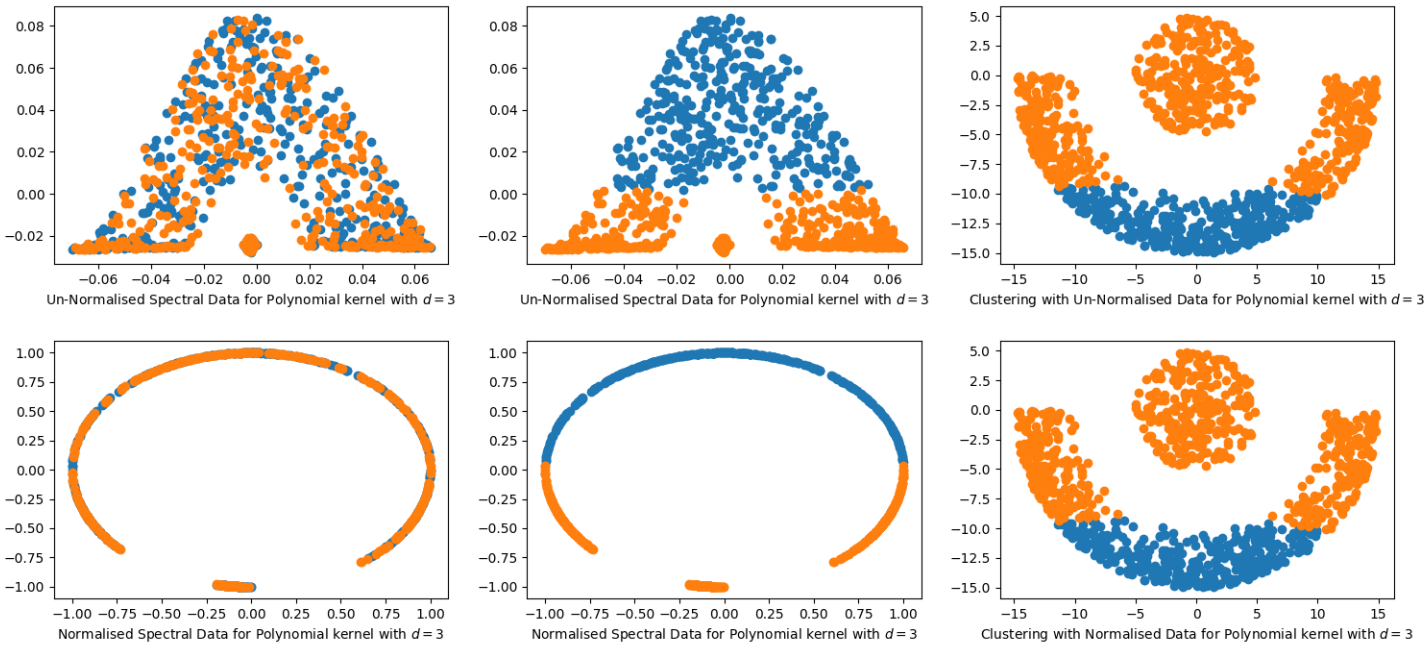
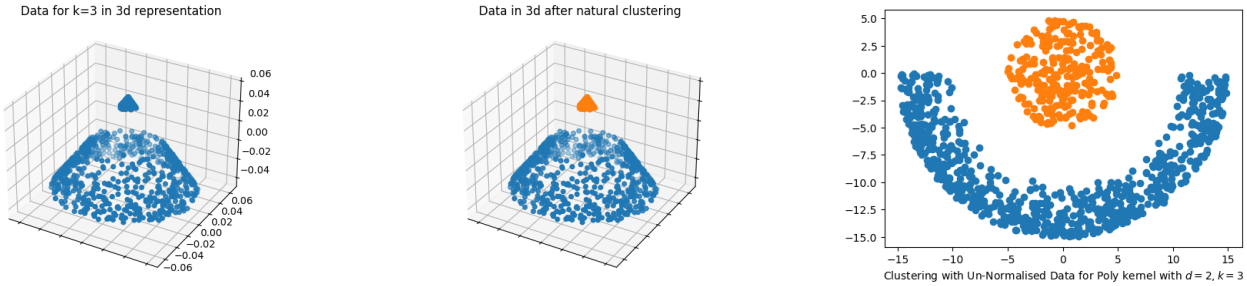


Figure 19: Spectral Clustering with Polynomial Kernel $d = 3$

Few observations that I made after trying out different kernels for spectral clustering are:

- The final clusters are largely dependent on the initialisation.
- If we have a good initialisation we can get proper clustering in radial kernel for the values of σ ranging between 3 and 4.
- The above is because if we observe (Fig 17) there is a possibility of Voronoi-Regions.
- Such voronoi regions are not possible for polynomial kernels for whatever the values of the power with $k = 2$.
- But, on taking k = 3 and polynomial kernel with $d = 2$. We can clearly see a natural voronoi seperation and with trivial initialisation. The same is shown below.
- Concluding that for $k = 2$ radial kernel with $\sigma = 3.5$ is best choice given that the initialisation is a good one. Else with flexible k polynomial kernel with d = 2 and natural initialisation is best suitable for this dataset.



(a) Kernel Data in 3d (b) Clustered Data in 3d (c) Original Data Clusters

Figure 20: Clustering of data using Polynomial Kernel d=2 and 3 eigen_vectors

2.4 Part iv

Instead of using the method suggested by spectral clustering to map eigenvectors to cluster assignments, assign data point i to cluster l whenever.

$$l = \text{argmax } v_i^j, j = 1, 2, \ldots, k$$

where $v^j \in \mathbb{R}^n$ is the eigenvector of the Kernel Matrix associated with the j^{th} largest eigen value.

2.4.1 Polynomial Kernel

Plotting the clusters according to the above method for different Polynomial Kernels with exponents = [2, 3, 4, 5, 8, 10, 15, 25, 50]

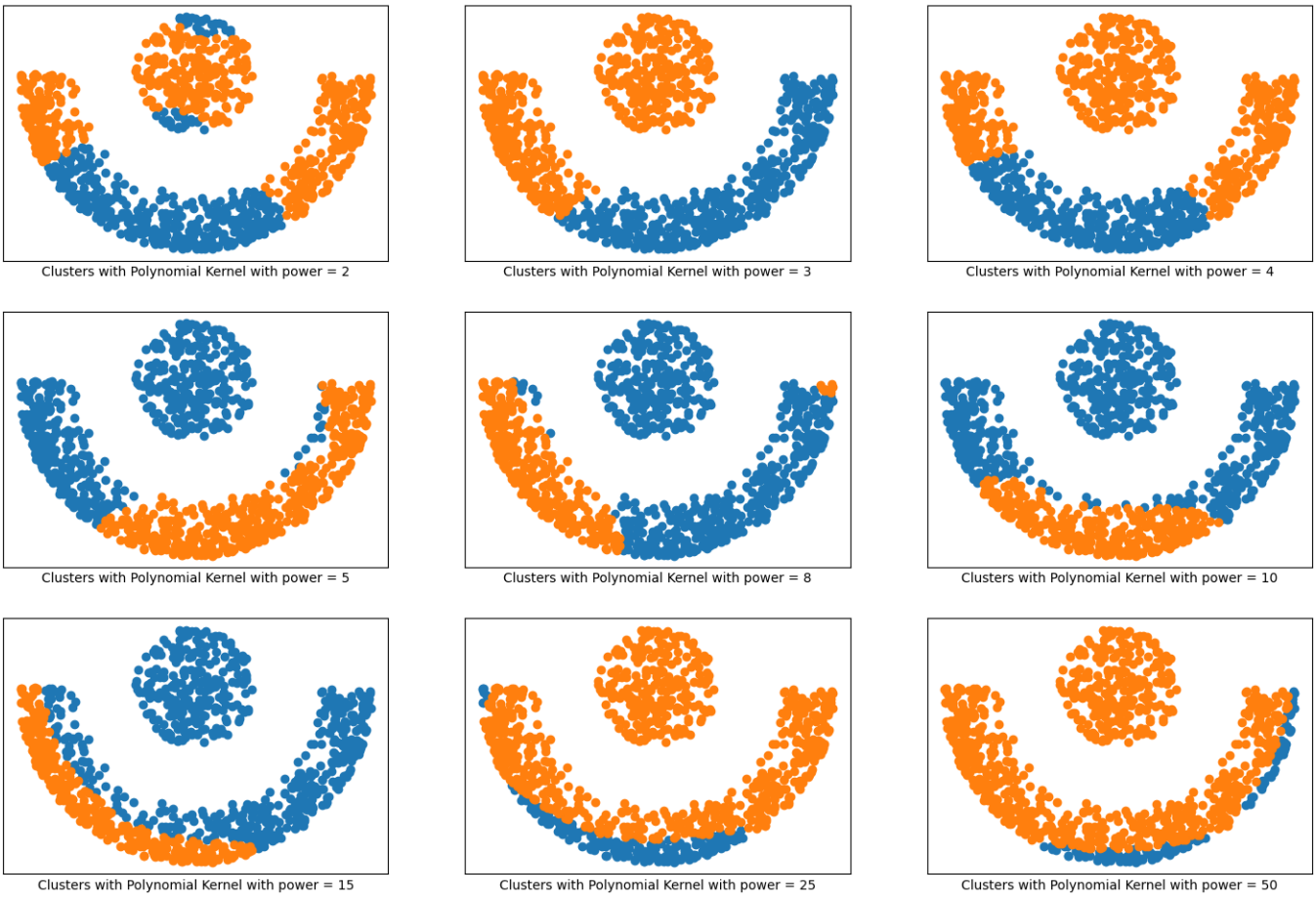


Figure 21: Polynomial Kernel Clustering for different Powers

Few observations can be derived from the above clusterings.

- For higher powers one of the two components is dominating for all the data points.
- So polynomials with higher powers can't be used to cluster this data using this method.
- For lower powers there is no proper clustering happening with this method.
- This method is not suitable for clustering for any polynomial kernel for this data.

2.4.2 Radial Kernel

Plotting the clusters according to the above method for different Radial Kernels with

$$\sigma = [0.1, 1, 10, 10^2, 10^3, 10^4]$$

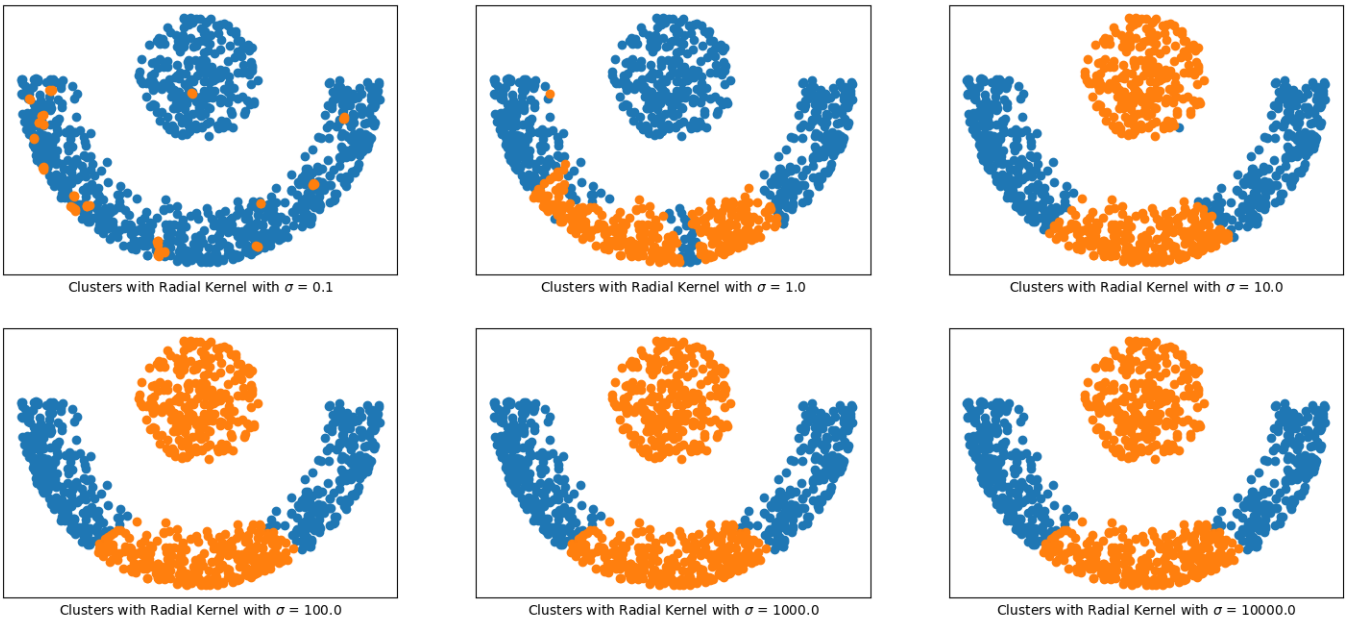


Figure 22: Radial Kernel Clustering for different σ

Few observations that can be derived from the above clusterings.

- For higher values of σ the clusters are the same because in the radial kernel the denominator is dominating and making all the values close to 0 for any higher values of σ not much difference can be observed.
- For very smaller values of σ all the points are getting assigned to the same cluster(can see the top left), not giving proper cluster.
- For moderate values of σ clustering is happening but not a proper one.
- So this method is not suitable for this data even for radial kernel whatever the values of σ be.