1 Introduction

In this homework we perform two tasks. In the first task we do face recognition in the following manner,

1. In the first step we carry out a Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) of the images to transform the high-dimensional image data to a low-dimensional manifold
2. Next, we use a nearest-neighbor classifier to classify the face data

In the second task of this homework we design an object detector (car detector) with an arbitrarily defined false positive rate using the cascaded Adaboost algorithm

2 Face Recognition using PCA and LDA with Nearest Neighbor Classifier

2.1 Principal Component Analysis (PCA)

In PCA, a high-dimensional feature space is converted into a low-dimensional subspace by calculating the eigenvectors of the covariance matrix of the high-dimensional space and retaining features corresponding to a particular number of largest eigen vectors. The following steps are involved in implementing PCA:

1. First, each gray scale image in the training data of size 128×128 is converted into its vectorized form of size $\vec{x}_i = 16348 \times 1$. We normalize each vectorized image $\vec{x}_i$ so that it is invariant to illumination. After normalizing the training images we calculated the mean image vector of the training set $\vec{m}$,

$$\vec{m} = \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} \vec{x}_i, \tag{1}$$

   where N_{tr} is the total number of images in the training set.

2. Next form the vector X which is given by,

$$X = \begin{bmatrix} \vec{x}_1 - \vec{m} & \vec{x}_2 - \vec{m} & ... & \vec{x}_{n_{tr}} - \vec{m}. \end{bmatrix} \tag{2}$$

3. In the next step, to calculate eigen vectors $\vec{w}$ the covariance matrix $C = XX^T$, we instead calculate the eigen vectors $\vec{u}$ of the matrix $X^T X$ and then calculate $\vec{w}$ from $\vec{u}$ using,

$$\vec{w} = X\vec{u}. \tag{3}$$

   Following this procedure makes the eigen value decomposition of C numerically stable. After calculating $\vec{w}$ normalize it.

4. We now select the largest K eigen vectors from the normalized $\vec{w}$. The matrix W_k that carries out the dimensionality reduction in PCA is given by,

$$W_k = \begin{bmatrix} \vec{w}_1 & \vec{w}_2 & ... & \vec{w}_K. \end{bmatrix} \tag{4}$$

   The feature values corresponding to the vectorized image $\vec{x}_i$ is given by,

$$\vec{y}_i = W_k^T \left(\vec{x}_i - \vec{m} \right) \tag{5}$$

5. For an image in the test set, we first vectorize the image, normalize it and then determine its feature values using Equation (5). The final step in the face recognition classifier is to use a nearest-neighbor classifier to identify its closest match from the training set.

2.2 Linear Discriminant Analysis (LDA)

In the LDA, the objective is to find subspaces in the high-dimensional image space which maximally discriminate classes. This is done by finding the eigen vectors $\vec{w}_i$ which maximize the Fisher Discriminant Function,

$$J(\vec{w}_i) = \frac{\vec{w}_i^T S_B \vec{w}_i}{\vec{w}_i^T S_W \vec{w}_i}, \tag{6}$$

where S_B is the between-class scatter and S_w is the within-class scatter. S_B and S_W are defined as,

$$S_B = \frac{1}{|\mathtt{C}|} \sum_{i=1}^{|\mathtt{C}|} (\vec{m}_i - \vec{m}) (\vec{m}_i - \vec{m})^T, \tag{7}$$

$$S_W = \frac{1}{|\mathtt{C}|} \sum_{i=1}^{|\mathtt{C}|} \frac{1}{|\mathtt{C_i}|} \sum_{k=1}^{|\mathtt{C_i}|} (\vec{x}_i - \vec{m}_i) (\vec{x}_i - \vec{m}_i)^T, \tag{8}$$

where $|\mathtt{C}|$ is the set of all classes, $\vec{m}_i$ is the mean image of the i^{th} class, $\vec{m}$ is the global mean image, $|\mathtt{C_i}|$ is the i^{th} class and $\vec{x}_i$ is an image in the i^{th} class. In order to calculate $\vec{w}_i$ we use the Yu and Yang's algorithm. This algorithm is implemented using the following steps:

1. First each gray scale image in the training set of size 128×128 is converted into its vectorized form of size $\vec{x}_i = 16348 \times 1$ and then normalized to form a vectorized image $\vec{x}_i$ so that it is invariant to illumination.

2. After normalizing the training vector images $\vec{x}_i$ we calculate $\vec{m}$ using Equation 1 and $\vec{m}_i$ using,

$$\vec{m}_i = \frac{1}{|\mathtt{C_i}|} \sum_{k \in |\mathtt{C_i}|} \vec{x}_i. \tag{9}$$

3. Next, form the matrix M,

$$M = \begin{bmatrix} \vec{m}_1 - \vec{m} & \vec{m}_2 - \vec{m} ... & \vec{m}_N - \vec{m} \end{bmatrix}, \tag{10}$$

   where N is the number of classes. In terms of M, S_B can be written as $S_B = MM^T/|\mathtt{C}|$

4. In the Yu and Yang's algorithm, instead of directly calculating the eigen vectors of S_B we calculate the eigen vectors of $MM^T/|\mathtt{C}|$. If $\vec{u}$ is the eigen vectors of $MM^T/|\mathtt{C}|$ sorted in descending order of the corresponding eigen values, then the eigen vectors $\vec{V}$ of S_B are given by,

$$\vec{V} = M\vec{u} \tag{11}$$

5. In the next step we form the matrix $Y = [\vec{V}_1, \vec{V}_2 ...]$ and D_b which is the diagonal eigen value matrix of S_B. Computer $Z = YD_b^{-1/2}$. ($D_b^{-1/2}$ is element wise operation)

6. We now compute the eigen vectors of $Z^T S_W Z$ using the same computational trick we used in PCA and to calculate eigen vectors of S_B. Rewrite $Z^T S_W Z$ as,

$$Z^T S_W Z = \left(Z^T X_W\right)\left(Z^T X_W\right)^T, \tag{12}$$

where X_W is,

$$X_W = \begin{bmatrix}\vec{x}_{11} - \vec{m}_1 & \vec{x}_{21} - \vec{m}_1 & \vec{x}_{31} - \vec{m}_1 & \ldots & \vec{x}_{k1} - \vec{m}_1 & \ldots & \vec{x}_{1N} - \vec{m}_N & \ldots & \vec{x}_{kN} - \vec{m}_N\end{bmatrix} \tag{13}$$

7. If $\vec{U}$ indicates the eigen vectors for $Z^T S_W Z$, sort them in ascending order based on the eigen values and the select the smallest K eigen vectors from $\vec{U}$. Let $\hat{U}$ denote the set of the K eigen vectors. The matrix W_k that reduces the dimensionality in LDA is given by,

$$W_k^T = \hat{U}^T Z^T. \tag{14}$$

8. Normalize each eigen vector in W_p and the feature values corresponding to the vectorized image $\vec{x}_i$ is given by,

$$\vec{y}_i = W_k^T \left(\vec{x}_i - \vec{m}\right). \tag{15}$$

9. For images in the test set, vectorize the image first, normalize it and then use Equation 15 to get the feature values. To classify the image in the test set, compare the feature values with the feature values in the training set using a nearest neighbor classifier.

2.3 Comparison of Performance of PCA and LDA

To evaluate the performance of PCA and LDA for face recognition we define the accuracy of prediction as,

$$\text{Accuracy} = \frac{\text{Number of Test Images Correctly Classified}}{\text{Total Number of Test Images}} \tag{16}$$

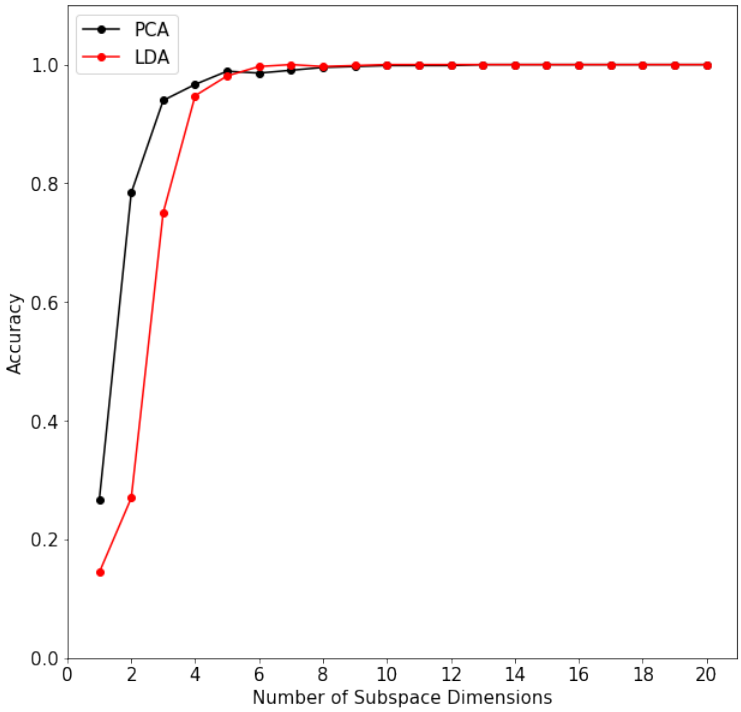

Figure 1: Comparison of accuracy for PCA and LDA

From the figure, we can make the following observations,

1. In general we can see that LDA approach for dimensionality reduction does not always outperform PCA.
2. For dimensionality of the subspace below 5, PCA predicts more accurate results in comparison to LDA. However from subspace dimensionality 5 to 10 LDA performs better than PCA.
3. LCA achieves 100% accuracy for dimensionality 7 while PCA requires 13 subspace dimensionality to reach 100% accuracy.

3 Object Detection using Cascaded Adaboost Algorithm

In the second part of this homework we carry out object detection, namely car detection using a cascaded Adaboost classifier based on the Viola-Jones detector. The cascaded Adaboost classifier consists of series of strong classifiers one after another. Each strong classifier comprises of a number of weak classifiers. By specifying a false-positive rate and true-positive rate or detection rate for each of the strong classifiers, we can achieve the desired level of false-positive rate for the cascade of strong classifiers while achieving a relatively high detection rate. The first step in cascaded Adaboost classifier is feature generation. This is described in the following section.

3.1 Haar Type Feature Extraction

In order to build the features, Haar type rectangular features are used here. The sum of the pixels in the rectangle features can be easily obtained using the integral image representation. In this study

we use two-rectangle features, Type 1 horizontal operators of the form $1 \times 2, 1 \times 4, 1 \times 8, ...1 \times 40$ and Type 2 vertical operators $2 \times 2, 4 \times 2, ...20 \times 2$. Using these operators, the total number of features for each is 11,900.

3.2 Best Weak Classifier

In Adaboost classifier, each feature acts like a weak classifier. To construct a strong classifier we need to determine the best weak classifier. This is done in the following manner,

1. For each feature, (11900 in this case), all the images in the training set are sorted in ascending order of the feature value. For each image in the sorted list, we determine T^+ the total sum of positive weights, T^- the total sum of positive weights, S^+ the sum of positive weights below the current sample and S^- which is the sum of the negative weights below the current sample. The threshold for each feature is calculated as,

$$e = \min\left(S^+ + (T^- - S^-), S^- + (T^+ - S^+)\right). \tag{17}$$

   The feature which gives the minimum value of e is selected as the best weak classifier.

2. The weak classifier is defined as,

$$h(x, f, p, \theta) = \begin{cases} 1, & \text{if } pf(x) < p\theta \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

   where x is the image, f the value of the best feature that yields minimum e, θ is the threshold and p is the polarity. $p = -1$ if $(S^+ + (T^- - S^-) < S^- + (T^+ - S^+))$ and $p = 1$ other wise.

3.3 Strong Classifier from Weak Classifiers

The steps for building a strong classifier from the best weak classifiers are,

1. Given N training examples, label the positive examples as 1 and the negative examples as 0.
2. Initialize the weights as, $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ where m and l are the number of negative and positive examples respectively.
3. For $t = 1, ...T$:
   (a) Normalize the weights $w_{t,i} = w_{t,i} / \sum_i w_{t,i}$
   (b) Get the best weak classifier $h_t(x)$ which has the minimum weighted error ϵ_t
   (c) Calculate $\beta_t = \epsilon_t/(1 - \epsilon_t)$ and $\alpha_t = log(1/\beta_t)$
   (d) Update the weights $w_{t+1,i} = w_{t,i}\beta^(1 - e_i)_t$ where $e_i = 0$ if image x_i is classified correctly and $e_i = 1$ if it is classified incorrectly.
4. The final strong classifier is given by,

$$C(x) = \begin{cases} 1, & \text{if } \sum_t \alpha_t h_t(x) > \frac{1}{2} \sum_t \alpha_t \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

5. The stopping criterion is based on the pre-defined value of detection rate and false-positive rate for each strong classifier. Here we use 1.0 for the true detection rate and 0.5 false-positive rate for each strong classifier

3.4 Cascade of Strong Classifiers

In the final step of the cascaded Adaboost classifier, a cascade of strong classifiers is constructed. Positive results (true positive or false positive) from the first strong classifier triggers the evaluation of the second classifier. Negative results (true negative, false negative) are discarded from a classifier and are not considered further. For each stage of the cascade, the best weak classifier is updated based on the input samples. The advantage of a cascade is that even if the false positive rate of a strong classifier is high, the false-positive rate of the cascaded classifier is significantly lower.

3.5 Performance of Cascaded Adaboost Classifier

The performance of the cascaded Adaboost classifier by calculating the false-positive rate (FPR) and false-negative rate (FNR) which is given by,

$$\text{FPR} = \frac{Total \quad Number \quad of \quad Misclassified \quad Negative \quad Images}{Total \quad Number \quad of \quad Negative \quad Images} \tag{20}$$

$$\text{FNR} = \frac{Total \quad Number \quad of \quad Misclassified \quad Positive \quad Images}{Total \quad Number \quad of \quad Positive \quad Images} \tag{21}$$

3.5.1 Training Results

Table 1: Number of Weak Classifiers in each stage of Cascade during Training Stage

Stage	1	2	3	4	5	6	7	8
Number of Weak Classifiers	8	13	9	8	8	8	5	2

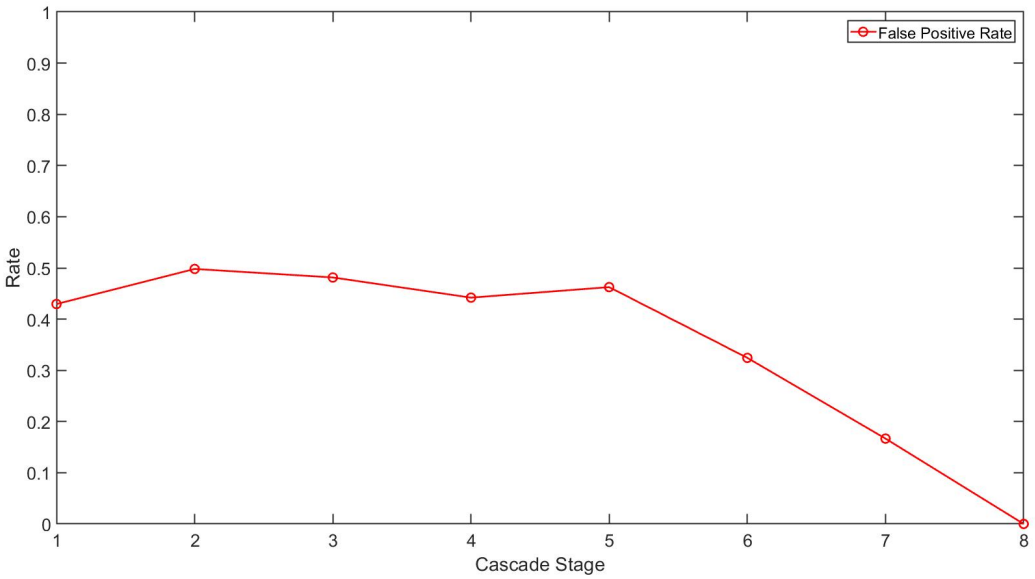

Figure 2: Change in FPR with the number of stages during the training step

3.5.2 Test Results

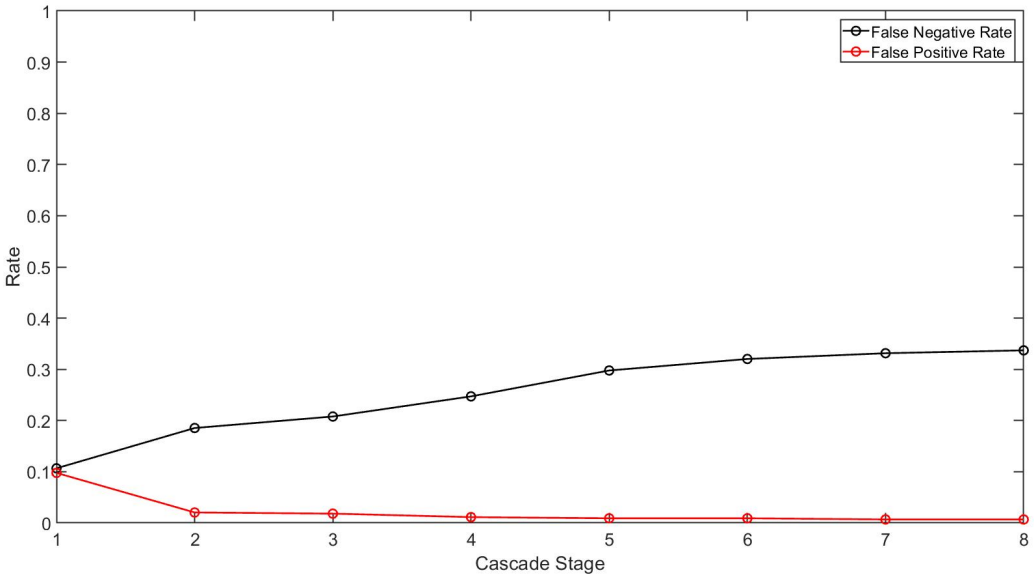

Figure 3: Change in FPR and FNR with the number of stages during the testing step

The following observations can be made,

1. The FPR goes on decreasing as the number of stages increases. The final value of FPR is 0.0068.
2. The value of FNR goes on increasing with the number of stages. The final value of FNR is 0.337.

4 Implementation

1. PCA and LDA is implemented in the Python notebook **hw10_TejasPant_PCA_LDA.ipynb**
2. Cascaded Adaboost classifier is implemented in MATLAB script **hw10_TejasPant_Adaboost.m**