Assignment-3

Shivanshu Shekhar

April 21, 2023

1 Introduction

By lowering the dimensionality of the dataset, the statistical method known as Principal Component Analysis (PCA) can be used to find patterns in data. It is frequently employed in data visualization and exploratory data analysis.

Finding the linear combinations of the initial variables that can account for the most variation in the data is how PCA works. Principal components are the names given to these linear combinations, which stand for new, agnostic variables.

The first principal component captures the largest amount of variation in the data, the second principal component captures the second largest amount of variation, and so on. By reducing the dimensionality of the dataset to a few principal components, PCA allows us to visualize and analyze complex data in a simpler and more intuitive way.

Data reduction, feature extraction, and data visualization are just a few of the uses for PCA. It is frequently employed in a variety of industries, including image processing, signal processing, and banking.

Assume we have a dataset of n observations with p variables:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \tag{1}$$

Calculate the mean of each variable:

$$\bar{x_j} = \frac{1}{n} \sum_{i=1}^{n} x_{ij} \tag{2}$$

Standardize the data by subtracting the mean from each observation and dividing by the standard deviation:

$$z_{ij} = \frac{x_{ij} - \bar{x_j}}{s_j} \tag{3}$$

where s_j is the standard deviation of variable j.

Calculate the covariance matrix of the standardized data:

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (z_i - \bar{z})(z_i - \bar{z})^T \tag{4}$$

where z_i is the vector of standardized observations for the i-th observation, and $\bar{z}$ is the mean vector of standardized observations.

Calculate the eigenvectors and eigenvalues of the covariance matrix:

$$Sv = \lambda v \tag{5}$$

where v is the eigenvector, and λ is the corresponding eigenvalue.

Sort the eigenvectors by their corresponding eigenvalues in descending order, and select the top k eigenvectors with the largest eigenvalues to form the new k-dimensional feature subspace.

Project the standardized data onto the new feature subspace by taking the dot product of the standardized data and the selected eigenvectors:

$$z_{new} = W^T z \tag{6}$$

where W is a matrix of the selected eigenvectors, and z_{new} is the new k-dimensional feature vector for each observation.

2 Questions-1

We first load both the train and test data using a wrapper function which uses **scipy.io.loadmat** function and this returns a numpy array of images and a numpy array of the corresponding labels.

We then plot one random image from each class to visualize the training and testing dataset. We also calculate the mean image using a wrapper function over **numpy.mean** for both the train and test set and plot that aswell.

We subtract the mean training image from both the training and testing dataset for centering the data, We also visualize them.

Below is the plot for the same:

Figure 1: Train

Figure 2: Test

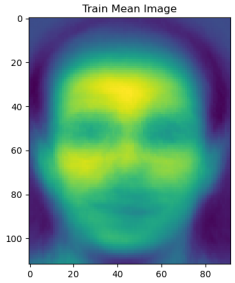


Figure 3: Train Mean

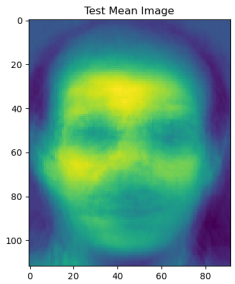


Figure 4: Test Mean

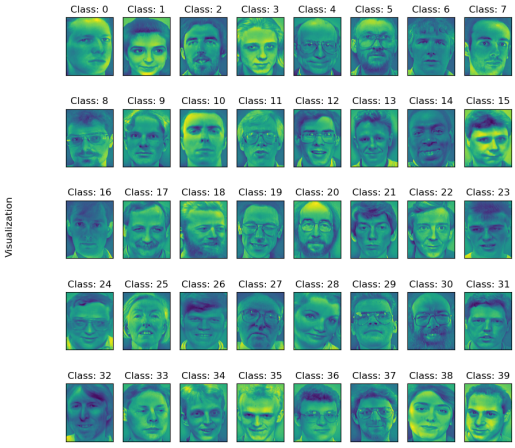


Figure 5: Train Centered

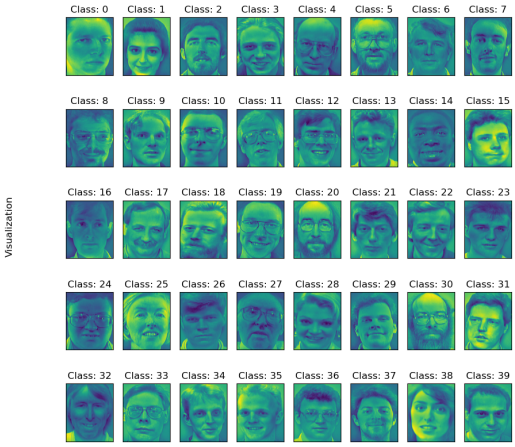


Figure 6: Test Centered

3 Question-2

As mentioned earlier we first need compute the covariance matrix for the centered data by using the formula:

$$S = \frac{X^T X}{N} \tag{7}$$

Where X is $N \times D$ matrix with N samples and each sample having D dimension.

After we get S we need to find its eigenvalues, but we see a clear problem with that, the computational complexity of **numpy.linalg.eig** is $O(D^3)$ and for us, our image had $D = 10304$ which makes the entire computation of the eigenvalues not possible.

We note that in this case N¡¡D and so the data itself will lie in a subspace of at most N-1 dimension in the D dimensional vector space, so it is rather pointless to calculate all the eigenvalues as $D - N + 1$ eigenvalues will be certainly zero.

To efficiently do the computation we change the eigenvalue equation to the following:

$$\frac{XX^T}{N}(Xu) = \lambda(Xu) \tag{8}$$

We see that now XX^T has dimension $N \times N$ and it still has the same eigenvalues as $X.TX$. Further, we also see that $u = k * Su$ where k is some normalization constant. (Note: We also sort the eigenvalues in descending order).

Using the above method we find our eigenvalue decomposition and plot the eigenvectors(eigenfaces) below:

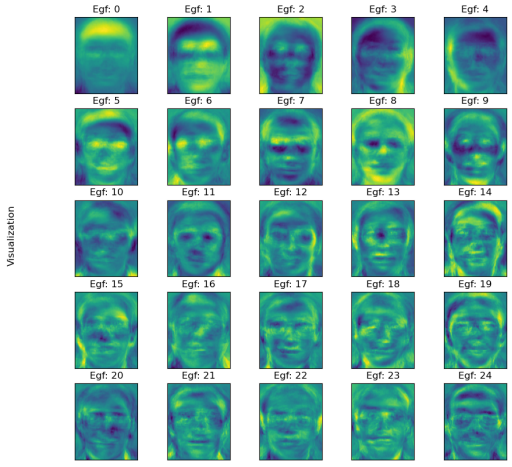


Figure 7: EigenFaces

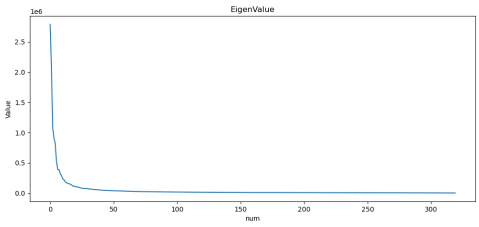
We see that the first eigenfaces correspond to the general aspects of the faces present in the train set, and the later eigenfaces correspond to the specific aspects of a particular image. We see that the first 5 eigenfaces correspond to 5 structures of the face and in the last few eigenfaces we see that some sort of spectacles are getting formed around the eyes.

We next find the number of top eigenvalues required to keep 95% of the data variance, we do the following by using a helper function shown below:

```
def findNumE(Eval: np.ndarray, percent: float):
    i, tmp, den = 0, 0, Eval.sum()
    while(tmp<percent):
        tmp += Eval[i]/den;i+=1

    return i
```

We find that we need to keep 164 top eigenvalues from the 320 eigenvalues to maintain 95% of the data variance.

Finally, we also plot the values of variance associated with each eigenvector in descending order.



4 Question-3

We finally use the 164 eigenvectors obtained in the last question to project our data to a space of dimension 164. We project both the train and test data to this subspace.

We do this using the following lines of code:

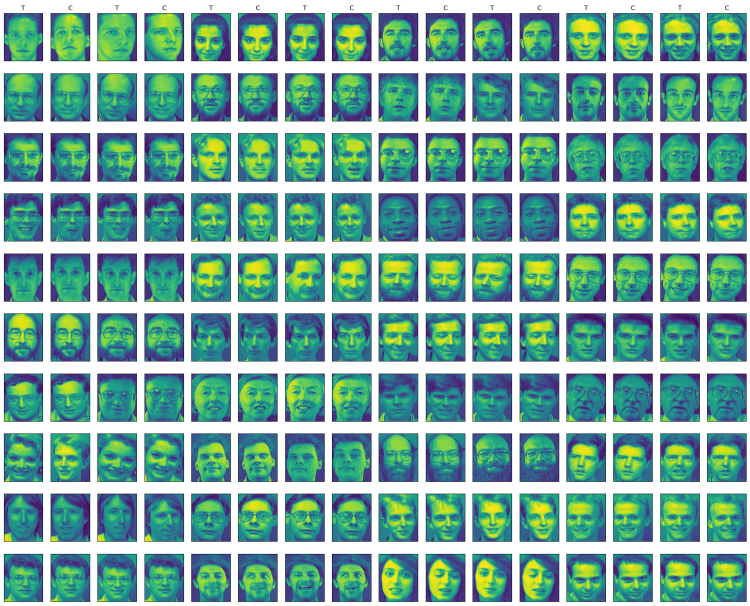
```
def flatten(data: dict):
    for label in data.keys():
        data[label] = list(map(lambda x: x.reshape(-1,), data[label]))

    return data

def collapse(data: dict):
    ret = []
    flat = flatten(data)
    for val in flat.values():
        for v in val:
            ret.append(v)

    return np.array(ret)

def convert(X: dict, U: np.ndarray):
    collapsed = collapse(X.copy())
```

```
return (U@collapsed.T).T
```

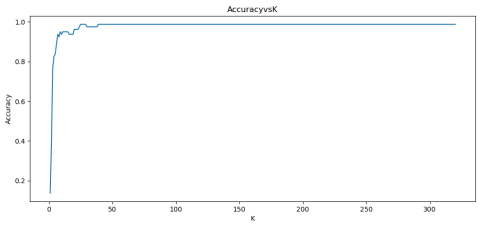
We call the **convert** function on the data to project it from a higher dimension to a lower dimension.

Next, for every test image we try to find the closest train image to it. We use the Euclidean distance for this. (Note: This is being done in the lower space representation.)

We get 98.75% accuracy on the test set for this. We plot the test images with the predicted train image below:



We next plot the variation of the test accuracy with the number of eigenvectors considered for projection.



We see that the test accuracy doesn't change much after 50 eigen components.

5 Conclusion

To summarize, Principal Component Analysis (PCA) is a statistical technique used to identify patterns in data by reducing the dimensionality of the dataset. It works by identifying the linear combinations of the original variables that explain the maximum amount of variation in the data and creating new variables, called principal components, that are uncorrelated with each other. By reducing the dimensionality of the dataset to a few principal components, PCA allows us to visualize and analyze complex data in a simpler and more intuitive way.