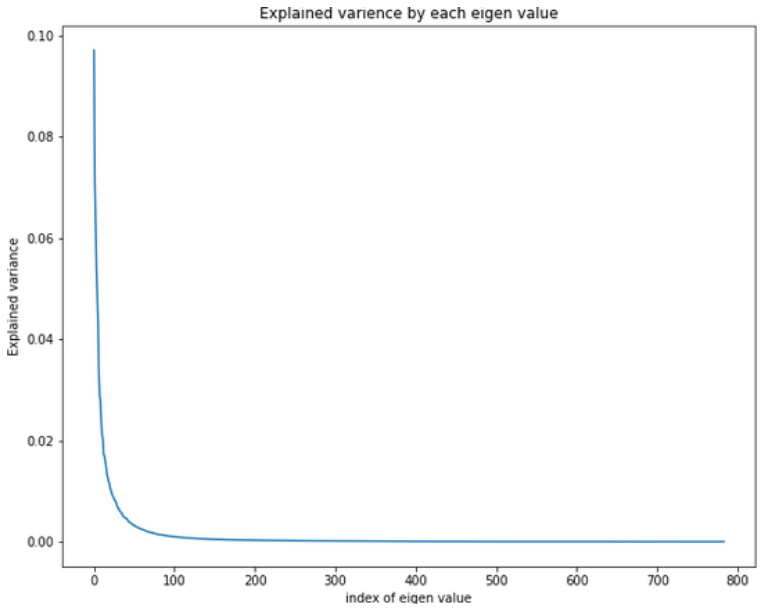
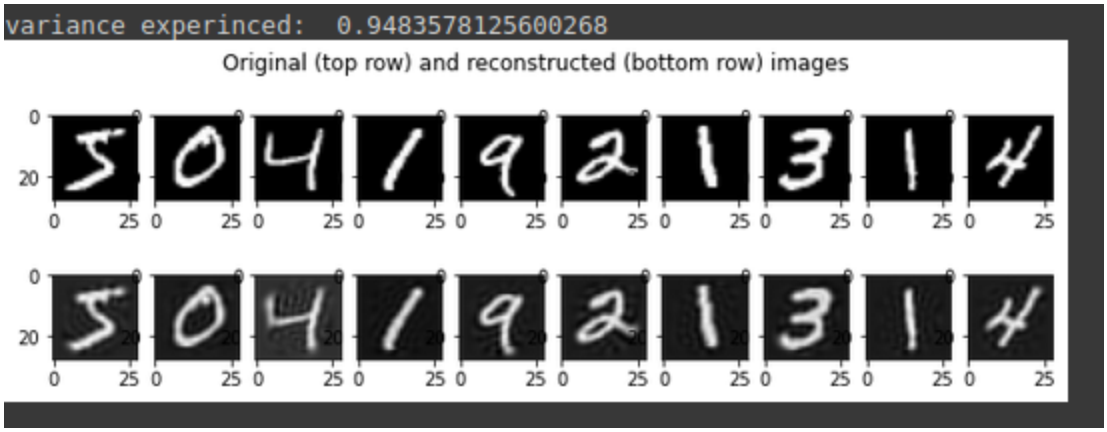
Name: Sukriti Shukla
Roll no: ED20B067

Q1)
i) This part requires 6 steps

- Calculate covariance matrix
- Calculate eigen values and eigen vectors of covariance matrix
- Sort the eigen values and sort the eigen vectors according to respective eigenvalues.
- Select top K eigen values as top k principal components.
- Variance explained by each component is = eigenvalues/eigenvalues.sum()



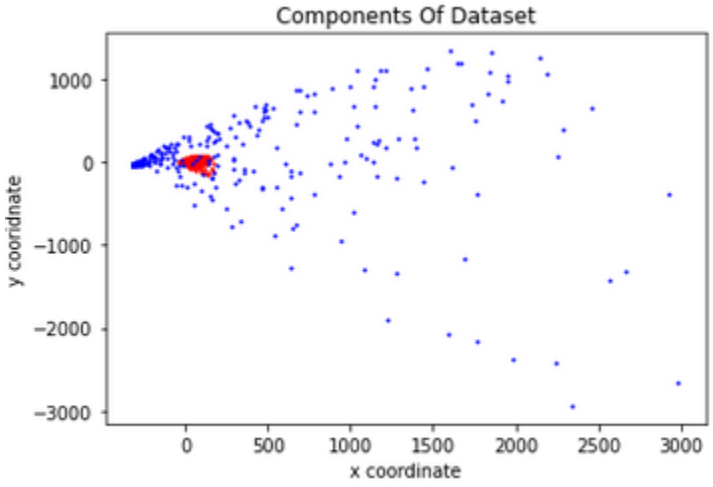
ii) I reconstructed 10 images from the dataset using different number of dimensions, after calculation cumulative variance for different number of dimensions,taking 150 top principal components gave variance nearly about 0.95, which is the desired value.



iii) I took first 600 images for kernel pca ,due to the amount of time program took in execution,

Then I did the following steps:

- Calculate K matrix value using the formula k[i][j]= (1+np.matmul(X_train[:,i].T, X_train[:,j]))**d, for different values of d
- Then I centered the K matrix.
- Calculated top 2 eigen values and corresponding eigen vectors.Then I calculated alpha using: alpha = eigen_vector/(np.sqrt(eigen_value))
- Then I calculated components of the data set using the the formula components = np.matmul(alpha.T,Kc)
  Then I plotted the data for different values of d



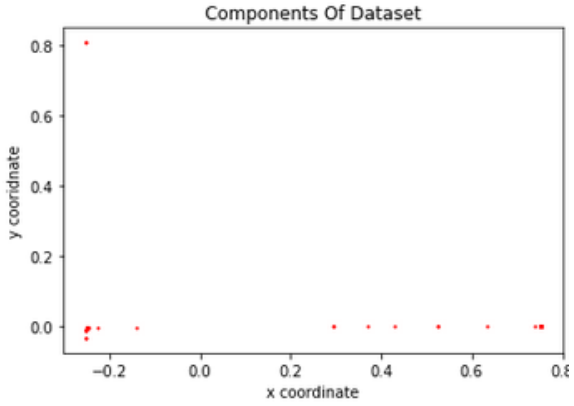
For the next kernel function I calculated K values using

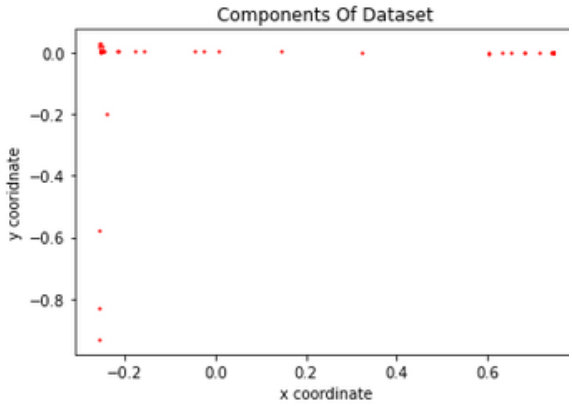
k[i][j] =

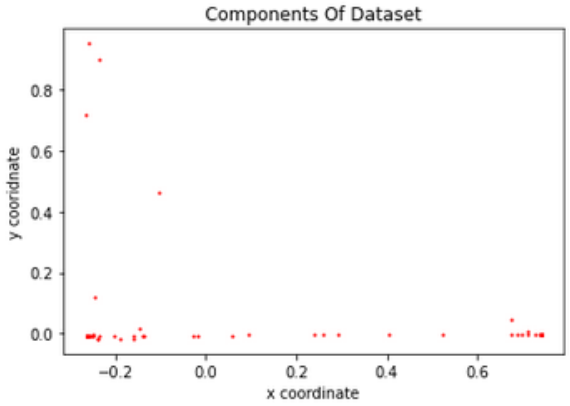
np.exp(-(np.matmul((X_train[:,i]-X_train[:,j]).T,(X_train[:,i]-X_train[:,j])))/(var**2))

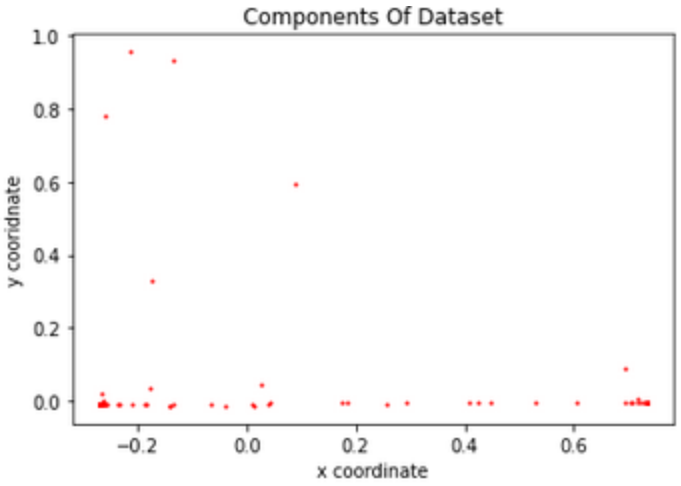
Then I repeated the above steps again, to plot various components of data for different values of sigma.

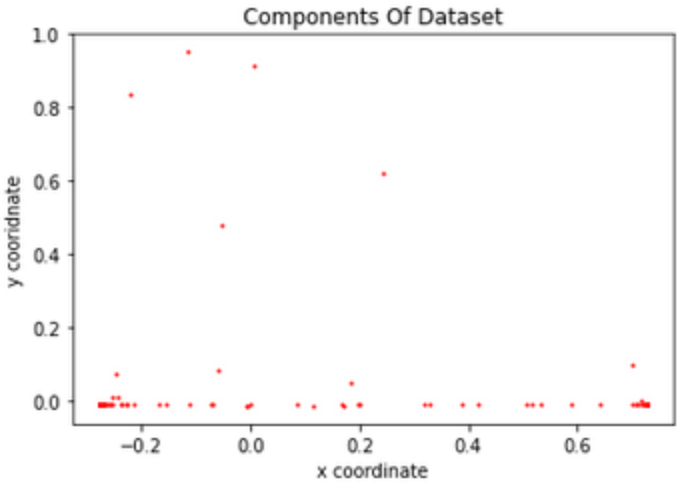
iv) By visually examining the data, it is evident that using the Gaussian kernel or Radial Basis Function (RBF) results in dense clustering of data for lower values of sigma, which then splits along the axes for higher sigma values. However, this does not provide any meaningful insight into the data. On the other hand, using a polynomial kernel results in data being widely spread out in the plot, which gives the impression that it could be a useful method for separating the data.

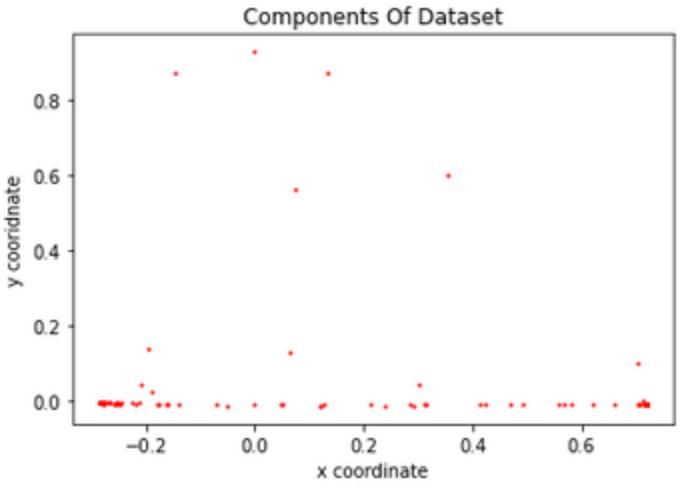












Q2)

i) I made 5 functions for the k means

- Initialization: To randomly initialize the k centroids .

- Reassign_clusters: To reassign clusters to the data points based on their distances with the current centroid.
- Reassign_centroids: To recalculate centroid of each cluster taking mean of all the datapoints.
- Plot_clusters: to plot the different clusters in different colors.

Then I ran the loop for 5 random initializations:

- Initialize the centroid .
- Get clusters for that initial centroid.
- Then reassign centroids according to the new clusters.
- Then I used num_iterations to perform the lloyds algorithm given number of times, I came across the the number of iterations using various trials and tests. We can also use an infinite loop to rum the algorithm as long as it does not converge testing if consecutive clusters are not same but this took lot of times for some initializations but produced no better results than my trial iteration values thats why I preferred to give iterations a fixed number.
- Then after getting final clusters we can plot them using the plot functions

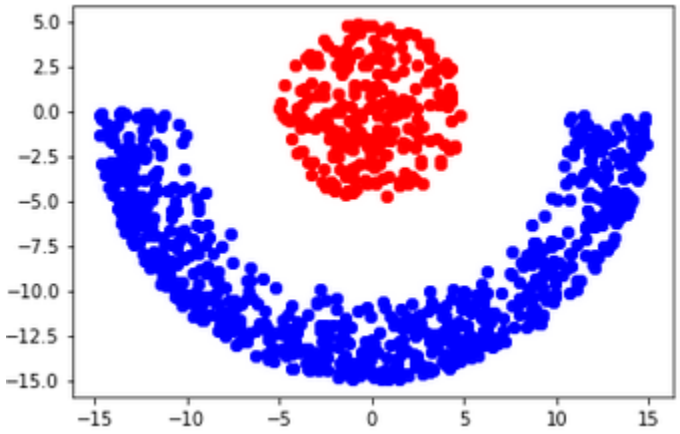
ii) I plotted vornoi regions and showed them on the clusters. Since the vernoi_plot-2d library didn't support drawing regions for 2 clusters, I wrote a code to draw the perpendicular bisector of the final centroids in case there are 2 clusters.

iii) I tried using polynomial kernel but results were not satisfactory, So I moved to use RBF or the gaussian kernel, I tried plotting data for different values of sigma but the result was not satisfactory , So I tried replicating the approach shown in the research paper
https://ai.stanford.edu/~ang/papers/nips01-spectral.pdf

Given a set of points $S = \{s_1, \ldots, s_n\}$ in $\mathbb{R}^l$ that we want to cluster into k subsets:

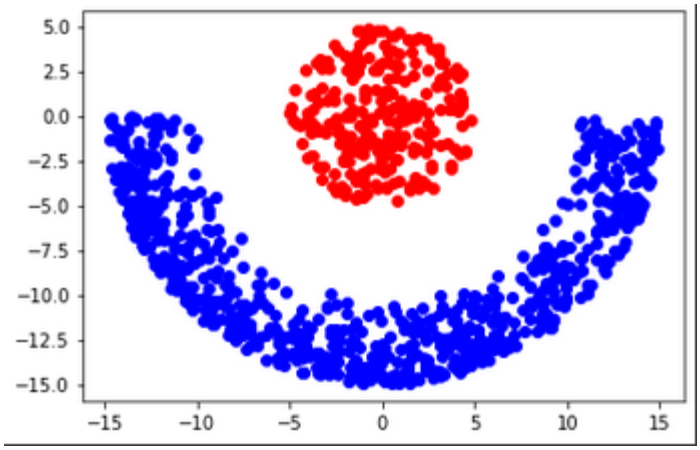
1. Form the affinity matrix $A \in \mathbb{R}^{n \times n}$ defined by $A_{ij} = \exp(-||s_i - s_j||^2/2\sigma^2)$ if $i \neq j$, and $A_{ii} = 0$.
2. Define D to be the diagonal matrix whose (i,i)-element is the sum of A's i-th row, and construct the matrix $L = D^{-1/2}AD^{-1/2}$.[1]
3. Find $x_1, x_2, \ldots, x_k$, the k largest eigenvectors of L (chosen to be orthogonal to each other in the case of repeated eigenvalues), and form the matrix $X = [x_1 x_2 \ldots x_k] \in \mathbb{R}^{n \times k}$ by stacking the eigenvectors in columns.
4. Form the matrix Y from X by renormalizing each of X's rows to have unit length (i.e. $Y_{ij} = X_{ij}/(\sum_j X_{ij}^2)^{1/2}$).
5. Treating each row of Y as a point in $\mathbb{R}^k$, cluster them into k clusters via K-means or any other algorithm (that attempts to minimize distortion).
6. Finally, assign the original point s_i to cluster j if and only if row i of the matrix Y was assigned to cluster j.

And then I was able to get the desired results.



iv)z = np.argmax(h, axis=1)

And then I used plot_cluters to plot data according to this clustering.


5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−15
−10
−5
0
5
10
15