Course: DD2427 - Exercise Set 4

In this exercise you will model skin colour with a multi-variate Gaussian. Let $\mathbf{x}$ be the vector that contains the data associated with a pixel's colour (its rgb value for instance) then assume

$$p(\mathbf{x}\,|\,\text{skin pixel}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_s|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_s)^T\Sigma_s^{-1}(\mathbf{x}-\boldsymbol{\mu}_s)\right) \tag{1}$$

where $|\Sigma_s|$ denotes the determinant of the matrix Σ_s. However, the values of the parameters $(\boldsymbol{\mu}_s, \Sigma_s)$, defining this model, are unknown. These need to be learned from training data.

Exercise 1: *Learning the multi-variate Gaussian*

To learn the parameters of the multi-variate Gaussian, we require training data. Fortunately, lots of people have built databases of colour images of human faces and have made them available publicly. We will use images available at http://vis-www.cs.umass.edu/lfw/. The complete database contains a large number of images, far more than needed for our purposes, therefore you should just download a small proportion of them available at: http://vis-www.cs.umass.edu/lfw/lfw-bush.tgz. You can `gunzip` and `tar` to extract the images. Place them in a separate directory. You will use these to learn the colour of skin pixels and you will, in reality, only require a small number of them to learn the multi-variate Gaussian model.

In the downloaded images the centre of the face is at the centre of the image and composes roughly 30% of the area of the image (exact details are available at the face database web-site). If you grab the pixels in a relatively small rectangular region centred at the centre of the image, then you can be quite sure that these will be skin coloured pixels. Your first task is to write a function that takes as input the name of the file and a number, `p`, in `[0,1]`. The function then grabs the centre rectangular region of size (`floor(p*W)`)(`floor(p*H)`) from the image where `W` and `H` are the width and height of the original image. Call this function via

```
function cim = GrabCenterPixels(im_fname, p)
```

The returned array `cim` will have size `floor(p*H)`$\times$`floor(p*W)`$\times$`3`. You can turn the pixel data in this array into a more useful shape with the command:

```
rgb_data = reshape(cim, [size(cim,1)*size(cim,2), 3]);
```

This reshapes `cim` into the array, `rgb_data`, of size `size(cim,1)*size(cim,2)`$\times$`3`. Each row of the matrix corresponds to the *rgb* value of one skin pixel. The first column of this array corresponds to the *r* data, the second column to the *g* data and the third to the *b* blue data.

Next read in `n` ($\sim$20) of the downloaded face images. From each image extract the central pixels with the function you've just written, keep a record of these skin pixels and then compute the mean value of the *rgb* pixel values and the covariance of these pixels. Use the `Matlab` functions `mean` and `cov` to compute these quantities. Perform this task in the function

```
function [mu, Sigma] = TrainColourModel(DirName, n)
```

which takes as input the name of the directory containing the face images and the number, `n`, of images to be used for training. The output will be the mean vector, `mu` (size `3` $\times$ `1`), of the *rgb* pixel data you have extracted and its covariance, `Sigma` (size `3` $\times$ `3`).

Matlab Tip Remember you can concatenate an array `A` of size `n_a`$\times$`3` and another array `B` of size `n_b`$\times$`3` into a new array of size `(n_a + n_b)`$\times$`3` with the command: `C = [A;B];`

When I performed this task with `n=20`, `p=.2` and used the first `n` images, I calculated `mu` and `Sigma` as:

$$\texttt{mu} = \begin{bmatrix} 176.91 & 129.18 & 103.88 \end{bmatrix}, \quad \texttt{Sigma} = 1000 * \begin{bmatrix} 1.7264 & 1.4316 & 1.4479 \\ 1.4316 & 1.4268 & 1.4498 \\ 1.4479 & 1.4498 & 1.5935 \end{bmatrix}$$

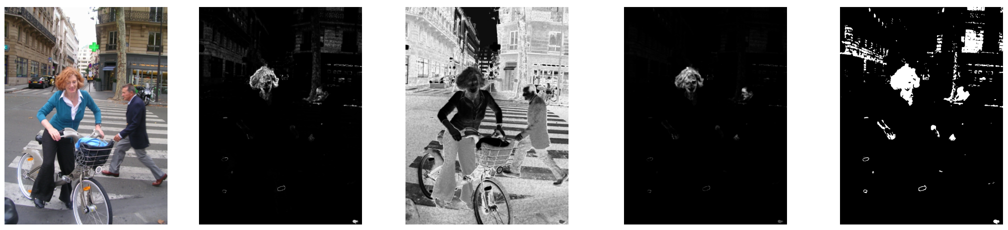

(a) image **(b)** $p(\mathbf{x}|\text{skin})$ **(c)** $p(\mathbf{x}|\text{non-skin})$ **(d)** $\frac{p(\mathbf{x}|\text{skin})}{p(\mathbf{x}|\text{non-skin})}$ **(d)** classification

Figure 1: The quantities evaluated to decide via a likelihood ratio test if a pixel corresponds to skin or not. These results were obtained using an *rgb* representation of color.

Exercise 2: *Skin Colour Likelihood*

At this stage you have learnt the parameters for

$$p(\mathbf{x} \mid \text{skin pixel})$$

where $\mathbf{x}$ is a pixel's *rgb* value. Now you'll investigate the usefulness of this model. Load the image `bike_small.jpg`. For each pixel in `bike_small.jpg` we want to compute $p(\mathbf{x} \,|\, \text{skin pixel})$ as defined in equation (1) where $\mathbf{x}$ contains the *rgb* value of the pixel and $\boldsymbol{\mu}_s$ = `mu` and Σ_s = `Sigma`. The `Matlab` functions `det` and `inv` compute the determinant and inverse of a matrix. Write a function that evaluates equation (1) for a set of points. These points are contained in the array `xs` of size `N`×`d` where `N` is the number of points and `d` is the dimension of each point.

```
function lvals = GaussLikelihood(xs, mean, Sigma)
```

Matlab Tip If `lvals` is an array of size `N`×`1`, then it can be turned into an array of dimensions equal to the test image `im` with the command:

```
im_lvals = reshape(lvals, [size(im, 1), size(im, 2)]);
```

Run your new function on the rgb pixel values obtained from the image `bike_small.jpg`; display the results. These results should look something like those in figure 1(b).

Exercise 3: *Use HSV colour model*

Is the *rgb* colour model the best represention of colour for this task of detecting skin pixels ? In this exercise you will investigate using the HSV colour model instead. Fortunately, you can reuse, with only minor additions, most of the code you've already written. Before you start, note that as the hue of a colour is an angle, it is easier to let $\mathbf{x} = (\cos(\text{hue}), \sin(\text{hue}), \text{saturation}, \text{value})$ to avoid the nuisance of $0° = 360°$. The four things you have to do are

- Amend the function `TrainColourModel` to take an extra input parameter. This will be a flag, `m`, to indicate which colour model you are using. Thus it becomes:

  ```
  function [mu, Sigma] = TrainColourModel(DirName, n, m);
  ```

  Once you have grabbed the training data then if `m` indicates you are using the HSV colour model then convert the *rgb* pixel data into *hsv* pixel data via the *Matlab* command `rgb2hsv`.

- From the *hsv* data create the augmented *hsv* data where its first column is cos(hue), the second sin(hue), the third saturation and the last value.

- The rest of the function should work as before assuming you haven't hard coded in the fact that your feature has dimension 3. You should get a new mean vector and covariance matrix.

- Convert the test image pixels into the same form of the HSV data as the training data and then compute the likelihood of each pixel corresponding to skin using `GaussLikelihood`. Display the result. It should look as in figure 2(b).

Exercise 4: *Likelihood ratio test to classify pixels as skin or non skin*

Finally, you can classify a pixel in your image as skin if

$$\frac{p(\mathbf{x}\,|\,\text{skin pixel})}{p(\mathbf{x}\,|\,\text{non-skin pixel})} > 1 \tag{2}$$

using the liklihood ratio test for classes with equal priors.

However, this requires some model/representation of $p(\mathbf{x}\,|\,\text{non-skin pixel})$. There are obviously lots of options for this. One suggestion is to use a multi-variate Gaussian as in the case of skin pixels. However, if you use this, training data (lots of non-skin pixels) is required from which you can learn the parameters of the Gaussian. Luckily for you I have downloaded a few such images from `flickr` for this task.

Download these from the course website and put them into a separate directory. Now you can reuse the function `TrainColourModel`, though you may want to grab the whole image as opposed to a central rectangle, to obtain a mean vector and covariance for the distribution describing the colour pixels for the *background* (or at least our sparsely sampled version of the background). If you run `GaussLikelihood` on the pixel data obtained from the image `bike_small.jpg` you should get an image that looks as figure 1(c), for an *rgb* representation and as figure 2(c), for a *hsv* representation.

If you plot the ratio of the likelihood values for the skin and non-skin models you should get results as in figures 1(d) and 2(d). Finally, you can create an image from the binary classification rule based on equation (3). To do this, create an array full of zeros, *Matlab* command `zeros`, and set an entry to one if for that pixel its likelihood of being skin coloured is greater than that of being non-skin coloured. The results you get should be similar to those in figures 1(e) and 2(e) depending on which colour model you use.

It may be useful to save the different mean vectors and covariance matrices you have calculated and to gather the commands you used to classify the pixels in an image into one function. It can be called with an image array as input and returns the binary image of classified pixels:

```
function bim = SkinClassifier(im)
```

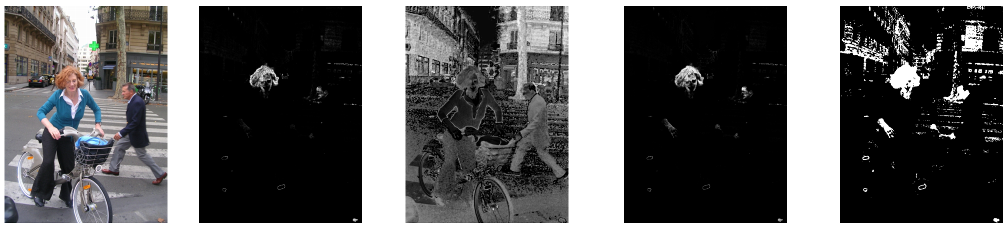
Mathematical Tip: In most cases it is better to compute log-likelihoods as oppose to likelihoods. That is compute

$$\log(\, p(\mathbf{x} \,|\, \text{skin pixel})\,) = -\frac{d}{2}\log(\,2\pi\,) - \frac{1}{2}\log(|\Sigma_s|) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_s)^T\Sigma_s^{-1}(\mathbf{x}-\boldsymbol{\mu}_s)$$

The classification can then be computed with this test:

$$\log(\, p(\mathbf{x} \,|\, \text{skin pixel})\,) - \log(\, p(\mathbf{x} \,|\, \text{non-skin pixel})\,) > 0 \tag{3}$$

This avoids computing the relatively expensive `exp` function and also is more numerically stable for values of $\mathbf{x}$ that produce very small likelihood values. For this exercise you can use either the likelihood or the log-likelihood. But, in general, it is probably better to use the log-likelihood values.



(a) image **(b)** $p(\mathbf{x}|\text{skin})$ **(c)** $p(\mathbf{x}|\text{non-skin})$ **(d)** $\frac{p(\mathbf{x}|\text{skin})}{p(\mathbf{x}|\text{non-skin})}$ **(d)** classification

Figure 2: The quantities evaluted to decide via a likelihood ratio test if a pixel corresponds to skin or not. These results were obtained using a *hsv* representation of color.

Exercise 5: *Finding your classmate*

Download `StudentImages.tar` from the course website. It contains images of some of `DD2424` students over the years. Apply your skin classifier to the image `Student1.jpg`. You can either write your function `FindBiggestComp` or download it from the course website. This function finds the largest connected componenent of skin pixels in the binary image:

```
function [bX, bY] = FindBiggestComp(bim)
```

It returns one vector containing the $x-$coordinates, the other the $y-$ coordinates, of the corners of the bounding box. Plot this bounding box on the original image and see if it overlaps with the face, see figure 3.

(If you write the function yourself use *Matlab* function `bwlabeln` to find the connected components in the image. Find the largest one. Compute the smallest rectangle that encloses this connected component - aka the bounding box. The *Matlab* function `regionprops` can help you do this.)

Well done you have written your very first face detector! Not to rain on your parade though this is not a particularly robust or clever one. See what happens when you run the same process on the other student images...

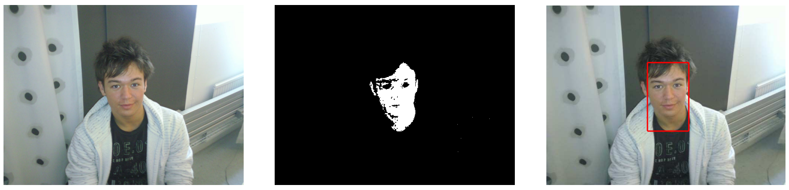


Figure 3: Using the HSV representation and simple blob analysis a face is found.

For the lecture on: 12th of April

Bring the following:

- *a print out of the functions you wrote for this exercise* ***`GrabCenterPixels`****,* ***`TrainColourModel`****,* ***`GaussLikelihood`****,* ***`SkinClassifier`****.*
- *a print out of the images shown in figure 3 for the images* ***`Student1.jpg`*** *and one other of the Student images.*