Multivariate Gaussian and PCA

Assignment 2

Ankan Sarkar (210050013)
Soham Joshi (210051004)

Contents

Introduction

This is the report of Assignment-2 of the course CS215[Awa22] offered in the autumn semester of '22 in IIT Bombay, by Prof. Suyash Awate. In this report, we will cover the solutions, along with empricial observations and how well they align with the existing theory. We have coded this assignment using MATLAB. The entire code can be accessed in this repository[22] under the folder "code", the graphs and results are given under the folder "results" and this report can be found under the folder "report". Sections $1, \cdots, 6$ of this report correspond to questions $1, \cdots, 6$ of the problem statement. So without further ado, let's start exploring.

1 Sampling within a Euclidean Plane

In this section, we shall deal with method to sample points uniformly within an ellipse and a triangle.

1.1 Ellipse

The algorithm to sample points uniformly in an ellipse relies upon the fact that $\sqrt{rand()}$ samples points linearly between 0 and 1. The variable names correspond as :

1. a : semi-major axis of the ellipse
2. b : semi-minor axis of the ellipse
3. t : Angle of rotation corresponding to a point in the auxiliary circle

Algorithm 1 Random point in an elipse

if $a > b$ **then**
  $r \leftarrow a \times \sqrt{rand()}$
  $t \leftarrow floor(2\pi \times rand())$
  $x \leftarrow a \times cos(t)$
  $y \leftarrow b \times sin(t)$
else
  $r \leftarrow b$
  $t \leftarrow floor(2\pi \times rand())$
  $x \leftarrow a \times cos(t)$
  $y \leftarrow b \times sin(t)$
end if
return x, y

Now, implementing the algorithm in matlab, we have got the following histogram

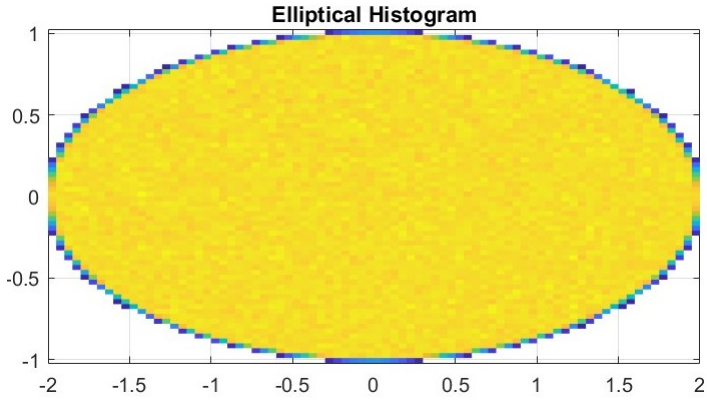


Figure 1: Elliptical Histogram

1.2 Triangle

Now, our aim is to sample points uniformly from a triangle with vertices at $(0,0),(\pi,0),(\frac{\pi}{3},e)$. In order to implement this, the key idea of the algorithm is again that $\sqrt{rand()}$ samples points linearly. Hence, we divide the triangle into two parts along the line $x = \frac{\pi}{3}$ and

1. Choose a sub-triangle with probability proportional to the area
2. Sample points uniformly in that part using the linear distribution

The algorithm for the same is as follows :

Algorithm 2 Random point in a Triangle

if $rand() > 1/3$ **then**
  $x \leftarrow \frac{\pi}{3}\sqrt{rand()}$
  $y \leftarrow x \times rand() \times e \times \frac{3}{\pi}$
else
  $x \leftarrow \frac{2\pi}{3}\sqrt{rand()}$
  $y \leftarrow x \times rand() \times e \times \frac{3}{2\pi}$
  $x \leftarrow \pi - x$
end if
return x, y

Upon implementing this algorithm, the empirical graph is a near-uniform distribution as shown below :

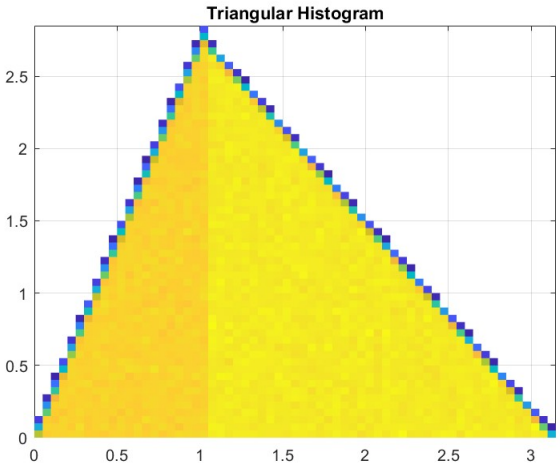


Figure 2: Triangular Histogram

2 Multivariate Gaussian

Here, we generate N points (N taking values $10, 10^2, 10^3, 10^4, 10^5$) from a multivariate 2D Gaussian probability density function with mean $\mu = [1, 2]^{'}$ and a covariance matrix $C = \begin{pmatrix} 1.6250 & -1.9486 \\ -1.9486 & 3.8750 \end{pmatrix}$

2.1 Sampling points

In order to sample points from this 2D gaussian distribution, we use the following approach :

$$X = AW + \mu \tag{1}$$

$$C = A.A^T \tag{2}$$

Hence, we obtain the matrix A from C via Cholesky Decomposition[Wik22], and since W is a 2×1 matrix of Random variables with standard normal distribution, and μ is given, we can find X and hence, sample X by just obtaining values of W. (Matlab offers inbuilt functionality for a standard gaussian random variable). Hence, we proceed with the following algorithm :

2.2 Sample mean and covariance

For each value of N , we repeated the experiment 100 times, and plotted a boxplot of the error between the true mean μ and the ML estimate $\hat{\mu}_N$, where

Algorithm 3 Random point in a 2D Gaussian

procedure GAUSSIAN(μ, C)
  $[V, D] \leftarrow eig(C)$
  $A \leftarrow V\sqrt{D}$
  $W \leftarrow randn(2, 1)$
  $X \leftarrow AW + \mu$
return X
end procedure

the error measure is :

$$error = \frac{\|\mu - \hat{\mu}_N\|_2}{\|\mu\|_2} \tag{3}$$

Upon this procedure, we got the following result :

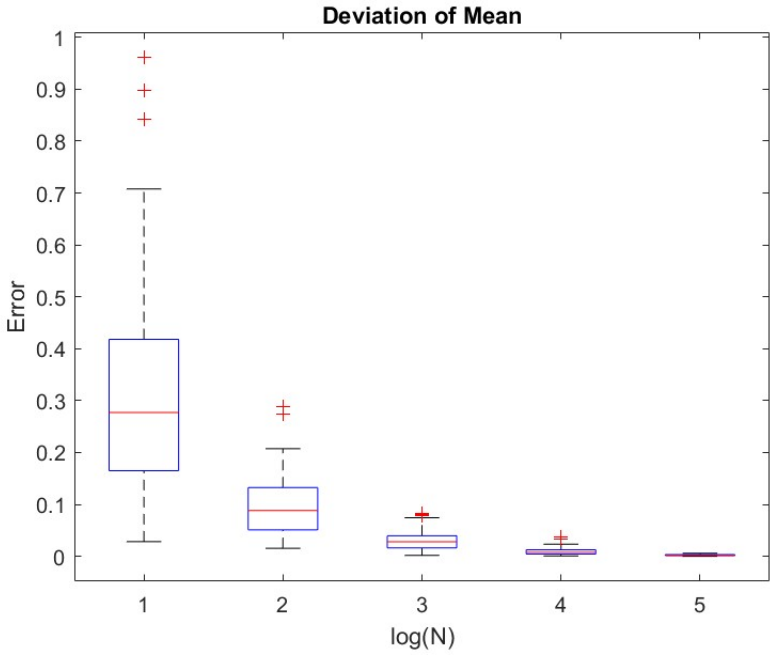


Figure 3: Error in Mean

Similarly, we repeat the procedure with the covariance matrix where the measure of the error is given as :

$$error = \frac{\|C - \hat{C}_N\|_{Fro}}{\|C\|_{Fro}} \tag{4}$$

Upon measuring errors for all values of N, we get the following boxplot :

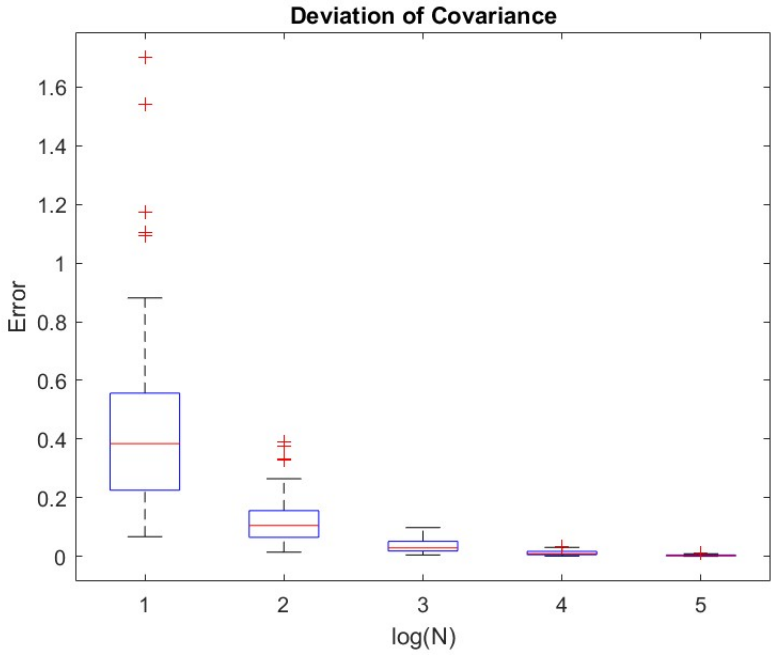
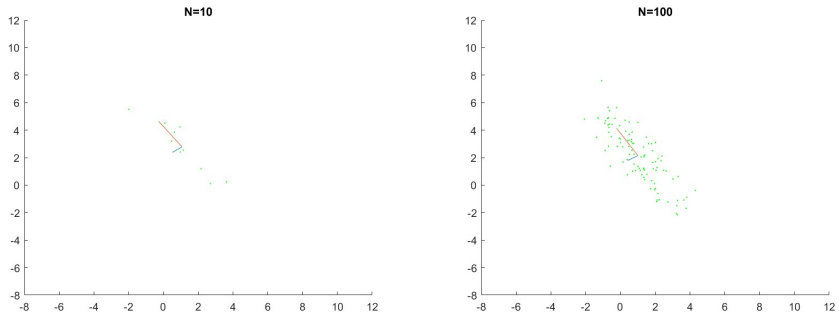


Figure 4: Error in Covariance

2.3 PCA

Now, we apply principal component analysis on the multivariate gaussian. In order to do so, we found the principal eigenvectors of the sample covariance matrix in decreasing order of eigenvectors, and showed the principal modes of variation of the data by plotting a line starting at the empirical mean and going a distance equal to the empirical eigenvalue's square root along a direction given by the empirical eigenvector.
We obtained the following scatter plots :



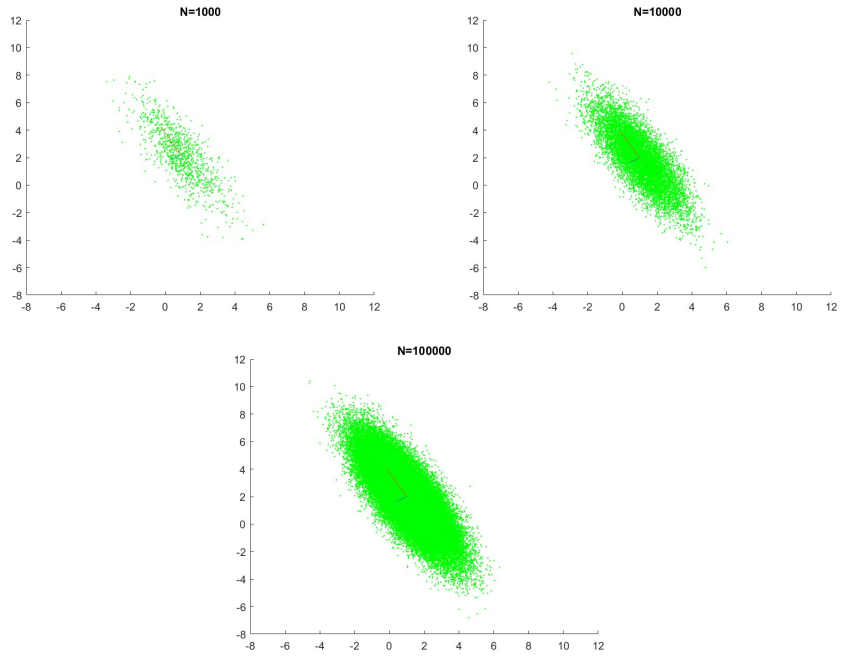


Figure 5: Scatter Plots for different values of N

Hence, we conclude the analysis of this 2D multivariate gaussian.

3 PCA and Hyperplane Fitting

The dataset for this question is the set of points of the form $(x, y) \in \mathbb{R}^2$ in the file "points2D_Set1.mat" in the folder "data". We assume each observation (x, y) is drawn independently from the joint probability density function $P(X, Y)$ of random variables X and Y.

3.1 Estimating linear relationship using PCA

Here we use principal component analysis (PCA) to best approximate a linear relationship between random variables X and Y. The method for the same is as follows :

1. Find the maximum eigenvalue and the corresponding eigenvector v of the joint distribution (X, Y)
2. Find the sample mean of the joint distribution
3. Plot the line with one point as the mean $slops = \frac{v_y}{v_x}$

Algorithm 4 Line fitting

procedure LINE(X)
$\mu \leftarrow sum(X, 2)/size(X)$
$C \leftarrow (X - \mu).(X - \mu)^{'}$
$[V, D] \leftarrow eig(C)$
$v \leftarrow V(argmax(D))$
$slope \leftarrow v(2)/v(1)$
$Plot(\mu, slope)$
end procedure

Algorithmically this corresponds to :

Upon implementing this in matlab, we get the following scatter plot :

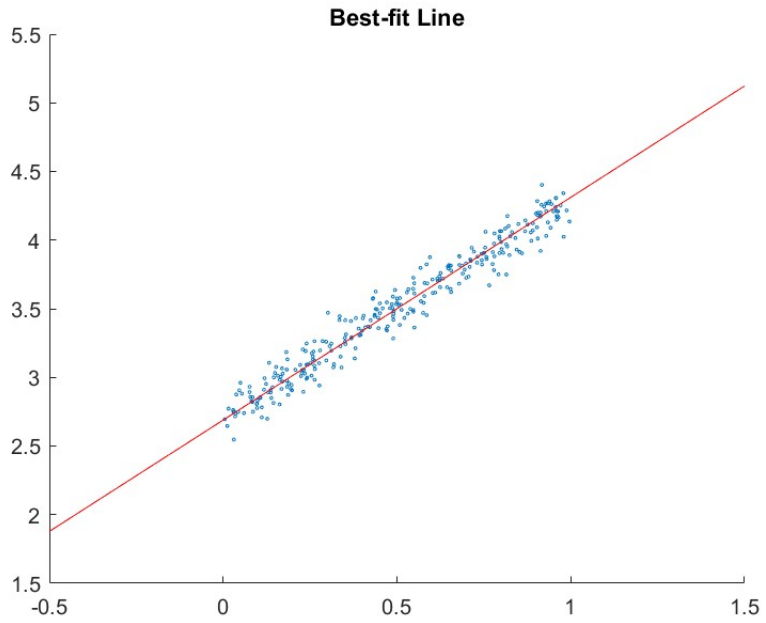


Figure 6: Scatter Plot of the 1^{st} Dataset

Similarly, we repeated this experiment with another dataset "points2D_Set2.mat" in the folder "data". We got another scatter plot as shown below :

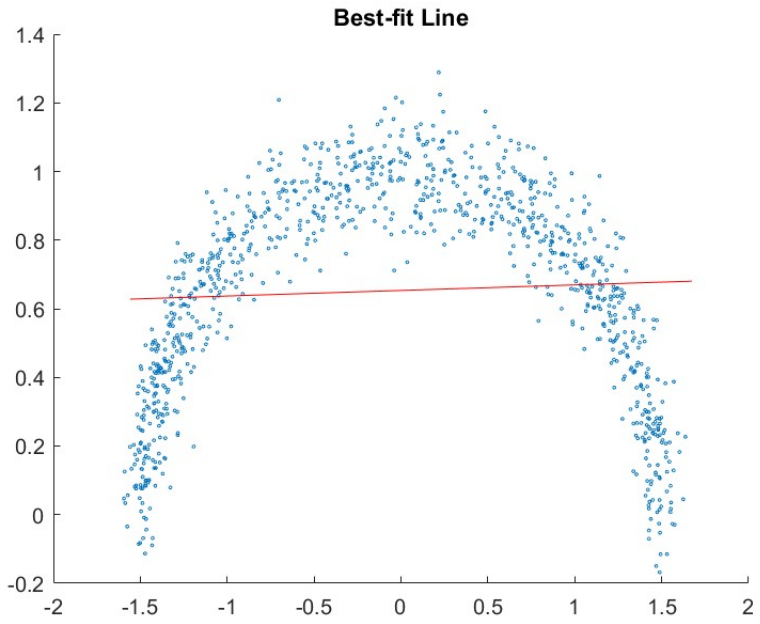


Figure 7: Scatter Plot of the 2^{nd} Dataset

This dataset is crescent shaped and hence, linear fitting is not possible. But why is the fitted line horizontal? This happens purely as a mathematical feature of the principal component analysis, since projection along the horizontal direction yields the maximum variance. Hence, quality of a fit to a crescent shaped dataset is relatively poor, as compared to the nearly linear dataset.

4 Principal Component Analysis (PCA)

In this question, we have used the MNIST dataset, present in the folder "data" and stored as "mnist.mat", using the entire training set of 60000 examples

4.1 Computing mean

For this purpose, we put each image into one of the vectors depending on its label, and then we take the mean of each of the vectors of images. Visualising the same, we obtain the following images :

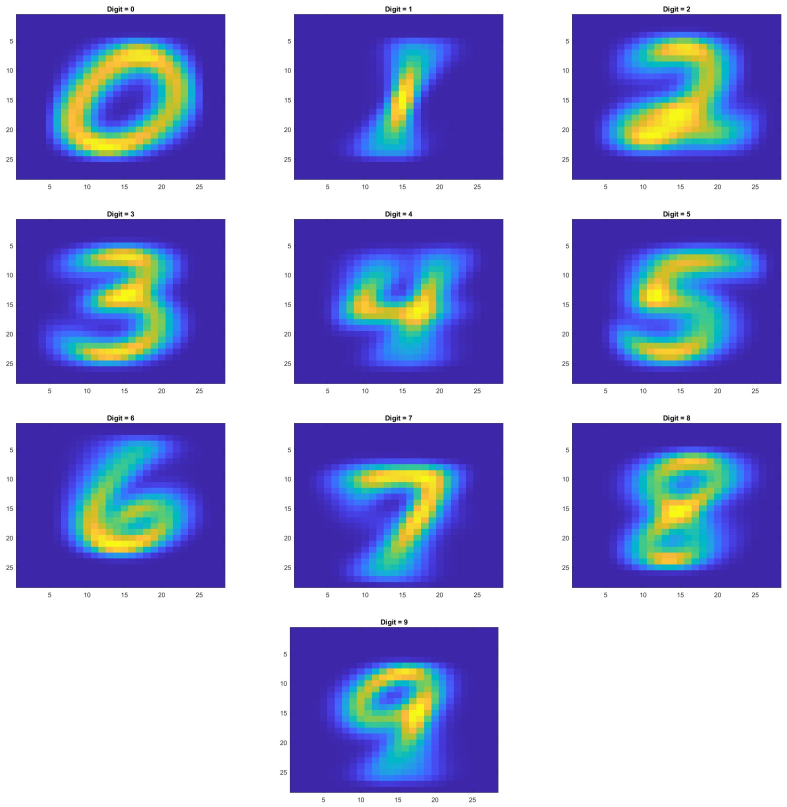


Figure 8: The Mean Digits

Hence, these images give us an insight about what an "average" digit looks like.

4.2 Computing covariance

Similar to the procedure of mean, we now compute covariance, but instead of taking mean of each vector, we compute the matrix $\frac{X.X^T}{N}$, where X is the vector of images with the same digit, and N is the number of images in the vector X. Now, using the spectral decomposition, we can obtain :

$$[V, D] = eig(C) \tag{5}$$

Plotting the eigenvalues decreasing order, we get the following plot :

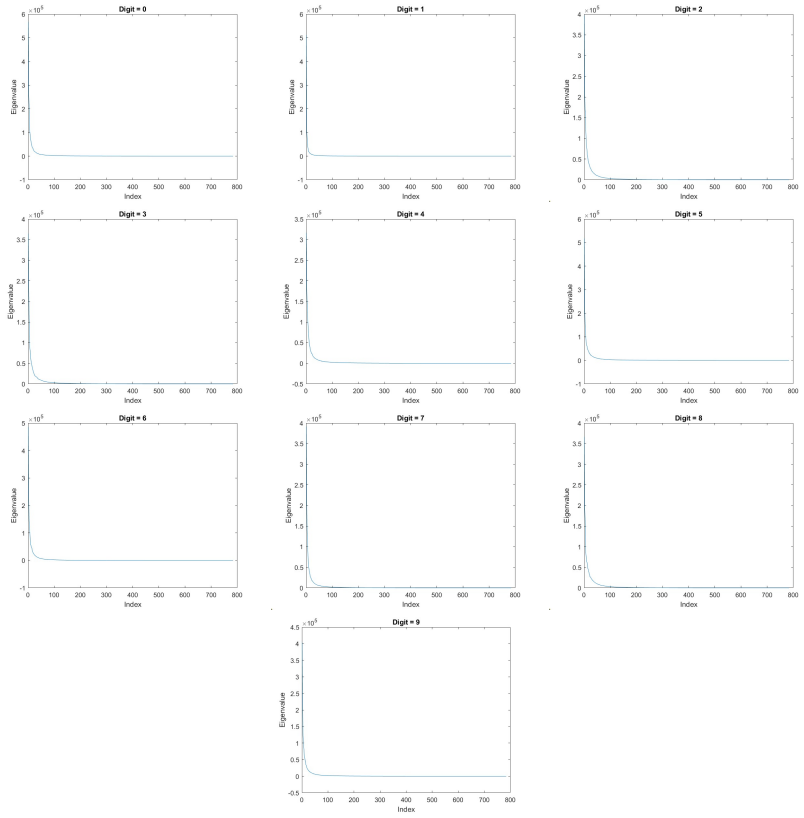


Figure 9: The Eigenvalues of the Digits

As we can clearly see, the principle modes of variation are far less than $28^2 = 784$, hence the features of the image occupy far less dimensions than what are available. This is to be expected since every digit has certain characteristics which roughly speaking remain conserved. From the empirical data itself, we observe that the digits occupy roughly 50 spatial dimensions each.

4.3 Principal Modes of Variation

The principal mode of variation is calculated by simply finding the eigenvector corresponding to the largest eigenvalue of the covariance matrix C. Let this eigenvalue and eigenvector be λ_1, v_1. respectively. Now, we plot three vectors for each digit, namely

1. $\mu - \sqrt{\lambda_1} v_1$
2. μ
3. $\mu + \sqrt{\lambda_1} v_1$

Upon computing these vectors and plotting them, we obtained the following results :

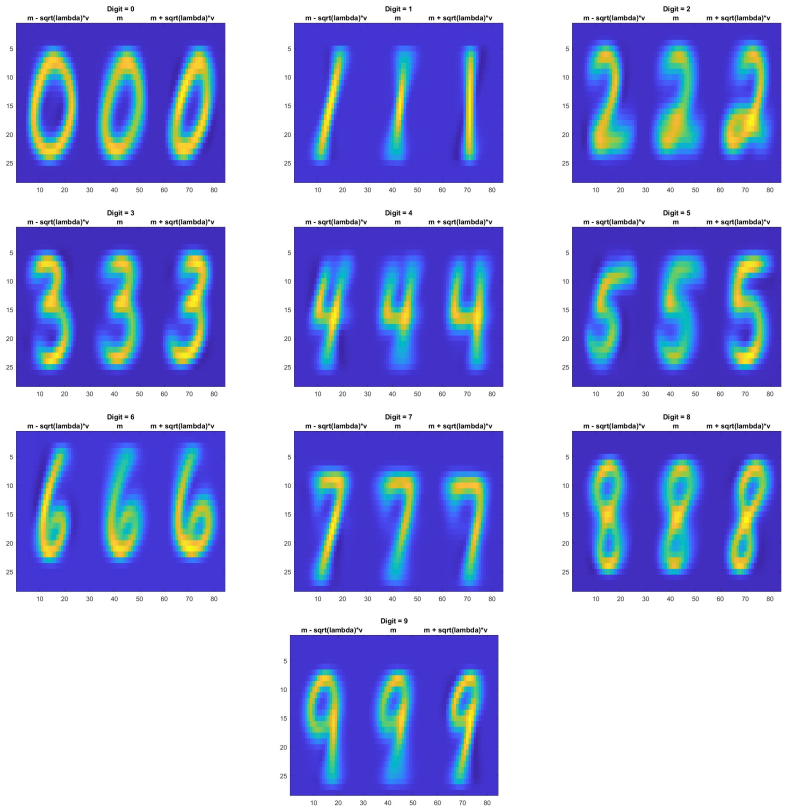


Figure 10: The Triplet of Images

As we can see, the left image has its principal features faded the most, while the right image has its principal features bolded. In order to achieve this, we have pre-processed the data by **rescaling** the data to $[0, 1]$ including the mean and the eigenvectors This has been done in order to avoid irregularities due to different intensities of different images.
In particular for the digit 1, we can see that the eigenvector has to be composed of pixels lying on a vertical line, hence upon performing this operations, the **features** of the digit start to **fade** away in one picture and get **emphasized** in the other.

5 Principal Component Analysis (PCA) for Dimensionality Reduction

In this question, we have used the MNIST dataset, present in the folder "data" and stored as "mnist.mat", using the entire training set of 60000 examples. Here, we to re-represented the images using only 84 coordinates (instead of 28 2 = 784) in an 84-dimensional basis for some 84-dimensional hyperplane within the original Euclidean space, such that the chosen 84-dimensional hyperplane maximized the total dispersion of the original data (for the chosen digit) within the hyperplane.

In order to do so, we first computed the covariance matrix for each digit, based on the label of the digit, hence obtaining 10 covariance matrices C, as $C = X.X^T/N$, for where X has column vectors stacked, all with the same label.

Now, computing the 84 coordinates for each image is easy, we just take the **component of the image-mean along each of the 84 eigenvectors** in the image space, hence, obtaining a set of 84 numbers. (The code for this can be obtained in the "code" folder)

5.1 Regenerating Images

In this section, we provide an algorithm for regenerating/reconstructing the image using the 84 coordinates obtained before and the knowledge of the designed 84-dimensional basis. In order to implement this in matlab, we have designed a function which takes in the basis and the coordinates as the input, giving the reconstructed image as the output. The process for the same is simple, we simply scale the eigenvectors up by the corresponding coordinates and take their linear combination, finally adding the sample mean to it.

Algorithm 5 Image regeneration

procedure IMAGE(X, B, μ)
  $img \leftarrow B.X^{'} + \mu$
return img
end procedure

This algorithm works since we can write the following equation :

$$Y = \sum_{i=1}^{N} X_i \times B_i \tag{6}$$

$$\Rightarrow Y = B.X \tag{7}$$

where B_i is the i^{th} column of B and X_i is the i^{th} element of the $N \times 1$ vector X.

Upon implementing this algorithm on randomly generated images of one digit of each type, we get the following comparison between original and regenerated images.

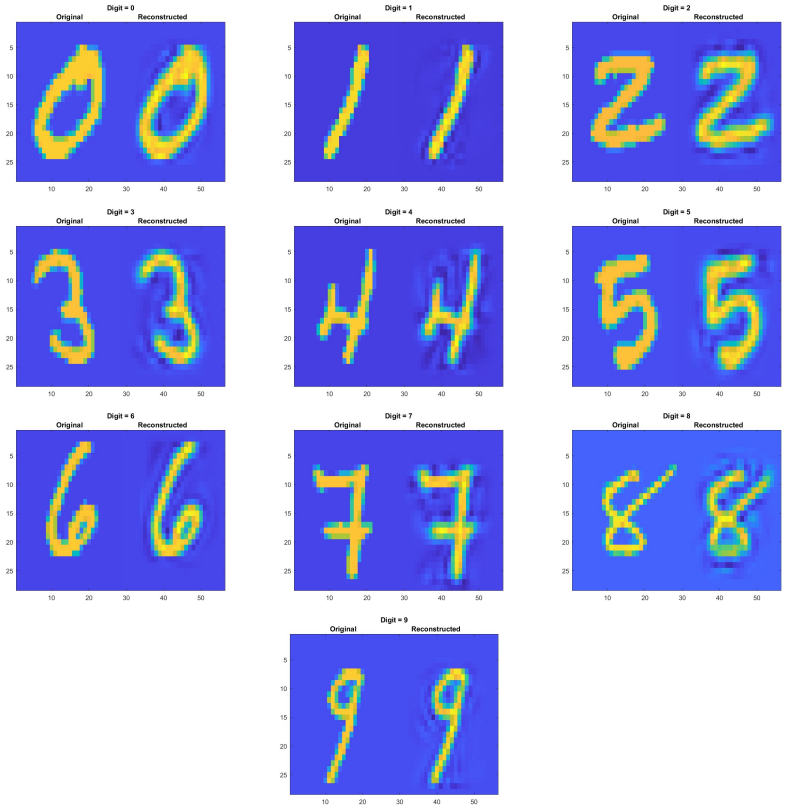


Figure 11: The Doublet of Images

As we can observe, the reconstructed and the original images are nearly identical. Hence, the images can be constructed using 84 dimensions, that is, the ambient dimensions of the images are close to 84 which is much less than 784.

6 Principal Component Analysis (PCA) for fruit dataset

In this section, we shall use the dataset provided within the folder "data_fruit" in the folder "data". Each datum is an image of size 80 × 80 pixels with 3 color channels red (R), green (G), and blue (B), i.e., a 80 × 80 × 3 array. For PCA, each image is resized to a vector of length 19200 . For visualization, we

have reshaped each vector back to a RGB image of size 80 × 80 pixels using the function reshape(), followed by a shifting and rescaling of the values into the range [0, 1] , followed by displaying the matrix using the function image().

6.1 Plotting mean and eigenvalues

Upon preprocessing the fruit images, we obtain a 1920 × 16 matrix, with each column being an image. Let this vector be denoted by X, then the mean and covariance matrices, top 4 eigenvectors are computed as (matlab code):

$$\mu = sum(X, 2)/16 \tag{8}$$

$$C = X.X^T/16 \tag{9}$$

$$[V, D] = eigs(C, 4) \tag{10}$$

Hence, we obtained the following images for the mean and the top 4 eigenvectors.

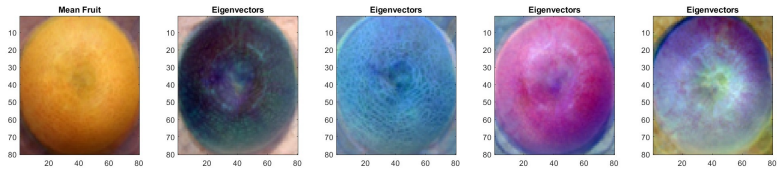


Figure 12: Mean & the top 4 Eigenvectors

Moreover, we obtained the top 10 eigenvalues and plotted them in descending order to get the following graph :

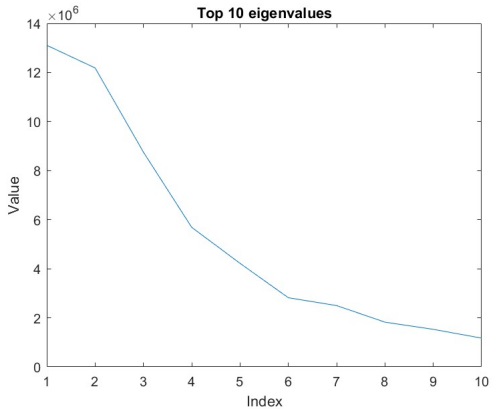


Figure 13: Top 10 Eigenvalues

6.2 Closest Image Representation

In this section we found the closest representation of each fruit image as a linear combination of the top 4 eigenvectors added to the mean, using the measure of closeness as the frobenius norm of the difference. In the 784-dimensional image space, the eigenvectors form a 4-dimensional hyperplane. Hence, the problem reduces to finding the vector in the span of eigenvectors such that distance between a point and a hyperplane is minimum. But we know by the least squares approximation[Uni22], that the solution to this problem is given by the projection of the point on the hyperplane. Hence, the solution is given as :

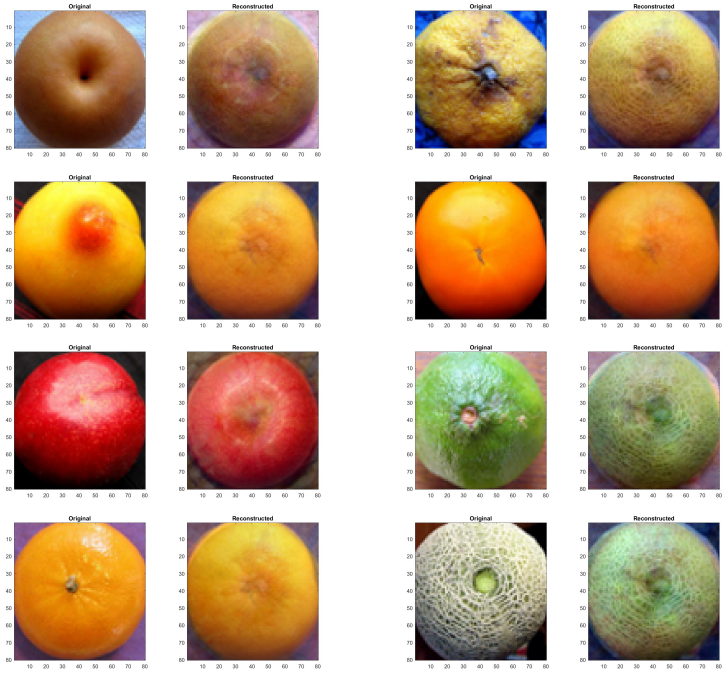
$$b_0 =< u-\mu, v_1 > \lambda_1 + < u-\mu, v_2 > \lambda_2 + < u-\mu, v_3 > \lambda_3 + < u-\mu, v_4 > \lambda_4 + \mu \tag{11}$$

Assuming all eigenvectors v_i are unit norm. Vectorising this, let the column vector stacked eigenvector matrix be given by V(19200×4)we obtain :

$$c = (u-\mu)^T V \tag{12}$$

$$b_0 = Vc^T + \mu \tag{13}$$

where u is 4×1 vector and b_0 is the desired projection.
Using this, we got the following closest representations of all 16 images :



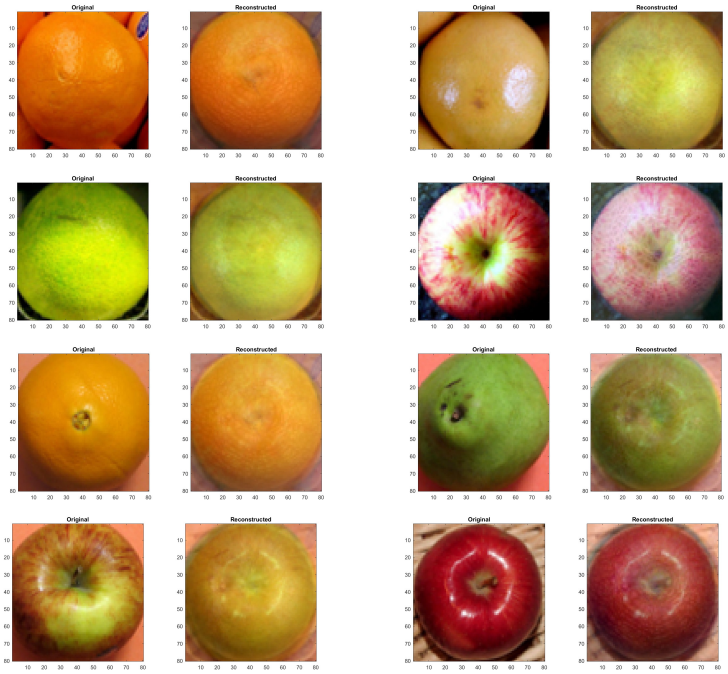


Figure 14: Original & Reconstructed Fruits

6.3 Generating fruit images

In this section we have used all of the top 4 eigenvectors and the mean image to sample random images to generate "fruit" images. Now, we can sample random images by assuming that the distribution arises from a gaussian. Hence, for a multivariate gaussian with covariance matrix having eigenvalues λ_1, λ_2, λ_3, λ_4. Hence, we can generate images using cholensky decomposition of C. Hence, we can find A such that $C = AA^T$, and then, we can sample using :

$$X = AW + \mu \tag{14}$$

where W is a 4×1 vector and A is a 784×4 matrix, μ is known before-hand. Hence, an algorithm for the same is as follows :

Algorithm 6 Image generator

procedure IMAGE(V, μ, λ)
  $D \leftarrow diag(\lambda)$
  $A \leftarrow V\sqrt{D}$
  $W \leftarrow randn(4, 1)$
  $X \leftarrow AW + \mu$
return X
end procedure

where λ is the vector containing all eigenvalues, V contains column stacked eigenvectors. Hence, applying this algorithm, we get the following generated images :

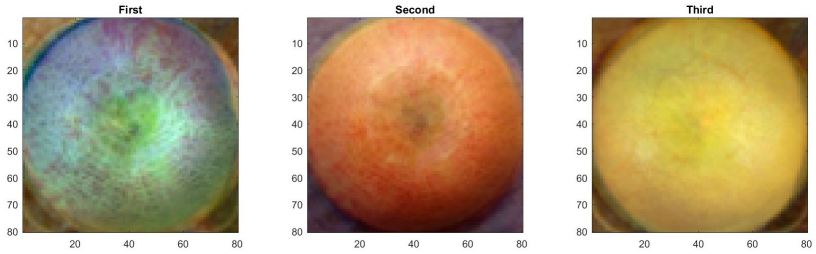


Figure 15: 3 Sample Fruits

Bibliography

[Awa22] Suyash Awate. *Data Analysis and Interpretation*. University Lecture. 2022.

[22] *Github repository*. 2022. URL: https://github.com/Ihsoj-Mahos/CS215-Assignment2.

[Uni22] Brown University. *Least squares approximation*. 2022. URL: https://www.cfm.brown.edu/people/dobrush/cs52/Mathematica/Part5/least.html.

[Wik22] Wikipedia. *Cholesky Decomposition*. 2022. URL: https://en.wikipedia.org/wiki/Cholesky_decomposition.