CS5691 Pattern Recognition and Machine Learning

Assignment 1

Ruthwik Chivukula

ME21B166

Contents

1 PCA: Principal Component Analysis

In this section, we will be implementing PCA (both Vanilla and Kernel) on a subset of the MNIST dataset consisting of 1000 images uniformly distributed over 10 classes and observing the results to draw useful insights. The MNIST dataset is a collection of handwritten digits between 0 to 9 with a shape of 28x28, below are a few samples from the dataset:

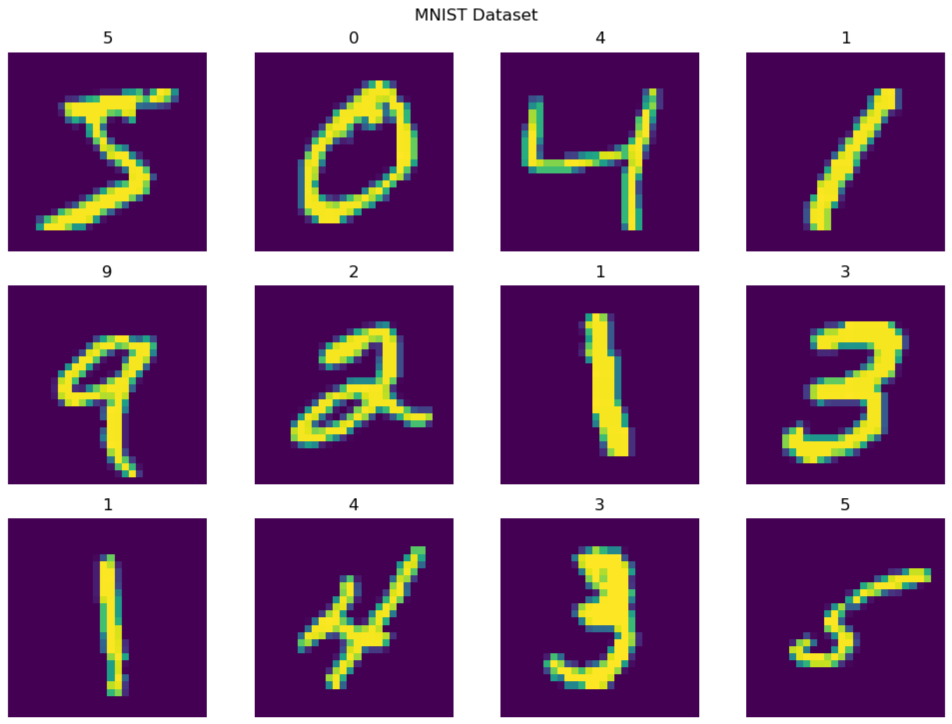

Figure 1: MNIST Dataset

1.1 Vanilla PCA and Variance Explanation

Flattened the 28x28 image to 784x1 and centered it before applying PCA over the 1000 images. Collected the principal components and reshaped them into 28x28 for analysis. Below are the images of the top 12 principal components along with their variance %. To capture **95 %** of the variance, only the **top 130** principal components will be required (obtained from code), which portrays the compression and feature extraction property of PCA.

The first principal component appears like a zero, which explains that most of the data points have features similar to the digit zero (like the curves and turns). The remaining principal components appear to be a superposition of different digits.

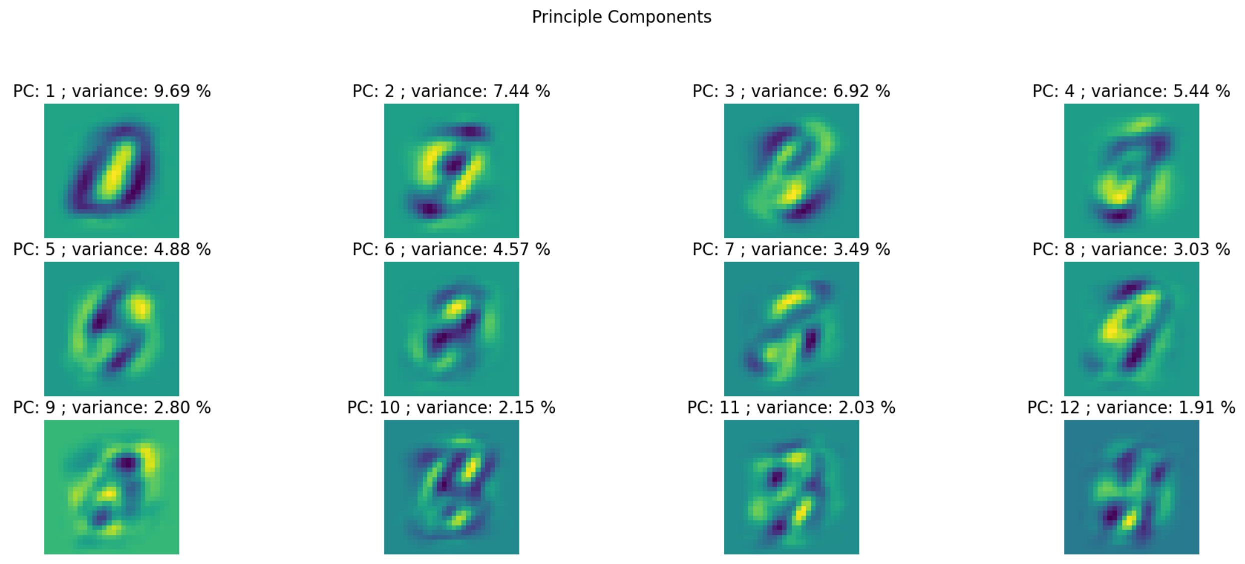

Figure 2: Top 12 Principal Components

1.2 Reconstruction

Reconstruction expresses the data points as a linear combination of the principal components. So we want to choose a d < 784 which helps us obtain the desired image reconstruction.

I tried for different values of d ranging from 50 to 784. For 784 obviously, the reconstructed image will be the same as the actual image but our purpose is to compress so d has to be < 784. As mentioned earlier for d $= 130$, 95% of the variance in the dataset is captured hence **d = 130 would be good enough to reconstruct the images**.

Note: While plotting I have added the mean to the reconstructed images to compare it with the actual images.

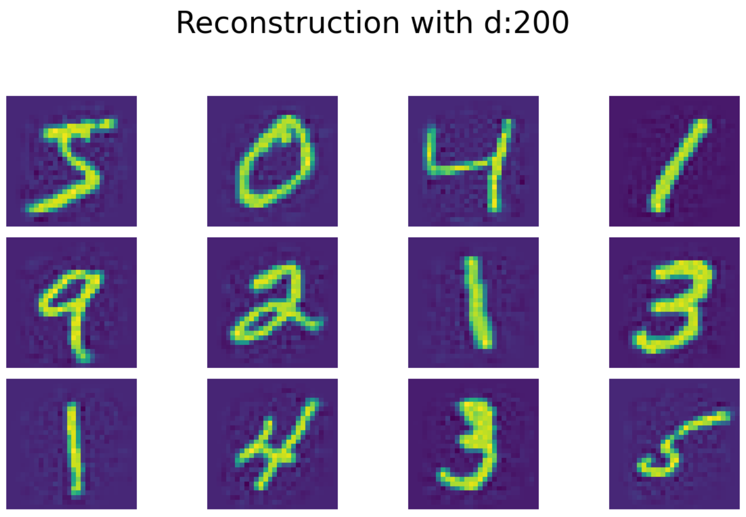

Figure 3: Reconstructed Samples with top 200 components

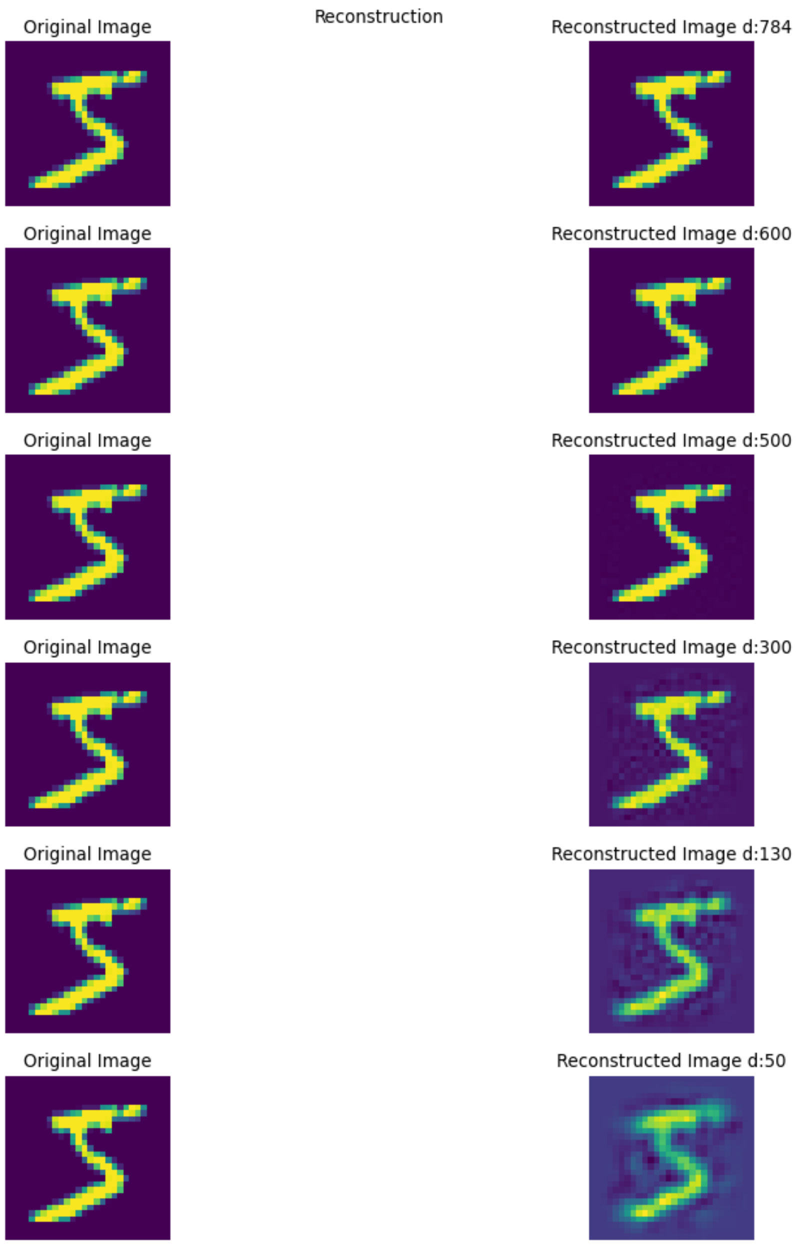
Reconstruction
Original Image
Reconstructed Image d:784
Original Image
Reconstructed Image d:600
Original Image
Reconstructed Image d:500
Original Image
Reconstructed Image d:300
Original Image
Reconstructed Image d:130
Original Image
Reconstructed Image d:50

Figure 4: Reconstructed Samples for different d values

Depending on the use case, the d that we choose will vary, here as the dataset is not intricately detailed considering only 95 % variance should suffice.

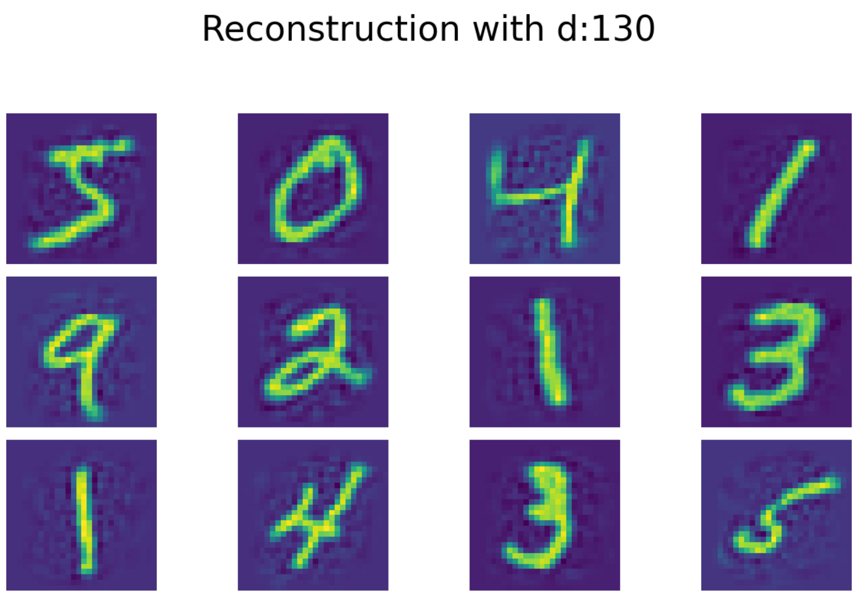

Figure 5: Reconstructed Samples for d = 130

1.3 Kernel PCA

Vanilla PCA learns only linear manifolds but the underlying relation between the principal components may be non-linear. To tackle this problem, we adopt Kernel PCA which uses kernel operations to move to a higher dimension where the data points can be expressed as a linear combination of principal components. Here in this section, we look into particularly Polynomial and Radial Basis Function (RBF) kernels.

For polynomial kernel, d = 2, 3 and 4 have been tested and the results are as shown below. To comment on the performance of the kernel PCA, I have used the ground truth labels to color the data points which can help us draw better insights.

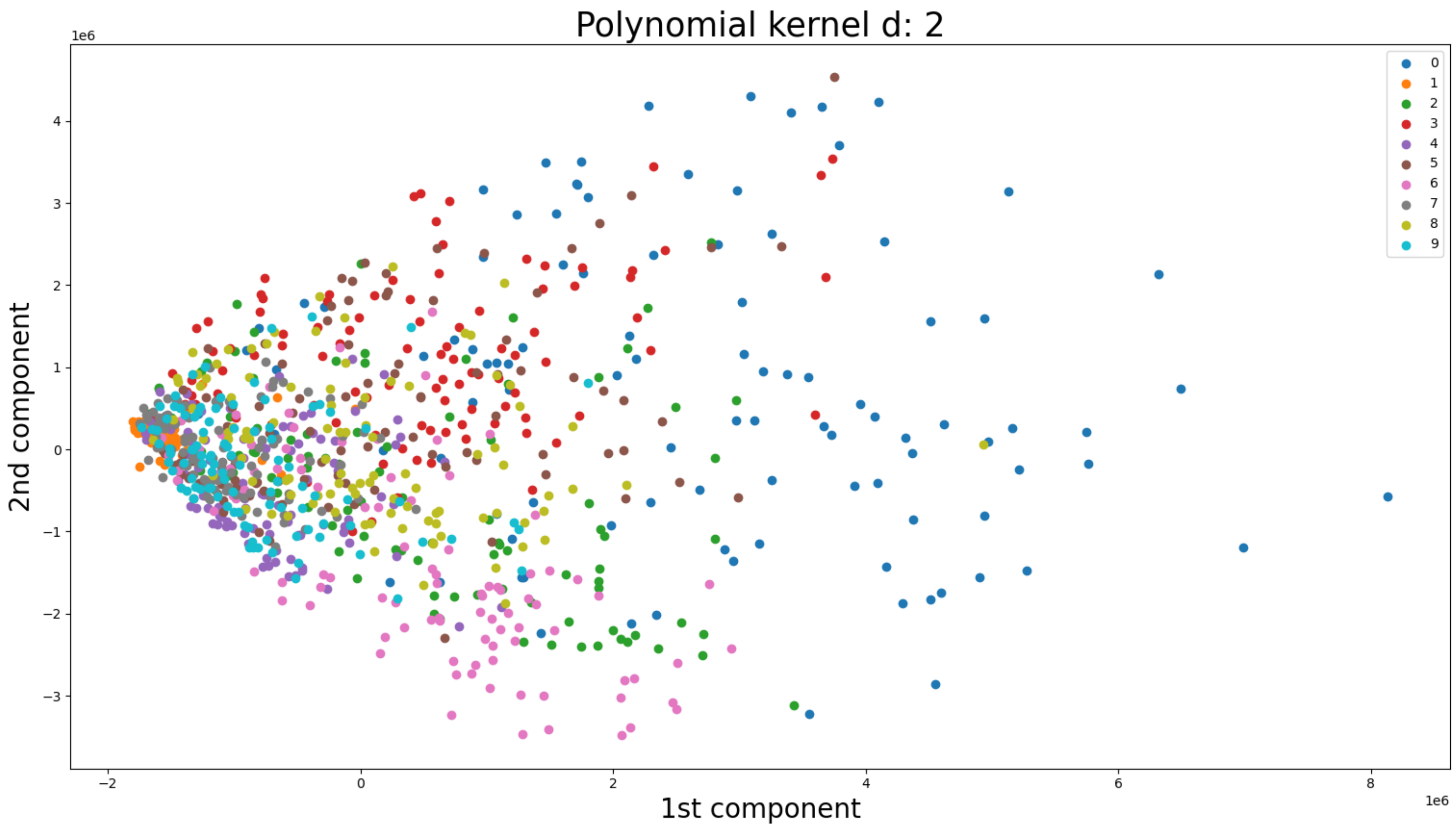

Figure 6: Polynomial kernel d = 2

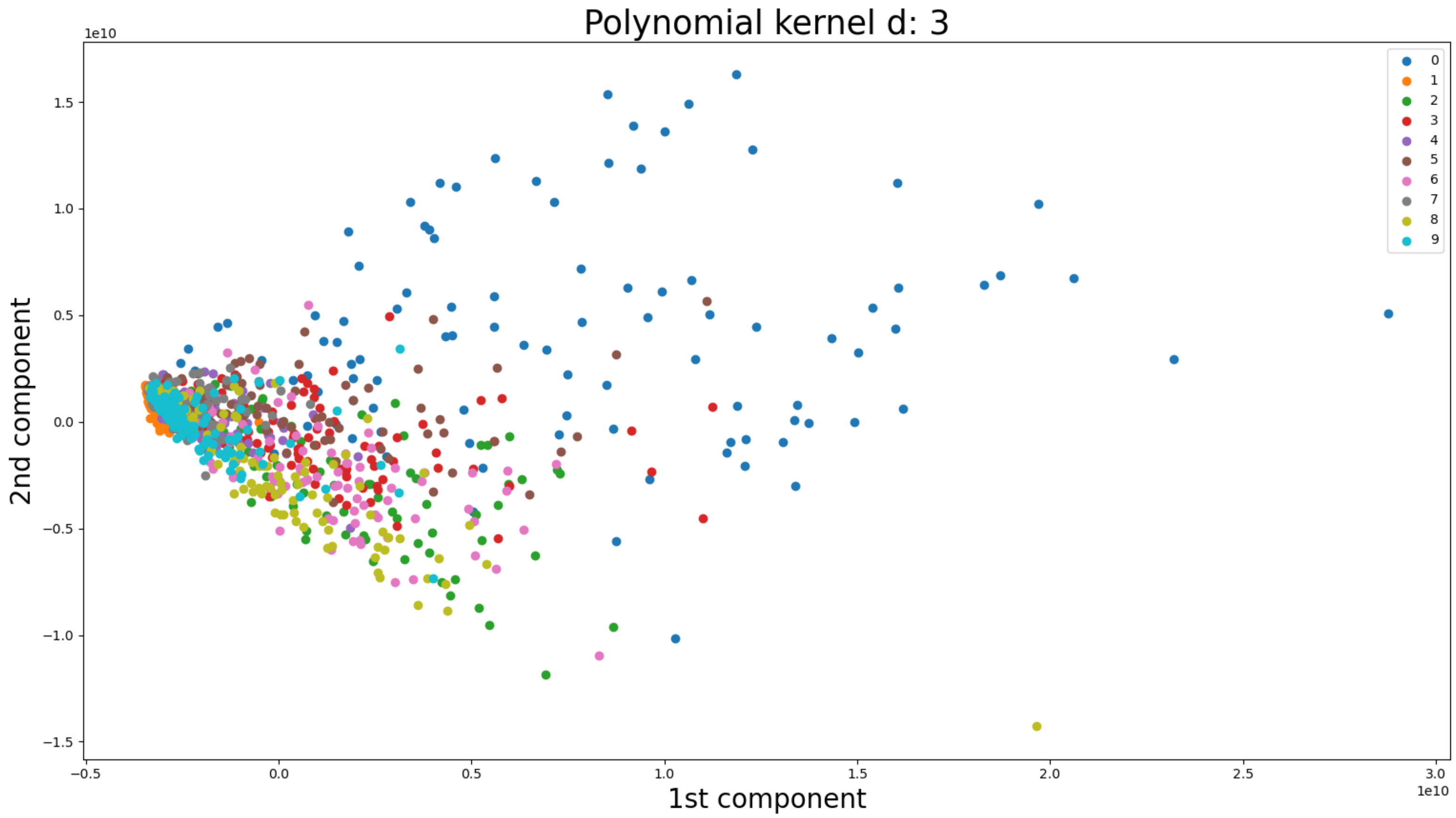

Figure 7: Polynomial kernel d = 3

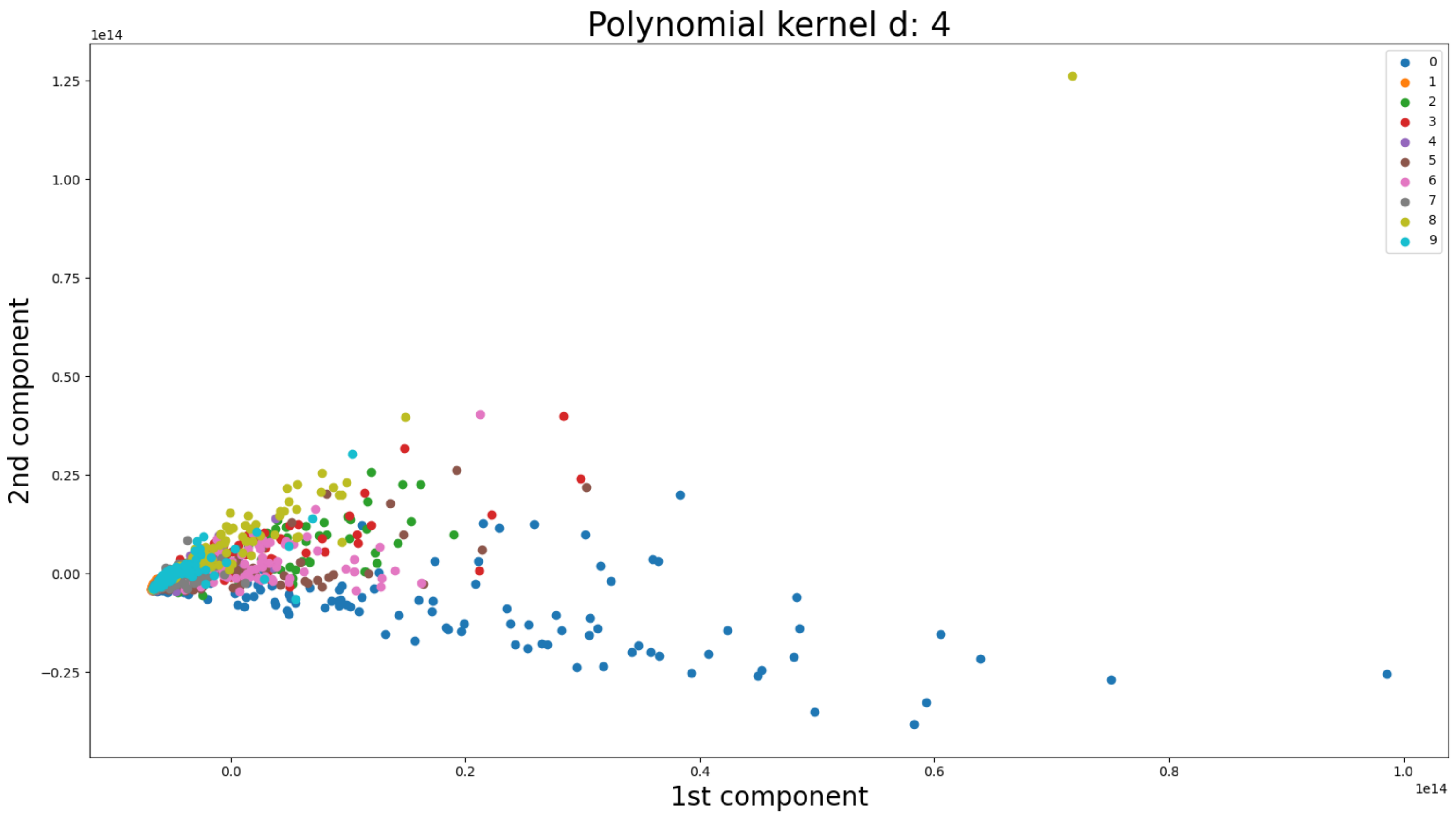

Figure 8: Polynomial kernel d = 4

If I had to compare among polynomial kernels, d = 2 is better than d = 3 is better than d = 4. The argument would be: the purpose of applying PCA is to find the directions in which there is high variance and represent the data as a combination of those directions. So having a more spread-out plot implies more variance and hence a better option.

Note: I have used color labels just for better interpretation. In general, as this is an unsupervised learning technique we won't have access to the ground truth labels. So without labels, the plot remains the same just that all scatter points will be of the same color, unlike the current plot.

We can observe that generally the digit 0 class is spread out (high variance) whereas the digit 9 class is clustered (low variance). Now let us plot for RBF kernel using different sigma values. I have tried out sigma = 1, 100, 1000 and 10000. Based on the plots I have decided what values are ideal.

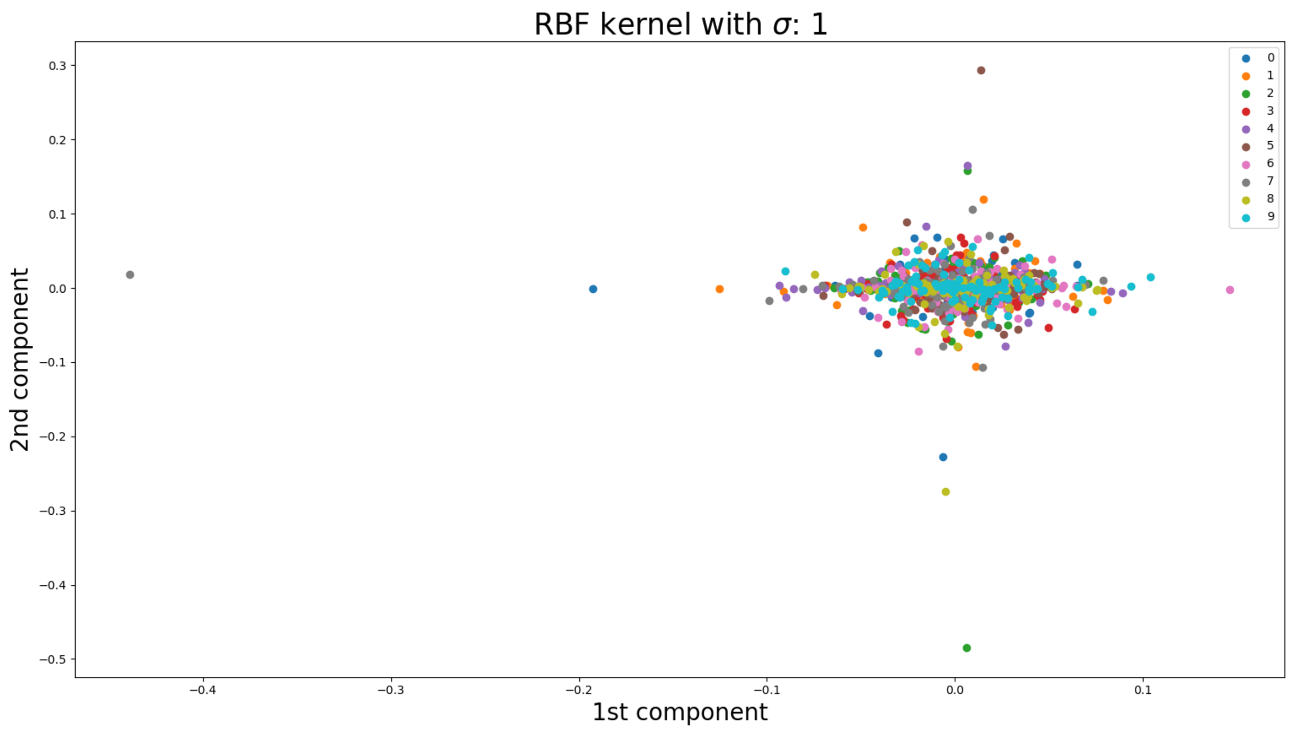

Figure 9: RBF kernel sigma = 1

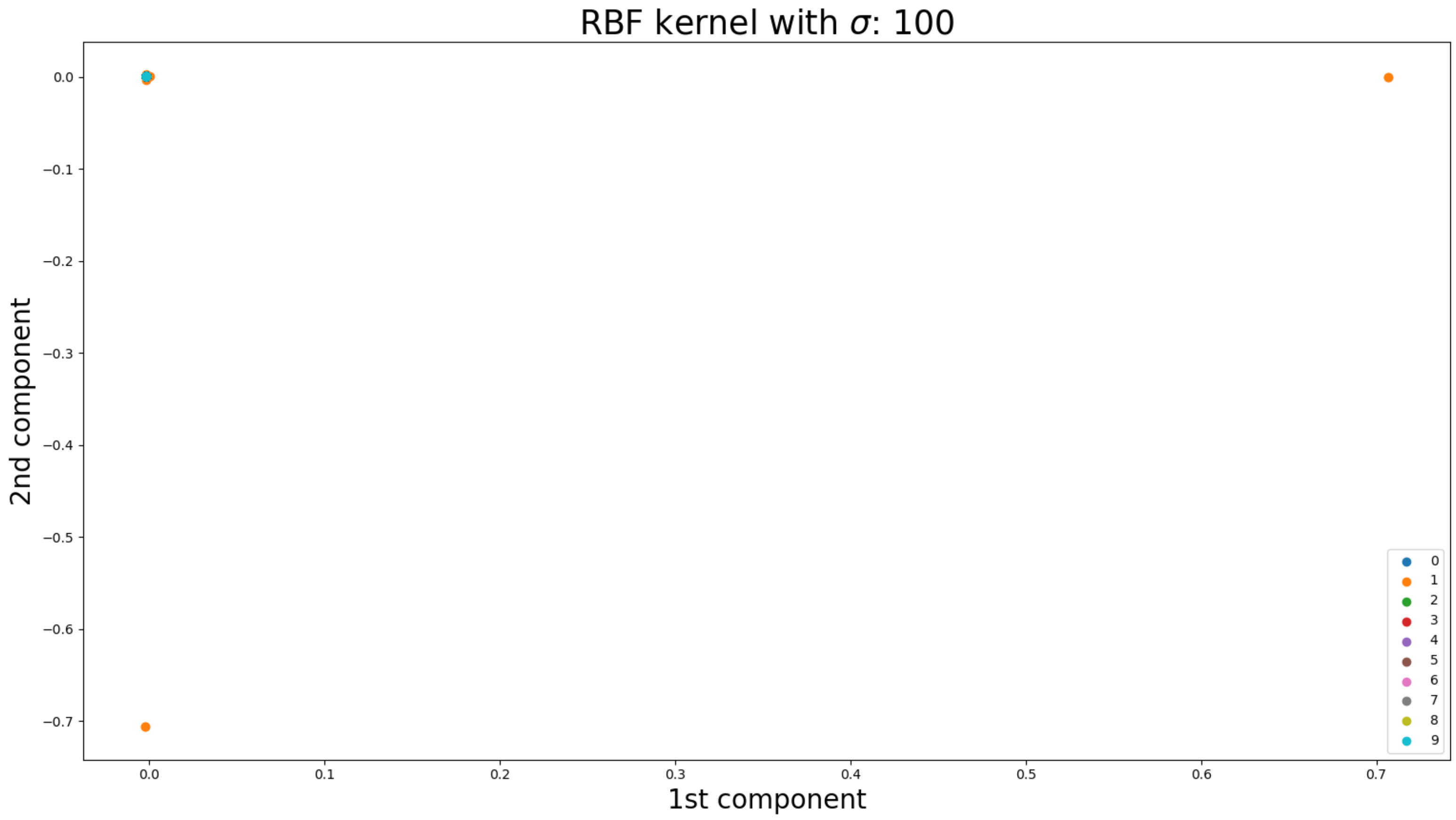

Figure 10: RBF kernel sigma = 100

For sigma = 1, the majority of the data points are clustered around (0,0) which is not desired. For sigma = 100, surprisingly all the data points are densely populated at three different locations. So both of these are not appropriate values of sigma. The argument is the same as above, we want data points to be scattered for high variance. The projections on the top 2 components hint at the variance along those components which needs to be as high as possible.

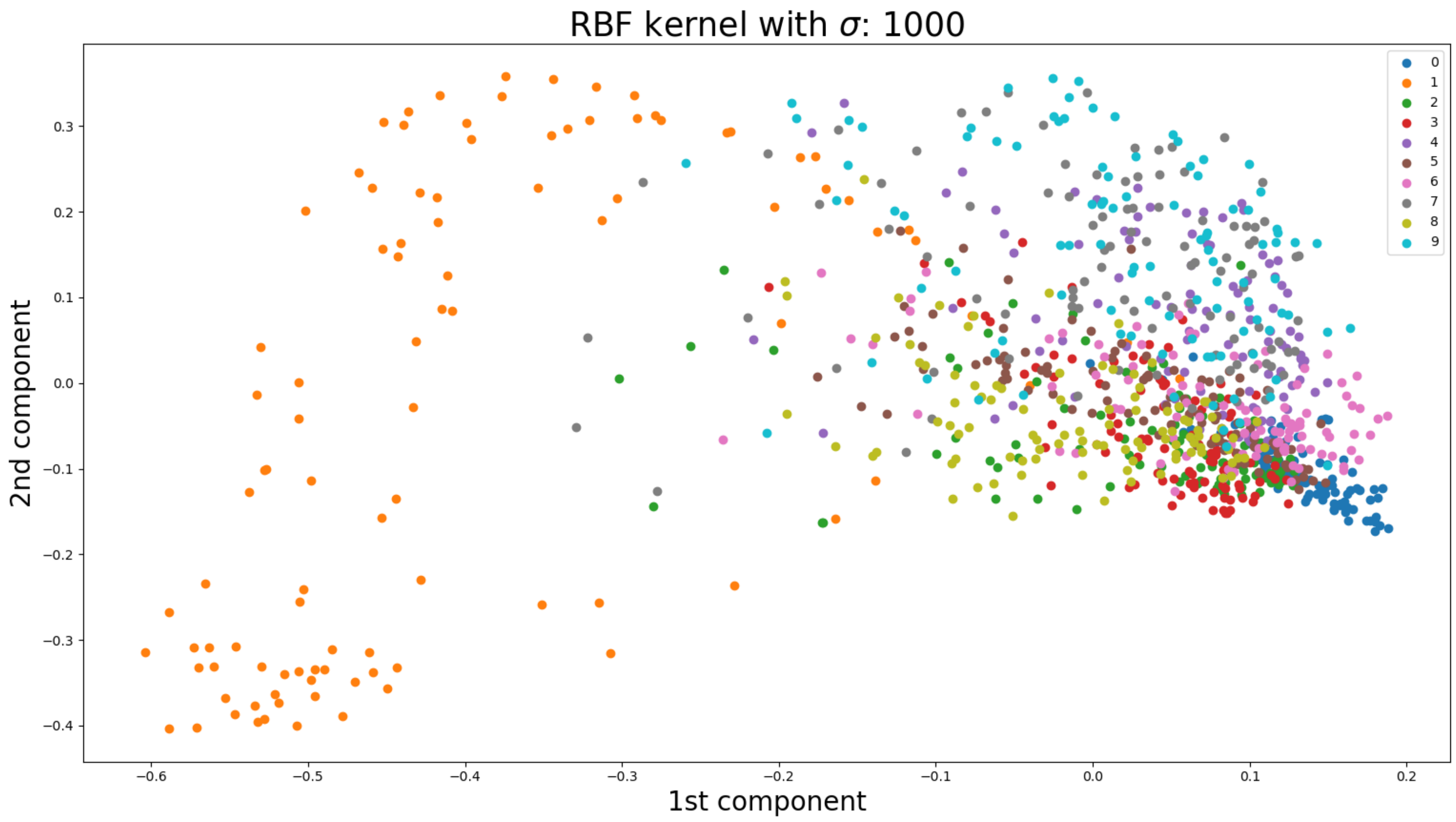

Figure 11: RBF kernel sigma = 1000

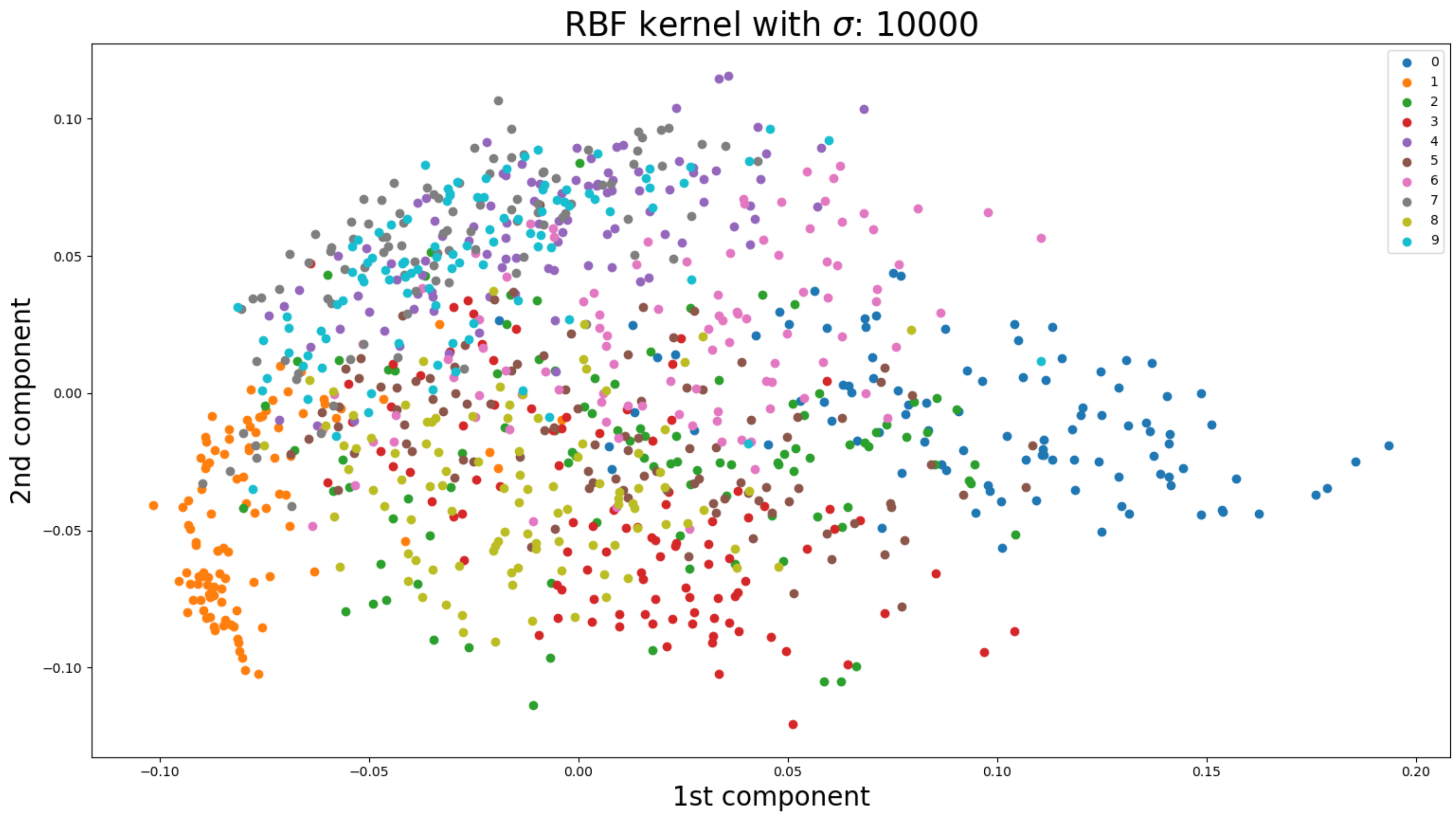

Figure 12: RBF kernel sigma = 10000

For sigma = 1000 and 10000, the data points are well scattered and spread out. sigma = 10000 is the best option which I would go for. Now comparing the best contestants from both sides, polynomial and RBF, i.e. d = 2 vs sigma = 10000. RBF with sigma = 10000 is what I would choose.

2 K-means

In this section, we will be applying different versions of K-means to the dataset and observing how the results are. The first section explores the vanilla K-means with different initialization. Next, we try out spectral clustering using different kernel functions and parameters.

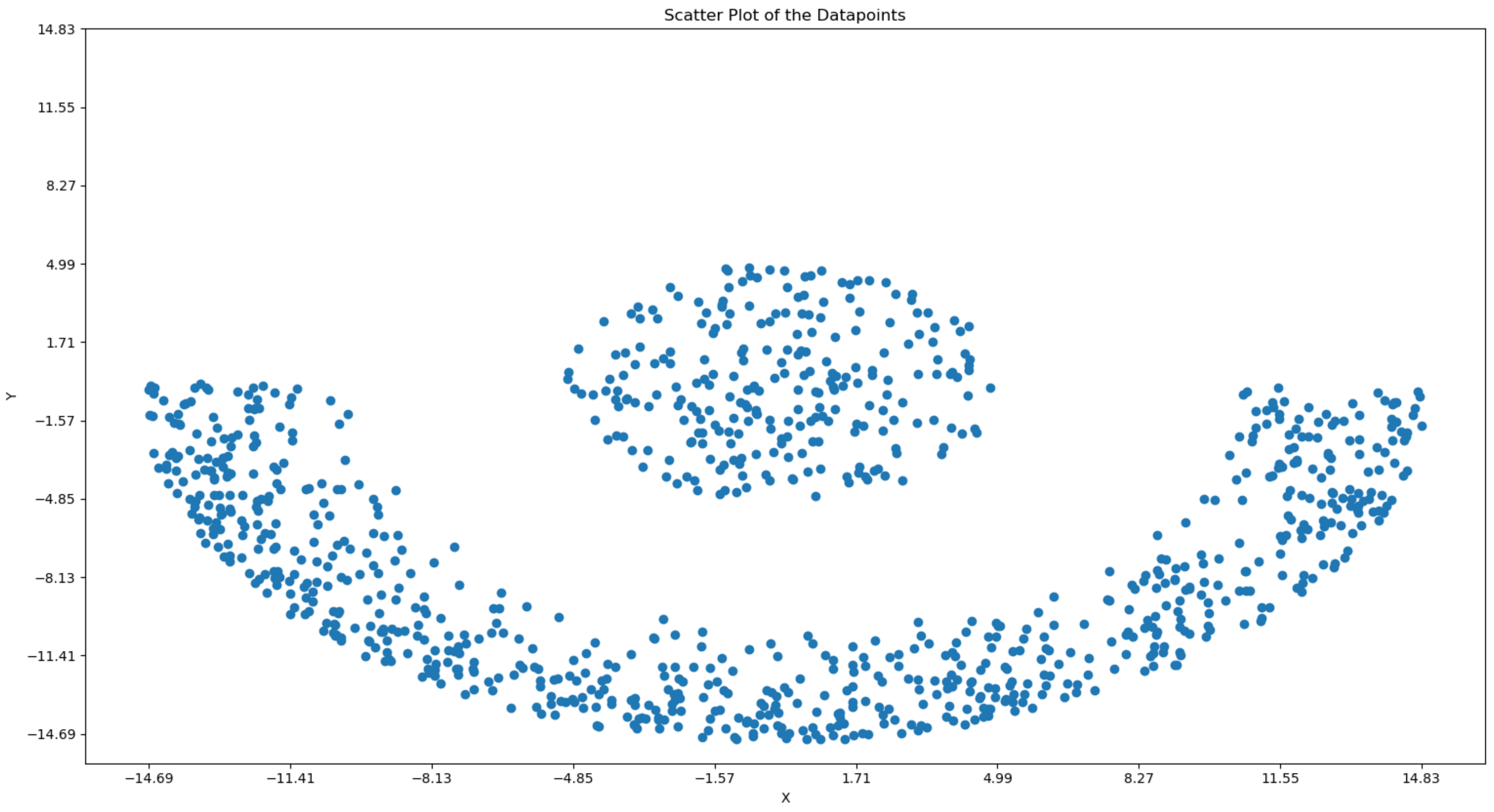

Figure 13: Datapoints

2.1 K-means for k =2 and 5 different random initialization

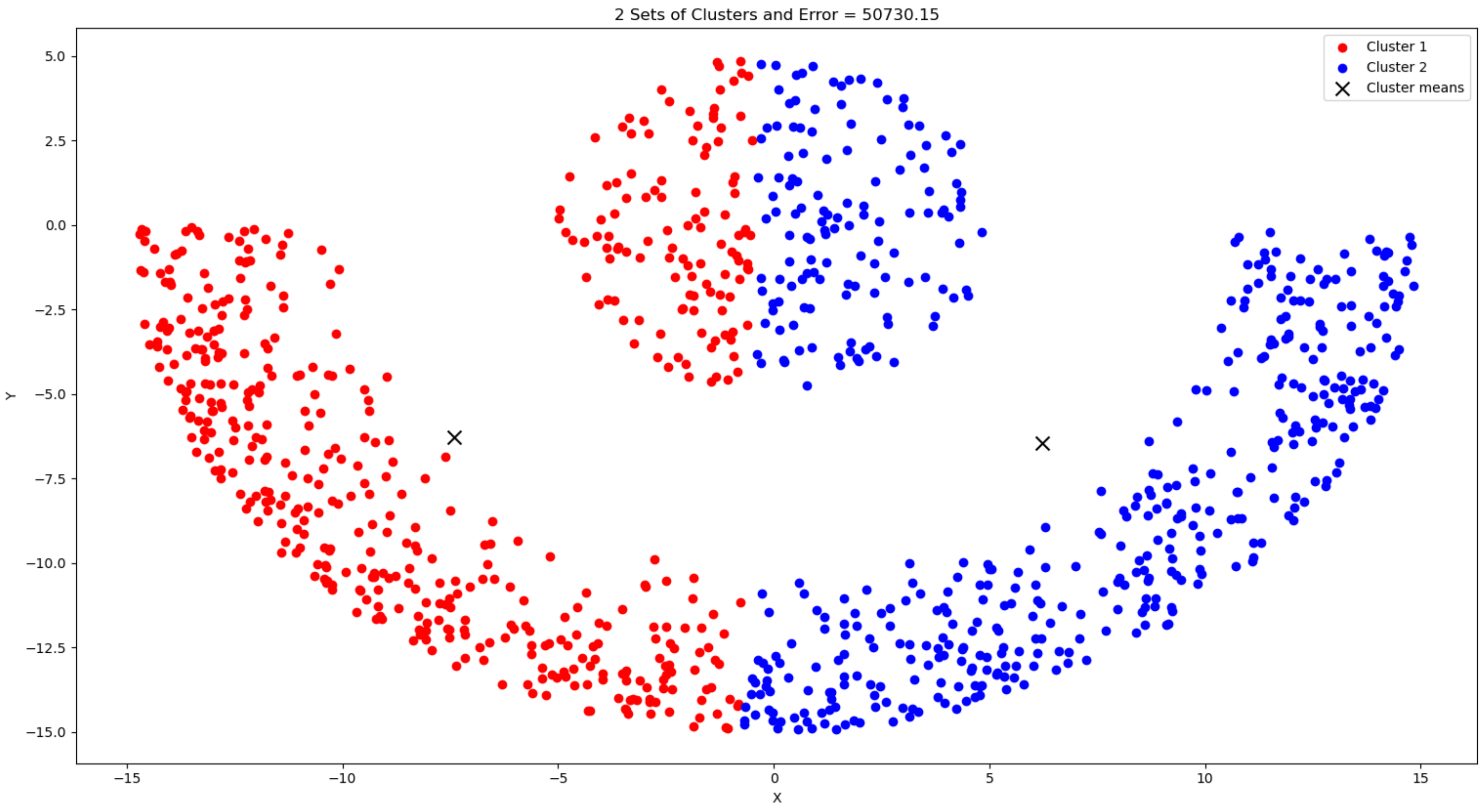

Figure 14: Random Initialization 1 clusters

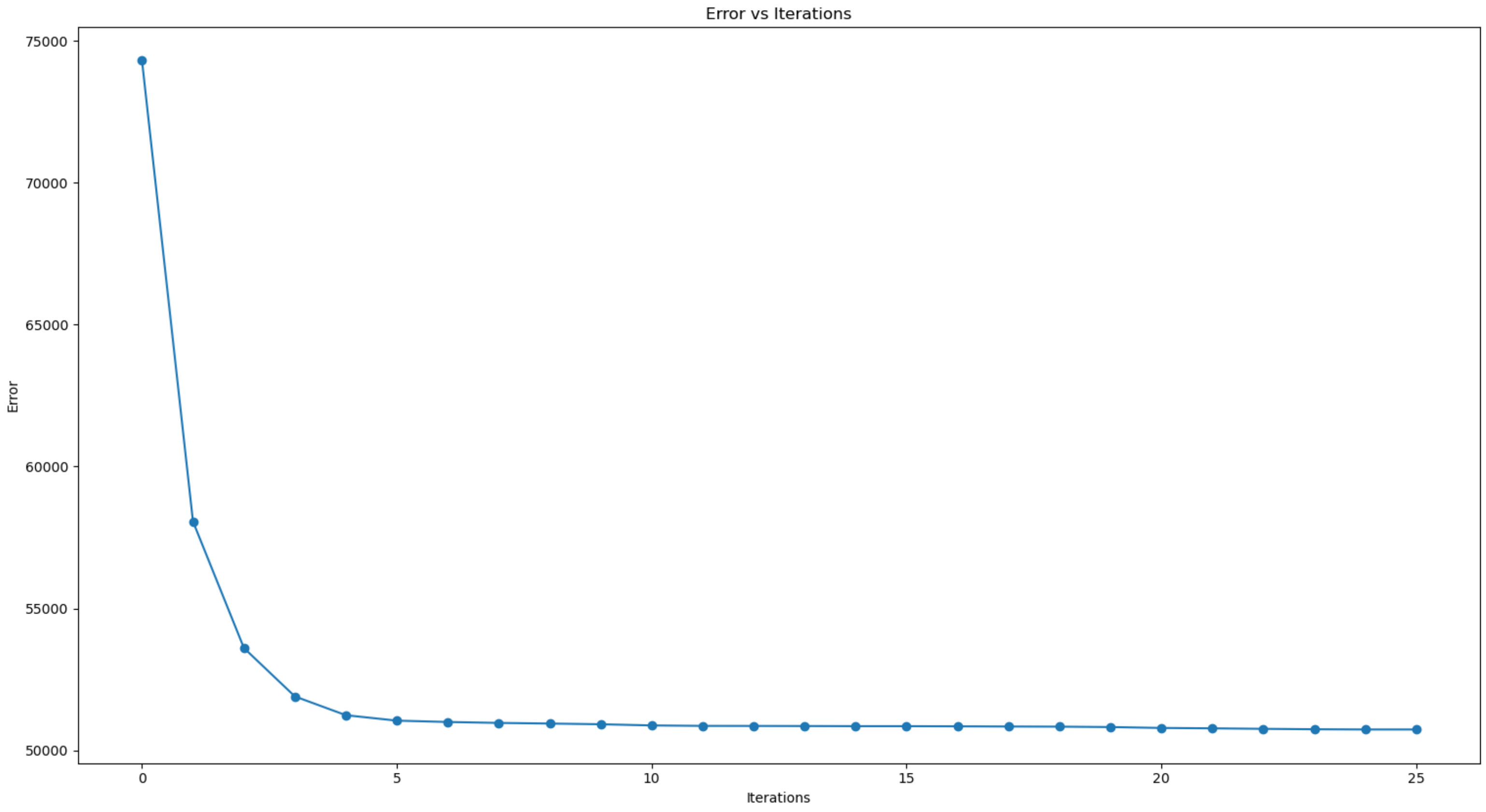

Figure 15: Random Initialization 1 error plot

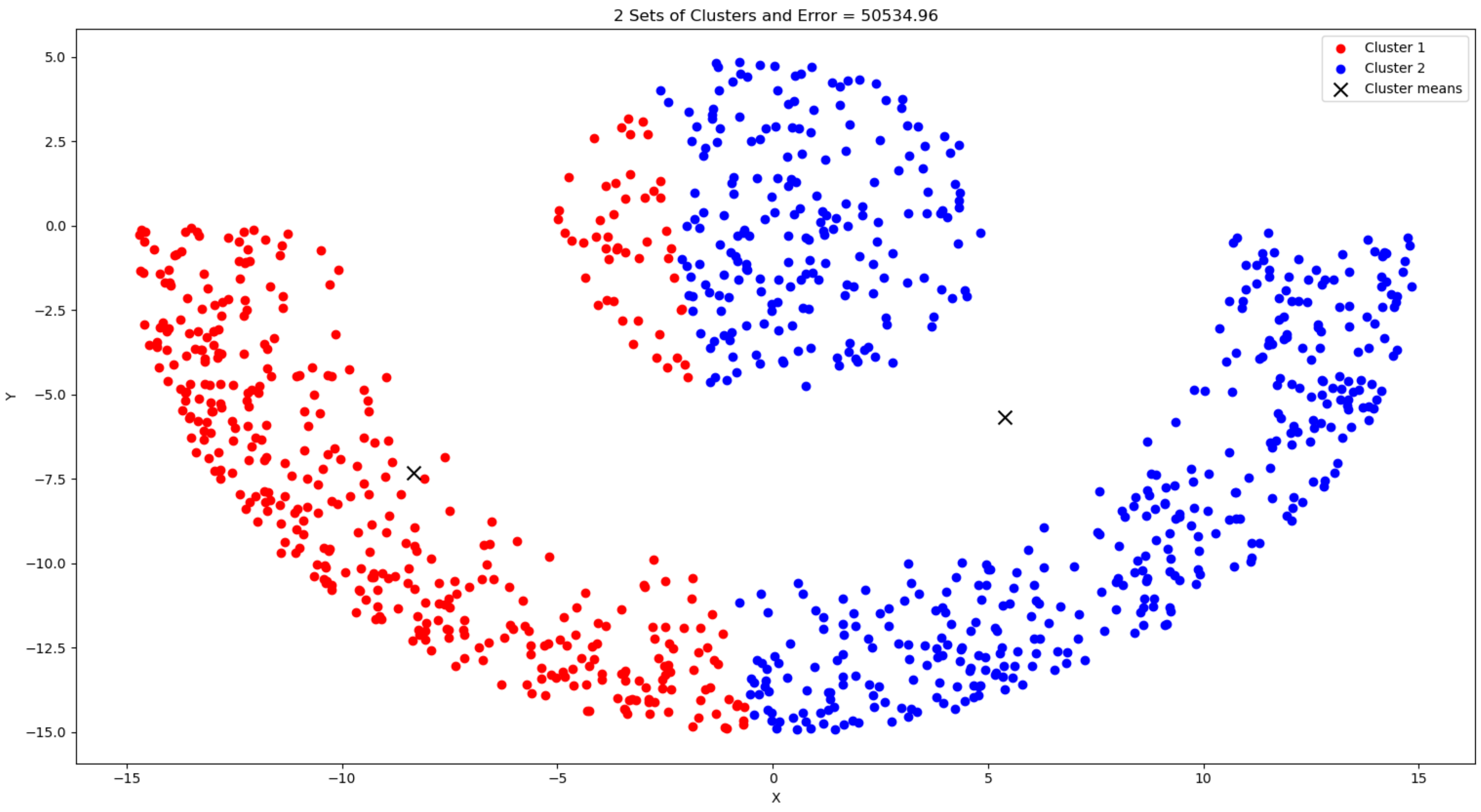

Figure 16: Random Initialization 2 clusters

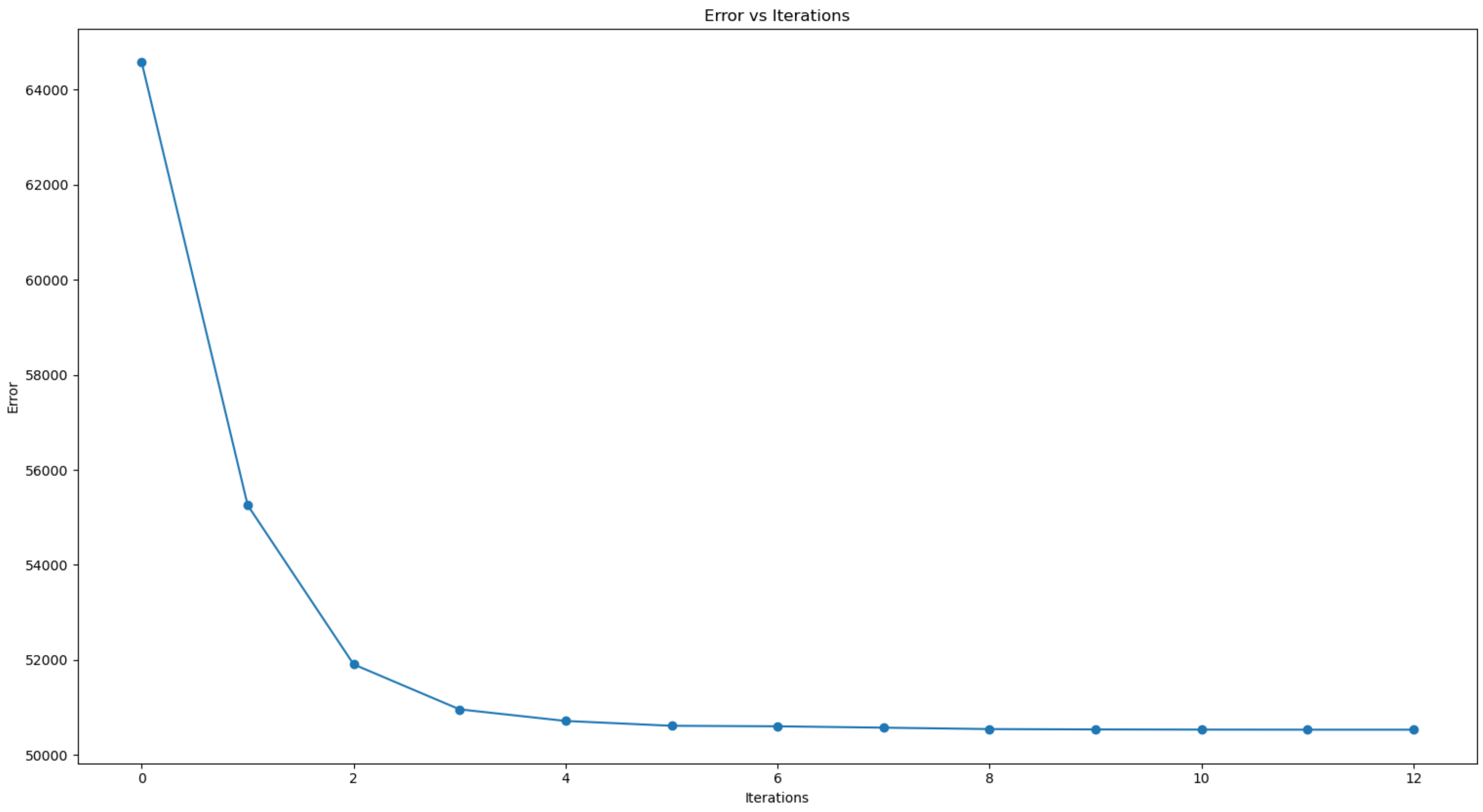

Figure 17: Random Initialization 2 error plot

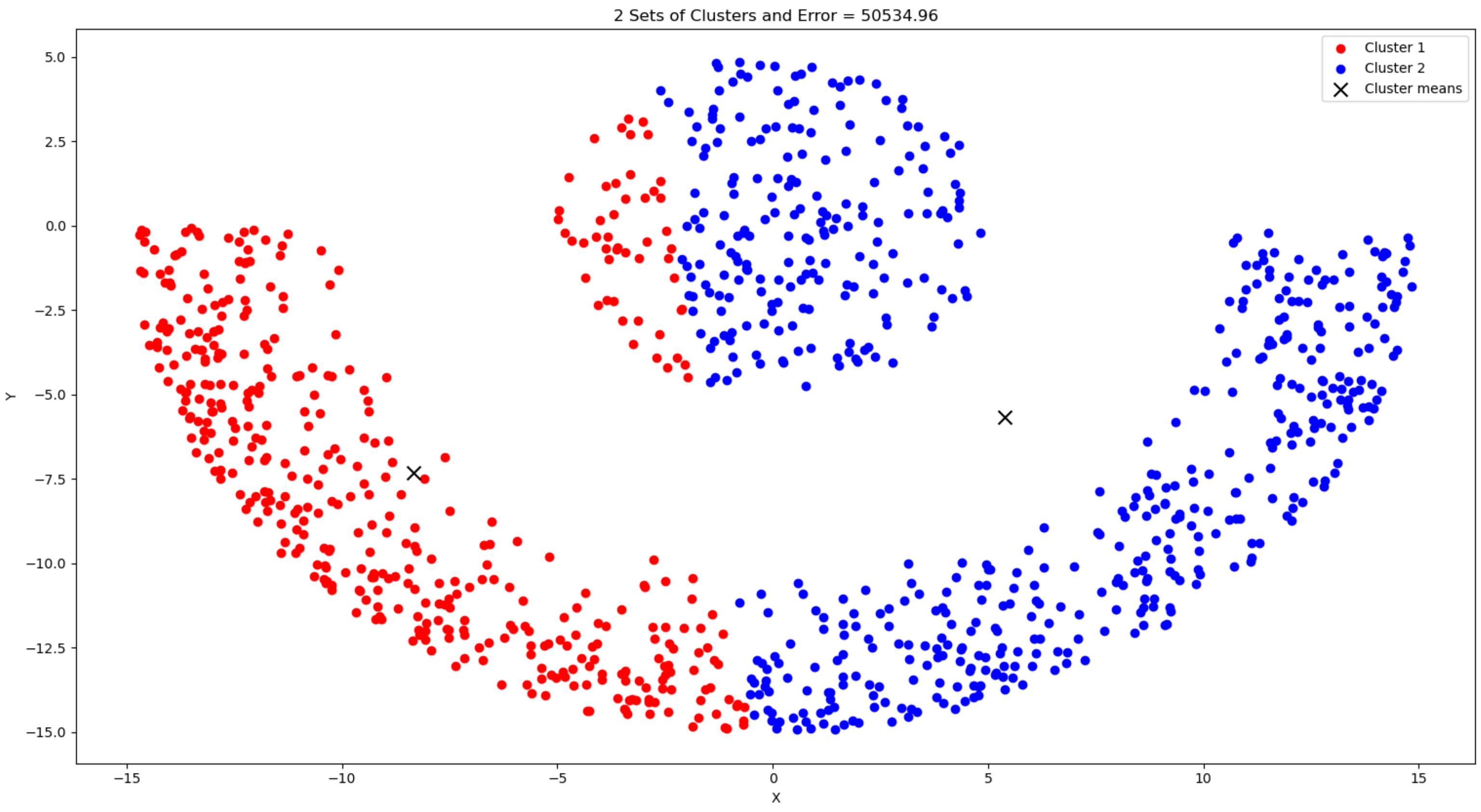

Figure 18: Random Initialization 3 clusters

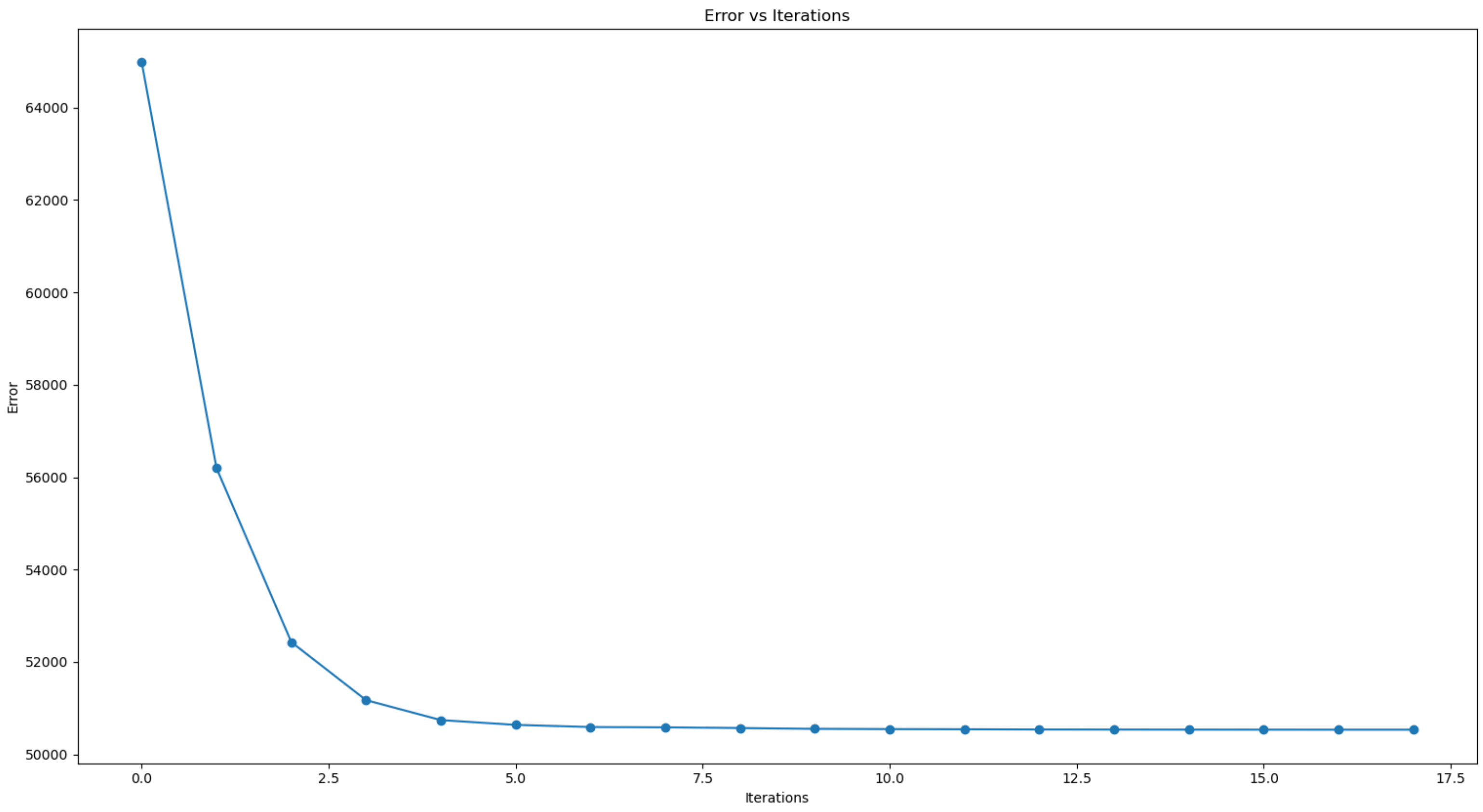

Figure 19: Random Initialization 3 error plot

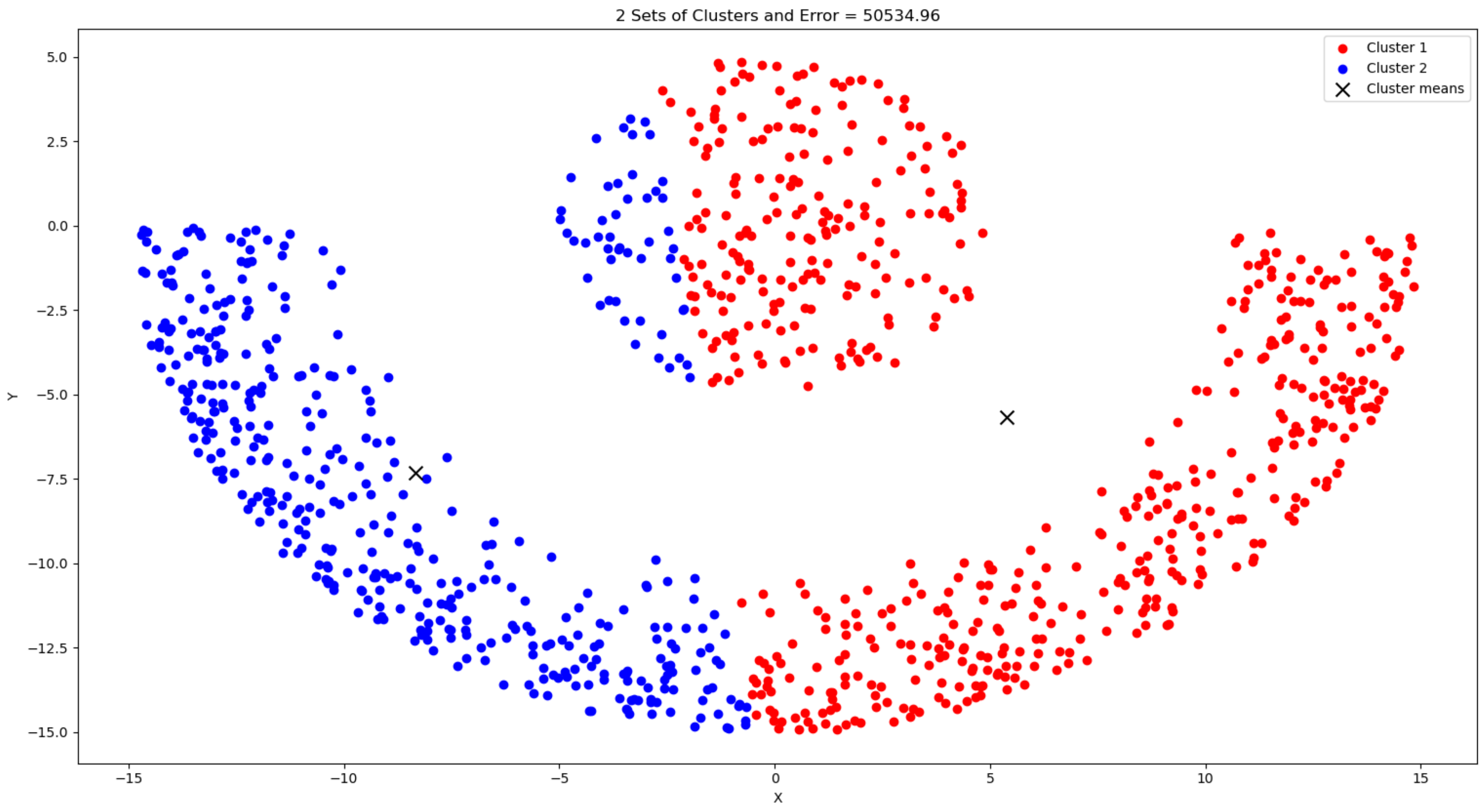

Figure 20: Random Initialization 4 clusters

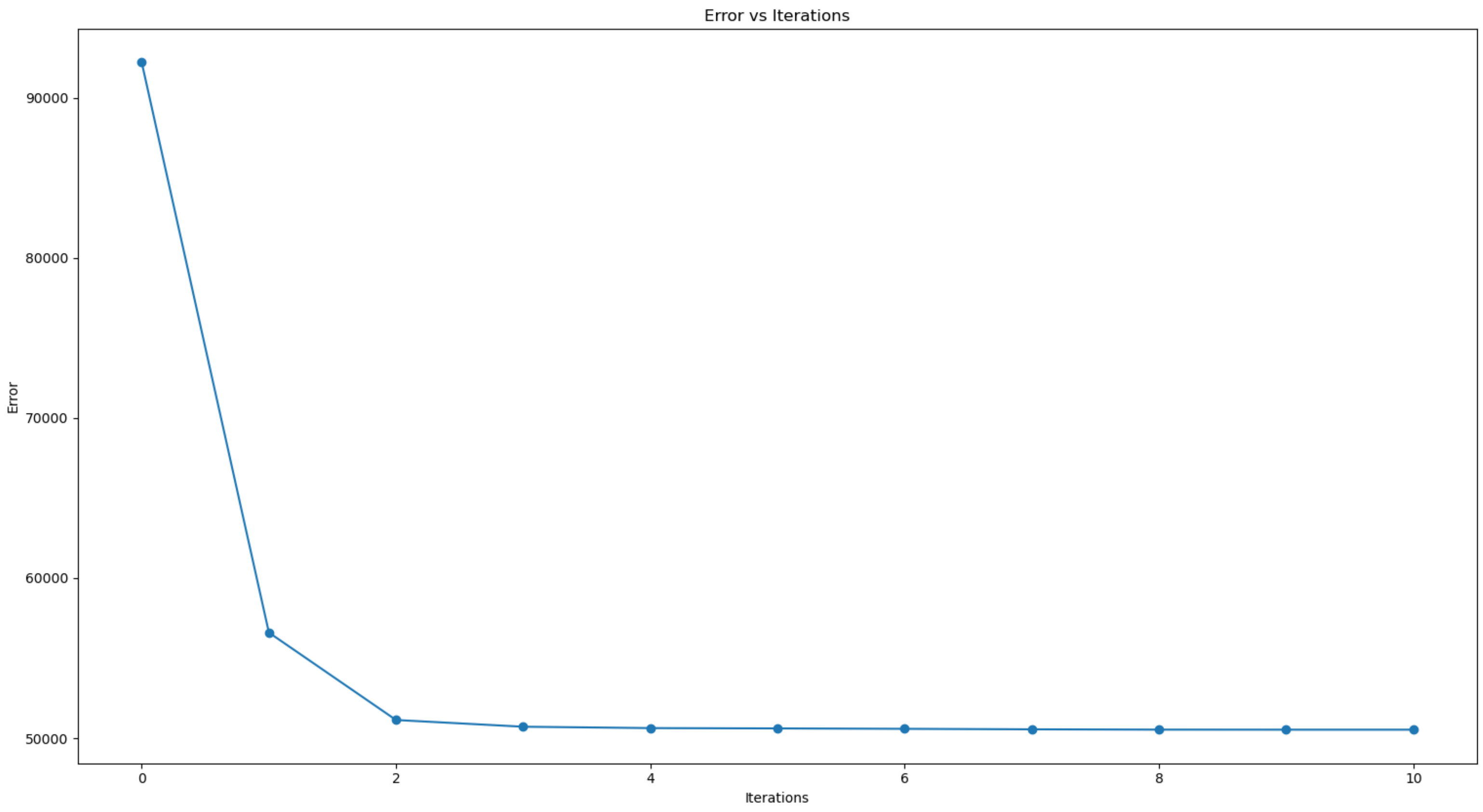

Figure 21: Random Initialization 4 error plot

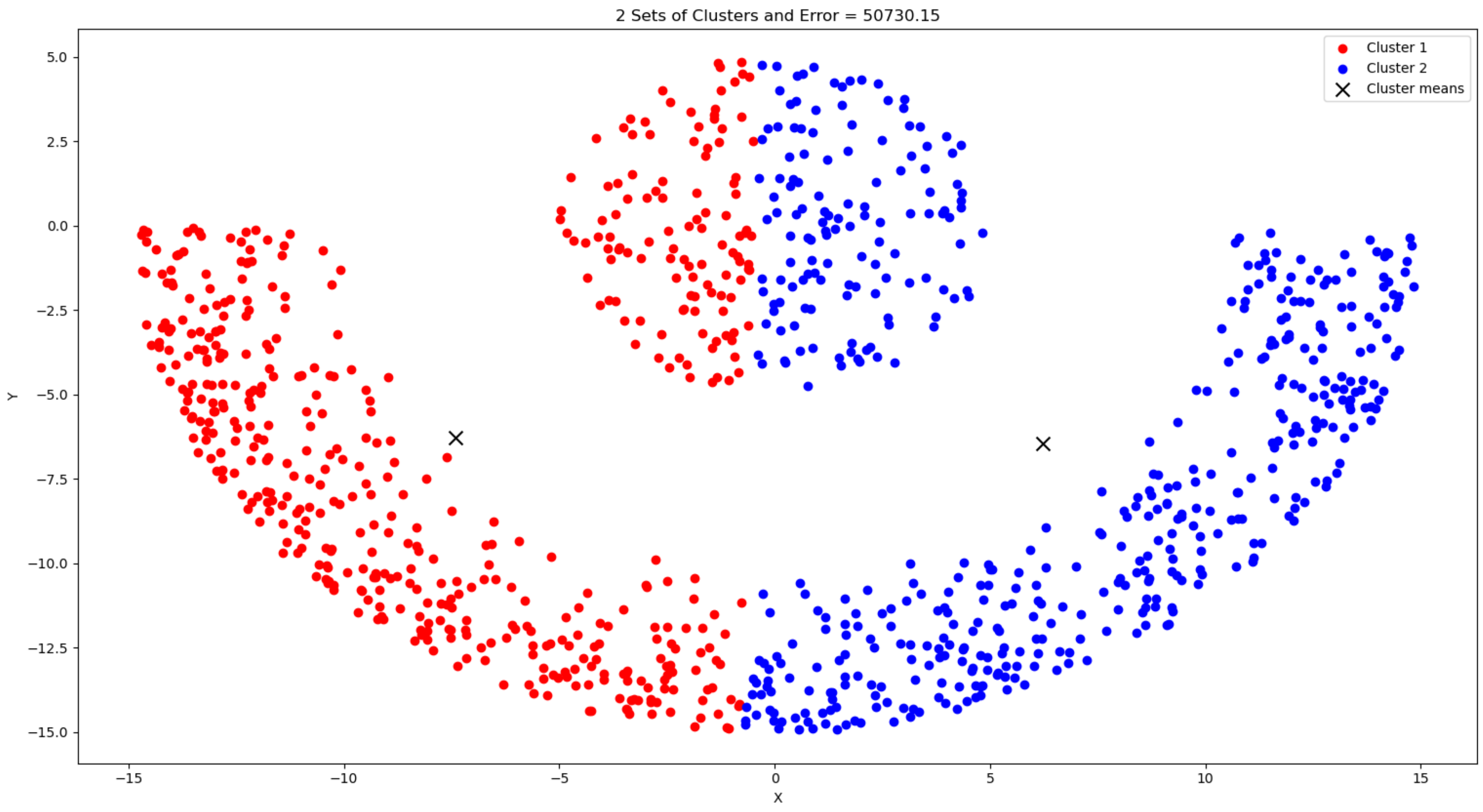

Figure 22: Random Initialization 5 clusters

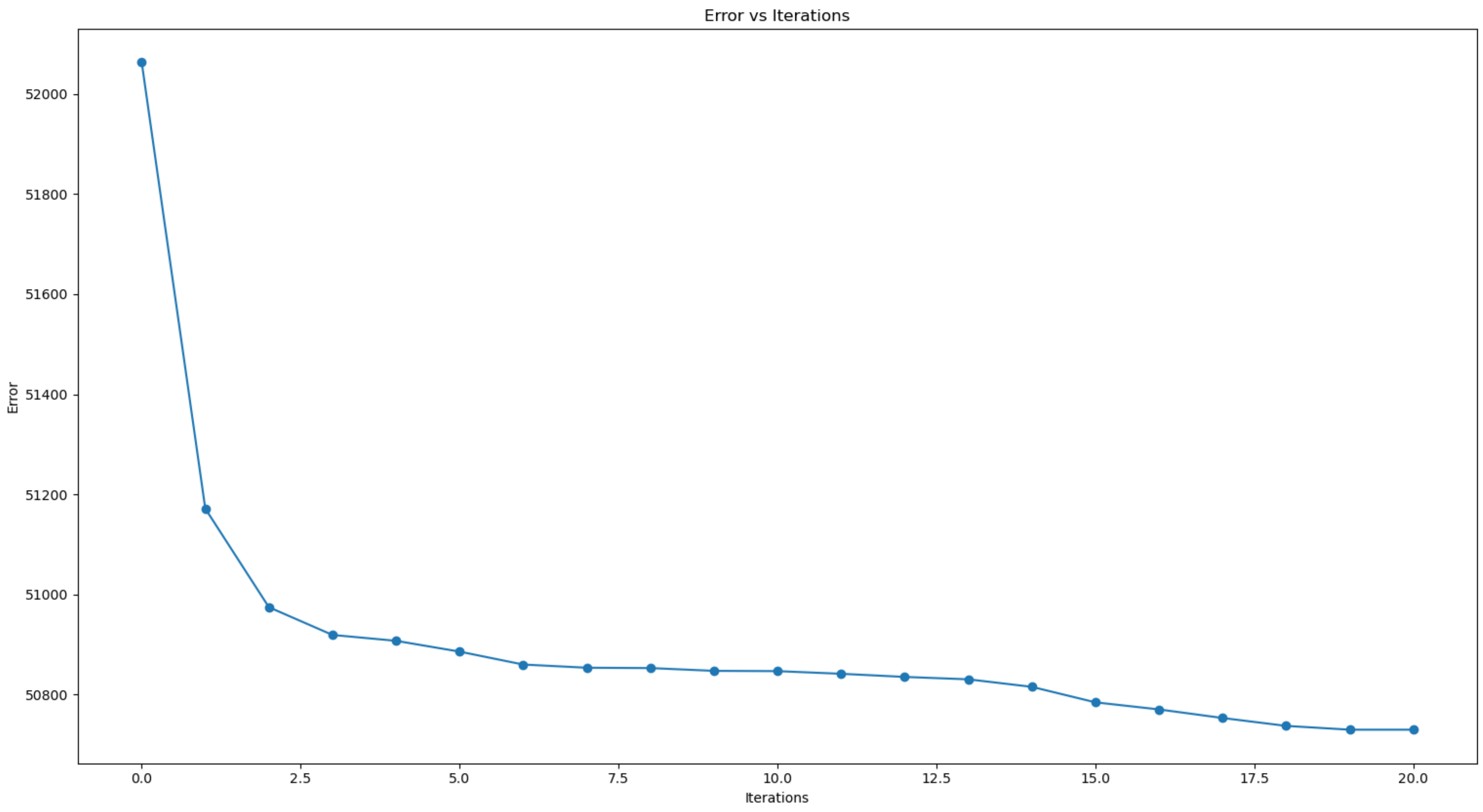

Figure 23: Random Initialization 5 error plot

Notice that the clustering depends upon the initialization. So the final cluster is not deterministic but one thing that we can confidently state is that any random initialization will converge to an optimum state. This can be proved using Llyod's algorithm. Even the error plots if you observe are not the same, they differ based on the initialization.

2.2 K-means for k = 2, 3, 4, 5

Applying the K-means algorithm on the dataset and clustering them into 2, 3, 4, and 5 clusters respectively. We are setting a random seed to fix the initialization.

For plotting the Voronoi regions, I am just showing colored cluster regions along with the cluster means. (as informed by the professor in the class)

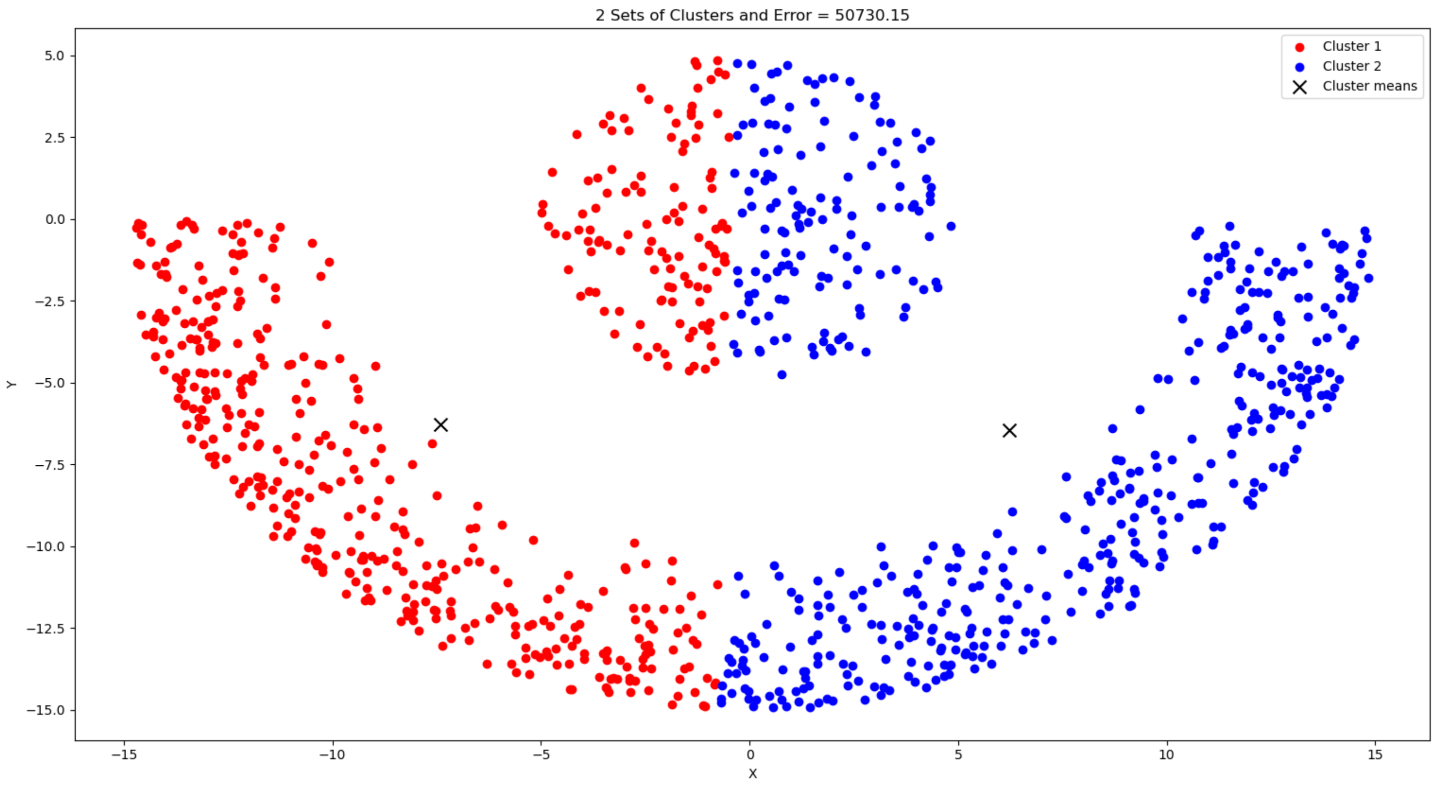

Figure 24: Clusters for K = 2

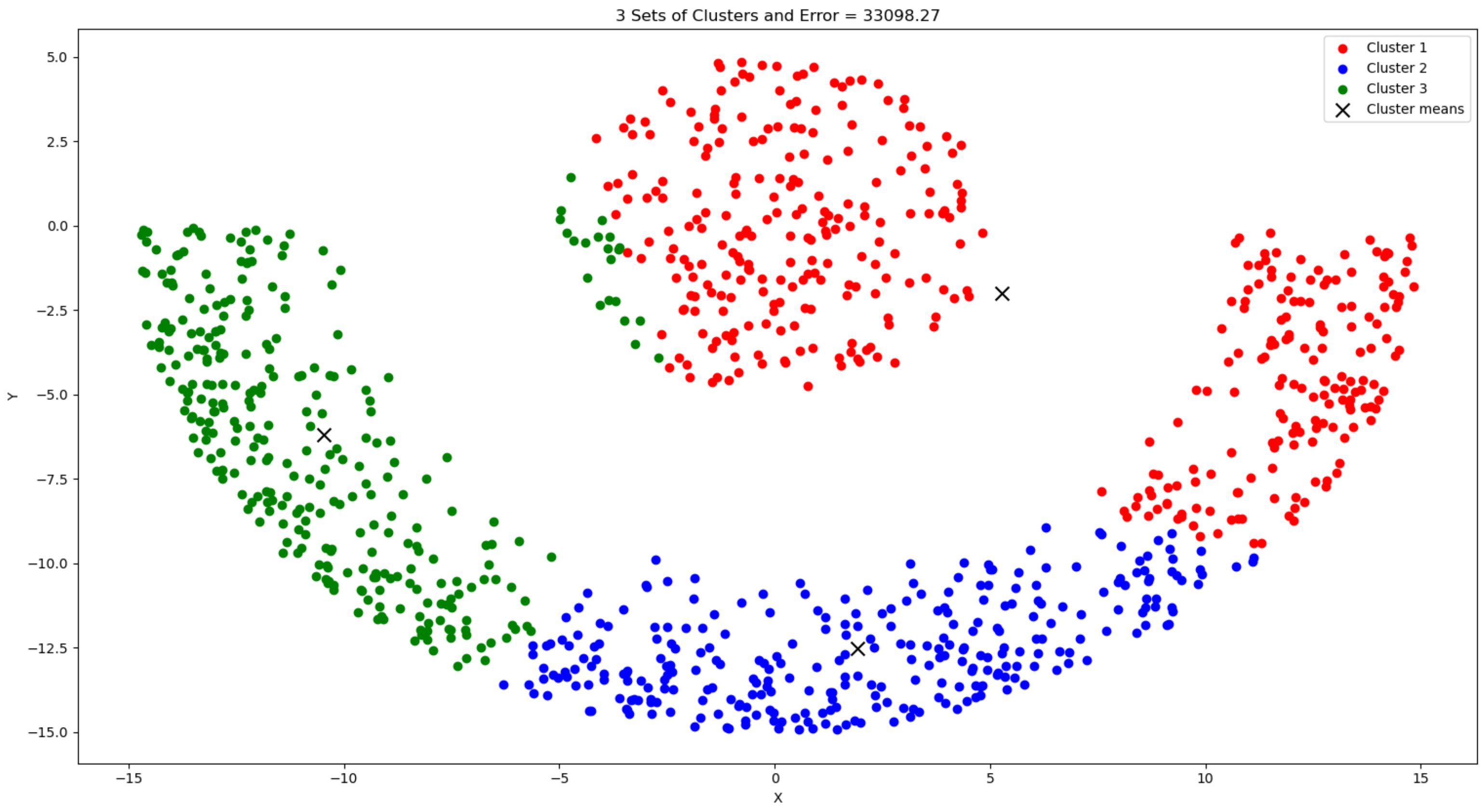

Figure 25: Clusters for K = 3

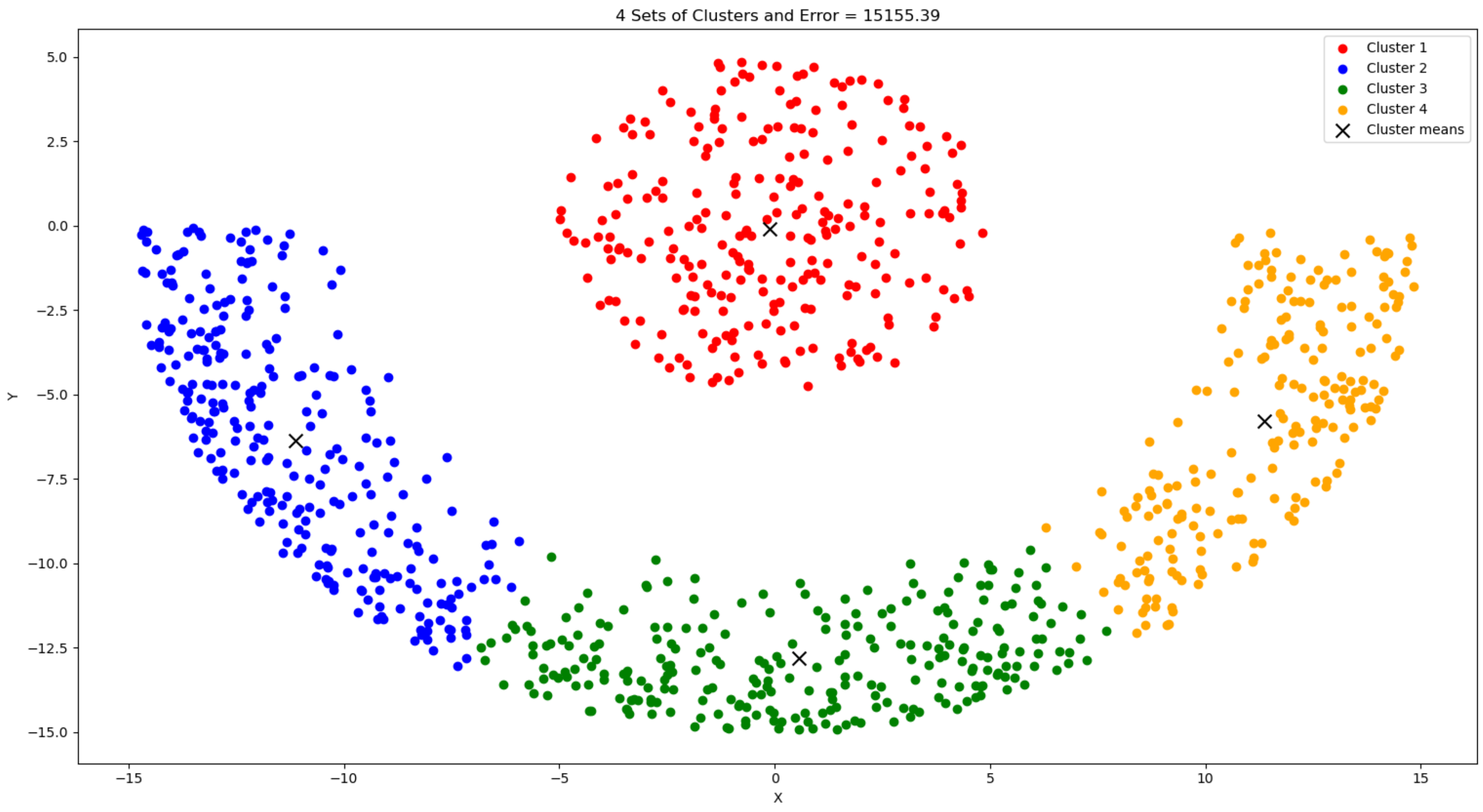

Figure 26: Clusters for K = 4

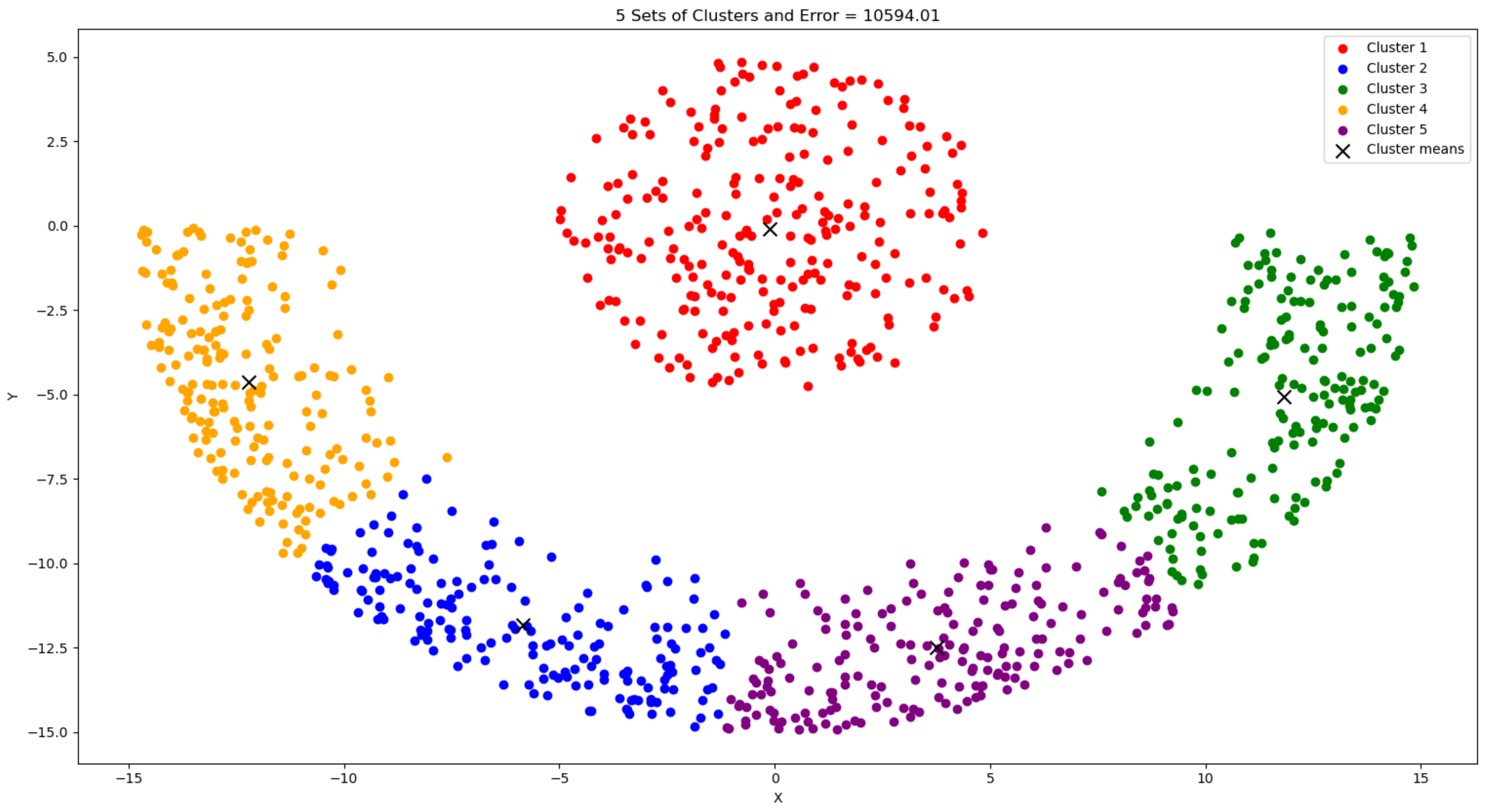

Figure 27: Clusters for K = 5

Note: Observe that the error is reducing as the K value increases. If K = 1000 the error becomes zero! But that doesn't help us, we need to find an optimum K that has less error and at the same time sensible enough for our use case. This can be done by adding a penalty function to the error which penalizes points as K increases, in this way, we can formulate the new objective function that would help us get the desired K.

2.3 Spectral Clustering

Spectral Clustering helps us to segregate data points that aren't linearly separable. The idea is similar to Kernel PCA only, we apply a kernel operation and try to perform clustering in the higher dimension. So we will be looking at different kernel functions that we have and choose the one that gives us best best results.

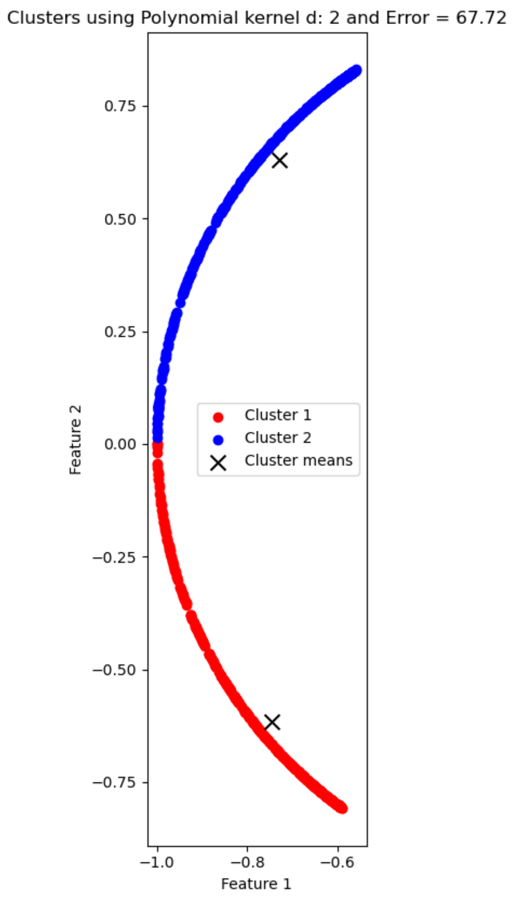

Figure 28: Spectral Clustering using Polynomial kernel with d = 2

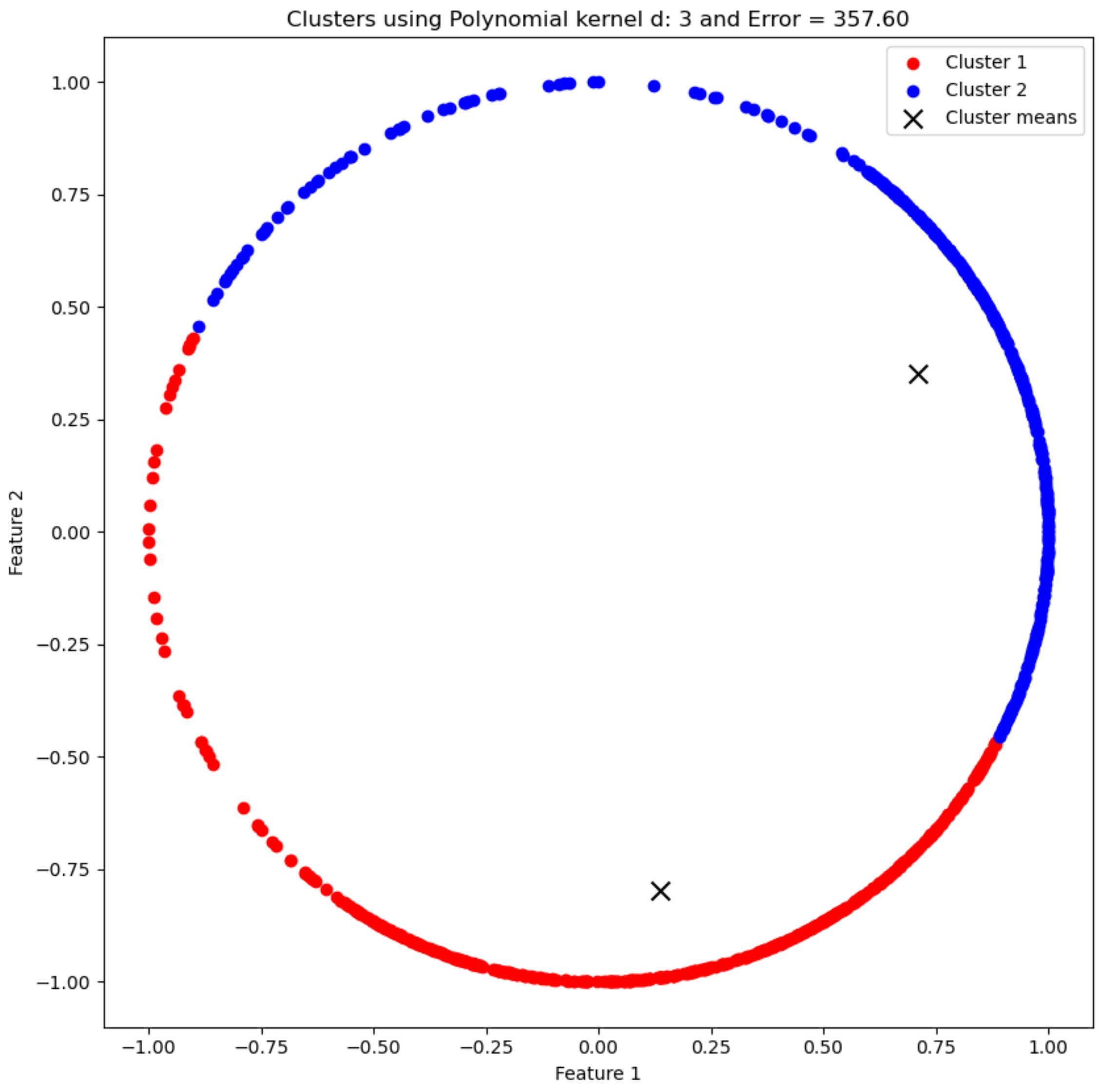

Figure 29: Spectral Clustering using Polynomial kernel with d = 3

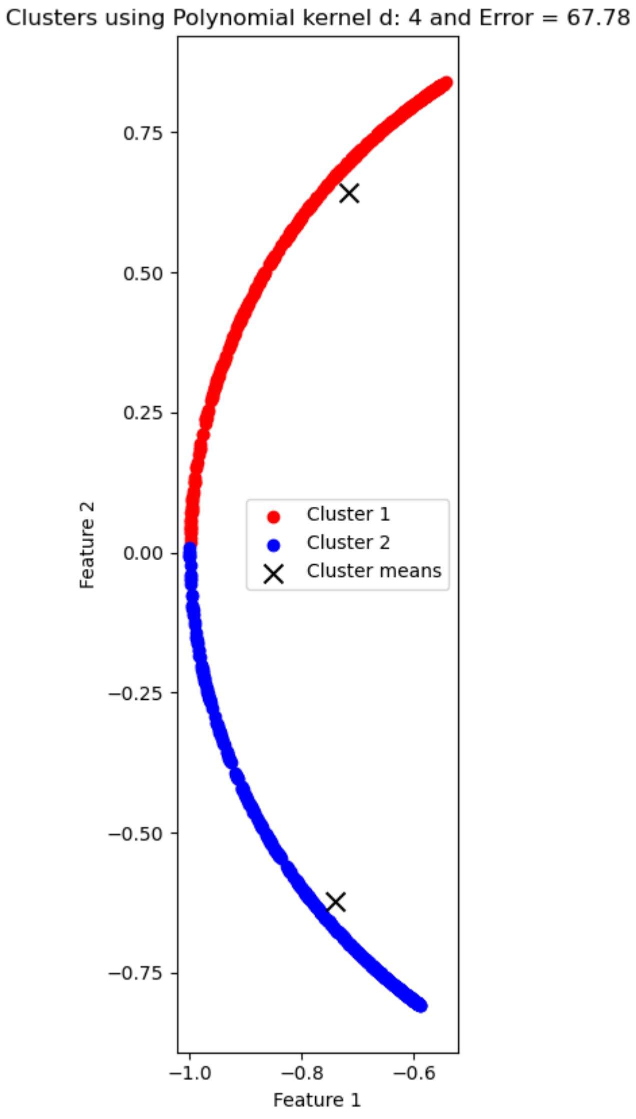

Figure 30: Spectral Clustering using Polynomial kernel with d = 4

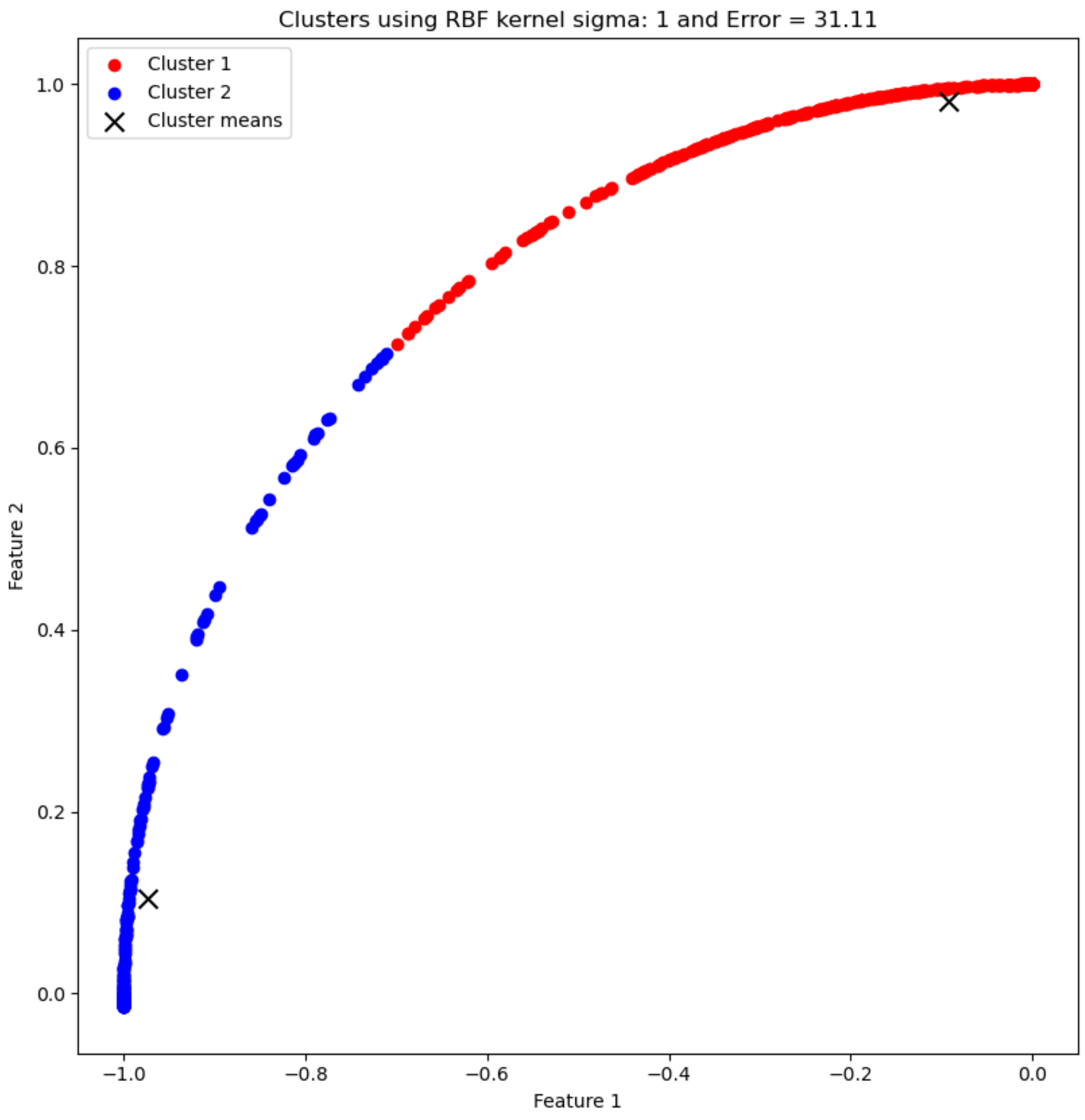

Figure 31: Spectral Clustering using RBF kernel with sigma = 1

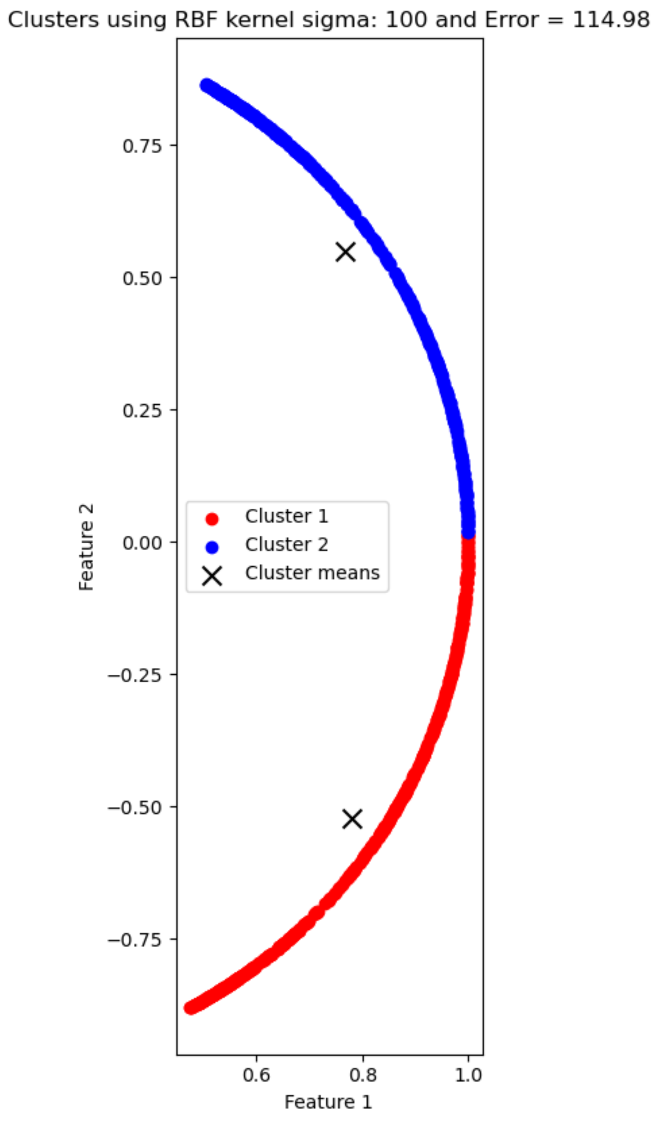

Figure 32: Spectral Clustering using RBF kernel with sigma = 100

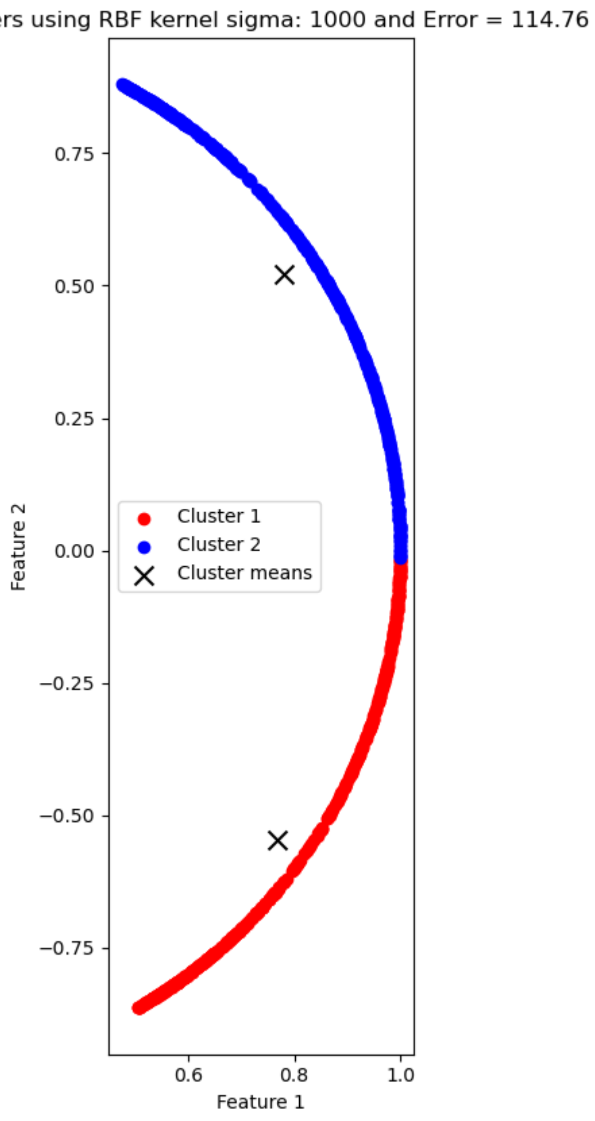

Figure 33: Spectral Clustering using RBF kernel with sigma = 1000

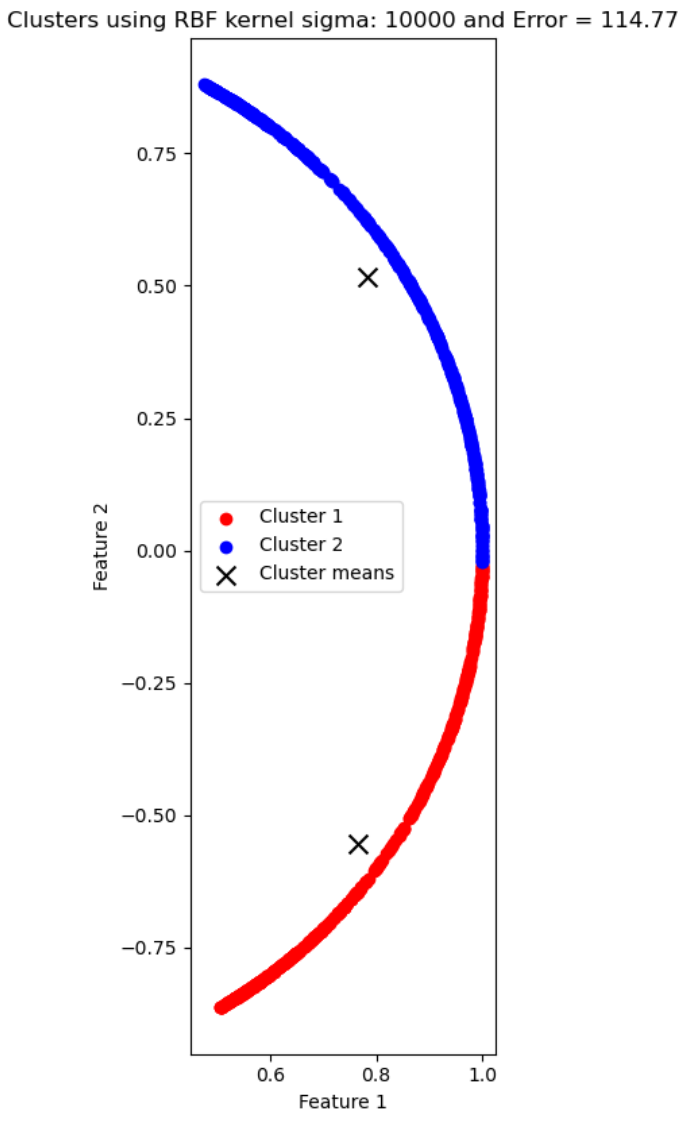

Figure 34: Spectral Clustering using RBF kernel with sigma = 10000

We can decide which kernel is the best based on the objective function value. Lesser the error, better the kernel. So as per that, RBF kernel with sigma = 1 is the best.

2.4 Alternate Clustering approach

Instead of the spectral clustering algorithm, we use an alternate clustering algorithm that was proposed in the question, setting the cluster as argmax of the elements in each of the rows. Let us observe how these results are.

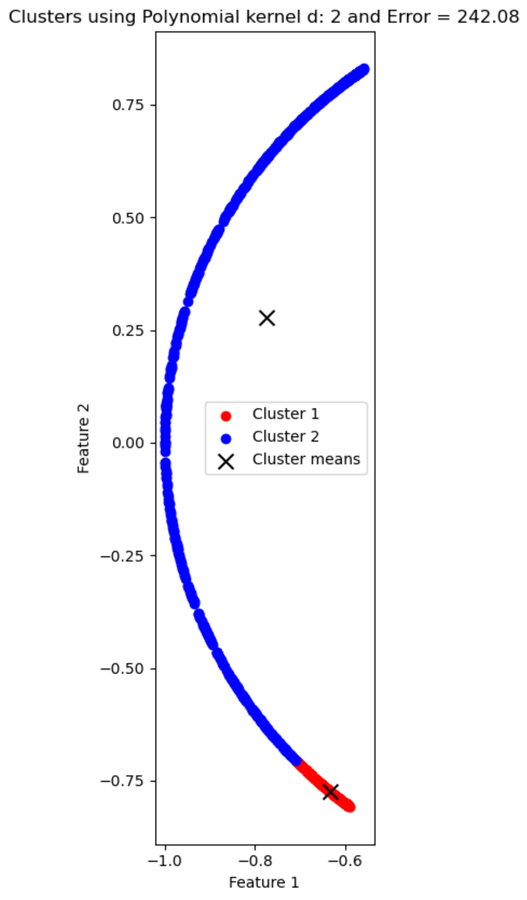

Figure 35: Alternate Clustering using Polynomial kernel with d = 2

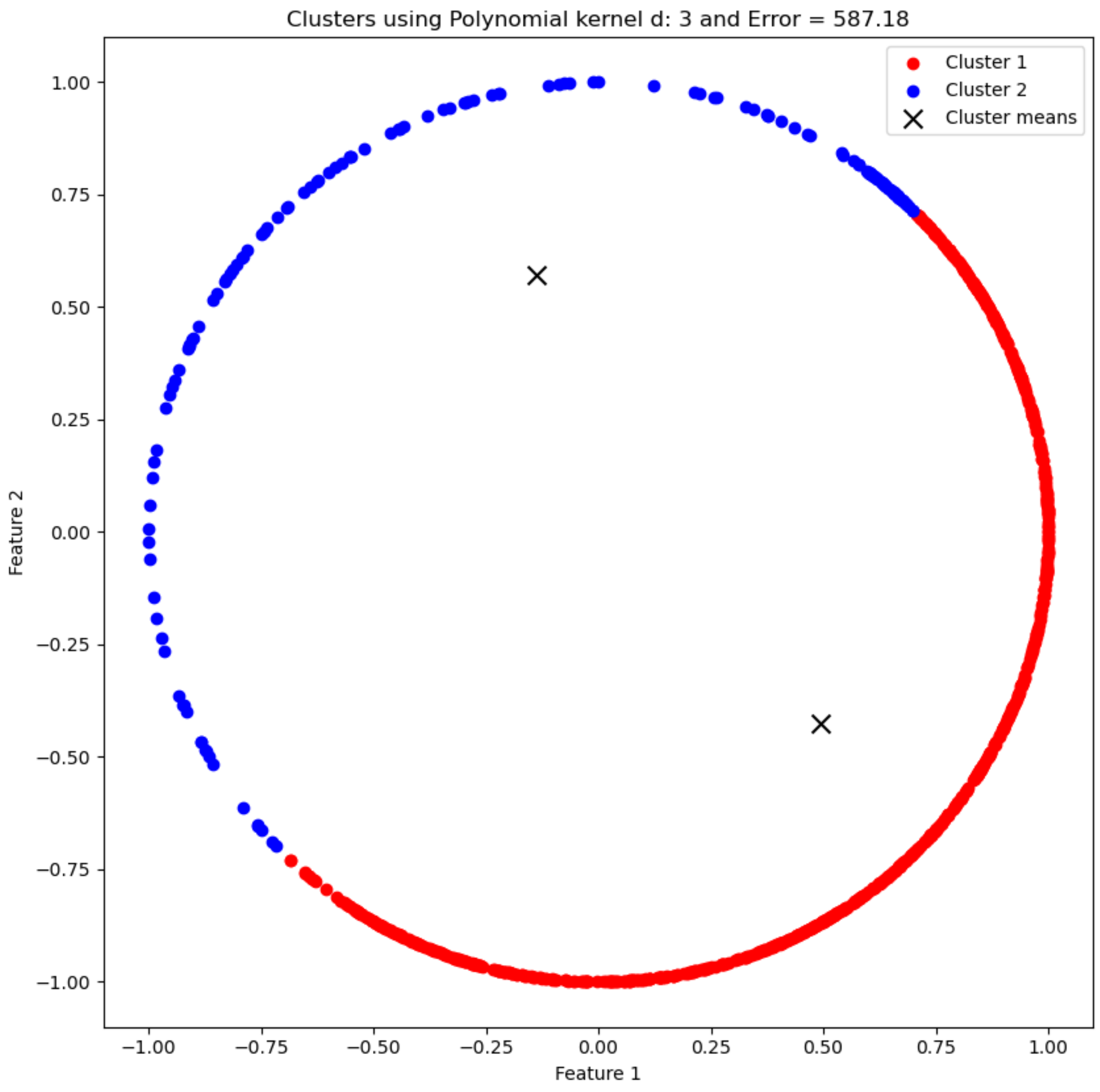

Figure 36: Alternate Clustering using Polynomial kernel with d = 3

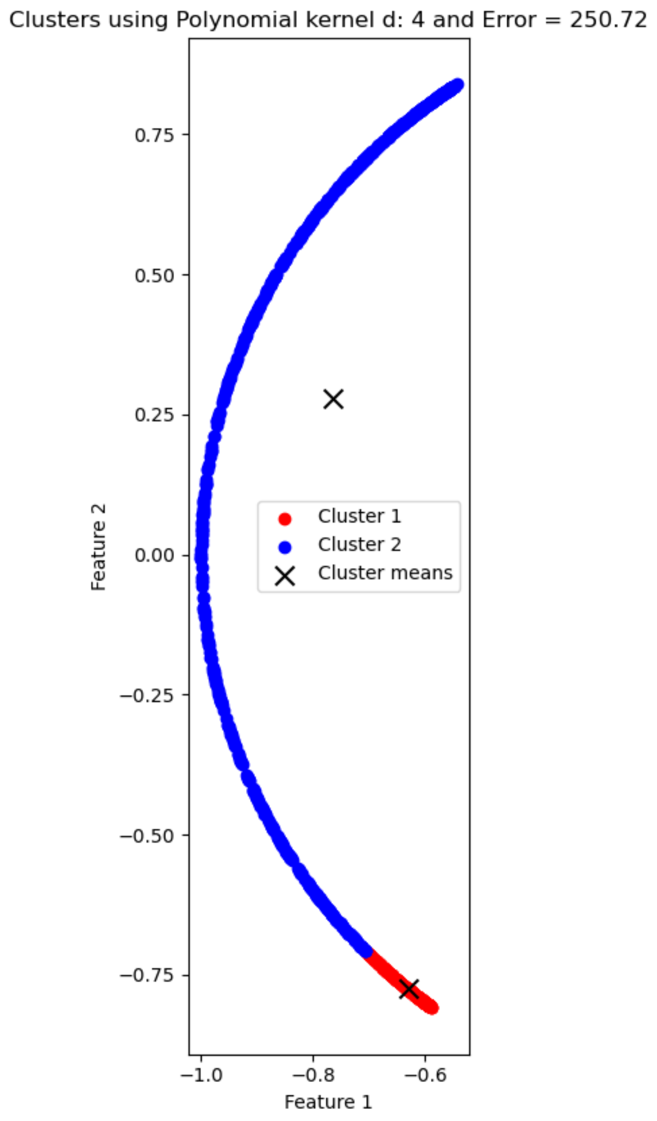

Figure 37: Alternate Clustering using Polynomial kernel with d = 4

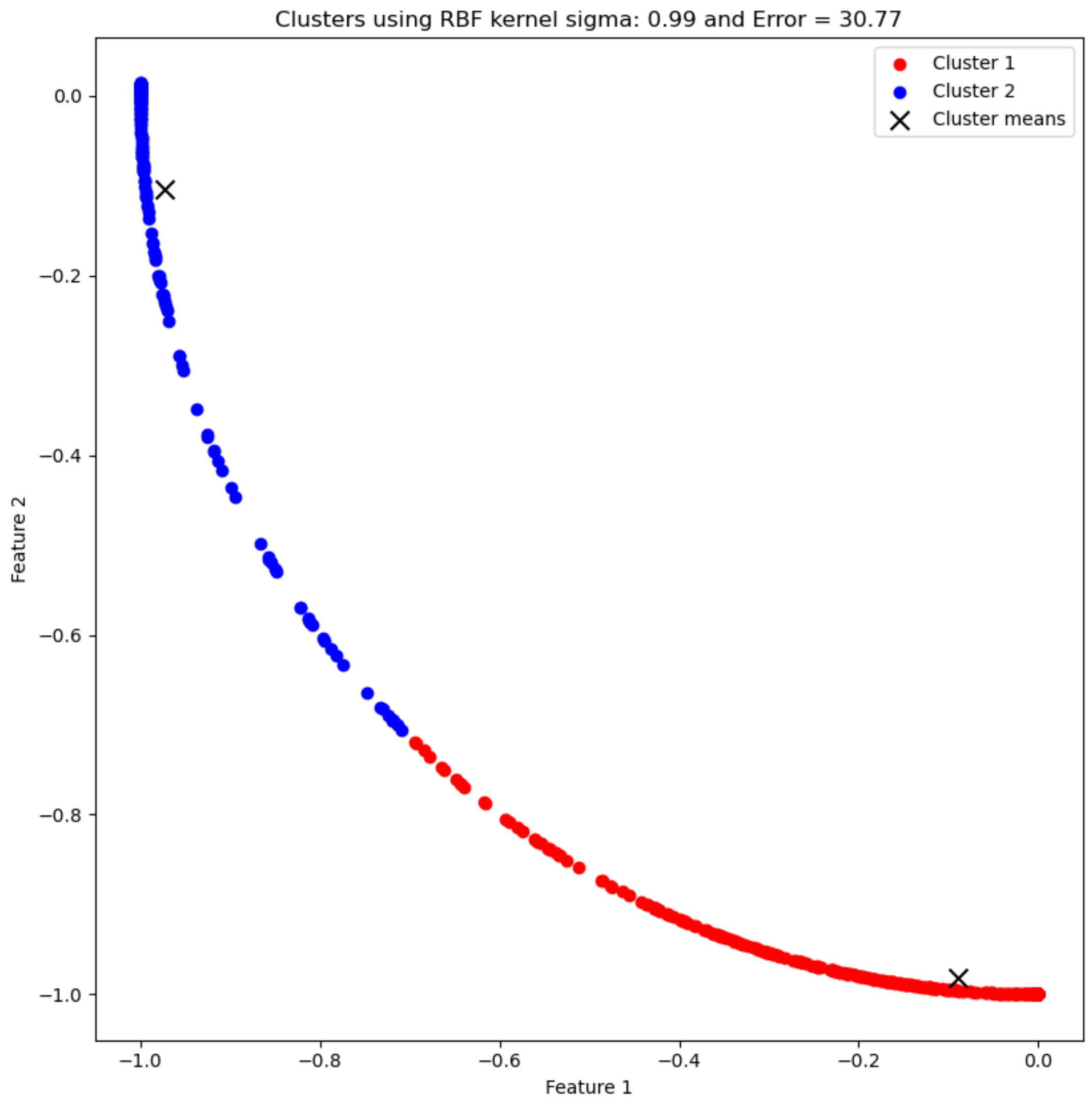

Figure 38: Alternate Clustering using RBF kernel with sigma = 0.99

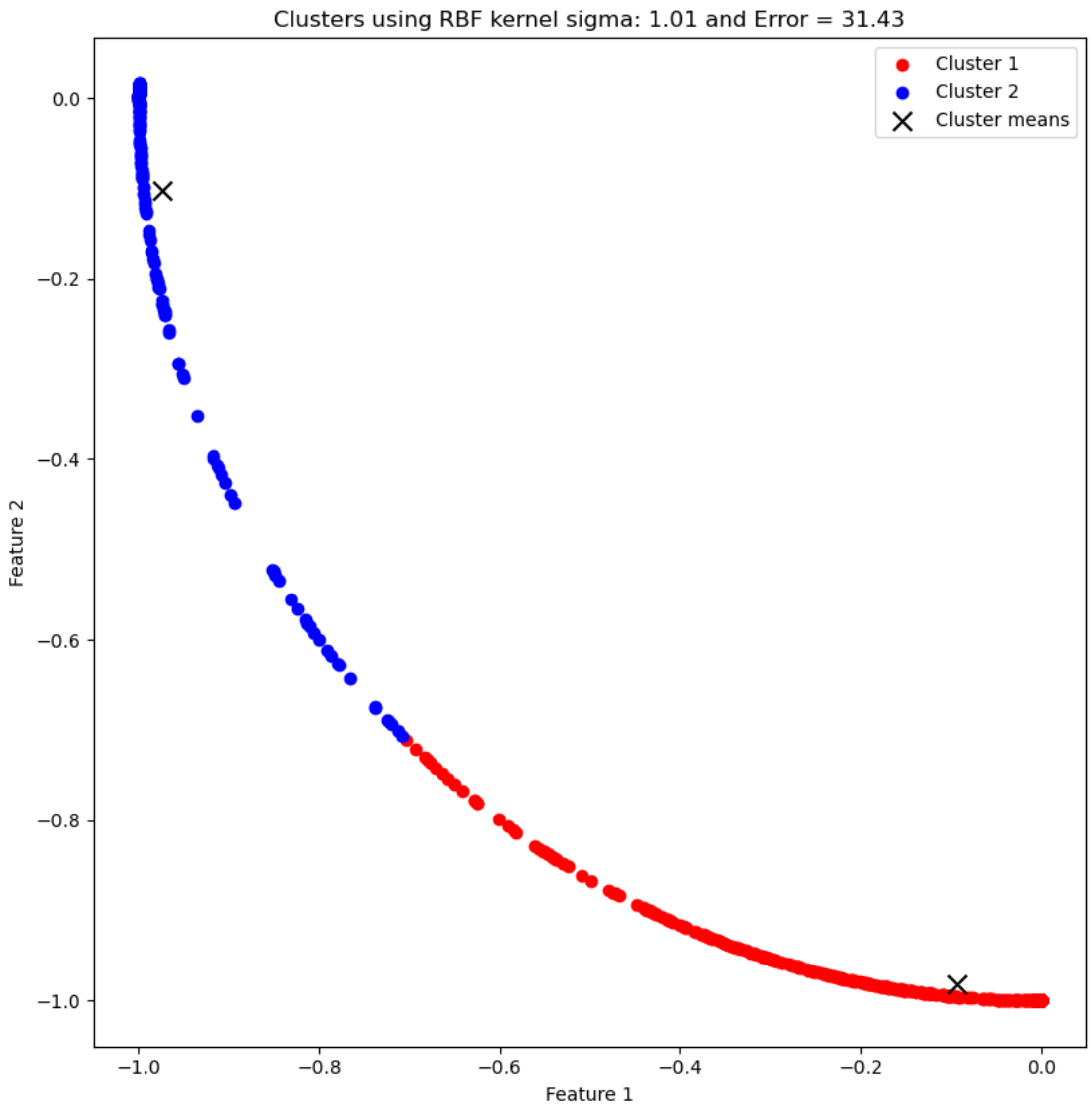

Figure 39: Alternate Clustering using RBF kernel with sigma = 1.01

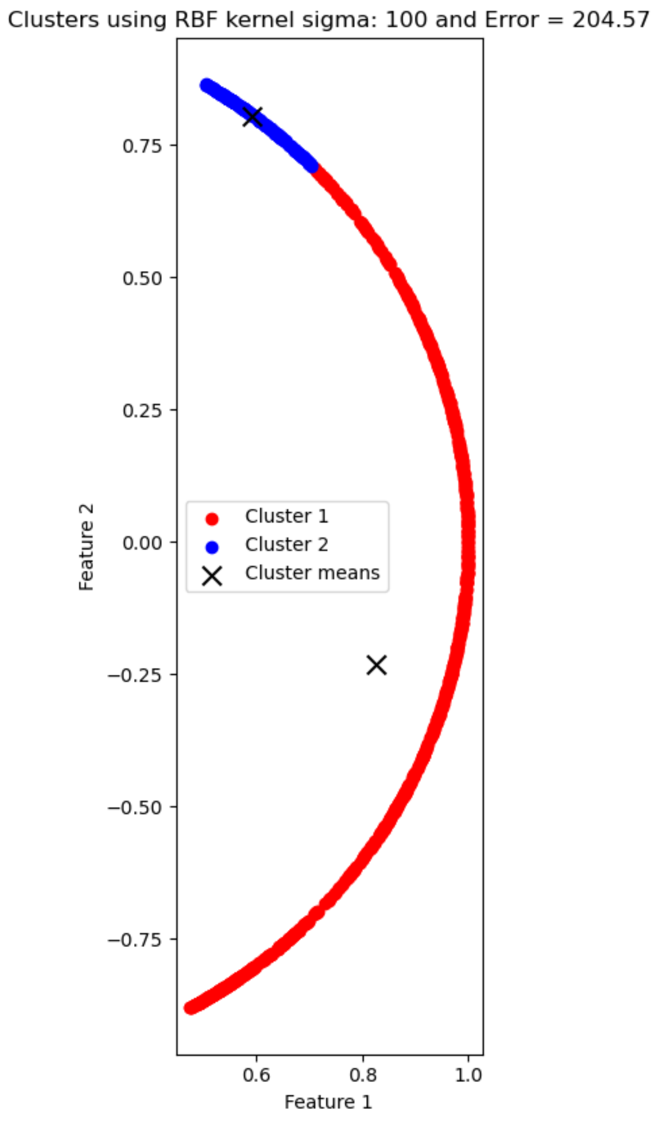

Figure 40: Alternate Clustering using RBF kernel with sigma = 100

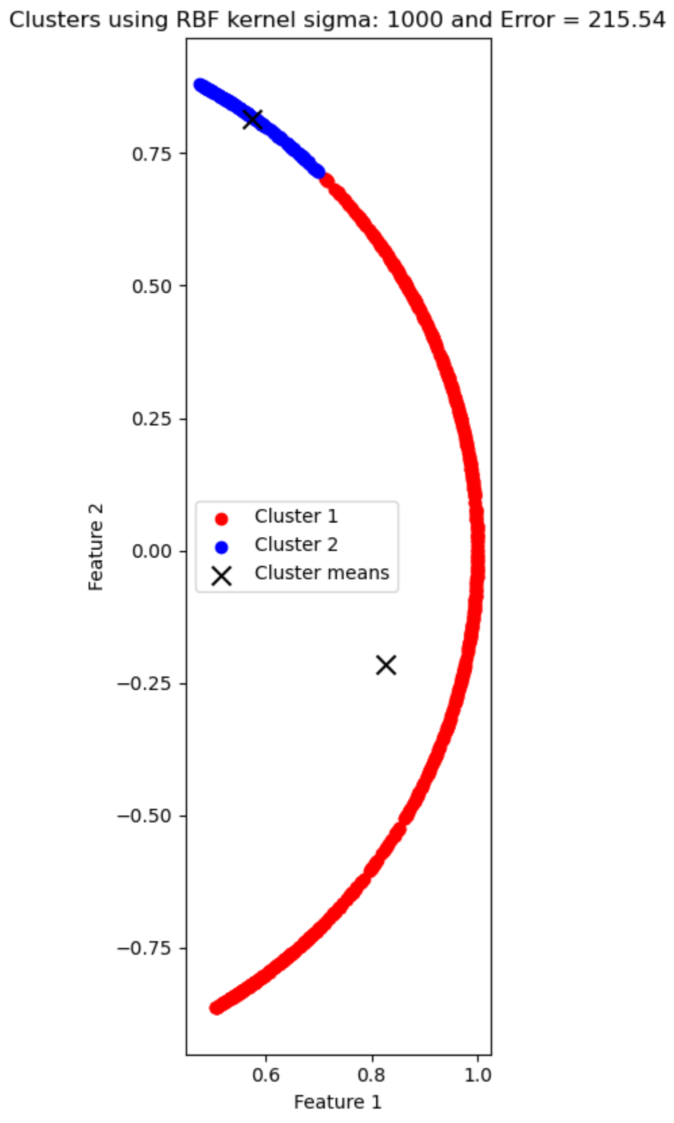

Figure 41: Alternate Clustering using RBF kernel with sigma = 1000

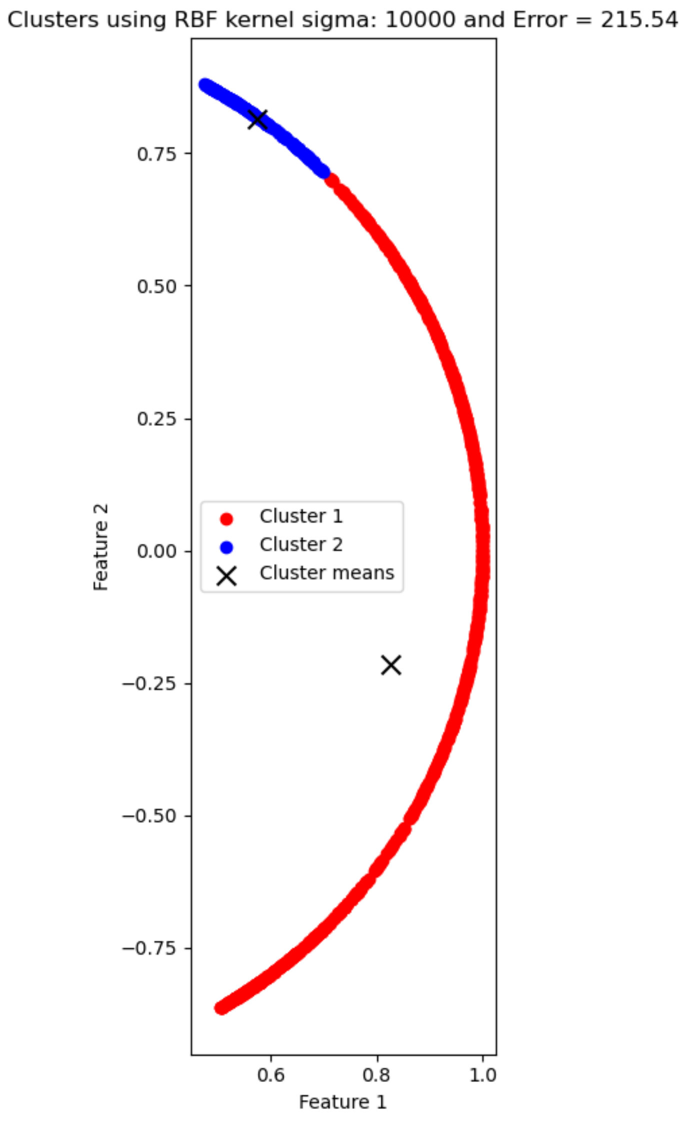

Figure 42: Alternate Clustering using RBF kernel with sigma = 10000

By comparing each of the plots with their corresponding spectral plots we can observe that the error in the alternate clustering approach is a lot higher, due to which we won't be opting for this approach.

We can also analyse it in a little more detail. So, if we observe the plots for alternate clustering mostly all the data points belong to one cluster. This happens because of the argmax condition that we have applied. The majority of the data points would belong to the cluster which is represented by the larger eigenvector.