CS5691: Pattern Recognition and Machine Learning
Assignment 1

Lalit Jayanti, ME21B096

Question 1

Part (i): Running PCA

The PCA algorithm is run on 1000 randomly chosen images (100 from each class 0-9) from MNIST dataset. This is achieved by taking each 28x28 image and converting it to a 784 dimensional vector.

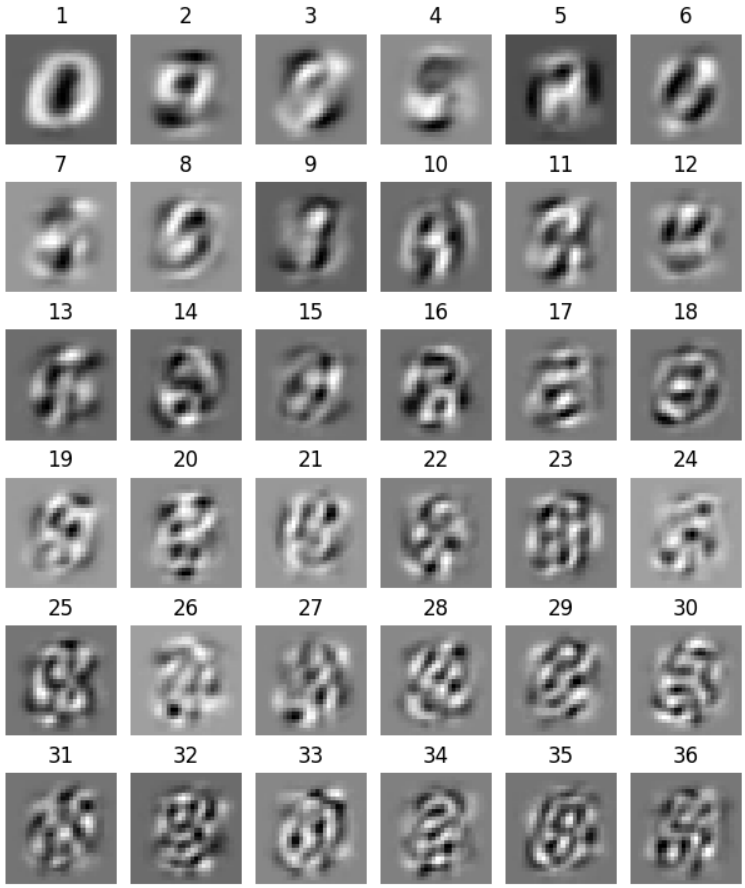


Figure 1: Images of the Principal Components

Figure 1 shows the image representation of the first 36 eigenvectors out of 784 (eigenpictures). Each eigenpicture shows a different feature that can be used to evaluate an image. For example eigenpicture-1 seems evaluates the presence of large loop.

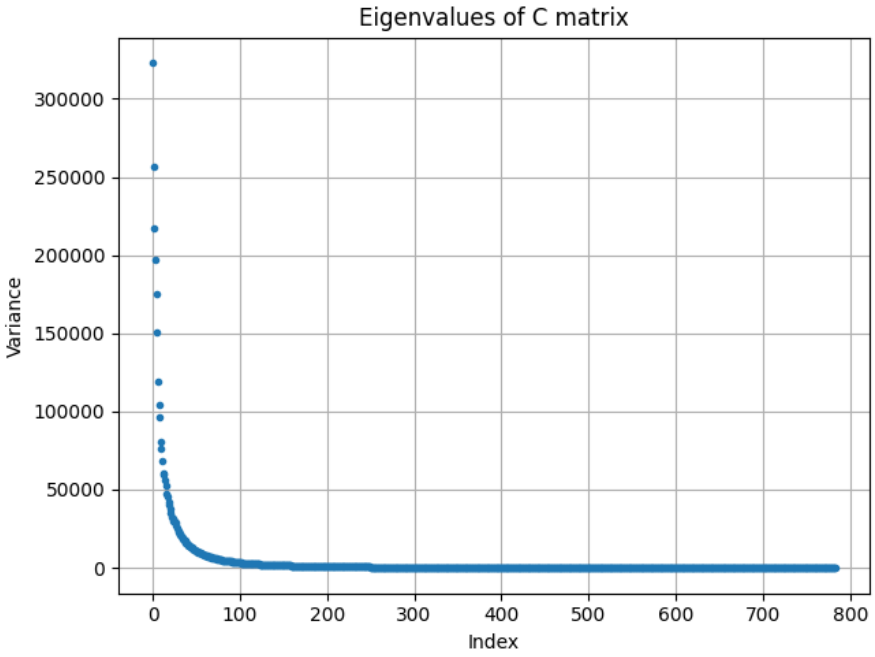


Figure 2: Variance explained

Figure 2 shows the plot of the variance explained by each component (variance is equal to the eigenvalue here) (from 1 to 784). The eigenvalues are sorted in descending order. It is observed the amount of variance explained decreases quickly, falls to nearly 0.0% in about 130 values. Numerical values of the variance are presented for the first 10 directions out of 784 in Table 1. For example it is seen that eigenvector-1 explains 9.37% variance while eigenvector-10 explains 2.34% only.

Variance	Percent Variance explained
323224.16	9.37
256903.07	7.45
217670.09	6.31
197227.50	5.72
174950.15	5.07
150401.47	4.36
118927.73	3.45
103884.25	3.01
96676.89	2.80
80626.47	2.34

Table 1: Variance explained by top few components

Implementation:

- Program for PCA: `scripts/pca.py - PCA.run()`
- Plotting: `scripts/question_1.ipynb - Sub-question i.`
- Data: `data/mnist/random_mnist_1000.npy`

Part (ii): Reconstruction

Figure 3: Reconstructed Images

Figure 3 shows the reconstruction using different dimensional representations, for 10 different images, (number of dimensions are indicated above the figure). As the dimensions used to represent increase the image becomes sharper and more recognizable. After the addition of around 130 components, subsequent changes have little effect, this in accordance with the observation in Table 2.

Index	Cumulative Variance	Percent Cumulative Variance
1	323224.16	9.37
10	1720491.78	49.89
50	2892267.12	83.87
100	3195398.73	92.66
132	3277532.96	95.04
180	3348406.63	97.10
300	3422916.11	99.26
784	3448467.69	100.00

Table 2: Cumulative Variance explained by the components

From Table 2 it is seen that more than 95% variance is explained by the first 132 components. Hence by a rule of thumb **dimension d=132** can be used suitably for a downstream task. Note that the cumulative variance changes drastically by

the addition of few components at the beginning (1-132), and changes little with the addition of subsequent components (133-784).

Implementation:

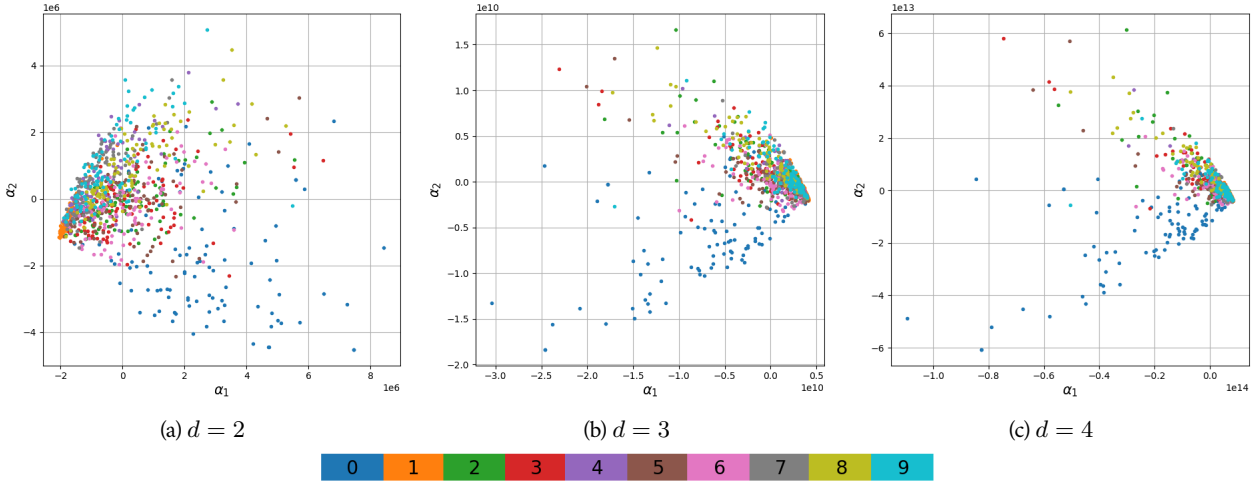
- Program for PCA reconstruction: `scripts/pca.py - PCA.reconstruct(self, index, order)`
- Plotting: `scripts/question_1.ipynb - Sub-question ii.`
- Data: `data/mnist/random_mnist_1000.npy`

Part (iii): Kernel PCA

A program is written to run the Kernel PCA algorithm on the same dataset as Part (i). Two different kernels are evaluated for the dataset.

(A) : Polynomial Kernel

$\kappa(x, y) = (1 + x^T y)^d$ for $d = \{2, 3, 4\}$



(a) $d = 2$ (b) $d = 3$ (c) $d = 4$

(d) Colormap for each class 0-9

Figure 4: Projection along first 2 principal components

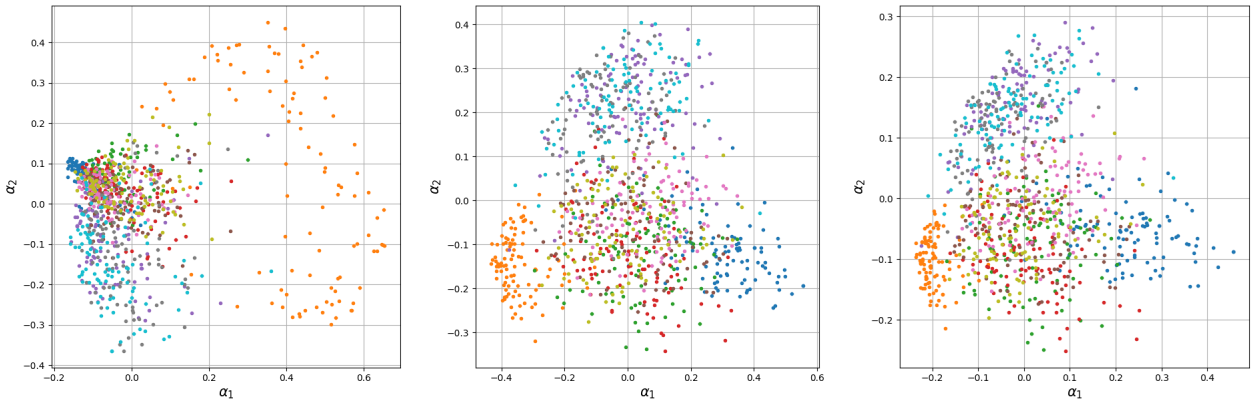
d	% Variance (α_1)	% Variance (α_2)	Components for 95%
2	8.87	5.19	500
3	8.44	4.68	549
4	7.91	4.44	506

Table 3: Variance explained by top 2 components and number of components required to explain 95% variance

Figure 4 shows the projection of the top 2 components for each kernel. It is observed that the projections bunch together about a single location. The variance explained by the top 2 components decreases as d increases. However, $d = 3$ requires more components to explain 95% variance.

(B) : Radial Basis Kernel

$\kappa(x, y) = \exp(\frac{-(x-y)^T(x-y)}{2\sigma^2})$ for $\sigma = \{1000, 2000, 4000\}$



(a) $\sigma = 1000$ (b) $\sigma = 2000$ (c) $\sigma = 4000$

(d) Colormap for classes 0-9

Figure 5: Projection along first 2 principal components

σ	% Variance (α_1)	% Variance (α_2)	Components for 95%
1.00	0.20	0.20	947
10.00	0.20	0.20	947
100.00	0.20	0.20	947
1000.00	2.84	1.72	834
1500.00	5.14	3.78	716
2000.00	6.61	5.09	594
3000.00	8.00	6.29	406
4000.00	8.57	6.77	295
10000.00	9.24	7.34	153
100000.00	9.37	7.45	132
1000000.00	9.37	7.45	132
10000000.00	9.37	7.45	132

Table 4: Variance explained by top 2 components and number of components required to explain 95% variance

Computations were carried out for 12 values of σ, results are presented in Table 4. Figure 5 shows the projection of the top 2 components for 3 of the 12 kernels tested.
It is observed that for values of $\sigma = \{1, 10, 100\}$ the value $(x-y)^T(x-y)$ is very large compared to σ^2, hence the value of $\kappa(x, y)$ is close to 0 for all x, y resulting in very less variance being explained by any single component, these kernels are unsuitable for any further analysis.

It is observed that for values of $\sigma = \{1e5, 1e6, 1e7\}$ the value $(x-y)^T(x-y)$ is very small compared to σ^2, hence the value of $\kappa(x, y)$ is approximated to to $1 - \frac{(x-y)^T(x-y)}{2\sigma^2}$ (By Taylor series) for all x, y resulting in this kernel showing similar characteristics to the usual inner product (Refer top 2 components of Table 1). Any further analysis of these kernels will be similar to the analysis of Part (i).

However, for values of $\sigma = 1000$ to 4000 significant variance is explained in the top 2 components. A trend is observed

where the variance explained by top 2 components increases, while the number of components required to explain 95% variance decreases.

Implementation:

- Program for Kernel PCA: `scripts/kernel_pca.py - kernelPCA.run()`
- Kernel (A): `scripts/kernel_pca.py - kernelPCA.polynomial_kernel()`
- Kernel (B): `scripts/kernel_pca.py - kernelPCA.radial_basis_kernel()`
- Plotting: `scripts/question_1.ipynb - Sub-question iii.`
- Data: `data/mnist/random_mnist_1000.npy`

Part (iv): Best Suited Kernel

A comparative study of Table 3 and Table 4 is conducted and the observations are listed among the following:

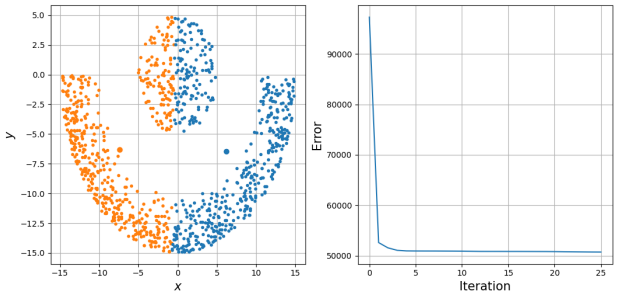
- The percent variance explained by the top 2 components for both the polynomial kernel ($d = 2, 3, 4$) and radial basis kernel (σ = 1000-4000) are nearly of the same magnitude (12-15%). Radial basis kernel explains slightly greater varaince in the top 2 components (1-2%).
- Although the magnitude of variance of the top 2 components of polynomial kernel are about $1e + 20$ times greater than the magnitudes of the variance of the top 2 components of the radial basis kernel.
- The components required to explain 95% variance for the polynomial kernel case (500-549) are greater in number than the components required to explain the same % variance for $\sigma = 3000,\ 4000$ (295-406) and less than the number of components required to explain the same % variance for $\sigma = 1500,\ 2000$ (594-716).
- A qualitative comparison of Figure 5 and Figure 4, shows that the data-points for all the classes appear bunched up in a single location for the polynomial kernel, where as the points for the radial basis kerenl are more spread out and display distinguishable structures.

From the above discussion, it is concluded that the radial basis kernel with $\sigma = 2000,\ 4000$ is a better kernel for down-stream tasks, owing to the structure displayed and variance explained, and thus being a better "compressed representation".

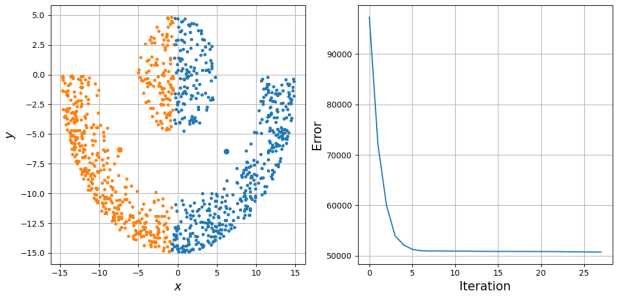
Question 2

Part (i): Running K-Means

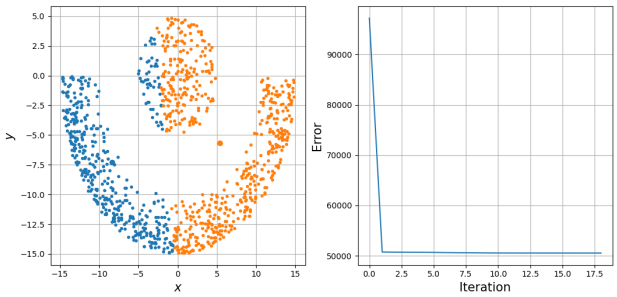
A program is written to run K-Means algorithm on the given dataset. Clustering is conducted with 5 random initializations. Results are presented in Figure 6.



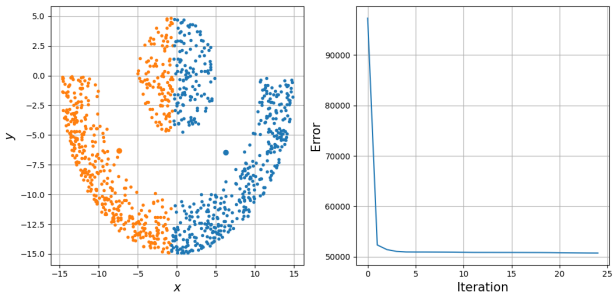
(a) Random seed 1



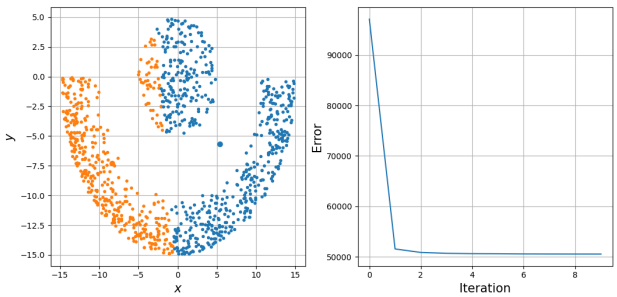
(b) Random seed 2



(c) Random seed 3



(d) Random seed 4



(e) Random seed 5

Figure 6: Five random initializations and corresponding Error plots

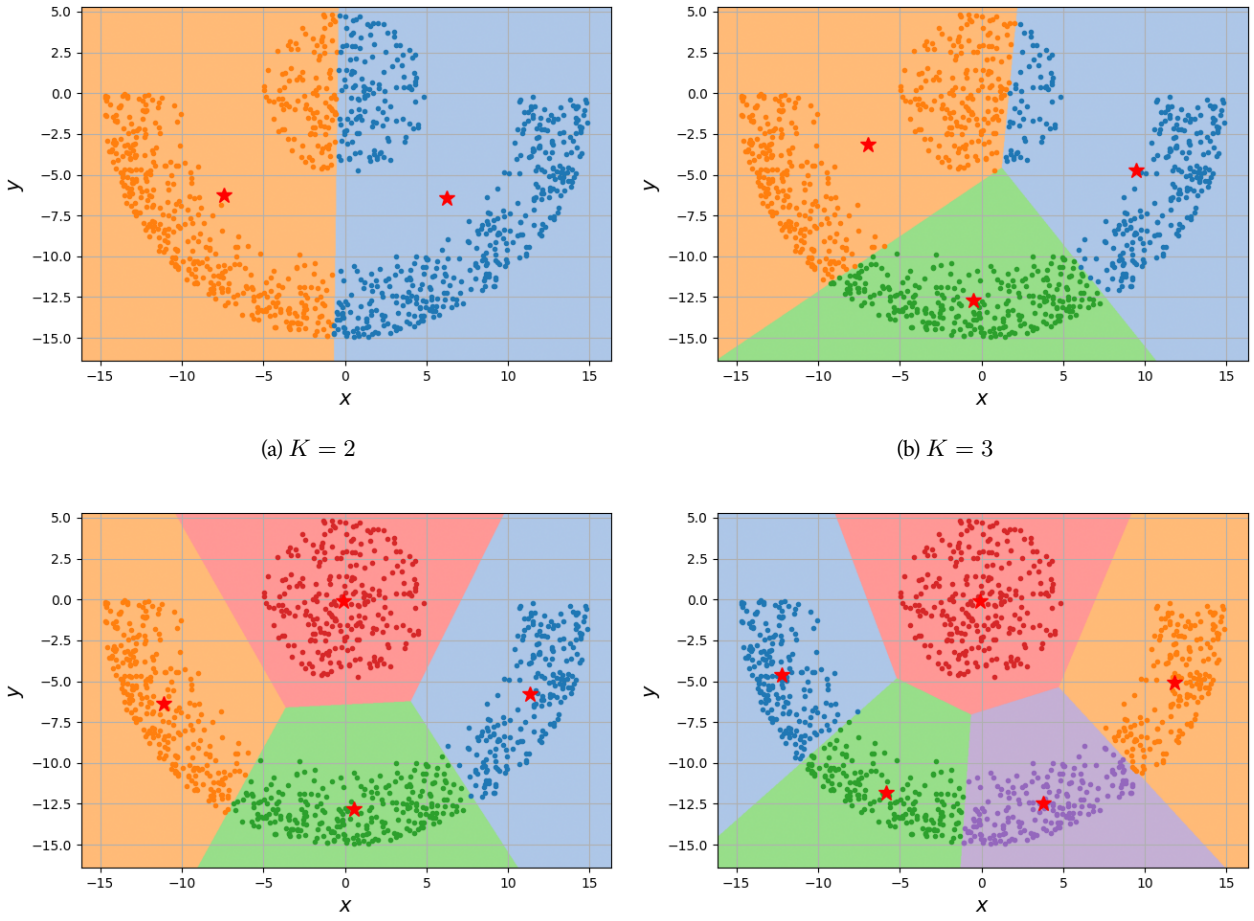
In all 5 cases, the initial error starts at around 97000 and settles to about 50700. The algorithm converges in iterations ranging from 10 to 25. It also observed that the clustering is based on the separation by a half-plane Voronoi region. Such clustering may be unsuitable for downstream tasks.

Implementation:

- Program for K-means: `scripts/k_means.py - Kmeans.run()`
- Plotting: `scripts/question_2.ipynb - Sub-question i.`
- Data: `data/cm_dataset_2 - cm_dataset.csv`

Part (ii): Voronoi Regions

A particular seed is fixed for random initialization. Cluster centers are obtained for $K = 2, 3, 4, 5$ (plotted as red stars). The corresponding Voronoi regions are plotted along with the clusters. It is observed that the Voronoi regions consist of intersections of half-planes. It is further verified that the K-means clustering algorithm is biased to this kind of linear separation of data-points.



(a) $K = 2$

(b) $K = 3$

(c) $K = 4$

(d) $K = 5$

Figure 7: Voronoi regions for $K = 2, 3, 4, 5$

Implementation:

- Program for Voronoi regions: `scripts/k_means.py - Kmeans.generate_voronoi()`
- Plotting: `scripts/question_2.ipynb - Sub-question ii.`
- Data: `data/cm_dataset_2 - cm_dataset.csv`

Part (iii): Spectral Relaxation of K-Means

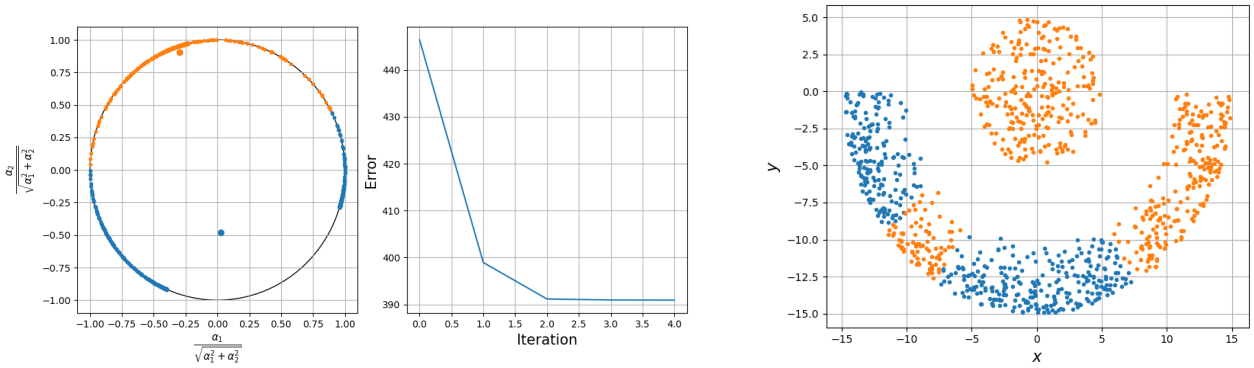
Spectral clustering algorithm is run on the same dataset as of Part (i). 3 different kernels are tested for clustering. The best results for each are presented in terms of the final clusters and error function. The best of these 3 is concluded at the end of this section.

(A) : Using Radial Basis Kernel

$\kappa(x, y) = \exp(\frac{-(x-y)^T(x-y)}{2\sigma^2})$ for $\sigma = 1.3$

Figure 8a shows the clustering of the normalized vectors according to the spectral clustering algorithm. Figure 8b shows the final clustering result obtained from this.

It is observed that some points are misclassified. However, unlike in Part (i), there is no observable bias to Voronoi regions.



(a) Clustering of Normalized vectors (b) Clustering Result

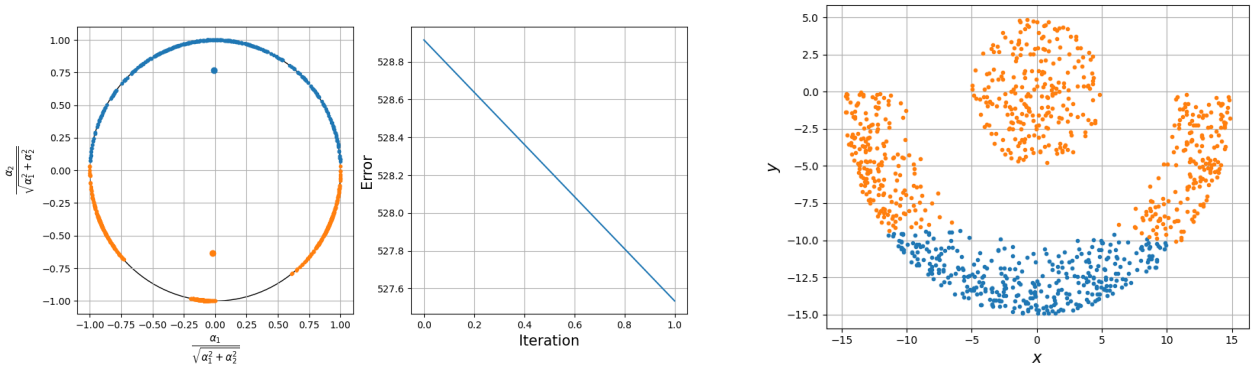
Figure 8: Spectral K-Means using Radial basis kernel

(B) : Using Polynomial Kernel

$\kappa(x, y) = (1 + x^T y)^d$ for $d = 3$

Figure 9a shows the clustering of the normalized vectors according to the spectral clustering algorithm. Figure 9b shows the final clustering result obtained from this.

It is observed that considerably more points are misclassified than in (A). Similar to (A), there is no observable bias to Voronoi regions. However, it may be interesting to note that the small cluster of points (exactly 250 points) visible at the bottom of Figure 9a actually correspond to central cluster of Figure 9b. But the spectral clustering algorithm does not find it optimal to cluster these points separately for the chosen kernel.



(a) Clustering of Normalized vectors (b) Clustering Result

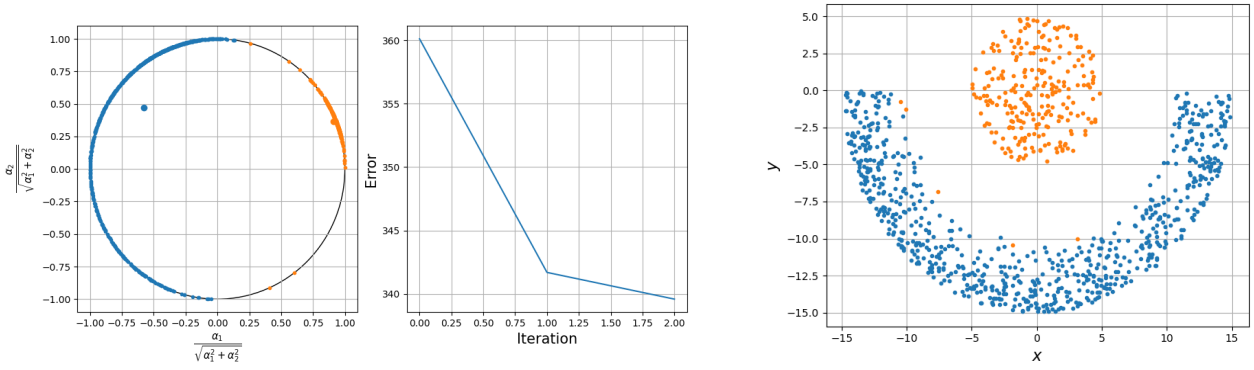
Figure 9: Spectral K-Means using Polynomial kernel

(C) : Using Custom Kernel

Appealing to the geometrical structure of the given dataset a new custom kernel is chosen for demonstration purposes. Here, the kernel is defined as follows,

$$
\begin{aligned}
x &= [x_1, x_2]^T \\
\phi(x) &= [x_1, x_2, \sqrt{{x_1}^2 + {x_2}^2}]^T && \text{Custom higher dimension mapping} \\
\kappa(x, y) &= \phi(x)^T \phi(y) && \text{Where, } x, y \in \mathbb{R}^2
\end{aligned}
$$

Note that $\kappa(x, y)$ is a valid kernel as the corresponding $\phi(x)$ is explicitly stated. It is observed that this kernel outperforms (A), (B) in clustering. A minute number of points are misclassified with this.



(a) Clustering of Normalized vectors

(b) Clustering Result

Figure 10: Spectral K-Means using Custom kernel

The motivation of kernel is that since the the clusters geometrically occupy circular/annular regions around the origin, each point's radial distance from origin is a good measure to cluster on. This fact is further demonstrated with Figure 11, where $\phi(x)$ is plotted for each x in 3 Dimensional space.

It is clearly verified that this high dimensional representation can be separated with help of a single plane (Voronoi region).

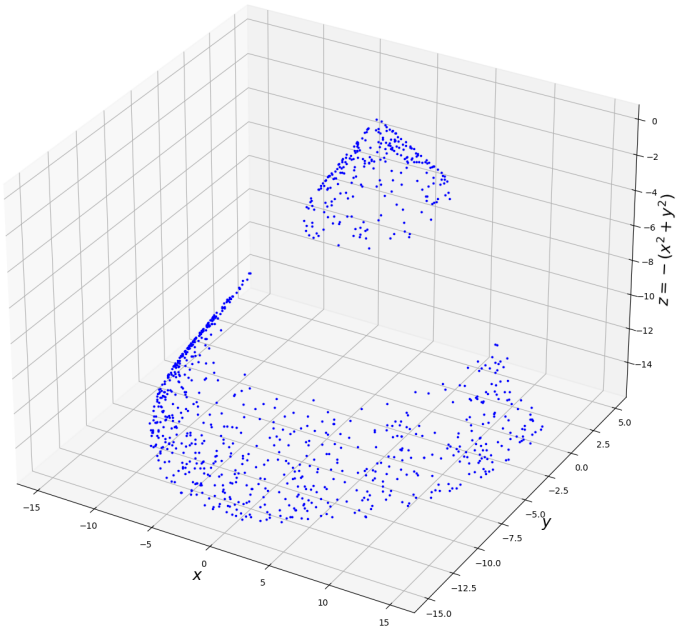
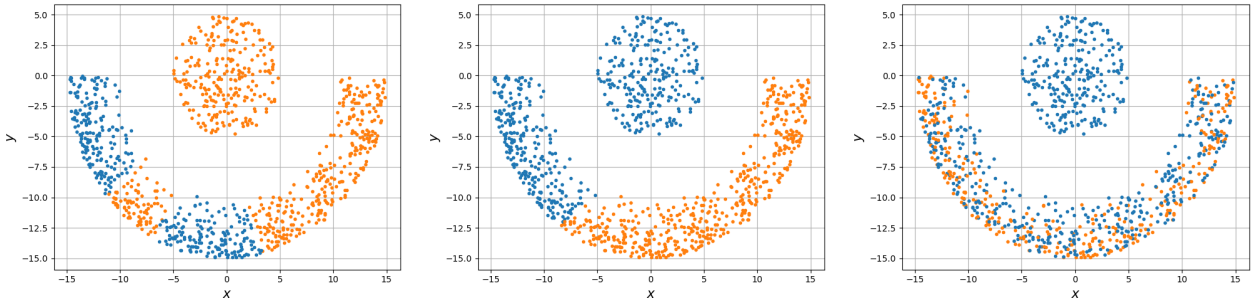


Figure 11: Visualization of the transform $\phi(x)$

Implementation:

- Program for spectral clustering algorithm: `scripts/spectral_k_means.py - spectralKmeans.run()`
- Kernel (A): `scripts/kernel_pca.py - kernelPCA.radial_basis_kernel()`
- Kernel (B): `scripts/kernel_pca.py - kernelPCA.polynomial_kernel()`
- Kernel (C): `scripts/kernel_pca.py - kernelPCA.custom_kernel()`
- Plotting: `scripts/question_2.ipynb - Sub-question iii.`
- Data: `data/cm_dataset_2 - cm_dataset.csv`

Part (iv): Alternate Mapping to Clusters



(a) Radial basis kernel $\sigma = 1.3$ (b) Polynomial kernel $d = 3$ (c) Custom kernel

Figure 12: Alternate mapping for 3 different kernels

It is observed that the alternate method suggested for mapping performs poorly on all instances in Figure 12 and especially poorly on Figure 12c.

In the 2 Dimensional case the proposed method is simply checking the greater component of the 2, or otherwise stated all eigenvector components are assigned a cluster based on which half-plane (described by the 2 components) they lie on (component-1 > component-2) or (component-2 > component-1).
The first insight that my be drawn from this is that,if we consider 2 points which have similar directions of the eigenvector components to be similar (as originally suggested by spectral clustering algorithm), then it can be observed that for 2 points which have similar eigenvector component directions then they will classified into 2 different clusters if corresponding components of one point are slightly greater than $45°$ and other slightly less than $45°$. This may result in poor performance.

A second insight (that verifies the first) may be drawn specific to this dataset is that, the cluster centres for Figure 8a, Figure 9a, occur away from the $45°$ line, hence the clustering result is close to the spectral clustering result. In contrary, one cluster center for Figure 10a occurs close to the $45°$ line, hence points which "actually" belong to the same cluster are misclassified.

Implementation:

- Program for spectral clustering algorithm: `scripts/spectral_k_means.py - spectralKmeans.run()`
- Program for Mapping: `scripts/spectral_k_means.py - spectralKmeans.max_clusters()`
- Plotting: `scripts/question_2.ipynb - Sub-question iv.`
- Data: `data/cm_dataset_2 - cm_dataset.csv`