Mini Project - 1

1 Principal Component Analysis

We use Singular Value Decomposition (SVD) for performing Pricipal Component Analysis (PCA). By doing SVD we get three matrices. U, Σ and V. We only need U for finding the principal components.

$$A = U\Sigma V^T$$

Now, U contains all the eigen vectors (Principal Componenents). Σ is a diagonal matrix which contains all th eigen values. We use functions from numpy for finding the SVD. Each column of U is a principal component. Also the principal components are sorted in descending order of their importance i.e. the 0th column contains the most important principal component and so on. Now this principal components are used for further processing.

2 Reconstruction from eigen faces

The best part of eigen faces is that we can reconstruct any face from a set of given eigen faces. So the reconstructed face is nothing but a weighted combination of different eigen faces.

More formally,

$w1 = (x - \mu).u1$, $w2 = (x - \mu).u2$ .....

$recon = \mu + w1 * u1 + w2 * u2 + ...$

Here, μ is the mean, x is our image that we want to reconstruct and u1, u2 .... are our eigen vectors.

We plot the Mean Square Error vs No. of Principal Components graph.

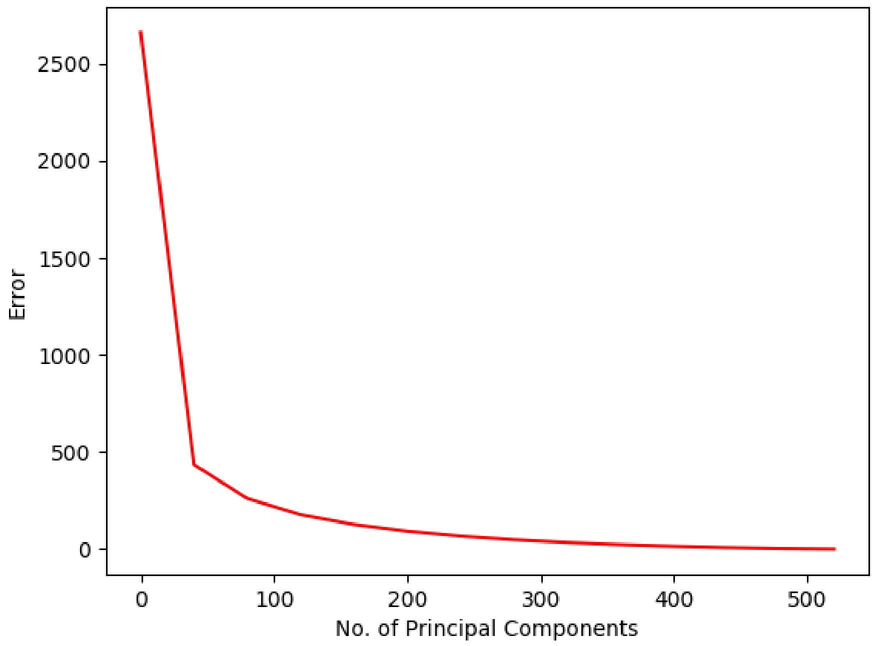

Figure 1: Mean Square Error vs Number of Principal Components

As we can see, there is an extremely sharp drop in mean square error initially as we add up the first few components. This shows us the relative importance of different eigen faces. An example of a reconstructed image is also given below.

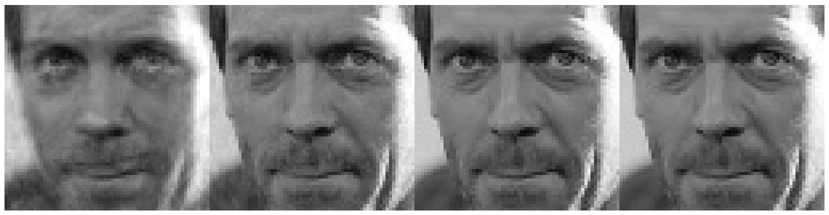

Figure 2: Left to right: Reconstructed images with 130, 350, 520 eigen vectors, original

3 Important Eigen Faces

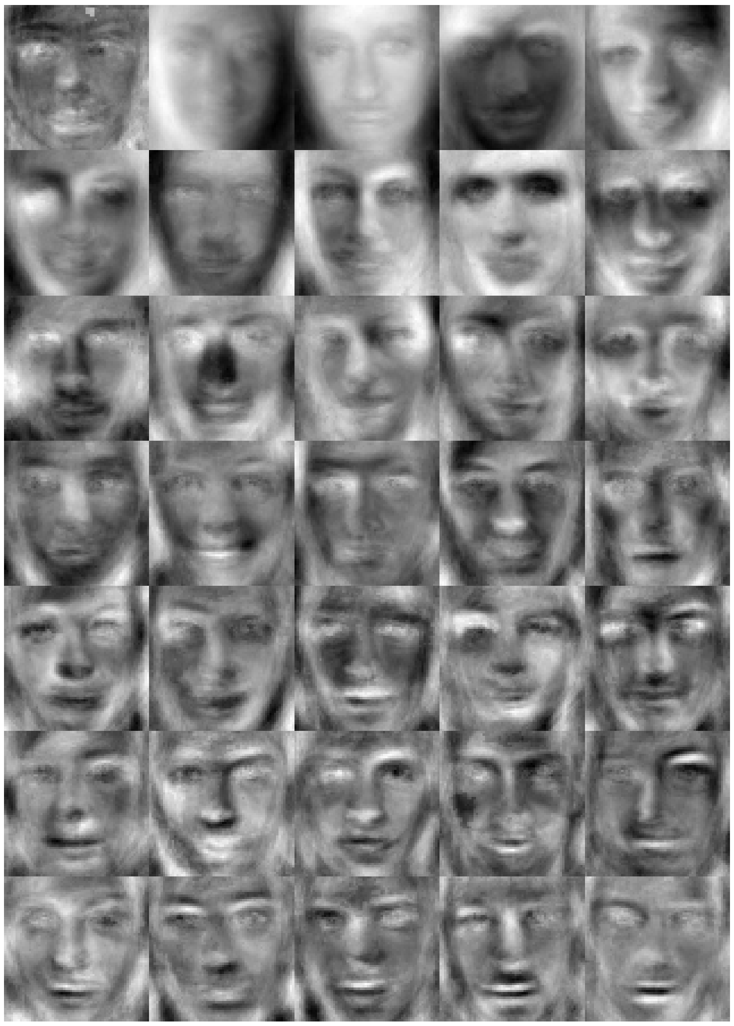

Figure 3: Important Eigen Faces

These are the first 32 eigen faces that was extracted after calculating PCA over the given dataset. As we can see that these captures certain aspects of the human face. For example, some of the faces seems to capture left eye open, right eye open, the front face, the mouth open etc features of the human face.

4 Scatter plots in different dimensions

We use different number of principal components to project the data and then plot them.

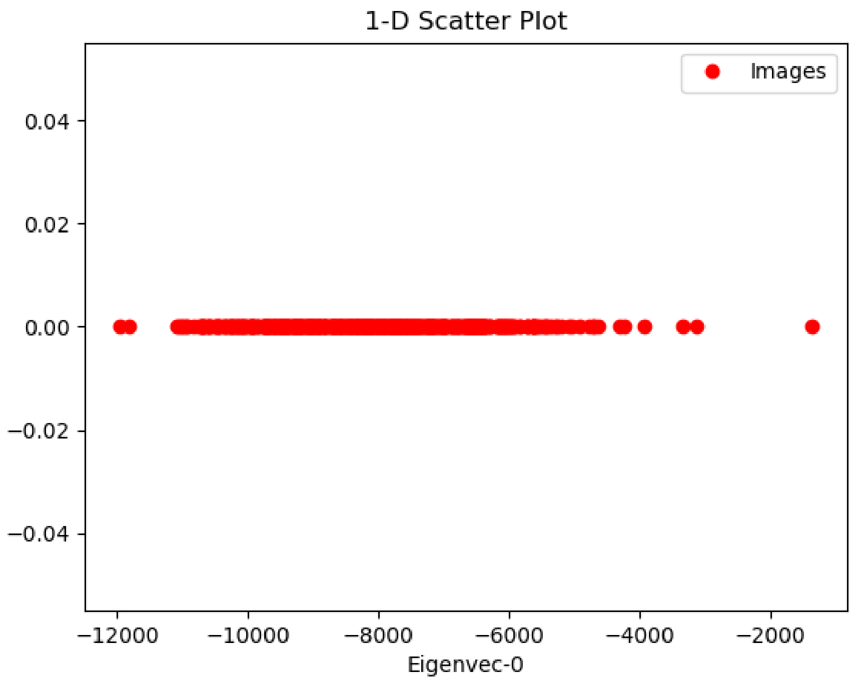

Figure 4: Scatter plot for 1D

In 1D our images are scattered like this in a straight line. As we can see its impossible to classify them into different groups (though different labels were not used for scatter plot) as there are no definite clusters in the data.

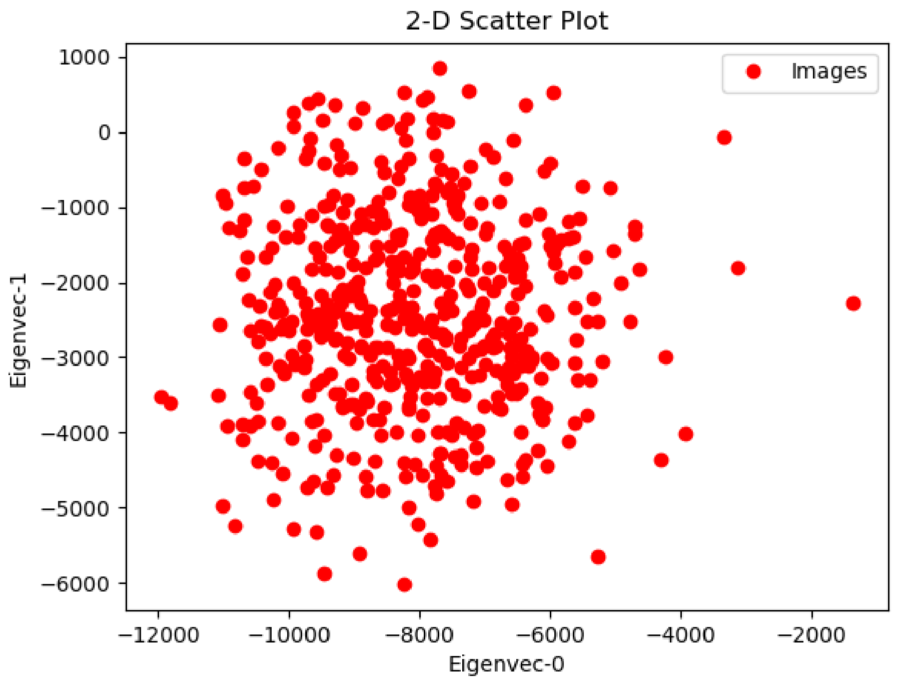

Figure 5: Scatter plot for 2D

In 2D all the images are scattered in slightly better manner. But still it is impossible for a linear classier to classify them correctly.

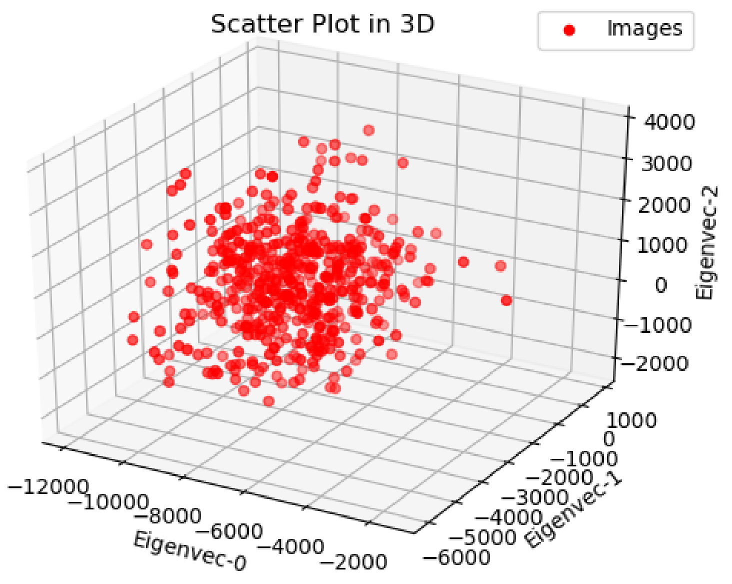

Figure 6: Scatter plot for 3D

In 3D all the images are still scattered to close to each other. Thus we need a far higher dimension to classify them successfully.

5 Naive Bayes Classifier

We project all the training images to 32-D. In this way every 256 X 256 image can be represented by a 32-D vector. For Naive Bayes classifier, We assume that our data is distributed as a normal distribution. We first estimate the μ and Σ from our training data. Our μ is a K X 32 matrix where K is the number of classes. Σ is a 32 X 32 co-varience matrix. We also take only the diagonal of

Σ as we treat each of our features independently. Now Bayes theorem is stated as -

$$p(C|X) = \frac{p(X|C).P(C)}{p(X)}$$

As p(X) is constant so we do not consider it. Now we take log both sides so that,

$$log(P(C|X)) = log(p(X|C)) + log(P(C))$$

Now,

$$p(X|C) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}}.e^{\frac{-1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)}$$

Here D is the dimension of the feature vector. This is 32 in our case.

By taking its log we have,

$$log(p(X|C)) = -log(2\pi)^{\frac{D}{2}} - \frac{-1}{2}log|\Sigma| - \frac{-1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)$$

Now we can ignore the first term as it does not contain μ or Σ. So, finally we can compare just,

$$\frac{-1}{2}log|\Sigma| - \frac{-1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu) + log(P(C))$$

between different classes and take the highest probable class. We got a training accuracy of **65%**. The training set must contain **atleast 32** images to get the 32 eigen faces.

One observation was made that, due to the presence of log(P(C)), the data should be evenly distributed. Otherwise the predictions made by the model would suffer.

6 Linear Classifier

We used a linear classifier to classify faces. We used a softmax classifier with cross entropy loss for this purpose. The softmax function is defined as,

$$P(y = j|x) = \frac{e^{W_j^T X}}{\Sigma e^{W^T X}}$$

Cross entropy loss is defined as,

$$J = -\Sigma t_i.log(p_i)$$

The gradient of this function is equal to,

$$\Delta J = x_i(p_i - t_i)$$

We then update the weights using gradient descent. The images were labelled using one-hot encoding technique. The linear classifier gives around **72.8%** training accuracy. The training set must contain **atleast 32** images to get the 32 eigen faces.