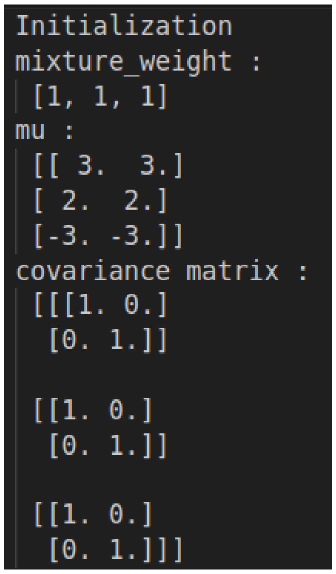
Name: Phonphrm Thawatdamrongkit
Student ID: 2330802

Homework 3 Fisherface

T1.
<u>ANSWER</u>

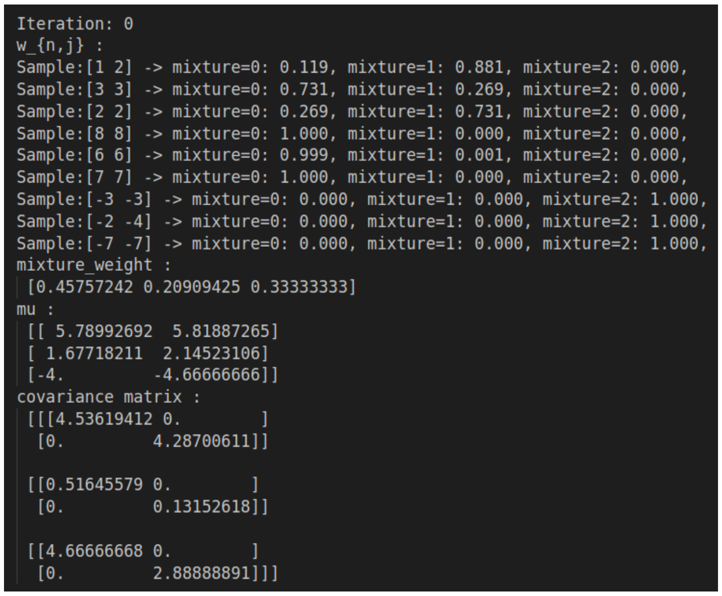
This is the initialization setting where all mixture weights are equal.



```
Initialization
mixture_weight :
 [1, 1, 1]
mu :
 [[ 3.  3.]
 [ 2.  2.]
 [-3. -3.]]
covariance matrix :
 [[[1. 0.]
  [0. 1.]]

 [[1. 0.]
  [0. 1.]]

 [[1. 0.]
  [0. 1.]]]
```

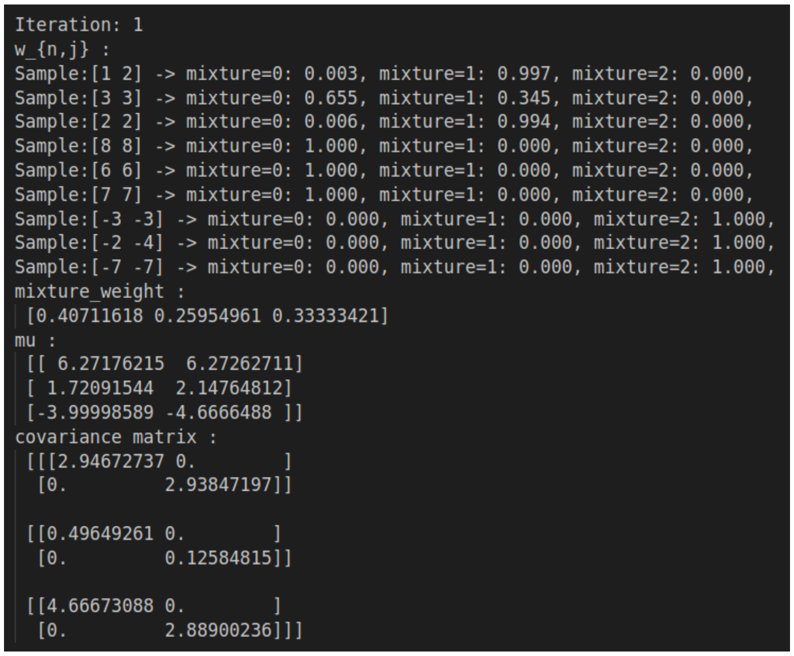
The following three images are the three iterations of EM algorithm:

First Iteration:



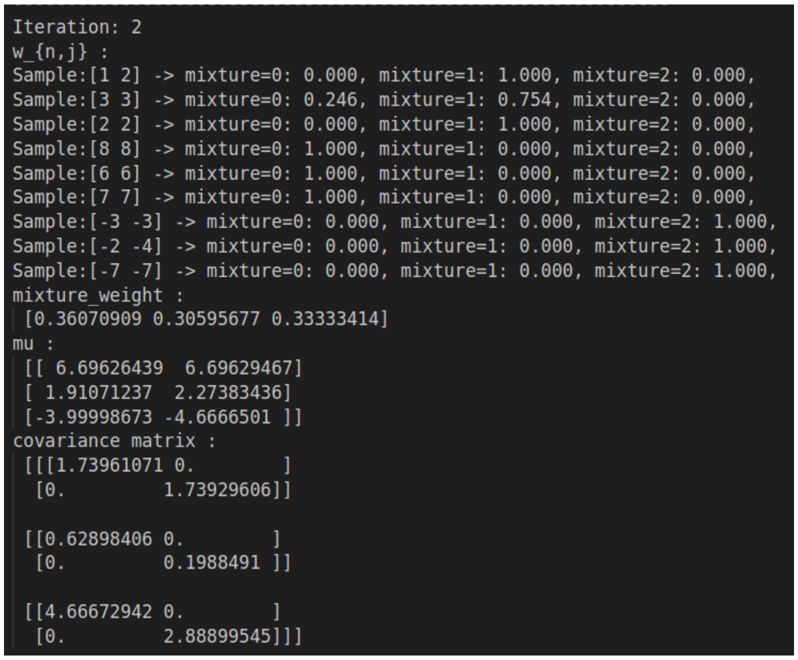
```
Iteration: 0
w_{n,j} :
Sample:[1 2] -> mixture=0: 0.119, mixture=1: 0.881, mixture=2: 0.000,
Sample:[3 3] -> mixture=0: 0.731, mixture=1: 0.269, mixture=2: 0.000,
Sample:[2 2] -> mixture=0: 0.269, mixture=1: 0.731, mixture=2: 0.000,
Sample:[8 8] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[6 6] -> mixture=0: 0.999, mixture=1: 0.001, mixture=2: 0.000,
Sample:[7 7] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[-3 -3] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
Sample:[-2 -4] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
Sample:[-7 -7] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
mixture_weight :
 [0.45757242 0.20909425 0.33333333]
mu :
 [[ 5.78992692  5.81887265]
 [ 1.67718211  2.14523106]
 [-4.         -4.66666666]]
covariance matrix :
 [[[4.53619412 0.        ]
  [0.         4.28700611]]

 [[0.51645579 0.        ]
  [0.         0.13152618]]

 [[4.66666668 0.        ]
  [0.         2.88888891]]]
```

Second Iteration:



```
Iteration: 1
w_{n,j} :
Sample:[1 2] -> mixture=0: 0.003, mixture=1: 0.997, mixture=2: 0.000,
Sample:[3 3] -> mixture=0: 0.655, mixture=1: 0.345, mixture=2: 0.000,
Sample:[2 2] -> mixture=0: 0.006, mixture=1: 0.994, mixture=2: 0.000,
Sample:[8 8] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[6 6] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[7 7] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[-3 -3] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
Sample:[-2 -4] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
Sample:[-7 -7] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
mixture_weight :
 [0.40711618 0.25954961 0.33333421]
mu :
 [[ 6.27176215  6.27262711]
 [ 1.72091544  2.14764812]
 [-3.99998589 -4.6666488 ]]
covariance matrix :
 [[[2.94672737 0.        ]
  [0.         2.93847197]]

 [[0.49649261 0.        ]
  [0.         0.12584815]]

 [[4.66673088 0.        ]
  [0.         2.88900236]]]
```

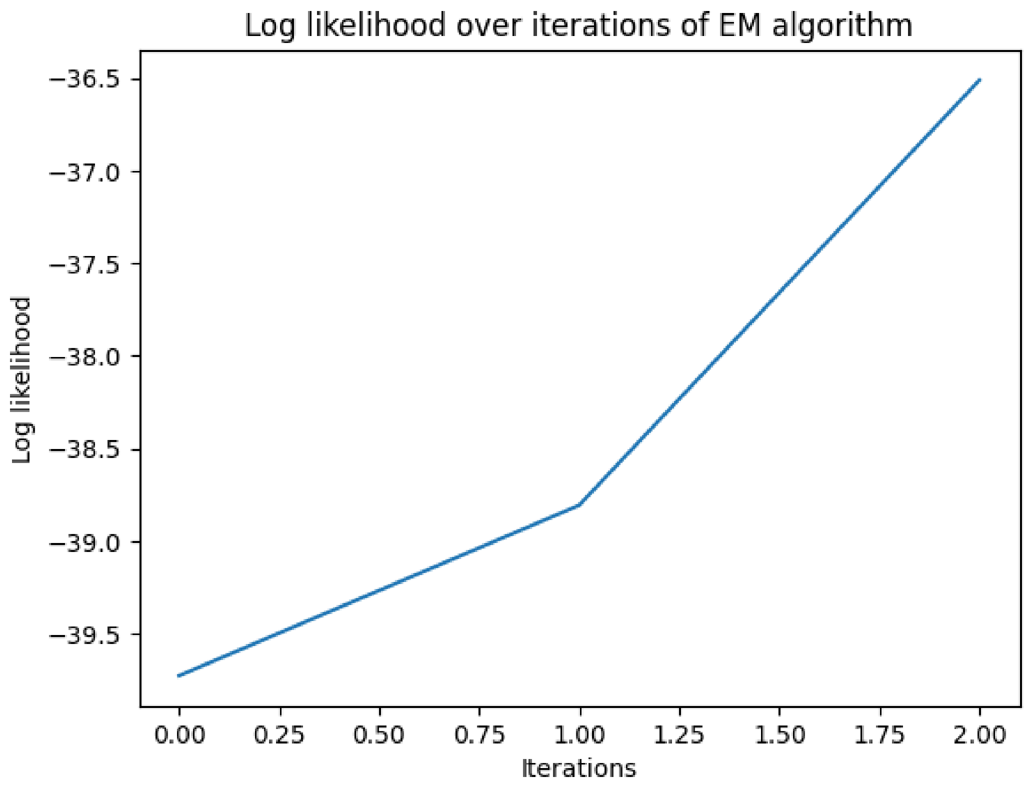
Third Iteration:



```
Iteration: 2
w_{n,j} :
Sample:[1 2] -> mixture=0: 0.000, mixture=1: 1.000, mixture=2: 0.000,
Sample:[3 3] -> mixture=0: 0.246, mixture=1: 0.754, mixture=2: 0.000,
Sample:[2 2] -> mixture=0: 0.000, mixture=1: 1.000, mixture=2: 0.000,
Sample:[8 8] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[6 6] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[7 7] -> mixture=0: 1.000, mixture=1: 0.000, mixture=2: 0.000,
Sample:[-3 -3] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
Sample:[-2 -4] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
Sample:[-7 -7] -> mixture=0: 0.000, mixture=1: 0.000, mixture=2: 1.000,
mixture_weight :
 [0.36070909 0.30595677 0.33333414]
mu :
 [[ 6.69626439  6.69629467]
 [ 1.91071237  2.27383436]
 [-3.99998673 -4.6666501 ]]
covariance matrix :
 [[[1.73961071 0.        ]
  [0.         1.73929606]]

 [[0.62898406 0.        ]
  [0.         0.1988491 ]]

 [[4.66672942 0.        ]
  [0.         2.88899545]]]
```

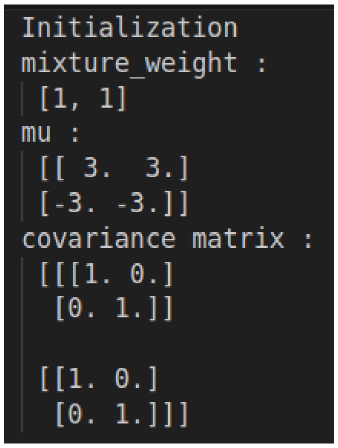
T2.
ANSWER

The likelihood goes up every iteration just as we learned in class.



T3.
ANSWER

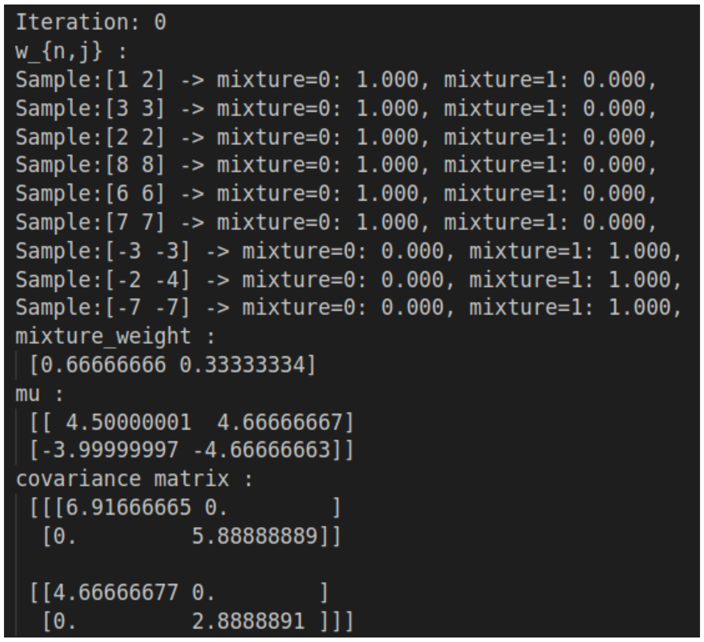
As we changed the number of mixtures from 3 to 2, the following images are the initialization.



```
Initialization
mixture_weight :
 [1, 1]
mu :
 [[ 3.  3.]
 [-3. -3.]]
covariance matrix :
 [[[1. 0.]
  [0. 1.]]

 [[1. 0.]
  [0. 1.]]]
```

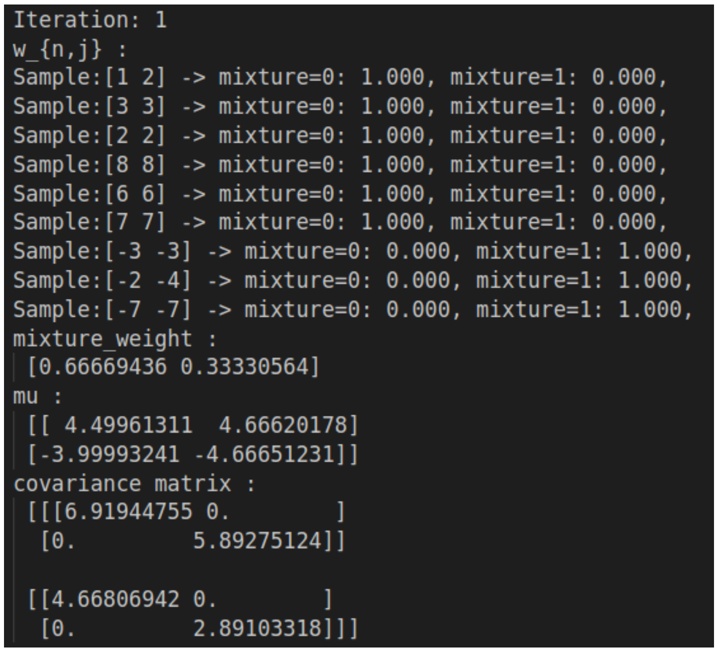
The following three images are the three iterations of EM algorithm:

First Iteration:



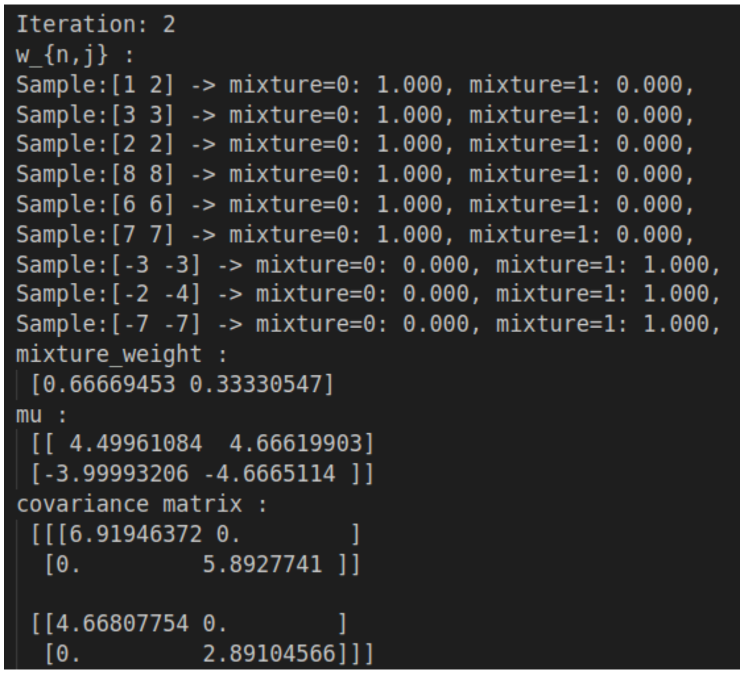
```
Iteration: 0
w_{n,j} :
Sample:[1 2] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[3 3] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[2 2] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[8 8] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[6 6] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[7 7] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[-3 -3] -> mixture=0: 0.000, mixture=1: 1.000,
Sample:[-2 -4] -> mixture=0: 0.000, mixture=1: 1.000,
Sample:[-7 -7] -> mixture=0: 0.000, mixture=1: 1.000,
mixture_weight :
 [0.66666666 0.33333334]
mu :
 [[ 4.50000001  4.66666667]
 [-3.99999997 -4.66666663]]
covariance matrix :
 [[[6.91666665 0.        ]
  [0.         5.88888889]]

 [[4.66666677 0.        ]
  [0.         2.8888891 ]]]
```

Second Iteration:



```
Iteration: 1
w_{n,j} :
Sample:[1 2] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[3 3] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[2 2] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[8 8] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[6 6] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[7 7] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[-3 -3] -> mixture=0: 0.000, mixture=1: 1.000,
Sample:[-2 -4] -> mixture=0: 0.000, mixture=1: 1.000,
Sample:[-7 -7] -> mixture=0: 0.000, mixture=1: 1.000,
mixture_weight :
 [0.66669436 0.33330564]
mu :
 [[ 4.49961311  4.66620178]
 [-3.99993241 -4.66651231]]
covariance matrix :
 [[[6.91944755 0.        ]
  [0.         5.89275124]]

 [[4.66806942 0.        ]
  [0.         2.89103318]]]
```

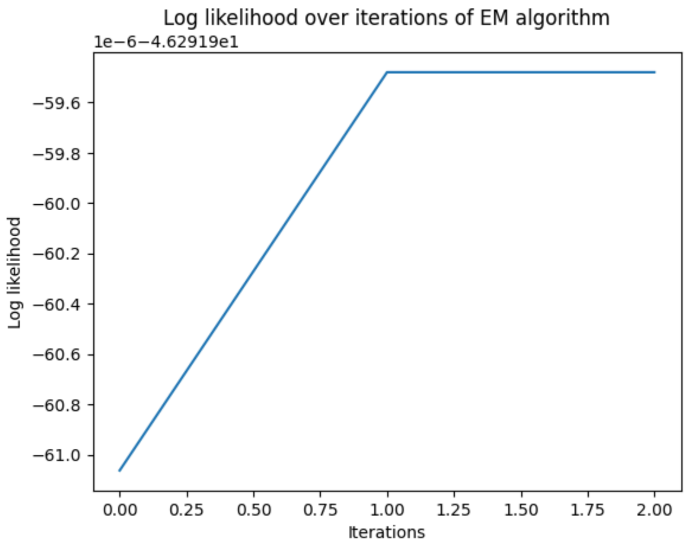
Third Iteration:



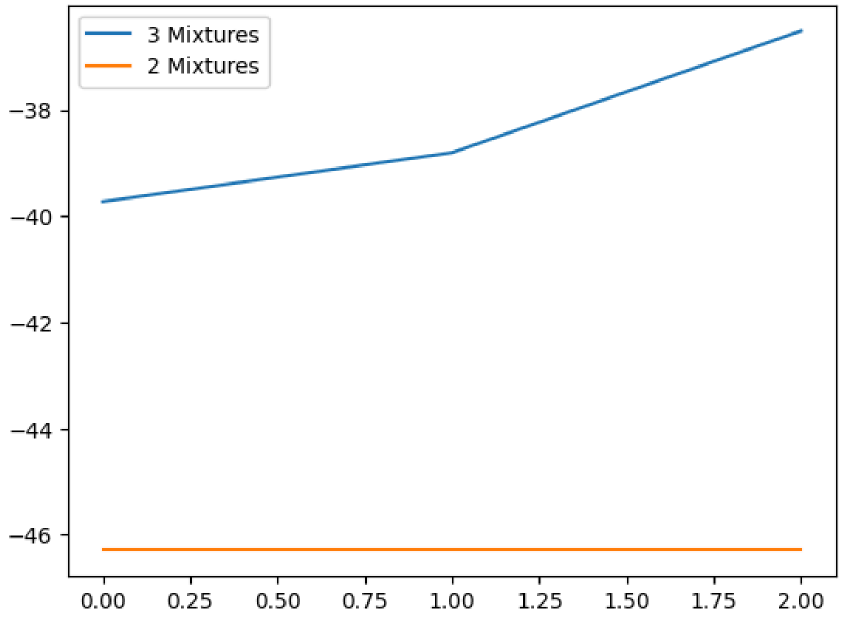
```
Iteration: 2
w_{n,j} :
Sample:[1 2] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[3 3] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[2 2] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[8 8] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[6 6] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[7 7] -> mixture=0: 1.000, mixture=1: 0.000,
Sample:[-3 -3] -> mixture=0: 0.000, mixture=1: 1.000,
Sample:[-2 -4] -> mixture=0: 0.000, mixture=1: 1.000,
Sample:[-7 -7] -> mixture=0: 0.000, mixture=1: 1.000,
mixture_weight :
 [0.66669453 0.33330547]
mu :
 [[ 4.49961084  4.66619903]
 [-3.99993206 -4.6665114 ]]
covariance matrix :
 [[[6.91946372 0.        ]
  [0.         5.8927741 ]]

 [[4.66807754 0.        ]
  [0.         2.89104566]]]
```

T4.
<u>ANSWER</u>

The image below shows the likelihood of two mixtures' settings.



The next figure depicts the comparison between two settings (two and three mixtures) in the same plot. Apparently, we can see that the three mixtures' settings provide a better likelihood than two mixtures.:



OT1.
ANSWER

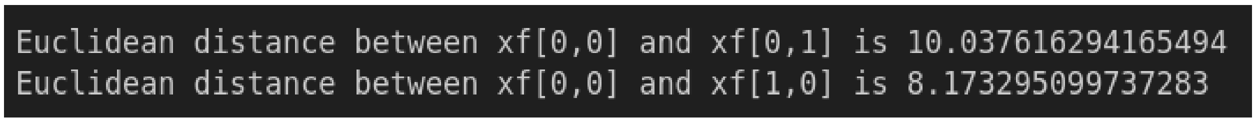
In this case with three iterations, the initialization with this extreme case causes the model to ignore the gaussian with mean (10000, 10000) and all data points converge to gaussian with mean (0,0). This is due to E-step trying to calculate the probability of every data point with respect to each gaussian so the probability that our data points, which is very small value, reside in the gaussian with mean (10000, 10000) is zero.

Thus, the good initialization trick for GMM is to initialize the starting mean with the help of K-means algorithm which clusters the data into groups with the means as representative of each group. Therefore, we can use those means as an initialization for our GMM. This method could guarantee that our initial mean will not be so extreme.

T5.

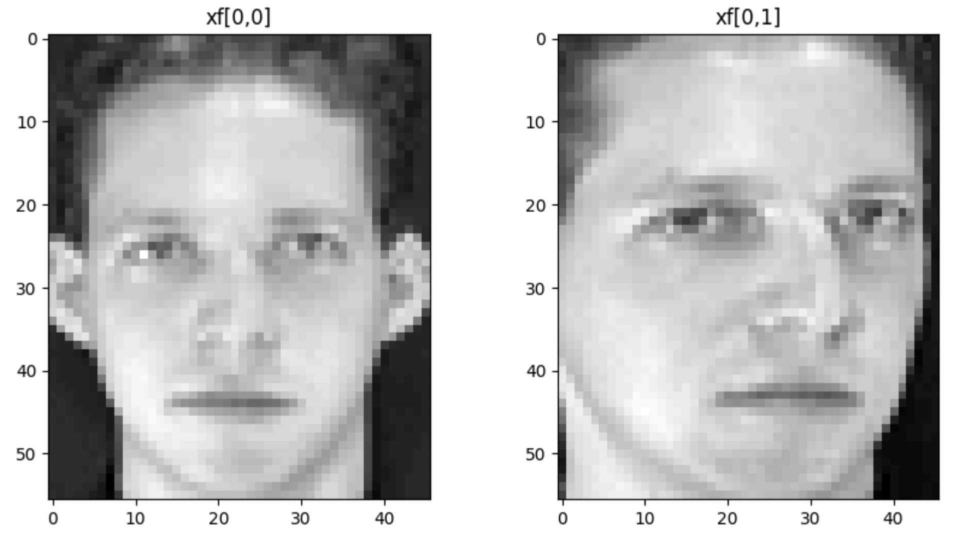
<u>ANSWER</u>

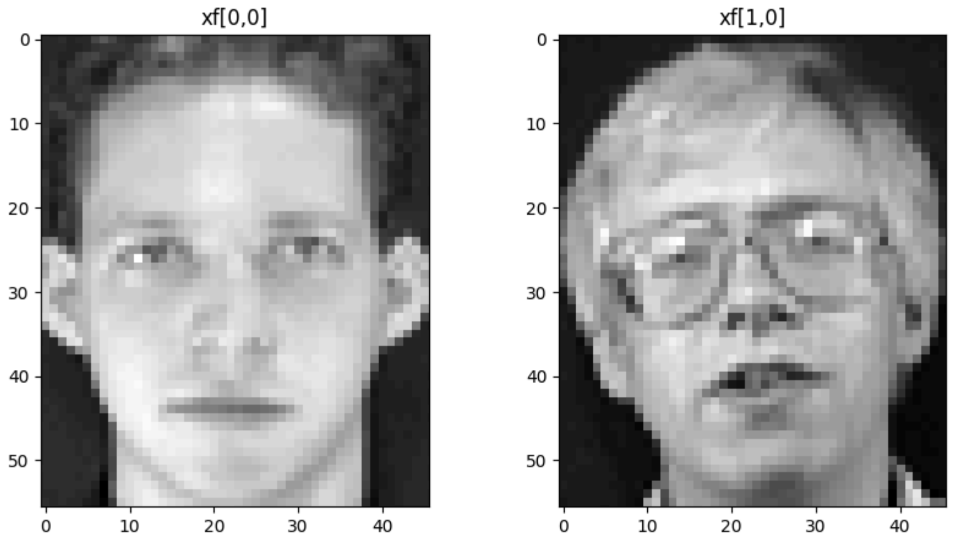
The Euclidean distance results are shown below which doesn't provide the expected number that we wanted. The numbers depict that the image from the same person has a higher distance than the image from a different person.



```
Euclidean distance between xf[0,0] and xf[0,1] is 10.037616294165494
Euclidean distance between xf[0,0] and xf[1,0] is 8.173295099737283
```

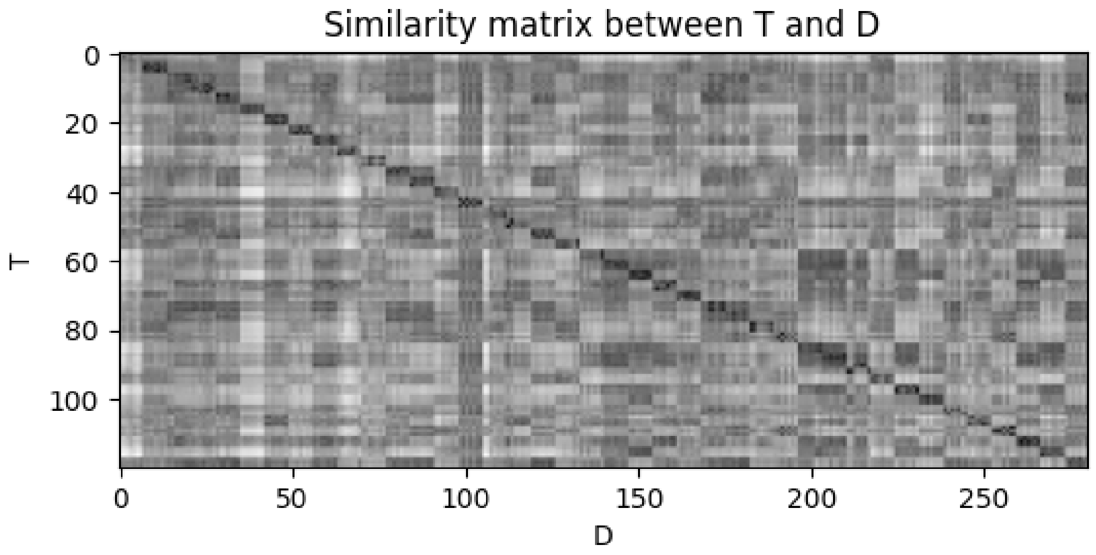
This might be because the orientation of the face is very different for the same person but the image from a different person has the same orientation so the distance is much smaller. Since, the Euclidean distance measures pixel by pixel so that it might be possible that orientation can play a crucial role for this face verification.





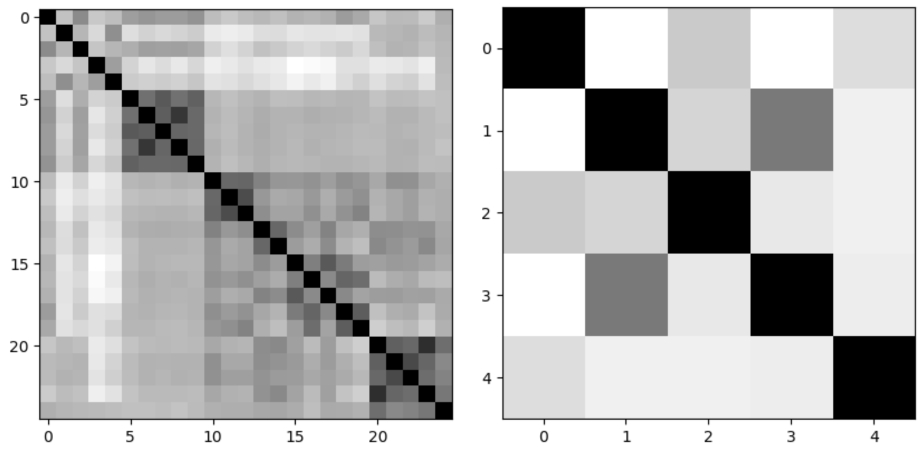
Therefore, the Euclidean distance numbers could be useful for face verification if we have fixed orientation of faces dataset (as little variation as possible).

T6.
<u>ANSWER</u>

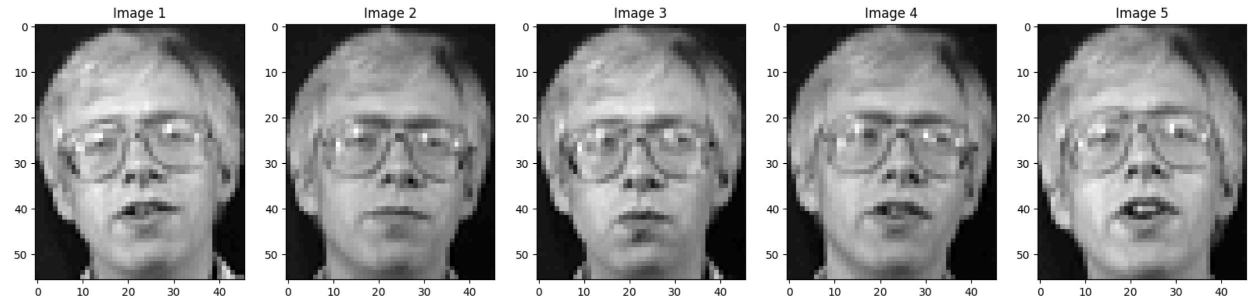


T7.
<u>ANSWER</u>

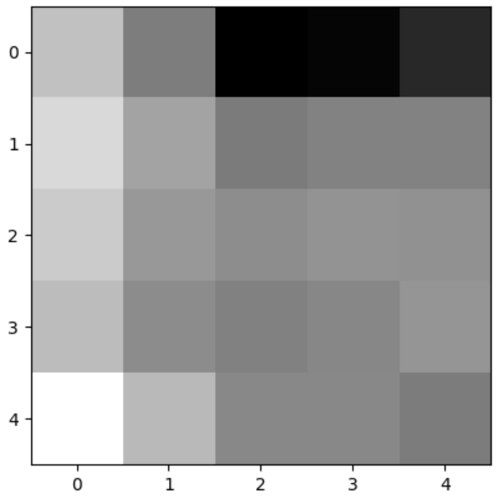
From the given example of similarity matrix above which use T and D as the same for both axes, the dataset used for making similarity matrix is 5 images from the first 5 people. Also, the area between [5:10, 5:10] is shown next to its main similarity matrix.



The pattern of black squares between [5:10, 5:10] in this similarity matrix shows that the black squares are from the same person which has lower Euclidean distance as shown below.



For the same pattern for the previous similarity matrix with the same area [5:10, 5:10] are shown below where the black squares represents the image from the same person as well:



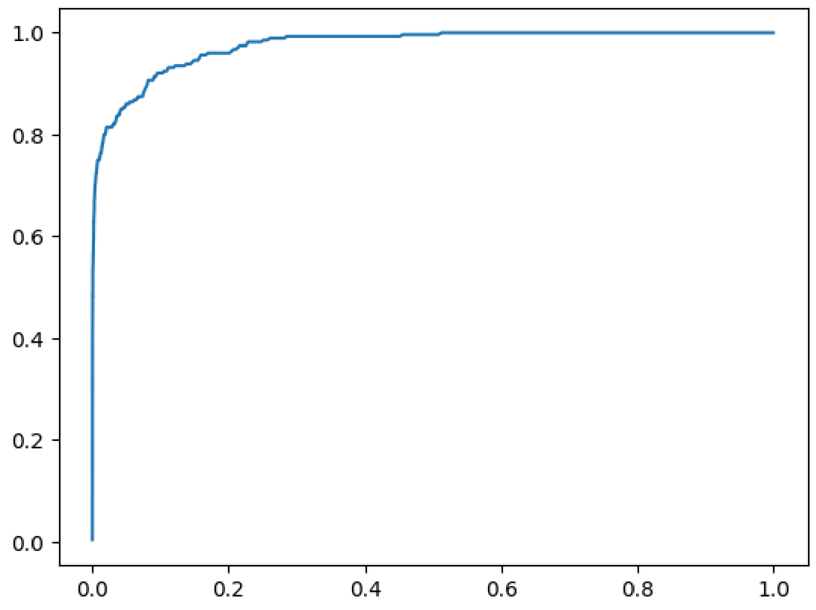
T8.
<u>ANSWER</u>

The output from threshold value equals to 10 is shown below:

- True positive rate: **0.9964285714285714**
- False alarm rate: **0.4564102564102564**

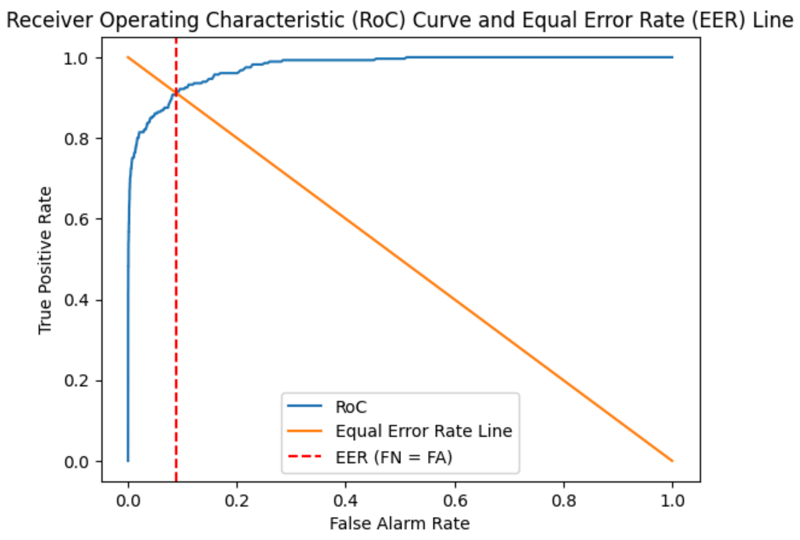
T9.
<u>ANSWER</u>

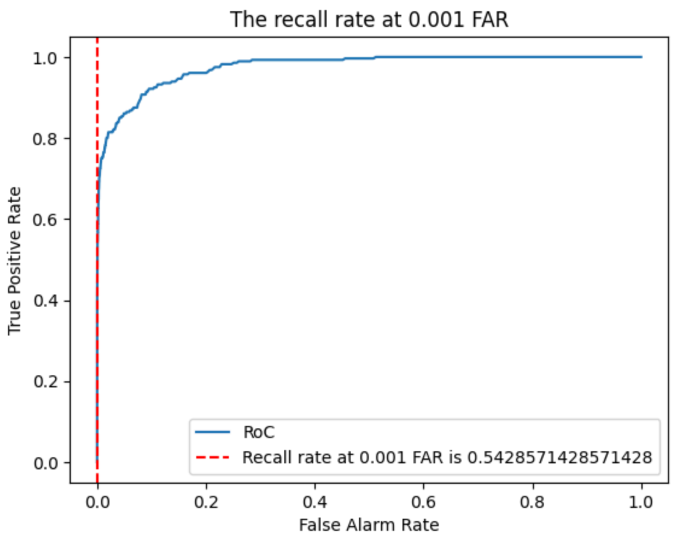
The minimum threshold for generating a RoC curve is the minimum value in the similarity matrix and the maximum threshold is the maximum value in the same similarity matrix. The RoC curve are shown below:



T10.
<u>ANSWER</u>

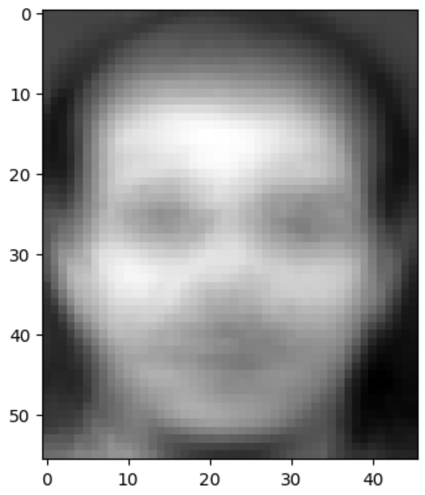
The Equal Error Rate is **0.9071428571428571** from my method and recall rate at 0.1% false alarm rate is **0.5428571428571428**. The images below show the plotting of EER and Recall.


Receiver Operating Characteristic (RoC) Curve and Equal Error Rate (EER) Line
True Positive Rate
False Alarm Rate
RoC
Equal Error Rate Line
EER (FN = FA)


The recall rate at 0.001 FAR
True Positive Rate
False Alarm Rate
RoC
Recall rate at 0.001 FAR is 0.5428571428571428

T11.

<u>ANSWER</u>



T12.
ANSWER

The size of the covariance matrix is **2576x2576** which is the flatten of the image's pixel and the rank of the covariance matrix is **119** which is the number of samples in the training set minus 1 (since we subtract mean for all images).

T13.
ANSWER

The size of Gram matrix is equal to the number of samples in the training set which is **120x120** and the rank of the Gram matrix equals to the rank of covariance matrix which is **119**. If we compute the eigenvalues from this Gram matrix, we will get the number of non-zero eigenvalues of **119** which is the same as rank.

T14.
ANSWER

The Gram matrix is indeed **symmetric** because we can proof that by the following:

$$(X^TX)^T = X^T(X^T)^T = X^TX$$

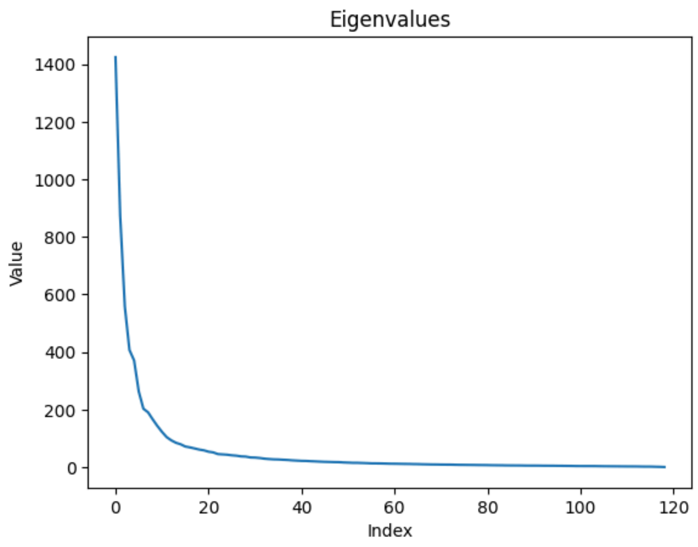
This shows that even if we transpose the matrix again, it gives back the same result.

T15.
ANSWER

Number of non-zero eigenvalues: **119**
The value of eigenvalue that is zero: **2.9825038297695136e-14**

T16.

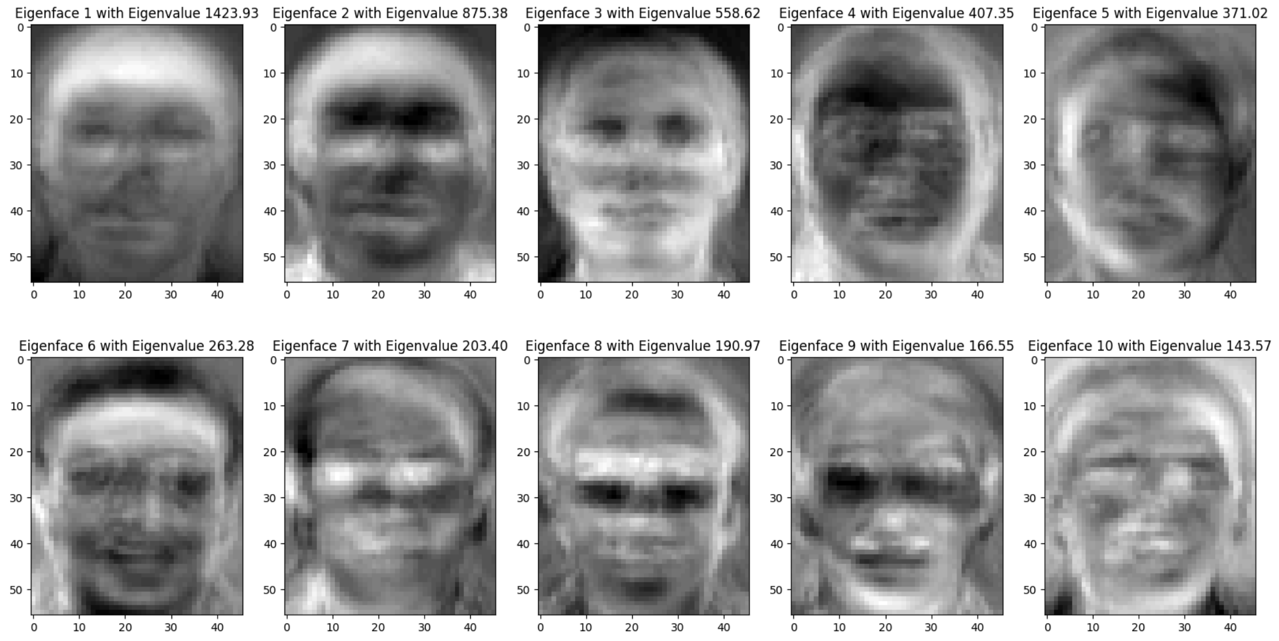
<u>ANSWER</u>



If we want to keep 95% of the variance, we will have **64** eigenvectors.

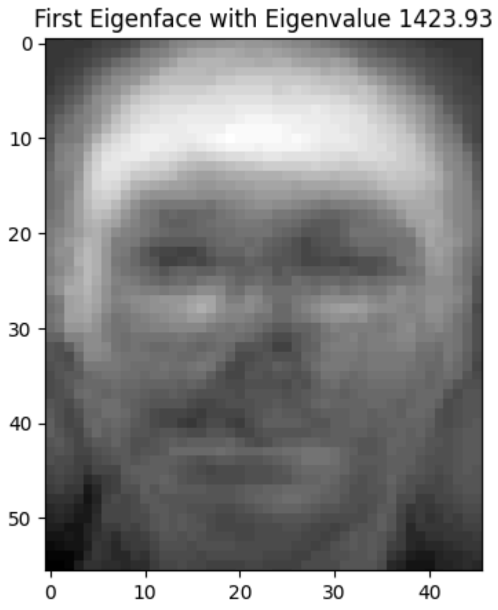
T17.

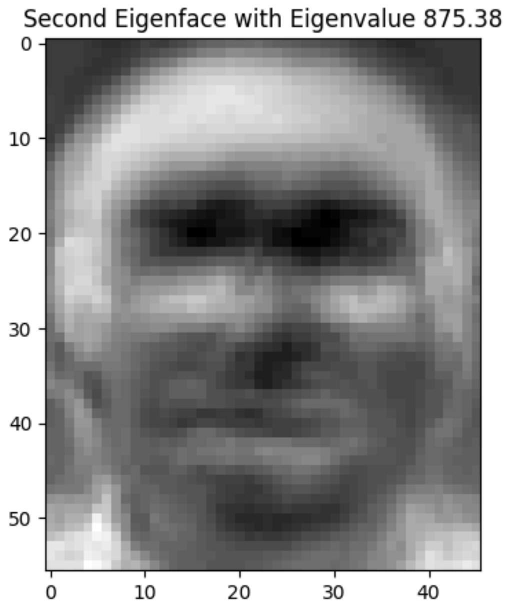
<u>ANSWER</u>

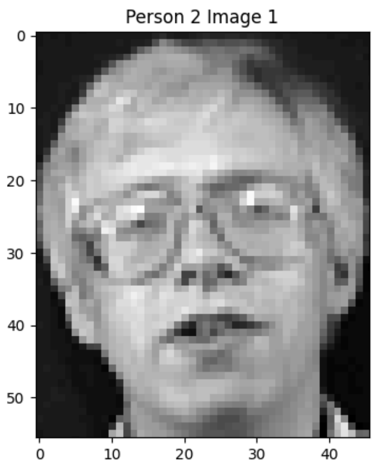

Eigenface 1 with Eigenvalue 1423.93
Eigenface 2 with Eigenvalue 875.38
Eigenface 3 with Eigenvalue 558.62
Eigenface 4 with Eigenvalue 407.35
Eigenface 5 with Eigenvalue 371.02
Eigenface 6 with Eigenvalue 263.28
Eigenface 7 with Eigenvalue 203.40
Eigenface 8 with Eigenvalue 190.97
Eigenface 9 with Eigenvalue 166.55
Eigenface 10 with Eigenvalue 143.57

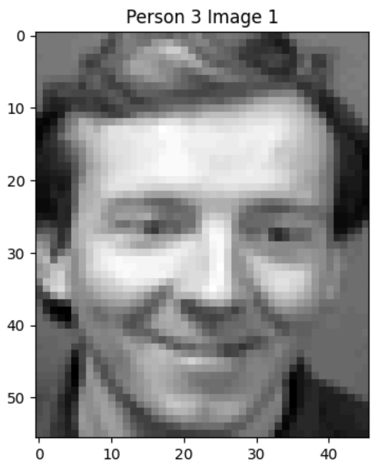
T18.
<u>ANSWER</u>

Since the these eigenvectors are derived from the covariance matrix which captures the most variance direction when doing the projection on these vectors, the first eigenvector(from sorted eigenvalues in descending order) represents the direction of the most variance in the data which in this case clearly represents the **hair** which is highlighted in white area. The second eigenvector highlighted **the area under the eyes, the outline of the face, a little of the neck area, and some area around the mouth which could refer to mouth shape**. This makes sense because as we can see from the dataset people usually have different hair style and face shape and details.



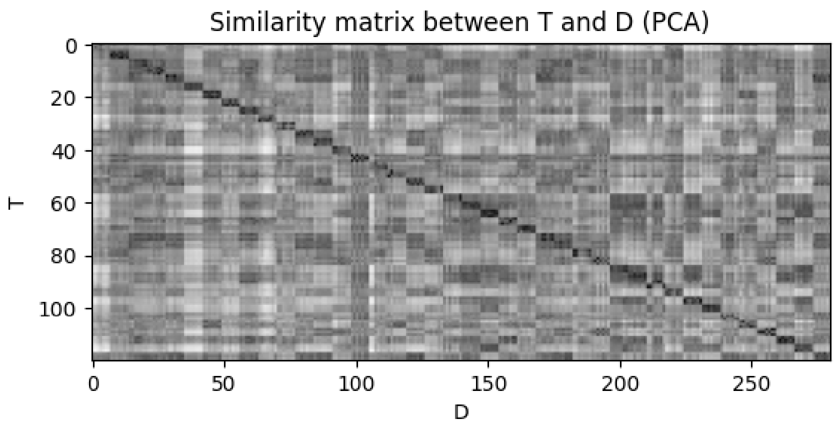


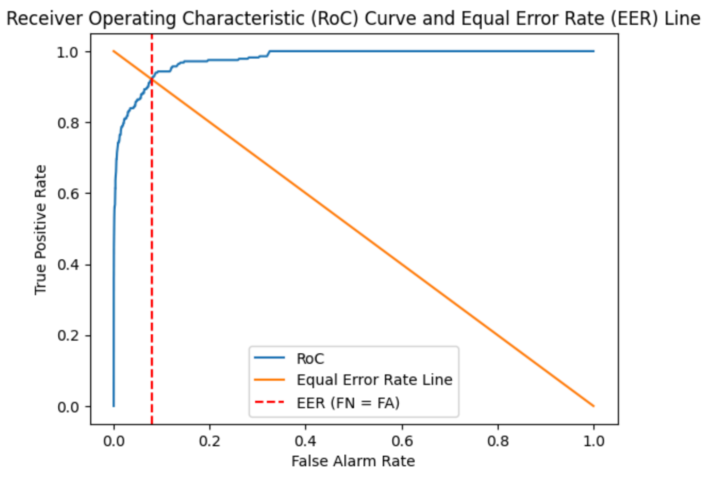


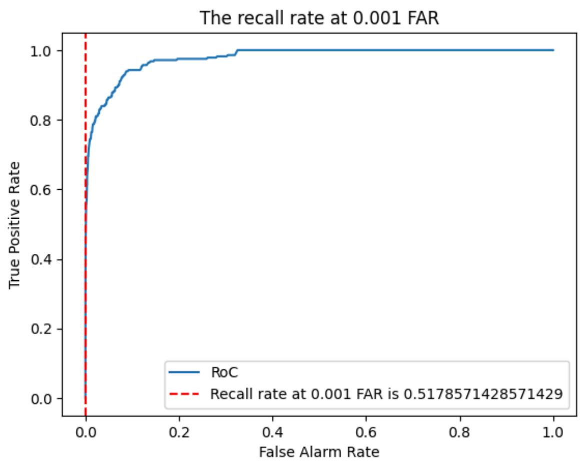


T19.
<u>ANSWER</u>

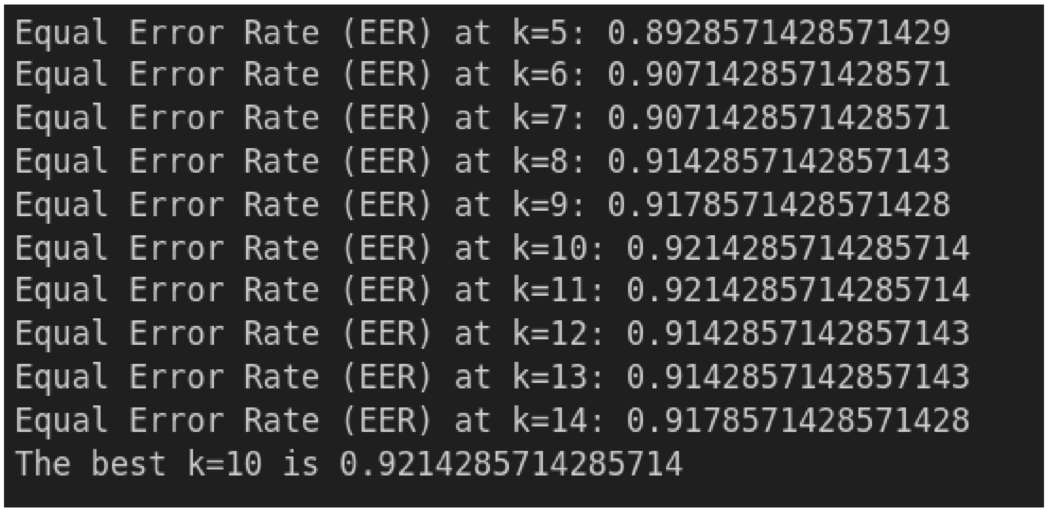
The EER is **0.9214285714285714** and the Recall is **0.5178571428571429**.







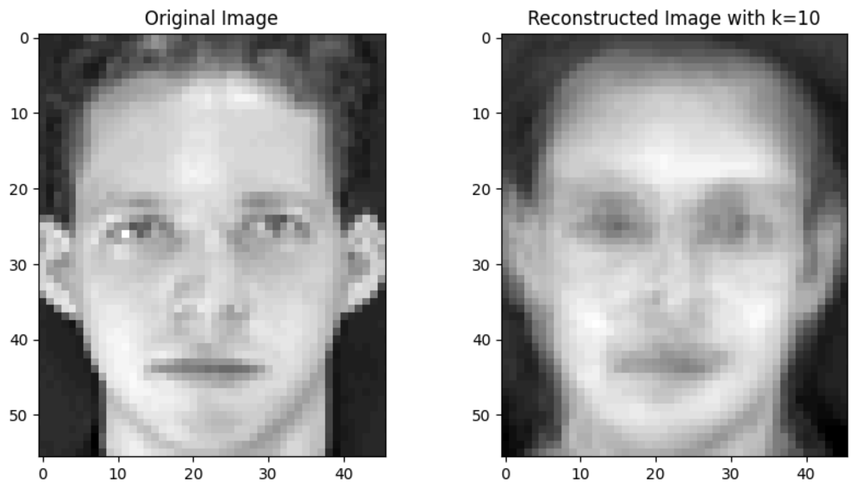
T20.
<u>ANSWER</u>



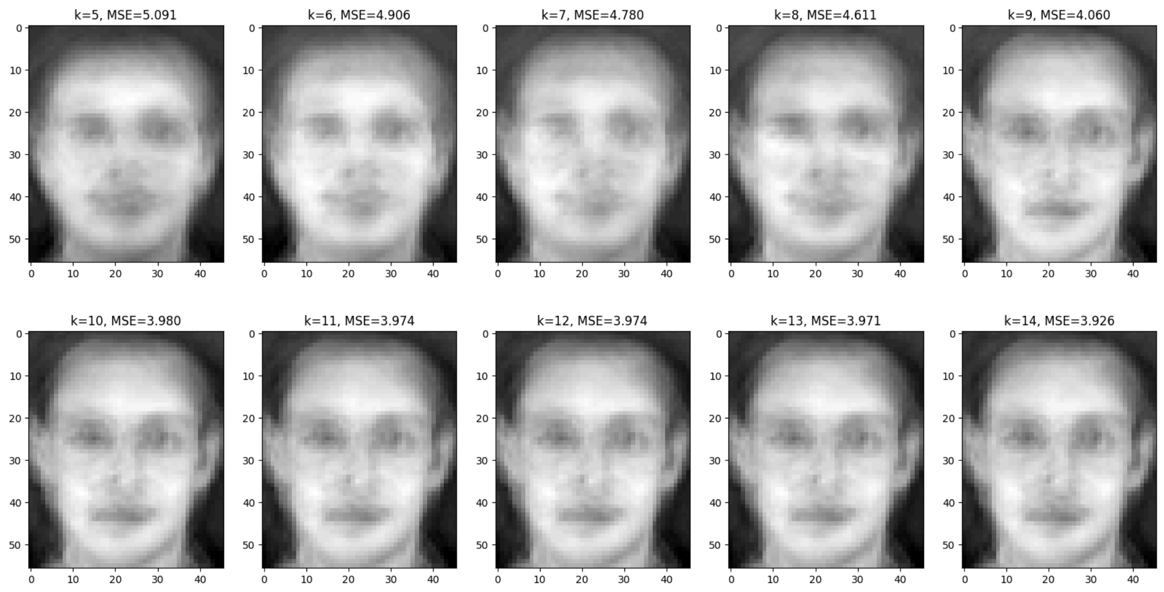
```
Equal Error Rate (EER) at k=5: 0.8928571428571429
Equal Error Rate (EER) at k=6: 0.9071428571428571
Equal Error Rate (EER) at k=7: 0.9071428571428571
Equal Error Rate (EER) at k=8: 0.9142857142857143
Equal Error Rate (EER) at k=9: 0.9178571428571428
Equal Error Rate (EER) at k=10: 0.9214285714285714
Equal Error Rate (EER) at k=11: 0.9214285714285714
Equal Error Rate (EER) at k=12: 0.9142857142857143
Equal Error Rate (EER) at k=13: 0.9142857142857143
Equal Error Rate (EER) at k=14: 0.9178571428571428
The best k=10 is 0.9214285714285714
```

OT2.
<u>ANSWER</u>

The MSE between the original and reconstructed image with k=10 is **3.980**.

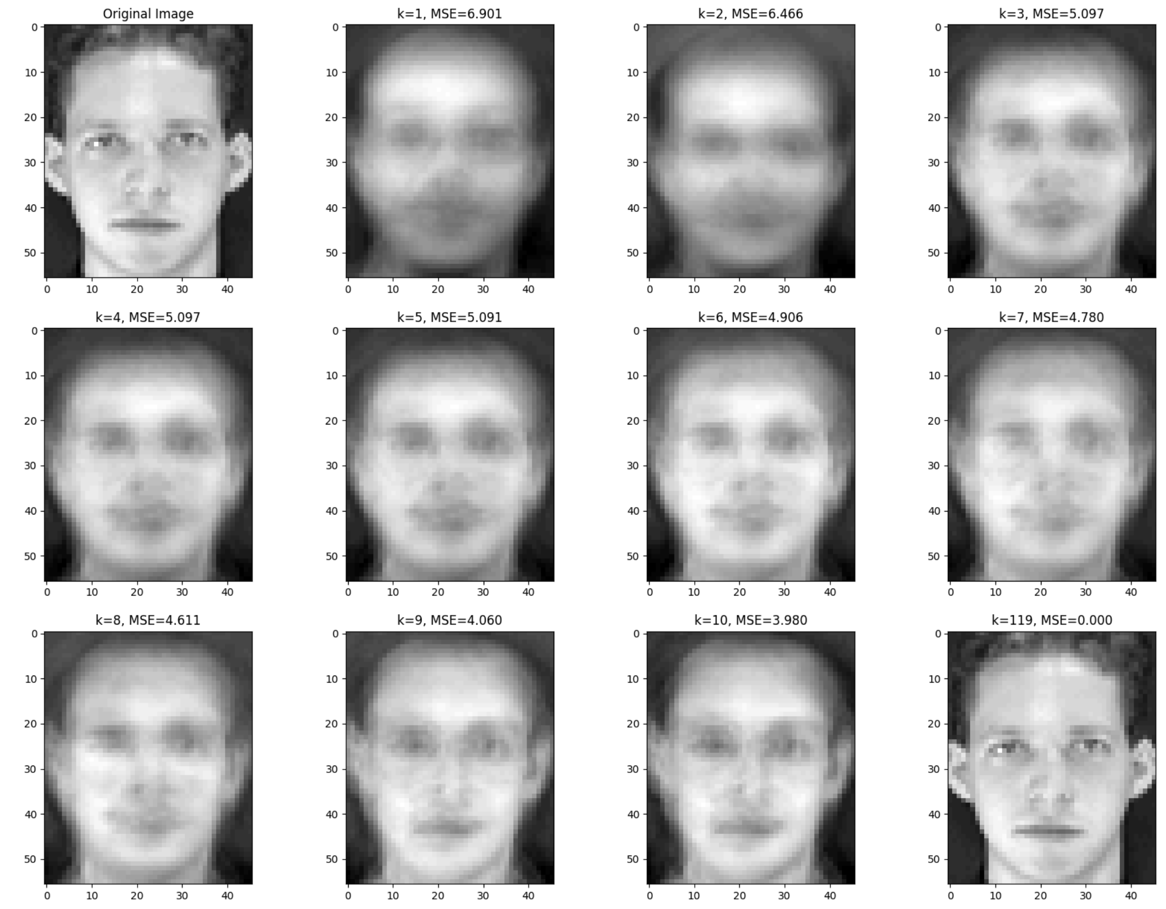


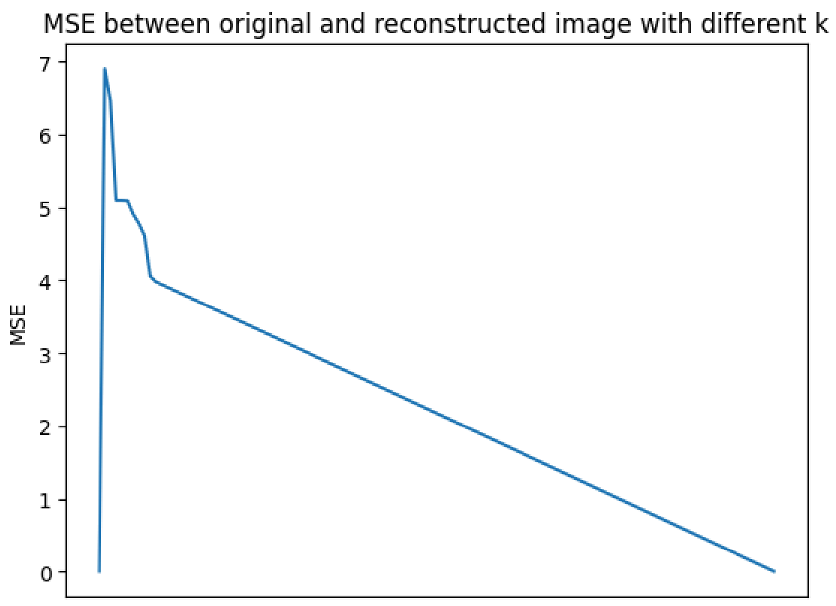
Moreover, the figure below shows the various value of **k**:


k=5, MSE=5.091
k=6, MSE=4.906
k=7, MSE=4.780
k=8, MSE=4.611
k=9, MSE=4.060
k=10, MSE=3.980
k=11, MSE=3.974
k=12, MSE=3.974
k=13, MSE=3.971
k=14, MSE=3.926

OT3.

<u>ANSWER</u>


Original Image
k=1, MSE=6.901
k=2, MSE=6.466
k=3, MSE=5.097
k=4, MSE=5.097
k=5, MSE=5.091
k=6, MSE=4.906
k=7, MSE=4.780
k=8, MSE=4.611
k=9, MSE=4.060
k=10, MSE=3.980
k=119, MSE=0.000



OT4.
<u>ANSWER</u>

The space we need to store 1,000,000 images is 1,000,000x2576x32 bits which is equal to **9826.67MB**. If we compress the database by using the first 10 eigenvalues, we need

- 1,000,000x10x32 bits for 10 eigenvalues (projection values)
- 10x2576x32 bits for 10 eigenfaces
- 2576x32 bits for the mean face

The total of compressed space is **38.256MB** which is around **256 times** smaller.

T21.
<u>ANSWER</u>

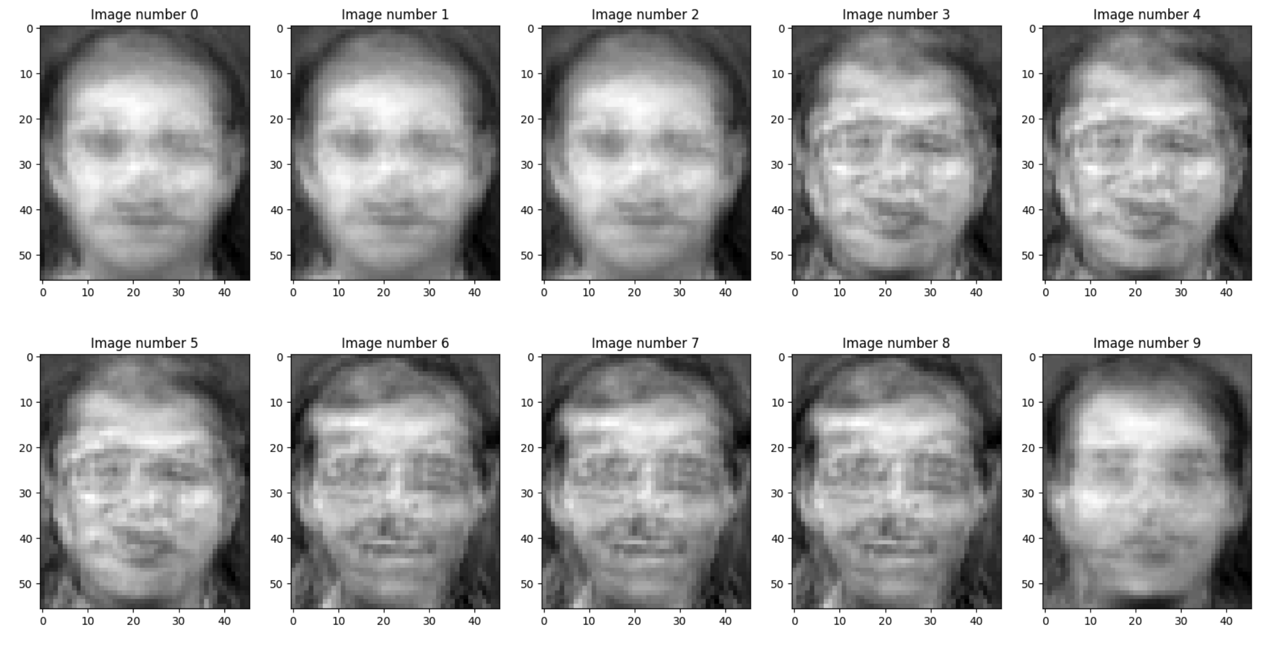
The number of PCA dimensions to keep in order for S_W to be full rank is **80** which is from the number of samples minus number of classes.

T22.
<u>ANSWER</u>

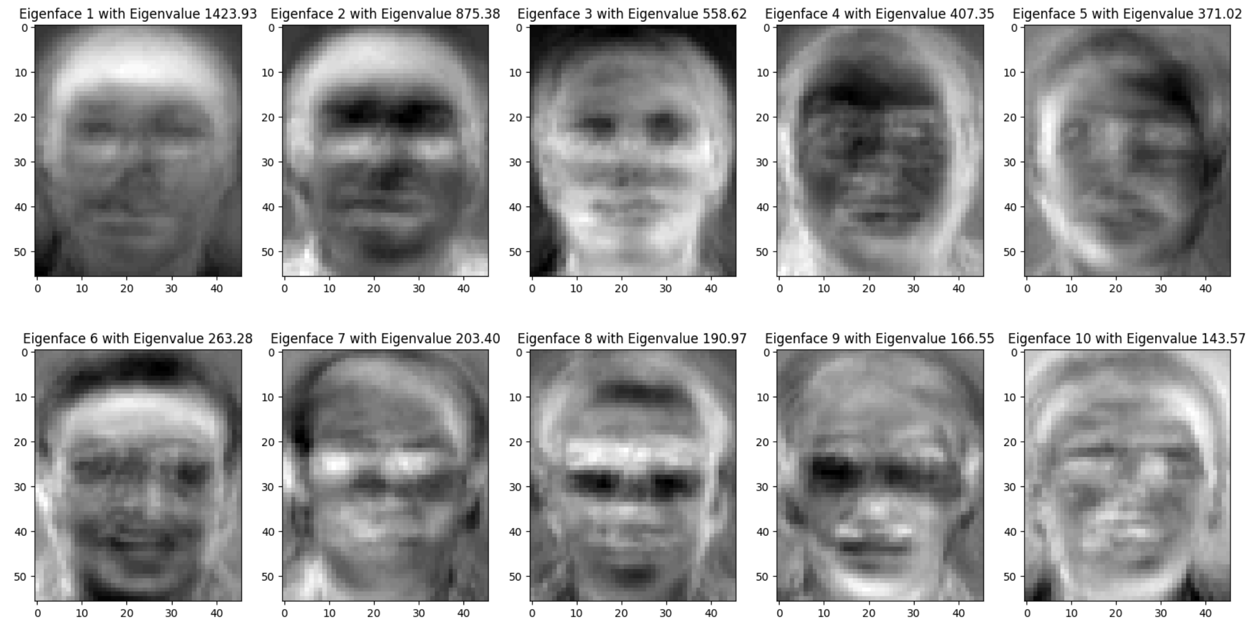
The $S_W^{-1}S_B$ is **not symmetric** unlike the PCA's covariance matrix which is symmetric. From PCA, we can use the method "numpy.linalg.eigh" (requires symmetric) to find the eigenvectors and eigenvalues but for the LDA we need to use "numpy.linalg.eig" which might contain some complex numbers. I tried both methods and found out that the "numpy.linalg.eigh" output has a rank that doesn't correspond to what I expected. However, the "numpy.linalg.eig" does give an expected rank which is 39. This rank came from $min(S_B, S_W)$ where the rank of S_B is 39 (number of classes minus 1) and S_W (number of samples minus number of classes) has rank 80. Therefore, the number of non-zero eigenvalues is **39**.

T23.
<u>ANSWER</u>

This is from LDA projection (Reconstruction for each person).



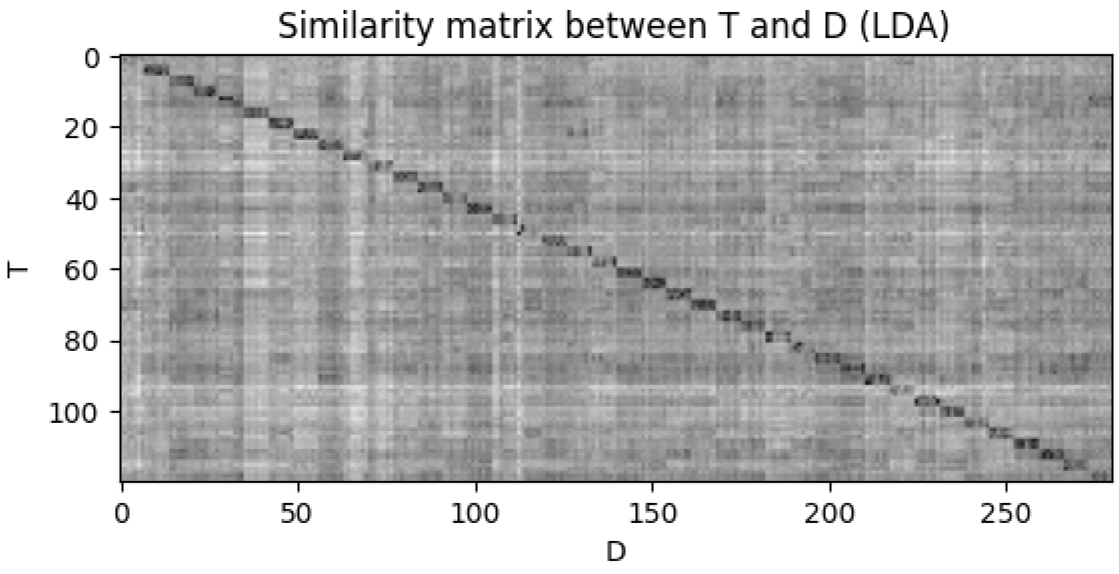
This is from PCA projection (Best eigenvectors).


Eigenface 1 with Eigenvalue 1423.93
Eigenface 2 with Eigenvalue 875.38
Eigenface 3 with Eigenvalue 558.62
Eigenface 4 with Eigenvalue 407.35
Eigenface 5 with Eigenvalue 371.02
Eigenface 6 with Eigenvalue 263.28
Eigenface 7 with Eigenvalue 203.40
Eigenface 8 with Eigenvalue 190.97
Eigenface 9 with Eigenvalue 166.55
Eigenface 10 with Eigenvalue 143.57

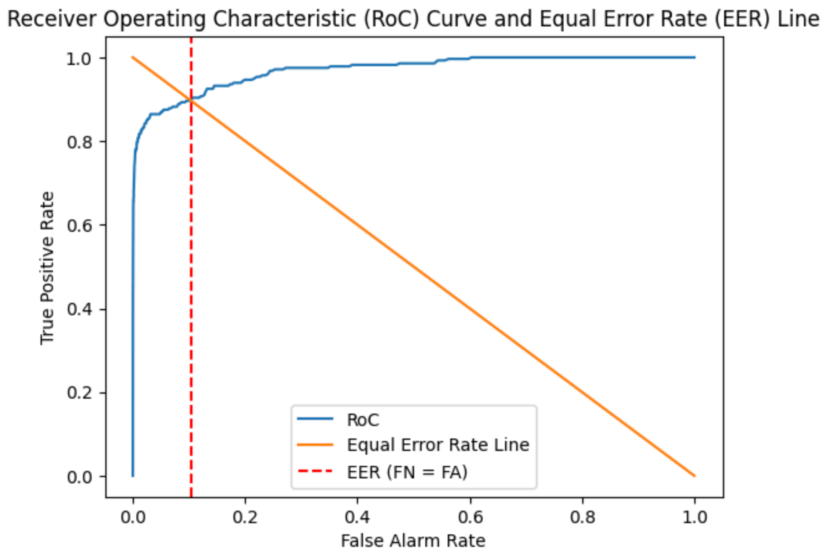
As we can quickly see that the PCA captures the features that make the variance high without considering if it is the same person. Meanwhile, the LDA considers the features that make each person different in which the edges and finer details.

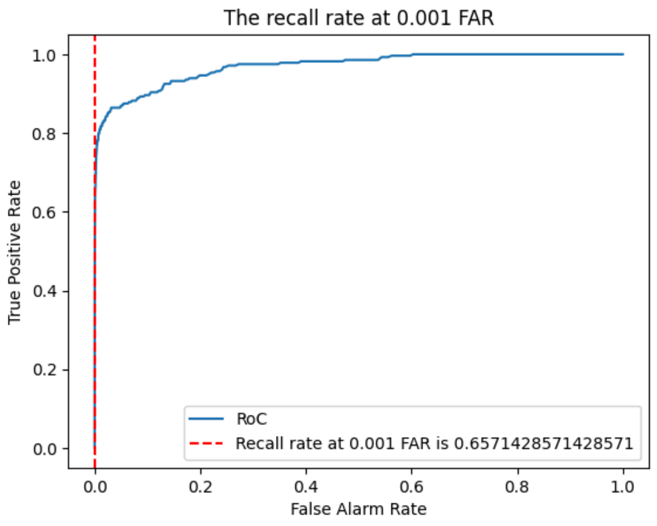
T24.

ANSWER



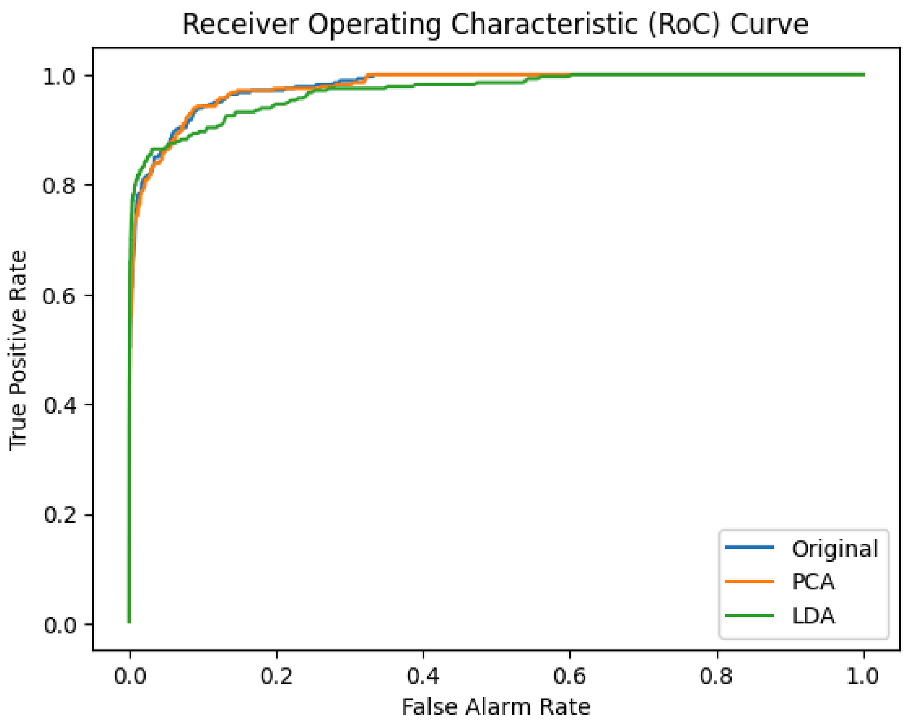
We can see that the LDA outperforms both PCA and no projection methods by clearly discriminating between each person and making the face verification process better. As we can see from the outside the diagonal line is mostly gray and white.





The EER for this method is **0.8964285714285715** and the recall is **0.6571428571428571.**

T25.
<u>ANSWER</u>

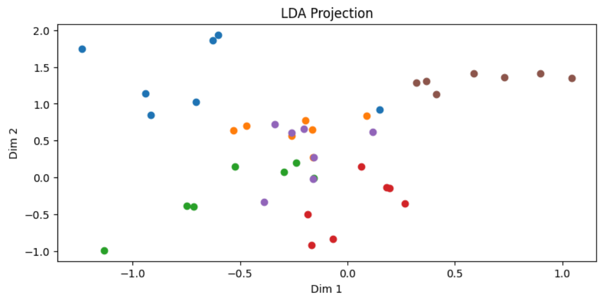


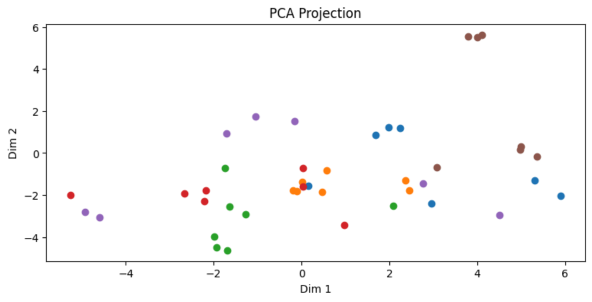
From the figure, we can see the PCA+LDA(fisherface) outperforms both methods in distinguishing a person's face in the area where the false alarm rate is about 0.0 - 0.1 which is the best method for face verification in this exercise as it is shown in green line at that area. However, the PCA and no projection depict very similar performance. To choose the model between these two methods, it might need to consider the use cases such as high true positive rate or lower false alarm rate.

OT5.

<u>ANSWER</u>

As we can see, the LDA can clearly separate the data points in the same class and also minimize the distance within its own class. On the other hand, the PCA method fails to minimize the distance within its own class but might still be able to distinguish the data between classes but didn’t perform as well as the LDA. The results came out as expected where each color denotes a different person.


LDA Projection
Dim 1
Dim 2


PCA Projection
Dim 1
Dim 2