CS5691: Pattern Recognition and Machine Learning

Assignment # 1

Pratham Sunkad, ME21B145

March 10, 2024

Contents

List of Figures

1 Principal Component Analysis on MNIST

1.1 Overview of the functions defined

(1) `def create_data` : Creates the dataset containing 100 images of each class (0-9). The shape of the dataset returned is $d \times n$, where $d = 784$ and $n = 1000$.

(2) `def bubble_sort` : It is used to sort the eigenvalues and eigenvectors obtained from np.linalg.eigh(covariance_matrix) in descending order of eigenvalues.

(3) `def cal_eigen` : Calculates eigenvalues and eigenvectors of the Covariance Matrix and returns these along with the mean of the dataset.

(4) `def reconstruct_data` : Reconstructs data using top d eigenvectors by linearly combining the projections along these eigenvectors.

(5) `def polynomial_kernel` : Takes in input the dataset and p to find the Kernel Matrix K using the polynomial kernel. The K matrix is centered using the centering relationship in the higher dimension.

(6) `def radial_kernel` : Takes in input the dataset and σ to find the Kernel Matrix K using the radial kernel. The K matrix is centered using the centering relationship in the higher dimension.

(7) `def cal_eigen_K` : Calculates eigenvalues and eigenvectors of the K matrix and returns only this.

(8) `def cal_projection` : Calculates α_k using eigenvectors and eigenvalues of the K matrix. Using this it finds the projection of the data-points on the top d ω_k's and return this.

1.2 Q1 subpart (i)

Pseudo Code:

(1) Create the dataset

(2) Calculate the covariance matrix of the dataset

(3) Find the eigenvalues and eigenvectors of this matrix

(4) Sort the eigenvectors and display them

Observations Regarding Eigenvectors:

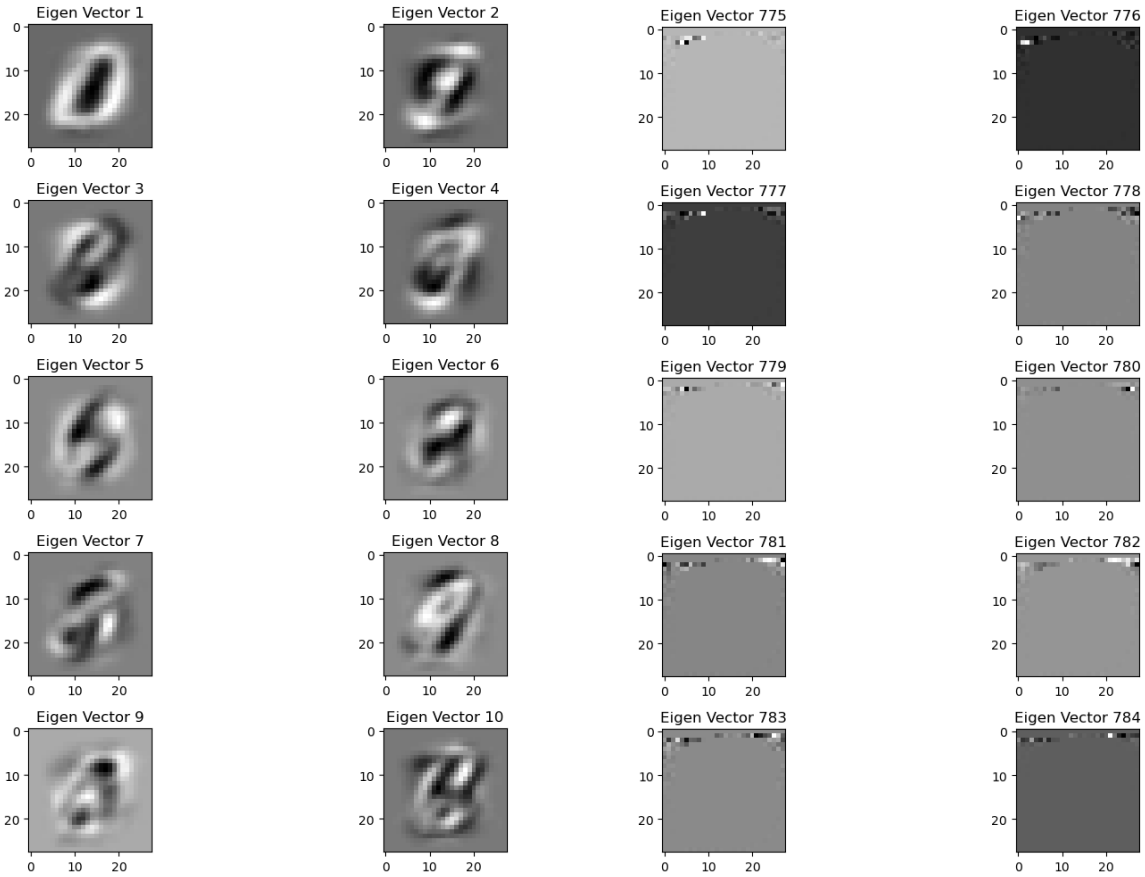
(1) The top 10 eigenvectors display strong features corresponding to the dataset

(2) Variance explained by the 1st eigenvector (corresponding to the highest eigenvalue) is maximum and this decreases with other eigenvectors

(3) The last 10 eigenvectors do not have any structure in them, thus contributing very little to explaining the variance of the dataset.

(4) Variance explained by 1st eigenvector = 9.68%

(5) Variance explained by 2nd eigenvector = 7.43%



(a) Top 10 eigenvectors of the dataset (b) Last 10 eigenvectors of the dataset

Figure 1: Eigenvectors of the dataset

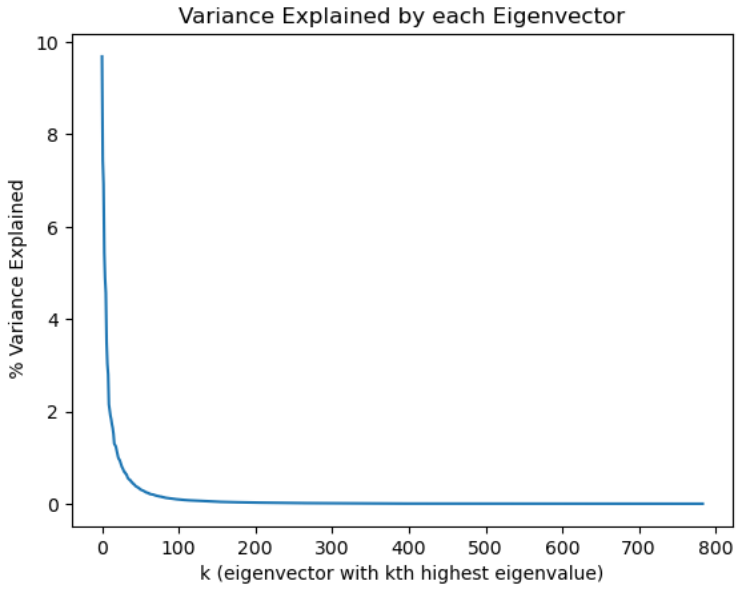


Figure 2: Variance explained by each eigenvector

1.3 Q1 subpart (ii)

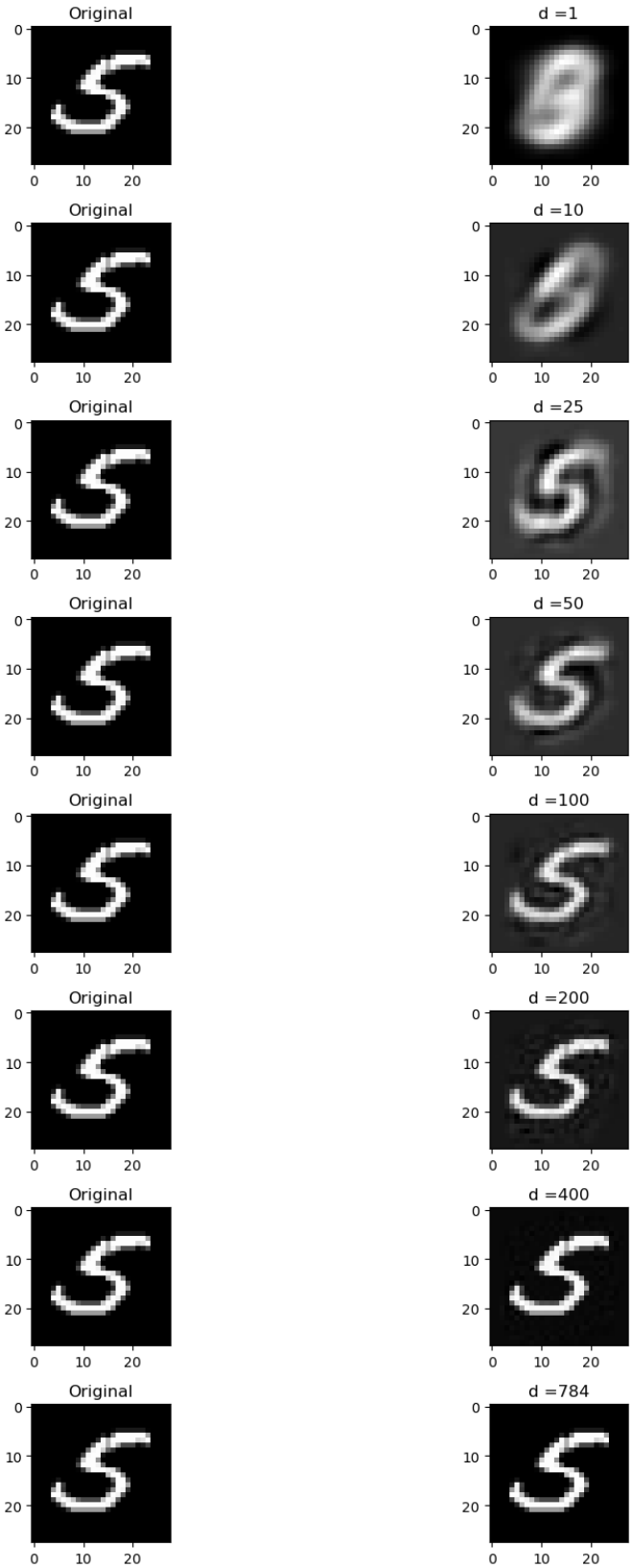


Figure 3: Reconstruction using different number of eigenvectors(d)

Observations:

(1) After calculating the eigenvectors and eigenvalues, we use `def reconstruct_data` to reconstruct the data with different number of eigenvectors

(2) The clarity of the images increases as d increases

(3) For $d = 1$, 10, 25 & 50 the reconstructed digits are still blurry and cannot be recognized

(3) Eigenvectors which can express 95% variance of the dataset is needed for downstream tasks

(4) $d = 113$ expresses 95% variance of the dataset, thus suitable for compression

(5) Using more eigenvectors than this can prove to be redundant as 95% variance is more than enough to capture important features of the dataset

1.4 Q1 subpart (iii)

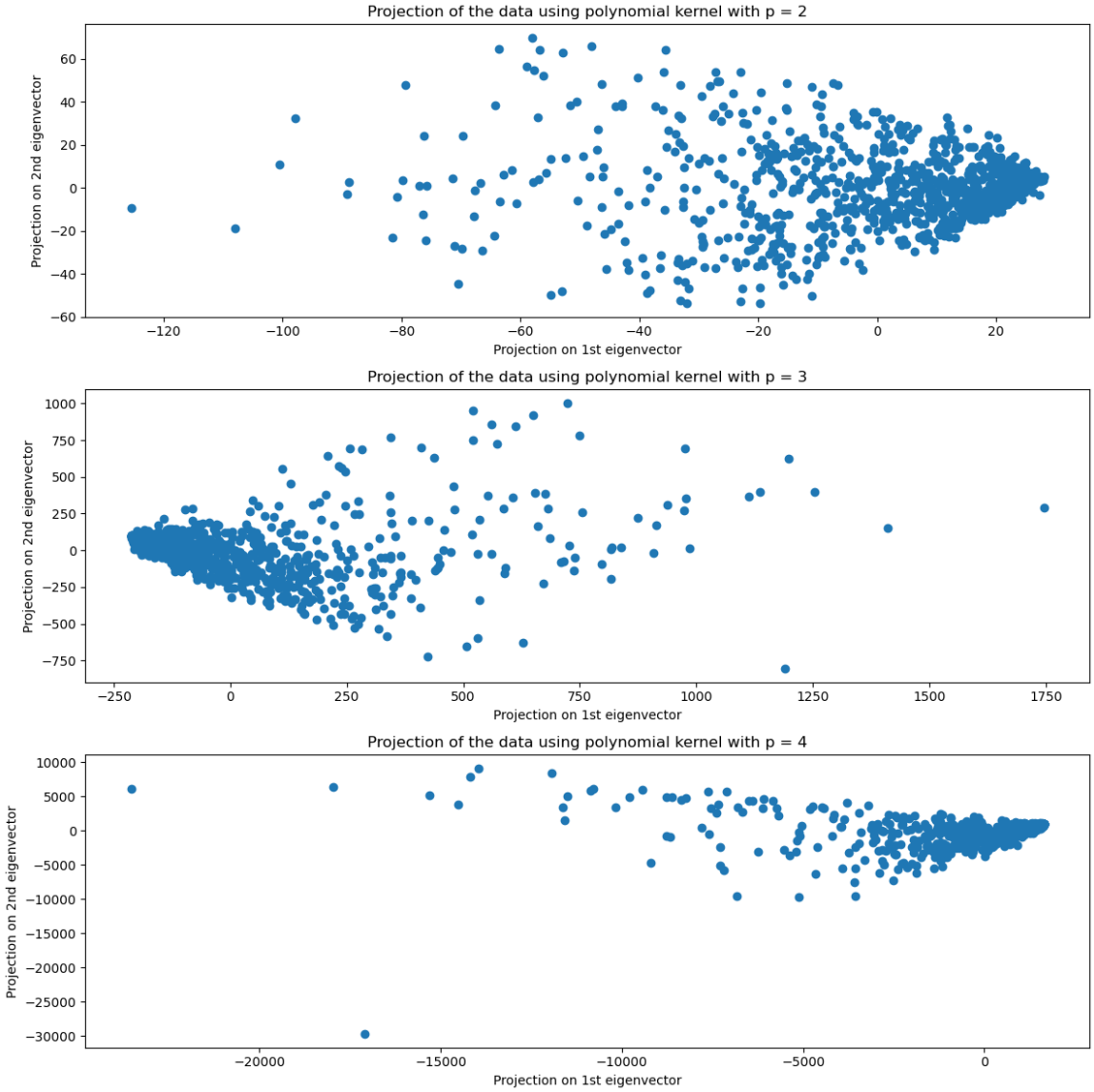


Figure 4: Projection Plots for d = 2,3,4 with polynomial kernel

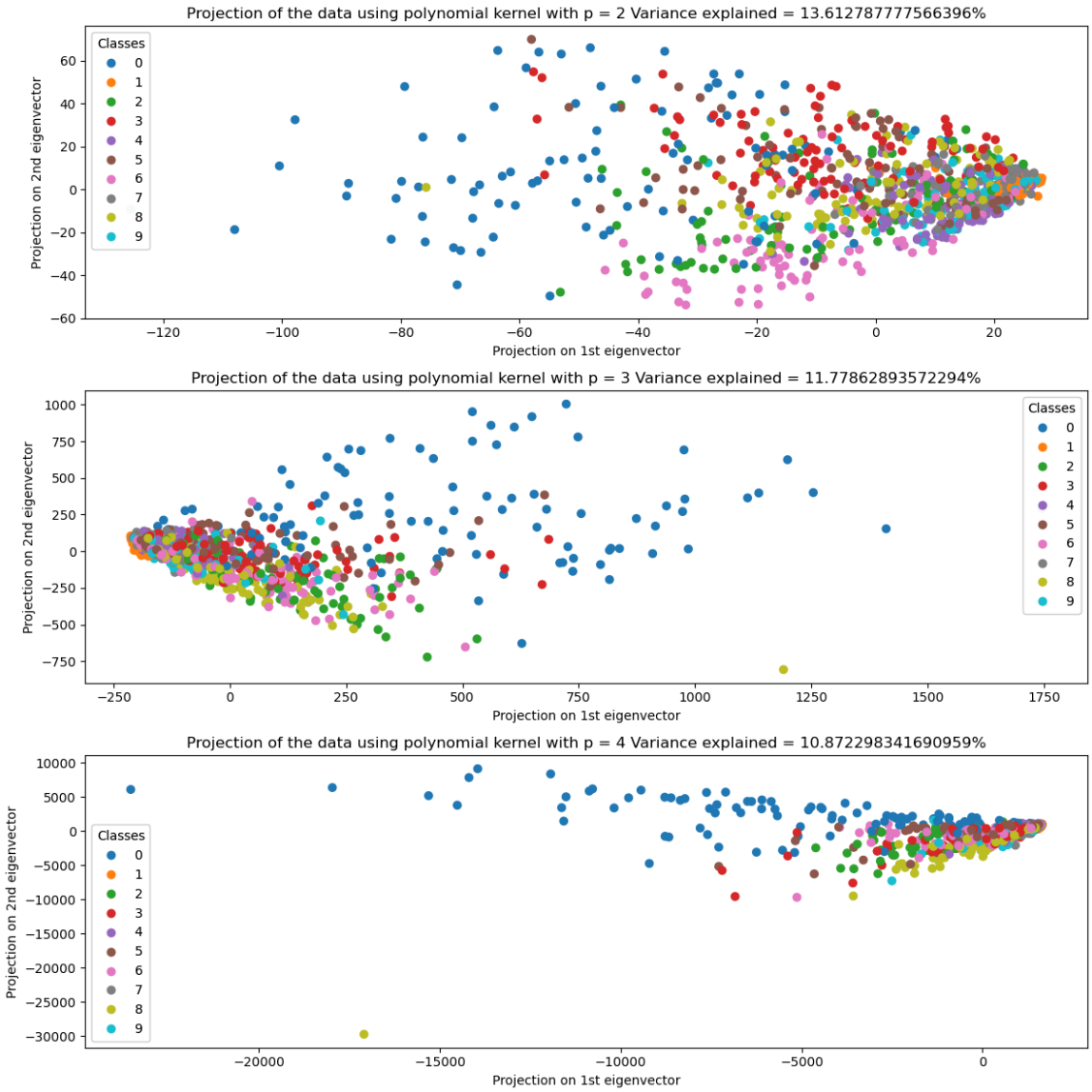


Figure 5: Projection Plots class wise for d = 2,3,4 polynomial kernel

Observations:

(1) The 2D scatter of the projections tries to capture the underlying structure in the data explained by top 2 eigenvectors.

(2) $d = 2$ helps in capturing the best relationship as the scatter of points expresses the highest variance.

(3) Lesser the overlap between projections of different classes, better explained is the data by the eigenvectors

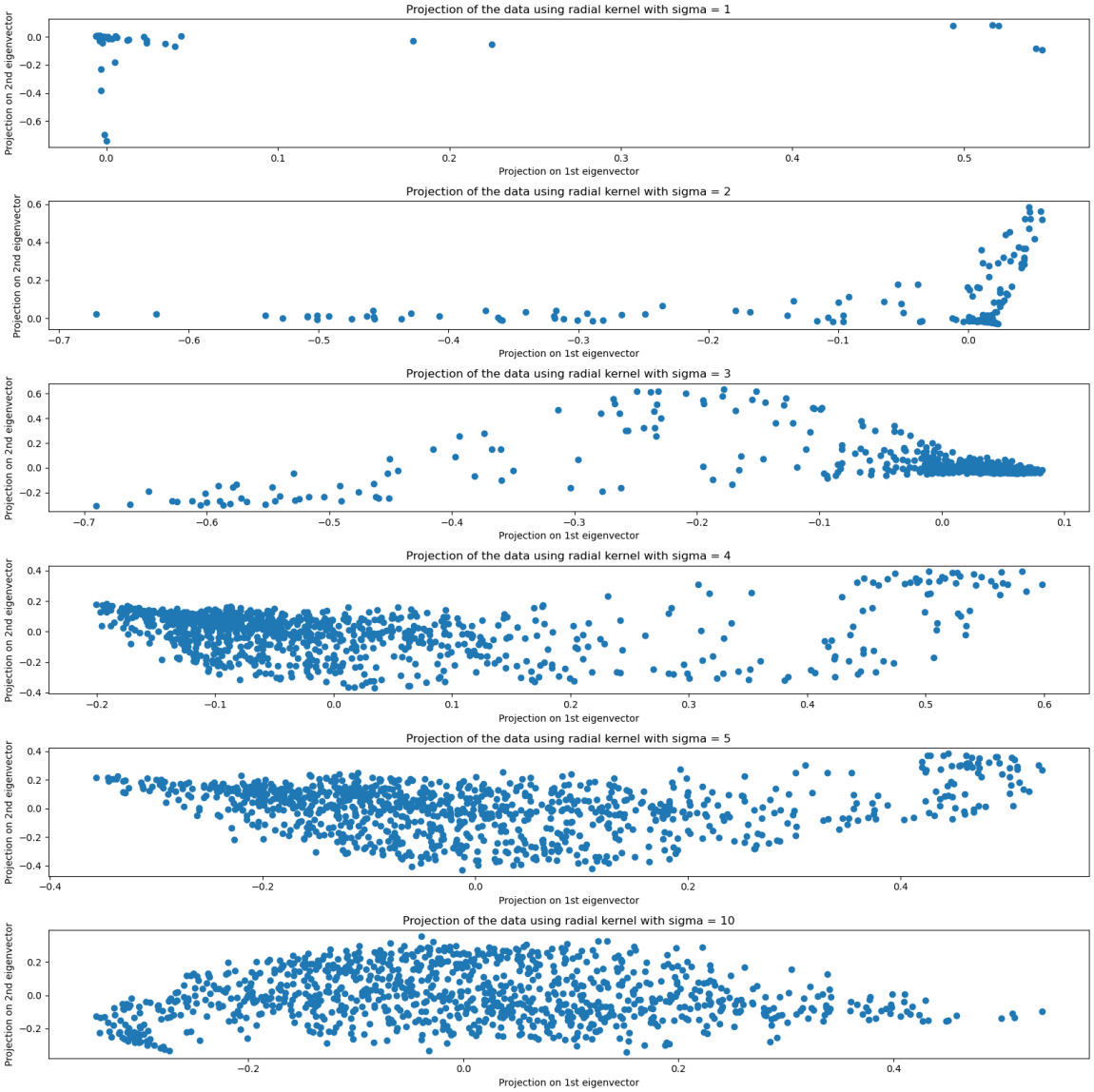


Figure 6: Projection Plots for sigma = 1,2,3,4,5,10 with radial kernel

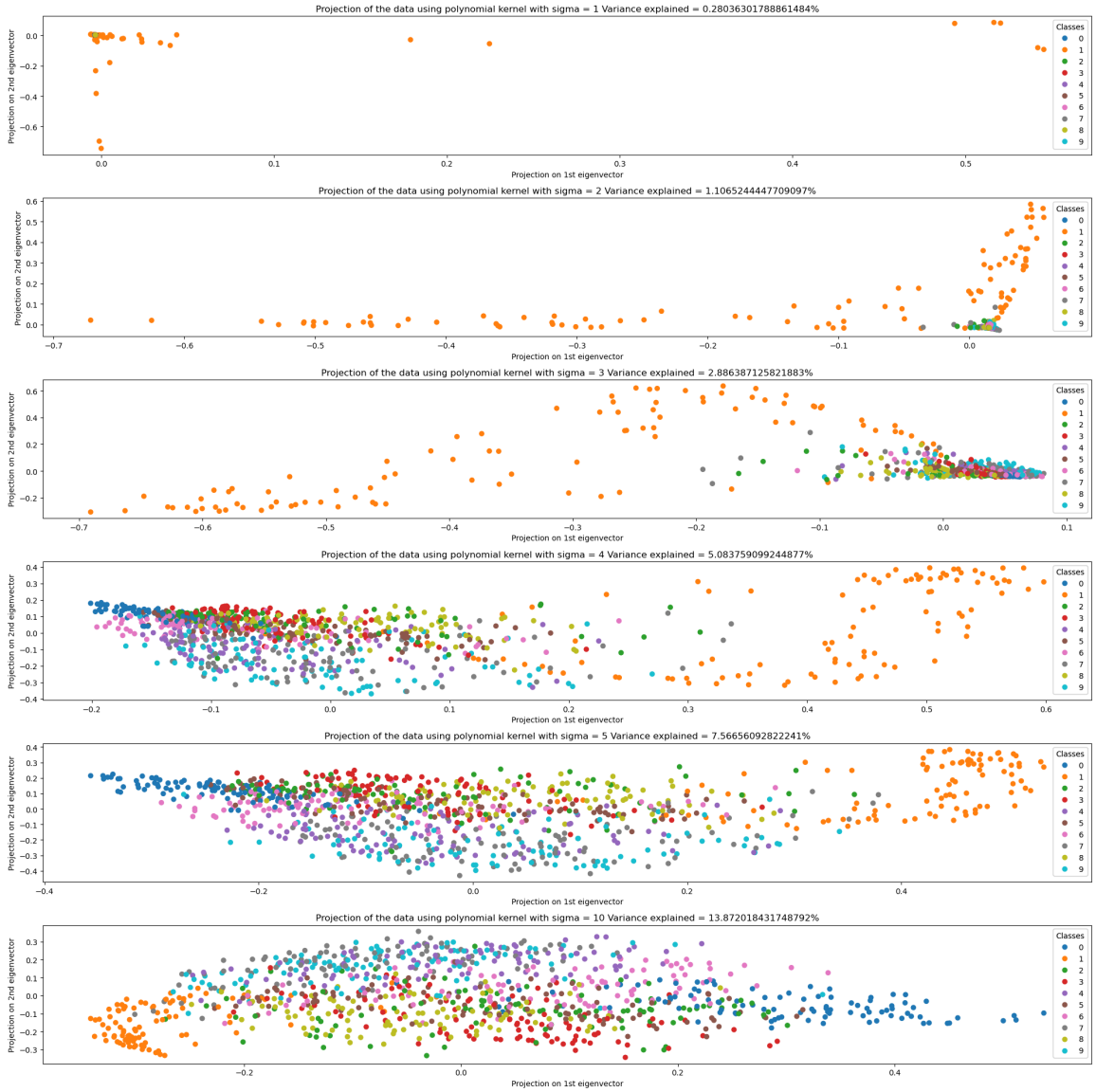


Figure 7: Projection Plots class wise for sigma = 1,2,3,4,5,10 with radial kernel

Note : I have used `torchvision.datasets.MNIST` to import the dataset (due to some issue with the Hugging Face module). Unlike the Hugging Face dataset, this dataset has all it's values normalized and in the range 0-1. Thus, the values of σ will be different for me than the rest! *Needless to say, the approach of finding the best σ remains the same.*

Observations:

(1) The 2D scatter of the projections tries to capture the underlying structure in the data explained by top 2 eigenvectors.

(2) There will be one particular value of σ which best explains the data.

(3) For $\sigma = 1$ the projected points are all very close to the origin and have very little projection values. The points are not separable. This indicates that σ can be of a higher value.

(4) For $\sigma = 2$ some projected points lie far from the origin but most of the points lie close to the origin. This again indicates that σ can still be higher.

(5) For $\sigma = 3$, the projected points are very diffcult to seperate. This indicates that the top 2 principal components are unable to capture variance of the data-points.

(6) For $\sigma = 4{,}5$ the projected points are not clumped together but seperated indicating higher variance.

(7) For $\sigma = 10$, the variance caputred is highest. The points are spread out and the principle components are able seperate the classes. This makes $\sigma = 10$ the best choice!

1.5 Q1 subpart (iv)

Which is better **Polynomial Kernel** or **Radial Basis Kernel**?
Ans : **Radial Basis Kernel**

- Radial Basis kernel maps the input features to an infinite space and is thus able to generate patterns which the polynomial kernel isn't able to.
- The points in the radial basis kernel($\sigma = 10$) are more spread out and capture more variance as compared to polynomial kernel($d = 2$).
- This shows that Radial Basis Kernel is much better at giving us an idea of the linear relationship between data-points in the high dimension.

2 K Means Clustering

2.1 Overview of the functions defined

(1) `def cal_mean` : Calculates the mean of the cluster points and stores it in an array called `mean_array`. Length of this array is k and each element in `mean_array` corresponds to the mean of a cluster

(2) `def reassign` : Given the new mean calculated from `def cal_mean`, it reassigns new clusters to data-points according to Lloyd's algorithm.

(3) `def cal_error` : Calculates the objective function which we are trying to minimize and returns it's value

(4) `def random_initialize` : Initializes each data-point to a random cluster at the beginning of the algorithm

(5) `def seed_initialize` : Initializes each data-point to a cluster with a fixed seed such that this process is not random anymore.

(6) `def plot_kmeans` : Takes input data-points, clusters to which they belong to and cluster centres to plot them with different colours.

(7) `def polynomial_kernel` : Takes in input the dataset and p to find the Kernel Matrix H using the polynomial kernel.

(8) `def radial_kernel` : Takes in input the dataset and σ to find the Kernel Matrix H using the radial kernel.

(9) `def bubble_sort` : It is used to sort the eigenvalues and eigenvectors obtained from np.linalg.eigh(H) in descending order of eigenvalues.

(10) `def cal_eigen` : Calculates eigenvalues and eigenvectors of the H matrix and returns these.

(11) `def spectral_means` : Takes the top k eigenvectors of the H kernel matrix and forms the new dataset by normalizing the rows of eigenvectors.

(12) `def kmeans` : Performs k means algorithm for 50 iterations and returns clusters, means and error.

(13) `def map_back` : Plots the original dataset with the clusters and centres obtained from either Polynomial kernel, Radial kernel or the modified kmeans algorithm

(14) `def modified_clustering` : Performs the clustering given in (iv) subpart of the assignment.

2.2 Q2 subpart (i)

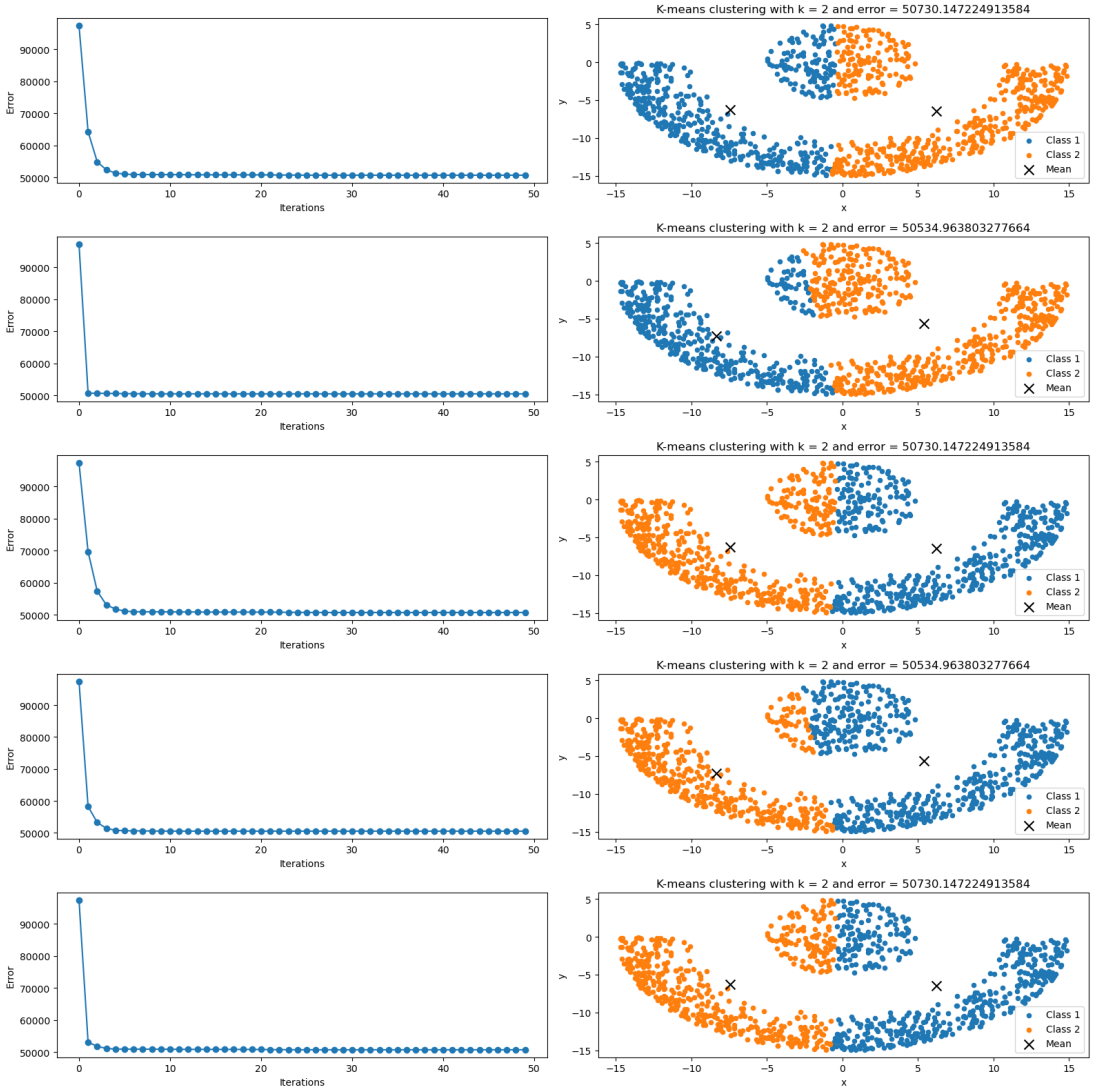


Figure 8: Cluster and Error plot for random initializations

Observations:

(1) 5 random initializations using $z_i =$ `np.random.randint(1, k+1)`, where z_i = 1,2,.....,k indicating which cluster data point x_i goes to.

(2) For each random initialization the algorithm converges within a few iterations.

(3) The final cluster arrangement depends a lot on the intialization.

2.3 Q2 subpart (ii)

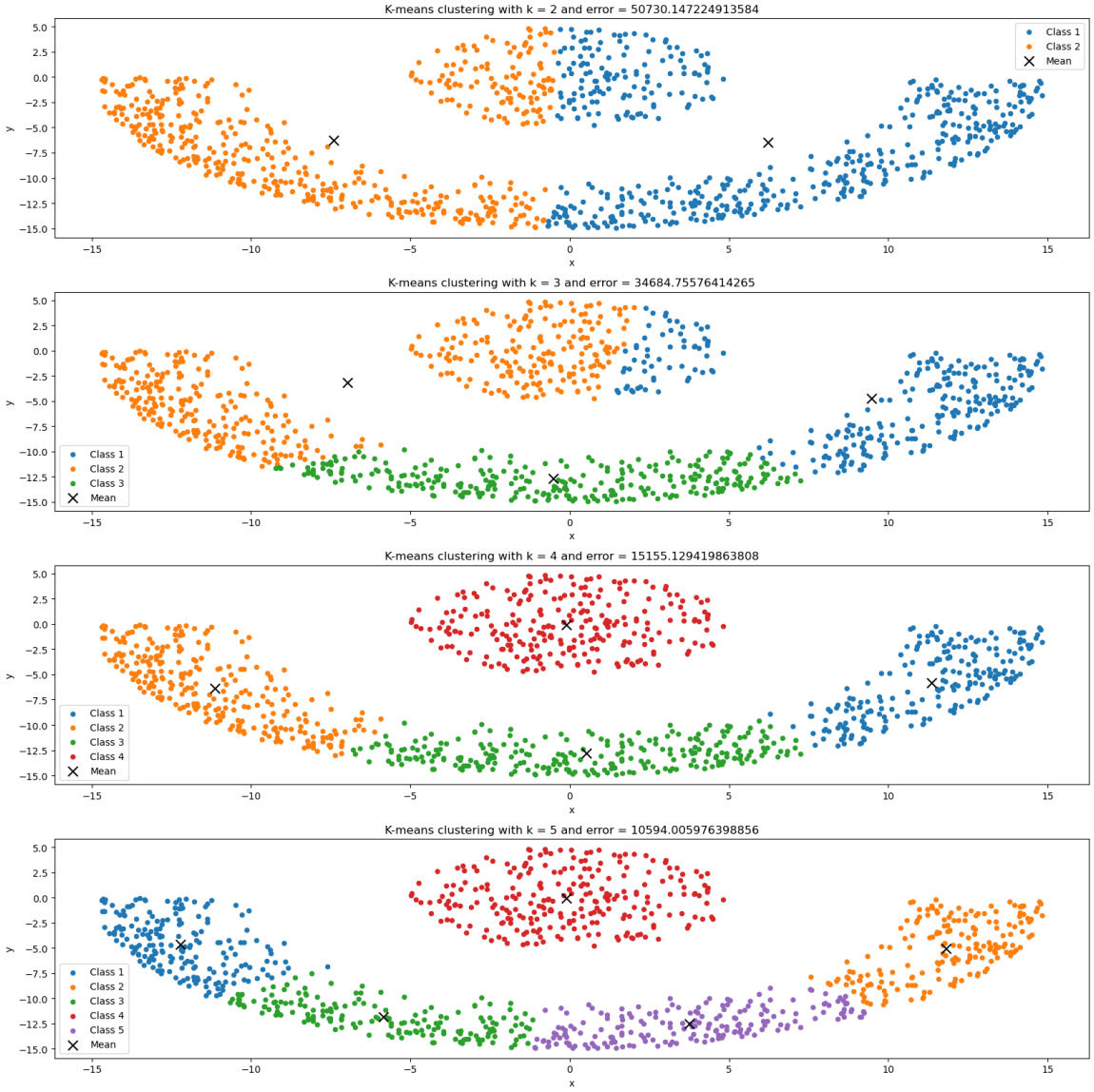


Figure 9: Vornoi Regions

Observations:

(1) Used `np.random.seed(0)`, fixing the initializations.

(2) Region where points are closest to a centre is that centre's Vornoi Region

2.4 Q2 subpart (iii)

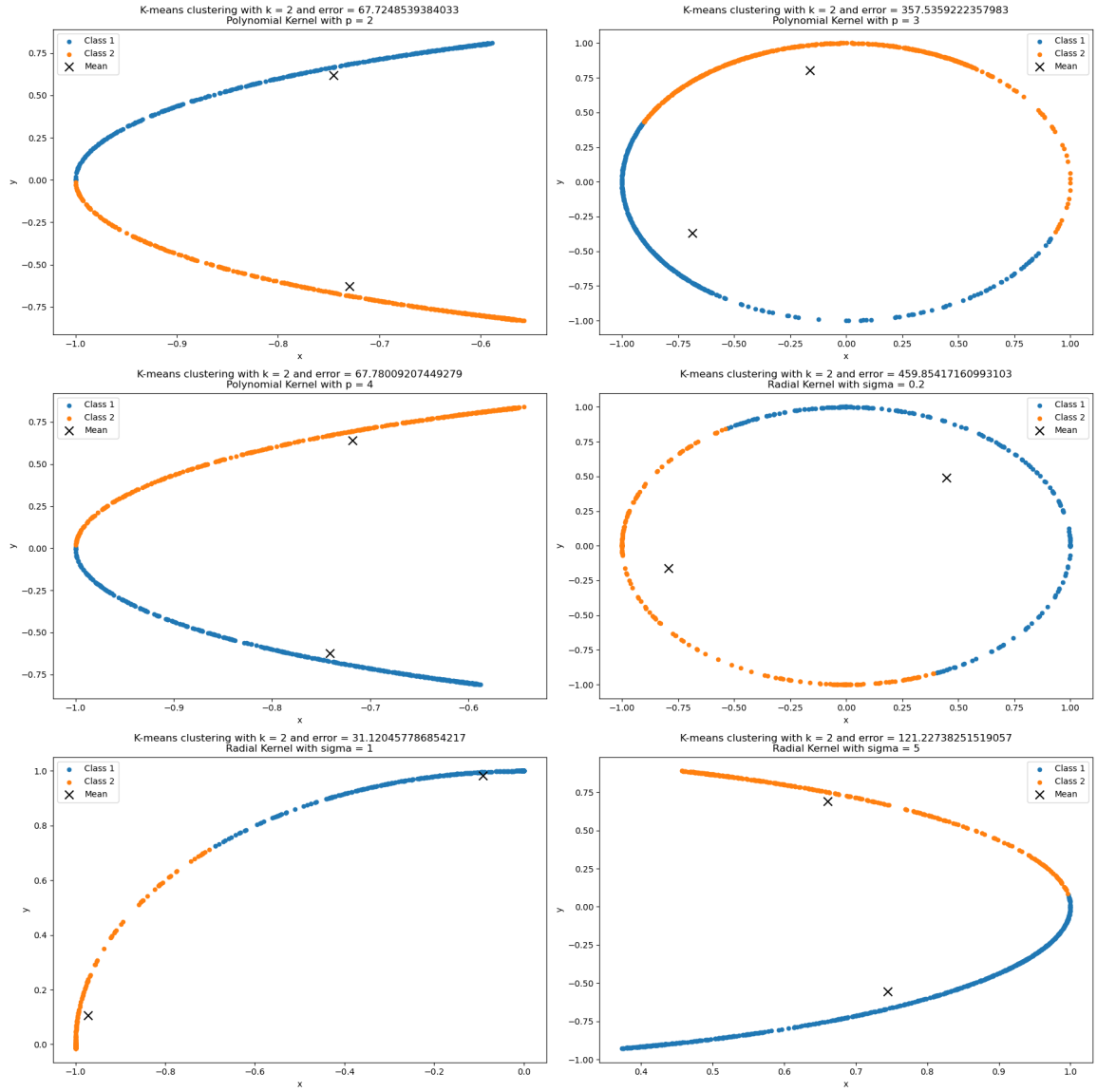


Figure 10: Spectral Clustering with Polynomial and Radial Kernels

Observations:

(1) The Error for Polynomial Kernel with $p = 3$ is low when mapped backed to the original dataset, This suggests that a cubic kernel might be suitable.

(2) Low valued errors for Polynomial Kernels with $p = 2$ & 4 in the spectral dimension but high errors when mapped back to the original dimension indicates that they are not suitable.

(3) $\sigma = 1$ has the least error in the spectral dimension but a higher error when mapped back to the original dataset.

(4) $\sigma = 5$ has the least error among all the other kernels when mapped back to the original dataset, thus clustering the data-points effectively. This makes it the most suitable kernel for this task!

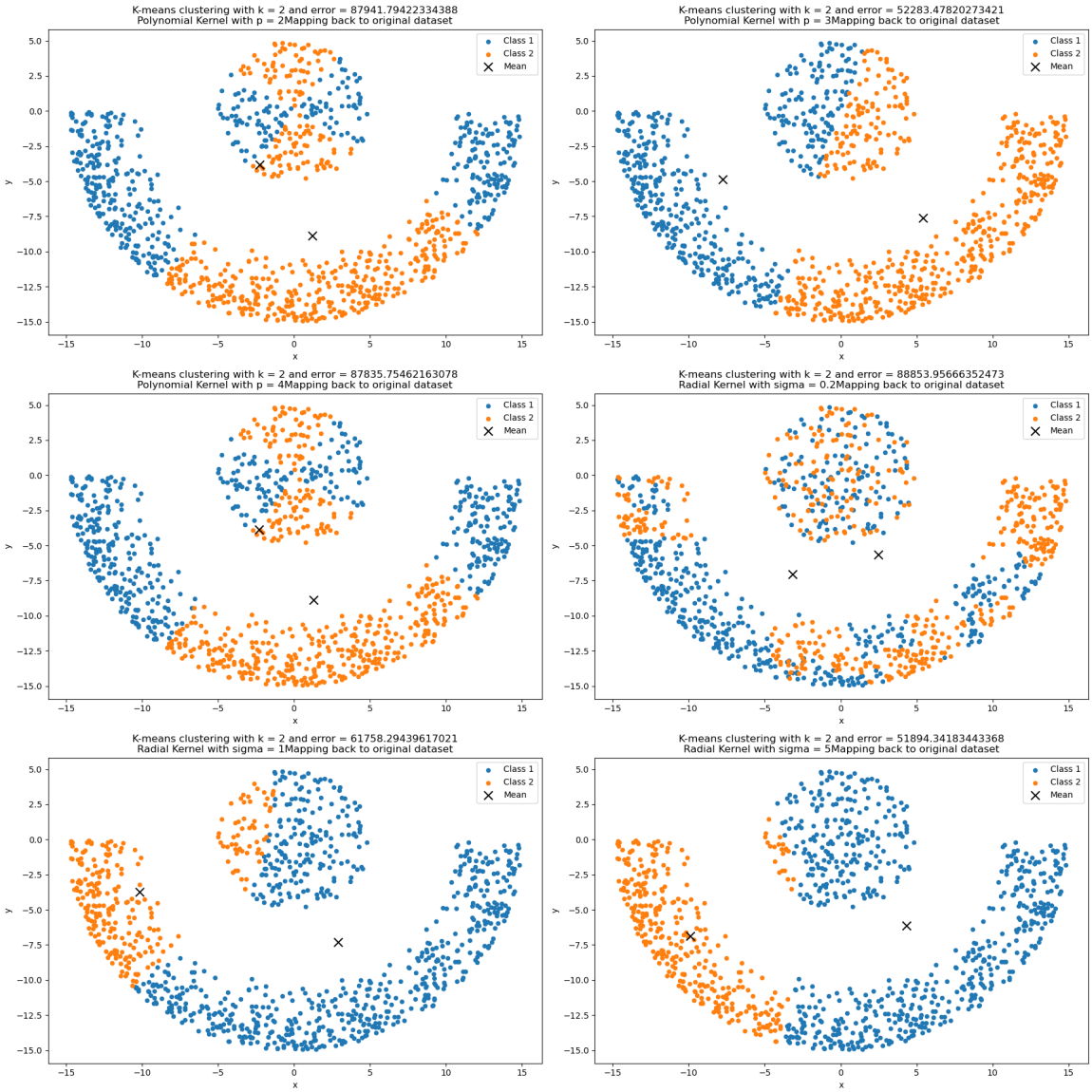


Figure 11: Clusters Mapped Back to original dataset after Spectral Clustering

2.5 Q2 subpart (iv)

Observations:

(1) The modified algorithm first calculates the Kernel Matrix H. Cluster that data point x_i belongs to $= l = \arg\max_{j=1,\ldots,k} H_{ij}$.

(2) This mapping performs worse than both the polynomial kernel and the radial kernel.

(3) This means that, just choosing the highest value in each row is not sufficient to assign the clusters. This means that the H computed cannot be represented as $ZL^{1/2}$. That is, the chosen H lies outside the inner circle of $ZL^{1/2}$.

(4) The original dataset has the least error when $\sigma = 5$.

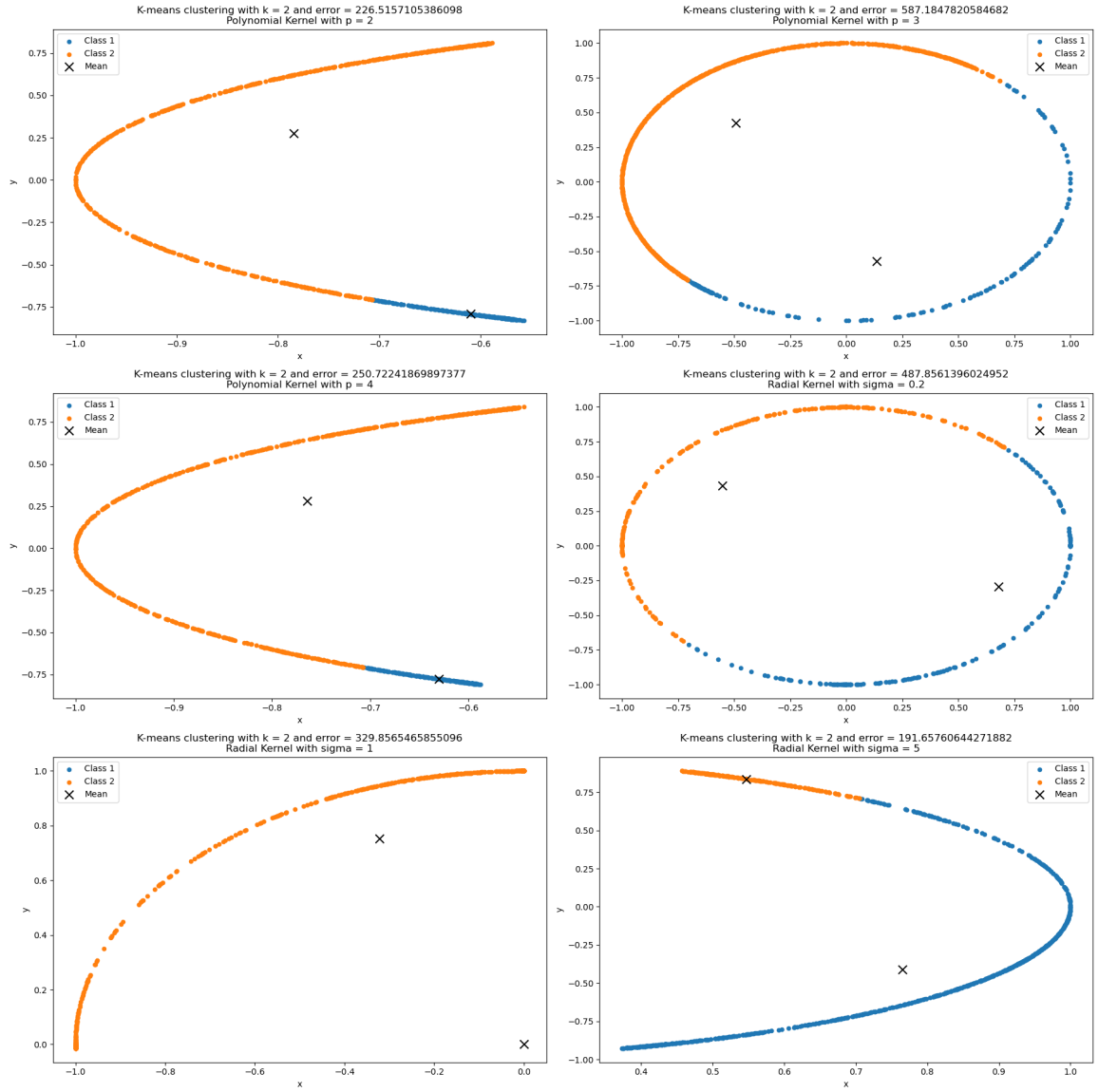


Figure 12: Modified Clustering Algorithm with Polynomial and Radial Kernels

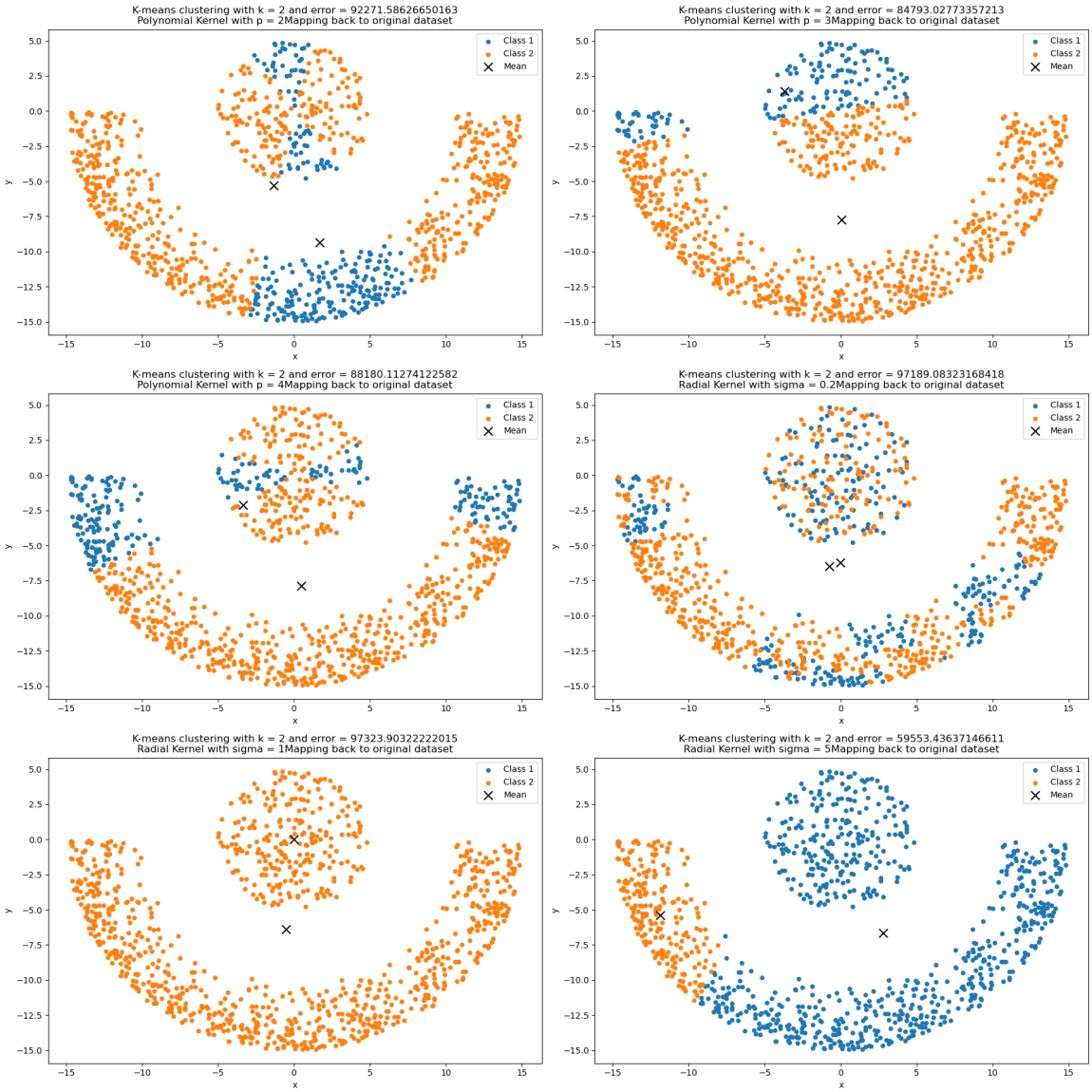


Figure 13: Clusters Mapped Back to original dataset after Modified Clustering Algorithm

******