CS5691 - PRML Assignment 1 Report

March 11, 2024

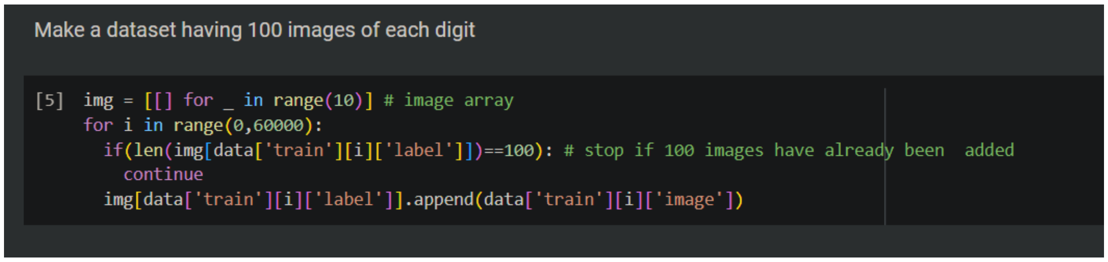
1 PCA and Kernel PCA

Download the MNIST dataset from https://huggingface.co/datasets/mnist. Use a random set of 100 images chosen from each class (0 to 9) as your dataset.

1.1 Principal Components and Variance

i.Write a piece of code to run the PCA algorithm on this data-set. Show the images of the principal components that you obtain. How much of the variance in the data-set is explained by each of the principal components?

We begin by importing the dataset and assigning it to a variable 'data' in order to use it. Now, we need to split the images using their labels and take 100 images of each label



Make a dataset having 100 images of each digit

```
[5] img = [[] for _ in range(10)] # image array
    for i in range(0,60000):
      if(len(img[data['train'][i]['label']])==100): # stop if 100 images have already been  added
        continue
      img[data['train'][i]['label']].append(data['train'][i]['image'])
```

Figure 1: Training data.

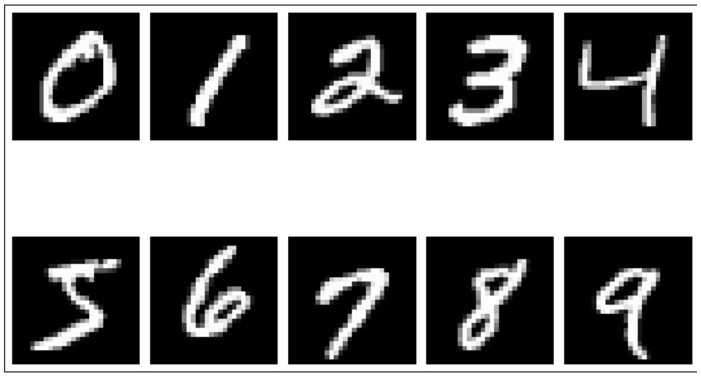
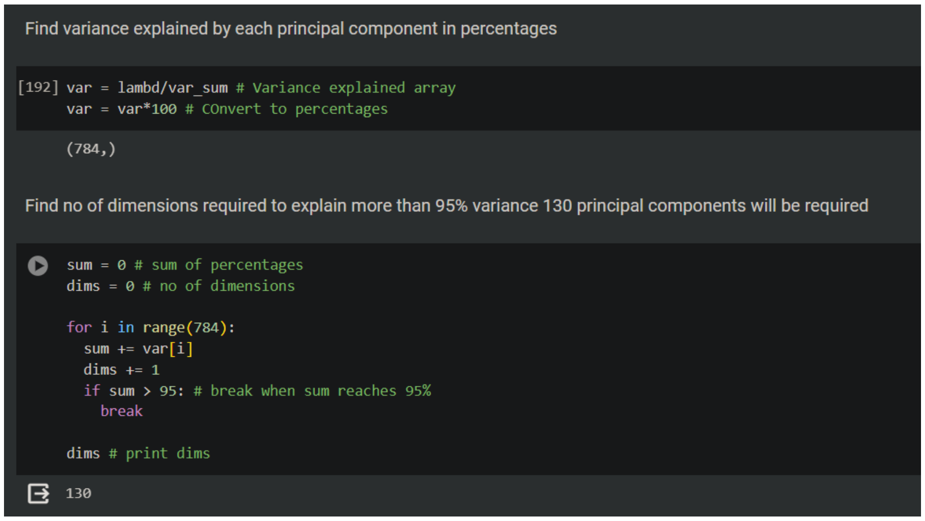


Figure 2: Representative images.

Next, we take these images, flatten their grayscale pixel values from a 28x28 matrix to a 784 length array and store it in a data matrix. We can then centre the dataset before proceeding with PCA

We can now proceed by calculating the covariance matrix and finding it's eigenvalues and eigenvectors

Now, we can calculate number of principal components required to explain 95percent of any datapoint. We need 130 dimensions to explain the dataset


Find variance explained by each principal component in percentages

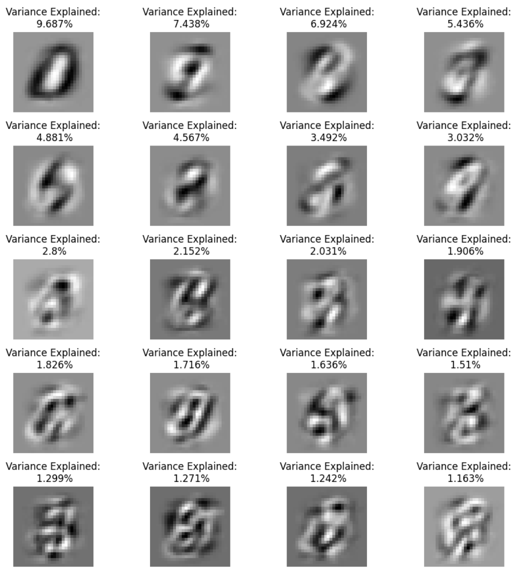
```
[192] var = lambd/var_sum # Variance explained array
      var = var*100 # COnvert to percentages

      (784,)
```

Find no of dimensions required to explain more than 95% variance 130 principal components will be required

```
sum = 0 # sum of percentages
dims = 0 # no of dimensions

for i in range(784):
  sum += var[i]
  dims += 1
  if sum > 95: # break when sum reaches 95%
    break

dims # print dims
```

```
130
```

Figure 3: Variance.



Variance Explained: 9.687%
Variance Explained: 7.438%
Variance Explained: 6.924%
Variance Explained: 5.436%
Variance Explained: 4.881%
Variance Explained: 4.567%
Variance Explained: 3.492%
Variance Explained: 3.032%
Variance Explained: 2.8%
Variance Explained: 2.152%
Variance Explained: 2.031%
Variance Explained: 1.906%
Variance Explained: 1.826%
Variance Explained: 1.716%
Variance Explained: 1.636%
Variance Explained: 1.51%
Variance Explained: 1.299%
Variance Explained: 1.271%
Variance Explained: 1.242%
Variance Explained: 1.163%

Figure 4: Top 20 Principal Components.

1.2 Reconstruction of the data

ii.Reconstruct the dataset using different dimensional representations. How do these look like? If you had to pick a dimension d that can be used for a downstream task where you need to classify the digits correctly, what would you pick and why?

Taking some random values of dimensions, I have reconstructed the data, here are the results. As explained before, 130 would be a good number as it explains 95percent of the data and that is confirmed by the reconstruction. Around 300 dims, we can be almost sure of the data. Hence, 300 can be used if accuracy is an important aspect.

1.3 Kernel PCA

iii. Write a piece of code to implement the Kernel PCA algorithm on this dataset for various choices of K that you think are reasonable for this problem. Plot the projection of each point in the dataset onto the top-2 components for each kernel. Use one plot for each kernel and in the case of (B), use a different plot for each value of K.

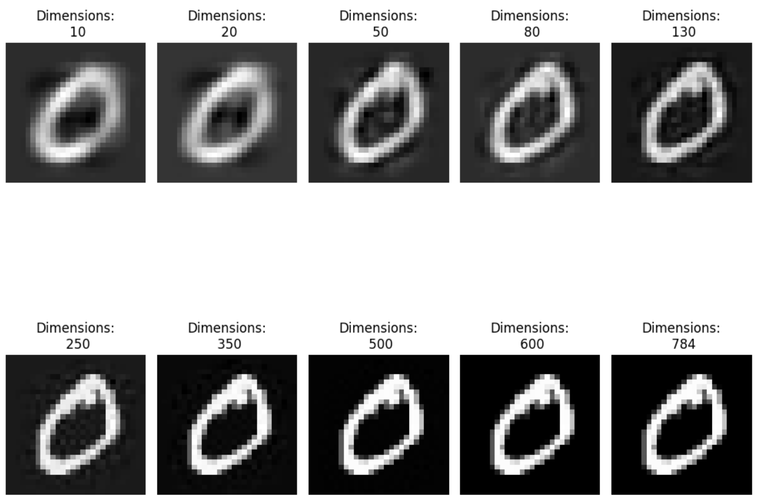
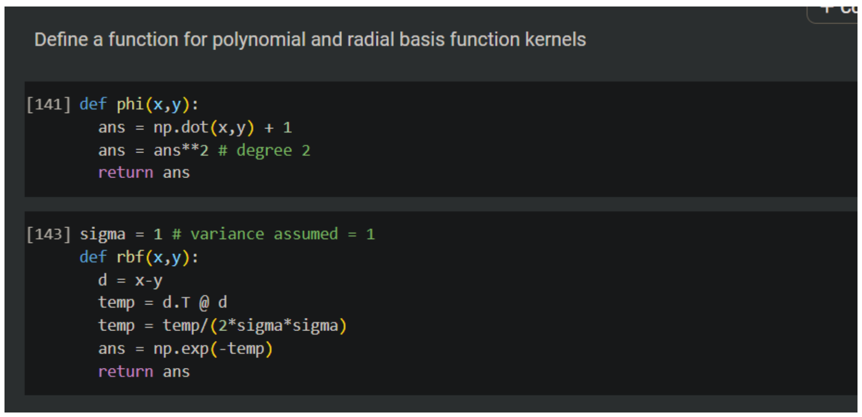


Figure 5: Reconstructed Data.

We define 2 kernel functions, the polynomial and the radial basis function. Using this we can find the K matrix, and it's eigenvalues and eigenvectors thereafter. Using these, the α matrix can be found



Define a function for polynomial and radial basis function kernels

```
[141] def phi(x,y):
        ans = np.dot(x,y) + 1
        ans = ans**2 # degree 2
        return ans
```

```
[143] sigma = 1 # variance assumed = 1
      def rbf(x,y):
        d = x-y
        temp = d.T @ d
        temp = temp/(2*sigma*sigma)
        ans = np.exp(-temp)
        return ans
```

Figure 6: Kernel Functions.

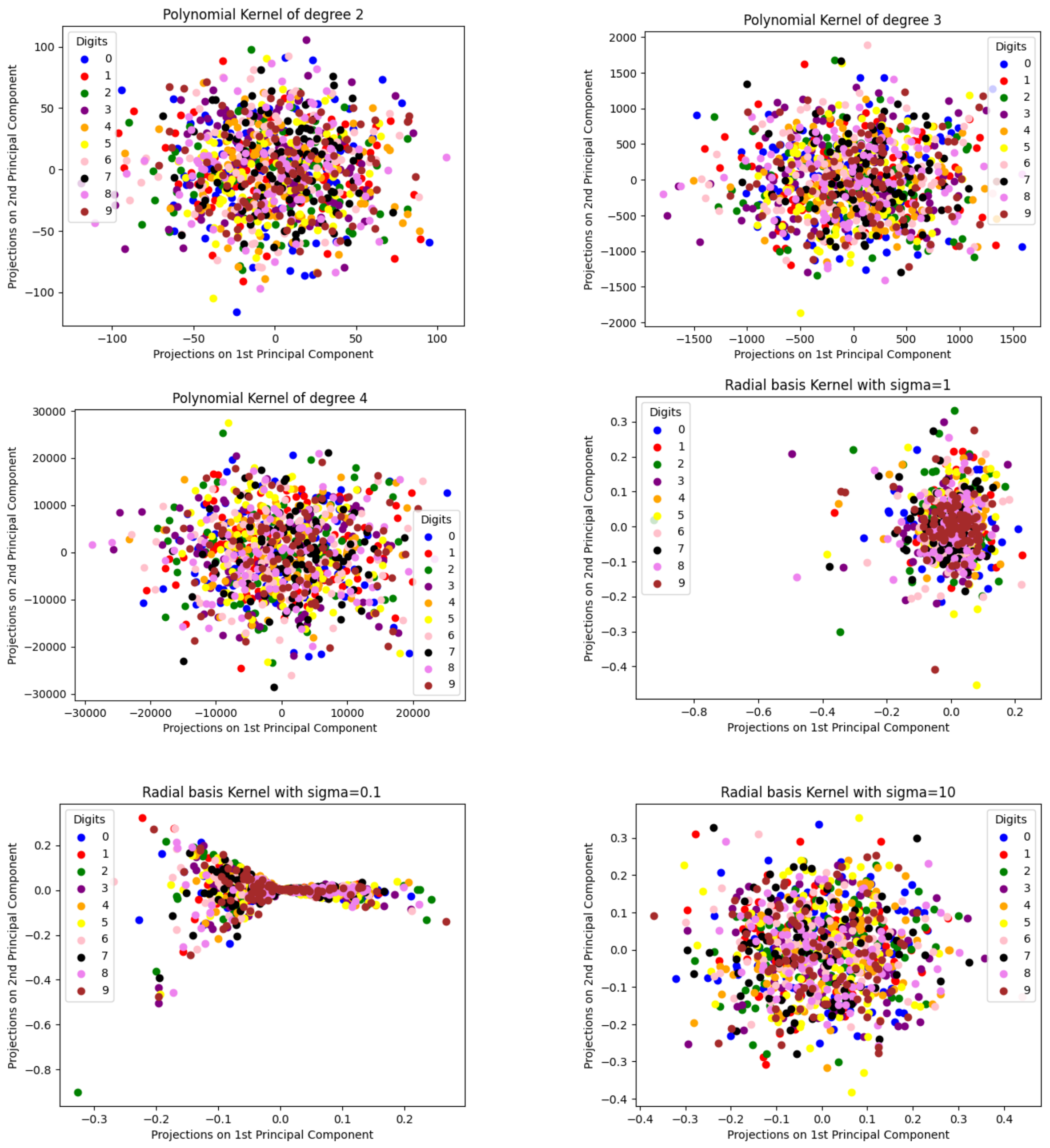


Figure 7: Kernel PCA Results.

2 K-Means

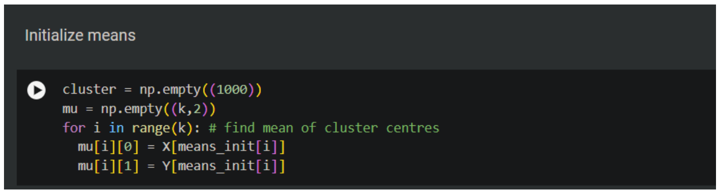
You are given a data-set with 1000 data points each in R2 (cm dataset 2.csv).

2.1 K-Means with random means for k=2

i. Write a piece of code to run the algorithm studied in class for the K-means problem with k = 2 . Try 5 different random initialization and plot the error function w.r.t iterations in each case. In each case, plot the clusters obtained in different colors.

First, we shall randomly pick the cluster centres and assign them as the cluster means. Next, we run Lloyd's Algorithm. We iterative successively wherein we re-assign points to clusters based on the absolute distance between them and the cluster centres. After re-assignment, we recalculate the means and find the error. if the error remains the same, the algorithm has converged

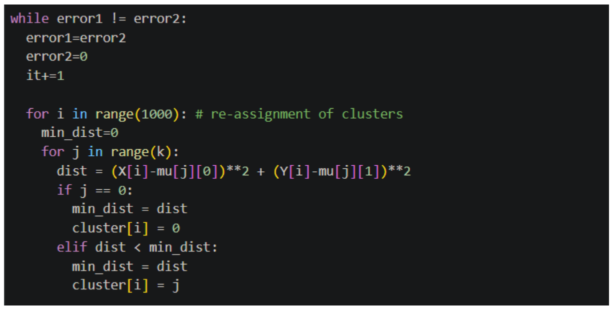
First, I have done for k=2 clusters and 5 different randomizations, the results are shown. Also, clustering for k=2,3,4,5 has also been plotted with a fixed randomization



Initialize means

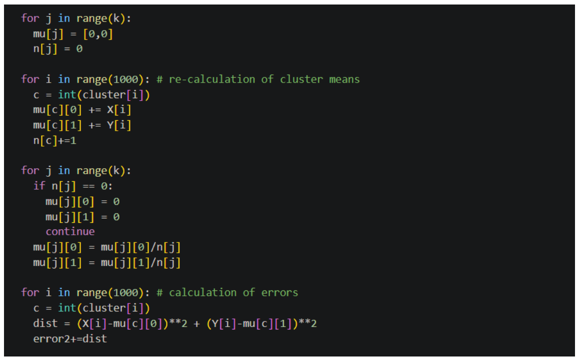
```
cluster = np.empty((1000))
mu = np.empty((k,2))
for i in range(k): # find mean of cluster centres
  mu[i][0] = X[means_init[i]]
  mu[i][1] = Y[means_init[i]]
```

Figure 8: Means Initialization.



```
while error1 != error2:
  error1=error2
  error2=0
  it+=1

  for i in range(1000): # re-assignment of clusters
    min_dist=0
    for j in range(k):
      dist = (X[i]-mu[j][0])**2 + (Y[i]-mu[j][1])**2
      if j == 0:
        min_dist = dist
        cluster[i] = 0
      elif dist < min_dist:
        min_dist = dist
        cluster[i] = j
```

Figure 9: Re-assignment of Clusters.

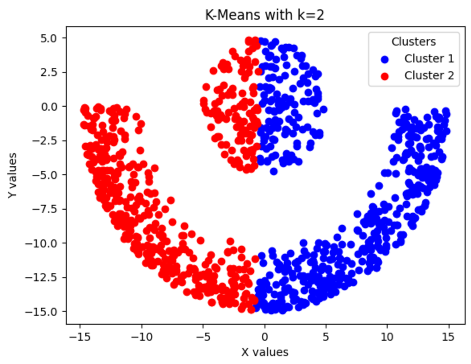


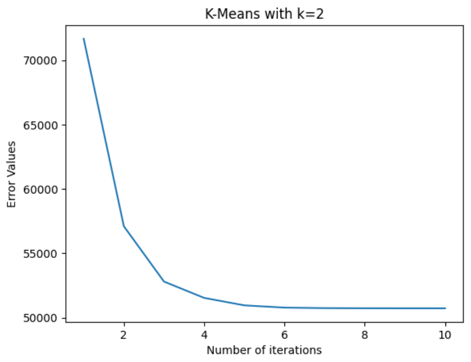
```
for j in range(k):
  mu[j] = [0,0]
  n[j] = 0

for i in range(1000): # re-calculation of cluster means
  c = int(cluster[i])
  mu[c][0] += X[i]
  mu[c][1] += Y[i]
  n[c]+=1

for j in range(k):
  if n[j] == 0:
    mu[j][0] = 0
    mu[j][1] = 0
    continue
  mu[j][0] = mu[j][0]/n[j]
  mu[j][1] = mu[j][1]/n[j]

for i in range(1000): # calculation of errors
  c = int(cluster[i])
  dist = (X[i]-mu[c][0])**2 + (Y[i]-mu[c][1])**2
  error2+=dist
```

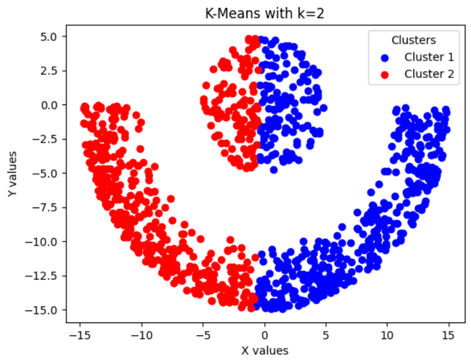
Figure 10: Re-calculation of cluster means.

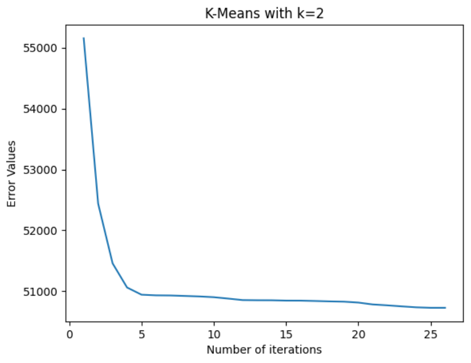
2.2 K-means for different k

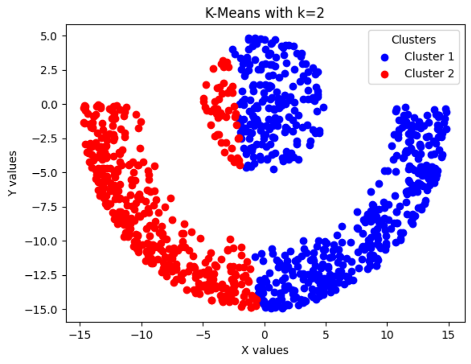
ii.Fix a random initialization. For K = f2; 3; 4; 5g, obtain cluster centers according to K-means algorithm using the fixed initialization. For each value of K, plot the Voronoi regions associated to

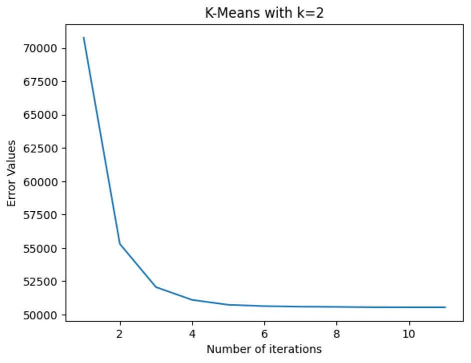












each cluster center. (You can assume the minimum and maximum value in the data-set to be the range for each component of R2).

Clustering using Lloyd's Algorithm is shown for different values of k

2.3 Spectral Clustering

iii.Run the spectral clustering algorithm (spectral relaxation of K-means using Kernel-PCA) k = 2. Choose an appropriate kernel for this data-set and plot the clusters obtained in different colors. Explain your choice of kernel based on the output you obtain.

2.4 Spectral Clustering without Lloyd's Algorithm

In this, instead of running Lloyd's algorithm over the rows, the row is assigned to cluster with the highest corresponding value in that row

2.5 Best suited kernel

This is a continuation of 1st question. It has been written in the last for easier formatting.

iv.Which Kernel do you think is best suited for this dataset and why?

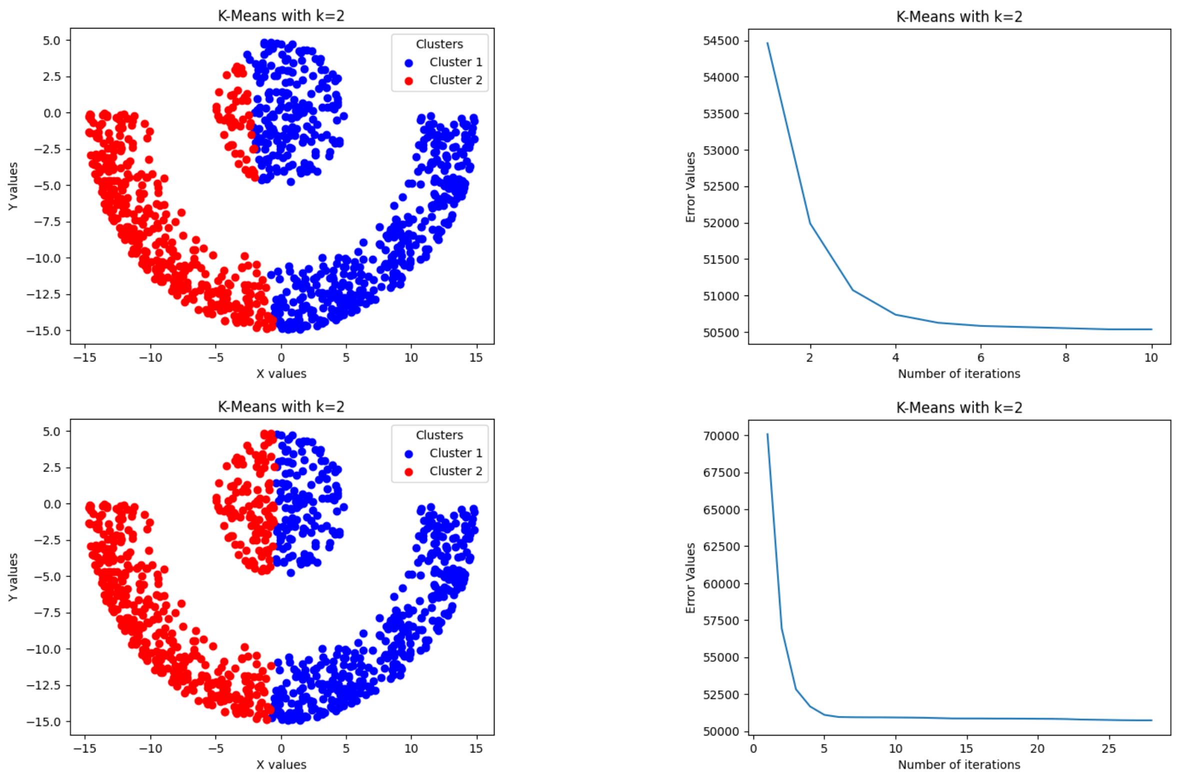


Figure 11: K-means for k=2.

Judging from the data we plotted, I would say, Radial basis kernel with $\sigma = 1$ is the best suited as it has projections concentrated near zero. This means that it can easily convey the data in very few dimensions

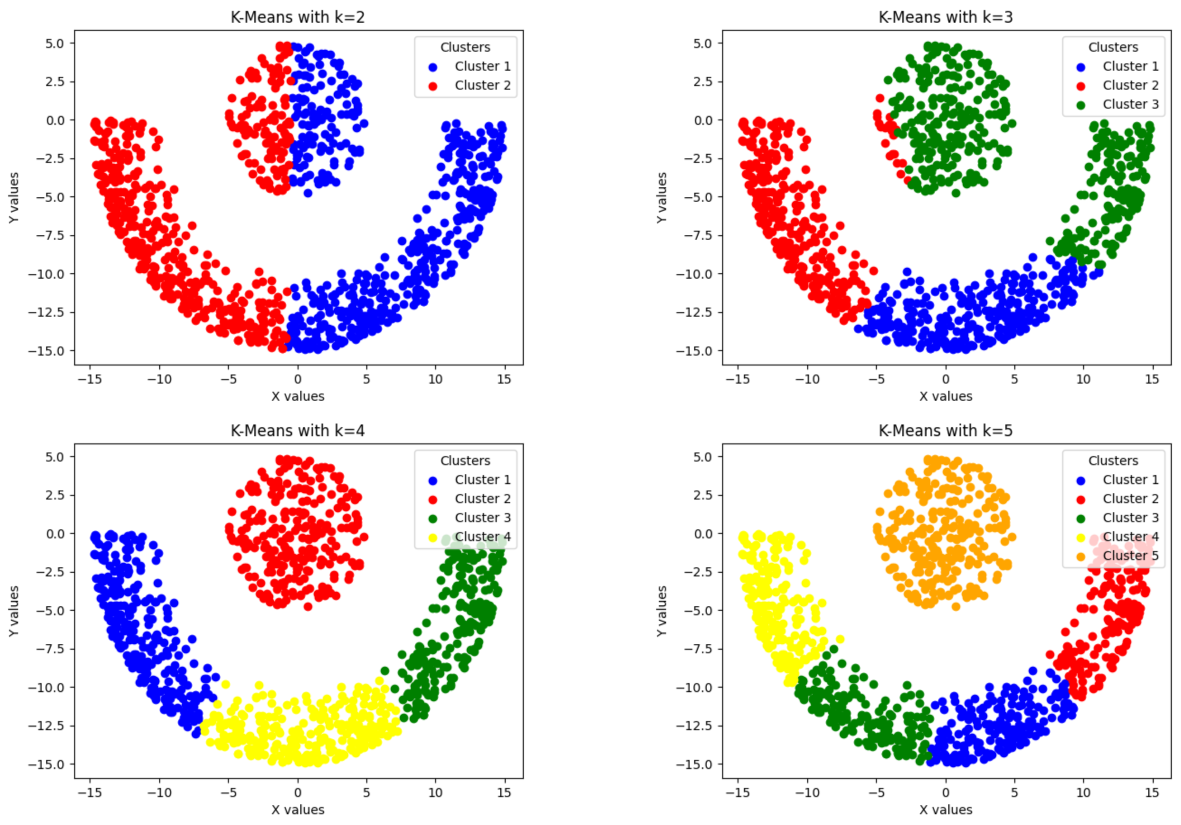
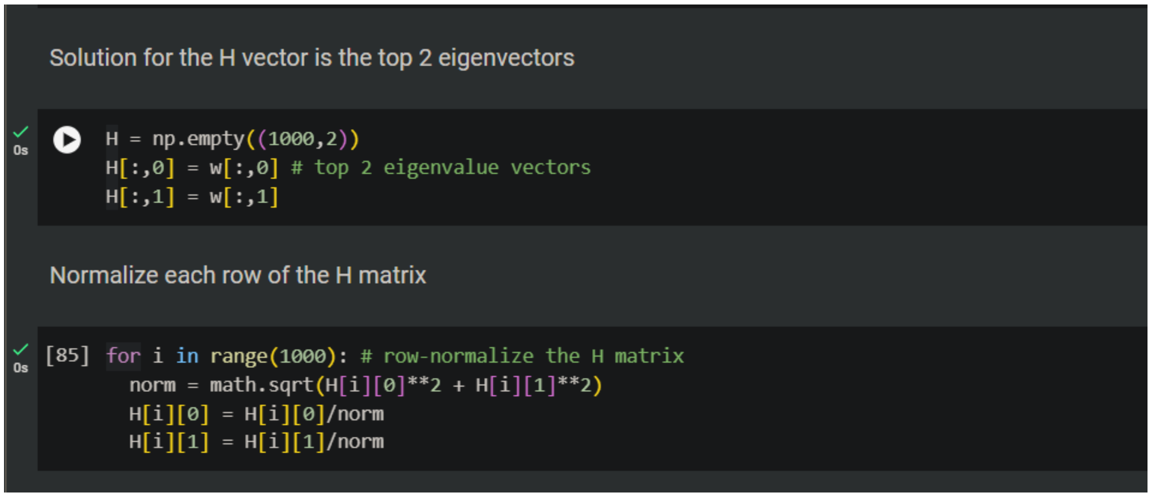


Figure 12: K-means for k=2,3,4,5.



Solution for the H vector is the top 2 eigenvectors

```
H = np.empty((1000,2))
H[:,0] = w[:,0] # top 2 eigenvalue vectors
H[:,1] = w[:,1]
```

Normalize each row of the H matrix

```
[85] for i in range(1000): # row-normalize the H matrix
       norm = math.sqrt(H[i][0]**2 + H[i][1]**2)
       H[i][0] = H[i][0]/norm
       H[i][1] = H[i][1]/norm
```

Figure 13: H matrix updation.

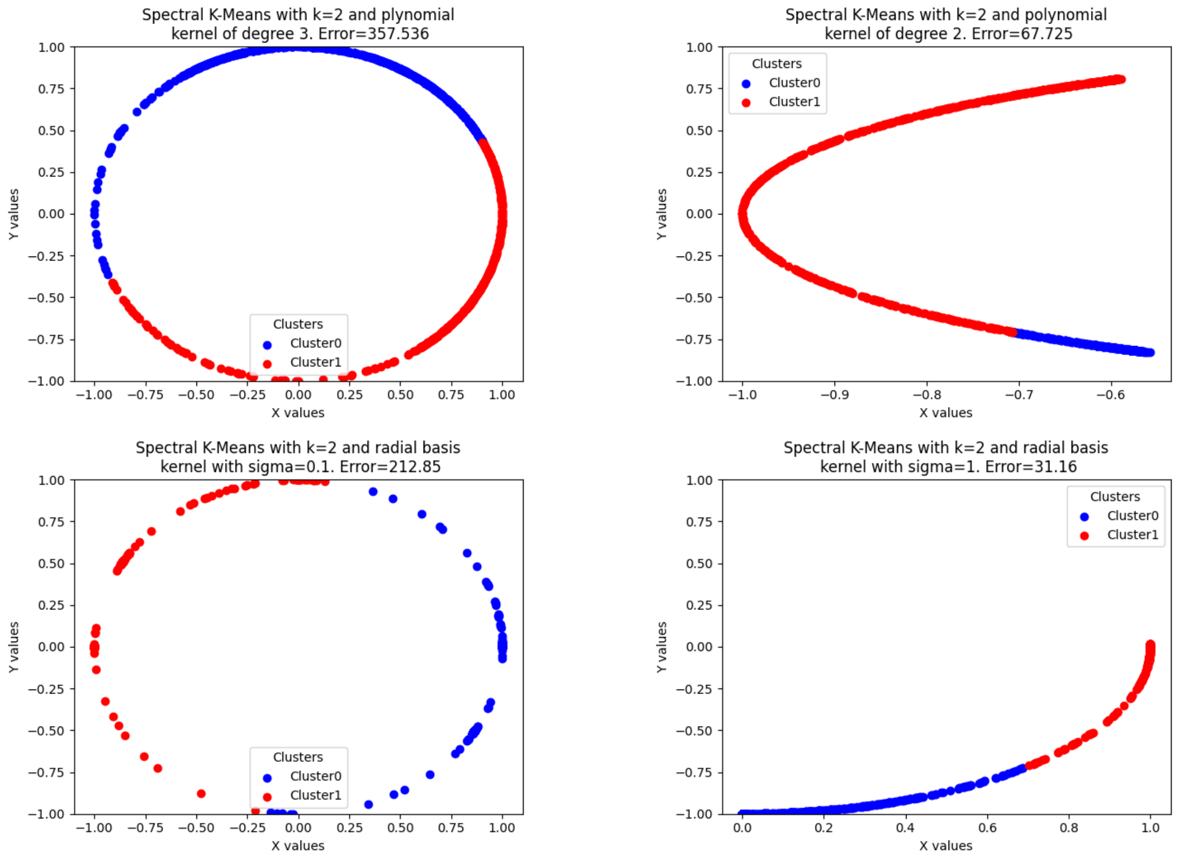
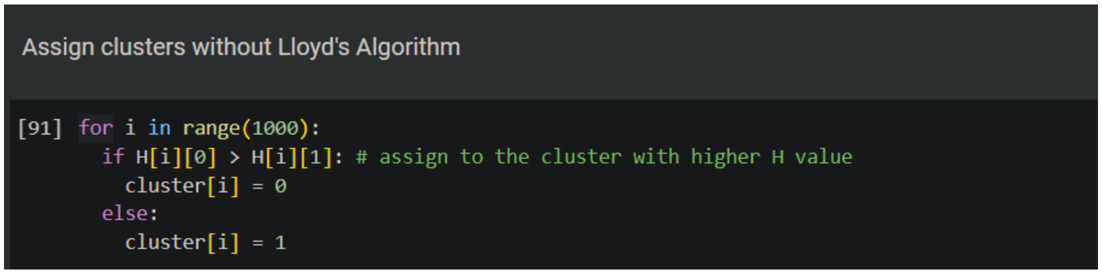


Figure 14: Spectral K-means for k=2.



Assign clusters without Lloyd's Algorithm

```
[91] for i in range(1000):
       if H[i][0] > H[i][1]: # assign to the cluster with higher H value
         cluster[i] = 0
       else:
         cluster[i] = 1
```

Figure 15: Cluster assignment from H matrix.

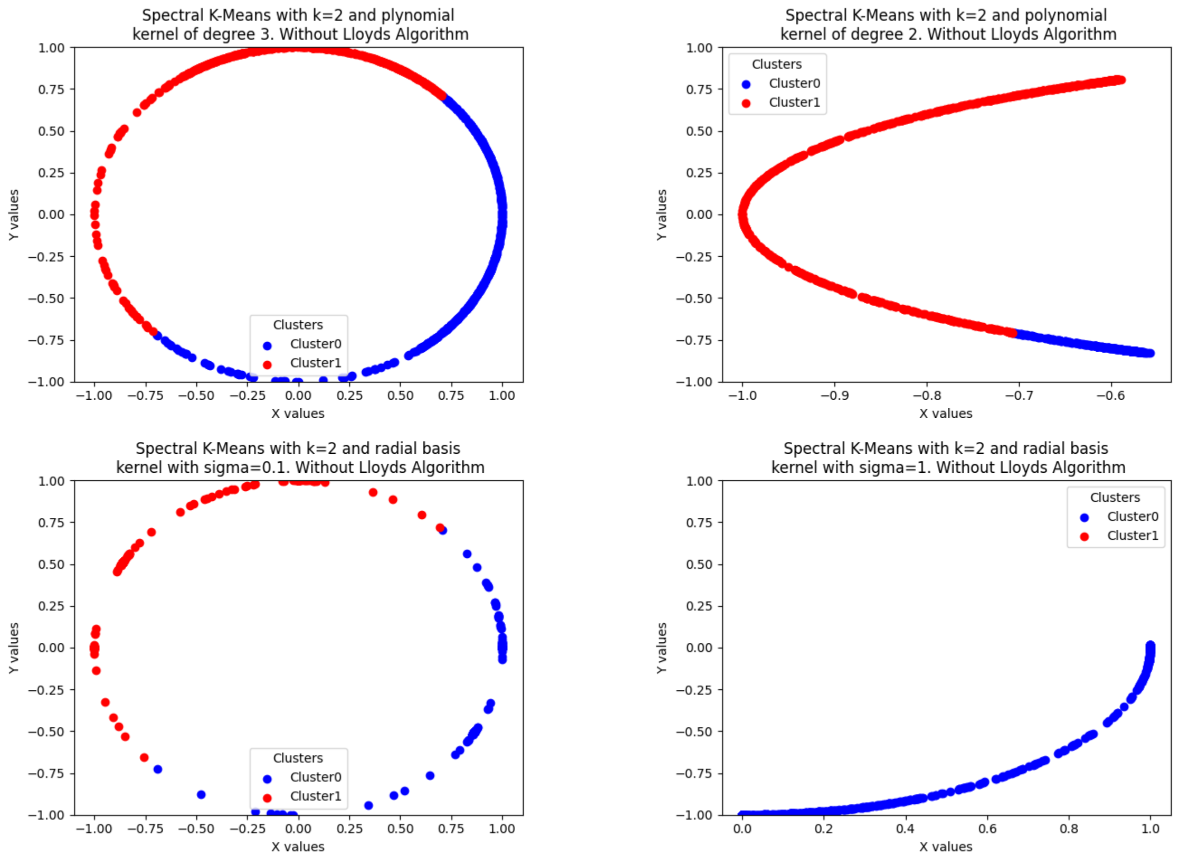


Figure 16: Spectral K-means without Lloyd’s Algorithm.