# CS 679: Pattern Recognition

University of Nevada, Reno - Spring 2024

Assignment 1

Jaleesa Houle

Due: March 11th, 2024

*Statement:* "I declare that all material in this assignment is my own work except where there is clear acknowledgment or reference to the work of others. I understand that both my report and code may be subjected to plagiarism detection software, and fully accept all consequences if found responsible for plagiarism, as explained in the syllabus, and described in UNR's Academic Standards Policy: UAM 6,502."

# 1. Theory

## Experiments 1 & 2

### General background and theory

Maximum Likelihood (ML) theory is an approach used to estimate parameters from data. To implement ML, we assume that there are some fixed, unknown parameters $\theta$ which describe given data $D$, where $D = \{x_1, x_2, ...x_n\}$, $x_i \in R^N$. The goal of ML is to find a point estimate $\hat{\theta}$ which best descibes the sample $D$. This is achieved by maximizing the likelihood of $\theta$ given the observed samples such that

$$\hat{\theta} = \arg\max_{\theta} p(D|\theta). \tag{1}$$

By assuming the samples in $D$ are drawn independently according to some distribution $p(\mathbf{x}|\theta)$, Equation 1 becomes

$$\begin{aligned} p(D|\theta) &= p(\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_n}|\theta) \\ &= \prod_{k=1}^{N} p(\mathbf{x_k}|\theta). \end{aligned} \tag{2}$$

This function can be further manipulated by taking the natural log, which turns the product into a sum and simplifies the calculations such that

$$\hat{\theta} = \arg\max_{\theta} \ln\, p(D|\theta); \tag{3}$$

$$\ln\, p(D|\theta) = \sum_{k=1}^{N} \ln\, p(\mathbf{x_k}|\theta). \tag{4}$$

In order to solve for $\hat{\theta}$, we set the gradient of Equation 3 equal to zero, i.e,

$$\begin{aligned} 0 &= \nabla_\theta \ln\, p(D|\theta) \\ &= \sum_{k=1}^{N} \nabla_\theta \ln\, p(\mathbf{x_k}|\theta). \end{aligned} \tag{5}$$

To estimate ML using multivariate Gaussian data with unknown parameters $\theta = (\mu, \Sigma)$ we assume that $p(x|\theta) = p(x|(\mu, \Sigma)) \sim N(\mu, \Sigma)$, $x \in R^N$. The probability density function (pdf) for a multivariate Gaussian distribution is defined as

$$p(\mathbf{x}|(\mu, \Sigma)) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)] \tag{6}$$

where $n$ is the number of features, $\mu$ is an $n$ x 1 vector that represents the mean of each feature $x_i$, $\Sigma$ is the covariance matrix (size $n$ x $n$) that represents the data $D$, and $\mathbf{x}$ is an $n$ x $m$ vector of data from $D$, where $m$ is the number of samples. By

substituting Equation 6 into Equation 5 and solving for the unknown parameters $\mu$ and $\Sigma$, we find that the ML solution for multivariate Gaussian data with unknown $\theta = (\mu, \Sigma)$ is

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^{N} \mathbf{x_k} \tag{7}$$

$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^{N} (\mathbf{x_k} - \hat{\mu})^T (\mathbf{x_k} - \hat{\mu}). \tag{8}$$

This result shows that the optimal $\hat{\mu}$ is the sample mean and the optimal $\hat{\Sigma}$ is the sample covariance for data in which the underlying distribution follows $p(\mathbf{x}|\theta) \sim N(\mu, \Sigma)$. Once the parameters of $D$ have been estimated, the data can then be classified using a Bayes classification approach, as described in Assignment 1.

**Parameter estimation based on randomly chosen training samples**

In order to test the accuracy of Bayesian classification using ML, we will use ML to estimate the parameters $\mu$ and $\Sigma$ with different training subsets of the generated sample data. For part (a), all sample data will be used to train/compute $\hat{\mu}$ and $\hat{\Sigma}$. For part (b), various sub-samples will be randomly chosen (using a uniform distribution) so that the total percent of data used to estimate $\hat{\mu}$ and $\hat{\Sigma}$ is (i) 0.01%, (ii) 0.1%, (iii) 1% and (vi) 10% of each sample.

**Determining misclassification rates**

This assignment asks us to report the misclassification rate of each class and the total misclassification rate, as was also the case for Assignment 1. For Experiments 1 & 2, we will determine misclassification rates in the same way as was previously described (i.e., by comparing the true class to the expected class for each sample drawn). The misclassification rate for each class will be determined as $n_i/N_i$, where $n_i$ is the number of samples misclassified and $N_i$ is the total number of samples produced from class $\omega_i$. The total misclassification rate will be calculated as $(n_1 + n_2)/(N_1 + N_2)$, i.e., the total number of samples misclassified divided by the total number of samples.

**Making assumptions to simplify parameter estimation**

For this assignment, we are asked to explore the effect of making assumptions about our ML estimated parameters. To do so, I will compute the ML estimates of each sample in 2 different ways: (1) by making no assumptions about the covariance matrix and (2) by assuming that the features of $D$ are uncorrelated, and by rounding the estimates of $\hat{\mu}$ and $\hat{\Sigma}$. In (1), data will almost always be characterized as case III since values will not be rounded therefore it is unlikely for the covariance matrices for $\omega_1$ and $\omega_2$ to be exactly the same. In (2), the assumption of no correlation between features combined with rounding the estimated covariance matrices should result in the data from Experiment 1 to be classified as case I, whereas the data in Experiment 2 will still be case III (since these were the cases found for data A

and B in Assignment 1). The results of these two assumptions will be compared by calculating a decision boundary for each set of parameter estimates, classifying the data (using the Bayesian Decision approach from Assignment 1), and comparing the corresponding misclassification rates.

# Experiment 3

### General background and theory

The theory for Experiment 3 is described in Yang and Waibel (1996). The authors found, after taking the image pixels from over 40 different faces and converting the image space from RGB to chromatic, that the resulting distribution of r and g pixel values appeared to be normally distributed. As a result, we can use the multivariate Gaussian ML approach described above to estimate the parameters for some skin pixel data in $R^2$. Then, we can use those estimated parameters to determine the likelihood that a pixel of an image is skin or not skin.

### Converting color space of an image

In general, color images are typically in RGB format. The pixel values of a digital image form a matrix of the shape [height $x$ width $x$ depth], where the depth corresponds to the individual channels (i.e. R, G, B). In this assignment, we are asked to convert the given images into two different color spaces.

For part (a), the data must be converted from RGB ($R^3$) to chromatic ($R^2$). This conversion is done using the relationships

$$\begin{aligned} r &= R/(R+G+B) \\ g &= G/(R+G+B). \end{aligned} \tag{9}$$

By converting the color space from $R^3$ to $R^2$, we are able to estimate the parameters for a 2D Gaussian distribution.

Similarly, for part (b), the data is converted from RGB to YCbCr using the relationships

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= -0.169R - 0.332G + 0.500B \\ C_r &= 0.500R - 0.419G - 0.081B. \end{aligned} \tag{10}$$

In this color space, the Y channel captures luminescence while the $C_b$ and $C_r$ channels capture chrominance information, so the two features used for this color space will be $C_b$ and $C_r$, allowing us to again model the skin color distribution as a 2D Gaussian.

### Determining location of skin pixels

The location of skin pixels in each training image was found using the corresponding reference image, which is all black in areas that are not classified as a person's face, and either white or blue in locations that were classified as a person's face. Using the reference image, we can extract the locations of known skin pixels. The

corresponding r and g values in those locations are then used as model features.

### Classifying the data as skin or not skin

Since the distribution of skin color is approximately Gaussian, the parameters of our skin color model can be estimated by the ML approach described above for Experiments 1 and 2. After using Equations 7 and 8 to determine $\hat{\mu}$ and $\hat{\Sigma}$, we can then build a Gaussian discriminant to classify all image pixels as skin or not skin, where

$$g(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\hat{\Sigma}|^{1/2}}\exp[-\frac{1}{2}(\mathbf{x}-\hat{\mu})^T\hat{\Sigma}^{-1}(\mathbf{x}-\hat{\mu})]. \tag{11}$$

We can classify a pixel as being skin if $g(\mathbf{x}) > t$, where $t$ is a threshold value. It can be seen from Equation 11 that if the expression $(\mathbf{x}-\hat{\mu})^T\hat{\Sigma}^{-1}(\mathbf{x}-\hat{\mu})$ goes to zero (i.e., if $\hat{\mu}$ is zero), $g(x)$ approaches $1/(2\pi)^{n/2}|\hat{\Sigma}|^{1/2}$. If that expression goes to $\infty$, then $g(x)$ approaches zero. Since the value of any given image pixel ranges from $[0, 255]$, we can see that $g(x)$ is positively bounded between $[0, c]$ where $c = 1/(2\pi)^{n/2}|\hat{\Sigma}|^{1/2}$. Thus, we can determine the appropriate threshold to classify whether a pixel is skin or not skin by assessing error classification rates at different thresholds ($t$) between [0,c].

### Determining error rates and the ideal threshold

There are generally four different classification results that can occur for any given data: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). TP occurs when a sample belongs to the skin class and is classified as skin, FP occurs when a sample is not skin but is classified as skin, TN occurs if a sample is not skin and is classified as not skin, and FN occurs if a sample is skin but is classified as not skin. The total amount of samples that are positive (i.e. determined to be skin) are $P = TP + FN$. The total amount of samples that are negative (i.e., determined as not skin) are $N = FP + TN$. We can assess error rates for each image by assessing the false accept rate (FAR) and the false reject rate (FRR). These values are found using the relationships

$$\begin{aligned} FAR &= FP/(FP + TN) = FP/N \\ FRR &= FN/(TP + FN) = FN/P. \end{aligned} \tag{12}$$

For this assignment, we are asked to determine the FAR and FRR for 20 uniformly distributed thresholds between [0,c]. The optimum threshold value $t$ is the one which minimizes the FAR and FRR error rates. This is also known as the equal error rate (EER), and is the point where FAR=FRR. To find the EER, I will use the SciPy Python package to interpolate the calculated FAR and FRR curves. Then, I will use a SciPy function solver which minimizes the difference between the FAR and FRR values at any $t$ so that the resulting solution is the threshold value where FAR $\approx$ FRR.

The ROC curves are found by plotting the FAR again the FRR. A classifier is defined as better if it has less area under the curve (AUC). We will visually compare

the classifier performance on different images at various thresholds by comparing ROCs plots.

## 2. Results and Discussion

### Experiment 1

#### a. Estimating parameters of sample A using ML and comparing results with Assignment 1

The given parameters for sample set A are shown in Table 1, alongside the ML estimates. For Assignment 1, we generated 60,000 samples from $[\mu_1, \Sigma_1]$ and 140,000 samples from $[\mu_2, \Sigma_2]$. The ML estimate of these parameters in Table 1 was calculated using all 200,000 samples. It can be seen that the ML estimated parameters are very close to the true values, though it is expected that the estimates would not be precise since the data was randomly generated. The misclassification rates in Table 1 correspond to the way the data was classified using the true and ML estimated parameters to create a decision boundary. While there is some small variance in misclassification rates of individual classes $\omega_1$ and $\omega_2$, the overall misclassification rate was approximately the same for both cases.

| **Parameters** | **True Values** | **ML Estimates** |
|---|---|---|
| $\mu_1^T$ | [1, 1] | [0.9947, 0.9993] |
| $\Sigma_1$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 0.9928 & -0.0091 \\ -0.0091 & 1.0003 \end{bmatrix}$ |
| $\mu_2^T$ | [4, 4] | [3.9990, 3.9952] |
| $\Sigma_2$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 0.9965 & -0.0015 \\ -0.0015 & 0.9972 \end{bmatrix}$ |
| **Class** | **Misclassification Rates** | |
| $\omega_1$ | 2.67% | 2.71% |
| $\omega_2$ | 1.01% | 0.99% |
| Overall | 1.51% | 1.51% |

Table 1: Comparison of the ML parameter estimates and true parameters for dataset A. The full sample (n=200,000) was used for estimating the parameters.

Figure 1 demonstrates that the decision boundaries are nearly identical if using the true parameters versus the ML estimated parameters. It can be seen in Table 1 that the ML covariance matrix estimates were slightly different between $\Sigma_1$ and $\Sigma_2$, so a case III (non-linear) decision boundary was calculated for the ML estimated parameters.

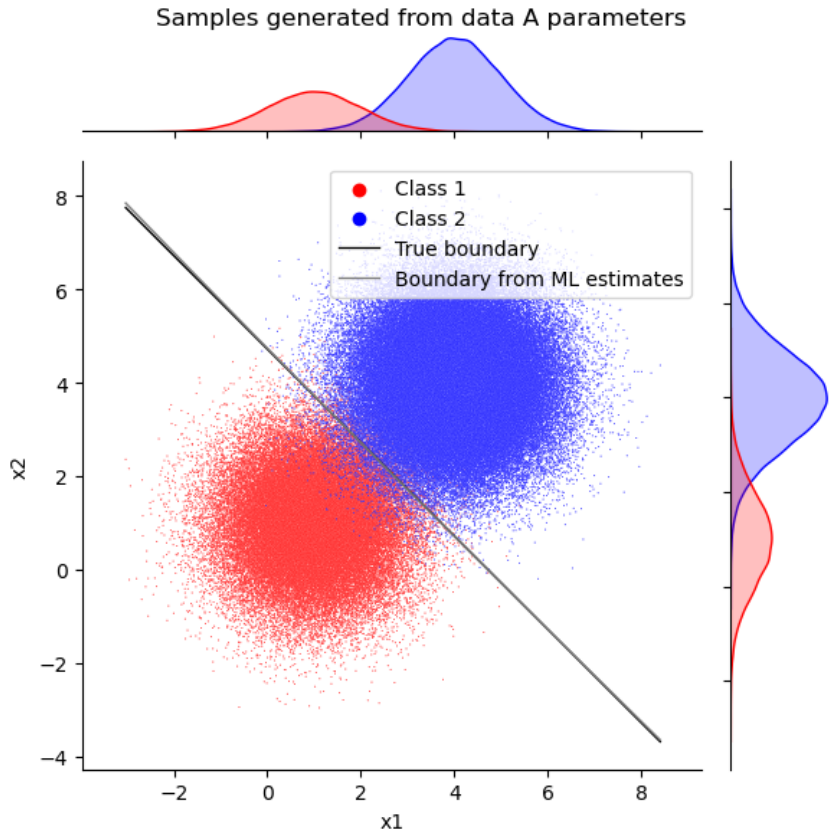



Figure 1: Unclassified sample data generated using the given parameters for data A in Assignment 1. The true boundary estimated from the given parameters is shown in black, while the boundary calculated from the ML estimated parameters is in grey.

### b. Testing classification accuracy with subsets of sample

Table 2 demonstrates that the amount of training data used to estimate the parameters of sample set A can impact the effectiveness of the corresponding decision boundary. In the case where 0.01% of the samples were used (i.e., 6 data points from sample 1 and 14 sample points from sample 2), the overall rate of misclassification is 9.30%, with 28.83% of samples from class $\omega_1$ being misclassified as $\omega_2$. The overall rate of misclassification drops quickly, however, as the amount of data used for ML estimation increases. In fact, using just 1% of the data (i.e., 600 samples from $\omega_1$ and 1400 samples from $\omega_2$) was enough to reduce the misclassification rate to 1.51%, which is the same overall rate seen for the true parameters in Table 1. Though there is still some variability between the true parameters and the estimated parameters using 1% of the data for training, it is clear that the ML approach can be an effective way to estimate unknown parameters of a given dataset.

| **Parameters** | **ML Parameter Estimates** | | | |
|---|---|---|---|---|
| % of samples used | 0.01% | 0.1% | 1% | 10% |
| $\mu_1^T$ | [0.8123, 0.6543] | [0.8226, 0.9173] | [0.9503, 0.9244] | [0.9870, 0.9800] |
| $\Sigma_1$ | $\begin{bmatrix} 1.2658 & 0.1310 \\ 0.1310 & 0.0641 \end{bmatrix}$ | $\begin{bmatrix} 1.2309 & 0.2217 \\ 0.2217 & 0.7379 \end{bmatrix}$ | $\begin{bmatrix} 1.0262 & 0.0496 \\ 0.0496 & 0.9915 \end{bmatrix}$ | $\begin{bmatrix} 0.9930 & -0.0152 \\ -0.0152 & 1.0134 \end{bmatrix}$ |
| $\mu_2^T$ | [3.8473, 4.3005] | [4.0996, 4.0555] | [4.0216, 4.0031] | [4.0092, 3.9863] |
| $\Sigma_2$ | $\begin{bmatrix} 0.6935 & 0.2232 \\ 0.2232 & 0.5442 \end{bmatrix}$ | $\begin{bmatrix} 1.0092 & 0.0144 \\ 0.0144 & 0.8406 \end{bmatrix}$ | $\begin{bmatrix} 1.0443 & 0.0219 \\ 0.0219 & 0.9954 \end{bmatrix}$ | $\begin{bmatrix} 1.0099 & -0.0003 \\ -0.0003 & 1.0025 \end{bmatrix}$ |
| **Class** | **Misclassification Rates** | | | |
| % of samples used | 0.01% | 0.1% | 1% | 10% |
| $\omega_1$ | 28.83% | 2.46% | 2.88% | 2.80% |
| $\omega_2$ | 0.93% | 1.31% | 0.93% | 0.95% |
| Overall | 9.30% | 1.66% | 1.51% | 1.51% |

Table 2: ML estimated parameters for dataset A using different percentages of the samples.

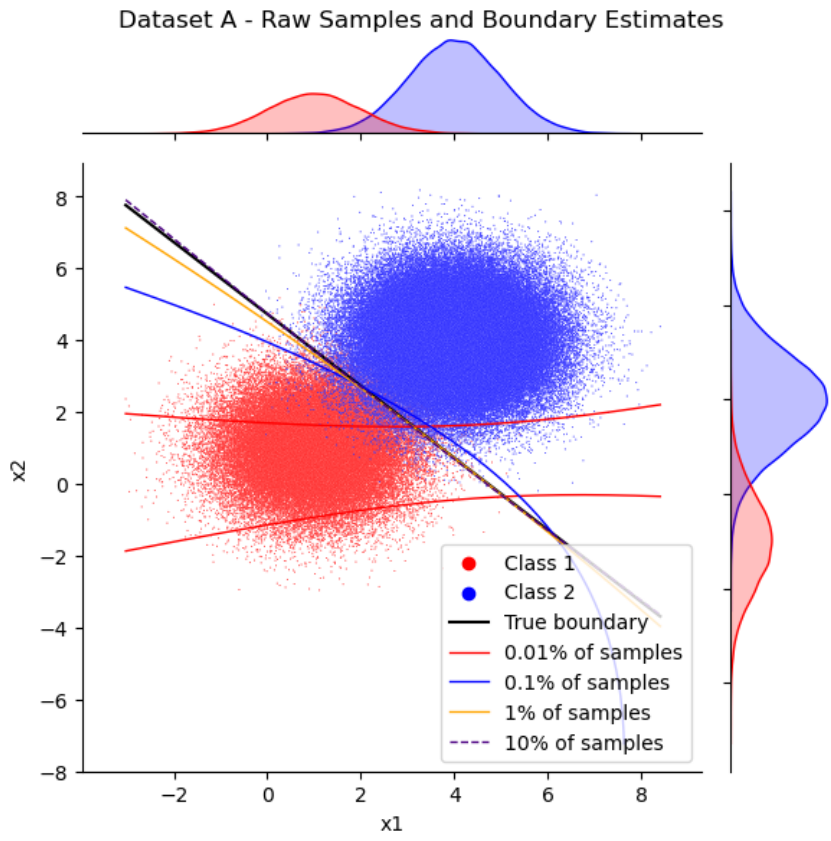



Figure 2: Unclassified sample data generated using the given parameters for data A in Assignment 1. Various decision boundaries created using ML estimates from Table 2 are shown as a means of visualizing how the data would be classified in each case.

We can see in Figure 2 that the boundaries created using ML estimates off of 0.01% and 0.1% are strongly non-linear, while the boundaries created using 1% or more of data set A quickly converge to the true boundary found using the given parameters.

**c. Comparison between parameter estimation assumptions**

| **Parameters** | **ML Estimates** | | | | |
|---|---|---|---|---|---|
| % of samples used | 0.01% | 0.1% | 1% | 10% | 100% |
| $\mu_1^T$ | [0.8, 0.7] | [0.8, 0.9] | [1, 0.9] | [1, 1] | [1, 1] |
| $\Sigma_1$ | $\begin{bmatrix} 1.3 & 0 \\ 0 & 0.1 \end{bmatrix}$ | $\begin{bmatrix} 1.2 & 0 \\ 0 & 0.7 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ |
| $\mu_2^T$ | [3.8, 4.3] | [4.1, 4.1] | [4, 4] | [4, 4] | [4, 4] |
| $\Sigma_2$ | $\begin{bmatrix} 0.7 & 0 \\ 0 & 0.5 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 0.8 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ |
| **Class** | **Misclassification Rates** | | | | |
| % of samples used | 0.01% | 0.1% | 1% | 10% | 100% |
| $\omega_1$ | 13.56% | 2.77% | 2.88% | 2.67% | 2.67% |
| $\omega_2$ | 0.80% | 1.15% | 0.93% | 1.01% | 1.01% |
| Overall | 4.63% | 1.63% | 1.51% | 1.51% | 1.51% |

Table 3: ML estimates using the assumption that features are not correlated and rounding estimates.

Part (b) demonstrated that the amount of data we use to estimate parameters can impact the accuracy of the resulting decision boundaries. Similarly, the assumptions we make about the estimated parameters can also have an impact. Table 3 lists the ML parameter estimates for sample set A when assuming that the features are not correlated, in addition to rounding the values. By making these assumptions, we find that the overall error rate for the 0.01% ML estimates is greatly improved from 9.30% (Table 2) to 4.63%. The overall misclassification rates for the 0.1% ML estimates also improved, albeit slightly, from 1.67% to 1.63%. It is clear that these two cases were the most impacted by these assumptions, as they both have higher rates of correlation in $\Sigma_1$ and $\Sigma_2$ as compared to the 1% and 10% estimates.

In Table 3, we can see that the ML estimated parameters for 1% of sample data are almost identical to the true parameters, with only $\mu_1$ being slightly off ([1, 0.9] versus [1, 1]). Aside from that small difference, the parameter estimates and misclassification rates have converged to the true parameters and corresponding misclassification rates shown in Table 1.

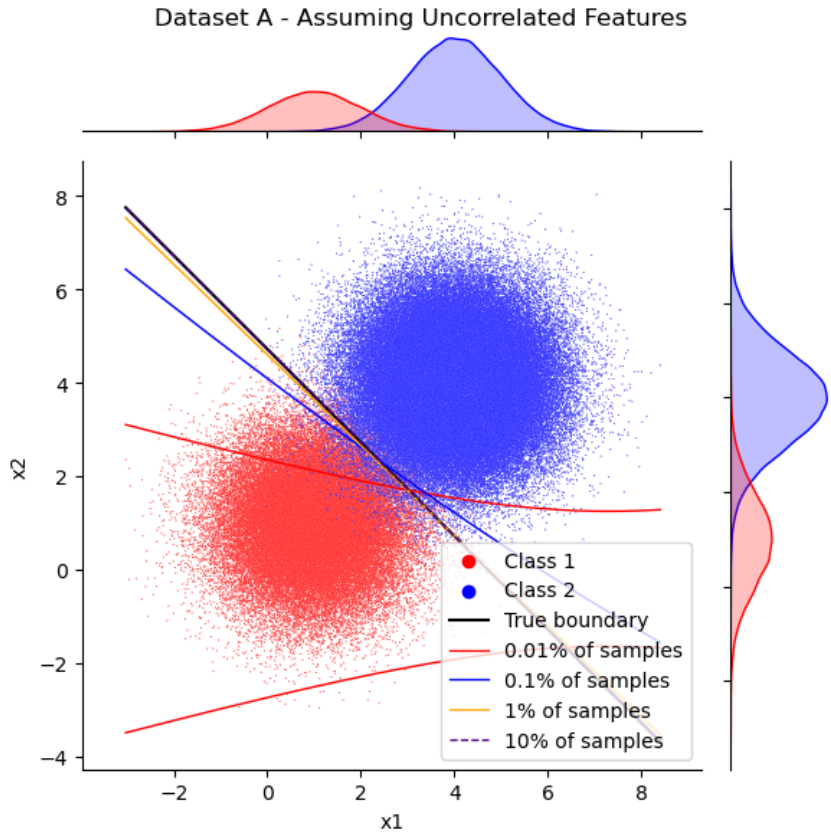



Figure 3: Unclassified sample data generated using the given parameters for data A in Assignment 1. Various decision boundaries created using ML estimates from Table 3 are shown for comparison with Figure 2.

As can be seen in Figure 3, making the assumption of uncorrelated features and rounding the estimates allows us to calculate the decision boundary using a case I discriminant, which simplifies the amount of calculations and the resulting decision boundary shape. These results demonstrate that making assumptions about the parameters to reduce the number of unknowns can be useful when dealing with smaller samples of training data.

# Experiment 2

### a. Estimating parameters of sample B using ML and comparing results with Assignment 1

The given parameters for sample set B are shown in Table 4, alongside the ML estimates using all 200,000 samples. We can see, once again, that the ML estimates and corresponding misclassification rates are very similar to the true parameter values and misclassification rates. Figure 4 demonstrates that the two boundaries are virtually identical.

| Parameters | True Values | ML Estimates |
|---|---|---|
| $\mu_1^T$ | $[1, 1]$ | $[0.9947, 0.9993]$ |
| $\Sigma_1$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 0.9928 & -0.0091 \\ -0.0091 & 1.000 \end{bmatrix}$ |
| $\mu_2^T$ | $[4, 4]$ | $[3.9904, 3.9972]$ |
| $\Sigma_2$ | $\begin{bmatrix} 4 & 0 \\ 0 & 8 \end{bmatrix}$ | $\begin{bmatrix} 3.9887 & -0.0087 \\ -0.0087 & 7.9712 \end{bmatrix}$ |
| **Class** | **Misclassification Rates** | |
| $\omega_1$ | 8.14% | 8.23% |
| $\omega_2$ | 7.29% | 7.25% |
| Overall | 7.55% | 7.54% |

Table 4: Comparison of the ML parameter estimates and true parameters for dataset B. The full sample (n=200,000) was used for estimating the parameters.

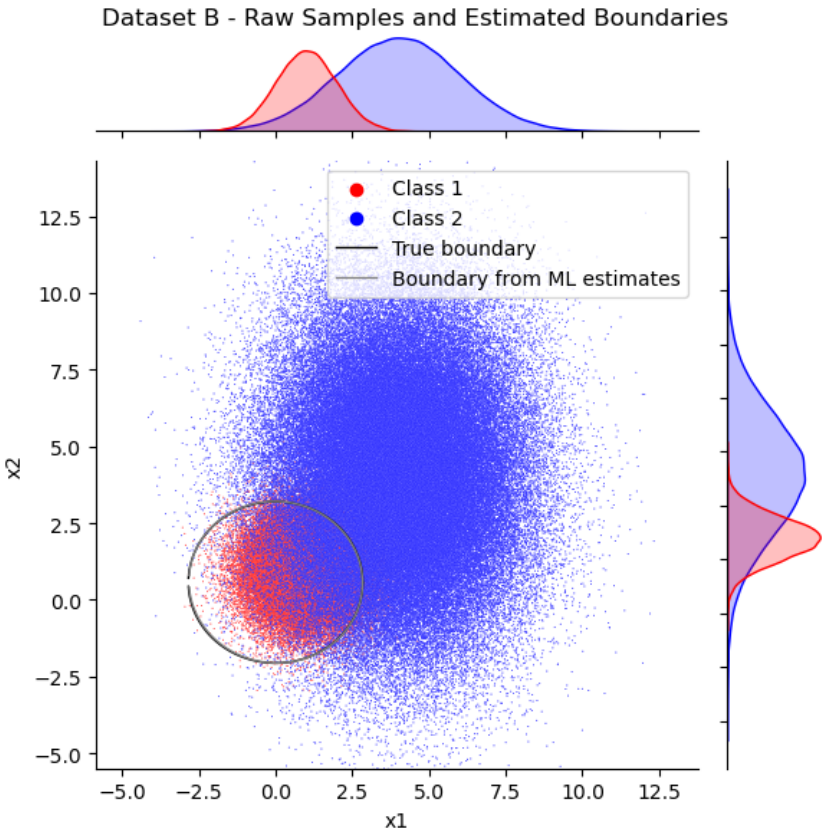



Figure 4: Sample data generated using the given parameters for data B in Assignment 1. The true boundary estimated from the given parameters is shown in black, while the boundary calculated from the ML estimated parameters is in grey.

### b. Testing classification accuracy with subsets of sample B

It can be seen in Table 5 that the ML estimates for data B are quite poor when using only 0.01% of the data. The overall misclassification rate for this case was 17.91%, with 53.57% of samples from $\omega_1$ being misclassified as $\omega_2$. We can see in Figure 5 that the decision boundary is much more narrow for the case where only 0.01% of samples were used, so many of the data points are classified as being from sample 2. The samples in data B have much more overlap than the samples in data A, so it makes sense that these misclassification rates would be higher. Despite this, the parameter estimates and misclassification rates improve significantly by using 0.1% of the data (i.e., 200 samples), and begin to converge to the true parameters when 1% or more of the data is used for estimation.

| **Parameters** | **ML Parameter Estimates** | | | |
|---|---|---|---|---|
| % of samples used | 0.01% | 0.1% | 1% | 10% |
| $\mu_1^T$ | [0.8123, 0.6543] | [0.8226, 0.9173] | [0.9503, 0.9244] | [0.9870, 0.9800] |
| $\Sigma_1$ | $\begin{bmatrix} 1.2658 & 0.1310 \\ 0.1310 & 0.0641 \end{bmatrix}$ | $\begin{bmatrix} 1.2309 & 0.2217 \\ 0.2217 & 0.7379 \end{bmatrix}$ | $\begin{bmatrix} 1.0262 & 0.0496 \\ 0.0496 & 0.9915 \end{bmatrix}$ | $\begin{bmatrix} 0.9930 & -0.01516 \\ -0.01516 & 1.01336 \end{bmatrix}$ |
| $\mu_2^T$ | [4.6011, 3.5682] | [4.1110, 4.2816] | [4.0062, 4.0611] | [3.9725, 4.0259] |
| $\Sigma_2$ | $\begin{bmatrix} 2.1769 & 1.2625 \\ 1.2625 & 5.5477 \end{bmatrix}$ | $\begin{bmatrix} 3.3623 & 0.08119 \\ 0.08119 & 8.0736 \end{bmatrix}$ | $\begin{bmatrix} 3.9815 & 0.1236 \\ 0.1236 & 8.3547 \end{bmatrix}$ | $\begin{bmatrix} 4.0098 & -0.0015 \\ -0.0015 & 8.0788 \end{bmatrix}$ |
| **Class** | **Misclassification Rates** | | | |
| % of samples used | 0.01% | 0.1% | 1% | 10% |
| $\omega_1$ | 53.57% | 10.16% | 8.76% | 8.33% |
| $\omega_2$ | 2.62% | 6.79% | 7.05% | 7.21% |
| Overall | 17.91% | 7.8% | 7.56% | 7.54% |

Table 5: ML estimated parameters for dataset B using different percentages of the samples.

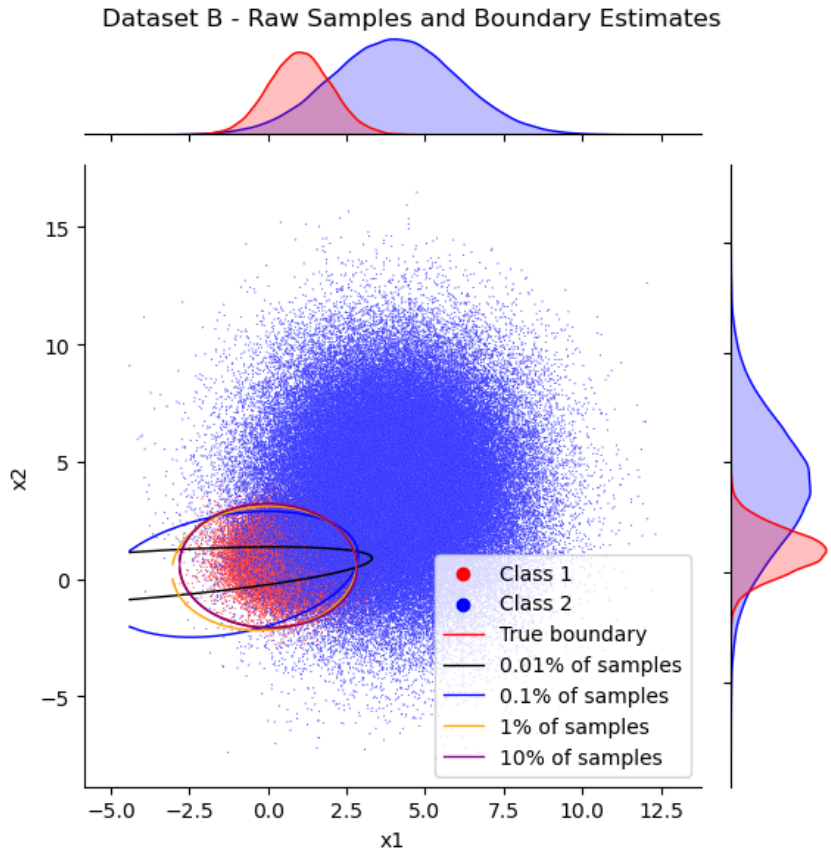



Figure 5: Unclassified sample data generated using the given parameters for data B in Assignment 1. Various decision boundaries created using ML estimates from Table 5 are shown as a means of visualizing how the data would be classified in each case.

### c. Comparison between parameter estimation assumptions

Once again, we can see in Table 6 that making the assumption that the ML estimated covariance matrices are uncorrelated, and rounding the estimates, improved the misclassification rates. For the case where only 0.01% of data was used, the overall misclassification rate dropped to 13.56% and the misclassification rate for $\omega_1$ dropped to 36.15%. These rates also dropped for the 0.1% case. We see that although the estimated parameters approach the true parameters when using 1% of the data, they don't quite converge as quickly as was the case with dataset A. This is likely due to the larger overlap between $\omega_1$ and $\omega_2$, which makes it more challenging to determine the most likely parameters for the data.

| **Parameters** | **ML Estimates** | | | | |
|---|---|---|---|---|---|
| % of samples used | 0.01% | 0.1% | 1% | 10% | 100% |
| $\mu_1^T$ | [0.8, 0.7] | [0.8, 0.9] | [1, 0.9] | [1, 1] | [1, 1] |
| $\Sigma_1$ | $\begin{bmatrix} 1.3 & 0 \\ 0 & 0.1 \end{bmatrix}$ | $\begin{bmatrix} 1.2 & 0 \\ 0 & 0.7 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ |
| $\mu_2^T$ | [4.6, 3.6] | [4.1, 4.3] | [4, 4.1] | [4, 4] | [4, 4] |
| $\Sigma_2$ | $\begin{bmatrix} 2.2 & 0 \\ 0 & 5.5 \end{bmatrix}$ | $\begin{bmatrix} 3.4 & 0 \\ 0 & 8.1 \end{bmatrix}$ | $\begin{bmatrix} 4 & 0 \\ 0 & 8.4 \end{bmatrix}$ | $\begin{bmatrix} 4 & 0 \\ 0 & 8.1 \end{bmatrix}$ | $\begin{bmatrix} 4 & 0 \\ 0 & 8 \end{bmatrix}$ |
| **Class** | **Misclassification Rates** | | | | |
| % of samples used | 0.01% | 0.1% | 1% | 10% | 100% |
| $\omega_1$ | 36.15% | 10.01% | 8.39% | 8.15% | 8.14% |
| $\omega_2$ | 3.88% | 6.70% | 7.24% | 7.30% | 7.29% |
| Overall | 13.56% | 7.69% | 7.59% | 7.55% | 7.55% |

Table 6: ML estimates using the assumption that features are not correlated and rounding estimates.

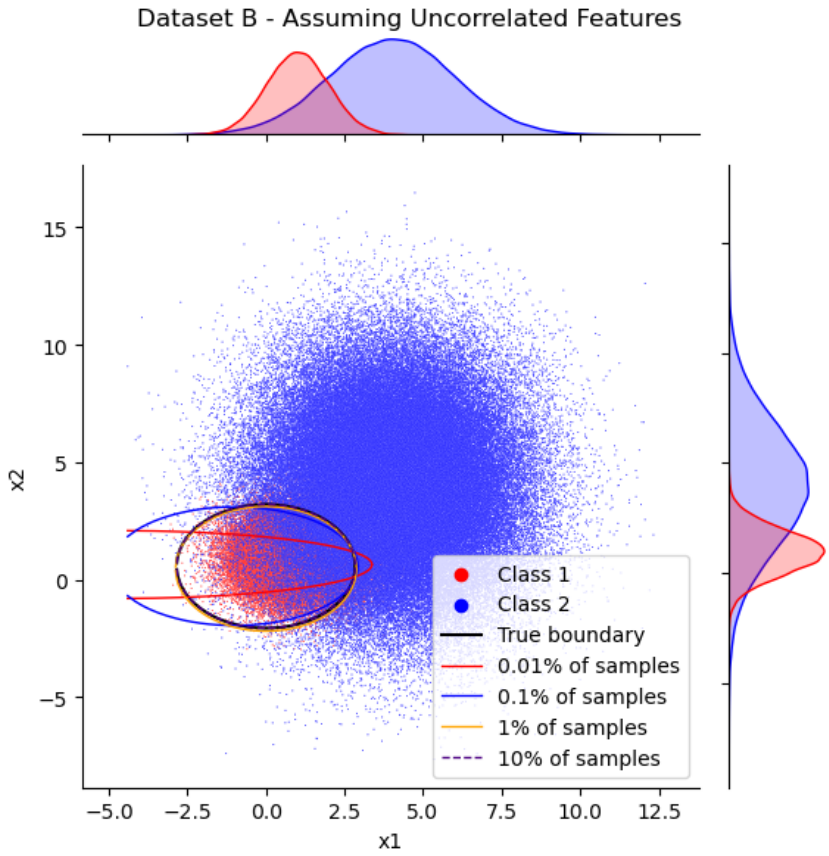



Figure 6: Unclassified sample data generated using the given parameters for data B in Assignment 1. Various decision boundaries created using ML estimates from Table 6 are shown for comparison with Figure 5.

Figure 6 shows that the decision boundaries for the ML estimates shifted under the assumption that features were uncorrelated, subsequently improving accuracy. It should be noted that although this assumption was helpful for data A and B parameter estimations (particularly in cases with only small amounts of training data) it may not be as beneficial or valid in cases where model features are strongly correlated.

# Experiment 3

### a. Building a model using the Chromatic Color Space

Figure 7 shows the training image used ('Training_1.ppm') to build a skin model using the chromatic color space and the corresponding reference image for face/skin pixels. The bottom left subplot shows the FAR and FRR rates calculated at different thresholds between [0,c], along with the approximate EER rate. The parameters for this model were estimated using the same ML approach as described in Experiments 1 and 2 (i.e., computing the sample mean and covariance of the skin pixels in the r,g space), and are listed in Table 7.

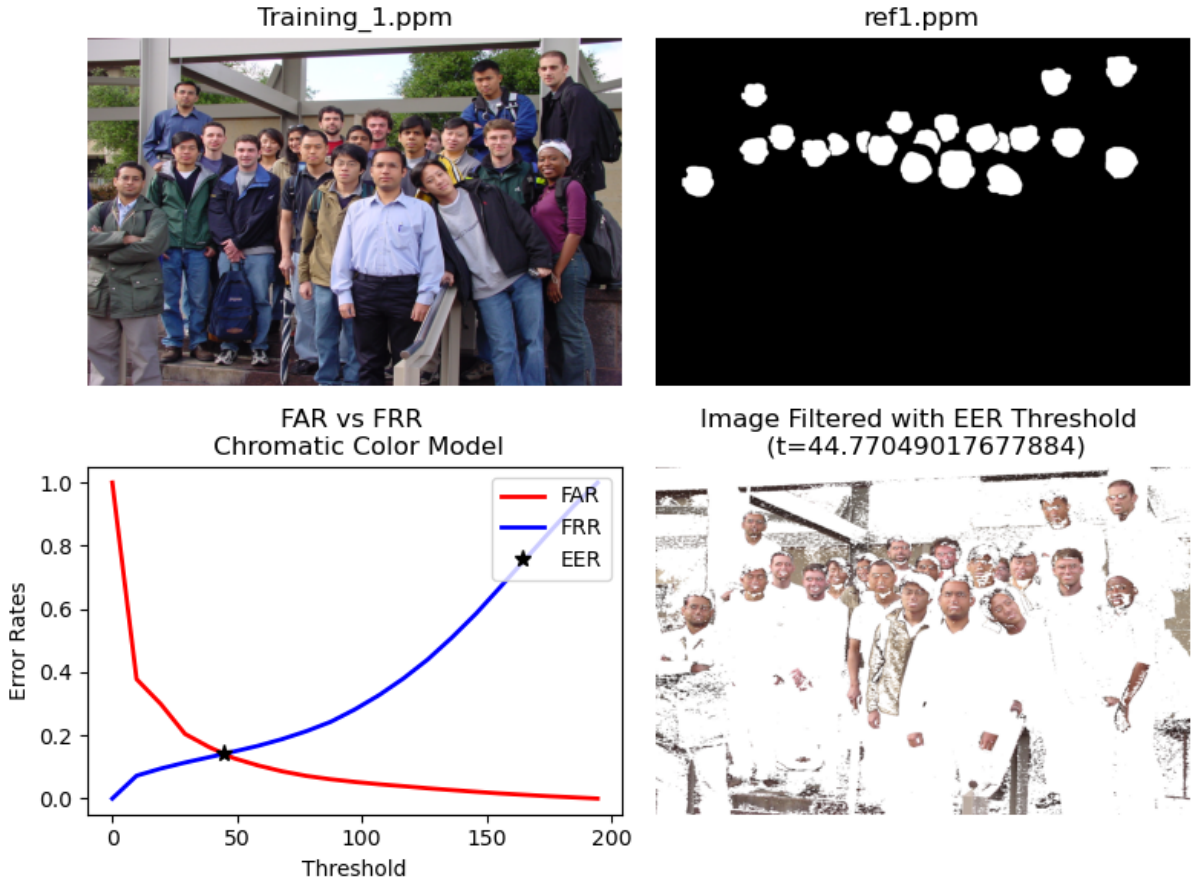



Figure 7: FAR and FRR rates for 'Training_1.ppm' alongside the image used to build the skin color model. Pixels in the bottom right subplot were classified as skin or not skin using the EER threshold from the bottom left subplot. Any pixel that was classified as not skin was assigned white.

Figures 8 and 9 show the results from testing the image classifier built from Chromatic color space parameter estimates in Table 7 on new images. We can see that the model performs well on new images, which demonstrates good model generalization.

| ML Parameter Estimates | | |
|---|---|---|
| **Parameters** | **Chromatic** | **YCbCr** |
| $\mu^T$ | [0.4322, 0.2958] | [-12.6913, 23.6343] |
| $\Sigma$ | $\begin{bmatrix} 0.002 & -0.001 \\ -0.001 & 0.0008 \end{bmatrix}$ | $\begin{bmatrix} 31.1970 & -21.7417 \\ -21.7417 & 48.9456 \end{bmatrix}$ |

Table 7: ML estimated parameters for experiment 3 skin color model. The estimates were found using skin data from 'Training_1.ppm'.

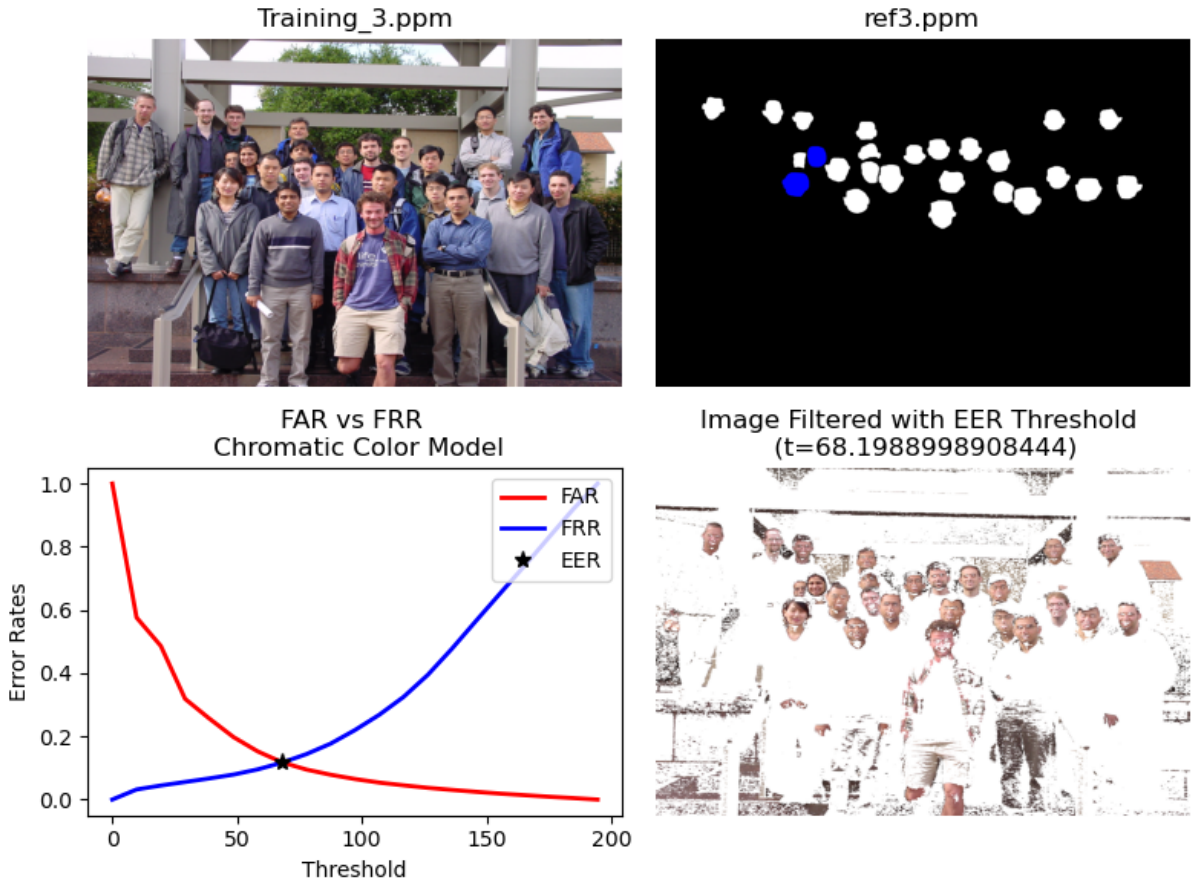



Figure 8: Test image, reference image, error rates/EER threshold, and the corresponding classified image, classified as skin or not skin using the EER threshold.

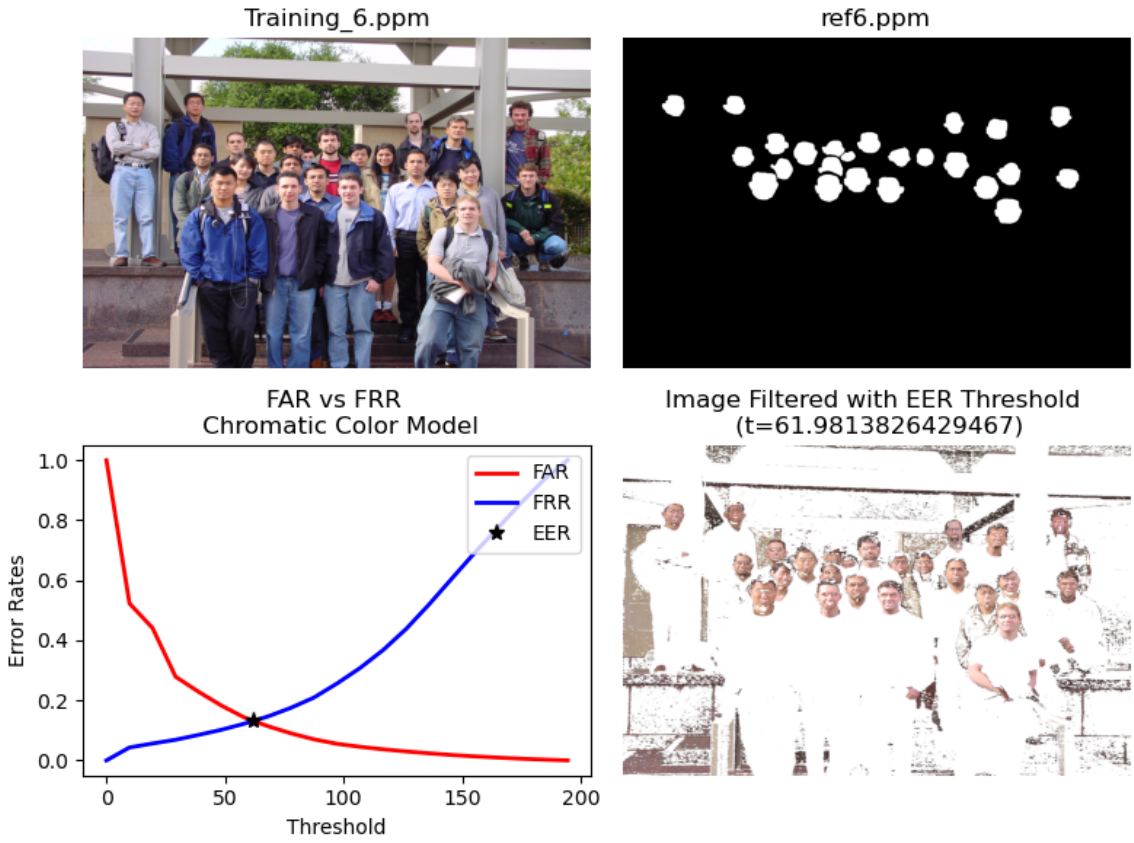



Figure 9: Test image, reference image, error rates/EER threshold, and the corresponding classified image, classified as skin or not skin using the EER threshold.

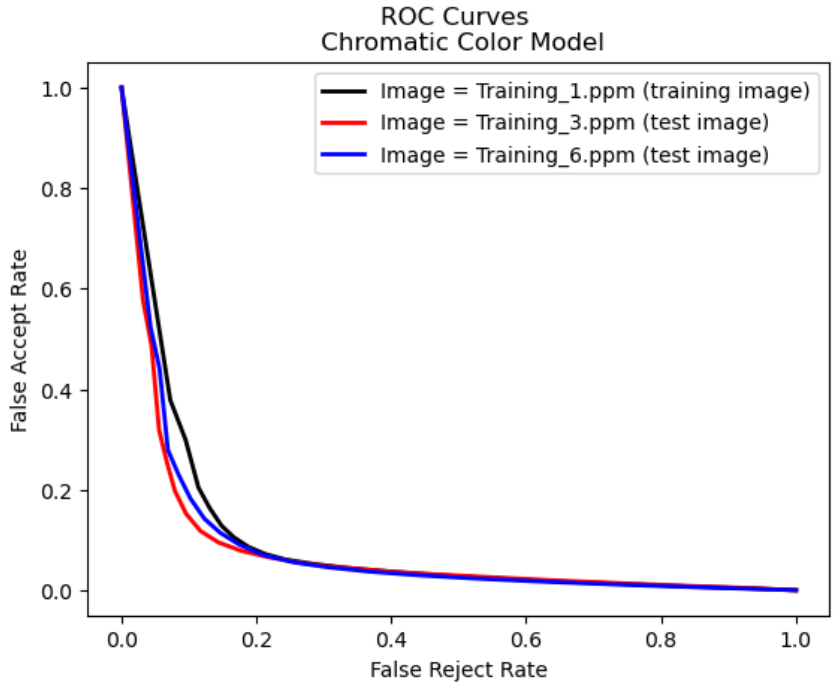



Figure 10: ROC curves for each of the 3 provided images using the chromatic color model.

Figure 10 demonstrates the ROC curve for each image used during training and testing of the Chromatic skin color model. Although 'Training_1.ppm' was used to training the model, we can see that the area under the curve (AUC) is slightly larger than the other two images, which indicates that our training image had higher error rates than the testing images during classification, albeit just slightly. Overall, the classifier performed similarly well on all three images.

**b. Building a model using the YCbCr Color Space**

The ML parameter estimates for a skin color model using the YCbCr color space were found using the same training image as in part a, and are listed Table 7. It should be noted that these parameters are not directly comparable, since the pixel color scale in the two color spaces is not the same. Even so, they are provided as a reference point. The training image and error rates for the YCbCr skin color model are shown in Figure 11.

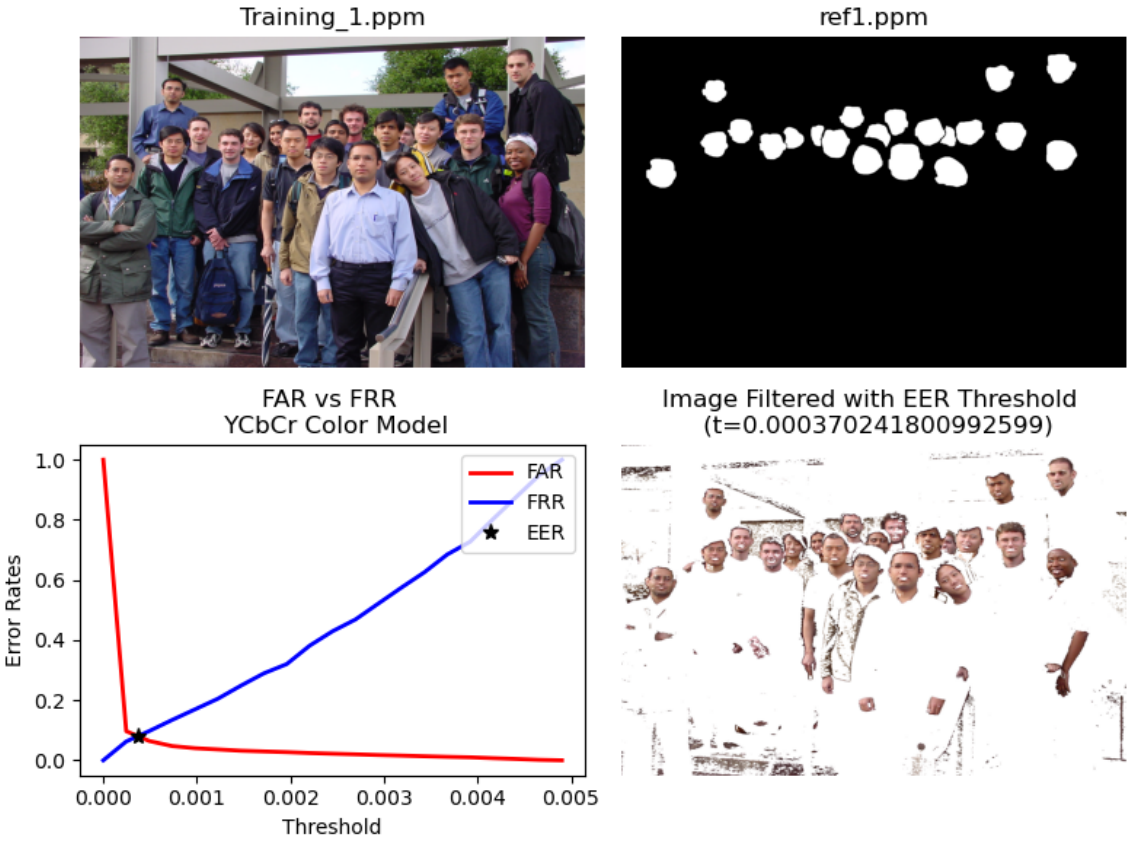



Figure 11: Training image, reference image, error rates/EER threshold, and the corresponding classified image, classified as skin or not skin using the EER threshold for Training_3.ppm.

Figures 12 and 13 demonstrate the classification results using the estimated EER for each respective image. Once again, we can see that the YCbCr model was able to perform well by using the training image (corresponding parameters listed in Table 7) to classify the test images ('Training_3.ppm' and 'Training_6.ppm').

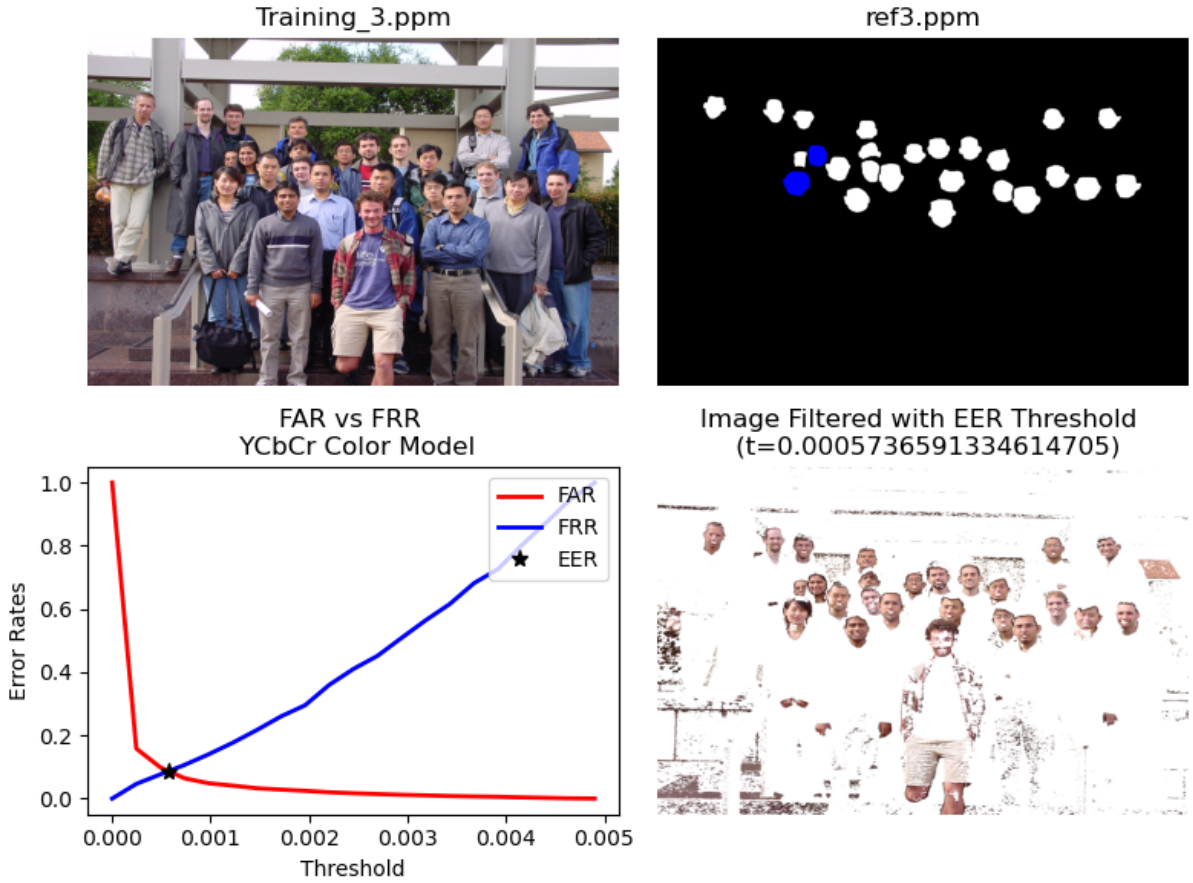



Figure 12: Test image, reference image, error rates/EER threshold, and the corresponding classified image, classified as skin or not skin using the EER threshold for Training_6.ppm.

It can be seen that although the parameter estimates were the same for both test images, the Equal Error Rates and corresponding thresholds varied slightly for each photo. This indicates that although the skin model applies well to new photos, it does require some amount of adjustment, and would perhaps not perform as well if a single threshold was chosen for categorizing every image. This falls in line with the results from Yang and Waibel (1996), since the second part of their paper focuses on a way to adjust the skin model to account for human movement and/or lighting changes in a video.

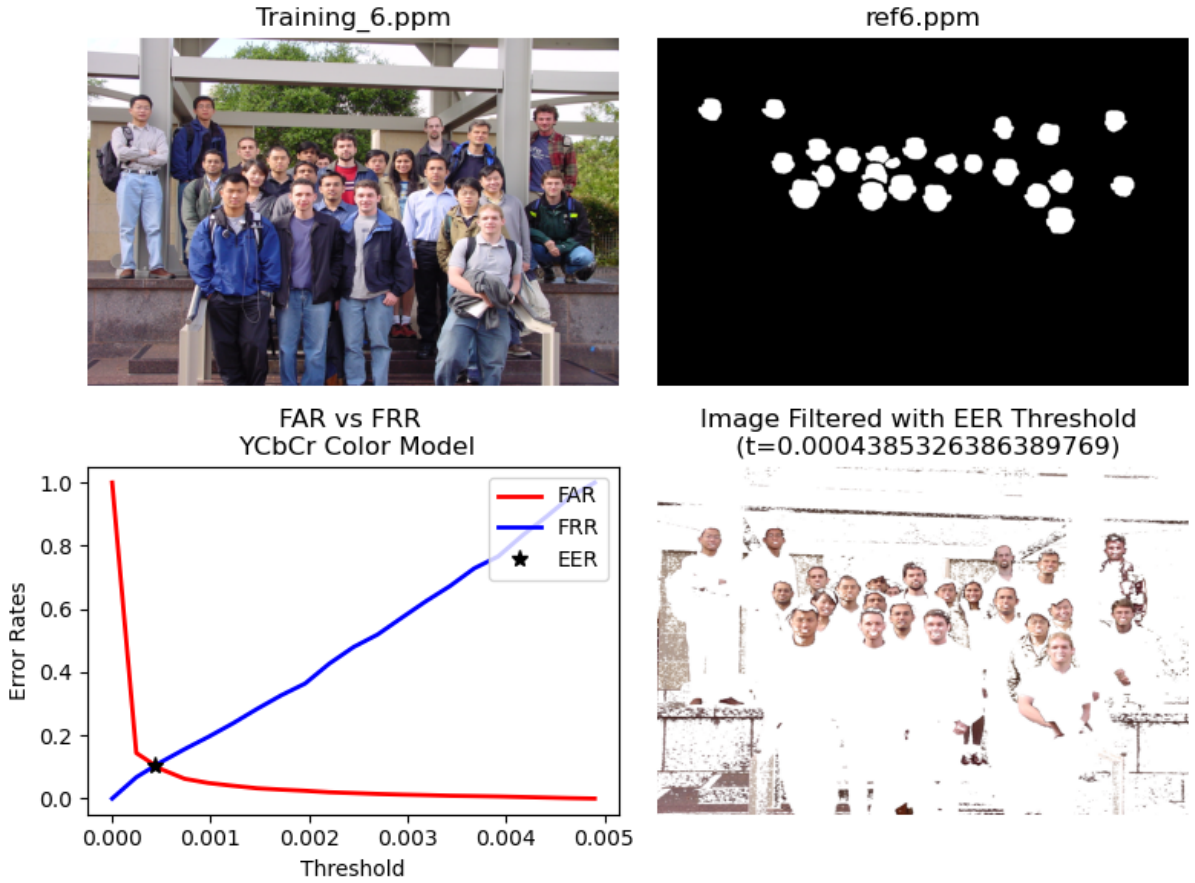



Figure 13: Test image, reference image, error rates/EER threshold, and the corresponding classified image, classified as skin or not skin using the EER threshold.

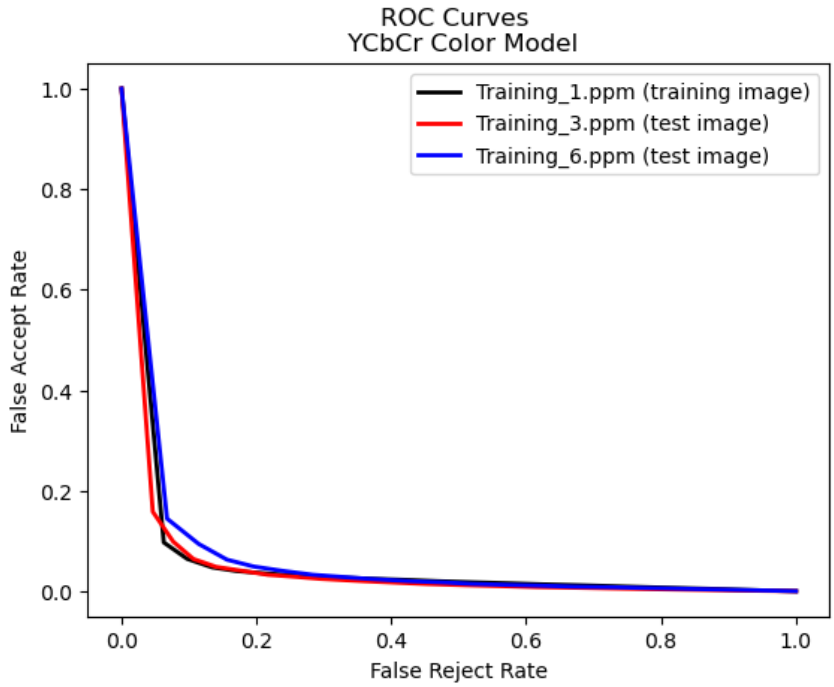



Figure 14: ROC curves for each of the 3 provided images using the YCbCr color model.

The ROC curves for the YCbCr skin color model are shown in Figure 14. We can see that the three images all have similar error rates at the calculated thresholds, indicating that the range of skin colors were indeed similar across photos and that the model had good generalization performance. It should be noted that we can see some areas of each image that are non-face/skin (i.e., arms and hands). These areas were not identified in the reference images, so their classification as skin would contribute to a False Positive error rate even though they are in fact skin. Thus, the estimated error rates of this model may be slightly higher than the the true error rates.

**c. Comparing the ROC curves between models**

We can see in Figure 15 that the AUC was smaller for the two test images when using a YCbCr model. This suggests that the YCbCr color space may be a better choice for accurately detecting skin in images. To further prove this, one could expand this analysis to additional images in order to better assess classification performance between color space models.

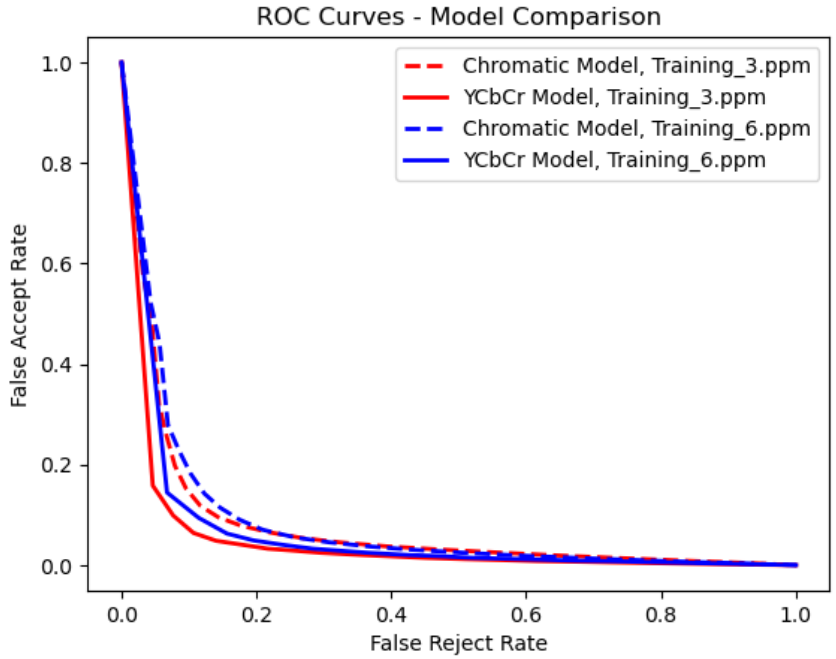



Figure 15: ROC curves for each of the 3 provided images using the YCbCr color model.

# References

Duda, R. O., Hart, P. E., et al. (2006). *Pattern classification*. John Wiley & Sons.

Yang, J. and Waibel, A. (1996). A real-time face tracker. In *Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96*, pages 142–147. IEEE.

# Appendix

## Code

Instructions to run code and generate figures:

All code used to generate the data and figures in this assignment is included in separate files which consist of the following:

'BayesianClassifierHW2.py': Python script used in Assignment 1, with some small modifications to allow for storing/using ML estimated parameters.

'MaxLikelihoodEstimator.py': Python script that inherits the functionality from BayesianClassifier.py and implements all functions needed for Experiments 1 and 2.

'ImageClassifier.py': Python script that performs the functions needed for Experiment 3. Calls upon 'MaxLikelihoodEstimator.py' to get ML estimates for the training data.

'CS679_ASSIGNMENT2_JaleesaHoule.ipynb': The actual data generation and analysis was performed in a Jupyter Notebook environment. A pdf of this notebook is also included in the pages below.

To run this code, download the python scripts and the Jupyter Notebook file into the same directory. Open the Jupyter Notebook and select 'run all'. This should re-run all of the analyses conducted for this assignment. The code requires installation of packages Open-CV, SciPy, Numpy, Pandas, Sympy, Matplotlib, and Seaborn. This code was run using Python 3.8.15.

# CS679_ASSIGNMENT2_JaleesaHoule

March 30, 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

[2]:
```
import BayesianClassifierHW2 as BC
import MaxLikelihoodEstimator as ML
import ImageClassifier as IC
```

## 1 Experiment 1

### 1.1 Part (a)

#### 1.1.1 Using exactly the same 200,000 samples from assignment #1 (i.e., 60,000 samples from N( 1,Σ1) and 140,000 samples from N( 2,Σ2)), estimate the parameters of each distribution using the ML approach. Then, classify all 200,000 samples using a Bayes classifier, count the number of misclassifications (for each class and overall), and compare your results with those obtained from assignment #1.

[3]:
```
# data A params
mu1 = np.array([1,1])
cov1 = np.array([[1,0],[0,1]])
mu2 = np.array([4,4])
cov2 = np.array([[1,0],[0,1]])

mu= [mu1,mu2 ]
cov =[cov1, cov2]
n_samples= [60000,140000]
```

[4]:
```
datasetA_ML = ML.ML_Estimator(mu,cov, n_samples)

## classify using the ML estimated mu and cov params
datasetA_ML.classify_using_ML_estimates(print_params=True,)

ML_estimated_boundary = datasetA_ML.boundary_function

datasetA_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60000
Number of samples used from class 2 : 140000

 ML Estimated Means:

Sample 1 : [0.9947335309849321, 0.9993340872355969]
Sample 2 : [3.9989997647685853, 3.9951898904939456]

 ML Estimated Covariances:

Sample 1 : [[ 0.99284676 -0.00905706]
 [-0.00905706  1.00031409]]
Sample 2 : [[ 0.99645636 -0.00154648]
 [-0.00154648  0.99716691]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [1627, 1399]
Misclassification rate by class: [0.02711667 0.00999286]
Total misclassification rate: 0.01513

 ****************************************************************
```

### 1.1.2 Comparing with assignment 1 results

[5]:
```
datasetA = BC.SampleData(mu,cov, n_samples)
datasetA.run_all_analysis()
datasetA.get_true_error(print_params=True)
```

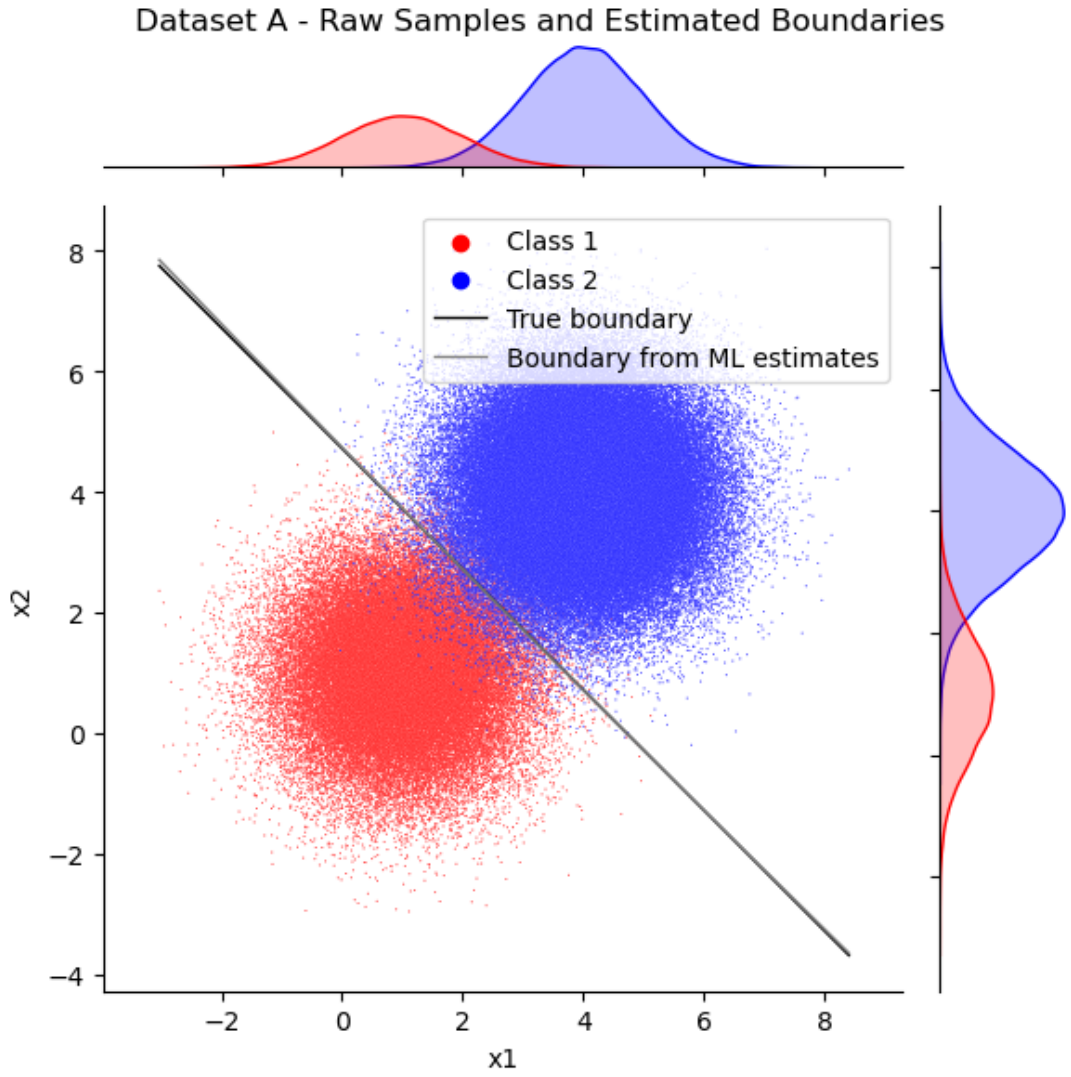
```
 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [1603, 1414]
```

```
Misclassification rate by class: [0.02671667 0.0101    ]
Total misclassification rate: 0.015085

 ****************************************************************
```

### 1.1.3 Plotting the boundaries from true and estimated parameters

[6]:
```
ax = datasetA_ML.make_plot(plot_type='seaborn', sort_type='raw', title␣
 ↪='Dataset A - Raw Samples and Estimated Boundaries', show_boundary=False)
# add true boundary from given params
ax.ax_joint.plot(np.array(datasetA.df.x1.sort_values()), np.array(datasetA.
 ↪boundary_function(datasetA_ML.df.x1.sort_values())), lw=1,color='k',␣
 ↪label='True boundary')

#add ML estimated boundary (no assumptions/rounding for estimated covariance␣
 ↪matrix)
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary[0](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='grey', label='Boundary from ML estimates')


ax.ax_joint.legend(loc='upper right')
```

[6]:
```
<matplotlib.legend.Legend at 0x7fa34a2dbf70>
```




## 1.2 Part (b)

**1.2.1 Next, you will test how the number of training data affects parameter estimation and consequently, classification accuracy. For this, consider using only (i) 0.01%, (ii) 0.1%, (iii) 1%, and (vi) 10% of the samples from each density (randomly chosen) to estimate the parameters of the two densities using ML. Then, classify all 200,000 samples for each case, count the number of misclassified samples (for each class and overall), and compare your results with those obtained in (1.a).**

### 1.2.2 (i) 0.01%

[7]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True, training_size=0.0001)
ML_estimated_boundary_001 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6
Number of samples used from class 2 : 14

 ML Estimated Means:

Sample 1 : [0.8123355251448224, 0.6542570359524086]
Sample 2 : [3.847348394361645, 4.300525252663553]

 ML Estimated Covariances:

Sample 1 : [[1.26582959 0.13102945]
 [0.13102945 0.06413848]]
Sample 2 : [[0.69345853 0.22318866]
 [0.22318866 0.54422446]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [17298, 1304]
Misclassification rate by class: [0.2883     0.00931429]
Total misclassification rate: 0.09301

 ********************************************************************
```

### 1.2.3 (ii) 0.1%

[8]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True, training_size=0.001)
ML_estimated_boundary_01 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60
Number of samples used from class 2 : 140

 ML Estimated Means:

Sample 1 : [0.8225988179488187, 0.9172535695041794]
Sample 2 : [4.0995752056489545, 4.055502434602157]

 ML Estimated Covariances:

Sample 1 : [[1.23085745 0.22173152]
 [0.22173152 0.7378921 ]]
Sample 2 : [[1.0092033  0.01435404]
 [0.01435404 0.8405696 ]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [1478, 1840]
Misclassification rate by class: [0.02463333 0.01314286]
Total misclassification rate: 0.01659

 ****************************************************************
```

#### 1.2.4 (iii) 1%

[9]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True, training_size=0.01)
ML_estimated_boundary_1 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 600
Number of samples used from class 2 : 1400
```

```
 ML Estimated Means:

Sample 1 : [0.9503036800748929, 0.9244012004219841]
Sample 2 : [4.021595533686607, 4.00309676415328]

 ML Estimated Covariances:

Sample 1 : [[1.02624498 0.04957027]
 [0.04957027 0.99147816]]
Sample 2 : [[1.04434352 0.02185456]
 [0.02185456 0.99537656]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [1729, 1300]
Misclassification rate by class: [0.02881667 0.00928571]
Total misclassification rate: 0.015145

 ********************************************************************
```

### 1.2.5 (vi) 10%

[10]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True, training_size=0.1)
ML_estimated_boundary_10 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6000
Number of samples used from class 2 : 14000

 ML Estimated Means:

Sample 1 : [0.9870200000375329, 0.9800307852674556]
Sample 2 : [4.009156500018937, 3.9862694088777926]

 ML Estimated Covariances:
```

```
Sample 1 : [[ 0.9930342  -0.01515895]
 [-0.01515895  1.01335585]]
Sample 2 : [[ 1.00985040e+00 -2.57746743e-04]
 [-2.57746743e-04  1.00245075e+00]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [1682, 1335]
Misclassification rate by class: [0.02803333 0.00953571]
Total misclassification rate: 0.015085

 ********************************************************************
```

### 1.2.6 (v) 100%

[11]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True, training_size='all')
ML_estimated_boundary_all = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60000
Number of samples used from class 2 : 140000

 ML Estimated Means:

Sample 1 : [0.9947335309849321, 0.9993340872355969]
Sample 2 : [3.9989997647685853, 3.9951898904939456]

 ML Estimated Covariances:

Sample 1 : [[ 0.99284676 -0.00905706]
 [-0.00905706  1.00031409]]
Sample 2 : [[ 0.99645636 -0.00154648]
 [-0.00154648  0.99716691]]

 ********************************************************************
```

```
**********************************************************************

 Empirical Error Stats:

N samples misclassified: [1627, 1399]
Misclassification rate by class: [0.02711667 0.00999286]
Total misclassification rate: 0.01513

**********************************************************************
```

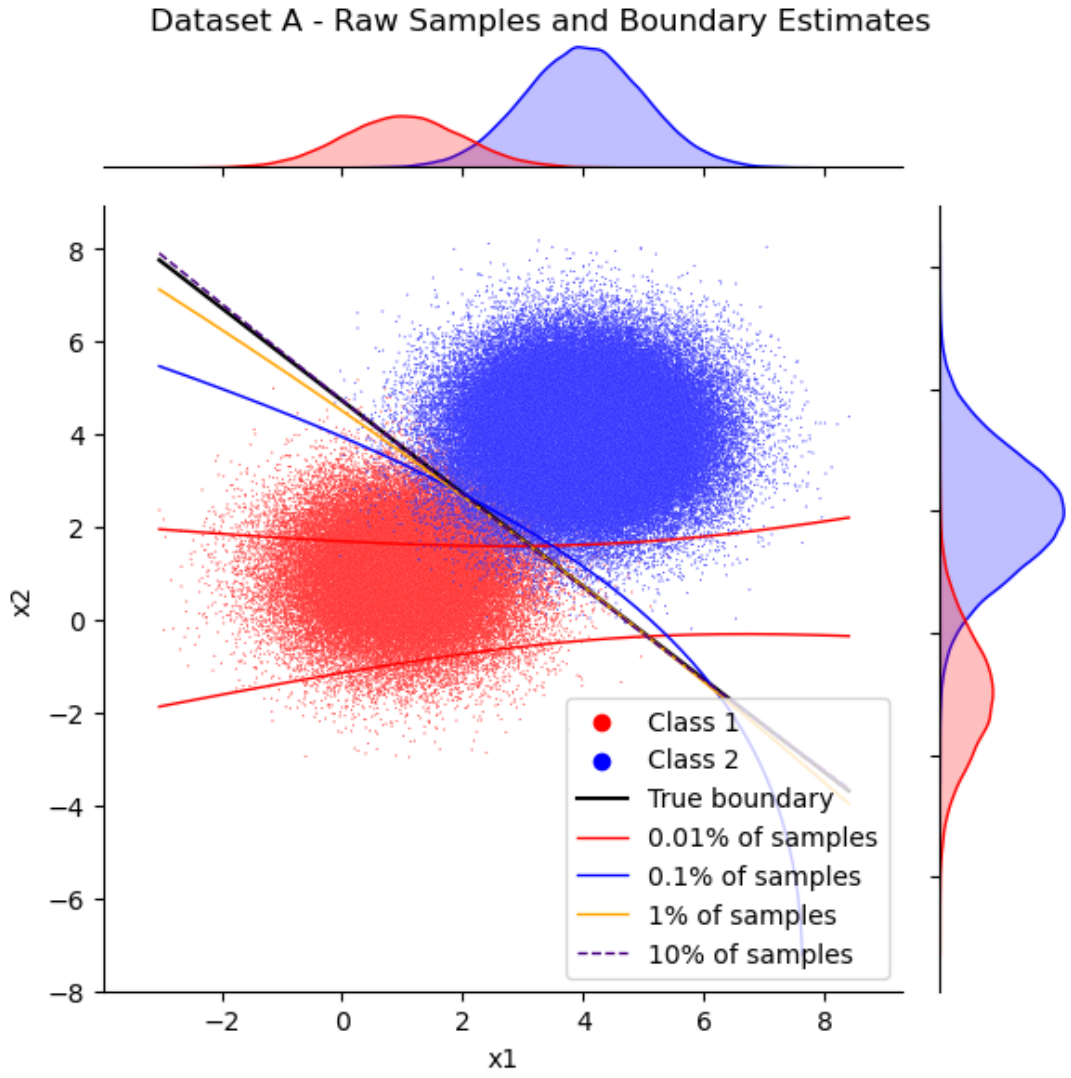
### 1.2.7 Plotting the boundaries using each set of estimated parameters

[12]:
```
ax = datasetA_ML.make_plot(plot_type='seaborn', sort_type='raw', title='Dataset␣
 ↪A - Raw Samples and Boundary Estimates', show_boundary=False)

#100%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.array(datasetA.
 ↪boundary_function(datasetA_ML.df.x1.sort_values())), lw=1.5,color='k',␣
 ↪label='True boundary')


#0.01%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[0](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red', label='0.01% of samples')
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[1](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red')

#.1%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[1](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue',label='0.1% of samples')

#1%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_1[1](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='orange', label='1% of samples')

#10%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_10[0](datasetA_ML.df.x1.sort_values())),'--',␣
 ↪lw=1, color='indigo', label='10% of samples')
```

```
ax.ax_joint.set_aspect('auto')
ax.ax_joint.legend(loc='lower right')
```

```
/Volumes/MacBackup/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[12]: <matplotlib.legend.Legend at 0x7fa34a294850>




### 1.3 sub-analysis:

#### 1.3.1 Experiment with (a) choosing the optimum Case number based on the estimated covariance matrices (since we do not really know the true covariance matrices in practice) and (b) choosing the optimum Case number after explicitly setting the off-diagonal elements of the covariance matrices to zero by assuming that the features are uncorrelated. Carefully analyze your results.

#### 1.3.2 choosing the optimum Case number after explicitly setting the off-diagonal elements of the covariance matrices to zero

#### 1.3.3 (i) using 0.01% of data

[13]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
↪round_estimates=True, training_size=0.0001)
ML_estimated_boundary_001 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6
Number of samples used from class 2 : 14

 ML Estimated Means:

Sample 1 : [0.8 0.7]
Sample 2 : [3.8 4.3]

 ML Estimated Covariances:

Sample 1 : [[1.3 0. ]
 [0.  0.1]]
Sample 2 : [[0.7 0. ]
 [0.  0.5]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [8135, 1123]
Misclassification rate by class: [0.13558333 0.00802143]
Total misclassification rate: 0.04629

 ********************************************************************
```

### 1.3.4 (ii) using 0.1% of data

[14]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
↪round_estimates=True, training_size=0.001)
ML_estimated_boundary_01 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60
Number of samples used from class 2 : 140

 ML Estimated Means:

Sample 1 : [0.8 0.9]
Sample 2 : [4.1 4.1]

 ML Estimated Covariances:

Sample 1 : [[1.2 0. ]
 [0.  0.7]]
Sample 2 : [[1.  0. ]
 [0.  0.8]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [1661, 1603]
Misclassification rate by class: [0.02768333 0.01145   ]
Total misclassification rate: 0.01632

 ********************************************************************
```

### 1.3.5 (iii) using 1% of data

[15]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
↪round_estimates=True, training_size=0.01)
ML_estimated_boundary_1 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 600
Number of samples used from class 2 : 1400

 ML Estimated Means:

Sample 1 : [1.  0.9]
Sample 2 : [4. 4.]

 ML Estimated Covariances:

Sample 1 : [[1. 0.]
 [0. 1.]]
Sample 2 : [[1. 0.]
 [0. 1.]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [1725, 1297]
Misclassification rate by class: [0.02875    0.00926429]
Total misclassification rate: 0.01511

 ********************************************************************
```

### 1.3.6 (vi) using 10% of data

[16]:
```
datasetA_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
↪round_estimates=True, training_size=0.1)
ML_estimated_boundary_10 = datasetA_ML.boundary_function
datasetA_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6000
Number of samples used from class 2 : 14000

 ML Estimated Means:

Sample 1 : [1. 1.]
Sample 2 : [4. 4.]

 ML Estimated Covariances:

Sample 1 : [[ 1. -0.]
 [-0.  1.]]
Sample 2 : [[ 1. -0.]
 [-0.  1.]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [1603, 1414]
Misclassification rate by class: [0.02671667 0.0101    ]
Total misclassification rate: 0.015085

 ****************************************************************
```

### 1.3.7 (v) using 100% of data

[17]:
```
## classify using the ML estimated mu and cov params; rounding the estimates␣
↪and assuming no correlation between features
datasetA_ML.classify_using_ML_estimates(corr=False, round_estimates=True,␣
↪print_params=True)
datasetA_ML.get_true_error(print_params=True)
```

```
ML_estimated_boundary_rounded = datasetA_ML.boundary_function
```

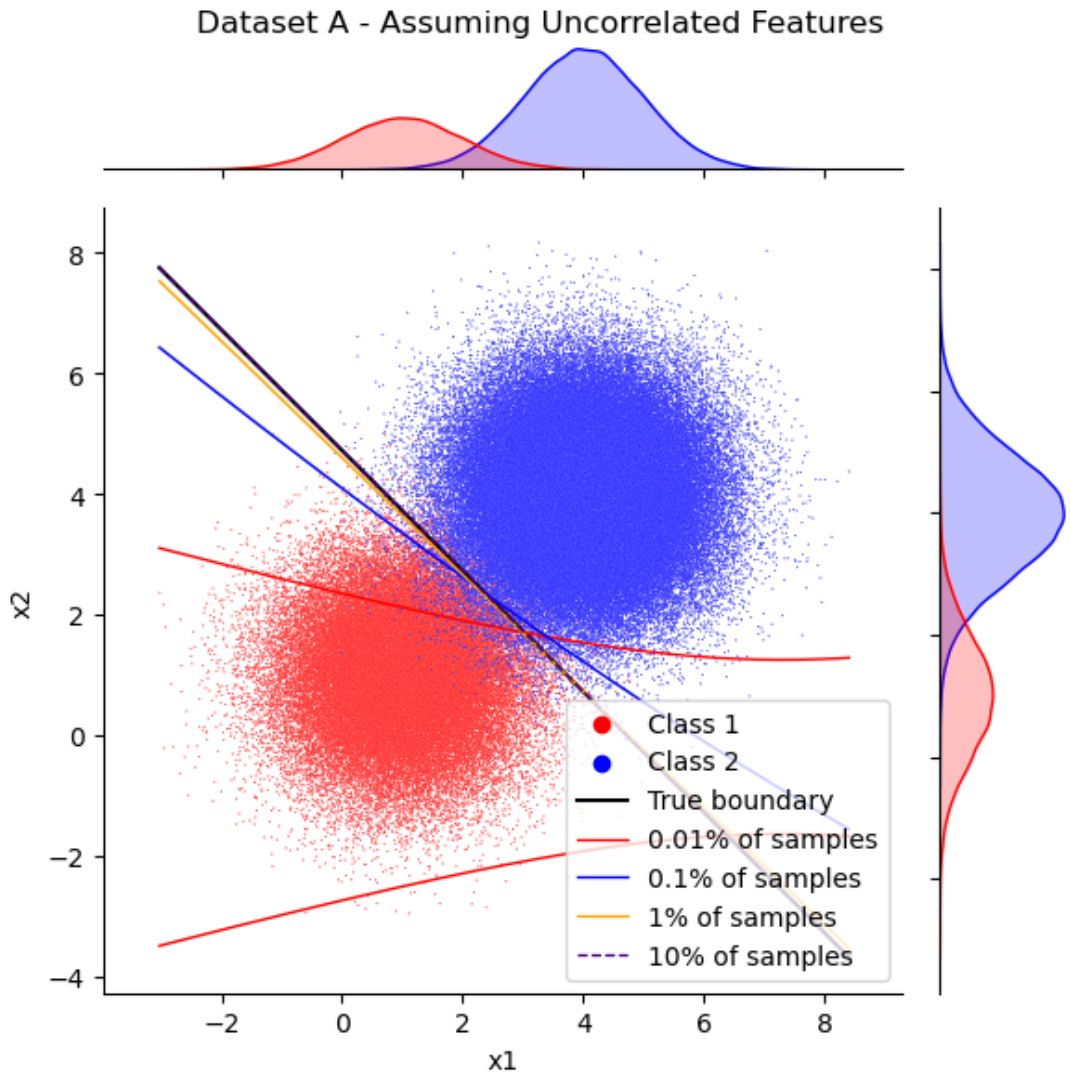
```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60000
Number of samples used from class 2 : 140000

 ML Estimated Means:

Sample 1 : [1. 1.]
Sample 2 : [4. 4.]

 ML Estimated Covariances:

Sample 1 : [[ 1. -0.]
 [-0.  1.]]
Sample 2 : [[ 1. -0.]
 [-0.  1.]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [1603, 1414]
Misclassification rate by class: [0.02671667 0.0101    ]
Total misclassification rate: 0.015085

 ****************************************************************
```

### 1.3.8 Plotting the boundaries using each set of estimated parameters

[18]:
```
ax = datasetA_ML.make_plot(plot_type='seaborn', sort_type='raw', title='Dataset␣
 ↪A - Assuming Uncorrelated Features', show_boundary=False)

#100%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_rounded(datasetA_ML.df.x1.sort_values())), lw=1.
 ↪5,color='k', label='True boundary')
```

```
#0.01%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[0](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red', label='0.01% of samples')
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[1](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red')

#.1%
#ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[0](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue',label='0.1% of samples')
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[1](datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue',label='0.1% of samples')

#1%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_1(datasetA_ML.df.x1.sort_values())),␣
 ↪lw=1,color='orange', label='1% of samples')

#10%
ax.ax_joint.plot(np.array(datasetA_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_10(datasetA_ML.df.x1.sort_values())),'--', lw=1,␣
 ↪color='indigo', label='10% of samples')



ax.ax_joint.set_aspect('auto')
ax.ax_joint.legend(loc='lower right')
```

[18]: <matplotlib.legend.Legend at 0x7fa332aa13d0>




## 2 Experiment 2

**2.1 Repeat experiment 1 using the 2D Gaussian densities for sample B; for comparison purposes, use exactly the same 200,000 samples from assignment #1.**

**2.2 Part (a)**

### 2.2.1 Using exactly the same 200,000 samples from assignment #1 (i.e., 60,000 samples from N( 1,Σ1) and 140,000 samples from N( 2,Σ2)), estimate the parameters of each distribution using the ML approach. Then, classify all 200,000 samples using a Bayes classifier, count the number of misclassifications (for each class and overall), and compare your results with those obtained from assignment #1.

[19]:
```
# data B params
mu1 = np.array([1,1])
cov1 = np.array([[1,0],[0,1]])
mu2 = np.array([4,4])
cov2 = np.array([[4,0],[0,8]])

mu= [mu1,mu2 ]
cov =[cov1, cov2]
n_samples= [60000,140000]
```

[20]:
```
datasetB_ML = ML.ML_Estimator(mu,cov, n_samples)

## classify using the ML estimated mu and cov params
datasetB_ML.classify_using_ML_estimates(print_params=True,)

ML_estimated_boundary = datasetB_ML.boundary_function

datasetB_ML.get_true_error(print_params=True)
```

```
 **********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60000
Number of samples used from class 2 : 140000

 ML Estimated Means:

Sample 1 : [0.9947335309849321, 0.9993340872355969]
Sample 2 : [3.990379780987892, 3.99717090754034]

 ML Estimated Covariances:

Sample 1 : [[ 0.99284676 -0.00905706]
 [-0.00905706  1.00031409]]
Sample 2 : [[ 3.98866766 -0.00874819]
 [-0.00874819  7.97165084]]

 **********************************************************************
```

```
 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [4936, 10143]
Misclassification rate by class: [0.08226667 0.07245   ]
Total misclassification rate: 0.075395

 ****************************************************************
```

### 2.2.2 Comparing with assignment 1 results

[21]:
```
datasetB = BC.SampleData(mu,cov, n_samples)
datasetB.run_all_analysis()
datasetB.get_true_error(print_params=True)
```

```
 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [4885, 10206]
Misclassification rate by class: [0.08141667 0.0729    ]
Total misclassification rate: 0.075455

 ****************************************************************
```

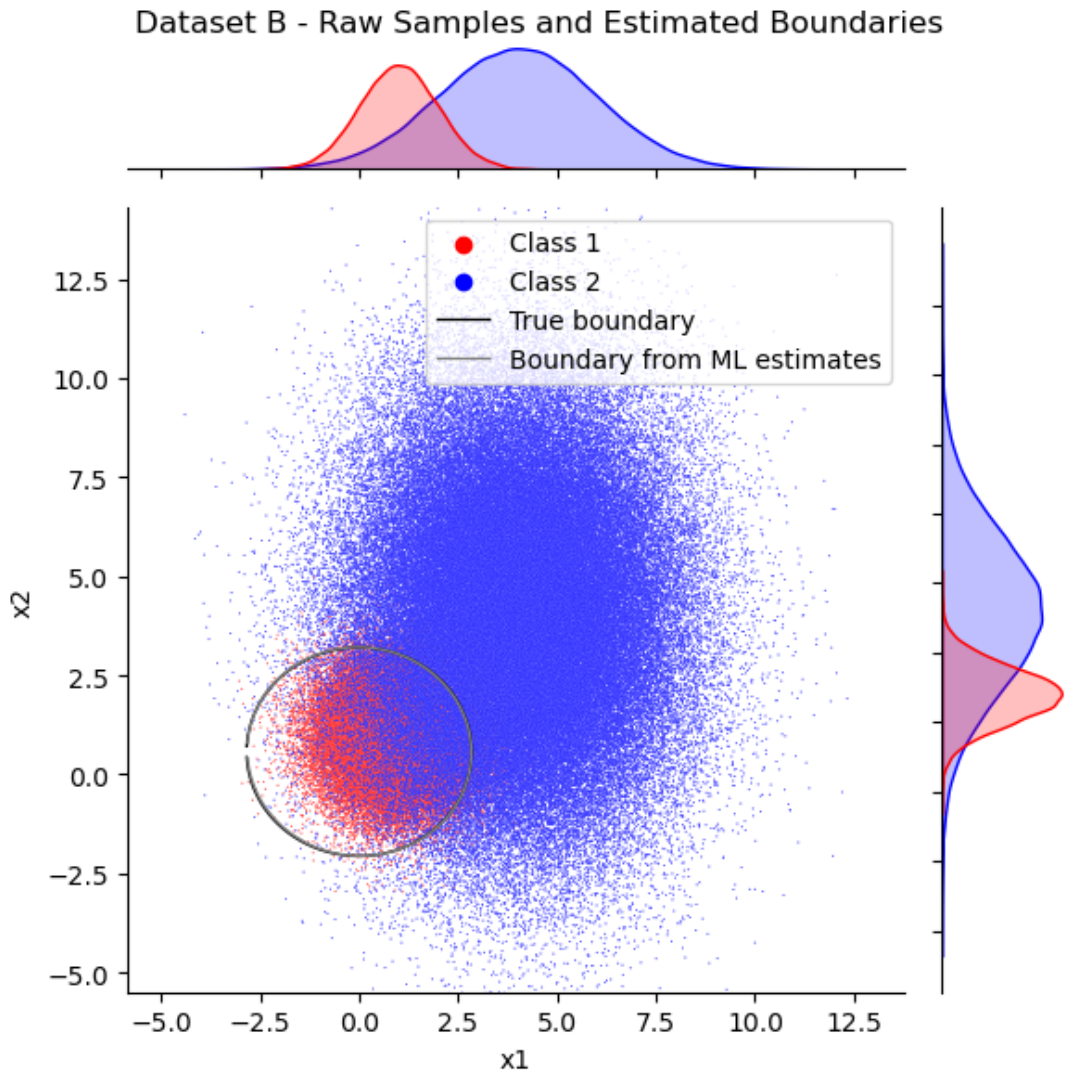
### 2.2.3 Plotting the boundaries from true and estimated parameters

[22]:
```
ax = datasetB_ML.make_plot(plot_type='seaborn', sort_type='raw', title␣
 ↪='Dataset B - Raw Samples and Estimated Boundaries', show_boundary=False)
# add true boundary from given params
ax.ax_joint.plot(np.array(datasetB.df.x1.sort_values()), np.array(datasetB.
 ↪boundary_function[0](datasetB_ML.df.x1.sort_values())), lw=1,color='k',␣
 ↪label='True boundary')
ax.ax_joint.plot(np.array(datasetB.df.x1.sort_values()), np.array(datasetB.
 ↪boundary_function[1](datasetB_ML.df.x1.sort_values())), lw=1,color='k')


#add ML estimated boundary (no assumptions/rounding for estimated covariance␣
 ↪matrix)
```

```
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='grey', label='Boundary from ML estimates')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='grey')



ax.ax_joint.legend(loc='upper right')
```

```
/Volumes/MacBackup/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[22]: <matplotlib.legend.Legend at 0x7fa33298c850>




## 2.3 Part (b)

**2.3.1 Next, you will test how the number of training data affects parameter estimation and consequently, classification accuracy. For this, consider using only (i) 0.01%, (ii) 0.1%, (iii) 1%, and (vi) 10% of the samples from each density (randomly chosen) to estimate the parameters of the two densities using ML. Then, classify all 200,000 samples for each case, count the number of misclassified samples (for each class and overall), and compare your results with those obtained in (1.a).**

### 2.3.2 (i) 0.01%

[23]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True, training_size=0.0001)
ML_estimated_boundary_001 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6
Number of samples used from class 2 : 14

 ML Estimated Means:

Sample 1 : [0.8123355251448224, 0.6542570359524086]
Sample 2 : [4.601050505327105, 3.5682360579764194]

 ML Estimated Covariances:

Sample 1 : [[1.26582959 0.13102945]
 [0.13102945 0.06413848]]
Sample 2 : [[2.17689785 1.26254571]
 [1.26254571 5.54766826]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [32144, 3674]
Misclassification rate by class: [0.53573333 0.02624286]
Total misclassification rate: 0.17909

 ********************************************************************
```

### 2.3.3 (ii) 0.1%

[24]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True, training_size=0.001)
ML_estimated_boundary_01 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60
Number of samples used from class 2 : 140

 ML Estimated Means:

Sample 1 : [0.8225988179488187, 0.9172535695041794]
Sample 2 : [4.111004869204315, 4.281641212609684]

 ML Estimated Covariances:

Sample 1 : [[1.23085745 0.22173152]
 [0.22173152 0.7378921 ]]
Sample 2 : [[3.36227842 0.08119872]
 [0.08119872 8.07362643]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [6095, 9505]
Misclassification rate by class: [0.10158333 0.06789286]
Total misclassification rate: 0.078

 ****************************************************************
```

#### 2.3.4 (iii) 1%

[25]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True, training_size=0.01)
ML_estimated_boundary_1 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 600
Number of samples used from class 2 : 1400
```

```
 ML Estimated Means:

Sample 1 : [0.9503036800748929, 0.9244012004219841]
Sample 2 : [4.006193528306562, 4.061081393252568]

 ML Estimated Covariances:

Sample 1 : [[1.02624498 0.04957027]
 [0.04957027 0.99147816]]
Sample 2 : [[3.98150624 0.12362808]
 [0.12362808 8.35474819]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [5257, 9864]
Misclassification rate by class: [0.08761667 0.07045714]
Total misclassification rate: 0.075605

 ********************************************************************
```

#### 2.3.5 (vi) 10%

[26]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True, training_size=0.1)
ML_estimated_boundary_10 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6000
Number of samples used from class 2 : 14000

 ML Estimated Means:

Sample 1 : [0.9870200000375329, 0.9800307852674556]
Sample 2 : [3.972538817755585, 4.025898493021303]

 ML Estimated Covariances:
```

```
Sample 1 : [[ 0.9930342  -0.01515895]
 [-0.01515895  1.01335585]]
Sample 2 : [[ 4.00980299e+00 -1.45803576e-03]
 [-1.45803576e-03  8.07880317e+00]]

 **********************************************************************


 **********************************************************************


 Empirical Error Stats:

N samples misclassified: [4999, 10087]
Misclassification rate by class: [0.08331667 0.07205   ]
Total misclassification rate: 0.07543

 **********************************************************************
```

#### 2.3.6 (v) 100%

[27]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True, training_size='all')
ML_estimated_boundary_all = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 **********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60000
Number of samples used from class 2 : 140000

 ML Estimated Means:

Sample 1 : [0.9947335309849321, 0.9993340872355969]
Sample 2 : [3.990379780987892, 3.99717090754034]

 ML Estimated Covariances:

Sample 1 : [[ 0.99284676 -0.00905706]
 [-0.00905706  1.00031409]]
Sample 2 : [[ 3.98866766 -0.00874819]
 [-0.00874819  7.97165084]]

 **********************************************************************
```

```
 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [4936, 10143]
Misclassification rate by class: [0.08226667 0.07245   ]
Total misclassification rate: 0.075395

 ****************************************************************
```

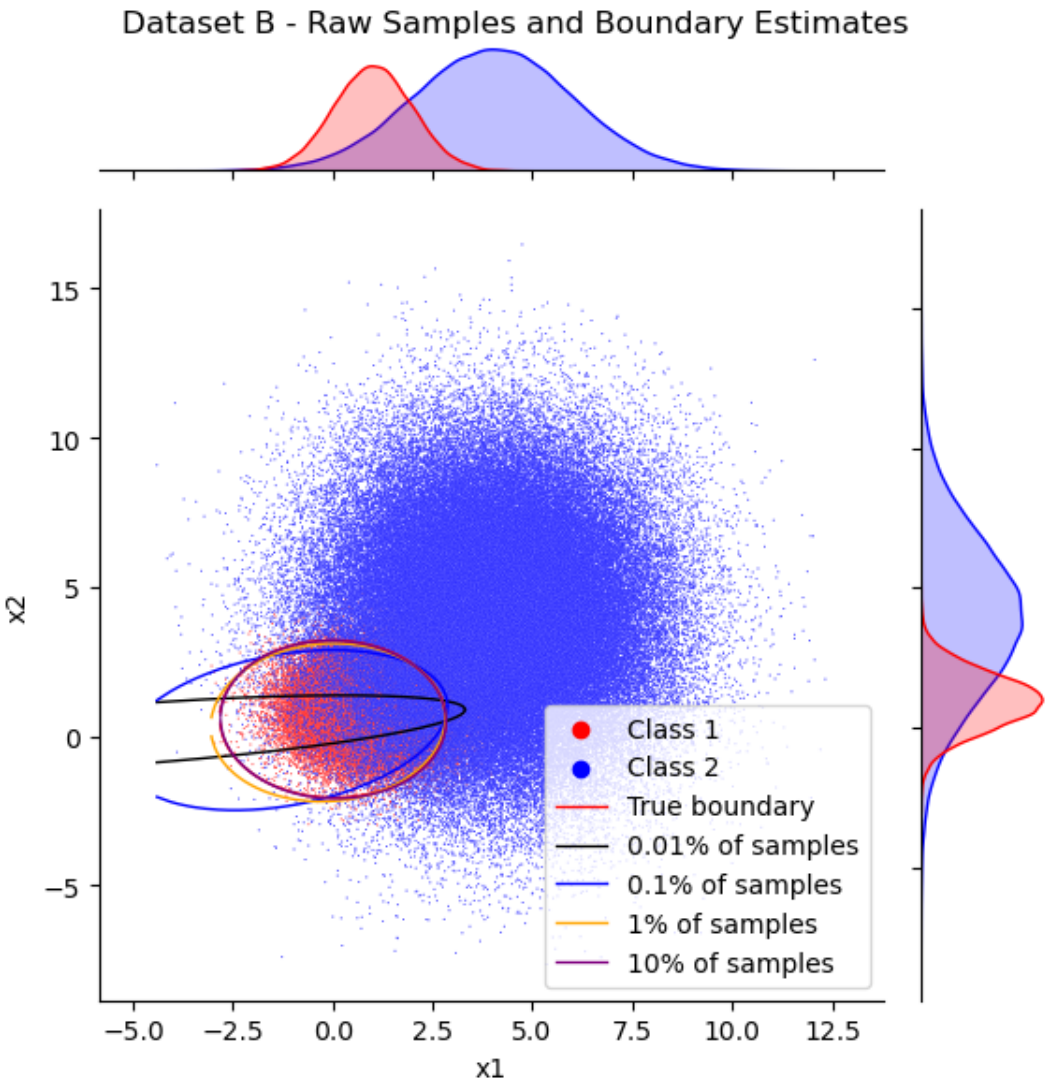
### 2.3.7 Plotting the boundaries using each set of estimated parameters

[28]:
```
ax = datasetB_ML.make_plot(plot_type='seaborn', sort_type='raw', title='Dataset␣
 ↪B - Raw Samples and Boundary Estimates', show_boundary=False)


# add true boundary from given params
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_all[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red', label='True boundary')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_all[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red',)


#add ML estimated boundary (no assumptions/rounding)
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='k', label='0.01% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='k')


#add ML estimated boundary (assume no correlation)
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue',label='0.1% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue')
```

```
#add ML estimated boundary (no correlation, rounded values)
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_1[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='orange', label='1% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_1[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='orange')

#add ML estimated boundary (no correlation, rounded values)
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_10[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='purple', label='10% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_10[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='purple')



ax.ax_joint.set_aspect('auto')
ax.ax_joint.legend(loc='lower right')
```

```
/Volumes/MacBackup/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[28]: <matplotlib.legend.Legend at 0x7fa32f8d0190>




### 2.4 sub-analysis:

#### 2.4.1 Experiment with (a) choosing the optimum Case number based on the estimated covariance matrices (since we do not really know the true covariance matrices in practice) and (b) choosing the optimum Case number after explicitly setting the off-diagonal elements of the covariance matrices to zero by assuming that the features are uncorrelated. Carefully analyze your results.

#### 2.4.2 choosing the optimum Case number after explicitly setting the off-diagonal elements of the covariance matrices to zero

### 2.4.3 (i) using 0.01% of data

[29]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
 ↪round_estimates=True, training_size=0.0001)
ML_estimated_boundary_001 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6
Number of samples used from class 2 : 14

 ML Estimated Means:

Sample 1 : [0.8 0.7]
Sample 2 : [4.6 3.6]

 ML Estimated Covariances:

Sample 1 : [[1.3 0. ]
 [0.  0.1]]
Sample 2 : [[2.2 0. ]
 [0.  5.5]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [21691, 5425]
Misclassification rate by class: [0.36151667 0.03875   ]
Total misclassification rate: 0.13558

 ********************************************************************
```

### 2.4.4 (ii) using 0.1% of data

[30]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
↪round_estimates=True, training_size=0.001)
ML_estimated_boundary_01 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60
Number of samples used from class 2 : 140

 ML Estimated Means:

Sample 1 : [0.8 0.9]
Sample 2 : [4.1 4.3]

 ML Estimated Covariances:

Sample 1 : [[1.2 0. ]
 [0.  0.7]]
Sample 2 : [[3.4 0. ]
 [0.  8.1]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [6004, 9379]
Misclassification rate by class: [0.10006667 0.06699286]
Total misclassification rate: 0.076915

 ****************************************************************
```

### 2.4.5 (iii) using 1% of data

[31]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
↪round_estimates=True, training_size=0.01)
ML_estimated_boundary_1 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 600
Number of samples used from class 2 : 1400

 ML Estimated Means:

Sample 1 : [1.  0.9]
Sample 2 : [4.  4.1]

 ML Estimated Covariances:

Sample 1 : [[1. 0.]
 [0. 1.]]
Sample 2 : [[4.  0. ]
 [0.  8.4]]

 ****************************************************************


 ****************************************************************


 Empirical Error Stats:

N samples misclassified: [5033, 10140]
Misclassification rate by class: [0.08388333 0.07242857]
Total misclassification rate: 0.075865

 ****************************************************************
```

### 2.4.6 (vi) using 10% of data

[32]:
```
datasetB_ML.classify_using_ML_estimates(print_params=True,corr=False,␣
 ↪round_estimates=True, training_size=0.1)
ML_estimated_boundary_10 = datasetB_ML.boundary_function
datasetB_ML.get_true_error(print_params=True)
```

```
 ****************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 6000
Number of samples used from class 2 : 14000
```

```
 ML Estimated Means:

Sample 1 : [1. 1.]
Sample 2 : [4. 4.]

 ML Estimated Covariances:

Sample 1 : [[ 1. -0.]
 [-0.  1.]]
Sample 2 : [[ 4.  -0. ]
 [-0.   8.1]]

 ********************************************************************


 ********************************************************************


 Empirical Error Stats:

N samples misclassified: [4890, 10215]
Misclassification rate by class: [0.0815     0.07296429]
Total misclassification rate: 0.075525

 ********************************************************************
```

### 2.4.7 (v) using 100% of data

[33]:
```
## classify using the ML estimated mu and cov params; rounding the estimates␣
 ↪and assuming no correlation between features
datasetB_ML.classify_using_ML_estimates(corr=False, round_estimates=True,␣
 ↪print_params=True)
datasetB_ML.get_true_error(print_params=True)
ML_estimated_boundary_rounded = datasetB_ML.boundary_function
```

```
 ********************************************************************

ML Param Estimates of Randomly Generated Samples:

Number of samples used from class 1 : 60000
Number of samples used from class 2 : 140000

 ML Estimated Means:

Sample 1 : [1. 1.]
```

```
Sample 2 : [4. 4.]

 ML Estimated Covariances:

Sample 1 : [[ 1. -0.]
 [-0.  1.]]
Sample 2 : [[ 4. -0.]
 [-0.  8.]]

 ***********************************************************************


 ***********************************************************************


 Empirical Error Stats:

N samples misclassified: [4885, 10206]
Misclassification rate by class: [0.08141667 0.0729    ]
Total misclassification rate: 0.075455

 ***********************************************************************
```

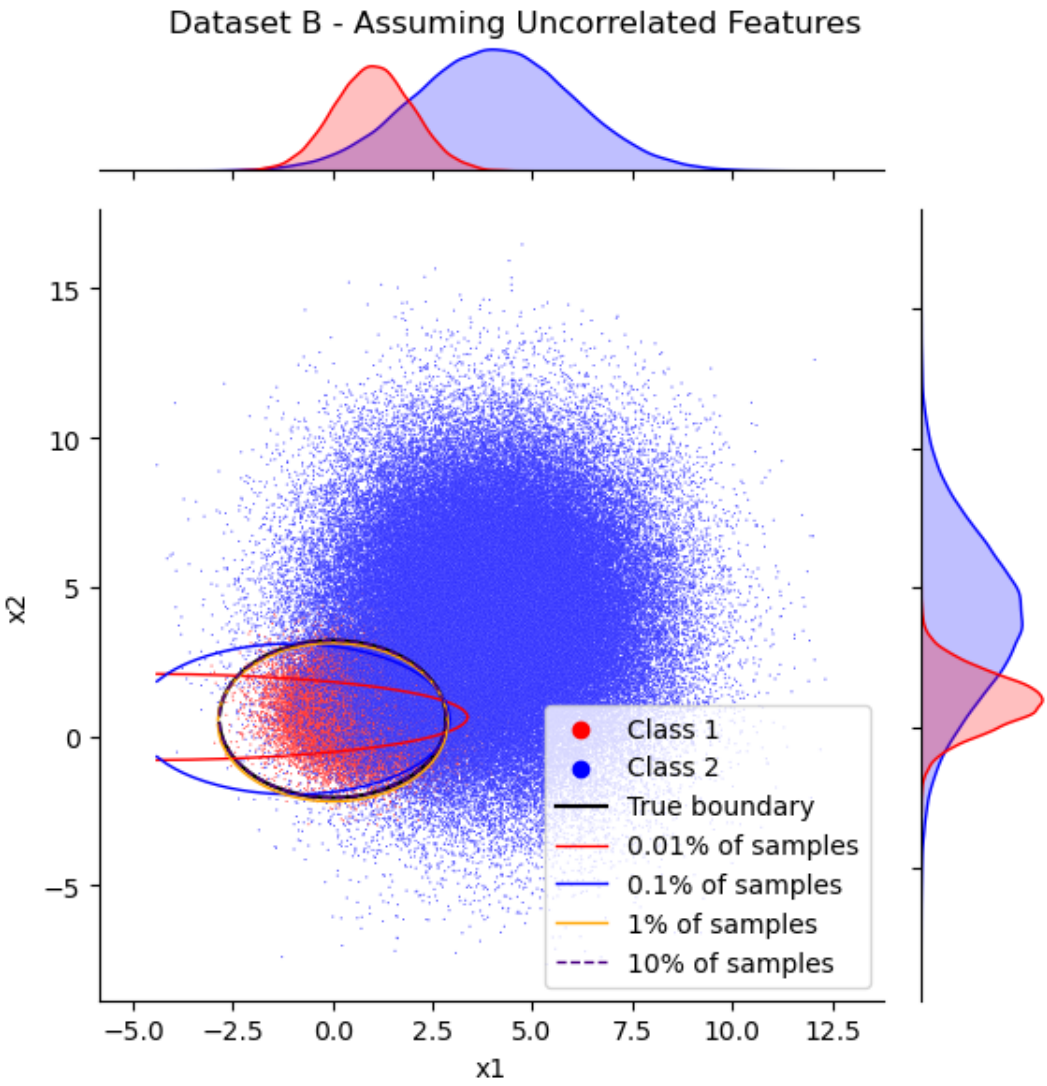
### 2.4.8 Plotting the boundaries using each set of estimated parameters

[34]:
```
ax = datasetB_ML.make_plot(plot_type='seaborn', sort_type='raw', title='Dataset␣
 ↪B - Assuming Uncorrelated Features', show_boundary=False)

#100%
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_rounded[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1.5,color='k', label='True boundary')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_rounded[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1.5,color='k',)




#0.01%
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red', label='0.01% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_001[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='red')
```

```
#.1%
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue',label='0.1% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_01[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='blue')

#1%
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_1[0](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='orange', label='1% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_1[1](datasetB_ML.df.x1.sort_values())),␣
 ↪lw=1,color='orange')

#10%
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_10[0](datasetB_ML.df.x1.sort_values())),'--',␣
 ↪lw=1, color='indigo', label='10% of samples')
ax.ax_joint.plot(np.array(datasetB_ML.df.x1.sort_values()), np.
 ↪array(ML_estimated_boundary_10[1](datasetB_ML.df.x1.sort_values())),'--',␣
 ↪lw=1, color='indigo')




ax.ax_joint.set_aspect('auto')
ax.ax_joint.legend(loc='lower right')
```

```
/Volumes/MacBackup/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[34]: <matplotlib.legend.Legend at 0x7fa30e7ff880>




[ ]:

## 3 Experiment 3

**3.1** **Face detection using skin color is a popular approach. While color images are typically in RGB format, most techniques transform RGB to a different color space (e.g., chromatic, HSV, etc.). This is because RGB values are more sensitive to variations in brightness due to illumination changes.**

**3.2** **(a) Implement the skin-color methodology of [Yang96 "A Real-time Face Tracker"] which uses the chromatic color space**

### 3.2.1 Use 'Training_1.ppm' to build the skin color model

[35]:
```
img1 = cv2.imread('Data_Prog2/Training_1.ppm')
img2 = cv2.imread('Data_Prog2/ref1.ppm')
```

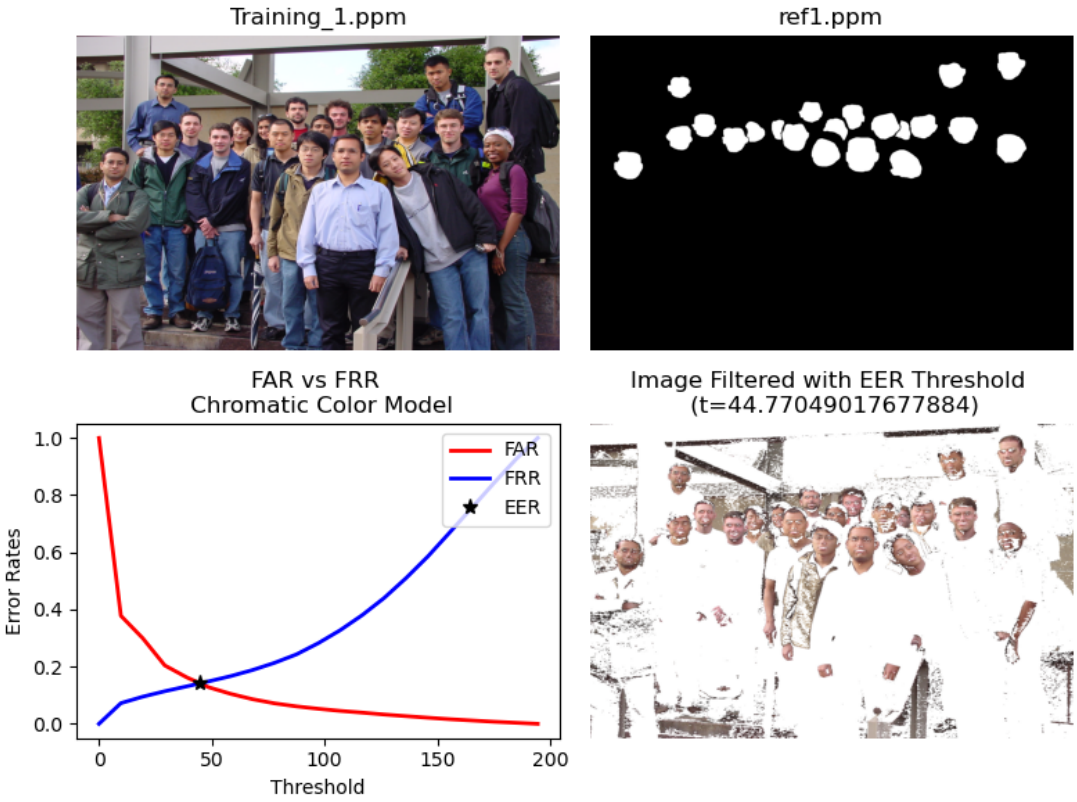
[36]:
```
experiment3_training_rg = IC.ImageClassifier()
experiment3_training_rg.classify_image(training_img = img1,ref_img=img2,␣
 ↪color_space='chromatic',analysis_type='train',)
```

[37]:
```
print('mu_hat:', experiment3_training_rg.mu)
print('cov_hat', experiment3_training_rg.cov)
```

```
mu_hat: [[0.43222356 0.29577095]]
cov_hat [array([[ 0.00243324, -0.00111725],
       [-0.00111725,  0.00078844]])]
```

[38]:
```
fig, ax=  plt.subplots(ncols=2, nrows=2, figsize=(8,6),layout='tight')
ax[0,0].imshow(experiment3_training_rg.training_img, aspect="auto")
ax[0,1].imshow(experiment3_training_rg.ref_img, aspect="auto")
ax[0,0].axis('off')
ax[0,1].axis('off')
ax[0,0].set_title('Training_1.ppm')
ax[0,1].set_title('ref1.ppm')


ax[1,0].plot(experiment3_training_rg.ROC_df.t,experiment3_training_rg.ROC_df.
 ↪FPR, color='red',lw=2,label='FAR')
ax[1,0].plot(experiment3_training_rg.ROC_df.t,experiment3_training_rg.ROC_df.
 ↪FRR, color='blue',lw=2, label='FRR')
ax[1,0].plot(experiment3_training_rg.EER_thresh, experiment3_training_rg.
 ↪EER_value,'*', color='k',markersize=8, label='EER')
ax[1,0].set_title('FAR vs FRR \n Chromatic Color Model')
ax[1,0].legend(loc='upper right')
ax[1,0].set_xlabel('Threshold')
ax[1,0].set_ylabel('Error Rates')
ax[1,1].imshow(experiment3_training_rg.EER_filtered_image, aspect="auto")
ax[1,1].set_title('Image Filtered with EER Threshold \n (t=%s)' %␣
 ↪experiment3_training_rg.EER_thresh[0])
ax[1,1].axis('off')
```

[38]:
```
(-0.5, 1855.5, 1391.5, -0.5)
```




### 3.2.2 Test the model on training image 3

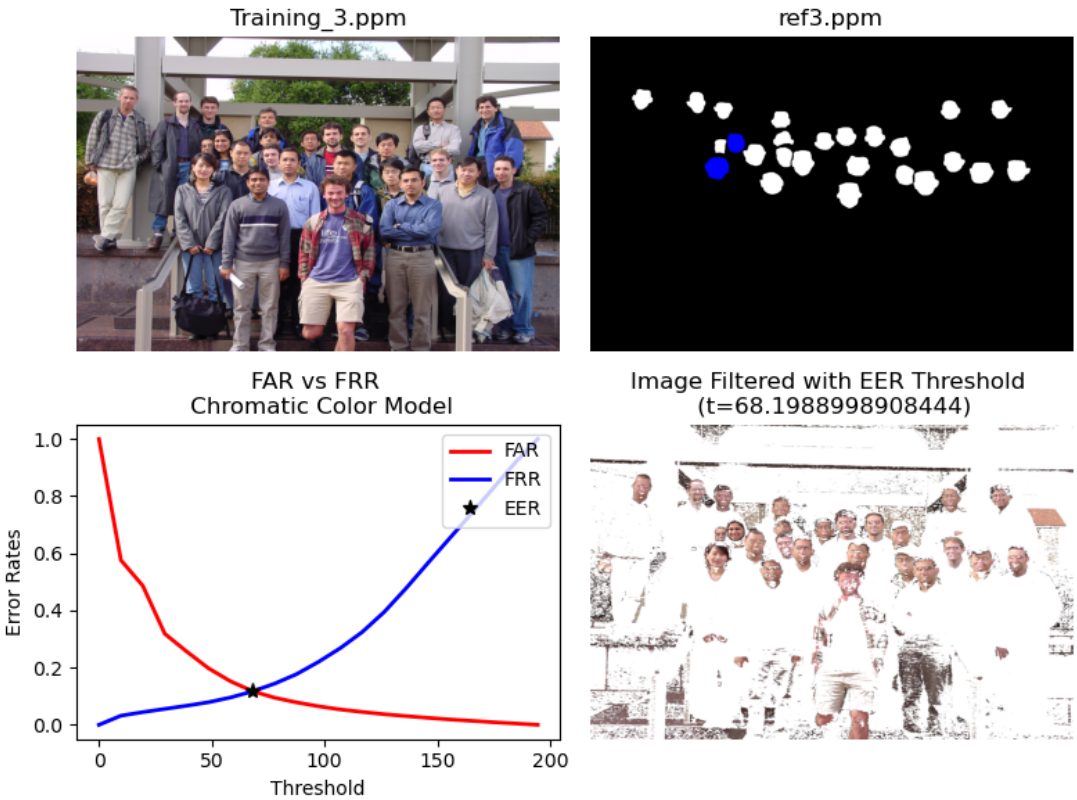
[39]:
```
img3 = cv2.imread('Data_Prog2/Training_3.ppm')
img4 = cv2.imread('Data_Prog2/ref3.ppm')
```

[40]:
```
experiment3_test1_rg = IC.ImageClassifier()
experiment3_test1_rg.classify_image(training_img = img3,ref_img=img4,␣
↪params=[experiment3_training_rg.mu,experiment3_training_rg.cov],␣
↪color_space='chromatic', analysis_type='test')
```

[41]:
```
fig, ax=  plt.subplots(ncols=2, nrows=2, figsize=(8,6),layout='tight')
ax[0,0].imshow(experiment3_test1_rg.training_img, aspect="auto")
ax[0,1].imshow(experiment3_test1_rg.ref_img, aspect="auto")
ax[0,0].axis('off')
ax[0,1].axis('off')
ax[0,0].set_title('Training_3.ppm')
ax[0,1].set_title('ref3.ppm')
```

```
ax[1,0].plot(experiment3_test1_rg.ROC_df.t,experiment3_test1_rg.ROC_df.FPR,␣
 ↪color='red',lw=2,label='FAR')
ax[1,0].plot(experiment3_test1_rg.ROC_df.t,experiment3_test1_rg.ROC_df.FRR,␣
 ↪color='blue',lw=2, label='FRR')
ax[1,0].plot(experiment3_test1_rg.EER_thresh, experiment3_test1_rg.
 ↪EER_value,'*', color='k',markersize=8, label='EER')
ax[1,0].set_title('FAR vs FRR \n Chromatic Color Model')
ax[1,0].legend(loc='upper right')
ax[1,0].set_xlabel('Threshold')
ax[1,0].set_ylabel('Error Rates')
ax[1,1].imshow(experiment3_test1_rg.EER_filtered_image, aspect="auto")
ax[1,1].set_title('Image Filtered with EER Threshold \n (t=%s)' %␣
 ↪experiment3_test1_rg.EER_thresh[0])
ax[1,1].axis('off')
```

[41]: (-0.5, 1855.5, 1391.5, -0.5)



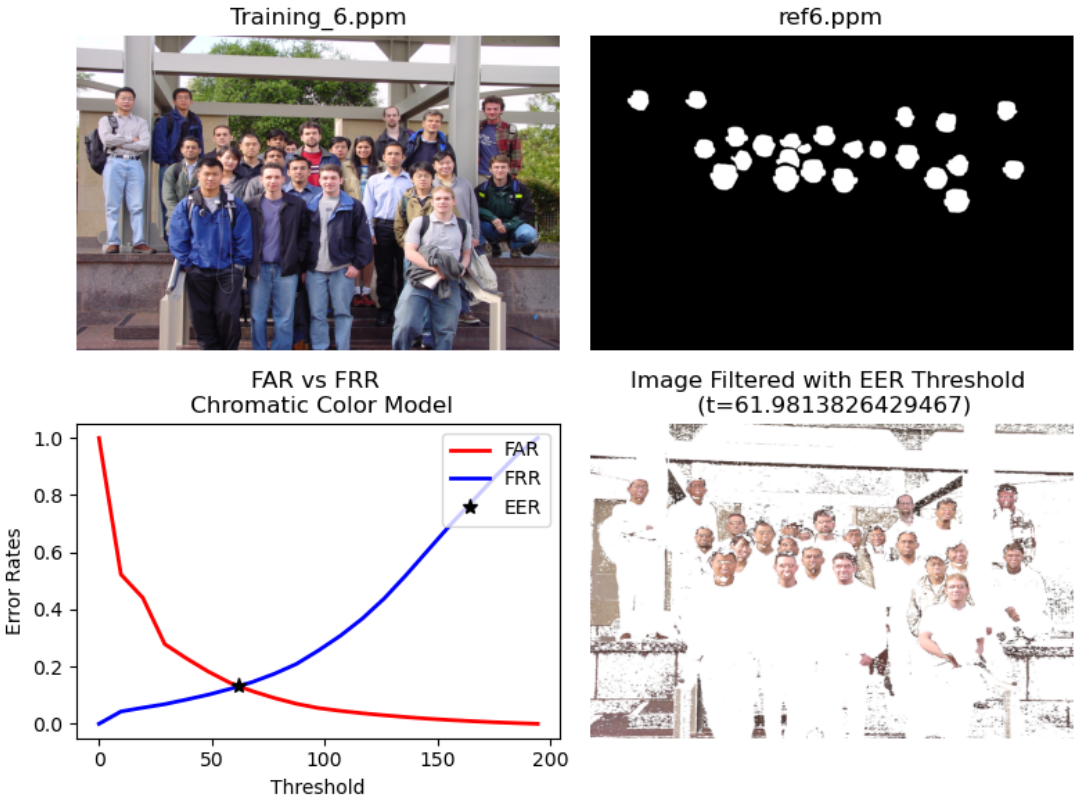
### 3.2.3 Test the model on training image 6

[42]:
```
img5 = cv2.imread('Data_Prog2/Training_6.ppm')
img6 = cv2.imread('Data_Prog2/ref6.ppm')
```

[43]:
```
experiment3_test2_rg = IC.ImageClassifier()
experiment3_test2_rg.classify_image(training_img = img5,ref_img=img6,␣
 ↪params=[experiment3_training_rg.mu,experiment3_training_rg.cov],␣
 ↪color_space='chromatic', analysis_type='test')
```

[44]:
```
fig, ax=  plt.subplots(ncols=2, nrows=2, figsize=(8,6),layout='tight')
ax[0,0].imshow(experiment3_test2_rg.training_img, aspect="auto")
ax[0,1].imshow(experiment3_test2_rg.ref_img, aspect="auto")
ax[0,0].axis('off')
ax[0,1].axis('off')
ax[0,0].set_title('Training_6.ppm')
ax[0,1].set_title('ref6.ppm')


ax[1,0].plot(experiment3_test2_rg.ROC_df.t,experiment3_test2_rg.ROC_df.FPR,␣
 ↪color='red',lw=2,label='FAR')
ax[1,0].plot(experiment3_test2_rg.ROC_df.t,experiment3_test2_rg.ROC_df.FRR,␣
 ↪color='blue',lw=2, label='FRR')
ax[1,0].plot(experiment3_test2_rg.EER_thresh, experiment3_test2_rg.
 ↪EER_value,'*', color='k',markersize=8, label='EER')
ax[1,0].set_title('FAR vs FRR \n Chromatic Color Model')
ax[1,0].legend(loc='upper right')
ax[1,0].set_xlabel('Threshold')
ax[1,0].set_ylabel('Error Rates')
ax[1,1].imshow(experiment3_test2_rg.EER_filtered_image, aspect="auto")
ax[1,1].set_title('Image Filtered with EER Threshold \n (t=%s)' %␣
 ↪experiment3_test2_rg.EER_thresh[0])
ax[1,1].axis('off')
```

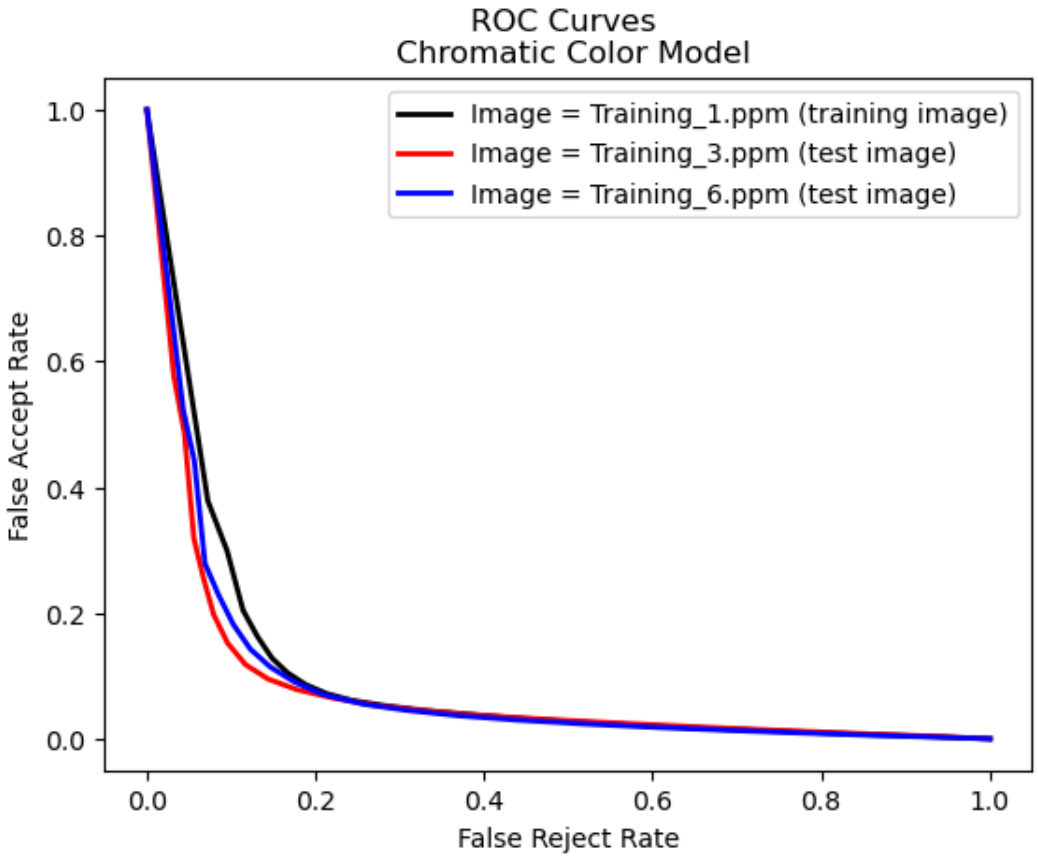
[44]:
```
(-0.5, 1855.5, 1391.5, -0.5)
```




### 3.2.4 generate ROC plots (i.e., FPR in the x-axis vs FNR in the y-axis) by varying the threshold t. To generate a reasonably smooth ROC curve, select 20 different thresholds in the interval [0, c] (i.e., uniformly distributed using a step=c/20) where c is the normalizing factor of the Gaussian function (i.e., c=1/2 |Σ|1/2) which is the max value achieved by g(x) when =0).

[45]:
```
plt.plot(experiment3_training_rg.ROC_df.FRR,experiment3_training_rg.ROC_df.
 ↪FPR,color='k', lw=2, label='Image = Training_1.ppm (training image)')
plt.plot(experiment3_test1_rg.ROC_df.FRR,experiment3_test1_rg.ROC_df.
 ↪FPR,color='red', lw=2, label='Image = Training_3.ppm (test image)')
plt.plot(experiment3_test2_rg.ROC_df.FRR,experiment3_test2_rg.ROC_df.
 ↪FPR,color='blue', lw=2, label='Image = Training_6.ppm (test image)')
plt.ylabel('False Accept Rate')
plt.xlabel('False Reject Rate')
plt.title('ROC Curves \n Chromatic Color Model')
plt.legend(loc='upper right')
```

[45]:
```
<matplotlib.legend.Legend at 0x7fa2bdc403a0>
```




### 3.3 (b) In this experiment, you will investigate the effect of using different features for classification. For this, you would need to repeat (3.a) using the YCbCr color space. The RGB components can be converted to the YCbCr components using the given transformation.

#### 3.3.1 Use 'Training_1.ppm' to build the skin color model

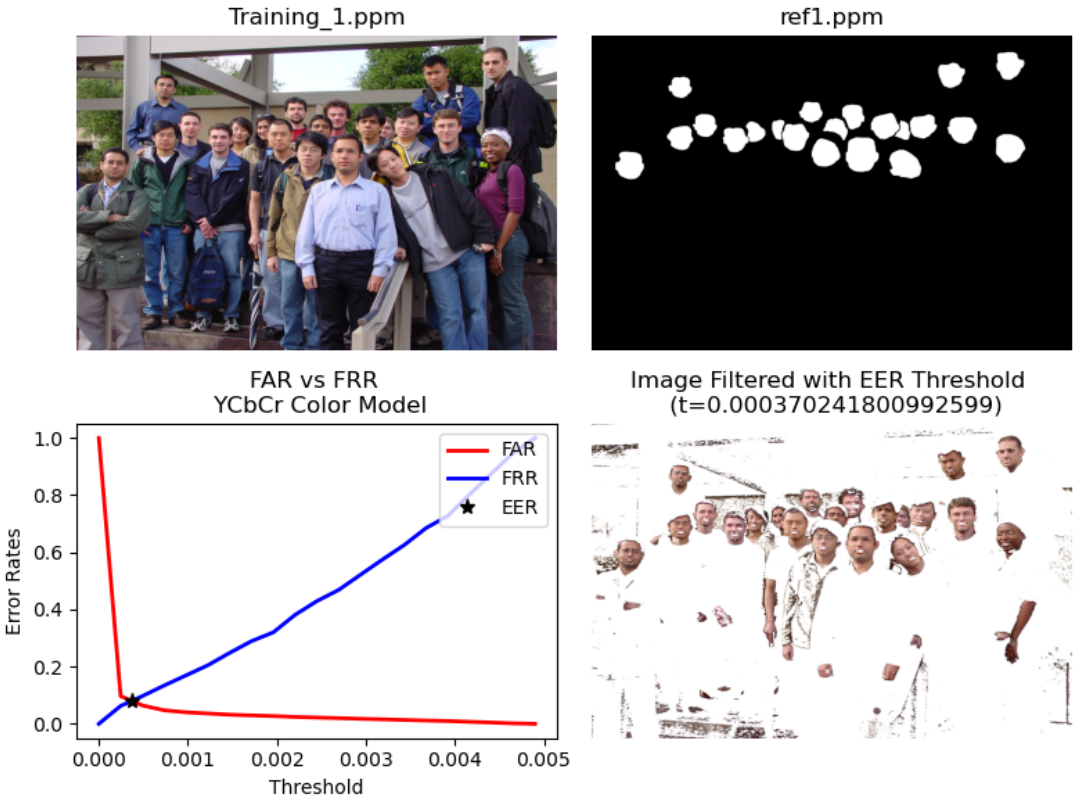
[46]:
```
experiment3_training_YCbCr = IC.ImageClassifier()
experiment3_training_YCbCr.classify_image(training_img = img1,ref_img=img2,␣
↪color_space='YCbCr',analysis_type='train')
```

[47]:
```
print('mu_hat:', experiment3_training_YCbCr.mu)
print('cov_hat', experiment3_training_YCbCr.cov)
```

```
mu_hat: [[-12.69134026  23.63426875]]
cov_hat [array([[ 31.19704223, -21.74165815],
       [-21.74165815,  48.94555422]])]
```

[48]:
```
fig, ax=  plt.subplots(ncols=2, nrows=2, figsize=(8,6),layout='tight')
ax[0,0].imshow(experiment3_training_YCbCr.training_img, aspect="auto")
ax[0,1].imshow(experiment3_training_YCbCr.ref_img, aspect="auto")
ax[0,0].axis('off')
ax[0,1].axis('off')
ax[0,0].set_title('Training_1.ppm')
ax[0,1].set_title('ref1.ppm')


ax[1,0].plot(experiment3_training_YCbCr.ROC_df.t,experiment3_training_YCbCr.
 ↪ROC_df.FPR, color='red',lw=2,label='FAR')
ax[1,0].plot(experiment3_training_YCbCr.ROC_df.t,experiment3_training_YCbCr.
 ↪ROC_df.FRR, color='blue',lw=2, label='FRR')
ax[1,0].plot(experiment3_training_YCbCr.EER_thresh, experiment3_training_YCbCr.
 ↪EER_value,'*', color='k',markersize=8, label='EER')
ax[1,0].set_title('FAR vs FRR \n YCbCr Color Model')
ax[1,0].legend(loc='upper right')
ax[1,0].set_xlabel('Threshold')
ax[1,0].set_ylabel('Error Rates')
ax[1,1].imshow(experiment3_training_YCbCr.EER_filtered_image, aspect="auto")
ax[1,1].set_title('Image Filtered with EER Threshold \n (t=%s)' %␣
 ↪experiment3_training_YCbCr.EER_thresh[0])
ax[1,1].axis('off')
```

[48]:
```
(-0.5, 1855.5, 1391.5, -0.5)
```




### 3.3.2 Use 'Training_3.ppm' to test the skin color model

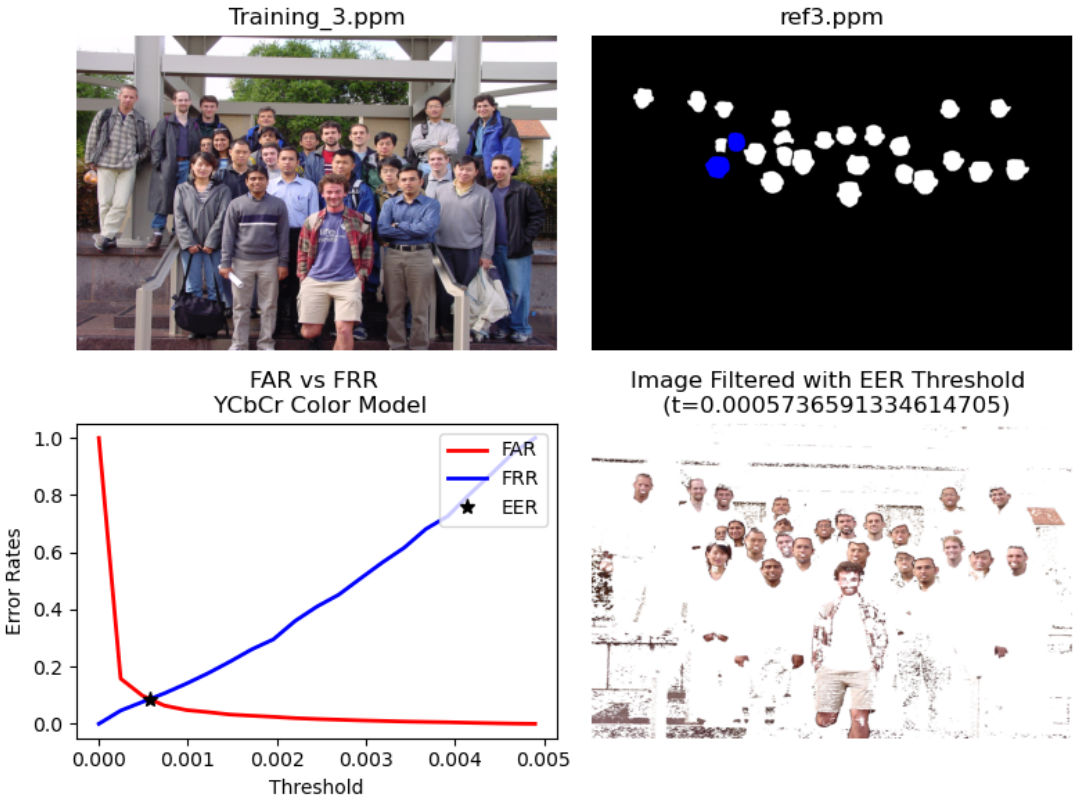
[49]:
```
experiment3_test1_YCbCr = IC.ImageClassifier()
experiment3_test1_YCbCr.classify_image(training_img = img3,ref_img=img4,␣
↪params=[experiment3_training_YCbCr.mu,experiment3_training_YCbCr.cov],␣
↪color_space='YCbCr', analysis_type='test') #x0=0.0005)
```

```
/Volumes/MacBackup/opt/anaconda3/lib/python3.8/site-
packages/scipy/optimize/_minpack_py.py:175: RuntimeWarning: The iteration is not
making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

[50]:
```
fig, ax=  plt.subplots(ncols=2, nrows=2, figsize=(8,6),layout='tight')
ax[0,0].imshow(experiment3_test1_YCbCr.training_img, aspect="auto")
ax[0,1].imshow(experiment3_test1_YCbCr.ref_img, aspect="auto")
ax[0,0].axis('off')
ax[0,1].axis('off')
ax[0,0].set_title('Training_3.ppm')
ax[0,1].set_title('ref3.ppm')
```

```
ax[1,0].plot(experiment3_test1_YCbCr.ROC_df.t,experiment3_test1_YCbCr.ROC_df.
 ↪FPR, color='red',lw=2,label='FAR')
ax[1,0].plot(experiment3_test1_YCbCr.ROC_df.t,experiment3_test1_YCbCr.ROC_df.
 ↪FRR, color='blue',lw=2, label='FRR')
ax[1,0].plot(experiment3_test1_YCbCr.EER_thresh, experiment3_test1_YCbCr.
 ↪EER_value,'*', color='k',markersize=8, label='EER')
ax[1,0].set_title('FAR vs FRR \n YCbCr Color Model')
ax[1,0].legend(loc='upper right')
ax[1,0].set_xlabel('Threshold')
ax[1,0].set_ylabel('Error Rates')
ax[1,1].imshow(experiment3_test1_YCbCr.EER_filtered_image, aspect="auto")
ax[1,1].set_title('Image Filtered with EER Threshold \n (t=%s)' %␣
 ↪experiment3_test1_YCbCr.EER_thresh[0])
ax[1,1].axis('off')
```

[50]: (-0.5, 1855.5, 1391.5, -0.5)

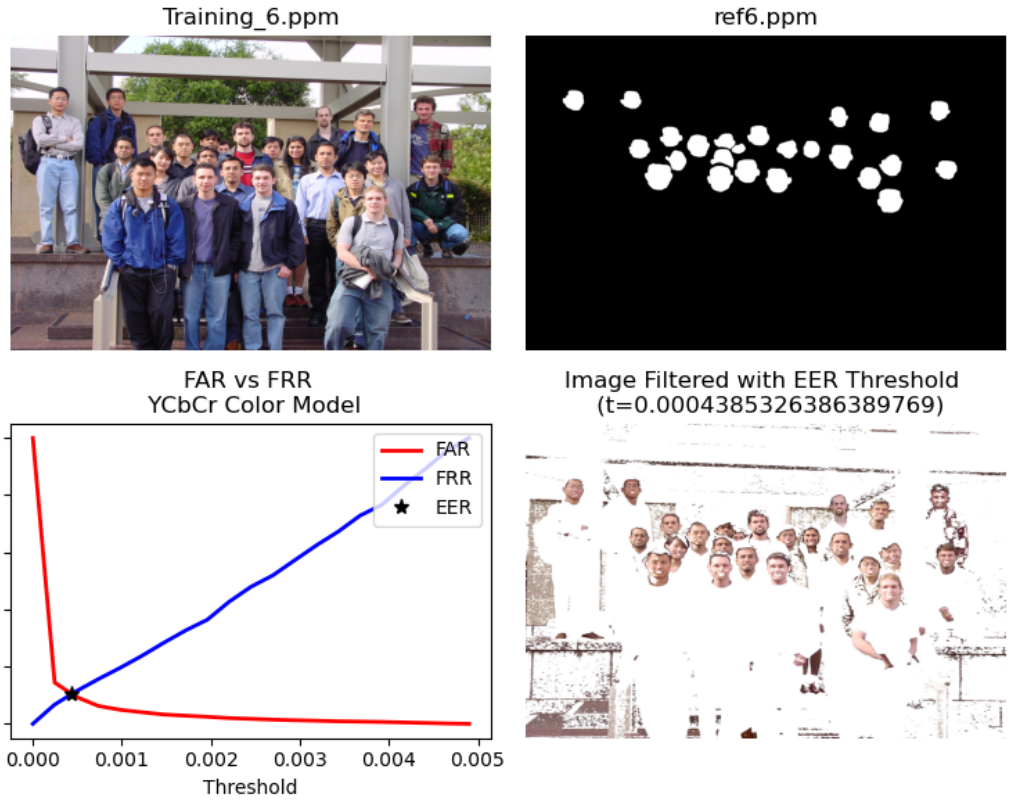


### 3.3.3 Use 'Training_6.ppm' to test the skin color model

[51]:
```
experiment3_test2_YCbCr = IC.ImageClassifier()
experiment3_test2_YCbCr.classify_image(training_img = img5,ref_img=img6,␣
↪params=[experiment3_training_YCbCr.mu,experiment3_training_YCbCr.cov],␣
↪color_space='YCbCr', analysis_type='test') # x0=0.0006)
```

[52]:
```
fig, ax=  plt.subplots(ncols=2, nrows=2, figsize=(8,6),layout='tight')
ax[0,0].imshow(experiment3_test2_YCbCr.training_img, aspect="auto")
ax[0,1].imshow(experiment3_test2_YCbCr.ref_img, aspect="auto")
ax[0,0].axis('off')
ax[0,1].axis('off')
ax[0,0].set_title('Training_6.ppm')
ax[0,1].set_title('ref6.ppm')


ax[1,0].plot(experiment3_test2_YCbCr.ROC_df.t,experiment3_test2_YCbCr.ROC_df.
↪FPR, color='red',lw=2,label='FAR')
ax[1,0].plot(experiment3_test2_YCbCr.ROC_df.t,experiment3_test2_YCbCr.ROC_df.
↪FRR, color='blue',lw=2, label='FRR')
ax[1,0].plot(experiment3_test2_YCbCr.EER_thresh, experiment3_test2_YCbCr.
↪EER_value,'*', color='k',markersize=8, label='EER')
ax[1,0].set_title('FAR vs FRR \n YCbCr Color Model')
ax[1,0].legend(loc='upper right')
ax[1,0].set_xlabel('Threshold')
ax[1,0].set_ylabel('Error Rates')
ax[1,1].imshow(experiment3_test2_YCbCr.EER_filtered_image, aspect="auto")
ax[1,1].set_title('Image Filtered with EER Threshold \n (t=%s)' %␣
↪experiment3_test2_YCbCr.EER_thresh[0])
ax[1,1].axis('off')
```

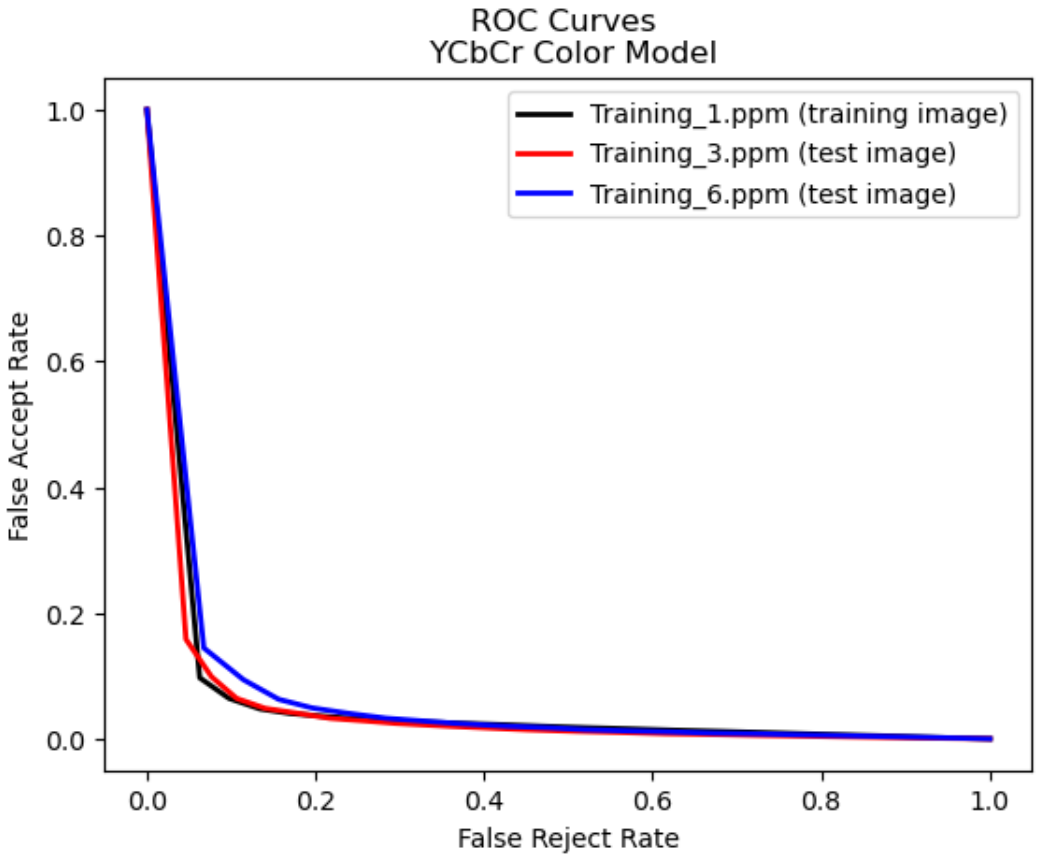
[52]:
```
(-0.5, 1855.5, 1391.5, -0.5)
```




### 3.3.4 ROC Curves

[53]:
```
plt.plot(experiment3_training_YCbCr.ROC_df.FRR,experiment3_training_YCbCr.
  ↪ROC_df.FPR,color='k', lw=2, label='Training_1.ppm (training image)')
plt.plot(experiment3_test1_YCbCr.ROC_df.FRR,experiment3_test1_YCbCr.ROC_df.
  ↪FPR,color='red', lw=2, label='Training_3.ppm (test image)')
plt.plot(experiment3_test2_YCbCr.ROC_df.FRR,experiment3_test2_YCbCr.ROC_df.
  ↪FPR,color='blue', lw=2, label='Training_6.ppm (test image)')
plt.ylabel('False Accept Rate')
plt.xlabel('False Reject Rate')
plt.title('ROC Curves \n YCbCr Color Model')
plt.legend(loc='upper right')
```

[53]:
```
<matplotlib.legend.Legend at 0x7fa27dd66a00>
```

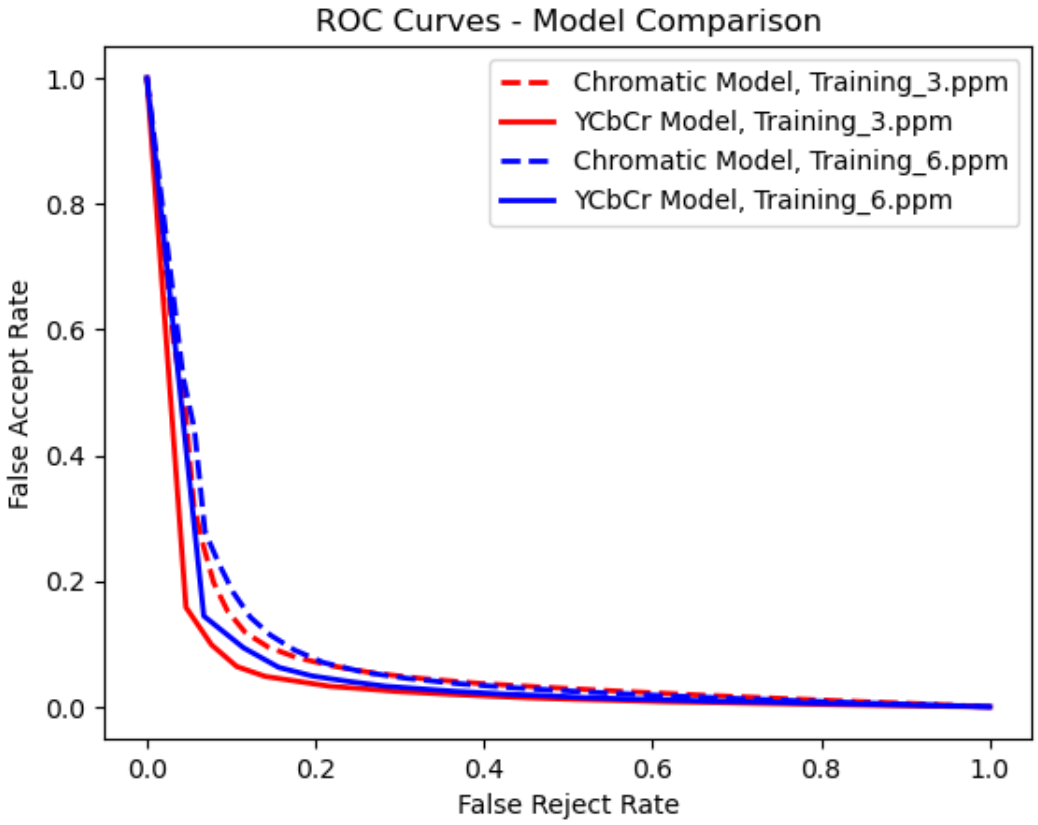



### 3.3.5 Comparing ROC Curves between models

[54]:
```
#plt.plot(experiment3_training_rg.ROC_df.FRR,experiment3_training_rg.ROC_df.
  ↪FPR,'--',color='k', lw=1, label='Chromatic Model, Training_1.ppm')
#plt.plot(experiment3_training_YCbCr.ROC_df.FRR,experiment3_training_YCbCr.
  ↪ROC_df.FPR,color='k', lw=1, label='YCbCr Model, Training_1.ppm')


plt.plot(experiment3_test1_rg.ROC_df.FRR,experiment3_test1_rg.ROC_df.
  ↪FPR,'--',color='red', lw=2, label='Chromatic Model, Training_3.ppm')
plt.plot(experiment3_test1_YCbCr.ROC_df.FRR,experiment3_test1_YCbCr.ROC_df.
  ↪FPR,color='red', lw=2, label='YCbCr Model, Training_3.ppm')


plt.plot(experiment3_test2_rg.ROC_df.FRR,experiment3_test2_rg.ROC_df.
  ↪FPR,'--',color='blue', lw=2, label='Chromatic Model, Training_6.ppm')
plt.plot(experiment3_test2_YCbCr.ROC_df.FRR,experiment3_test2_YCbCr.ROC_df.
  ↪FPR,color='blue', lw=2, label='YCbCr Model, Training_6.ppm')
```

```
plt.ylabel('False Accept Rate')
plt.xlabel('False Reject Rate')
plt.title('ROC Curves - Model Comparison')
plt.legend(loc='upper right')
```

[54]: <matplotlib.legend.Legend at 0x7fa27eeadc70>




[ ]: