Assignment 03: EM for Gaussian Mixture Models (GMMs): Image Segmentation Application

HE Yongyi from USTC

For this problem, we will use Gaussian mixture models (GMMs) for image segmentation and study the behavior of the algorithm on a simple image. The file "GMMSegmentTestImage.jpg" available on the website provides an image that is used as a test for color blindness. Color normal individuals viewing the image should see a clear message in the image revealed by a partitioning of the image into distinct regions that are characterized by homogeneity of some color attributes. However, these regions also show variation in other attributes.
We will segment the image using Gaussian mixture modeling and assess the results we obtain. For the GMM fitting in this assignment, you may choose to use your own program using the equations provided during our discussion of GMMs in the context of the EM algorithm in class, or you may choose to use an existing implementation from the web, or from the appropriate MATLAB$^{\text{TM}}$ toolbox or python package.

(a) To simplify things and allow us to visualize our results, we will work in a 2-dimensional space instead of directly using the 3-dimensional RGB image data Convert the image RGB data from its native 3-dimensional RGB color space into a 2-dimensional rg chromaticity space by applying the transformation

$$\begin{aligned} r &= \frac{R}{R+G+B} \\ g &= \frac{G}{R+G+B}. \end{aligned} \tag{1}$$

Transform the test image into rg chromaticity space and visualize your transformed image by mapping it back into an 8-bit RGB image via the transformation

$$\begin{aligned} \alpha &= \frac{255}{\max(r,g,1-(r+g))} \\ R_{out} &= \text{ round }(\alpha r) \\ G_{out} &= \text{ round }(\alpha g) \\ B_{out} &= \text{ round }(\alpha(1-(r+g))). \end{aligned} \tag{2}$$

Comment on what attributes are preserved and what attributes are lost in the process of conversion from the input RGB to the output RGB (equivalently from RGB space to rg chromaticity space.
Hint: If you are using MATLAB, the functions `imread()` and `imshow()` , will be helpful for

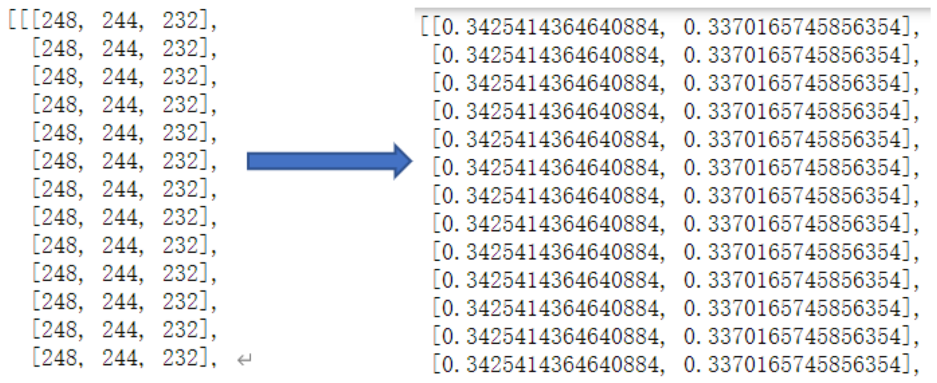
you in this exercise.

Solution:

First, convert the image RGB data from its native 3-dimensional RGB color space into a 2-dimensional rg chromaticity space by applying the transformation (1).

The sum of r, g, b will always be equal to 1. Knowing any two of these values, the third value can be computed. So we can look at the rg chromaticity space vertically (i.e., discard attribute b) without losing any information.

```python
im = Image.open('GMMSegmentTestImage.jpg')
pix = im.load()
width = im.size[0]
height = im.size[1]
RGB = [[0] * width ] * height
rg = []
co = []
for x in range(height):
    for y in range(width):
        r, g, b = pix[x, y]
        RGB[x][y] = [r,g,b]
        rg.append([r/(r+b+g), g/(r+b+g)]) # convert into 2D rg chromaticity space.
        co.append((r/(r+b+g), g/(r+b+g),b/(r+b+g))) # RGB to rgb
```

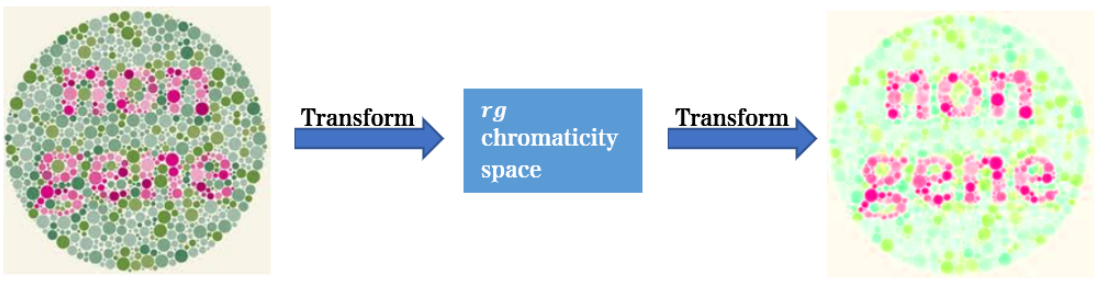


```
[[[248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
  [248, 244, 232],
```

```
[[0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
 [0.3425414364640884, 0.3370165745856354],
```

Then, in order to visualize the transformed image, we map back that into an 8-bit RGB image.

```python
width = im.size[0]
height = im.size[1]
im_out = im.copy()
pix_out = im_out.load()
co_out = np.array(co).reshape((height,width,3))
for x in range(height):
    for y in range(width):
        alpha = 255/(t[x][y].max()) # map back into RGB color value
        r_out = round(alpha*t[x][y][0])
        g_out = round(alpha*t[x][y][1])
        b_out = round(alpha*t[x][y][2])
        pix_out[x, y] = (r_out,g_out,b_out)
im_out.show()
```



In RGB space, pixels are identified by the intensity of the red, green and blue primary colors. Thus, a bright red can be represented as (255,0,0), and a dark red can be represented as (40,0,0). However, in the rg chromaticity space, colors are represented by the ratio of red, green and blue in the color, not by the intensity of each color. Because the sum of these ratios must be 1, we can only refer to the red and green ratios to represent the colors and can calculate the blue values as needed. **So the ratio of red, green and blue is preserved and the intensity information is lost in the process of conversion from RGB space to rg chromaticity space.** More professionally, the irradiance information is lost.

(b) Cluster the pixels in the image by fitting a $K = 3$ component GMM to the chromaticities in the image. Specifically, model the observed rg chromaticity values across the pixels in the image as a set of iid $d = 2$-dimensional rg vector observations from a Gaussian mixture with unknown mean and covariance and use the Expectation Maximization (EM) algorithm for GMM parameter estimation to estimate the unknown mean vectors and covariance matrices for the GMM. Because the EM algorithm is only locally convergent, you need to perform the GMM parameter estimation with multiple random initializations and select between these to obtain a "good" result. For determining a "good" result, you can either evaluate the likelihood function, or (more readily) you can visualize your results as described in the following parts and assess the fit based on the visualizations.

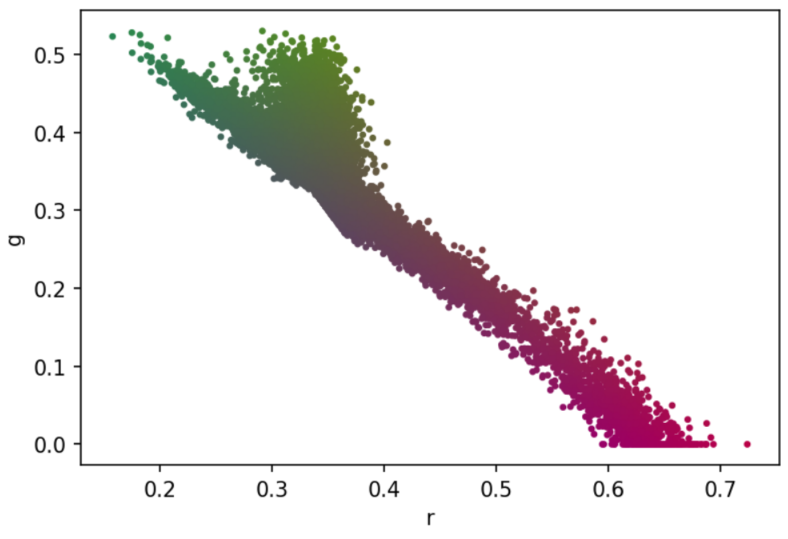
Hint: The MATLAB function `reshape()` will be helpful for re-organizing the data from the $2D$ two rg channel chromaticity image representation into a sequence of $2D$ rg vectors in this exercise.

1. Visualize the GMM fit you have obtained by making a scatter-plot of the rg chromaticities for the pixels in the image and superimposing contours for the components in your estimated GMM on this plot. In particular, for ea h mixture component in the GMM, indicate mean value of the mixture component on the scatter-plot by a "$+$" and plot the "3σ" contour that corresponds to locations at which the components' probability falls to a level that is a fraction $\exp(-3^2/2) = \exp(-9/2) \approx 0.0111$ of its peak value. You may find it useful to use the singular value decomposition (SVD) to obtain your contours - although for this 2D setting, you can also compute these analytically using the formula for the roots of a quadratic equation.
2. Use the GMM to compute, for each pixel, the posterior probability that it came from the mixture component j, for $j = 1, 2, \ldots K$. Visualize the results of the "soft" or probabilistic segmentation you have obtained in this process as a series of images. Specifically, for each mixture component j, create and display a normalized image that represents the posterior probability that a pixel belongs to the class j as an 8-bit gray-scale value, where gray-scale values of 0 corresponds to a probability of 0 and a gray-scale value of 255 corresponds to a probability of 1. Present one set of K such images for one of the "good" mixture model fits that you obtain. Comment on your results.
3. Comment on the effect of different initializations on your GMM fit and how these appear in the visualizations presented in the preceding two parts.

Solution:

First, make a scatter-plot of rg chromaticity using the primary color of each pixel in the image. Then we can roughly determine the region of each category and use it for subsequent comparison of the results.

```
a = np.array(rg).reshape(-1,2) # reshape rg space
plt.scatter(a[:,0],a[:,1],s=5,c=co) # 'co' is rgb color value list
plt.xlabel('r')
plt.ylabel('g')
plt.show()
```



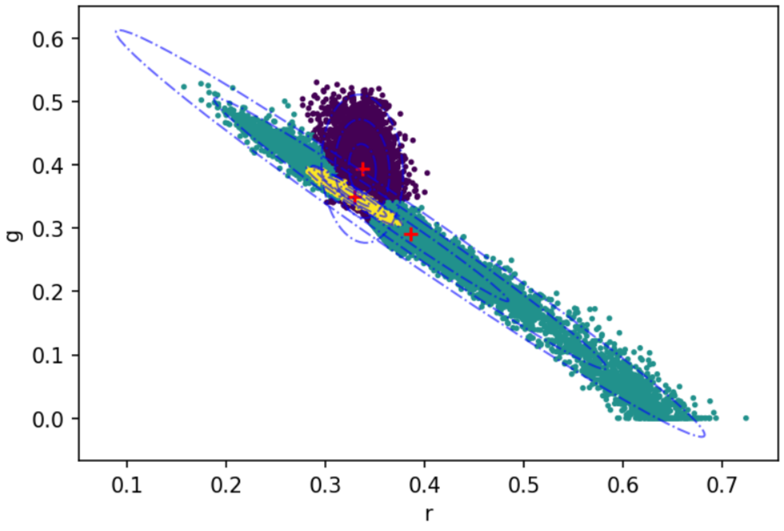
For each value of K, we use three random initializations to compare the results. **The means in the three random initializations are different, and the covariances and weights are the same.**

Initialization I

means_1:
```
[[0.33551952 0.43064332]
 [0.57371504 0.32928455]
 [0.39174263 0.26361887]]
```

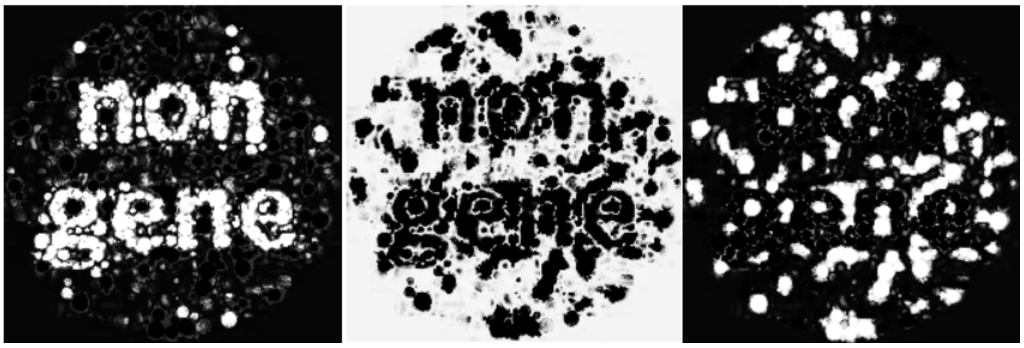
Visualization results of the GMM fit:

```python
# Draw elliptical contours using mean values and covariances.
def draw_ellipse(position, covariance, ax=None, **kwargs):
    ax = ax or plt.gca()
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)
    #   3-sigma contour
    for sigma in range(1, 4):
        ellipse = Ellipse(position, sigma * width, sigma * height,angle, **kwargs)
        ax.add_patch(ellipse)
        ellipse.set(fc='None',ls='-.',ec='blue',alpha=0.6,lw=1)

#   Visualize the GMM fit
def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=3, cmap='viridis')
    ax.scatter(gmm.means_[:,0],gmm.means_[:,1],s=50,marker='+',c='r')
    plt.xlabel('r')
    plt.ylabel('g')
    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```



We can see that all three classifications in this initialization converge to valid values, and have a good classification result. Then create and display normalized images for each category based on posterior probabilities.

```
def norm(im,j,p=post_p):
    # 'post_p' is the list of posterior probabilities fitted by GMM.
    pix = im.load()
    width = im.size[0]
    height = im.size[1]
    for x in range(height):
        for y in range(width):
            value = round(255 * p[x][y][j])
            pix[x, y] = (value,value,value)
    return pix
```



The classification result is good, the string 'non gene' can be clearly distinguished, but there is still some noise.

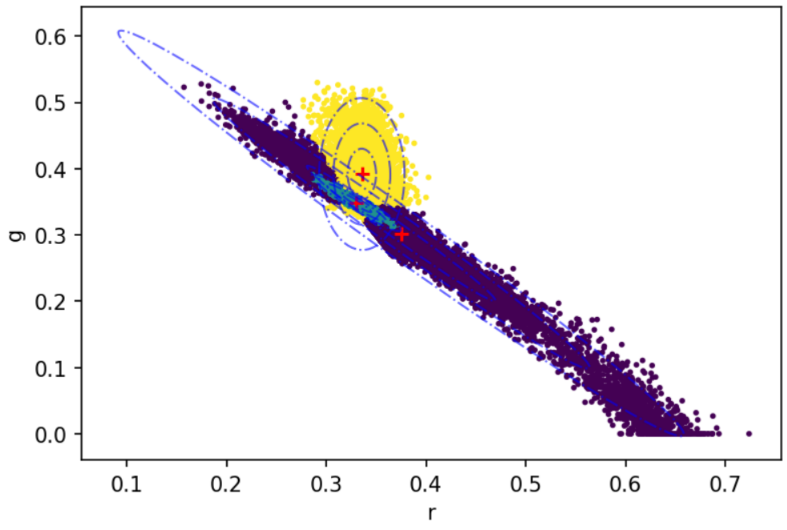
Last, the estimated mean vector and covariance matrix are

```
[[0.32920405 0.34970209]
 [0.38448459 0.29232533]
 [0.33689456 0.39348667]]
[[[ 2.54368186e-04 -2.37566899e-04]
  [-2.37566899e-04  2.46337968e-04]]

 [[ 9.71190960e-03 -1.03928265e-02]
  [-1.03928265e-02  1.13587762e-02]]

 [[ 1.89896183e-04 -4.11736424e-05]
  [-4.11736424e-05  1.51236021e-03]]]
```

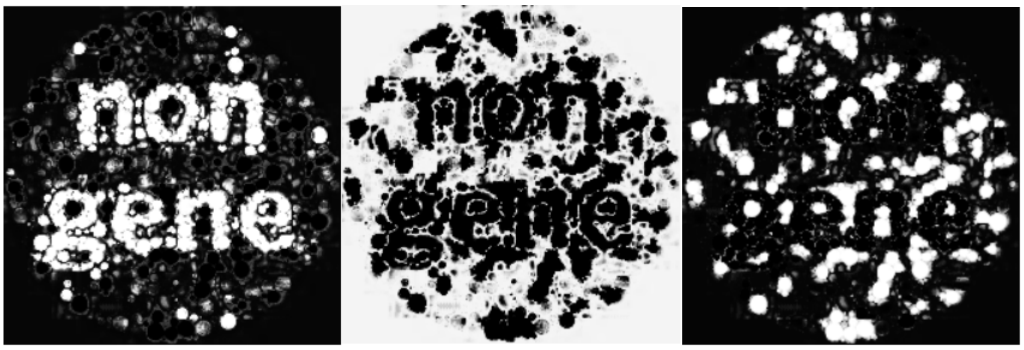
Initialization II

means_2:
```
[[0.49659042 0.2460994 ]
 [0.29488734 0.23042118]
 [0.50838317 0.30914619]]
```

Visualization results of the GMM fit:



We can also see that all three classifications in this initialization converge to valid values, and have a good classification result. Then create and display normalized images for each category based on posterior probabilities.



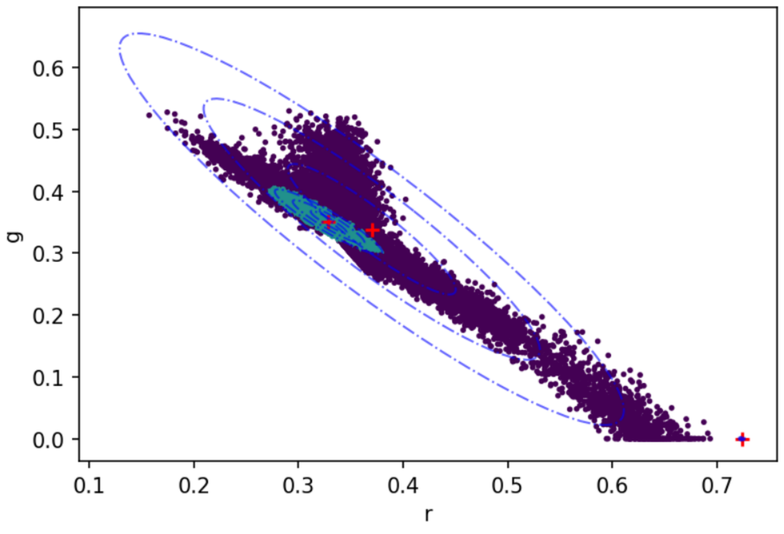
The classification result is also good, but in the first class of normalized images, there is more noise.

Last, the estimated mean vector and covariance matrix are

```
[[0.37495134 0.30233352]
 [0.33010778 0.34860081]
 [0.33566864 0.39192781]]
[[[ 8.92431029e-03 -9.53352040e-03]
  [-9.53352040e-03  1.03995581e-02]]

 [[ 2.21821380e-04 -2.02070596e-04]
  [-2.02070596e-04  2.01913358e-04]]

 [[ 2.03208035e-04 -6.47298923e-06]
  [-6.47298923e-06  1.45757735e-03]]]
```

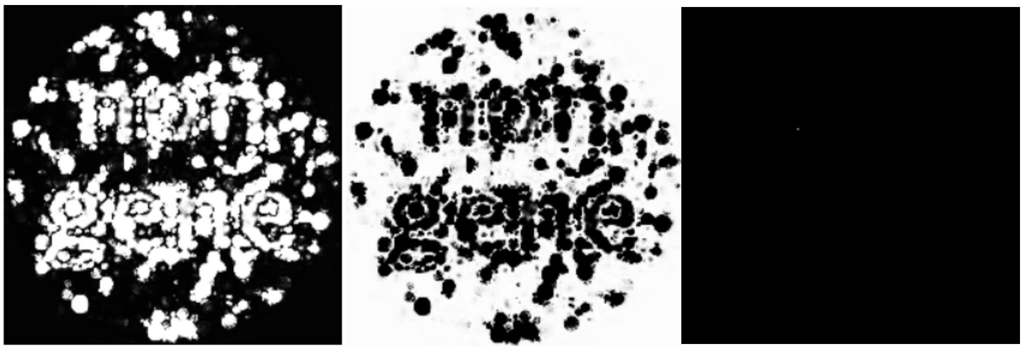
Initialization III

means_3:
```
[[0.4907868   0.4547058 ]
 [0.20943931 0.5363162 ]
 [0.55362552 0.44461485]]
```



Only two classifications converge to valid values.

Then create and display normalized images for each category based on posterior probabilities.



The classification result is bad, the strings cannot be distinguished, and even the normalized image of the third category becomes all black.

Last, the estimated mean vector and covariance matrix are

```
[[3.70035933e-001 3.38668782e-001]
 [3.28805023e-001 3.50570822e-001]
 [7.23756906e-001 1.17722670e-244]]
[[[ 6.45372942e-003 -7.83355944e-003]
  [-7.83355944e-003  1.11372821e-002]]

 [[ 2.93713697e-004 -2.81066553e-004]
  [-2.81066553e-004  3.08793875e-004]]

 [[ 1.00000000e-006 -3.82040181e-246]
  [-3.82040181e-246  1.00000000e-006]]]
```

Comment:

By visualizing the GMM fitting results, we can conclude that different initializations have an effect on the GMM fitting.

A not "good" initialization may lead to non-convergence of the classification results and not getting K valid categories, e.g., the mean of the third category in initialization III does not converge to a valid value, and the normalized image becomes an all-black plot. And, different initializations can lead to different classification results. Although the mean values of initialization I and initialization II both converge to the valid values and the differences in the GMM fit plots are small, it can be seen based on the normalized images that the results of initialization II generate more noise than initialization I. **So among these three random initializations, the result of initializing I is a "best" result.**

The above initialization is generated randomly using uniform distribution. However, observing the scatter plot of the pixel points, it is found that the pixel points are in a very concentrated region, and using such an initialization method may lead to results that do not converge to valid values. So a new random initialization method is considered, where $K = 3$ points are randomly selected as the initial mean of the GMM among the sample points in the rg chromaticity space. This means that the purpose of randomly generated initialization is achieved and the possibility of non-convergence of the results is reduced. And through experiments, the obtained results converge to the valid values and the classification results are good.

(c) Repeat the exercise in Step 1b for $K = 4$ and $K = 5$. Comment on the results.

Solution:

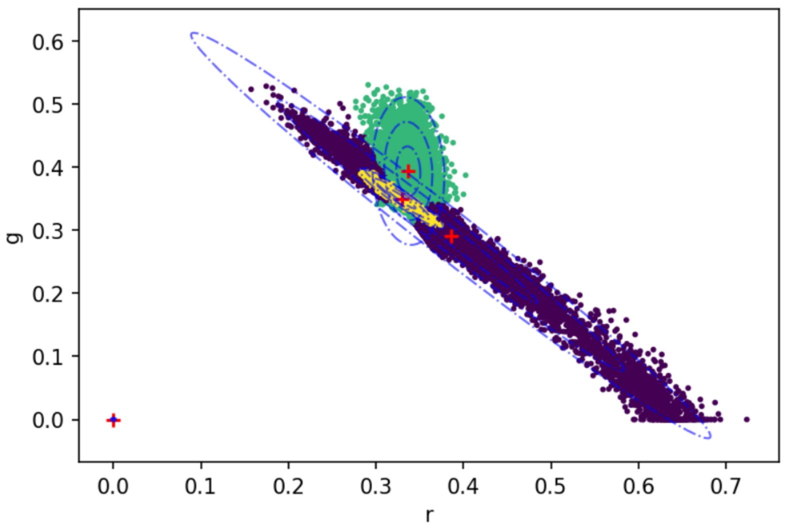
1. $K = 4$

Initialization I

means_1:

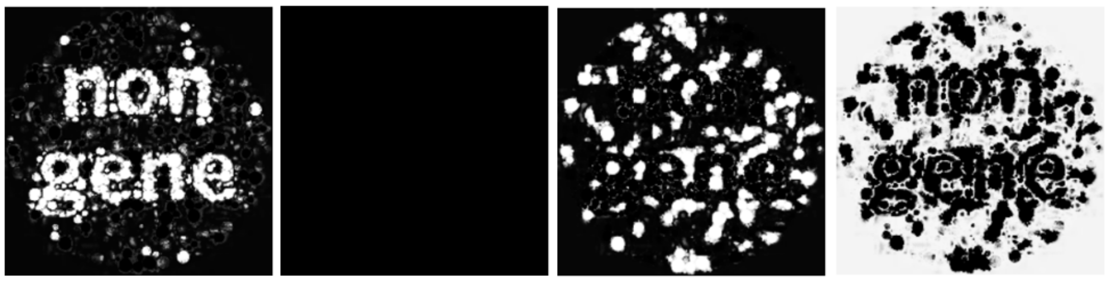
```
[[0.51558597 0.31206077]
 [0.20112963 0.10357921]
 [0.41702896 0.413454  ]
 [0.51778725 0.03091371]]
```

Visualization results of the GMM fit:



Only three classifications converge to valid values.

Then create and display normalized images for each category based on posterior probabilities.



The string 'non gene' can be clearly distinguished, but one of the normalized images is all black.

Last, the estimated mean vector and covariance matrix are

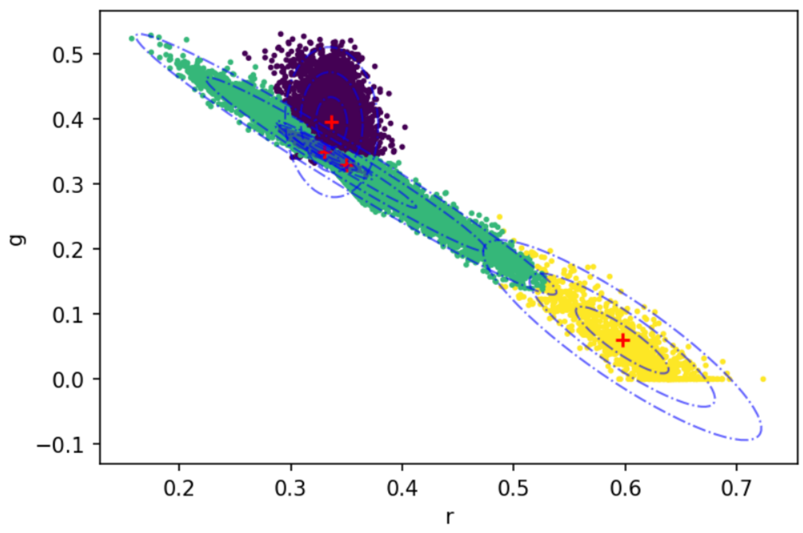
```
[[0.38564367 0.29112755]
 [0.         0.        ]
 [0.33696333 0.39342751]
 [0.32913339 0.3497818 ]]
[[[ 9.79628775e-03 -1.04863114e-02]
  [-1.04863114e-02  1.14649474e-02]]

 [[ 1.00000000e-06  0.00000000e+00]
  [ 0.00000000e+00  1.00000000e-06]]

 [[ 1.88575369e-04 -4.30503125e-05]
  [-4.30503125e-05  1.51925331e-03]]

 [[ 2.58922301e-04 -2.42484429e-04]
  [-2.42484429e-04  2.52037766e-04]]]
```

Initialization II

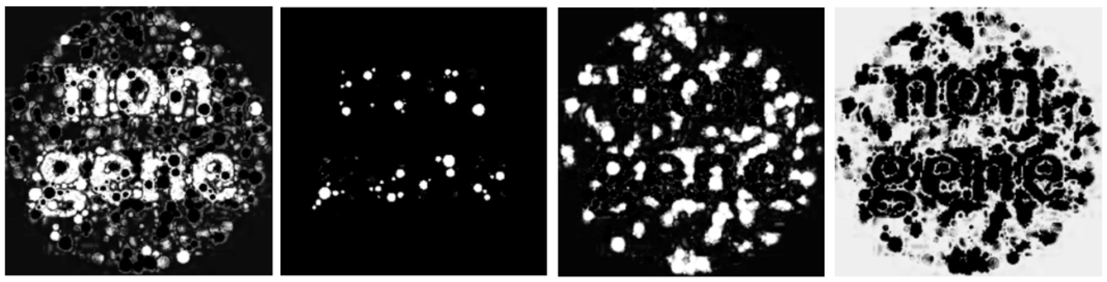
means_2:

```
[[0.08282263 0.55776831]
 [0.05038718 0.54143693]
 [0.42857205 0.15582196]
 [0.2414864  0.26453862]]
```

Visualization results of the GMM fit:



We can see that all four classifications in this initialization converge to valid values, and have a good classification result. Then create and display normalized images for each category based on posterior probabilities.



The classification result is good, the string 'non gene' can be distinguished, but there is still some noise.

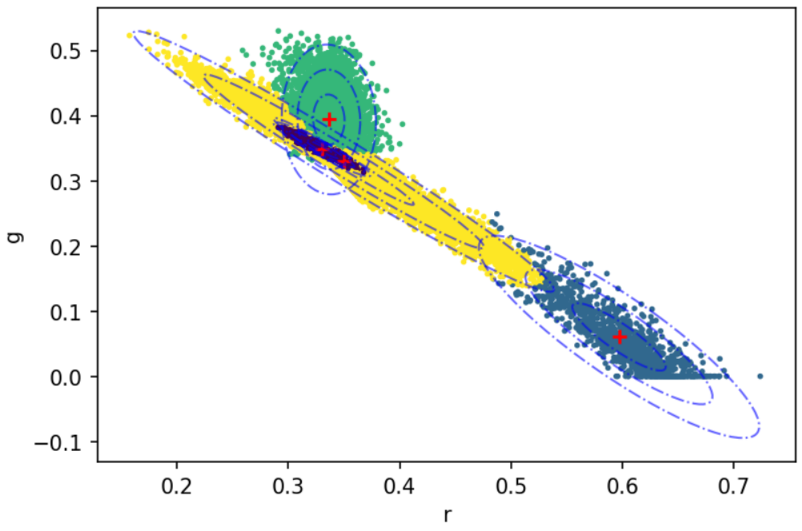
Last, the estimated mean vector and covariance matrix are

```
[[0.20657277 0.52112676]
 [0.22723334 0.42093084]
 [0.34558871 0.33244957]
 [0.33510603 0.37255739]]
[[[ 1.00000000e-06  5.23852945e-31]
  [ 5.23852945e-31  1.00000000e-06]]

 [[ 2.30730297e-06 -2.57446469e-06]
  [-2.57446469e-06  6.06991099e-06]]

 [[ 3.92446079e-03 -4.14728717e-03]
  [-4.14728717e-03  4.46468323e-03]]

 [[ 1.74684991e-04 -3.80669429e-05]
  [-3.80669429e-05  1.46241217e-03]]]
```

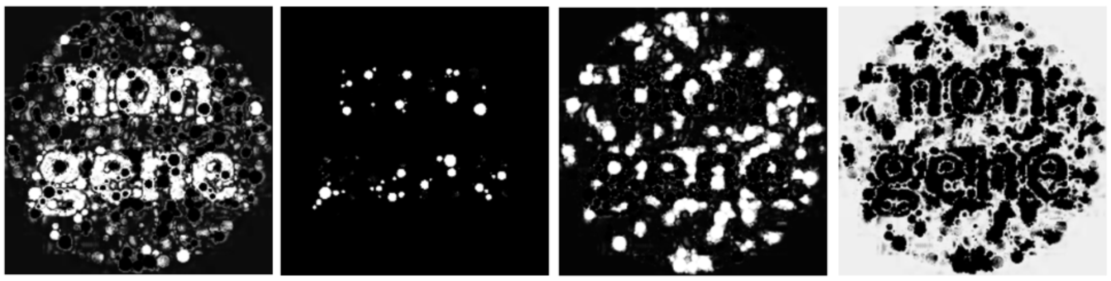
Initialization III

means_3:

```
[[0.0719059  0.29314639]
 [0.07691301 0.53514176]
 [0.39760634 0.09436908]
 [0.08380977 0.49056107]]
```

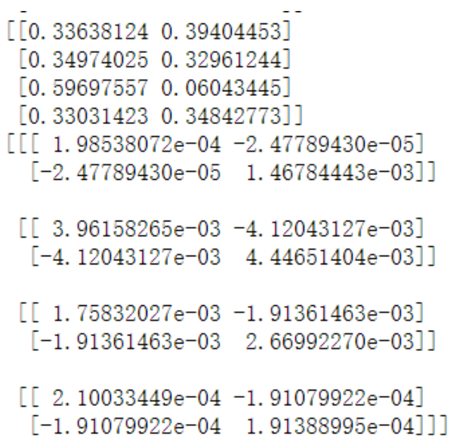


We can also see that all four classifications in this initialization converge to valid values, and have a good classification result. Then create and display normalized images for each category based on posterior probabilities.



The classification result is also good, the string 'non gene' can be distinguished, but there is still some noise. And as far as the results of normalized images are concerned, the classification effect of initialization III is not much different from that of initialization II.

Last, the estimated mean vector and covariance matrix are



```
[[0.33638124 0.39404453]
 [0.34974025 0.32961244]
 [0.59697557 0.06043445]
 [0.33031423 0.34842773]]
[[[ 1.98538072e-04 -2.47789430e-05]
  [-2.47789430e-05  1.46784443e-03]]

 [[ 3.96158265e-03 -4.12043127e-03]
  [-4.12043127e-03  4.44651404e-03]]

 [[ 1.75832027e-03 -1.91361463e-03]
  [-1.91361463e-03  2.66992270e-03]]

 [[ 2.10033449e-04 -1.91079922e-04]
  [-1.91079922e-04  1.91388995e-04]]]
```

Comment:

The overall conclusion has been summarized in the case of $K = 3$. Only the results of different initializations for the $K = 4$ case are commented here.

The mean value of one of the categories in initialization I did not converge to a valid value. And initialization II and initialization III have similar performance on classification results and normalized images. **So among these three random initializations, the result of both initialization II and initialization III is a "best" result.**

---

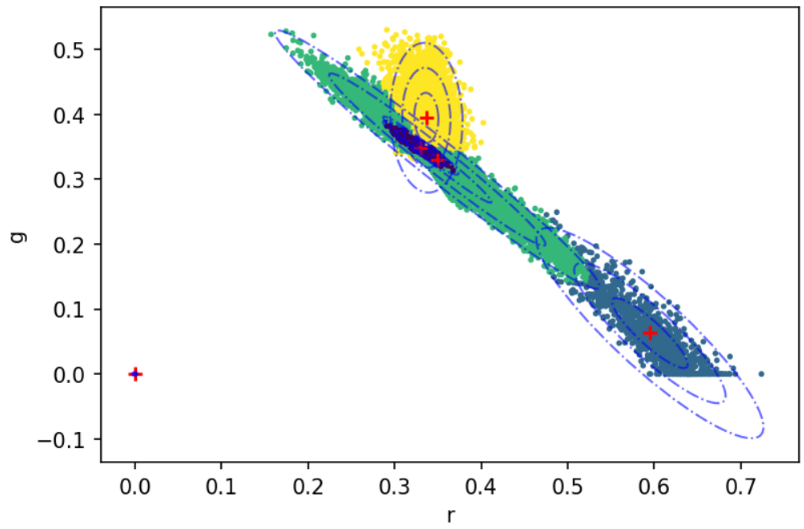
2. K = 5

Initialization I

means_1:

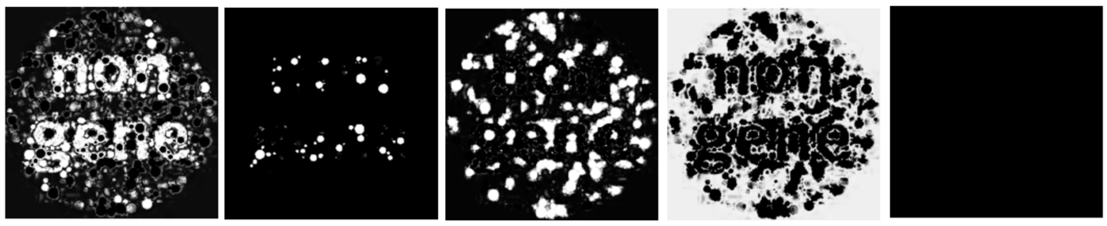
```
[[0.32879096 0.30966908]
 [0.55714656 0.54964928]
 [0.24570177 0.23125924]
 [0.41774794 0.62709883]
 [0.56760199 0.4043195 ]]
```

Visualization results of the GMM fit:



Only four classifications converge to valid values.

Then create and display normalized images for each category based on posterior probabilities.



The string 'non gene' can be distinguished, but there is lots of noise. And one of the normalized images is all black.

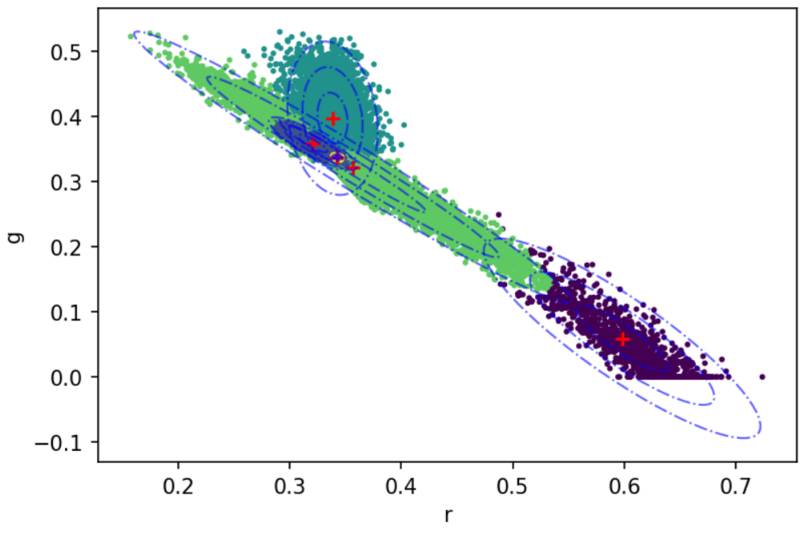
Last, the estimated mean vector and covariance matrix are

```
[[0.32924675 0.34963408]
 [0.4111392  0.27327315]
 [0.37944696 0.29619685]
 [0.33688792 0.39301362]
 [0.35379061 0.5198556 ]]
[[[ 2.50961427e-04 -2.33532252e-04]
  [-2.33532252e-04  2.41247525e-04]]

 [[ 1.18877553e-02 -1.15547041e-02]
  [-1.15547041e-02  1.14253219e-02]]

 [[ 9.19636433e-03 -1.00515892e-02]
  [-1.00515892e-02  1.11823600e-02]]

 [[ 1.91992932e-04 -4.52065843e-05]
  [-4.52065843e-05  1.52124269e-03]]

 [[ 1.00000000e-06  1.01743940e-17]
  [ 1.01743940e-17  1.00000000e-06]]]
```

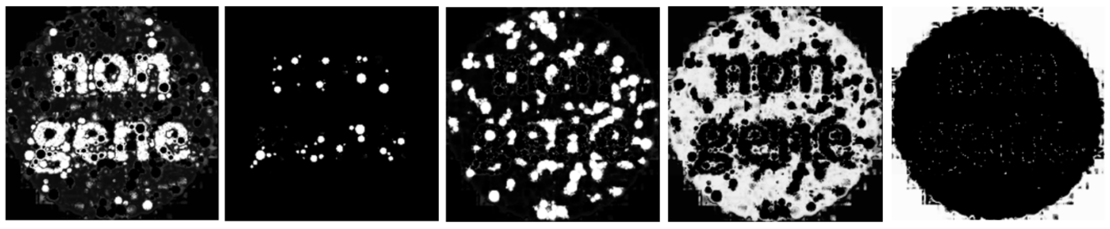
Initialization II

means_2:
```
[[0.57923357 0.18311606]
 [0.1758807  0.54479747]
 [0.47138732 0.63281037]
 [0.47330259 0.15268181]
 [0.32704339 0.19452989]]
```

Visualization results of the GMM fit:

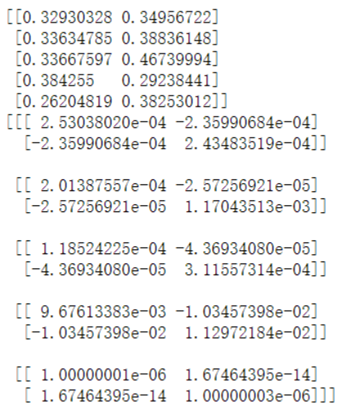


We can see that all five classifications in this initialization converge to valid values. But this may not be a good classification result, because we can see that there are three classifications with mean centers very close to each other.



The classification result is not bad. The string 'non gene' can be distinguished, but there is lots of noise.

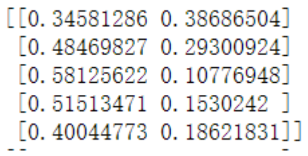
Last, the estimated mean vector and covariance matrix are



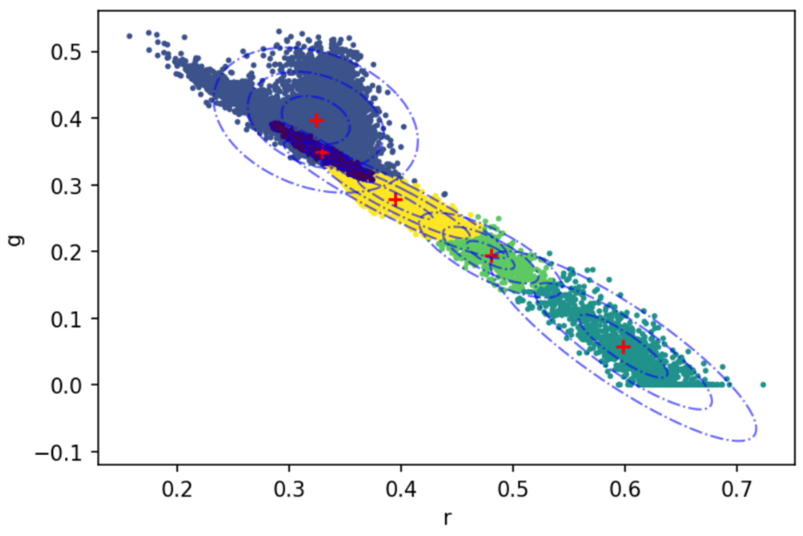
```
[[0.32930328 0.34956722]
 [0.33634785 0.38836148]
 [0.33667597 0.46739994]
 [0.384255   0.29238441]
 [0.26204819 0.38253012]]
[[[ 2.53038020e-04 -2.35990684e-04]
  [-2.35990684e-04  2.43483519e-04]]

 [[ 2.01387557e-04 -2.57256921e-05]
  [-2.57256921e-05  1.17043513e-03]]

 [[ 1.18524225e-04 -4.36934080e-05]
  [-4.36934080e-05  3.11557314e-04]]

 [[ 9.67613383e-03 -1.03457398e-02]
  [-1.03457398e-02  1.12972184e-02]]

 [[ 1.00000001e-06  1.67464395e-14]
  [ 1.67464395e-14  1.00000003e-06]]]
```

Initialization III

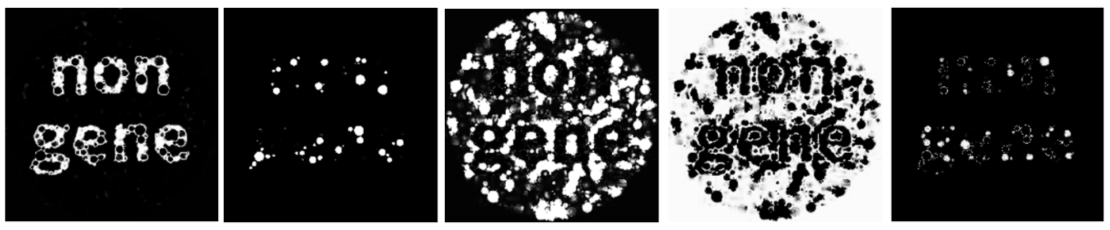
means_3:



```
[[0.34581286 0.38686504]
 [0.48469827 0.29300924]
 [0.58125622 0.10776948]
 [0.51513471 0.1530242 ]
 [0.40044773 0.18621831]]
```

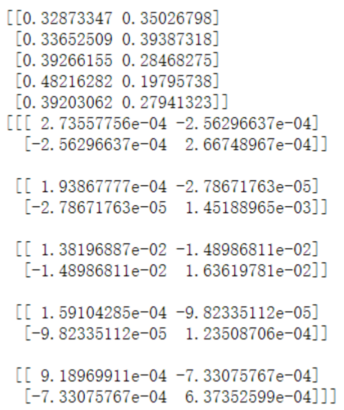


We can also see that all five classifications in this initialization converge to valid values, and have a good classification result. Then create and display normalized images for each category based on posterior probabilities.



The classification result is good, the string 'non gene' can be clearly distinguished.

Last, the estimated mean vector and covariance matrix are



```
[[0.32873347 0.35026798]
 [0.33652509 0.39387318]
 [0.39266155 0.28468275]
 [0.48216282 0.19795738]
 [0.39203062 0.27941323]]
[[[ 2.73557756e-04 -2.56296637e-04]
  [-2.56296637e-04  2.66748967e-04]]

 [[ 1.93867777e-04 -2.78671763e-05]
  [-2.78671763e-05  1.45188965e-03]]

 [[ 1.38196887e-02 -1.48986811e-02]
  [-1.48986811e-02  1.63619781e-02]]

 [[ 1.59104285e-04 -9.82335112e-05]
  [-9.82335112e-05  1.23508706e-04]]

 [[ 9.18969911e-04 -7.33075767e-04]
  [-7.33075767e-04  6.37352599e-04]]]
```

Comment:

The overall conclusion has been summarized in the case of $K = 3$. Only the results of different initializations for the $K = 5$ case are commented here.

The mean value of one of the categories in initialization I did not converge to a valid value. And although the classification results of initialization II all converge to valid values, the result is not a good one. Initialization III performs well on both classification results and normalized images, and the images can be clearly distinguished with little noise. **So among these three random initializations, the result of initialization III is a "best" result.**

In summary, the classification results for $K = 3$ and $K = 5$ were better than $K = 4$. Although the results of all three classifications can distinguish the string 'non gene', $K = 5$ is the clearest. However, both $K = 4$ and $K = 5$ showed overfitting, i.e., the pink color within the string was also divided into several categories. So $K = 3$ is optimal when overfitting is not desired, $K = 5$ is optimal when overfitting is allowed.