ECE 763 Luke Martin Project 1 Report and Results 2/25/2019

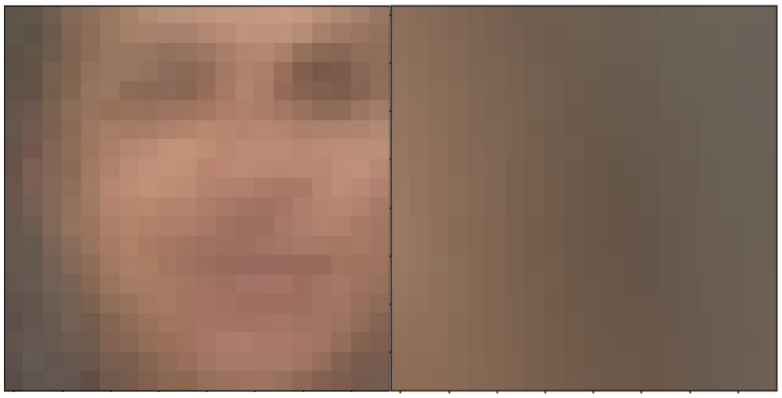
Task 1: Dataset Preparation

I downloaded my images from the Faces in the Wild dataset and used my own annotations for cropping the faces. I cropped the background images from these faces as well. Using this dataset, I resized the images to 20x20 RGB and saved 1000 face and nonface images for training and 100 face and nonface images for testing. This image dataset will be included in the file in which this report was submitted.

In order to greatly reduce my runtime and complexity, my data was prepared with PCA to greatly reduce the dimensionality of my data and allow my numbers to be much more easily and quickly calculated. This was especially useful when working with images because now I could view each image as a single datapoint. The PCA was formed using a built-in function within scipy and my training and testing images were fit to this PCA using some constraining dimension which I chose to be 20. If I increased this constraint, I could maintain more of the detail when reconstructing the images however, this value allowed enough detail to be seen regardless. This value ran much faster than most others with decent accuracy on my dataset.

Task 2: Average/Covariance Image Visualization

I built my own functions in order to visualize the mean and covariance of my training image datasets and had the following results:


Average of Face and background, respectively

Covariance of Face and background, respectively

Based on these results, I concluded that my training data has a bias that most of the images are primarily smiling and mostly looking to the right of the image (looking to their left). The background data ended up having a similar color to the face data which made sense since the patches were taken from those images themselves.

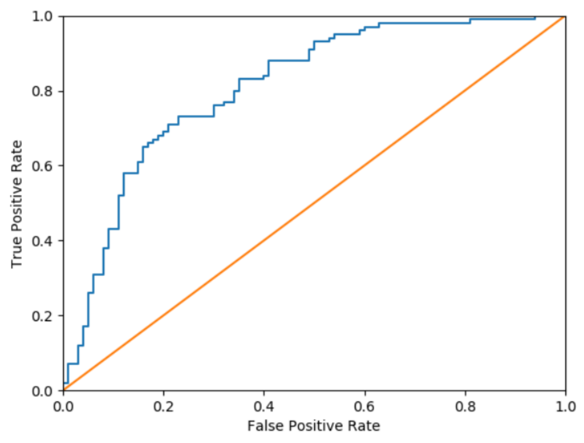
Task 3: Single Gaussian Model

As we discussed in lecture, a single Gaussian is not very robust in terms of classification. Lots of noise can cause issues as well as issues with faces and directions. From my average dataset, it can be seen that my faces primarily look to their left or to the right of the image so this may cause issues if I test a face looking the opposite direction. The equation I used was: $f(x) = \frac{1}{\sqrt{2*pi*\sqrt{|E|}}} * e^{(-\frac{(X-\mu)^T*(E)^{-1}*(X-\mu)}{2})}$

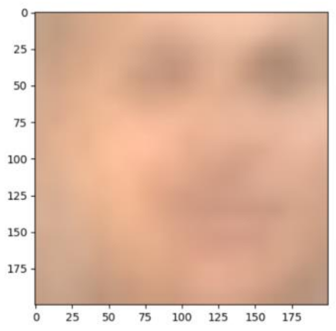
This equation is the pdf of the gaussian distribution and because of my preprocessing with PCA, the system did not overflow or underflow due to the low dimensionality of my inputs. In order to remove the chances of division by zero, a very small amount of noise was added to the denominator of this equation, it had an unnoticeable effect on the overall accuracy of the system.

Single Gaussian Results:

Using my system, the rate of false positives was 68% using my dataset. My false negative rate was 25%. This leads to my misclassification rate being 84% overall. The preprocessing could certainly be improved to make the system far more robust as is done in the next tasks. It was also more accurate since we used a threshold of 0.5 which would make it easier to make positive identifications. The AUC for my curve ended up being about 84.96% so this accuracy is most likely due to the dataset and thresholding values used for the processing.



ROC Curve:



Recovered Mean From PCA Data

This average image shows up after running this section of code. It is computed from the inverse of the PCA transform declared earlier for making the data more manageable. I added this in to show that the information can still be recovered even if some of the detail may be lost.

Task 4: Mixed Gaussian Model

To begin with mixed Gaussian modeling, I started by splitting my data up into 2 sections (in this case I was going for two Gaussian models). I computed the initial means and covariances of these two sections to get values for my initial expectation step of EM. I initialized the weights to be even for however many gaussians that are needed.

For the expectation step, I used the training data to calculate the probability that the input image belonged to each gaussian. Some of my values were very low and the system would error saying it was trying to divide by zero. To solve this, I added some very small noise to my Gaussian pdf function in order to make sure the division by zero was not possible, as previously described. This step was done by multiplying the lambda weight by the result of the gaussian function. This result was normalized and saved for the maximization step.

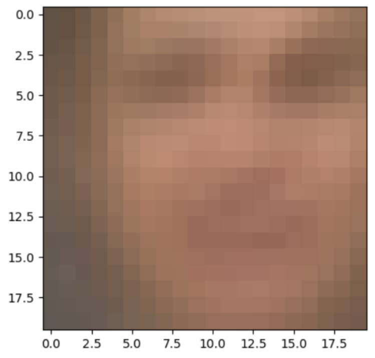
For the maximization step, the first step was to estimate the weight of each gaussian and use this to recalculate my weights, means and covariances. My new weights for each gaussian were stored within gamma, a vector with length equal to the number of gaussians. I used the following equations to calculate gamma, the means and the covariance matrices:

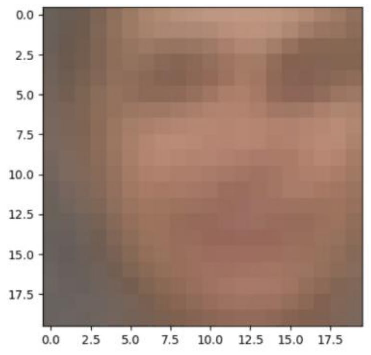
$$
\begin{aligned}
\lambda_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K}\sum_{i=1}^{I} r_{ij}} \\
\boldsymbol{\mu}_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{I} r_{ik}} \\
\boldsymbol{\Sigma}_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})^T}{\sum_{i=1}^{I} r_{ik}}.
\end{aligned}
$$

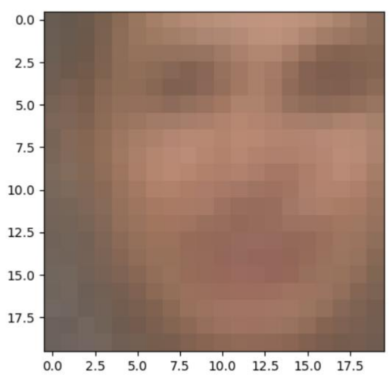
$r_{ik}$was the probability that each image belonged to which gaussian. My new gamma weights were calculated by summing the probabilities of every image belonging to any single gaussian and dividing that by the sum of the probability that every image belonged to these gaussians.

In order to calculate the new mean, I multiplied each image by the probability of that image belonging to each gaussian and divided this value by the sum of the probabilities of it belonging to that gaussian. Lastly, for the covariance matrix I computed the difference of each image with a new mean belonging to each gaussian and then multiplied this by its transpose resulting in a large matrix. I performed the same tasks as before and computed the total probability that this belonged to each gaussian and divided by the probability that it belonged to this gaussian as well.

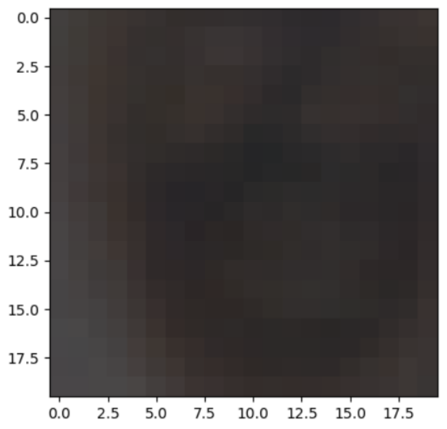
Mixed Gaussian Results:

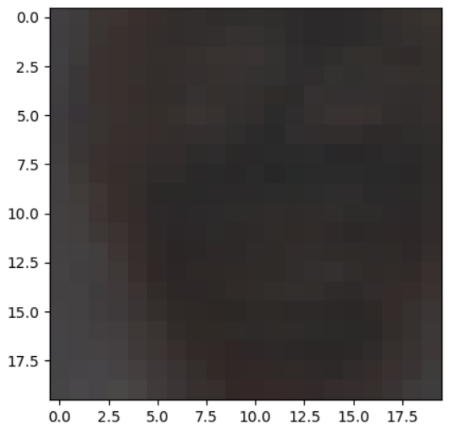


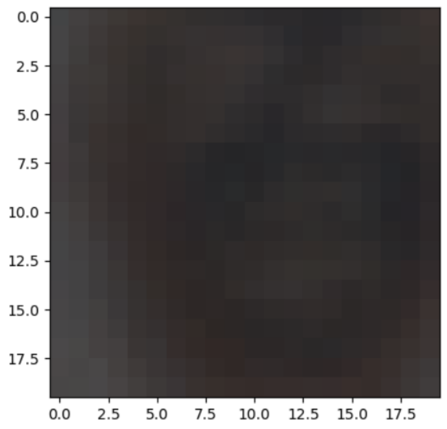




Faces from Different Means

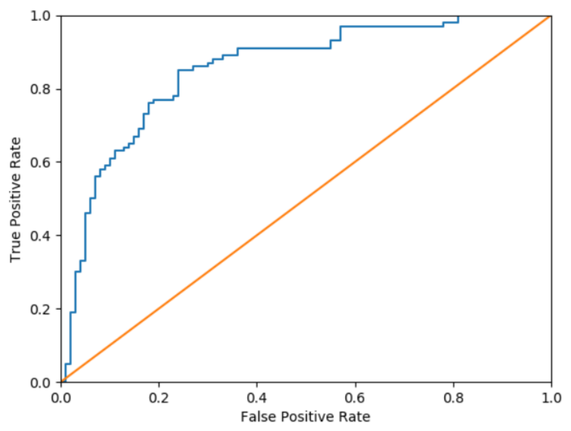






Faces from Different Covariances

The results of these images made sense since the average face is changing based on the weights as well as the covariances. The covariance images ended up a lot darker than the previous single Gaussian due to some normalization errors within my code.



ROC Curve for Mixture of Gaussian

The overall accuracy for the mixture of Gaussian ended up being 85.32%. This number can change based on the random seed used to find the starting datapoints. I chose to set the seed to optimize the model and see the same results once those values were found.

The MCR (Misclassification Rate) was 12.5% based on my data. This is a result of the False Negative Rate being 0 and a False Positive Rate of 25%. There may have ended up being some issues with some data overlap in the end that led to an unrealistically low false negative rate however these issues are being overlooked for now. There also may have been an issue with some of the equations I used in calculating the FNR and FPR

Task 5: t-Distribution

The first step of computing the t-distribution was to evaluate the initial values for the mean, covariance and nu value required for the t-distribution. The initial value for nu (the degrees of freedom) was initialized to some arbitrarily high value. The EM algorithm as discussed in lecture was used following the initialization step. The other values needed were the expected values of the hidden variable and the expected log of the hidden variable. The hidden variable took the shape of the Gamma function so the expected value was equal to: $\mathrm{E}[h_i] = \frac{(\nu + D)}{\nu + (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})}$. The log was equal to: $\mathrm{E}[\log[h_i]] = \Psi\left[\frac{\nu + D}{2}\right] - \log\left[\frac{\nu + (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})}{2}\right]$. In order to put this into code, I used the built-in function called psi which is the log derivative of the Gamma function. This was necessary due to the size and memory constraint of the machine I was using to run. Another equation used for this was the gammaln function which is the log of the Gamma function itself.

Another function I used went throught the full range of nu and calculate its value at each section. This cost function is where I computed the minimum value for nu to use for my pdf functions and to finish calculating my expected values. This was a part of the Maximization step for the t-distribution and the function follows this equation:

$$E\left[\log\left[Pr(h_i)\right]\right] = \frac{\nu}{2}\log\left[\frac{\nu}{2}\right] - \log\Gamma\left[\frac{\nu}{2}\right] + \left(\frac{\nu}{2}-1\right)\mathrm{E}[\log h_i] - \frac{\nu}{2}\mathrm{E}[h_i].$$

. Since I already had the values the expected value and expected log value, the numbers could be directly plugged into the equation and I could update my nu. This EM algorithm was repeated for 10 iterations or until the logliklihood reached a certain threshold..

This was the first step used to calculate the required pdf of the t-distribution. These values were computed for both the face dataset and the nonface dataset.

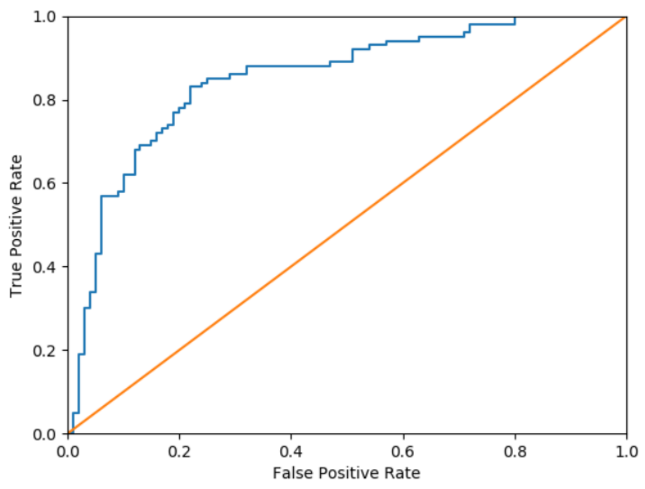
From lecture, we know that the t-distribution takes the form of:

$$\begin{aligned} Pr(\mathbf{x}) &= \mathrm{Stud}_{\mathbf{x}}\left[\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu\right] \\ &= \frac{\Gamma\left[\frac{\nu+D}{2}\right]}{(\nu\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}\Gamma\left[\frac{\nu}{2}\right]}\left(1+\frac{(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{\nu}\right)^{-\frac{\nu+D}{2}}, \end{aligned}$$

. However, this could not be directly used in the code due to size constraints and overflow/underflow issues I was experiencing. To begin, I had to take the log of the Gamma function again and this time raise an exponential to it. In algebra, this would mean the exponential and log funtion would cancel but in this case it allows my numbers to stay within a reasonable range that will not overflow the system. The other values could be directly entered and run. This function returned the pdf of the t-distribution and I moved on to the next step of testing images on it.

Results: t-distribution

The average image and covariance images each looked the same as in the previous results. The resulting



ROC Curve for t-Distribution, AUC = 84.96%

The False Positive Rate (FPR) was reported as 0.5 (50%), which was not so great and the False Negative Rate was also 0.5 or 50%. This led to the overall Misclassification rate of 50%.

The overall accuracy of the system for the training was 80.7%. These results are most likely effected by the dataset that I chose due to the positioning of the faces. The performance was not as quick as the Gaussian but still proved to work almost as effectively.

Task 6: Factor Analyzer

The first step in factor analysis was to initialize my mean. This average would stay constant throughout the process while my covariances would be changing and updating due to the EM algorithm for factor analysis. The next values I had to process were the phi values (covariance model) used in factor analysis and the number of factors, K. In my code, it will run for 25 iterations, just some arbitrary number of runs through the EM algorithm. I tried to use the non-PCA processed images to try and have a higher level of detail however this greatly slowed down the runtime by a drastic amount.

The Expectation step started with me calculating the new phi values and the expected value of the hidden variable. The equation I used was that found in the lecture notes:

$$\begin{aligned} \mathrm{E}[\mathbf{h}_i] &= (\mathbf{\Phi}^T\mathbf{\Sigma}^{-1}\mathbf{\Phi}+\mathbf{I})^{-1}\mathbf{\Phi}^T\mathbf{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}) \\ \mathrm{E}[\mathbf{h}_i\mathbf{h}_i^T] &= E\left[(\mathbf{h}_i-\mathrm{E}[\mathbf{h}_i])(\mathbf{h}_i-\mathrm{E}[\mathbf{h}_i])^T\right]+\mathrm{E}[\mathbf{h}_i]\mathrm{E}[\mathbf{h}_i]^T \\ &= (\mathbf{\Phi}^T\mathbf{\Sigma}^{-1}\mathbf{\Phi}+\mathbf{I})^{-1}+\mathrm{E}[\mathbf{h}_i]\mathrm{E}[\mathbf{h}_i]^T. \end{aligned}$$

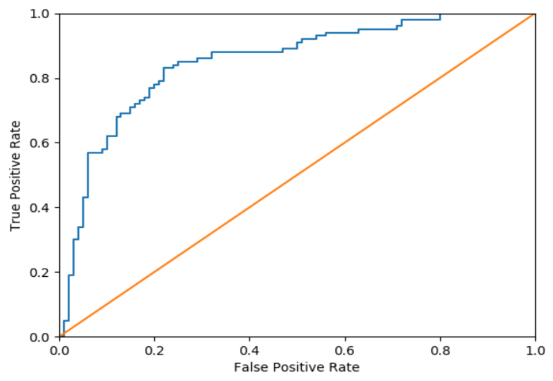
.

Once the expectation was taken for all of the hidden variables, I could start the maximization step. The goal of the maximization step was to get new phi values and new variances. The equations coded were also those from the lecture:

$$\begin{aligned} \hat{\boldsymbol{\mu}} &= \frac{\sum_{i=1}^{I}\mathbf{x}_i}{I} \\ \hat{\mathbf{\Phi}} &= \left(\sum_{i=1}^{I}(\mathbf{x}_i-\hat{\boldsymbol{\mu}})\mathrm{E}[\mathbf{h}_i]^T\right)\left(\sum_{i=1}^{I}\mathrm{E}[\mathbf{h}_i\mathbf{h}_i^T]\right)^{-1} \\ \hat{\mathbf{\Sigma}} &= \frac{1}{I}\sum_{i=1}^{I}\mathrm{diag}\left[(\mathbf{x}_i-\hat{\boldsymbol{\mu}})(\mathbf{x}_i-\hat{\boldsymbol{\mu}})^T-\hat{\mathbf{\Phi}}\mathrm{E}[\mathbf{h}_i](\mathbf{x}^T-\boldsymbol{\mu})\right] \end{aligned}$$

. I will note that I did not recalculate the mu value (average) because this value stayed constant through all of the factor analysis. Once all the values were updated and I reached the end, the function returned the average, variances and phi values.

Results: Factor Analyzer

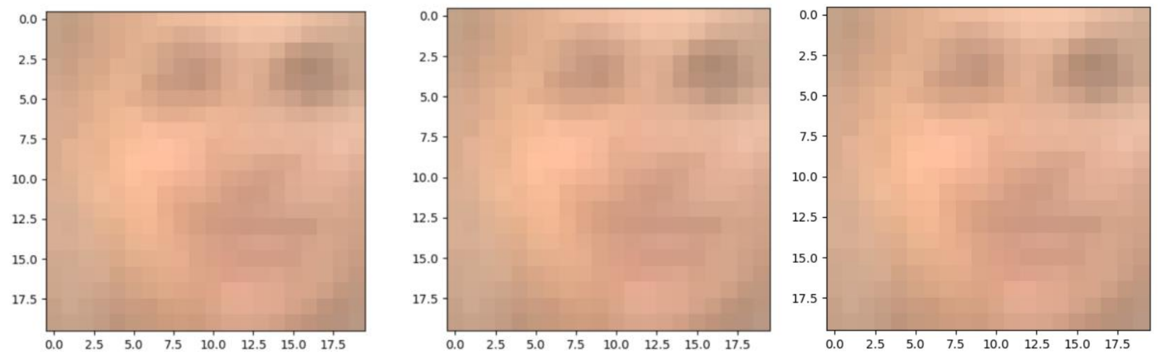


ROC Curve for the Factor Analyzer, AUC = 85.03%

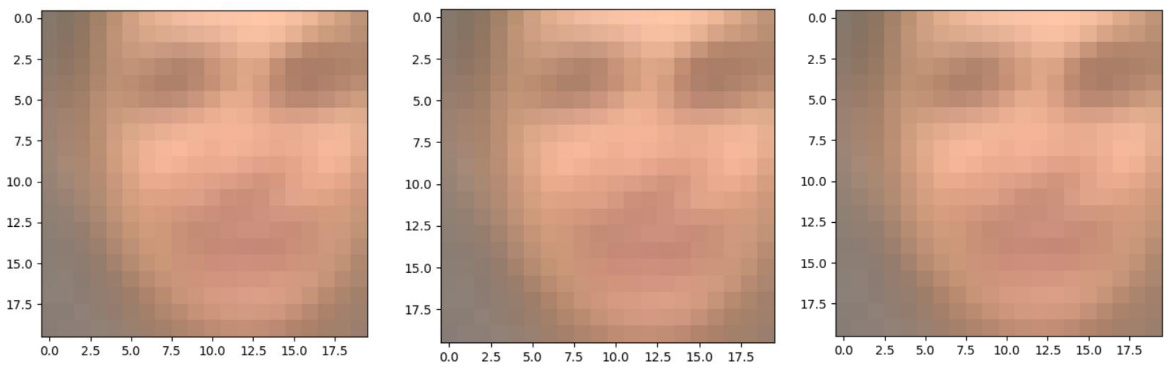
Since the initial values were randomized each time, some of the values were better if their initial values were more accurate. The accuracy for this ROC curve was 85.03%

The False Positive Rate was 25.66% and the False Negative Rate was 59.88%. This resulted in an overall Misclassification rate of 42.13%.

Sample Image from the result of the Factor Analyzer:



Although difficult to see, the images have slight variations.



Higher resolution images to see the subtle changes.

Task 7: Mixture of t-distribution

The mixture of t algorithm I used was a combination of the mixture of gaussian code only now I am using the t-Distribution. The initialization followed the same as the Mixture of Gaussian as I needed to set up the necessary variables and the number of distributions I was using. The starting datapoints were initialized using a fixed seed to maintain similar results across tests. I also had to initialize the degrees of freedom as I did for a regular t-Distribution. After this, I followed the same EM algorithm as the t-Distribution and adjusted it to loop through for the number of distributions desired.

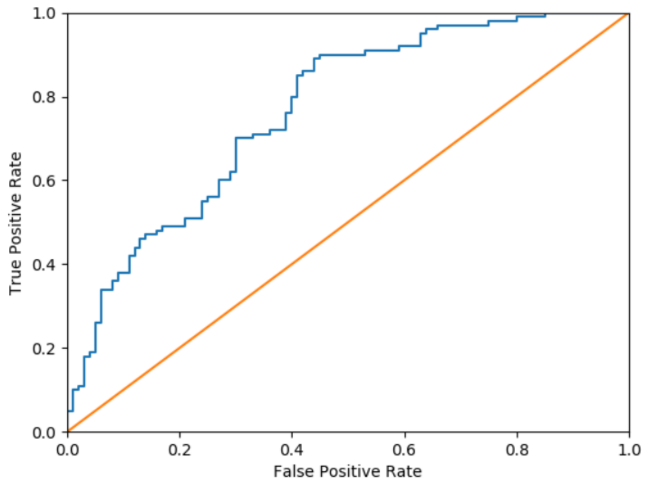
In the Expectation step, I calculated the Expected value for the hidden variables associated with multiple distributions. These probabilities were normalized after. I used a different pdf function to calculate the probabilities due to some overflow/underflow errors and division by zero errors that the system was causing with my original t-Distribution pdf. The new pdf function used the log of gammas primarily to make the data more manageable. A new step in the mixture of t-Distribution required me to calculate the distance for each datapoint to each of the t-Distributions. I used the built-in function for calculating the Mahalanobis distance. This function calculates and returns the distance from each datapoint to each distribution rather than to another datapoint. This function worked perfectly and allowed me to update my weights based on these values to see how many of my images associated with each set.

The Maximization step was primarily the same as the t-Distribution and Mixture of Gaussian. It updated new values for the mean and covariance after finding the new Expected value and Expected log likelihood of the hidden variables. The value for nu was updated the same as in the t-Distribution which minimized the cost function for the t-Distribution. The only change was that it was looped through for each t-distribution. The Mahalanobis distance was also calculated for the new weights following the other values.

I looped through this 5 times due to the much slower speed of the system. This model was the slowest to run. The following results are using only 5 iterations.

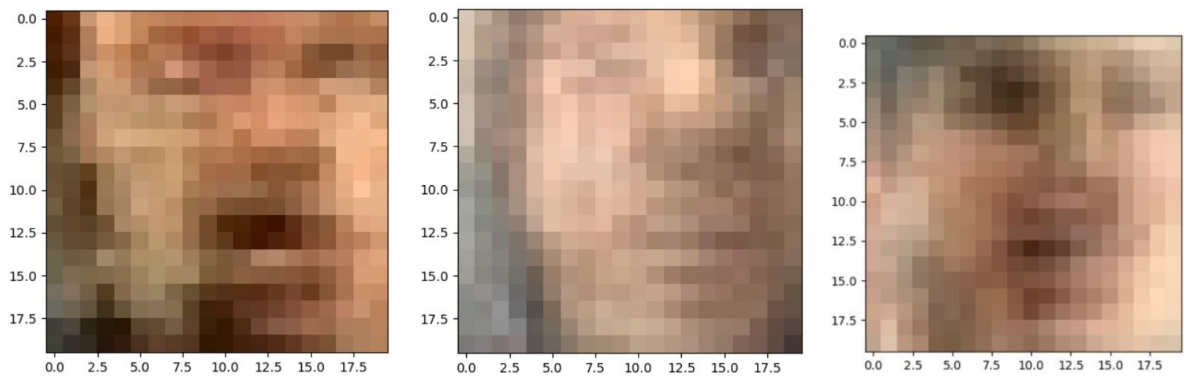
Results for Mixture of t-distributions:

It is important to note that I fixed the seed for the random number generator for the mixture of t-distribution. All of the randomized variables, i.e. starting datapoints, were fixed. I did this because there were a few cases where the algorithm would fail after only a couple of iterations. These results were produced using this fixed rng.



ROC Curve, Mixture of t-distributions. Accuracy: 80.63%

The overall False Positive Rate for this was 29% and the False Negative Rate was 51%. This led to an overall misclassification rate of 80%. Overall, the model could have been improved, however, this was the best results I could achieve with my current experience with Mixture of t-distributions.



Average Faces from Mixture of t-Distribution results