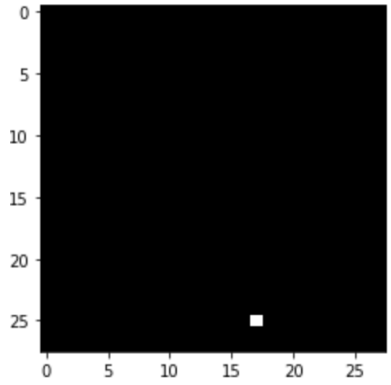
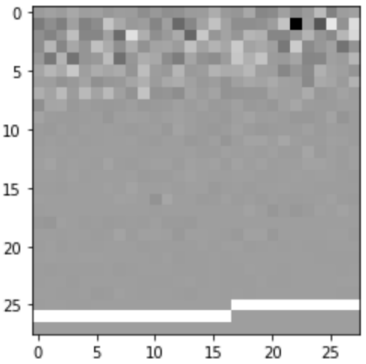
For this assignment, I have used Google Colab for a running program, I have used Python as the programming language, and I have used the Numpy library extensively as Numpy offers better computational advantages over traditional containers of Python3, so I am using Numpy array to store data and other functionality offered by numpy such as eigenvector calculation, matrix multiplication, matrix transpose function, etc. I am using matplotlib to plot our images. The following versions are used -

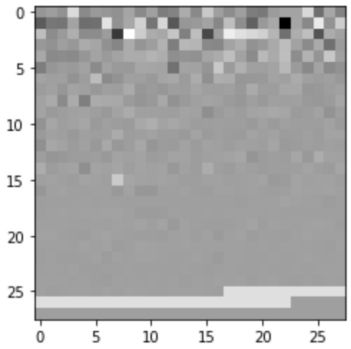
- ➔ Python Version - 3.8.10
- ➔ Numpy Version - 1.22.4

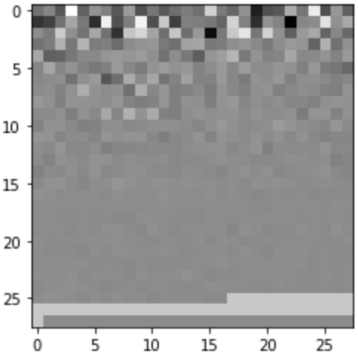
1. As given in the assignment, we are expected to run PCA and Kernel PCA algorithms on MNIST handwritten datasets. The dataset contains 70,000 images of each (28 X 28) pixel, and the dataset is split into X_Train of 60,000 images and X_Test of 60,000 images. All are labeled or mapped with Y, representing the number in the image.

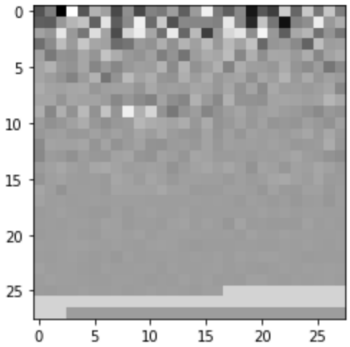
    1.1. For this bit of question, we are expected to write the PCA code. I have used the MNIST dataset present in the Keras dataset, and after data visualization, I came to the knowledge that all pixel values are between (0 - 255). First, I extracted the data set into an numpy array, then reshaped the dataset form a 3-Dimensional data set (60,000 X 28 X 28) to (60,000 X 784) and then changed the datatype from uint8 (unsigned integer 8-bit) to float64 (signed float 64-bit) as during data standardization I have to store float signed integer. I have used Sklearn preprocessing's Standardscalar to standardize the data set. After that, the feature matrix is just the transpose of the dataset, and the calculated covariance matrix with the help of numpy.cov. The cumulative variance or eigen value comes out to be around 717.011 and 95 % variance captured by top 335 dimensions alone. I have plotted the images at 40 interval while running the algorithm, here inserted some of them.

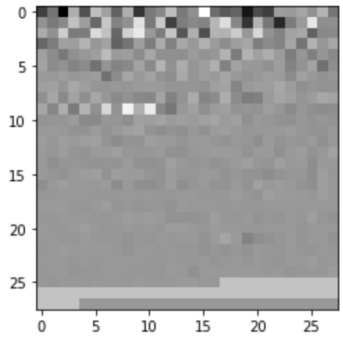

0th eigen vector

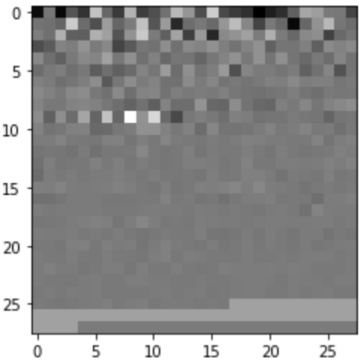

40th eigen vector

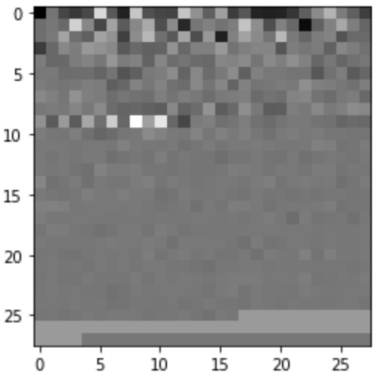

80th eigen vector

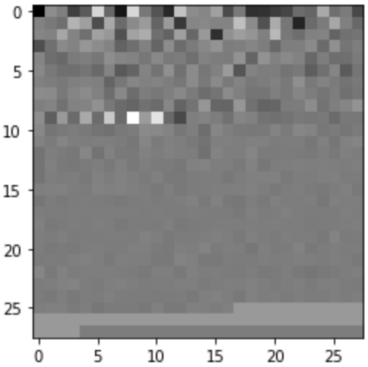

120th eigen vector


160th eigen vector

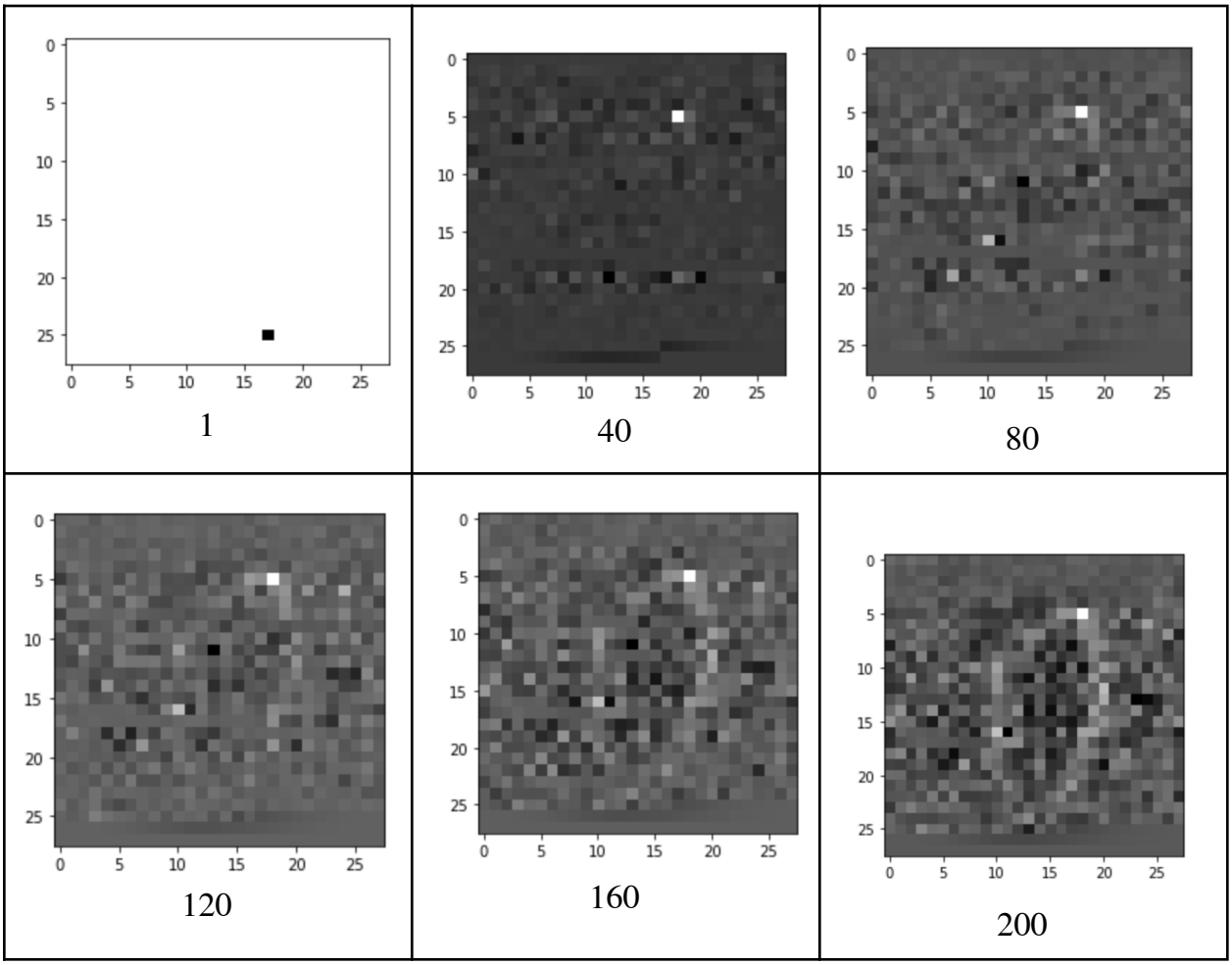

200th eigen vector

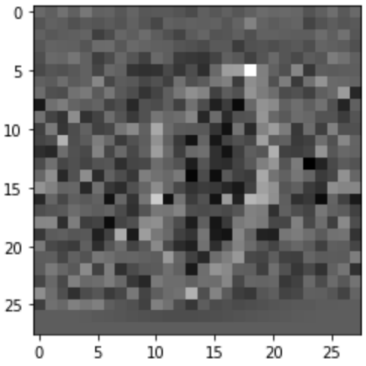

240th eigen vector


280th eigen vector

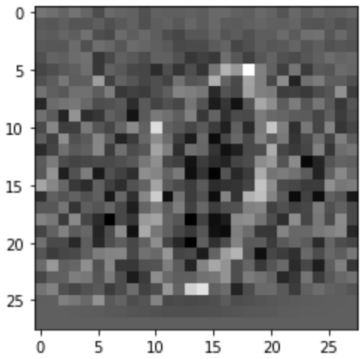

320th eigen vector

1.2 For this part of question we are expected to reconstruct the dataset using using the eigen vectors we have found out. I tried to reconstruct the 1000th data point using the matplotlib. I have projected the dataset along all the top eigen values corresponding eigen vectors and then stack them one by one on each, then reconstructing the data point. Here i have inserted the results. Number below the image represent the number of eigen vectors projection stacked.

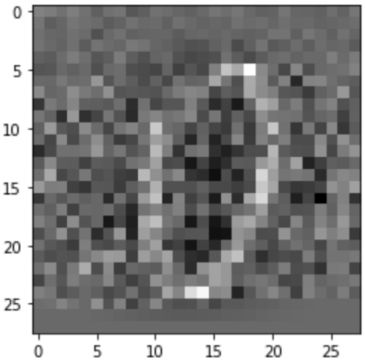




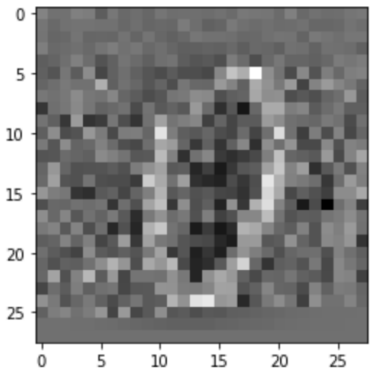
240



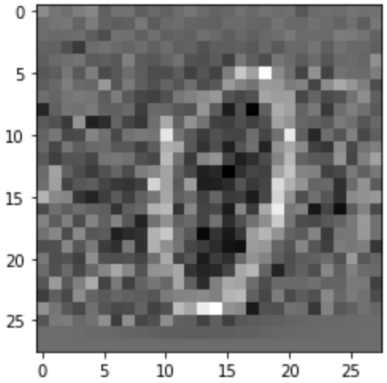
280



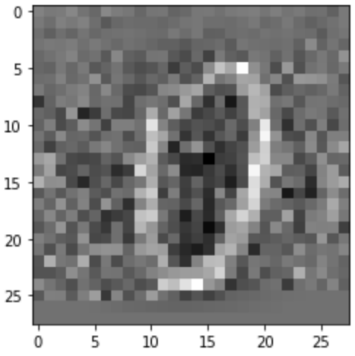
320



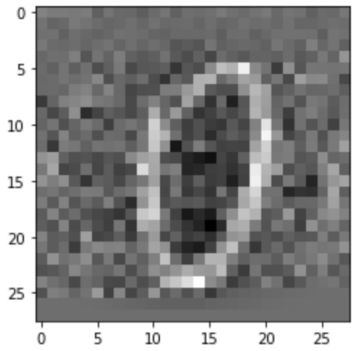
360



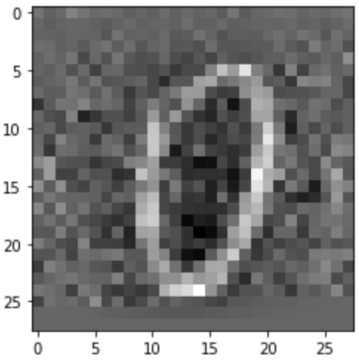
400



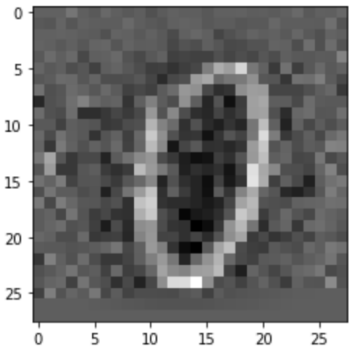
440



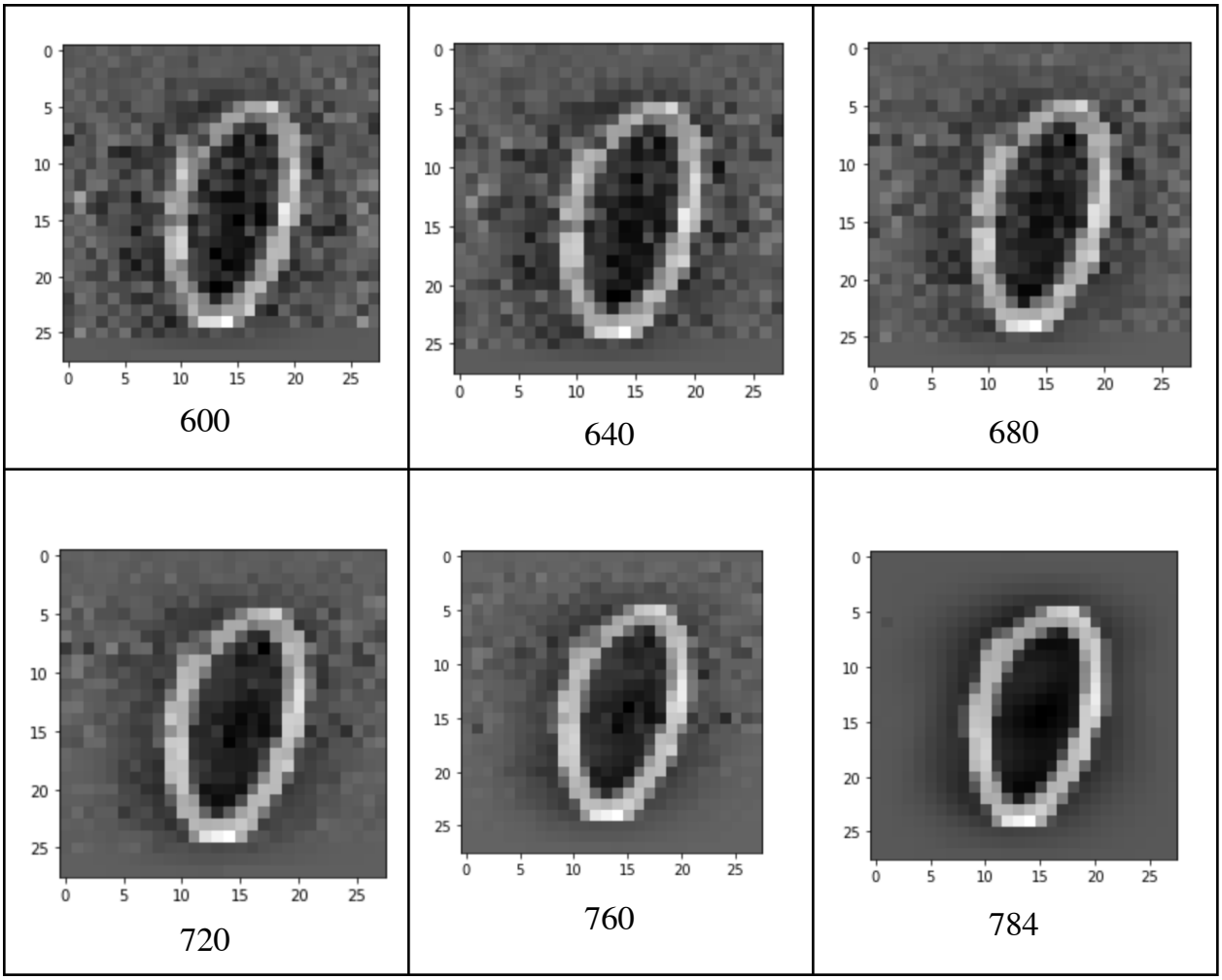
480



520



560



From the above we can see that near about top 350 principal directions we can see the 0 pattern but for better visualization we need more than 700 principal directions. Here out of 784 dimensions we are storing almost more than 700 principal directions to better vissualization because the dataset don't have any typical single pattern, there are 10 pattern for all dataset. So PCA will capture most common features present along all the data. If we have single digits data, we may require less principal directions to recognize the same digits correctly. As all digits are not alike soo we have to store more considerable number of dimensions to retain all data or in other words to recognize the digit correctly.

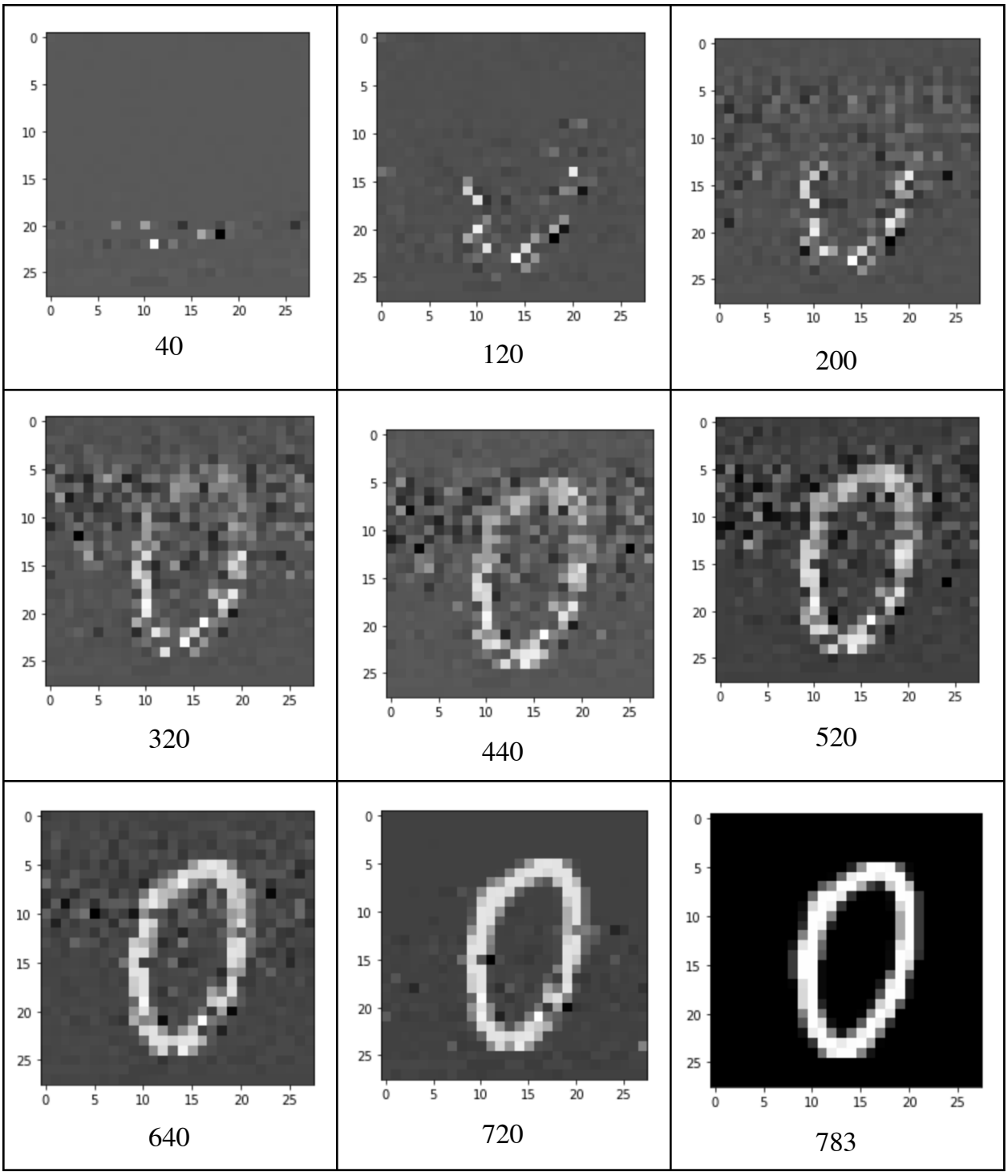
1.3 In this bit of question we are expected to write a kernel PCA algorithm where the given Kernels are as follows

a. $K(X,Y) = (1 + X^T Y)^d$,for d = {2,3,4}

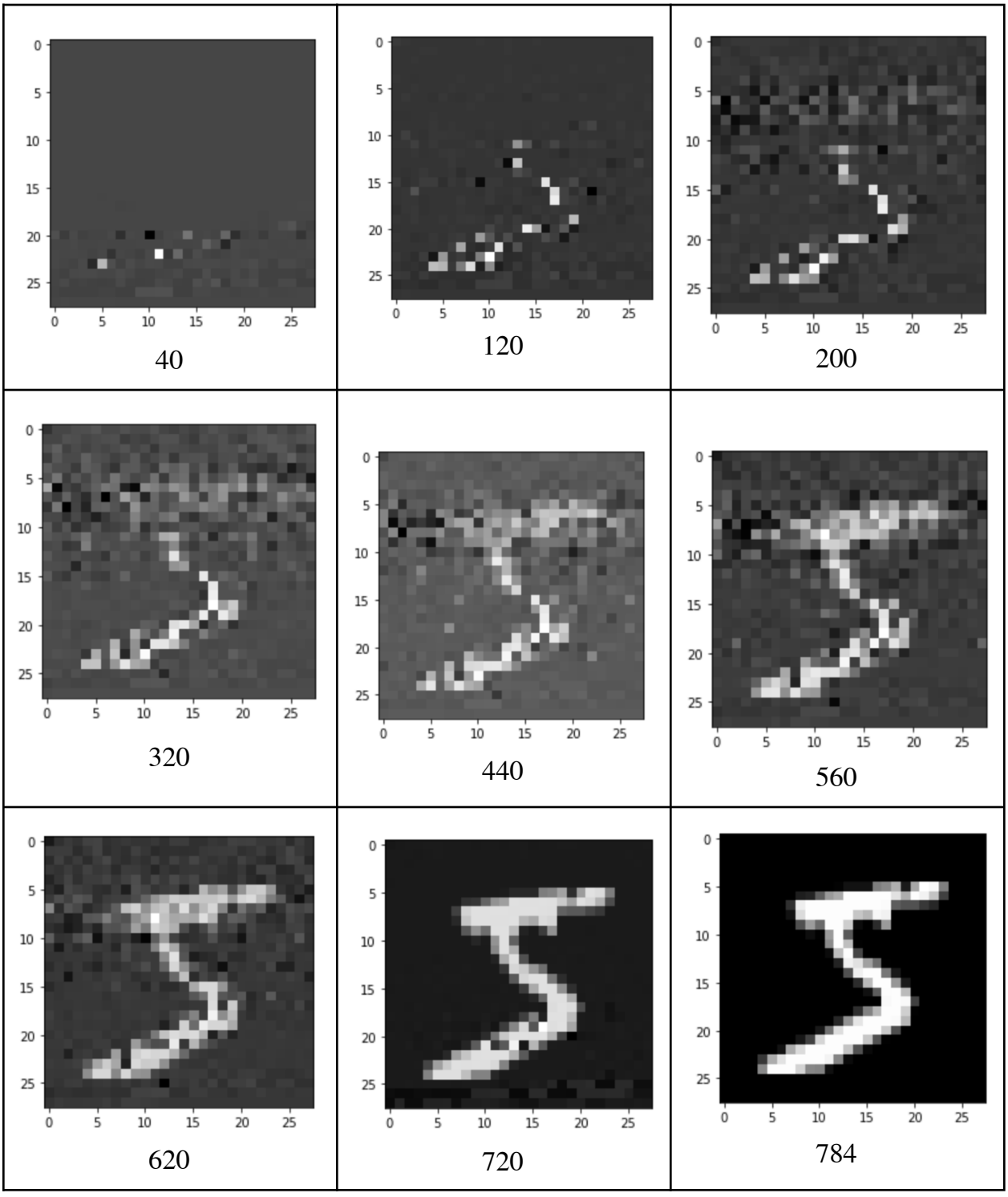
b. $K(X,Y) = exp(\frac{-(X-Y)^T(X-Y)}{2\sigma^2}$, for $\boldsymbol{\sigma}$ = {0.1,0.2 … 1.0},

and then we have to plot the top projections of each data points in the data set on to top 2 eigen values corresponding eigen vector.

Problems faced :- As earlier mentioned i am using google colab for all my computations, Standard Kernel PCA algorithm states that we have to calculate XX^T instead of $X^T X$ and then proceed for eigen values and eigen vectors calculations which is of D^3 order computations, but XX^T is of (60,000 X 60,000) dimensions, while calculating that only my colab system crashed, so there was no way that i can calculate those vectors. So instead of calculating XX^T, i calculated $X^T X$ and proceed with Kernel functions and reconstructed the datapoins and plot the projections along top 2 eigen values corresponding eigen vector data recreation. Here i have attached some reconstructed data from kernel of ($X^T X$) , Kernel function $K(X,Y) = (1 + X^T Y)^d$. Number below the image represent the number of eigen vectors projection stacked.

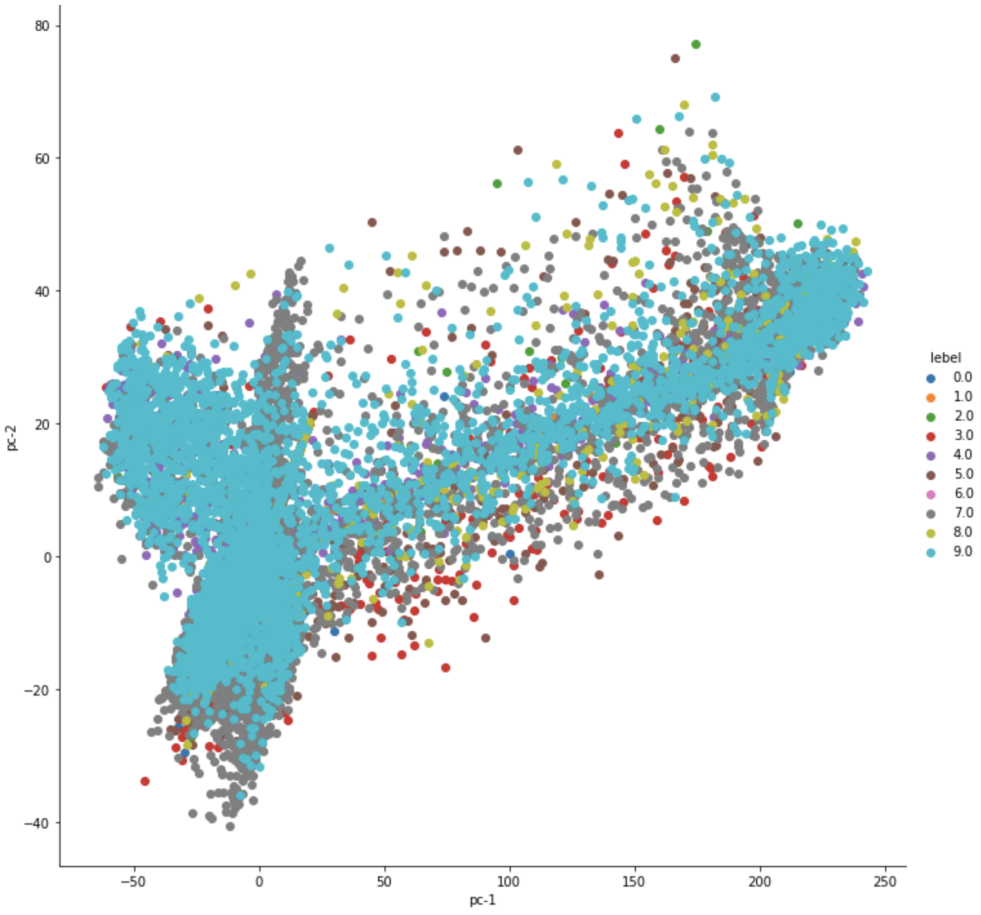

40
120
200
320
440
520
640
720
783

Another datapoint recreations -



Here we can see after applying the kernel function, we can retain more data with less number of principal directions mean we can visualize our data point with less number of principal directions,more variance captured with less principal directions.

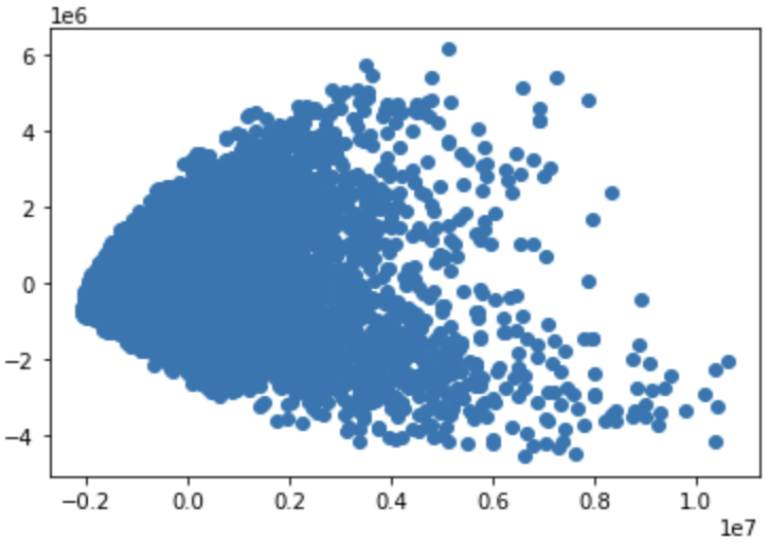
I have plotted the data points projections of each data point along the top 2 eigenvalues corresponding eigenvectors and attached them in the following.



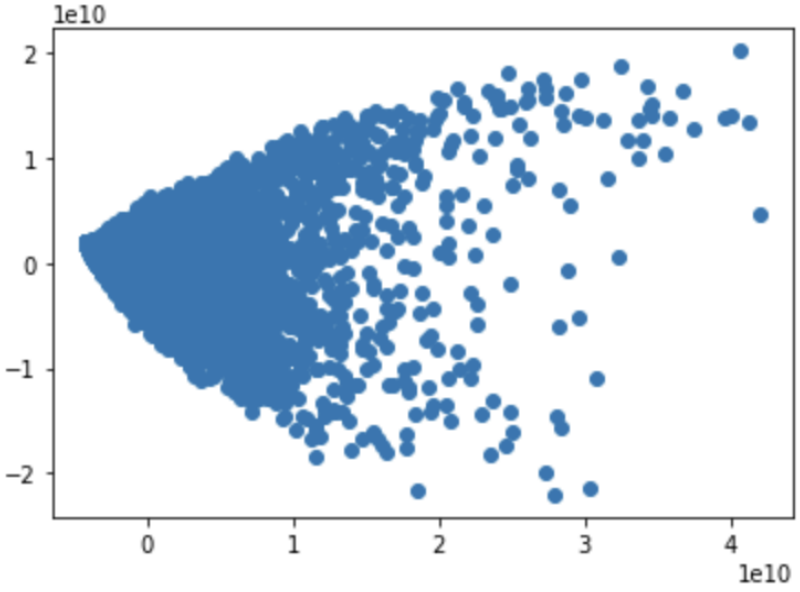
Again with fewer data points, I have tried to calculate the kernel function transformation of the covariance matrix. Here I have used the whole test data set initially, which contains 10,000 data points. After calculating XX^T the dimensions become (10,000 X 10,000), and then i have followed the standard centering procedure of kernel functions and plotted the data points projections of each data point along top 2 eigenvalues corresponding eigenvectors and attached in following.

a. Kernel function $K(X, Y) = (1 + X^T Y)^d$

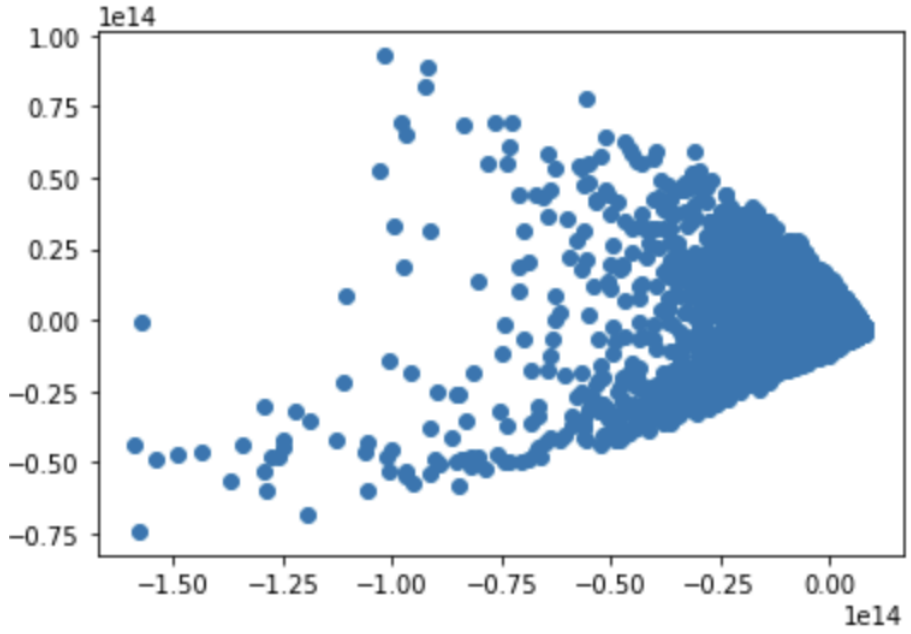
for d = 2, plot result are coming as follows (pc-1 on X-axis and pc-2 on Y-axis) -



for d = 3, plot result are coming as follows (pc-1 on X-axis and pc-2 on Y-axis) -



for d = 4, plot result are coming as follows (pc-1 on X-axis and pc-2 on Y-axis) -

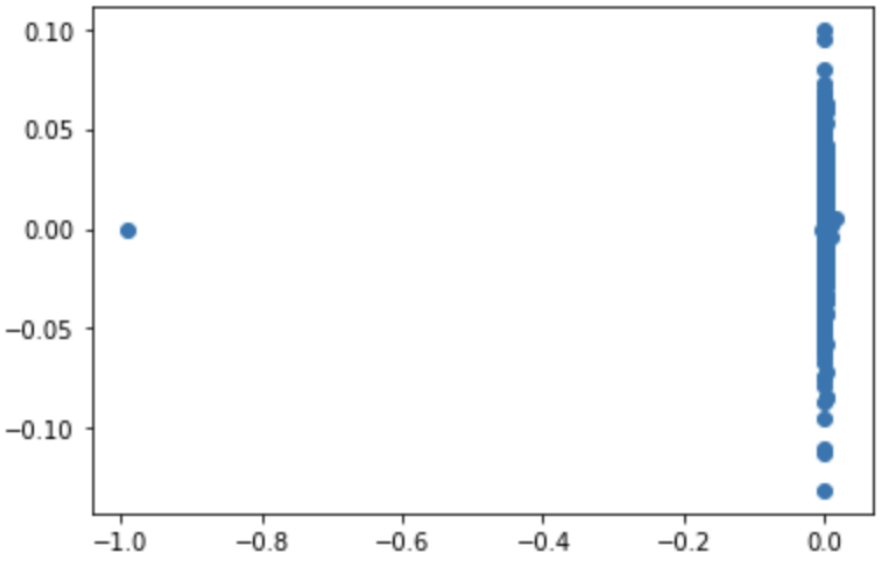


Problems faced - for each computation of eigen vector and corresponding eigenvalues we are almost taking more than 20min each time as the computations are in order of D^3.

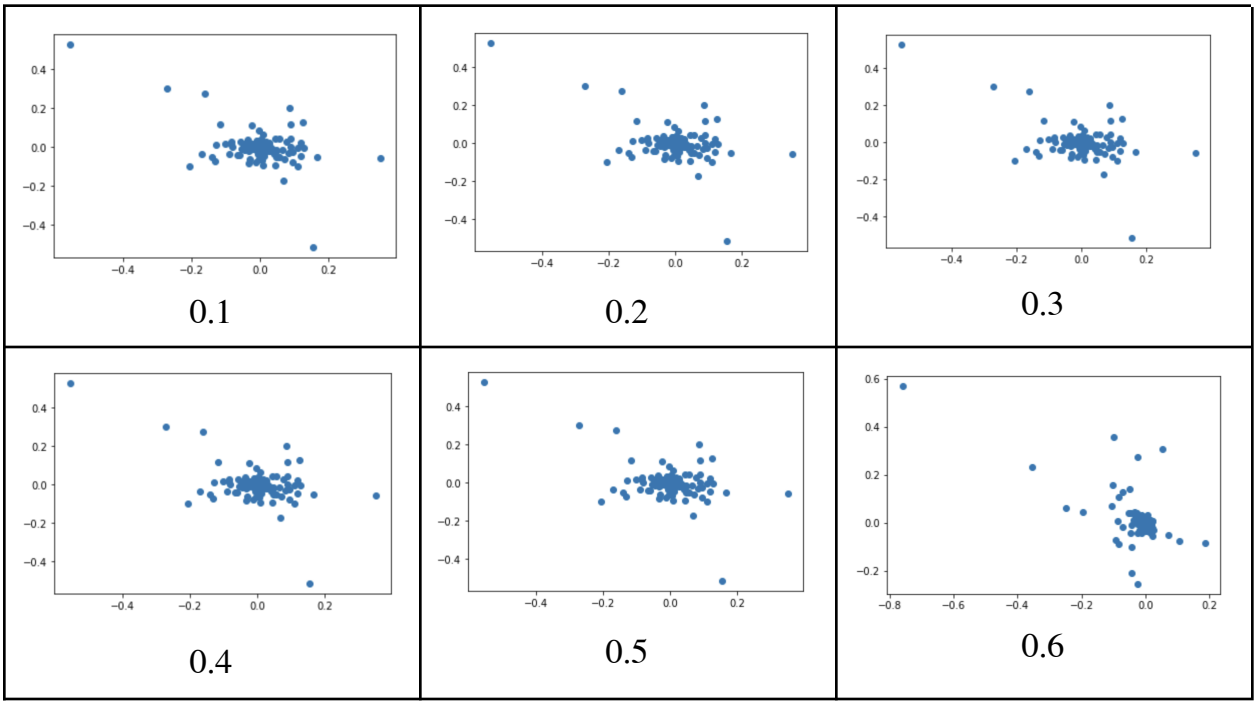
So for further computations I have reduced the sample size and taken only a few hundreds of data points and as the data points considered are not consistent in my case, so for further analysis of best kernel i will not consider the remaining kernels functions. First kernel for sigma = 0.1 i have calculated 2 times 1st one on whole 10,000 datapoints of X_Test and then reduced to few hundreds.

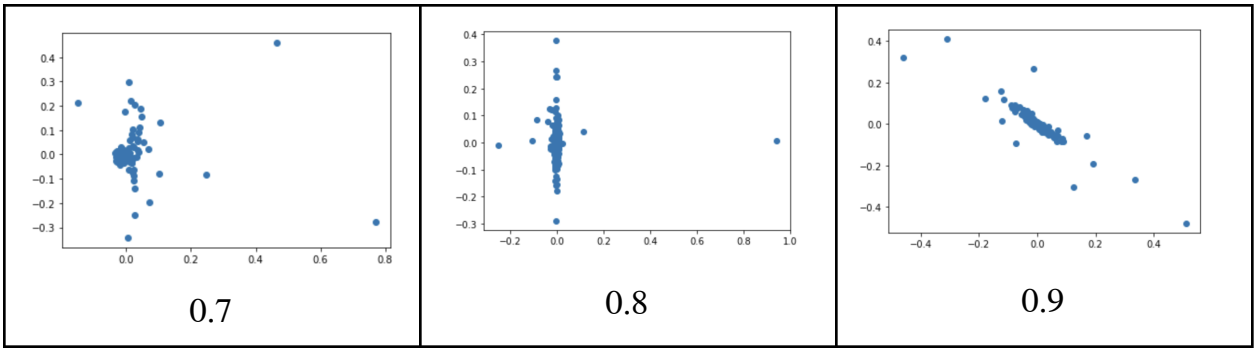
b. Kernel function $K(X,Y) = exp(\frac{-(X-Y)^T(X-Y)}{2\sigma^2}$

for sigma = 0.1 and on whole dataset of X_Test (pc-1 on X-axies and pc-2 on Y-axies) -

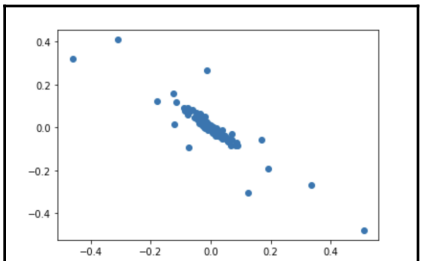


on few hundreds of data points for different sigma,pc-1 on X-axies and pc-2 on Y-axies, Number below the image represent sigma value -





0.7 0.8 0.9



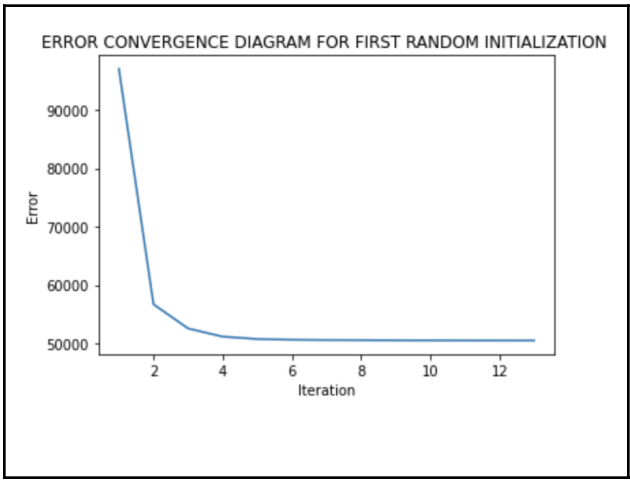
1.0

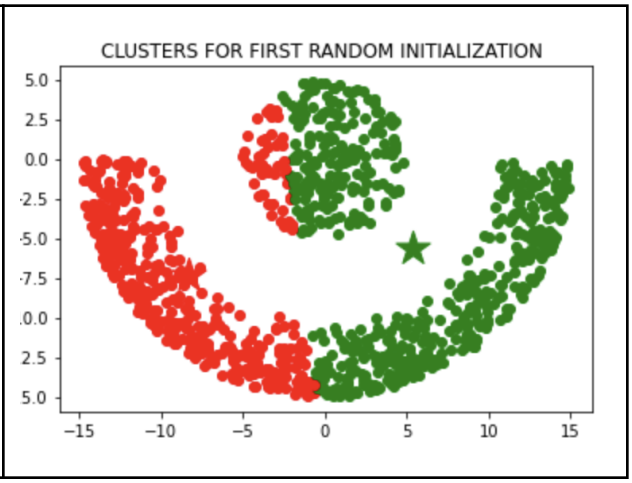
1.4 Among all the Kernel of $K(X,Y) = (1 + X^T Y)^d$ for d = {2,3,4} for d == 4 we are getting the maximum variance plot between top-2 eigenvector projections. Unfortunately due to computation constrain, we are not able to calculate the actual kernel matrix as dimensions will be (60,000 X 60,000) but again, with (10,000 X 10,000) we are consuming too much time for a single batch computation. So we reduced the batch size for computations. Hence it will not be correct to compare the results coming from different batch sizes, so while comparing, I have not considered the results of $K(X,Y) = exp(\frac{-(X-Y)^T(X-Y)}{2\sigma^2}$ function for any sigma value.

2. We are given dataset containing 1000 data points and each of 2-Dimension.

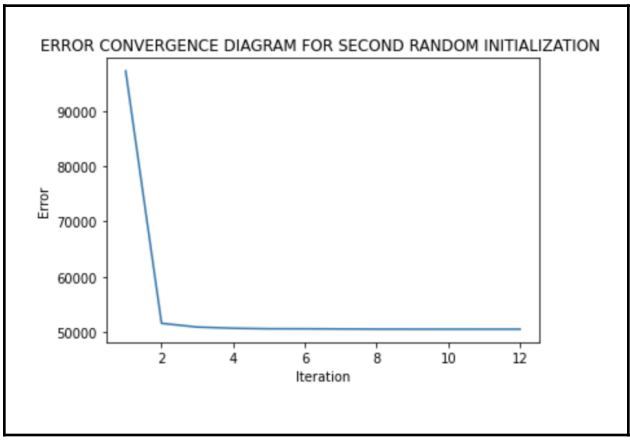
2.1. For this bit of question i have written K-means algorithm with K = 2 and tried for 5 different random initialization, plot of error function w.r.t iterations and the clusters obtained are attached below.

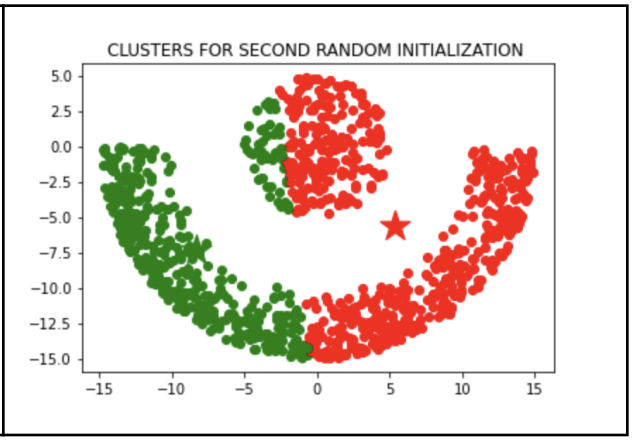
For random initialization - 1st plots -



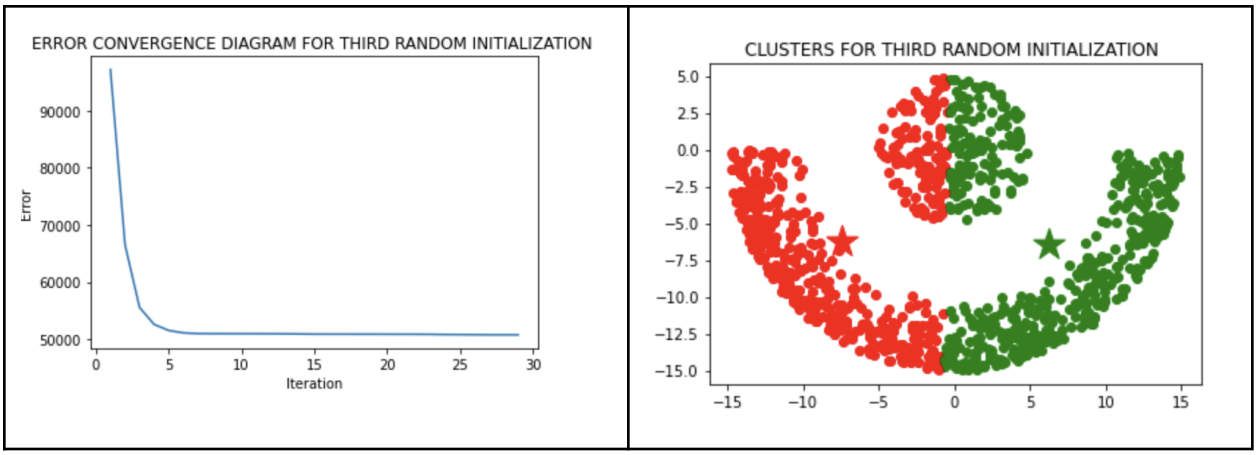


For random initialization - 2nd plots -

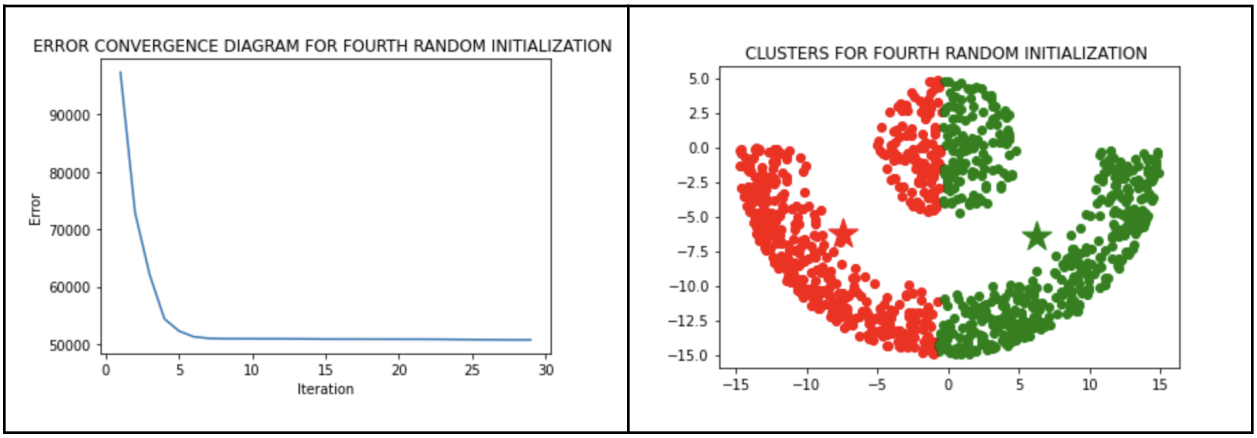




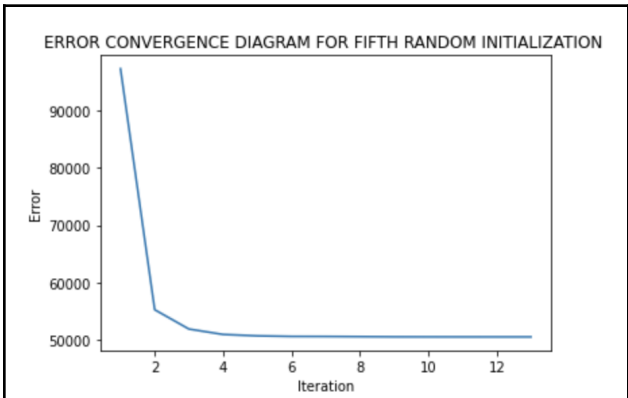
For random initialization - 3rd plots -



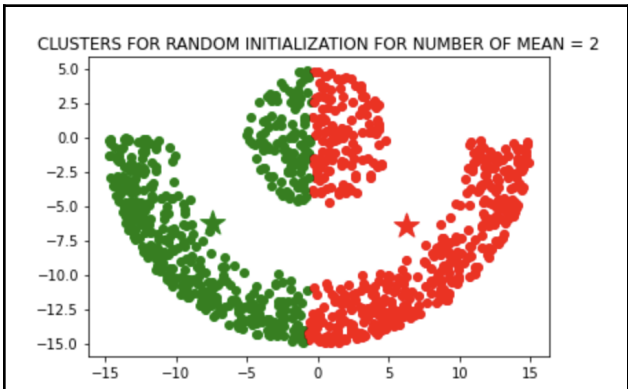
For random initialization - 4th plots -



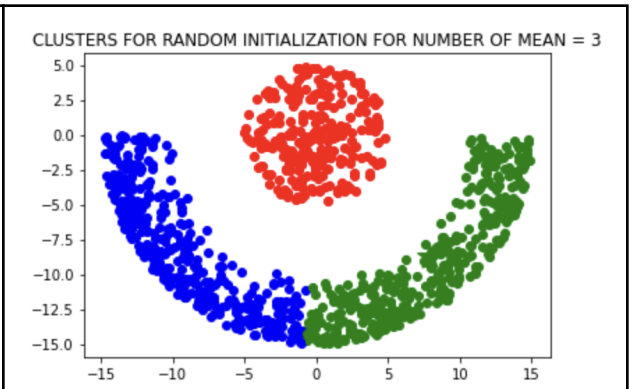
For random initialization - 5th plots -



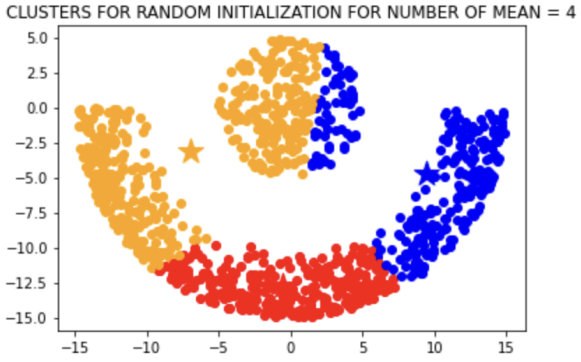
2.2 This part of the question asks to do a clustering for K = {2,3,4,5}, for a random initializations and after doing this computation, the Voronoi regions obtained are as follows -



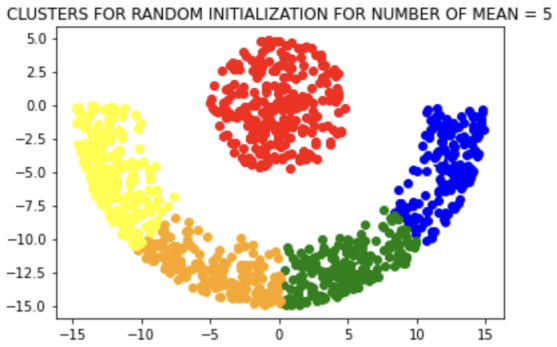
for k = 2



for k = 3

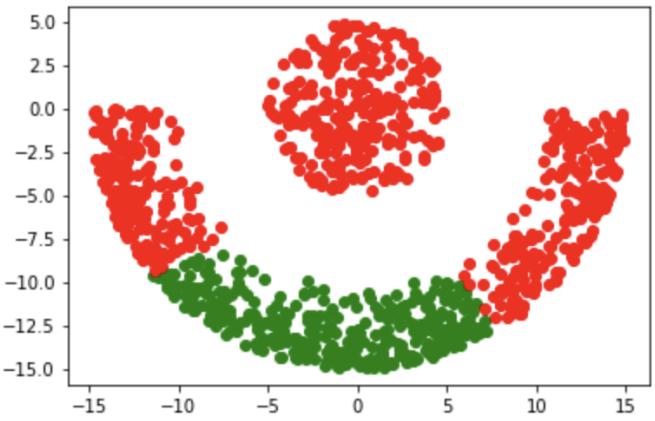


for k =4



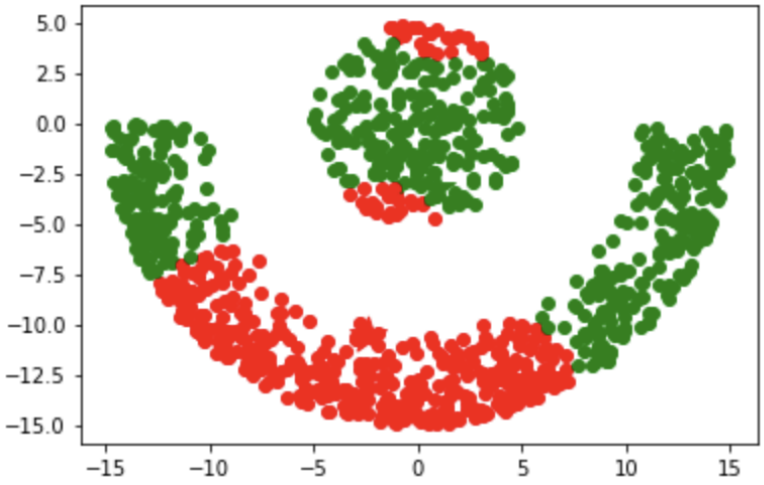
for k = 5

2.3 In this bit of question I have implemented spectral relaxation of K-means using Kernel PCA for k = 2. I have chosen $K(X, Y) = (1 + X^T Y)^2$ for computations after trying with different d values like (2,3,4) etc. Results shown below may vary significantly as the K-mean algorithm may result in different clusters depending on different initializations.



for k=2 spectral clustering algorithm

2.4 In this part of question i have assigned the data point ‘i’ to cluster ‘l’ ,where ‘l’ is max over all V_j^i ,and where V_j^i is the i-th eigen vector of the kernel matrix associated with the j-th largest eigen value. j ϵ {1,...,k} and i ϵ {1,..,n}. Results shown below may vary significantly as the K-mean algorithm may result in different clusters depending on different initializations.



for k=2 spectral clustering algorithm