Probability and Random Processes

Assignment-2 Report

Understanding of the algorithm

- Suppose the dataset consist of M images in the training set which are centred around the face so that the algorithm does not detect based on the background or other elements in the image.
- Here in the implementation, I have considered each image to be of 112 X 112 (NXN) (If not rescaled it). Flatten the image in the form of a matrix of 112X112 into a vector of 12544 (112*112) dimensions.

$$I_i = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}_{N \times N} \xrightarrow{\text{concatenation}} \begin{bmatrix} a_{11} \\ \vdots \\ a_{1N} \\ \vdots \\ a_{2N} \\ \vdots \\ a_{NN} \end{bmatrix}_{N^2 \times 1}$$

- Store all these vectors in a matrix of dimension M X N^2. Now, add each column of the matrix and divide by M so as to get the **mean face vector**.
- One can view the face obtained by reshaping the vector obtained into the NXN dimension.
- Now subtract this mean face vector from each image in the train set to obtain the normalised training matrix.
- We can now obtain the covariance matrix from the normalised training matrix by taking matrix product of the normalised training matrix with its transpose:

$$C = AA^T$$

- But, since A is N^2 X M matrix it would be a computationally high task to calculate its transpose and instead we calculate matrix product of A transpose A as that would lead to the formation of MXM matrix. Assuming M<<N^2 this would quickly happen.
- The covariance matrix formed is of dimension MXM and has M eigen vectors of dimensions MX1. We know that :

$$u_i = Av_i$$

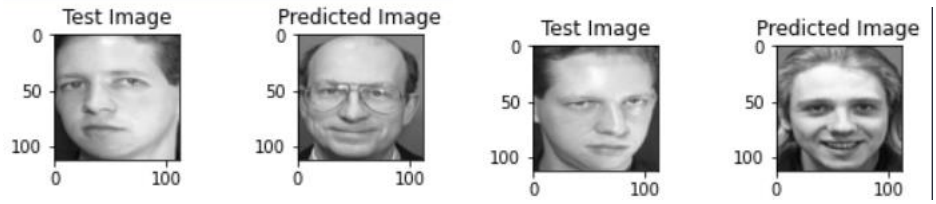
Here u represents the eigen vector of A(A.T) matrix whereas v represents the eigen vectors of the (A.T)A matrix.

- From these eigen vectors we choose some K eigen vectors (Heuristically). In the code I have chosen 50 eigen vectors by plotting the cumulative variance as these 50 eigen vectors represent more than 80% of the variance in data.
- These eigen vectors are sorted in decreasing order and first eigen vectors are selected to form a matrix. This matrix in the code is called the reduced_data (as we have reduced the data matrix in these eigen vectors).
- Now, each face in the training set is represented as a linear combination of these eigen vectors. These weights are computed by taking the projection of the data on these vectors using dot product.
- Again, a matrix is formed using the weights of all the training dataset images and stored in the variable w.
- Now, to check the correctness of the algorithm. We normalise the test image similar to above but here the mean face vector remains the same which was derived using the train set.

- We project the normalised test image on the eigen space that is formed using all the eigen vectors explained above to obtain the corresponding weights (**w_unknown vector**).
- Now, we subtract this w_unknown vector from the w (weight matrix defined above). We then find the norm of each of the column of the matrix. The norm with the minimum value will be the norm of the train image that is closest to the given test image.
- However, it may so happen that the closest image is not the one that is of the same person as that of the test image. It may so happen that the test image was not of a person in the train dataset thus, we define a threshold theta such that:
    1. If the norm is smaller than the threshold theta then we can say that the image is recognised as the image with which it gave the lowest score.
    2. Else, the image does not belong to the dataset.
- The value of threshold theta is defined heuristically and there is no fixed method to define it properly. In the implementation, I have defined it in such a way that although the accuracy of the test data 1 does not increase as compared to when there was not threshold but the overall result on the test dataset 2 would be high which is solely predicted based on the value of the threshold theta value.

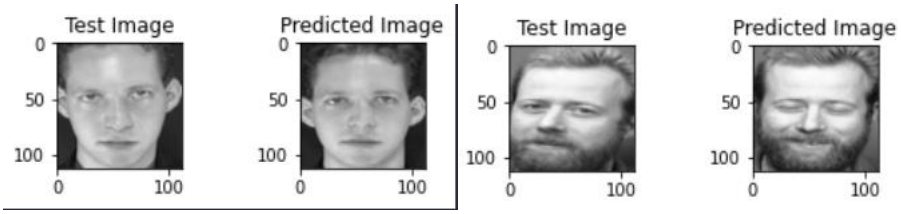
Results obtained by performing face recognition using the algorithm

- I have used two test datasets to evaluate the accuracy and to fix the threshold theta value.
- The first dataset has been created from the original train dataset. Such that now the new train dataset has 7/10 images of each category of images whereas the test1 dataset has 3 images of each type. This splitting is done in the code itself.
- The test2 dataset consist of images that have not been used in the training part so as to see how well the model performs on the unknown images.
- There were two parameters that needed to be set for the algorithm:
    1. The number of eigen vectors to be chosen
    2. Theta threshold value
- I set the number of eigen vectors by plotting the curve of the cumulative variance represented by the number of eigen vectors. From this I chose the top 50 eigen vectors having the maximum eigen values as these represented 80% of all the variance.
- **(Case-1)** For setting the theta threshold value, I did couple of experiments. First, I ran the algorithm without any theta threshold value. The following images (subset) were misclassified (By misclassified I mean that the images were not matched to their respective categories):

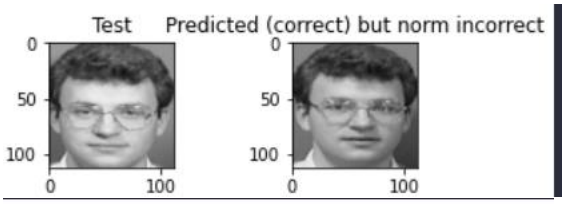


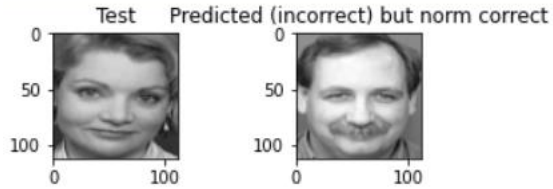
This initial implementation misclassified 10 images. To prevent this, I introduced the theta threshold parameter in the next run of the algorithm. I set its value to the minimum norm value amongst all the misclassified images. As this would lead to these all images getting classified as not in the dataset.

- **(Case-2)** But it turned out that, it did not work well. As some of the images that were being properly classified did not count into correct classified since there norm was greater than the theta threshold value. For example:

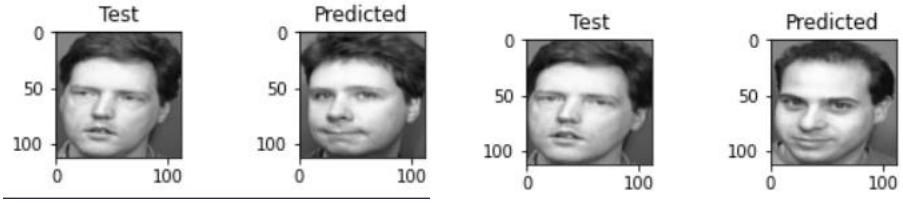


- **(Case-3)** Of all the images misclassified in case 2, I took the 70th percentile of these norm values and set it as the new threshold. This increased the accuracy considerably from 77% to 90%. Now the misclassified images included (These are the images which got misclassified due to the theta threshold value else they were correctly matched with their respective class):





- **(Case-4)** Then I evaluated the algorithm on the dataset containing images that did not belong to any person whose image was fed in the training of the algorithm. Here the results (Subset) obtained were as follows (wrong predictions):



Accuracy of the algorithm and inference of results

- The following cases are in accordance with the experiments done above (kindly refer the above paragraphs):
    1. Accuracy obtained in case-1 is: **90.99% (Here no threshold value was set) (Test set-1)**
    2. Accuracy obtained in case-2 is: **77.47% (Here threshold value was set) (Test set-1)**
    3. Accuracy obtained in case-3 is: **90.990% (Here threshold value was set) (Test set-1)**
    4. Accuracy obtained in case-4 is: **73.33% (Here threshold value was set) (Test set-2)**
- The above results are a clear indication of the fact that the algorithm heavily relies on the value of the parameters of theta threshold. By setting proper value of theta threshold, one can achieve really good result. At the same time at times although setting higher values of theta threshold might do better on images of persons in training dataset but it would do really bad on images that of person that have not been seen before by the model.
- Although it is not evident from the accuracy, but the number of eigen vectors selected also plays a key role. Although selecting higher number of eigen vectors might give us better result but then it would enhance the computation cost. Also, at times taking into account higher eigen vectors might lead to overfitting and the algorithm will not be able to predict better on new data.

Assumptions

- In the implementation of the algorithm, I have considered all the images to be square and if not I have resized them.
- All the images considered and do not depict any 3D feature of the face. That is these images are upright and frontal

References

- https://onionesquereality.wordpress.com/2009/02/11/face-recognition-using-eigenfaces-and-distance-classifiers-a-tutorial/
- https://github.com/vutsalsinghal/EigenFace
- https://github.com/manasbedmutha98/EigenFaces/blob/master/Eigenfaces%20for%20Recognition.ipynb
- AT&T dataset from:
  https://www.kaggle.com/datasets/kasikrit/att-database-of-faces