Math 521 HW3

Raj Mohanty
raj.mohanty@student.csulb.edu

2 Computing

2.1 Problem 1

First the images are loaded in the jupyter environment and some samples are displayed as shown figure 2.1

```
fig, ax = plt.subplots(1, 3)
ax[0].imshow(faces1[:,0].reshape(160,120).T,cmap='gray')
ax[1].imshow(faces1[:,5].reshape(160,120).T,cmap='gray')
ax[2].imshow(faces1[:,108].reshape(160,120).T,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0xb2164ed68>
```

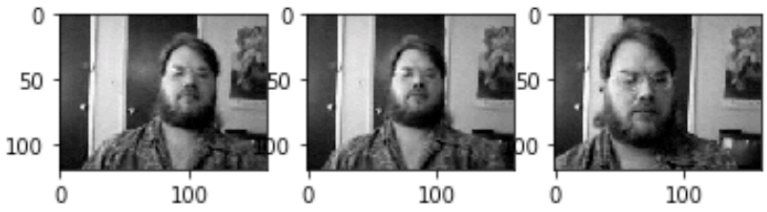


Figure 2.1 : Sample images from the data provided

(a) Ensamble Average Image

The ensamble average image is calculated taking the row averages of the 19200x109 matrix to get at 19200x1 vector.

The ensamble average image is plotted in figure 2.2

```
mu=np.mean(faces1,axis=1)
```

```
plt.imshow(mu.reshape(160,120).T,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0xb2178a940>
```

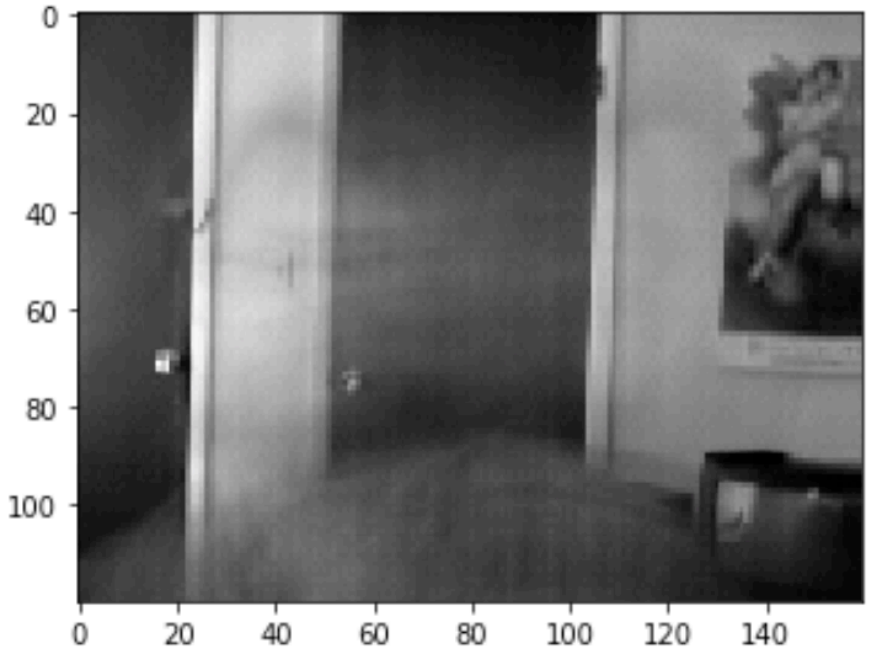


Figure 2.2 : Ensamble average image

(b) Mean subtracted image

In this case the column means are calculated for the 19200x109 matrix and then each value of the column vectors are subtracted by their corresponding means. This is done to make sure the pixel intensities of the images are centered around zero, so that they are all normalized.

We then plot a random mean subtracted image and its corresponding original image in figure 2.3. They are identical.

Original

```
plt.imshow(faces1[:,6].reshape(160,120).T,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0xb216a3ef0>
```

Mean subtracted

```
plt.imshow(mean_sub[:,6].reshape(160,120).T,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0xb221d6f98>
```

Figure 2.3 : Original and Mean subtracted image

(c) Eigen pictures by Snapshot method

To get the eigen pictures of the images we first calculated ensemble averaged covariance metrix Ct

$$C_t = X^T X$$

We chose this Ct to conveniently reduce the dimension to a 109x109 matrix since X is a 19200x109 matrix.

We then determine the eigenvalues and eigenvectors of Ct. The σs are the square root of the eigenvalues.

We construct the Σ matrix from the σs and V matrix has all the eigenvectors as the column vectors.

We know that

$$U = XV\Sigma^{+}$$

Where Σ^{+} is the pseudoinverse of Σ.

We utilize this to calulate the eigen pictures (namely u_i vectors)

The first 10 eigen pictures are displayed in figure 2.4

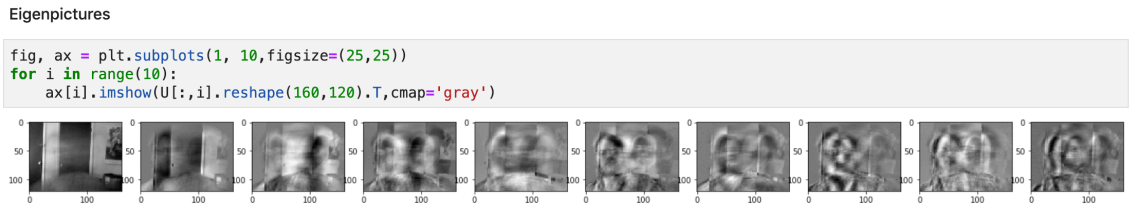


Figure 2.4 : Eigen pictures

We also verify this by using the svd function(see code).

(d) Partial reconstruction

We take first D basis vectors of the eigen pictures and project any random image to it . We then calculate the relative error or distance of the image and its projection.

$$relative\ error = \frac{\|x - x_D\|}{\|x\|} \qquad (1)$$

Where x is any particular image and x_D is its projection on the first D basis. We try different values of D and notice the

reconstructed images. At D=98 (Rank of X) we fully reconstruct the image. Figure 2.5 shows the reconstructed images with their D values and relative errors.

Figure 2.5 : Reconstructed images

(e) Graph of λ_i/λ_{max}

We plot the graph of λ_i/λ_{max} where λ_is are the eigenvalues of Ct.Figure 2.6 shows the plot.

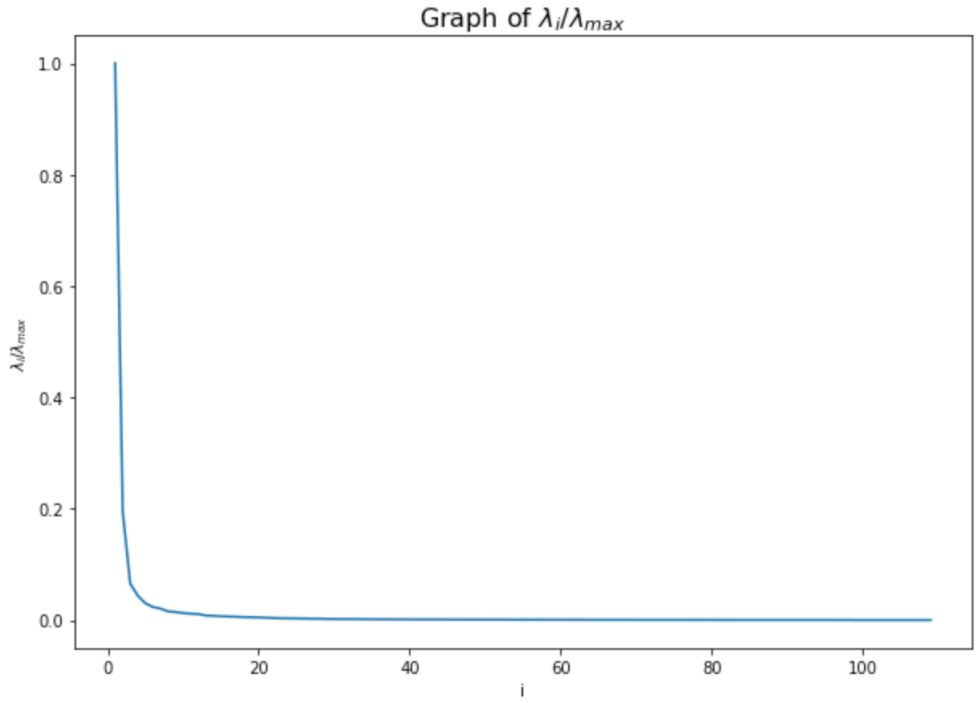


Figure 2.6 : Plot of λ_i/λ_{max}

We notice that the value drops pretty sharply. The value is very close to zero at around 98 which is the rank of the image matrix X.

(f) Classification algorithm using PCA

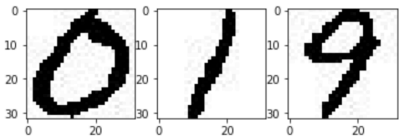
- The algorithm goes as follows:
- Given a set of pattern matrices (Gallery), calculate the SVD and get the left singular vectors of the pattern matrices separately.
- Given a Probe or test pattern (x), project the test pattern on all the different pattern basis (select first D basis). Call it x_D
- Calculate the relative error as in equation (1).
- The probe x is classified to the pattern which has the least relative error

2.2 Problem 2

We load the Gallery and Probe images and display them in figure 2.7.

```
fig, ax = plt.subplots(1, 3)
ax[0].imshow(train_digits[:,0].reshape(32,32).T,cmap='gray')
ax[1].imshow(train_digits[:,50].reshape(32,32).T,cmap='gray')
ax[2].imshow(train_digits[:,499].reshape(32,32).T,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0xb24f41518>
```



```
fig, ax = plt.subplots(1, 10,figsize=(15,15))
for i in range(10):
    ax[i].imshow(test_digits[:,i].reshape(32,32).T,cmap='gray')
    ax[i].set_xticks([])
    ax[i].set_yticks([])
```

Figure 2.7 : Showing some randomly selected gallery images and all probe images

We then apply the algorithm as described problem 2.1 (f).

Each image in the Probe is projected to the first D basis vectors of the Gallery patterns and the Probe image is assigned the pattern class where the relative error is the minimum.

Results:

The results are shown in table 2.1

	train_digits	distance	test_digit	dist_rank	difference
0	0	0.471979	0	1.0	0
11	1	0.427667	1	1.0	0
22	2	0.698496	2	1.0	0
33	3	0.643623	3	1.0	0
44	4	0.567575	4	1.0	0
55	5	0.745625	5	1.0	0
66	6	0.851196	6	1.0	0
77	7	0.556160	7	1.0	0
89	9	0.658987	8	1.0	1
99	9	0.443821	9	1.0	0

Table 2.1 : Result of classifying the Probe (test) images

We notice that algorithm is misclassifying the Probe image 8. That is because the images looks a bit like 9. The projection of that Probe image on all the basis is shown in figure 2.8.

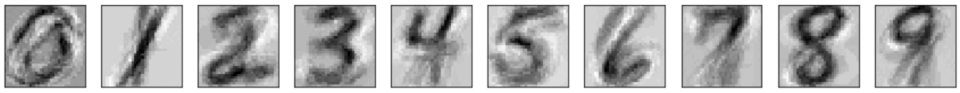


Figure 2.8 : Showing the projection of probe Image of 8 on the KL basis

We notice that the original kind of also looks like a 9. Original image in figure 2.9.

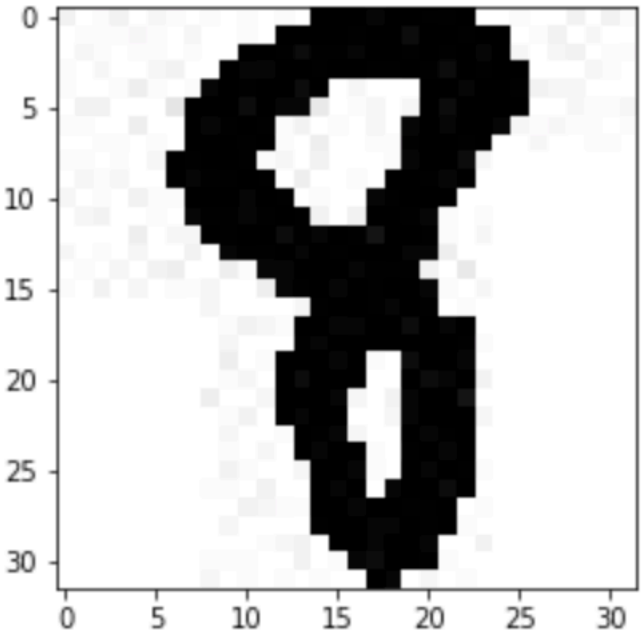


Figure 2.9 : Original Probe image.

In conclusion I believe this is a neat algorithm to apply PCA to classify images. However, this algorithm has its drawbacks as it did misclassify one image where as other algorithms like Convolutional neural networks will be able to classify it correctly.