CSE 847 (Spring 2021): Machine Learning— Homework 5

Instructor: Jiayu Zhou

Student Name: Kira Chan

The corresponding source matlab files can be accessed at `https://github.com/EvilJuiceBox/CSE847`

1 Clustering: K-means

1.1 Spectral relaxation vs kmeans

Spectral relaxation of the kmeans algorithm involve transforming the original m-dimensional space input data vector into a lower k-dimensional space. The data vector on a lower dimension is then clustered by applying the regular k-means method to obtain the clusters. In the kmeans implementation, we are trying to find a vector Y in the optimisation objective that splits the clusters into their separate rows, with zeroes filling out the rest of the entries. However, in practice, this matrix is extremely hard to find. Thus, using spectral relaxation, we are attempting to find a similar matrix but with potentially small entries instead of zero. Figure 1 shows the discussion with Dr. Zhou during his office hour. The matrix above shows the regular k means implementation, and the matrix with the SVD decomposition is the spectral relaxation method of the

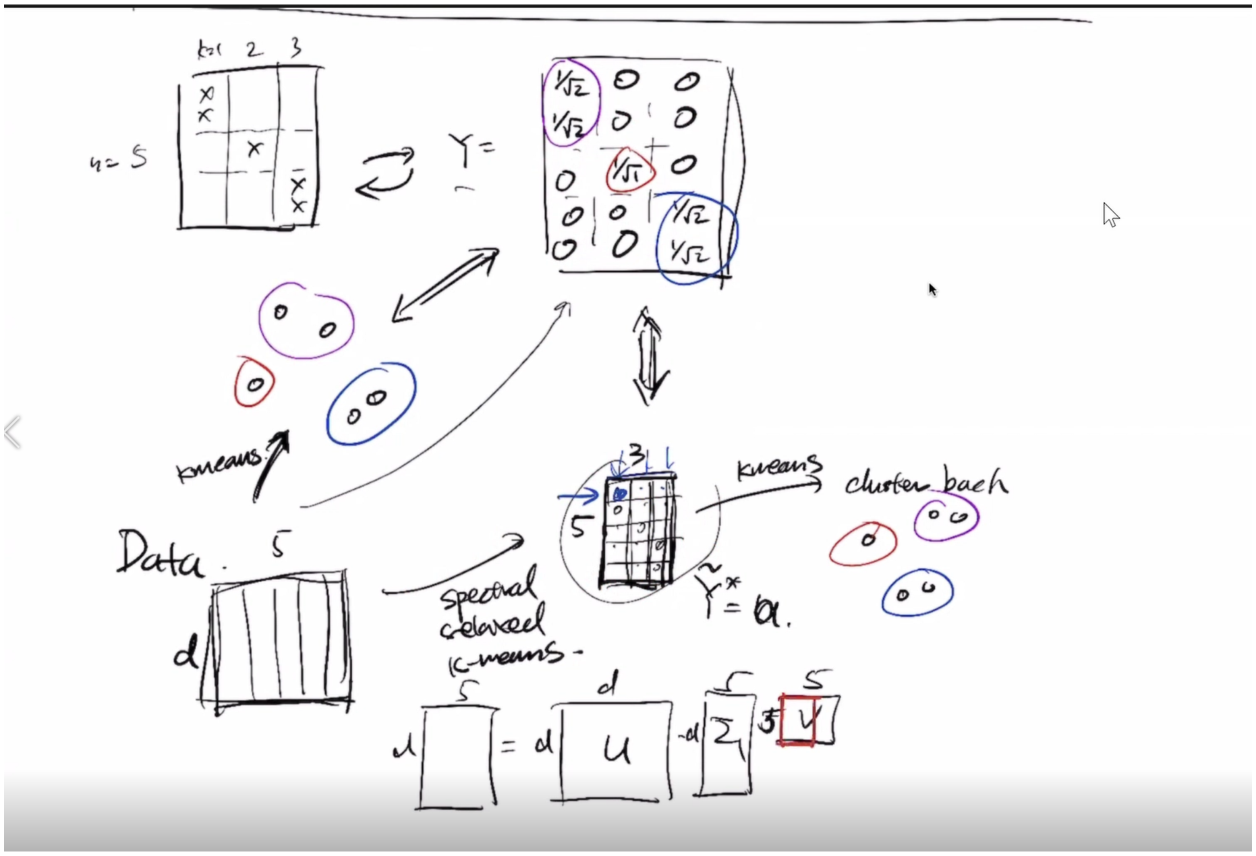

Figure 1: Regular kmeans vs Spectral relaxation in office hours

1.2 kmeans implementation

The alternating procedure k-means function has been implemented in python, tested with some sample data point, and plotted with the matlibplot library. The file used is **kmeancluster.py**. We have tried to implement the solution in Matlab, however, the array implementation was causing quite a bit of issue. So after half a day or debugging, we finally gave up and used python.

The algorithm works as follows. First, the user inputs the data and k (the number of desired clusters). Three random points of the input data set are chosen as the initial centroids. Next, each point of data on the input data set is then assigned to the cluster based on the mean square distance between the two points. Argmin is used to find the centroid with the closest distance to the point. Next, the mean of each cluster is recomputed as the new centroids. The process repeats until the position of the centroids no longer changes. We consider the algorithm as converged. Figure 2 shows a sample input where the algorithm converged and clustered the data into 3 clusters. Data points are denoted by circles, where starts denote the centroid values. Note that the centroids may or may not be actual dataset points. Finally, we tested the approach using 100 randomly generated data points with the algorithm. Figure 3 shows the algorithm results.

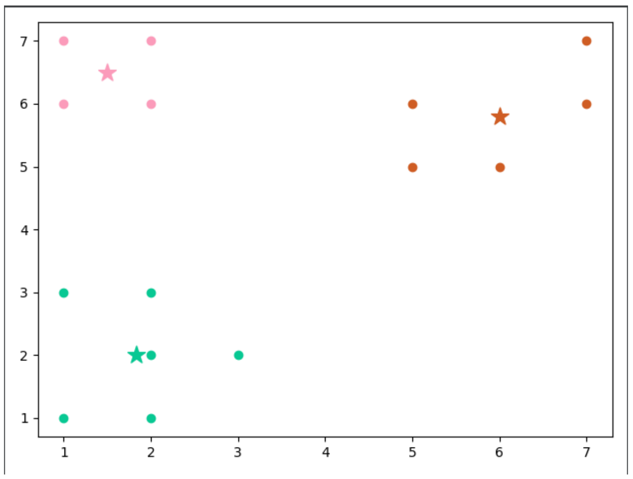

Figure 2: An example of the clustering algorithm output.

To fully demonstrate each generation of the algorithm, Figure 4 shows the centroid shifting its position over time. The colour scheme of the chosen points are randomly generated, and have no significance meaning other than to show points that belong to a centroid.

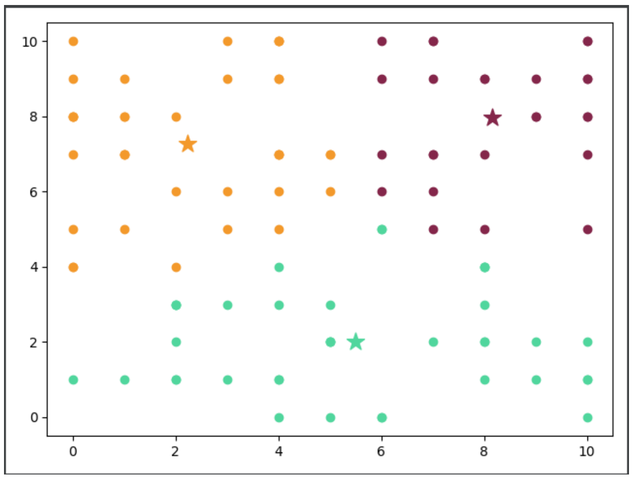

Figure 3: An example of the complex clustering algorithm output.

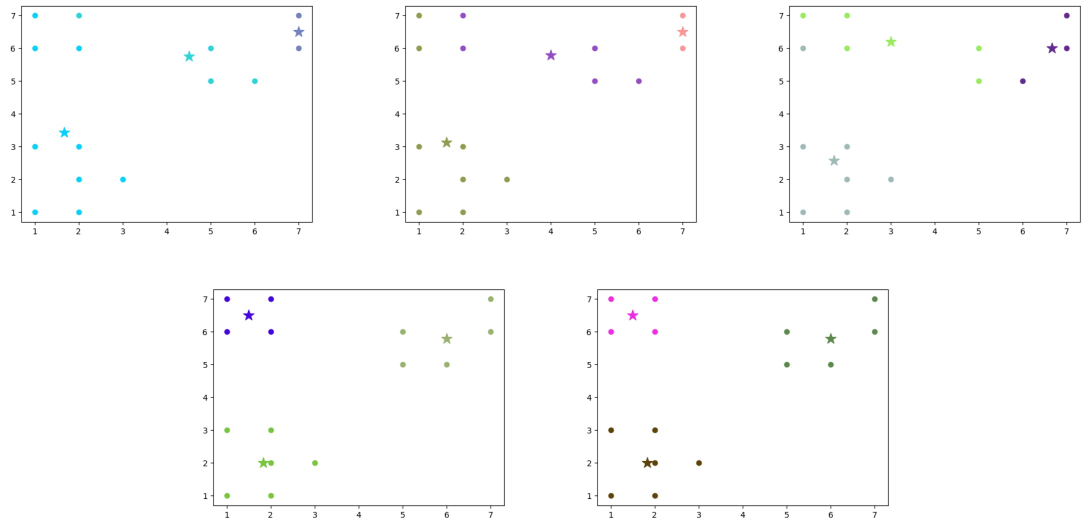

Figure 4: Step-by-step implementation of clustering

For the spectral relaxation, we used Matlab to generate the example dataset and use SVD to derive the Y used in the algorithm. The source code can be found in **spectralkmean.m**. The default k-means algorithm is used to compare the effects of spectral relaxation. For the first experiment, a simple dataset is created manually. Figure 5 shows the effects of spectral kmeans vs regular kmeans. Empirically, both methods produced the same clustering results. To further test the effectiveness of the spectral relaxation method, we use a more complex dataset with fifty randomly generated datapoints. Figure 6 shows the experiment result. Once again, we can see that both kmeans and spectral relaxation implementaion of kmeans produce similar results.

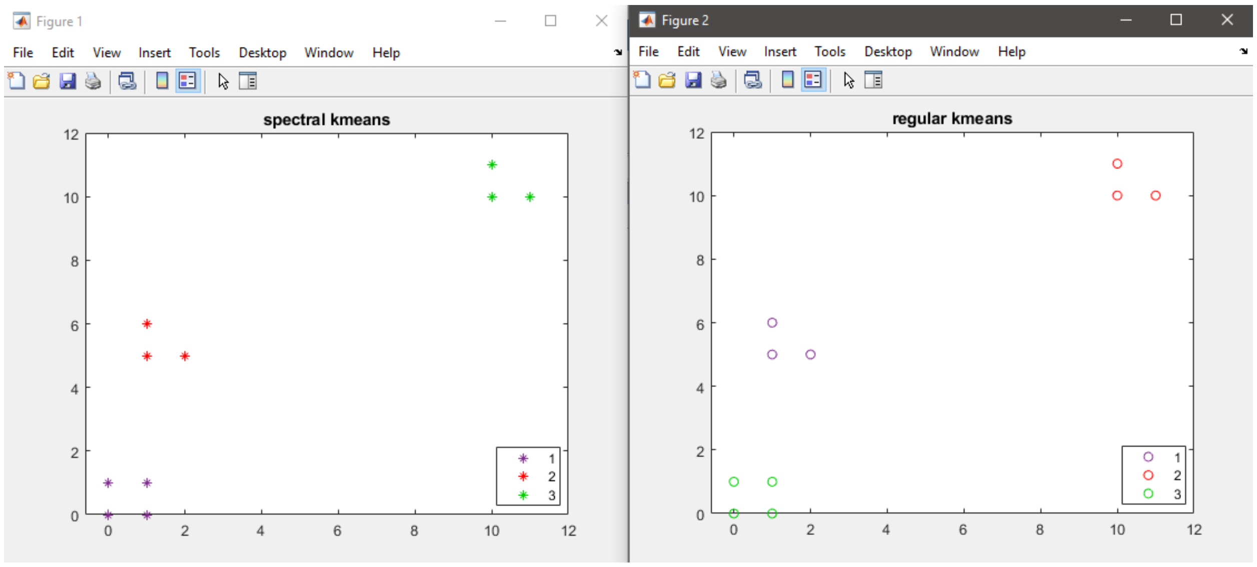

Figure 5: Regular k means vs Spectral relaxation visualisation

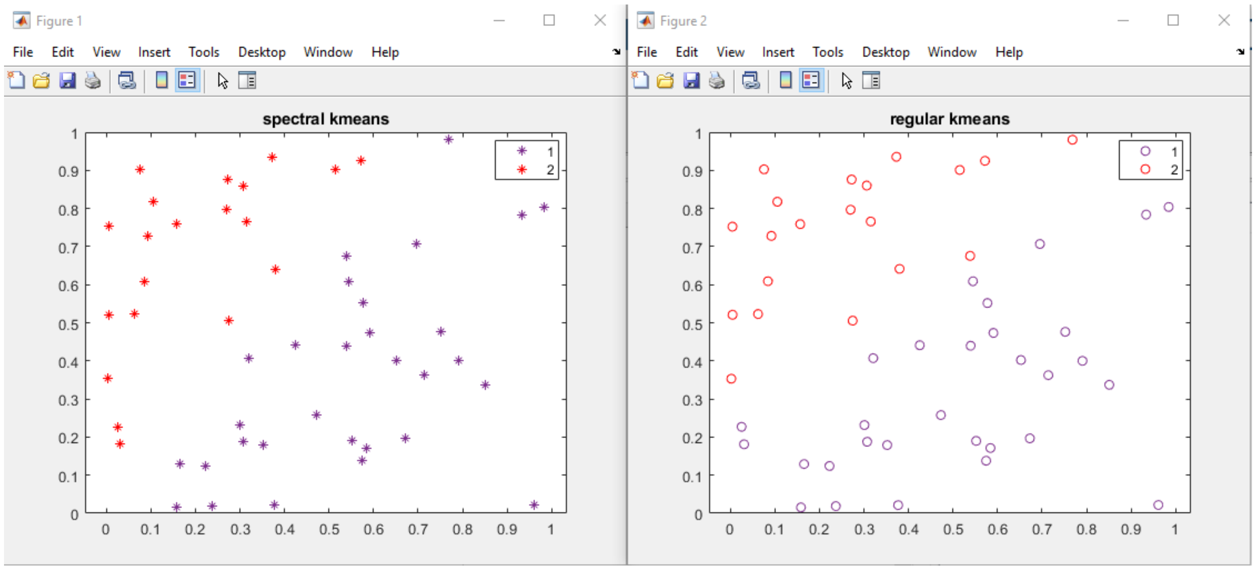

Figure 6: Regular k means vs Spectral relaxation visualisation 2

2 Principal Component Analysis

2.1 Question 1

Figure 7 shows the 2d plot of the provided points. If we apply PCA to the points, then intuitively the first PC for the plot is the line that passes through all the data point. This principal component has the least amount of distance between the data point to the principal component line (no mse, since every point is on the line).

The second PC is then the orthogonal line of the first PC. Of course, we will first have to shift the data such that the mean is back to the origin first.

Figure 8 shows the principal components for the points, where the red line indicates the first PC, and the blue line indicate the second PC. The plot is generated using the **pca.m** file in the github directory.

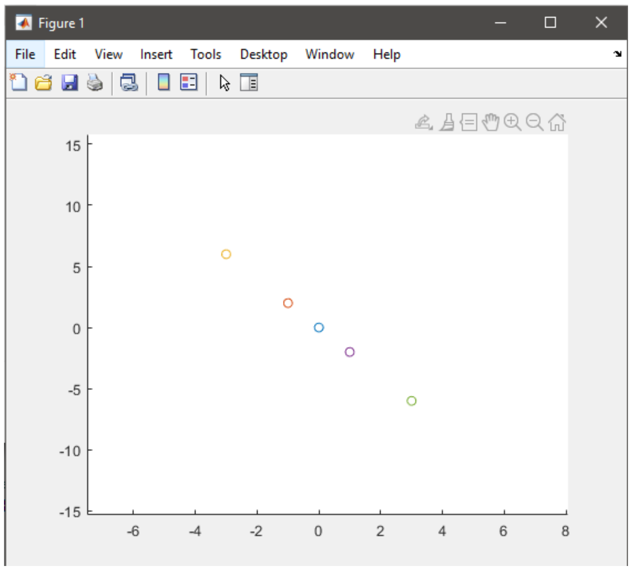

Figure 7: Plot of data points in 2d space

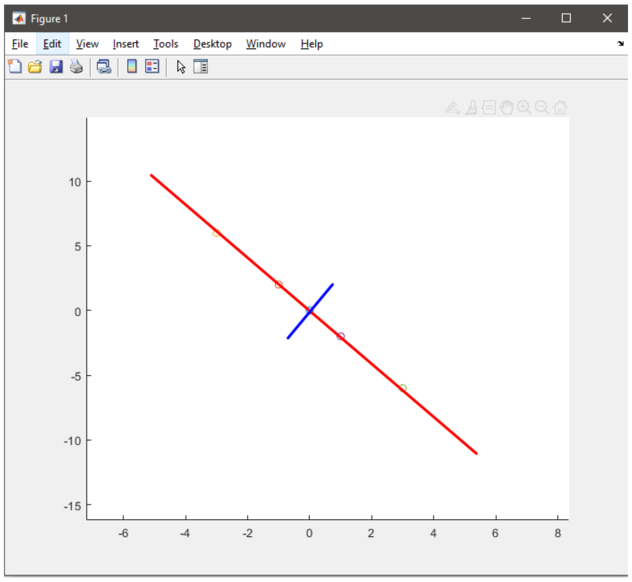

Figure 8: Plot with PCA lines

2.2 2. USPS dataset PCA implementation

For problem 2 of the PCA question, the implemenation is available at **pcasvd.m**. In the source code, we first load the dataset and shift the dataset to the original by subtracting the rows with the mean of the rows. Next, the SVD method is called to obtain the $U\Sigma V$ vectors.

To show the significance or effect of principal component analysis, we have used two plots. Figure 9 shows the plot of the magnitude of the eigenvalues. Empirically, the first eigenvalue has a large effect on the input dataset. The effect of the eigenvalues quickly decreased by the 50th principal component. The plot on the right indicate the percentage of variance accounted by the number of principal components. When 50 PCs are considered, 60% of the variance has been accounted for.

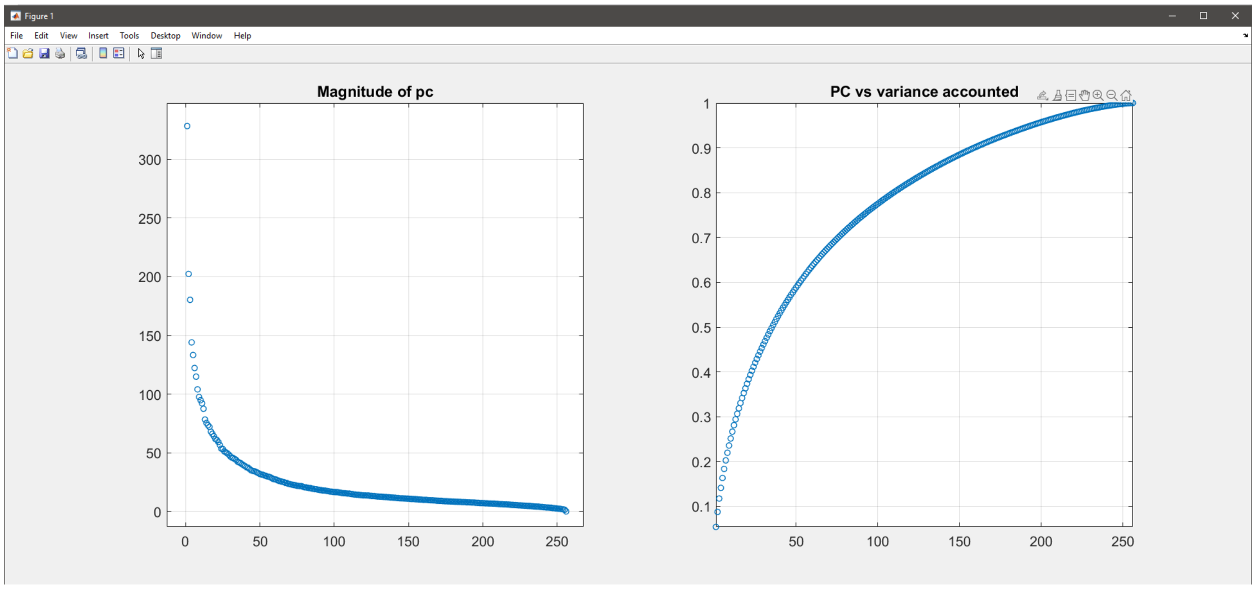

Figure 9: Significance of PCA

To test the feasibility of PCA, Figure 10 to 13 shows the data compression technique applied to a few numbers. For each number, the original image is shown on the right, then 10, 50, 100, 200 PC were used to show the data compression. As shown in the image, even with only 10 PCs, a human eye can still identify the digit clearly from the resulting image. The number used in the images are arbitrary chosen, and works with any row input from the original data matrix. The method getReducedImage(A, u, sigma, v, #) is used to generated each image automatically. # represents the row entry for the image.

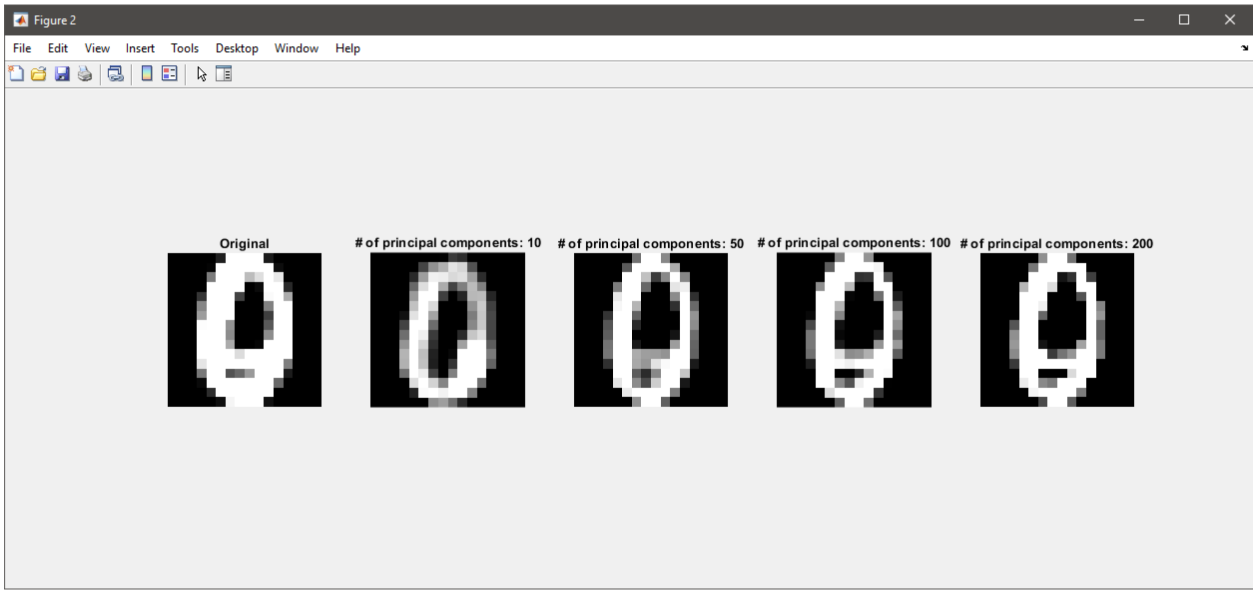
Figure 2
File Edit View Insert Tools Desktop Window Help
Original
# of principal components: 10
# of principal components: 50
# of principal components: 100
# of principal components: 200

Figure 10: PCA applied to the number 0

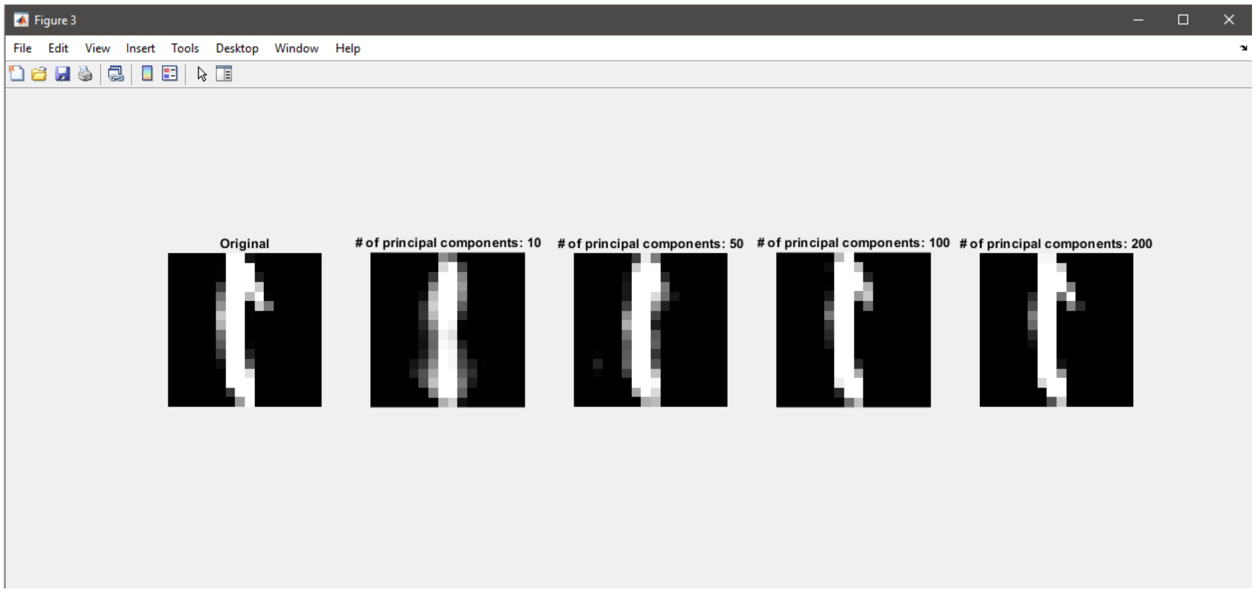
Figure 3
File Edit View Insert Tools Desktop Window Help
Original
# of principal components: 10
# of principal components: 50
# of principal components: 100
# of principal components: 200

Figure 11: PCA applied to the number 1

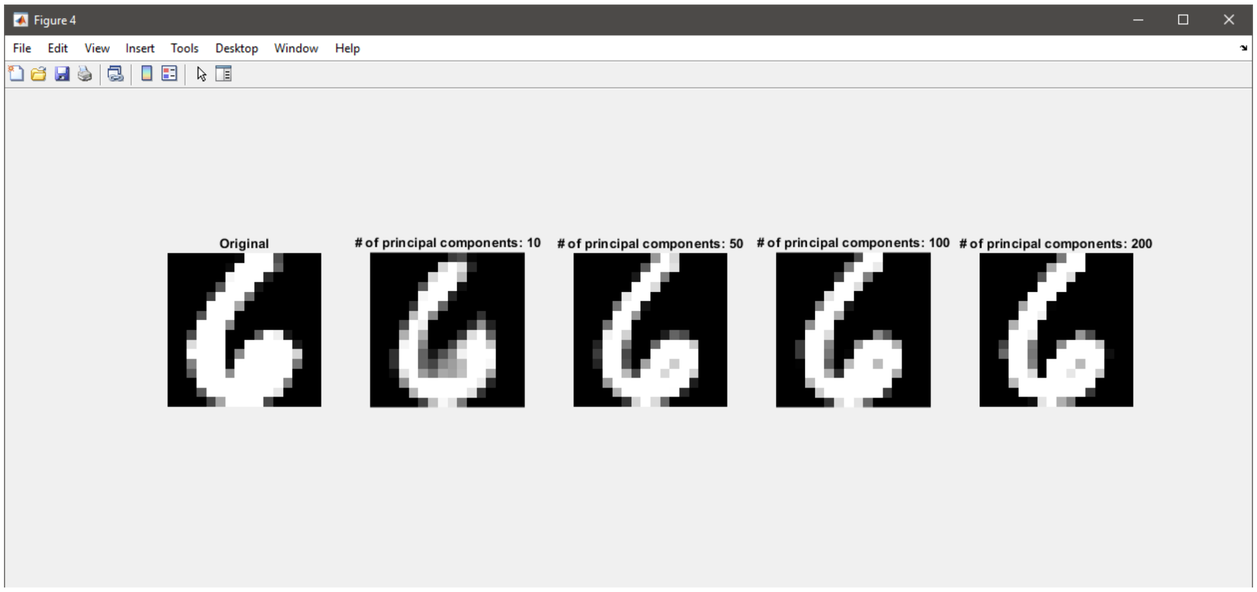
Figure 4
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original
# of principal components: 10
# of principal components: 50
# of principal components: 100
# of principal components: 200

Figure 12: PCA applied to the number 6

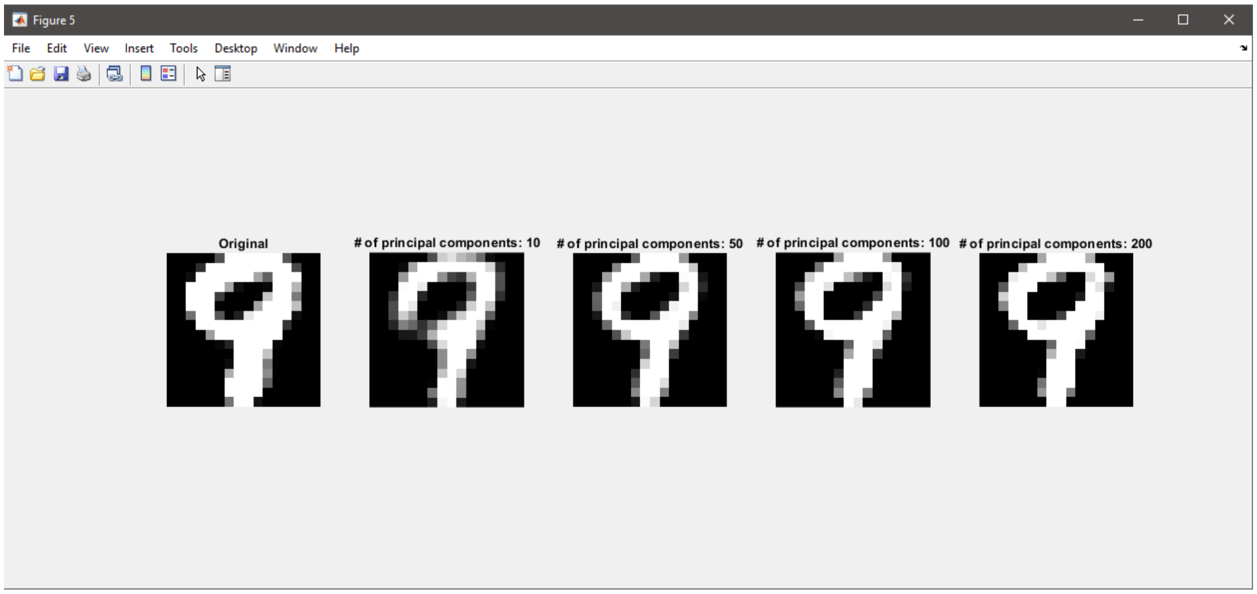
Figure 5
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original
# of principal components: 10
# of principal components: 50
# of principal components: 100
# of principal components: 200

Figure 13: PCA applied to the number 9