Programming Assignment 2

CS 479

https://github.com/alexander-novo/CS479-PA2

Nikhil Khedekar	Mehar Mangat	Alexander Novotny
33%	33%	33%
Section 2	Section 2	Section 1

Due: April 12, 2021
Submitted: April 12, 2021

Contents

1 Parts 1 & 2 **1**
- 1.1 Theory . . . 1
  - 1.1.1 Estimating the parameters of a multivariate Gaussian distribution . . . 1
- 1.2 Implementation . . . 2
- 1.3 Results and Discussion . . . 2
  - 1.3.1 Data Set A . . . 2
  - 1.3.2 Data Set B . . . 3

2 Part 3 **4**
- 2.1 Theory . . . 4
- 2.2 Implementation . . . 6
- 2.3 Results and Discussion . . . 7

1 Parts 1 & 2

1.1 Theory

1.1.1 Estimating the parameters of a multivariate Gaussian distribution

In practice, we do not usually know the exact parameters to the distributions of features that we are observing, so we must estimate them. One intuitively good way to estimate the parameters to a random variable X given a sample $X_1, X_2, \ldots, X_n$ is to choose the parameters θ that maximize the likelihood function

$$L_n(\theta) = \prod_{i=1}^{n} f_X(X_i; \theta), \tag{1}$$

where $f_X(X_i; \theta)$ is the density function evaluated with the given parameters θ. A $\hat{\theta}$ that maximizes the likelihood function, i.e.

$$\hat{\theta} = \arg\max_{\theta} L_n(\theta) \tag{2}$$

is called the "maximum likelihood estimator". This is an intuitively good estimator, since the only information that we have is the sample we observed, so we have have evidence that the likelihood of it happening is high - especially if n is large.

The maximum likelihood estimators for a multivariate Gaussian-distributed sample $\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_n \sim \mathcal{N}(\vec{\mu}, \Sigma)$ are

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \vec{X}_i, \tag{3}$$

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (\vec{X}_i - \hat{\mu})(\vec{X}_i - \hat{\mu})^\top, \tag{4}$$

known as the sample mean and sample covariance, respectively.

Estimators, as functions of random variables, are themselves random variables. We can analyse the properties of these random variables to see, for instance, what the expected values are:

$$\begin{aligned}
\mathbb{E}[\hat{\mu}] &= \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} \vec{X}_i\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\left[\vec{X}_i\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mu \\
&= \mu, \\
\mathbb{E}[\hat{\Sigma}] &= \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} (\vec{X}_i - \hat{\mu})(\vec{X}_i - \hat{\mu})^\top\right] \\
&= \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} \left[((\vec{X}_i - \vec{\mu}) - (\hat{\mu} - \vec{\mu}))((\vec{X}_i - \vec{\mu})^\top - (\hat{\mu} - \vec{\mu})^\top)\right]\right] \\
&= \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} (\vec{X}_i - \vec{\mu})(\vec{X}_i - \vec{\mu})^\top - \left(\frac{2}{n} \sum_{i=1}^{n} (\vec{X}_i - \vec{\mu})\right)(\hat{\mu} - \vec{\mu})^\top + (\hat{\mu} - \vec{\mu})(\hat{\mu} - \vec{\mu})^\top \frac{1}{n} \sum_{i=1}^{n} 1\right]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{X}_i - \vec{\mu})(\vec{X}_i - \vec{\mu})^\top - 2(\hat{\mu} - \vec{\mu})(\hat{\mu} - \vec{\mu})^\top + (\hat{\mu} - \vec{\mu})(\hat{\mu} - \vec{\mu})^\top\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\Big[(\vec{X}_i - \vec{\mu})(\vec{X}_i - \vec{\mu})^\top\Big] - \mathbb{E}\Big[(\hat{\mu} - \vec{\mu})(\hat{\mu} - \vec{\mu})^\top\Big] \\
&= \frac{1}{n}\sum_{i=1}^{n} Var[\vec{X}_i] - Var[\hat{\mu}] \\
&= \Sigma - Var\left[\frac{1}{n}\sum_{i=1}^{n}\vec{X}_i\right] \\
&= \Sigma - \frac{1}{n^2}\sum_{i=1}^{n} Var\Big[\vec{X}_i\Big] \\
&= \Sigma - \frac{1}{n}\Sigma \\
&= \frac{n-1}{n}\Sigma.
\end{aligned}
$$

We say then, that the maximum likelihood estimator for the covariance matrix is biased, since the expected value of it is not the real parameter. We can modify the maximum likelihood estimator to make it unbiased by adding a correction:

$$
\begin{aligned}
\hat{\Sigma} &= \frac{n}{n-1}\left(\frac{1}{n}\sum_{i=1}^{n}(\vec{X}_i - \hat{\mu})(\vec{X}_i - \hat{\mu})^\top\right) \\
&= \frac{1}{n-1}\sum_{i=1}^{n}(\vec{X}_i - \hat{\mu})(\vec{X}_i - \hat{\mu})^\top. \qquad (5)
\end{aligned}
$$

We can also analyse the variance of these estimators, and of course, a smaller variance is more desirable. We will not prove it, but it turns out that these two estimators (sample mean and adjusted sample variance) are minimum variance unbiased estimators for the Gaussian distribution, that is - there are no other unbiased estimators with less variance than these, so they are good choices of estimators.

1.2 Implementation

The implementation remains largely unchanged from the previous assignment, except for the following details. After generating the full sample, a Fisher-Yates shuffle is done on the first n observations from the sample, where n is the percent of observations chosen to use for the parameters. Then the estimators given in section 1.1.1 are calculated on these first n observations. In this way, all of the observations are kept in place, but we can sample a smaller number of them for training purposes. The discriminant case is calculated on these estimators just as before but using the Eigen `isApprox()` function to tell if two matrices are approximately equal (using some precision). Case 2 was selected if all matrices were approximately equal and Case 1 was selected if the first matrix (and therefore all matrices) was additionally approximately equal to the identity matrix times its diagonal entry.

1.3 Results and Discussion

1.3.1 Data Set A

A table of relative errors of the estimators and misclassifications rates based on the percentage of data used for training can be found in table 1. Relative errors are computed against the known values, and

a smaller error means the estimator was closer to the true value. As can be seen, the relative error and misclassification rate decrease as we increase the amount of data used to train. A comparison of decision boundaries can be found in fig. 1. The decision boundary generated when using only .01% of the data is distinctly quadric in form, while the boundary generated when using 100% of the data looks very linear (despite the algorithm selecting case 3) and similar to the known decision boundary. Despite the .01% decision boundary not deviating too much from the 100% decision boundary, this account for almost double the misclassification rate.

% Of Data Used	Class 1 Mean Relative Error	Class 2 Mean Relative Error	Class 1 Covariance Relative Error	Class 2 Covariance Relative Error	Misclassification Rate
0.01%	0.343856	0.0383057	1.25869	0.591934	0.03099
0.1%	0.0924604	0.0208206	0.390639	0.265735	0.017025
1%	0.0268747	0.0126041	0.0872637	0.0774813	0.01555
10%	0.0134439	0.00197025	0.0207129	0.0206128	0.015435
100%	0.00233848	0.000514983	0.00287099	0.00541295	0.015425

Table 1: A comparison of errors classifying data set A based on how much of the generated sample was used to train the estimators. Relative error in the matrices was calculated using the Frobenius norm.

Figure 1: A comparison of decision boundaries on Data Set A based on how much of the original data was used to train the estimators.

1.3.2 Data Set B

A table of relative errors of the estimators and misclassifications rates based on the percentage of data used for training can be found in table 2. As can be seen, the relative error and misclassification rate decrease as we increase the amount of data used to train. Interestingly, the jump from 10% data to 100% actually increases misclassification rate. A comparison of decision boundaries can be found in fig. 2. The decision boundary generated when using only .01% of the data is wildly different than

the 100% boundary, which looks very similar to the known value again. This is probably due to the relatively more complex Σ_2 in Data Set B compared to Data Set A.

% Of Data Used	Class 1 Mean Relative Error	Class 2 Mean Relative Error	Class 1 Covariance Relative Error	Class 2 Covariance Relative Error	Misclassification Rate
0.01%	0.529242	0.122196	0.707237	1.18457	0.16561
0.1%	0.369923	0.0208239	0.284875	0.103571	0.08019
1%	0.0609126	0.0090789	0.11097	0.0425527	0.07328
10%	0.0227833	0.00733497	0.0315286	0.0127938	0.07304
100%	0.00649618	0.00211637	0.00698406	0.00437193	0.07319

Table 2: A comparison of errors classifying data set B based on how much of the generated sample was used to train the estimators. Relative error in the matrices was calculated using the Frobenius norm.

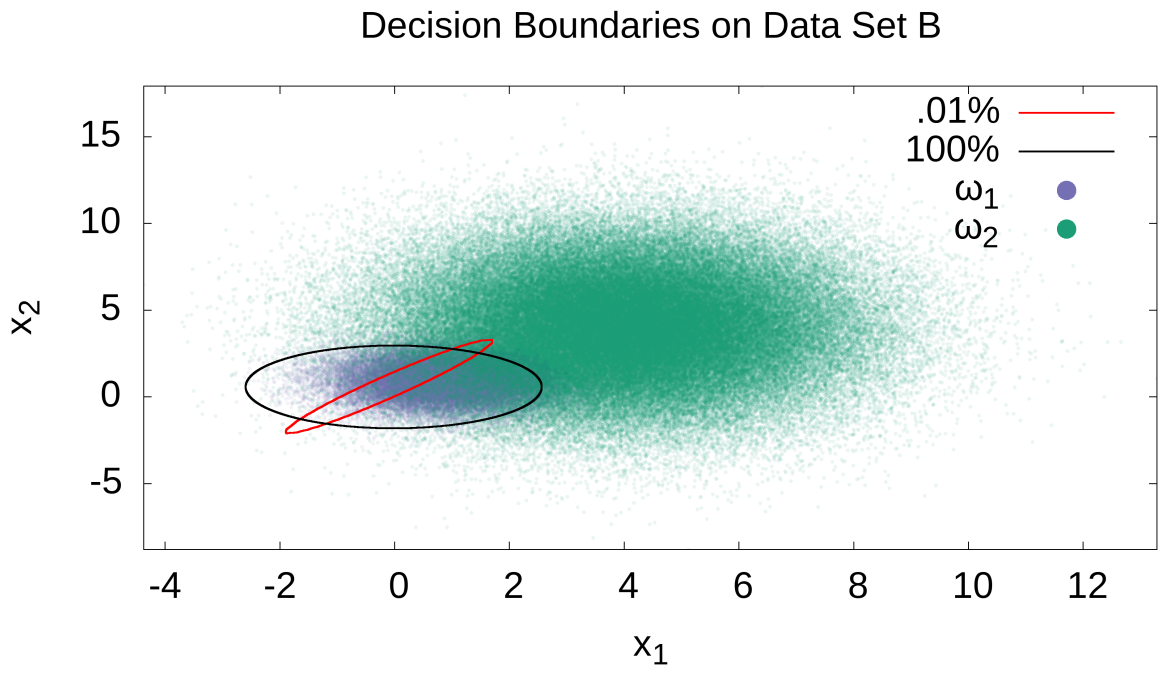

Figure 2: A comparison of decision boundaries on Data Set B based on how much of the original data was used to train the estimators.

2 Part 3

2.1 Theory

The final part of the project tackled the problem of skin color recognition. One common approach that was implemented in this section of the project was using the unique Gaussian-like nature of the space of colors given off by human skin to build a classifier. This was notably done in a paper written by Jie Yang and Alex Waibel [Yang96 "A Real-time Face Tracker"] for the purpose of building a face tracker. In the rest of this report, we use the terms skin color and facial color interchangeably.

Each individual pixel begins as a triple of RGB values. However, this isn't necessarily the best way to represent facial color information as it also encodes brightness, and is higher dimensional than what is

needed. Thus, we first convert the pixel values to the chromatic color space (normalizing the rgb data by removing brightness) via the following transformation:

$$r = \frac{R}{R+G+B} \tag{6}$$

$$g = \frac{G}{R+G+B} \tag{7}$$

The color blue is redundant as $r + g + b = 1$ due to normalization, and we have successfully removed a dimension from our data-set. The paper also notes that the skin-color distributions of different people do not significantly change between different lighting conditions or between races (in the latter case, color *intensity* varies, but not the actual colors).

We can then say that the distribution of skin-colors follows a Gaussian model, $\mathcal{N}(\mu, \Sigma)$, where μ is the sample mean, achieved by stacking $\overline{r}, \overline{g}$:

$$\begin{aligned} \overline{r} &= \frac{1}{N}\sum_{i=1}^{N} r_i, \\ \overline{g} &= \frac{1}{N}\sum_{i=1}^{N} g_i, \\ \hat{\mu} &= \begin{bmatrix} \overline{r} \\ \overline{g} \end{bmatrix}, \end{aligned} \tag{8}$$

and Σ:

$$\hat{\Sigma} = \begin{bmatrix} s_{rr}^2 & s_{rg}^2 \\ s_{gr}^2 & s_{gg}^2 \end{bmatrix}, \tag{9}$$

where s^2 is the adjusted sample covariance.

We can initialize our Gaussian parameters by taking in an image containing a face/set of faces and a corresponding image containing labels for said faces. All the pixels which identify as faces are then used to create this model via their sample mean/co-variances.

Given a sample y_i and dimensionality $n = 2$, we can then compute the likelihood of finding this random variable inside our estimated Gaussian distribution via Bayes Rule, which leads us to the pdf of a multivariate Gaussian distribution:

$$f(y_i|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp\left(-\frac{1}{2}(y_i - \mu)^\top \Sigma^{-1}(y_i - \mu)\right) \tag{10}$$

with a normalization constant

$$c = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \tag{11}$$

If this likelihood is greater than some given threshold value, then we classify that pixel as belonging to a face. The exact threshold value is a hyper-parameter, and in our case we want to set it according to an ROC curve. Higher threshold values lead to a "stricter" classifier and False Negatives increase, and lower threshold values lead to a "looser" classifier and False Positives increase. The point of intersection where FP = FN is our desired threshold value and is called the Equal Error Rate (EER).

2.2 Implementation

To begin, our code must first read/parse ppm files to act as the training set and labels for creating our Gaussian. PPM stands for "Portable Pixel Map" and while it is a fairly inconvenient and large file-type, its simplicity and lack of compression makes it a great candidate for high-accuracy programming.

The images are parsed using the popular computer vision library OpenCV. The project utilized very rudimentary functions to read/write files (`opencv::imread` and `opencv::imwrite`) and data structures to store pixel values before being converted into RGB/Chromatic color objects (`opencv::Mat`).

We can also experiment with what happens when we use the $\mathbf{YC_bC_r}$ color space, whose transformation from the RGB space is defined as:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= -0.169R - 0.332G + 0.500B \\ C_r &= 0.500R - 0.419G - 0.081B \end{aligned} \tag{12}$$

It is worth noting that since our code is vectorized by the compiler when compiled in Release mode, the building of the model and computation on the test data is completed within a few seconds.

2.3 Results and Discussion

The ROC plot using the chromatic color space can be found in fig. 3.

Figure 3: ROC curve for the chromatic color space classifier.

The EER occurs at a threshold of 135 and so we choose this as our threshold value for future classifications. The original and classified training image corresponding to this value can be seen in fig. 4.

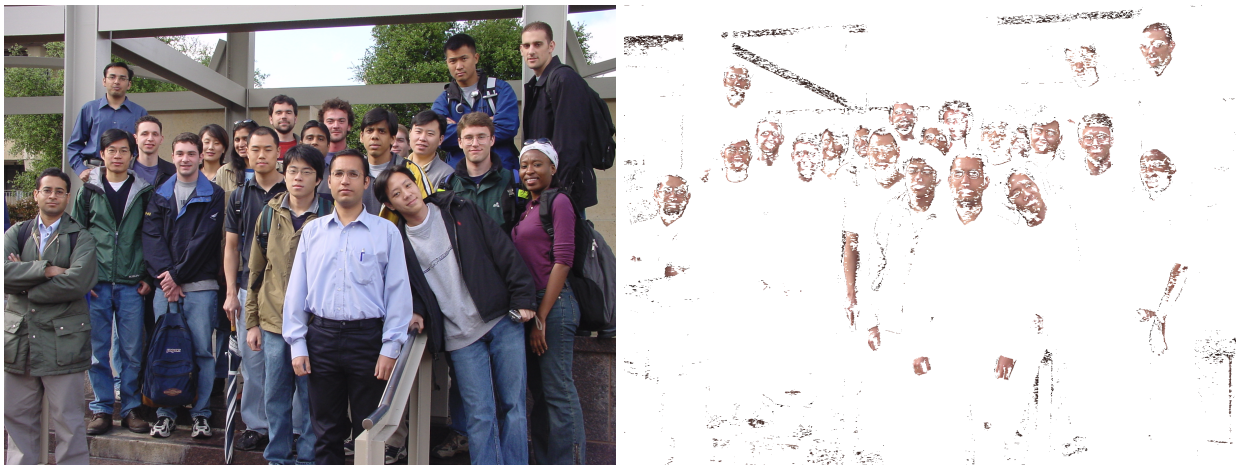

Figure 4: Training image classified in chromatic color space by the obtained threshold of 135

And its performance on the other test images at this threshold can be seen in fig. 5.

Figure 5: Testing images classified in chromatic color space by the obtained threshold of 135

The classifier can reliably detect facial features with very limited misclassification with respect to clothing. However, other skin-colored areas are also classified as faces, such as hand/arm skin, and beige-red colored objects in the environment. Still, the results are impressive considering the relative simplicity of our model and minimal computational costs, and is very well suited to simple facial-recognition tasks such as those found on IoT devices.

The ROC plot using the $\mathbf{YC_bC_r}$ color space can be seen below in fig. 6.

Comparing classification using the $\mathbf{YC_bC_r}$ color space, we observe a handful of differences compared to the chromatic color space. Most notably, the EER is encountered at a much smaller threshold value due to the normalizing constant of the Gaussian being much lower than that of the chromatic color space, given by eq. (11) above.

Moreover, in the $\mathbf{YC_bC_r}$ color space, we can see that environmental noise is significantly reduced compared to the chromatic color space; note that the beams in fig. 4 are misclassified, but not in fig. 8, and that more of the face is captured in fig. 8.

Using this new threshold of 0.002144315, we can see the original image and its classified counterpart in fig. 7.

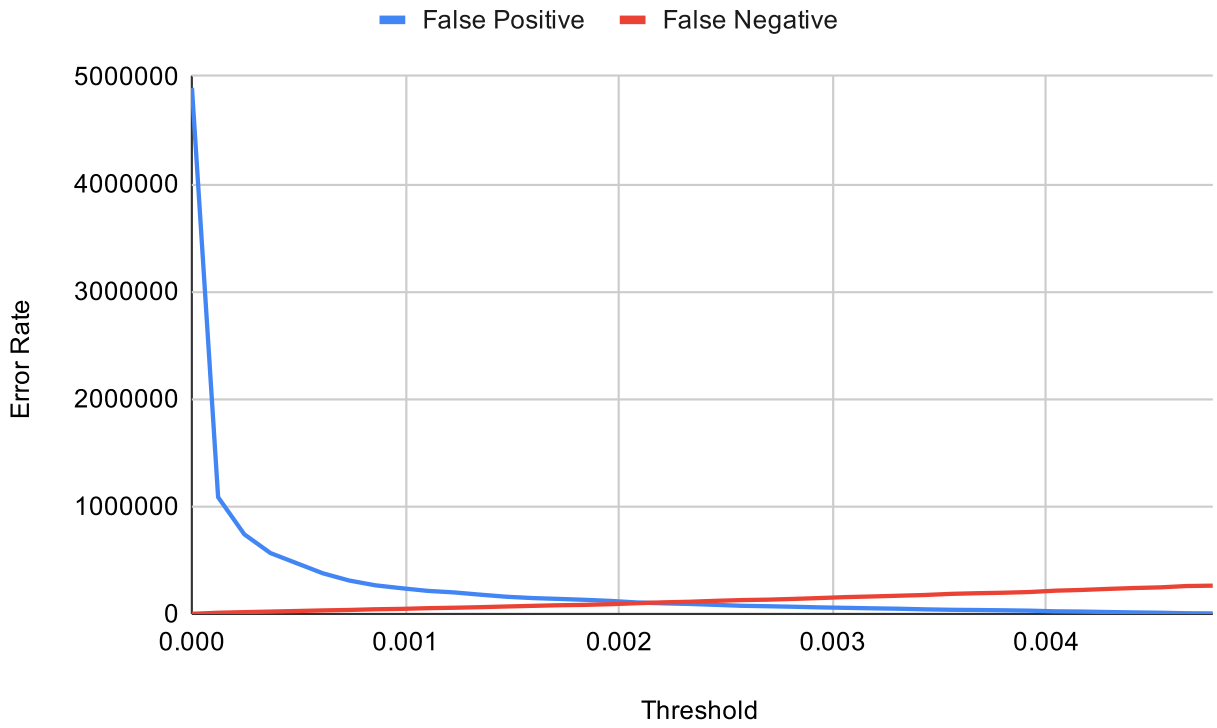

Figure 6: ROC curve for the obtained Gaussian with the two test images

Figure 7: Training image classified in $\mathbf{YC_bC_r}$ color space by the obtained threshold of 0.002144315

And the test images are classified by this new Gaussian in Figure 8.

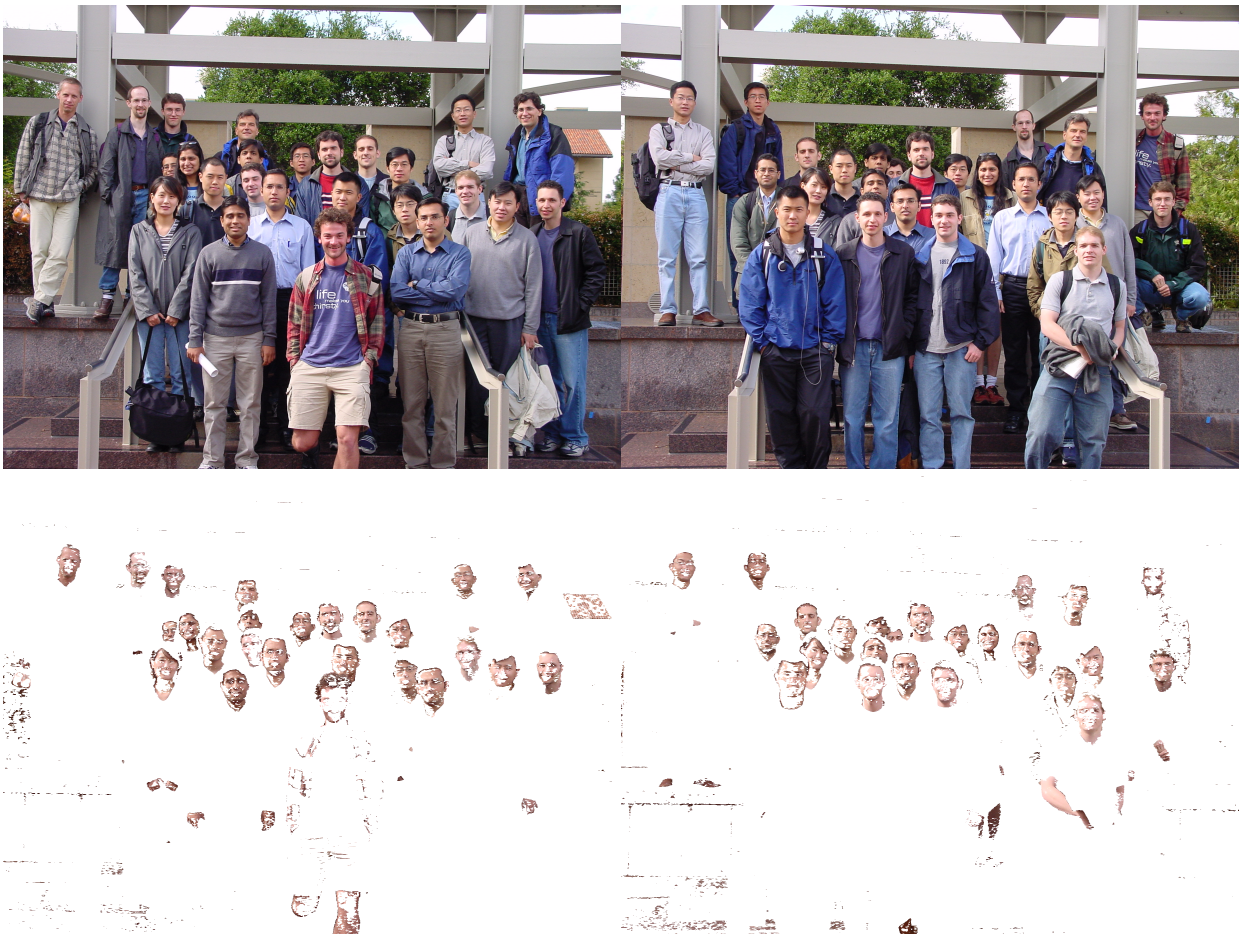

Figure 8: Testing images classified in $\mathbf{YC_bC_r}$ color space by the obtained threshold of 0.002144315

In $\mathbf{YC_bC_r}$, the lightness information encoded in the RGB data is linearly separated and transformed into a luma signal and two chrominance signals. Separating this information also has the added benefit of removing most of the correlation between the input channels, therefore providing a smaller normalizing constant and better classification while still only being 2-dimensional.

Plotting the false acceptance rate v.s. the false rejection rate for both color spaces allows us to compare the classifiers more objectively. This can be found in fig. 9.

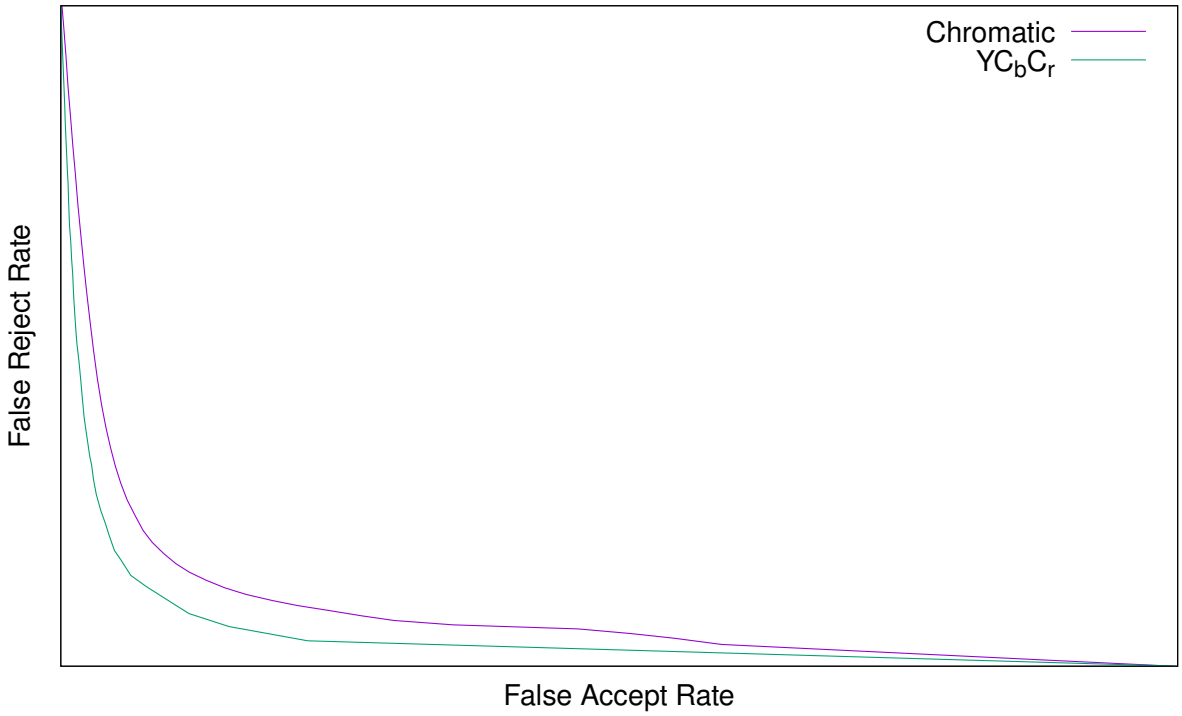

Figure 9: A comparison of error rates between the two classifiers.

As can be seen, the area under the $\mathbf{YC_bC_r}$ ROC curve is much less, and therefore it is a much better classifier than the chromatic classifier.