**ECE 661: Homework 10**

In the following assignment, we are required to complete two different tasks. First, we are required to conduct face recognition using PCA and LDA for dimensionality reduction and then use nearest neighborhood rule for the classification. The second task requires us to use a cascaded AdaBoost classifier for object detection.

**Task 1**

**Face Recognition**

**Principal Component Analysis (PCA):**

The principal component analysis is used to project a high dimensional data to a low dimensional representation. The following are the steps to conduct PCA:

1. Convert the images in training set from the RGB values to gray scale values.

2.Vectorize each of the 128 x 128 sized image into a 16384 x 1 vector where i = 0, 1, 2 … N-1 for N images in training set. Normalize the vectorized images

2. Calculate the global training mean

3. Create a matrix

4. Calculate the eigen vectors of the covariance matrix and carry out its eigen value decomposition.

5. Let be the eigen vectors of the covariance matrix in descending order of eigen values. The eigen vectors are given by, . Normalize the vectors.

6. Calculate a parameter K, where K is number of eigen values of eigen vectors that are greater than 0.5.

7. Select the first K vectors in to create the projection matrix .

8. Use the projection matrix to project the training images using the formula

9. Create an array Z that provides a label to training images. Use the projected train data and its corresponding labels to train the nearest neighbor model. I have used the KNN model implementation through the scikit-learn library in python.

10.Vectorize each of the 128 x 128 sized test images image into a 16384 x 1 2. Repeat steps 2 and 3 for the test images.

11. Project the testing data using the method described and step 8. Obtain classification of the testing data using the KNN model.

12. Determine the accuracy of the of the predicted output from the test data by comparing it to the labels created.

13. Repeat the process for varying values of K.

**Linear Discriminant Analysis (LDA)**

The LDA process finds the eigen vectors by maximizing the Fisher Discriminant Function given by,

,

where is the between class scatter and is the within class scatter

The between class scatter, is given by the following formula,

,

where is the class mean and is the global mean and C is the number of image in the dataset.

The within class scatter Sw is given by,

,

where Ci is the number of images in a given class and is the kth image vector in the ith class.

The Yu and Yang’s algorithm is used to find the eigen vectors and the projection matrix, since is singular in most cases.

**Yu and Yang’s algorithm**

Due to the singularity of the matrix the Yu and Yangs algorithm is used to calculate the eigen vectors and the projection matrix. Following are the steps in the Yu and Yangs algorithm to classify the images in the dataset:

1. Convert the images in training set from the RGB values to gray scale values.

2.Vectorize each of the 128 x 128 sized image into a 16384 x 1 vector where i = 0, 1, 2 … N-1 for N images in training set. Normalize the vectorized images

2. Calculate the global training mean

2. Calculate the training class mean, , where Ci is the number of images in a given class.

3.Calculate the between class scatter SB, given by , where is the class mean and is the global mean and C is the number of image in the dataset.

4. Carry out the eigen value decomposition of SB. Let V be the matrix consisting of the eigen vectors of SB in descending order. Y is a matrix of the first K eigen values in V. DB is the diagonal matrix consisting of the first K eigen values of SB in descending order. The eigen value decomposition of SB is done by first conducting the eigen value decomposition of A = . The eigenvalues of SB are equal to those of A. Eigen vectors of SB are equal to eigenvectors of A times .

5. Calculate the matrix

6. Conduct the eigen value decomposition of is given by the following matrix.

7. Let be the eigen vectors of the matrix in ascending order of eigen values. Let be the smallest K eigen vectors of

8. The projection matrix Wk is given by, . Normalize the eigen vectors in Wk.

9.Use the projection matrix to project the training images using the formula

10. Create an array Z that provides a label to training images. Use the projected train data and its corresponding labels to train the nearest neighbor model. I have used the KNN model implementation through the scikit-learn library in python.

11.Vectorize each of the 128 x 128 sized test images image into a 16384 x 1 2. 11. Project the testing data using the method described and step 9. Obtain classification of the testing data using the KNN model.

12. Determine the accuracy of the of the predicted output from the test data by comparing it to the labels created.

13. Repeat the process for varying values of K.

**Results of Task 1:**

Accuracy of PCA and LDA is calculated using the following formula

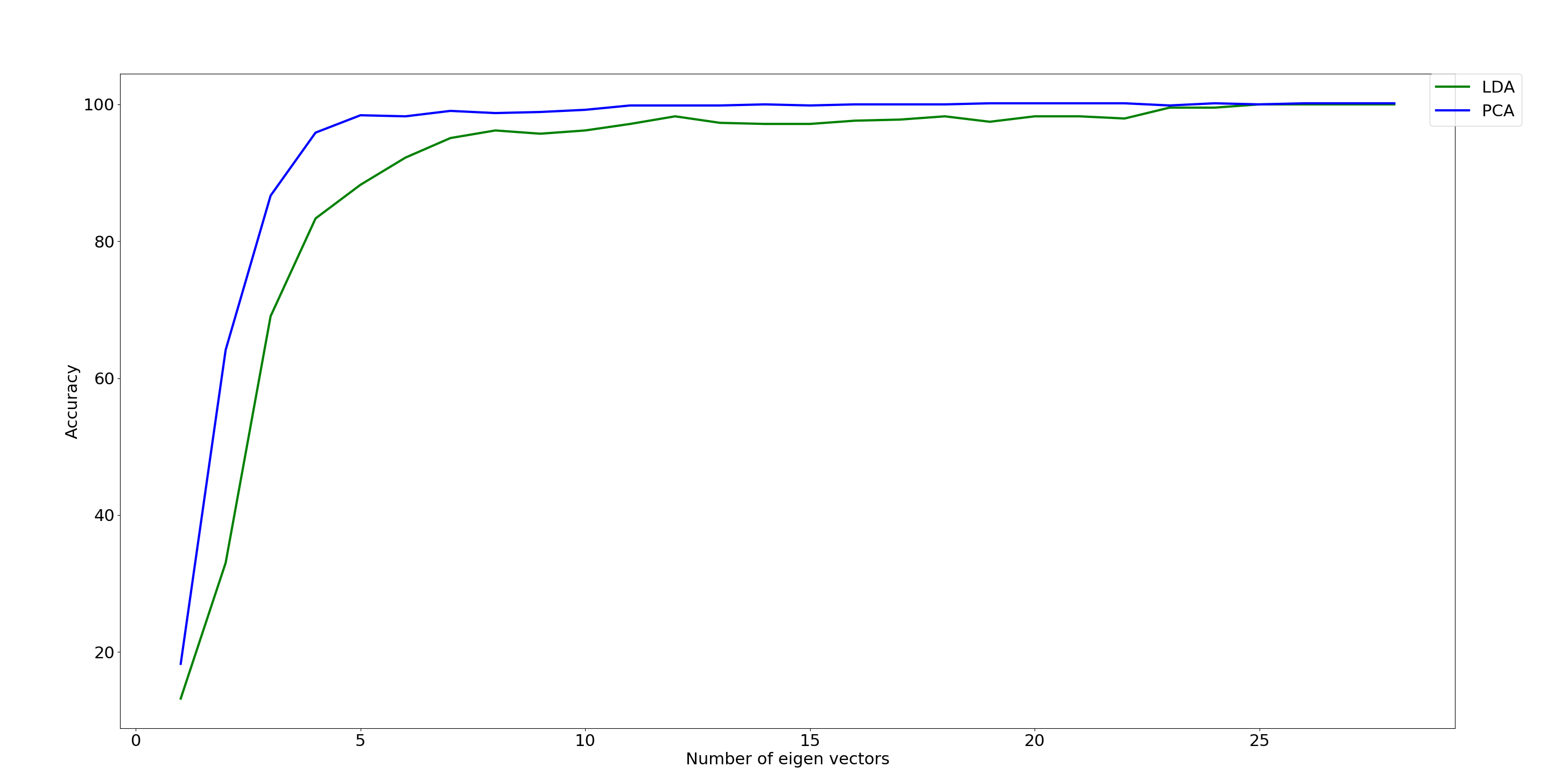


Figure 1: Plot showing the classification accuracy for both PCA and LDA

**Comparison table between PCA and LDA**

|  |  |  |
| --- | --- | --- |
| Subspace dimension | PCA accuracy | LDA accuracy |
| 1 | 0.18253 | 0.13174 |
| 2 | 0.64126 | 0.33016 |
| 3 | 0.86666 | 0.69048 |
| 4 | 0.95873 | 0.83333 |
| 5 | 0.98413 | 0.88254 |
| 6 | 0.98254 | 0.92222 |
| 7 | 0.99048 | 0.95074 |
| 8 | 0.98730 | 0.96190 |
| 9 | 0.98889 | 0.95714 |
| 10 | 0.99206 | 0.96191 |
| 11 | 0.99841 | 0.97143 |
| 12 | 0.99841 | 0.98254 |
| 13 | 0.99841 | 0.97302 |
| 14 | 1.0 | 0.97143 |
| 15 | 0.99841 | 0.97144 |
| 16 | 1.0 | 0.97619 |
| 17 | 1.0 | 0.97777 |
| 18 | 1.0 | 0.98254 |
| 19 | 1.0 | 0.97460 |
| 20 | 1.0 | 0.98254 |
| 21 | 1.0 | 0.98254 |
| 22 | 1.0 | 0.97937 |
| 23 | 0.99841 | 0.99524 |
| 24 | 1.0 | 0.99524 |
| 25 | 1.0 | 1.0 |
| 26 | 1.0 | 1.0 |
| 27 | 1.0 | 1.0 |
| 28 | 1.0 | 1.0 |

**Conclusion:**

1. In the lower subspace dimensions the PCA is seen to have more accuracy than LDA.

2. While theoretically LDA is supposed to reach a 100 percent accuracy faster than PCA. In this experiment PCA is seen to reach a 100 percent accuracy at a smaller subspace dimension of 14 than LDA which reaches 100 percent accuracy at a subspace dimension of 24.

**Task 2:**

Object detection with Cascaded AdaBoost Classifier

The AdaBoost classifier aggregates weak classifiers to create a strong classifier which is then used to classify the images in the data set. Following are the different procedures that go into training an AdaBoost Classifier:

**Extracting Haar Features**

The first step to building weak classifier is to extract Haar features from all images in the training data set. Following are the steps in extracting the Haar features from an image:

1. Convert the image from RGB to gray scale.

2. Calculate the integral image. I have used OpenCV implementation (cv2integral) to calculate the integral images. The integral images are calculated using the following formula

3. Calculate the horizontal and vertical Haar features such as [0,1] and [1,0]T . All the horizontal features and the vertical features are calculated using filter of sizes 1x2, 1x4, 1x6 , 1x8 ….. both horizontally and vertically. Thus, we get a total of 166000 for each image in the dataset.

**Build Weak Classifier**

Let T be the number of weak classifier required to build one strong classifier. The following is the procedure to calculate one weak classifier:

1. Extract one feature for all images (positive and negative) in the dataset and arrange the values in ascending.

2. The error for the selected feature is calculated using the following formula given by,

e = min(),

where is the total sum of the negative weights, is the total sum of the positive weights, is the sum of the negative weights below the current example and is the sum of the positive weights below the current example and.

3. The ith weak classifier in the T weak classifiers is defined as

where x is the image, F is the feature, is the threshold and p is the polarity. The sign of p is determined by the value of e, if <), p = -1 else p = 1.

4. Once the ith weak classifier is calculated the weights are updated using the following formula,

, if the sample is correctly classified and 1 if its incorrectly classified. is the weighted error.

**Build a Strong Classifier**

The following is the procedure to build a strong classifier:

1. Labels the images in the training data, the positive images are given the label of one while the negative images are given a label of 0.

2. Set the weights initially for all the images in the training set. All positive images are given a weight of and the negative images are given a weight of .

3. Normalize the weights,

4. For all the features find the best weak feature using the steps described in the section, build a weak classifier.

5. One the best weak classifier is calculated the weights are updated using the following formula,

, if the sample is correctly classified and 1 if its incorrectly classified. is the weighted error

6. Repeat steps 3-5 till you find T weak classifiers. I have set the maximum value of T to be 100 in this experiment.

7. The strong classifier is given by the formula, ,

During the training procedure in order to get a 100 % true detection rate, I have set the threshold to the minimum value of and during training it has been set to .

8. The stopping criteria is for the loop is determined by the true detection and the false positive rate. In this assignment I has set the target true detection rate during training to 1 and the maximum target false positive rate to 0.5

**Cascaded AdaBoost Classifier**

To integrate the existing, the adaBoost Classifier with the cascade algorithm the following procedure is followed:

1. The strong classifier is calculated using T weak classifiers using the procedure described in the section build a strong classifier.

2. Once the strong classifier is identified, the training data set is updated so the that contains all the positive image and the negative images that was misclassified by the previous strong classifier. This updated dataset is then used to generate and the second strong classifier.

3. This process is repeated till the updated training data set only contains positive images. Thus, ensuring a false positive rate of 0.

**Performance Analysis**

The performance analysis of the AdaBoost classifier is done by calculating the false positive and false negative rates during testing.

**Results from Task 2**

**Training results:**

Table showing false positive rates during training

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Stage | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| No. of Classifiers | 9 | 11 | 8 | 7 | 8 | 6 | 5 |
| Accumulate False positive rates | 0.45278 | 0.226394 | 0.080205 | 0.031286 | 0.010808 | 0.000568 | 0 |

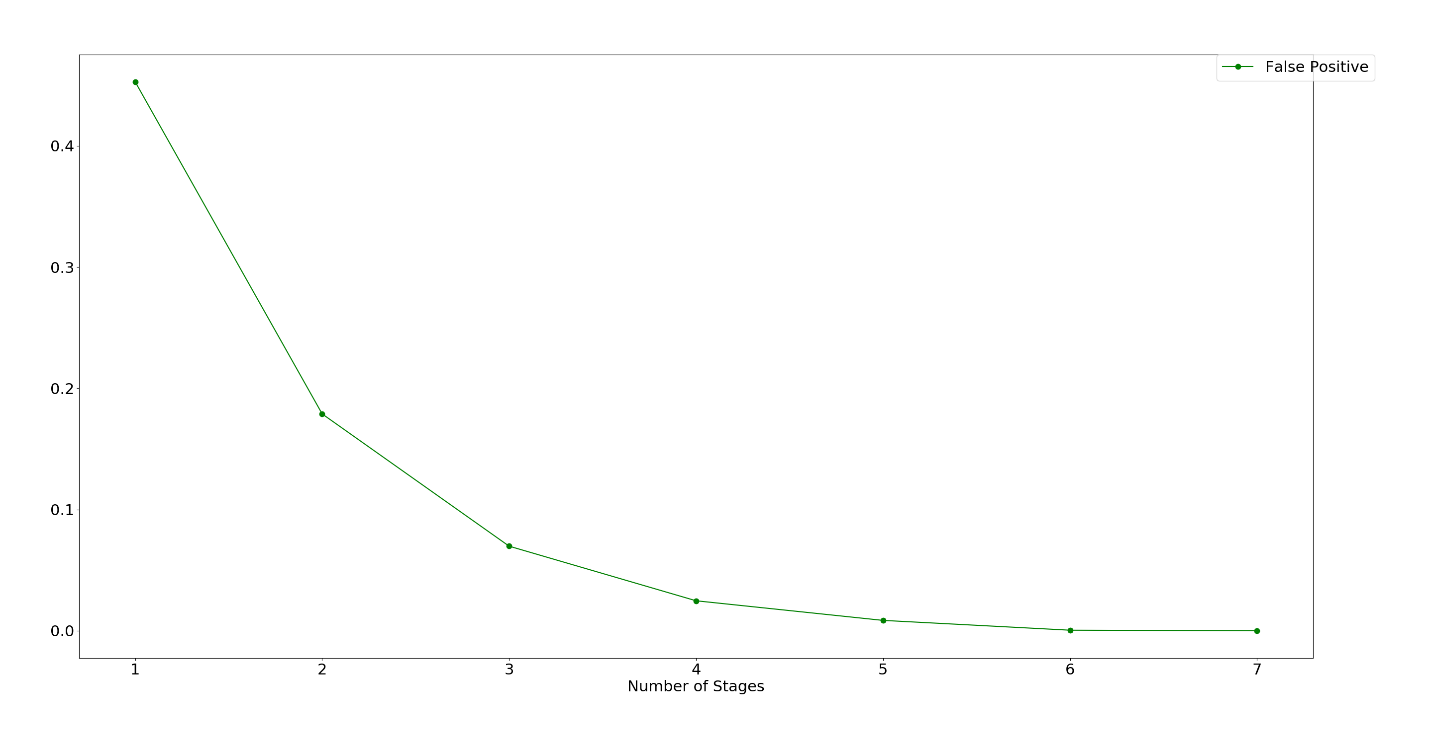


Figure 2: Accumulated false positive rates vs number of stages during training

**Training results:**

Table showing accumulates false positive and false negative rates during testing

|  |  |  |
| --- | --- | --- |
| No of stages | Accumulated False Positive Rates | Accumulates False Negative Rates |
| 1 | 0.1 | 0.12921348 |
| 2 | 0.00386364 | 0.2808671 |
| 3 | 0.000333678 | 0.34550845 |
| 4 | 5.081e-05 | 0.41536991 |
| 5 | 1.20096e-05 | 0.46792093 |
| 6 | 4.66738e-06 | 0.49781301 |
| 7 | 4.65677e-06 | 0.50063429 |

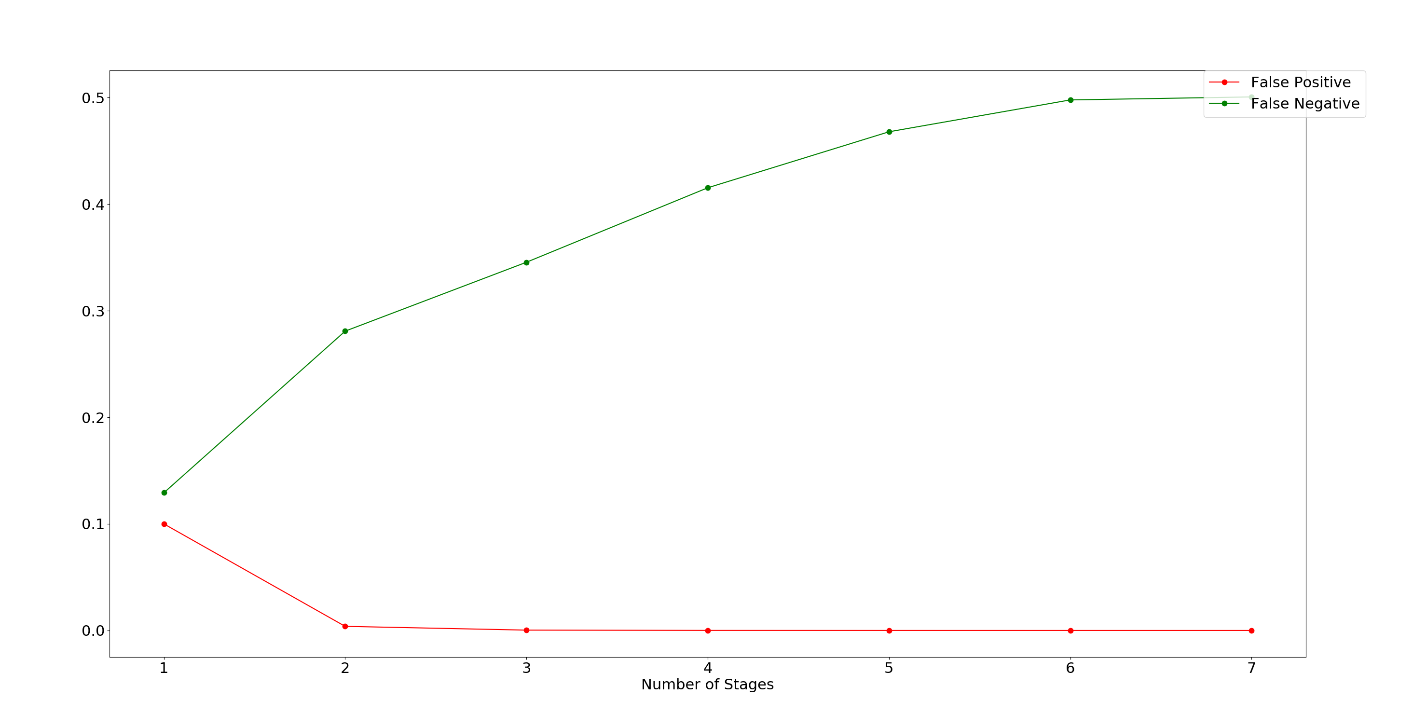


Figure 3: Accumulated false positive and false negative rates vs number of stages during testing

**Conclusions:**

From the graphs it is seen that the false positive rates decrease as the number of stages increase. The false negative rate is however seen to increase as the number of stages increase as the number of stages increase. This is because the false negative rate is equal to 1 - True Positive Rate, as the number of stages increase true positive rate decreases and thus false negative rate increases.