Assignment3

TASK A

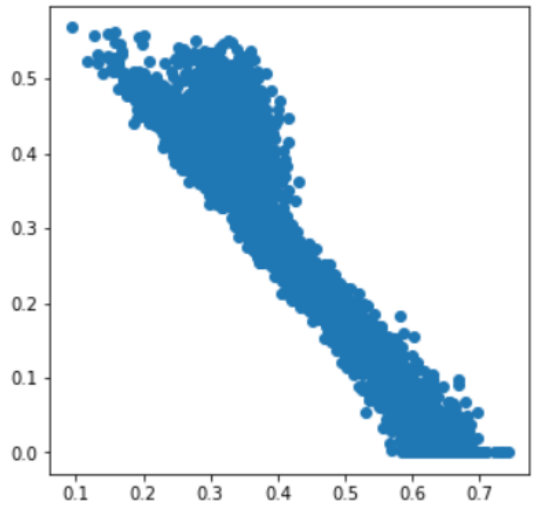
I take the given image for example.

- Firstly, transform the RGB space into rg chromaticity space and show the the distribution of the Red and Green color space by scatterplot which is shown below:

```python
from skimage import io
from skimage.io import imread, imshow
import matplotlib.pyplot as plt
import numpy as np
paint = imread('GMMSegmentTestImage.jpg')
paint_R = paint[:,:,0]*1.0/paint.sum(axis=2)
paint_G = paint[:,:,1]*1.0/paint.sum(axis=2)
plt.figure(figsize=(5,5))
plt.scatter(paint_R.flatten(),paint_G.flatten())
```

[2] 0.2s

```
<matplotlib.collections.PathCollection at 0x1b431570f10>
```

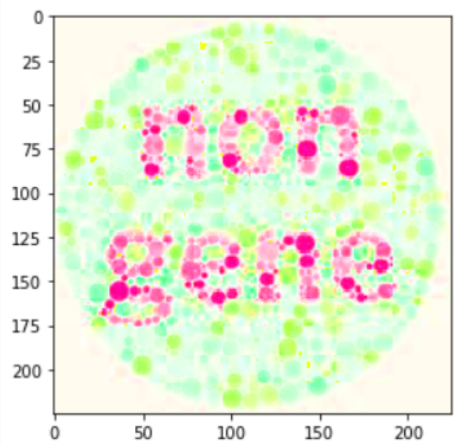


- Then, visualize my transformed image by mapping it back into an 8-bit RGB image

```python
alpha = 255/np.maximum(paint_R,paint_G,1-(paint_R+paint_G))
new_paint = paint
new_paint[:,:,0] = np.round(alpha*paint_R)
new_paint[:,:,1] = np.round(alpha*paint_G)
new_paint[:,:,2] = np.round(alpha*(1-(paint_R+paint_G)))
io.imshow(new_paint)
```

[15] ✓ 0.2s

```
<matplotlib.image.AxesImage at 0x1b4337702e0>
```



- what attributes are **preserved** and what attributes are **lost** in the process of conversion from RGB space to rg chromaticity?

let us see the transform:

$$r = \frac{R}{R+G+B} \tag{1}$$

$$g = \frac{G}{R+G+B} \tag{2}$$

in fact, there still has b:

$$b = \frac{B}{R+G+B} \tag{3}$$

we can easily get that:

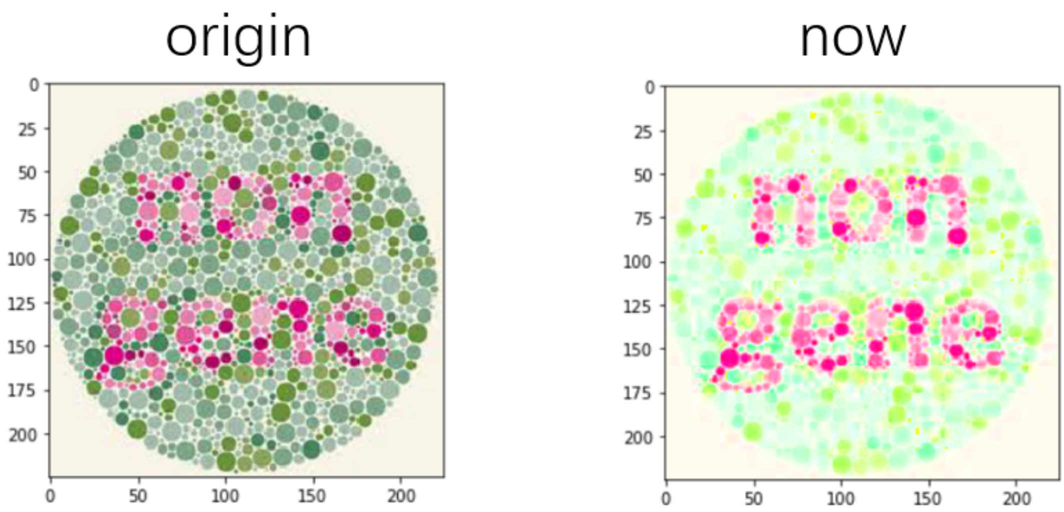
$$r+g+b=1 \tag{4}$$

and:

$$r:g:b = R:G:B \tag{5}$$

in the rg chromaticity space, we just throw away the "b" dimension.

but we could still get that "**the preserved attribute is the Ratio of three color channel values**"



compare the above 2 image, we can know that the reverse conversion is not possible with only two dimensions, as the **intensity information** is lost during the conversion to rg chromaticity, e.g. (1/3, 1/3, 1/3) has equal proportions of each color, but it is not possible to determine whether this corresponds to black, gray, or white. **And this is the attribute we lost.**

TASK B

Firstly, re-organizing the data from the 2D two rg channel chromaticity image representation into a sequence of 2D rg vectors in this exercise:

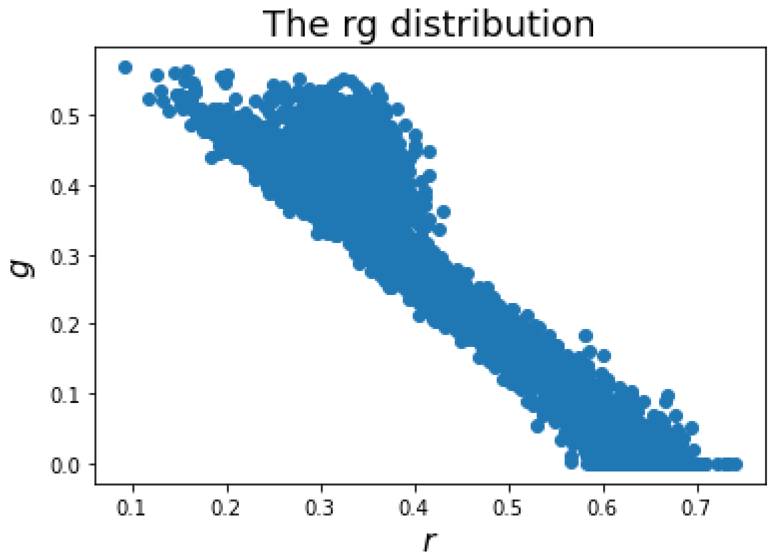
```
img = np.vstack((paint_R.flatten(),paint_G.flatten()))
img2 = img.T
print(img2.shape)
print(img2)
```

[29] ✓ 0.3s

```
(50625, 2)
[[0.34254144 0.33701657]
 [0.34254144 0.33701657]
 [0.34254144 0.33701657]
 ...
 [0.34254144 0.33701657]
 [0.34254144 0.33701657]
 [0.34254144 0.33701657]]
```

+ 代码 + 标记

here is the distribution of the rg chromaticicy space :



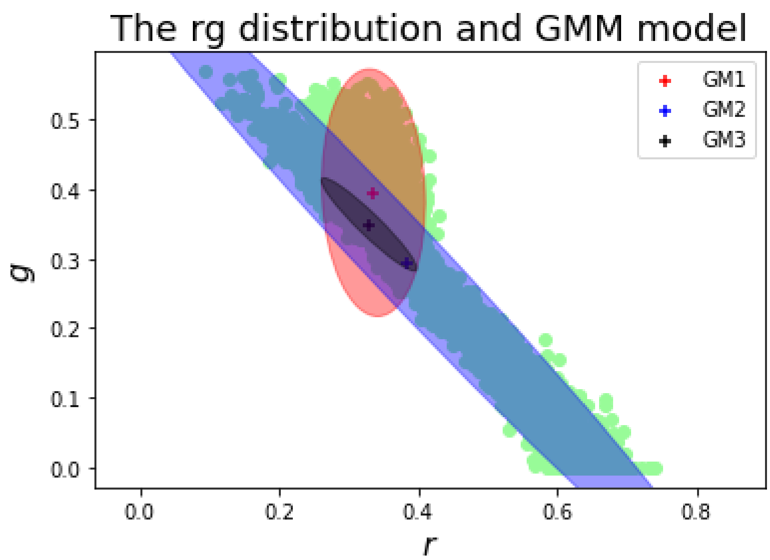
The we use EM algorithm for GMM(**Hint: here our K=3**) parameter estimation, and get the result as below:

```
print(gmm_model.means_)
print(gmm_model.covariances_)
```

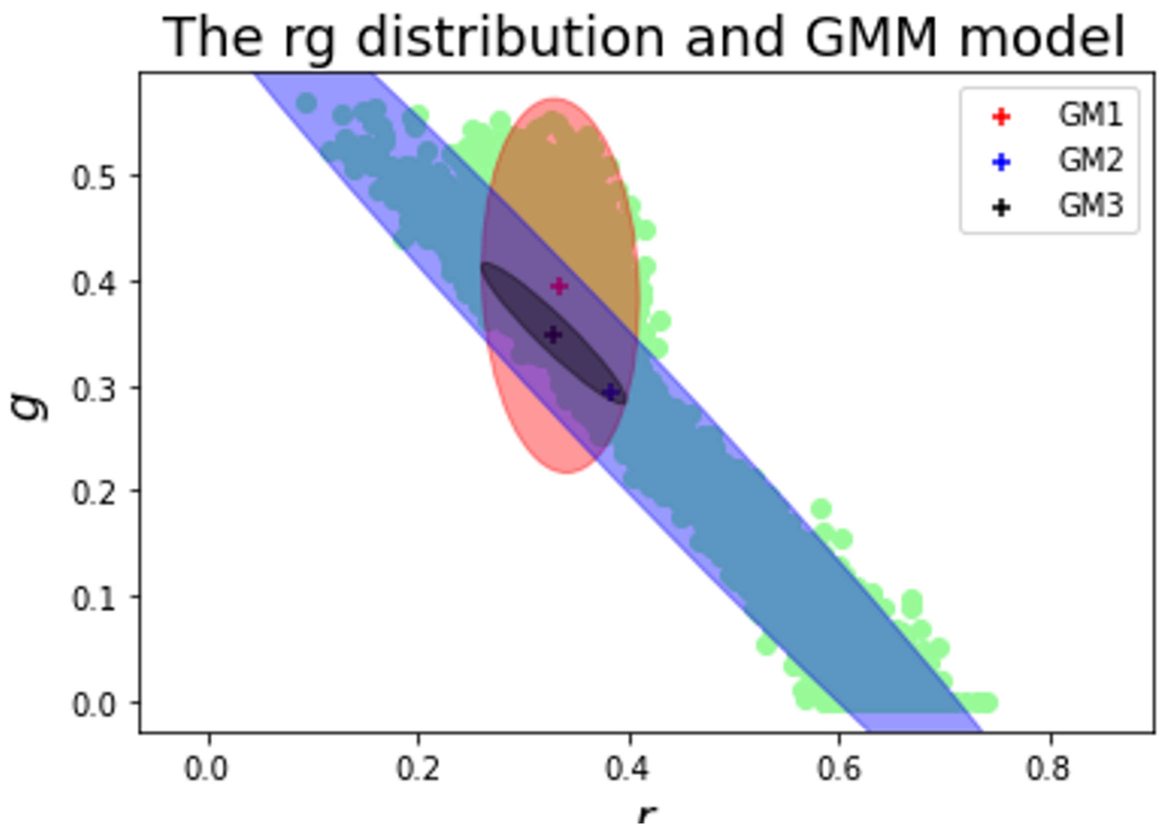
[38] ✓ 0.6s

```
[[0.33575619 0.39475245]
 [0.32960777 0.34918821]
 [0.38229313 0.29403256]]
[[[ 3.11266735e-04 -5.52412872e-05]
  [-5.52412872e-05  1.73977892e-03]]

 [[ 2.58347937e-04 -2.37331845e-04]
  [-2.37331845e-04  2.44114944e-04]]

 [[ 1.12898876e-02 -1.18171528e-02]
  [-1.18171528e-02  1.26964057e-02]]]
```

we now visualize the GMM model:



maybe the center maker is not very clear, so we zoom in:



for each GM, we could also get their alpha coefficient:

each GM(1 to 3)	weight
1	0.20421516
2	0.20442599
3	0.59135885

Now, we compute that for every pixel, the posterior probability that it came from the mixture component j , we based on this equation:

$$
\begin{aligned}
\gamma\left(z_k\right) &= p\left(z_k=1 \mid \boldsymbol{x}\right) \\
&= \frac{p\left(z_k=1\right) p\left(\boldsymbol{x} \mid z_k=1\right)}{p(\boldsymbol{x})} \\
&= \frac{p\left(z_k=1\right) p\left(\boldsymbol{x} \mid z_k=1\right)}{\sum_{j=1}^K p\left(z_j=1\right) p\left(\boldsymbol{x} \mid z_j=1\right)} \\
&= \frac{\pi_k \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}_{\boldsymbol{k}}, \boldsymbol{\Sigma}_{\boldsymbol{k}}\right)}{\sum_{j=1}^K \pi_j \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)}
\end{aligned} \tag{6}
$$

Hint: we use $\gamma\left(z_k\right)$ to represent the post probability of component k

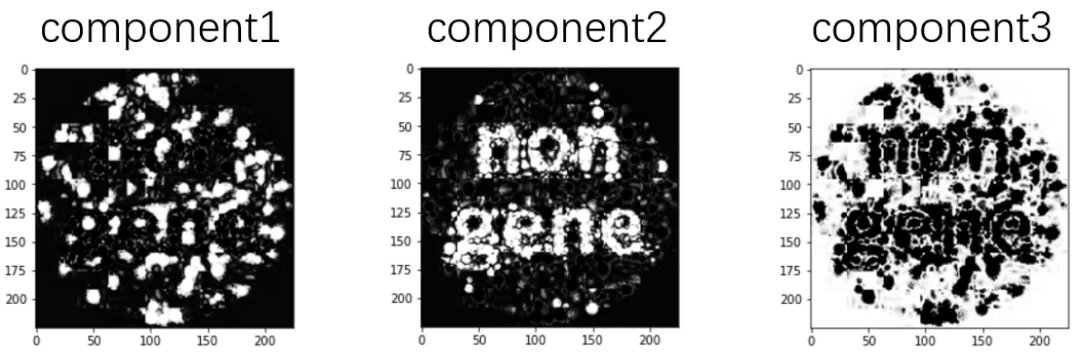
The result:

```
print(postProb.shape)
print(postProb)
[77]  ✓  0.8s
...  (50625, 3)
[[0.01953559 0.0183196  0.96214481]
 [0.01953559 0.0183196  0.96214481]
 [0.01953559 0.0183196  0.96214481]
 ...
 [0.01953559 0.0183196  0.96214481]
 [0.01953559 0.0183196  0.96214481]
 [0.01953559 0.0183196  0.96214481]]
```

for each component, we draw and display a normalized image, but firstly, we have to change the post probability to 8 bits gray-scale value, take component 1 for example:

```
component1_prob=(postProb.T[0]*255.0).reshape((-1,225))
component2_prob=(postProb.T[1]*255.0).reshape((-1,225))
component3_prob=(postProb.T[2]*255.0).reshape((-1,225))
print(component1_prob.shape)
print(component1_prob)
[86]  ✓  0.6s
...  (225, 225)
[[4.98157617 4.98157617 4.98157617 ... 4.98157617 4.98157617 4.98157617]
 [4.98157617 4.98157617 4.98157617 ... 4.98157617 4.98157617 4.98157617]
 [4.98157617 4.98157617 4.98157617 ... 4.98157617 4.98157617 4.98157617]
 ...
 [4.98157617 4.98157617 4.98157617 ... 4.98157617 4.98157617 4.98157617]
 [4.98157617 4.98157617 4.98157617 ... 4.98157617 4.98157617 4.98157617]
 [4.98157617 4.98157617 4.98157617 ... 4.98157617 4.98157617 4.98157617]]
+ 代码  + 标记
```

now, we could draw the gray image for each component:

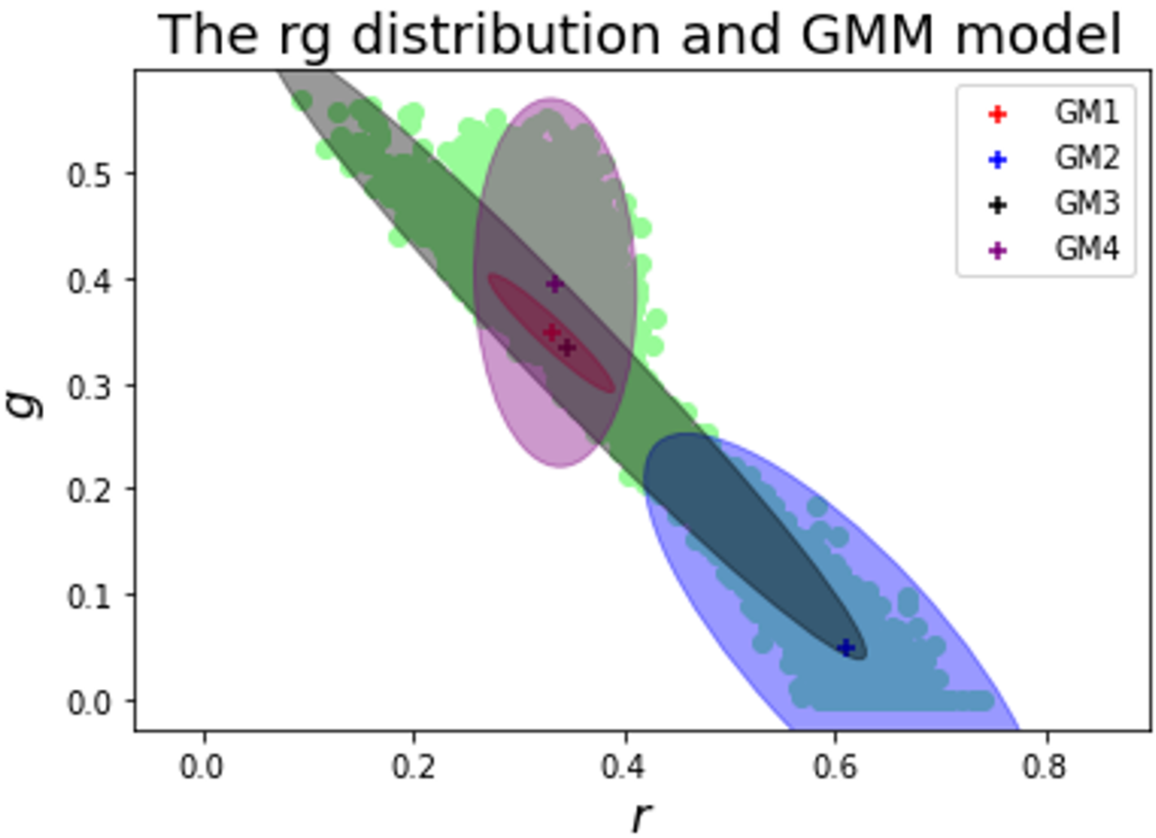


we could see that these 3 images are somewhat similar to the original image in some way. After superimposing their effects at each pixel, they will be very close to the original image. So, it is a wonderful way for us to use GMM to do image segmentation.

TASK C

K=4

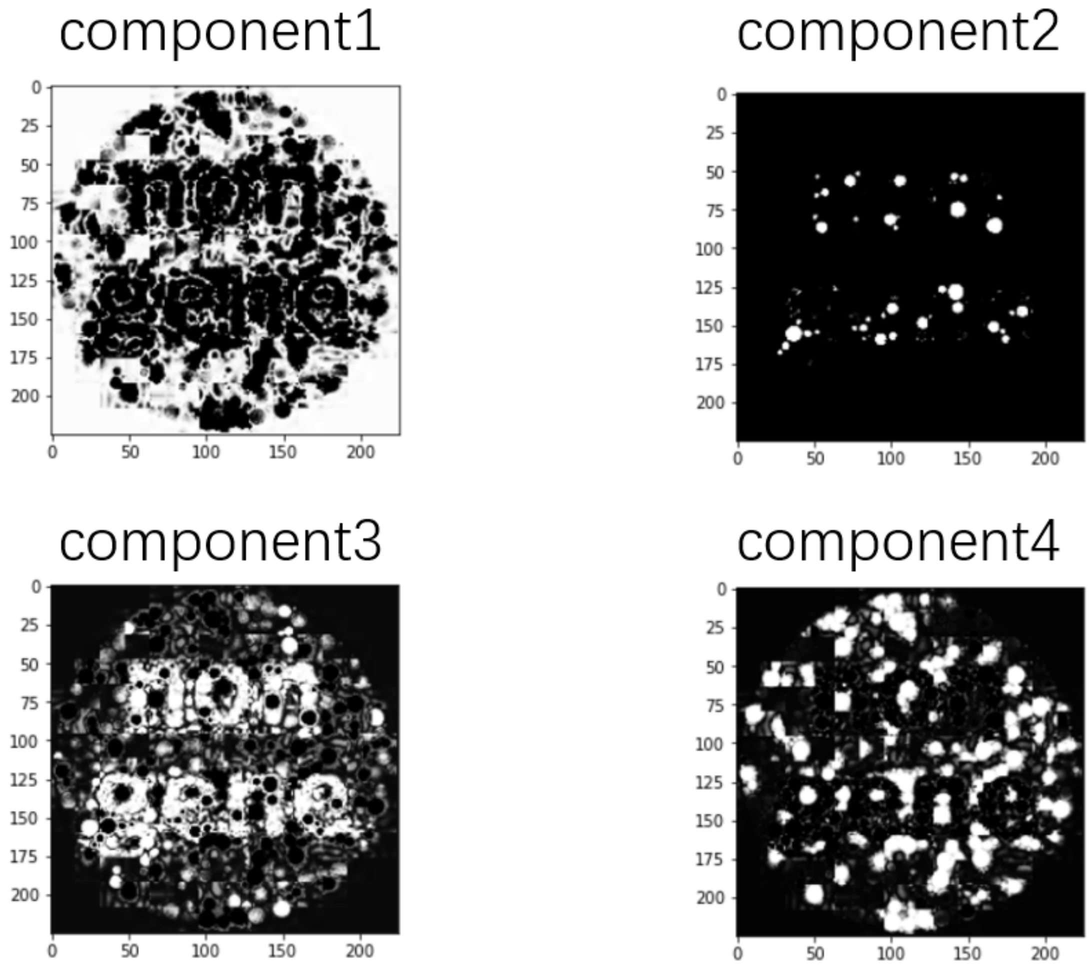
we repeat the above steps, just change the K's value to 4.



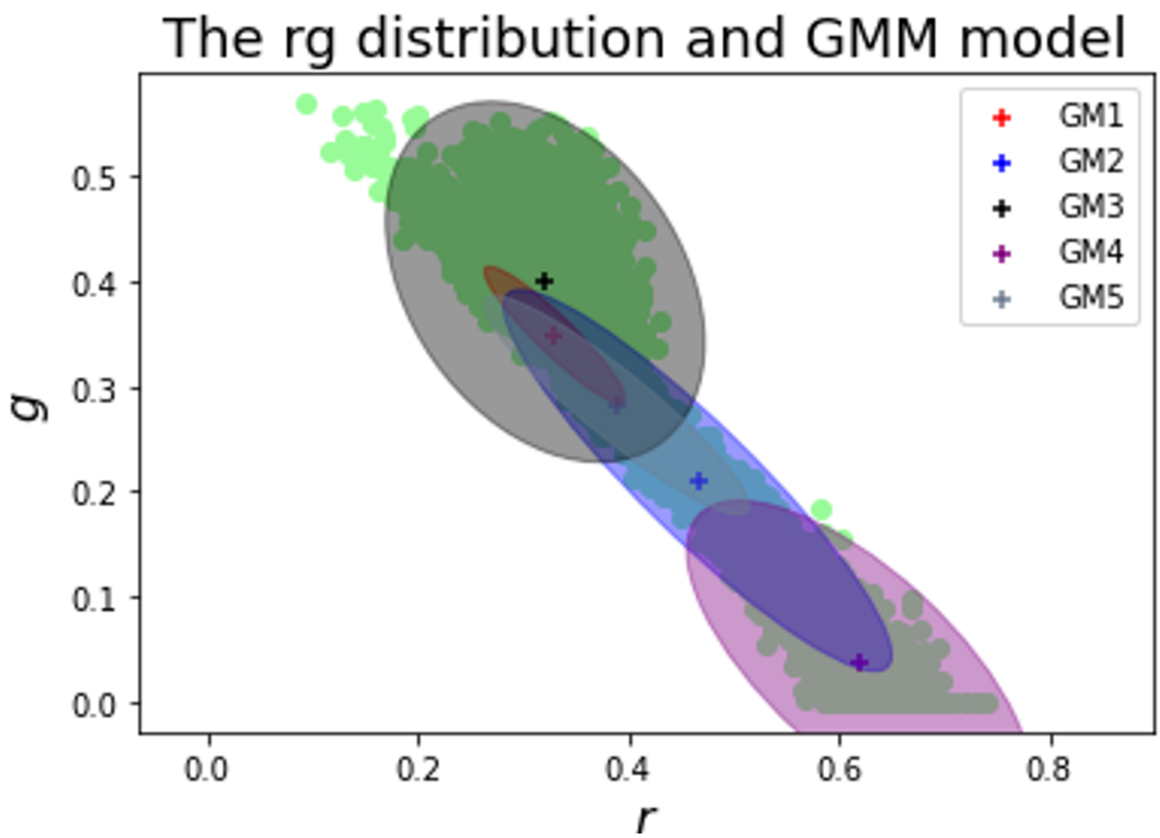

```
print(gmm_model.weights_)
```

[94] ✓ 0.3s

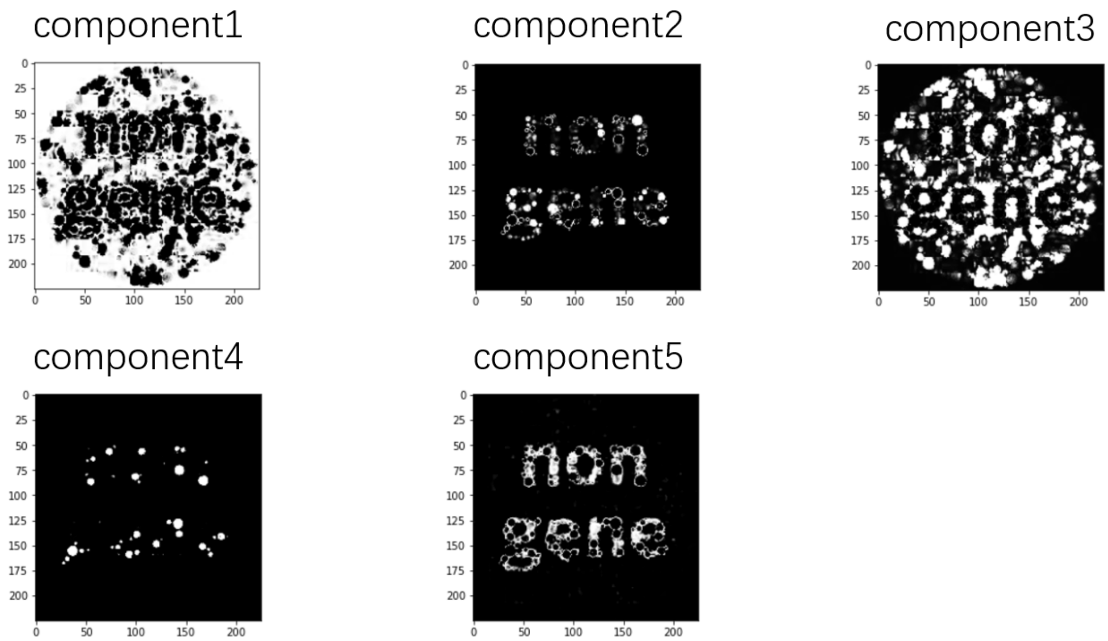
```
[0.51228649 0.02111221 0.26121783 0.20538348]
```


component1
component2
component3
component4

K=5


The rg distribution and GMM model
GM1
GM2
GM3
GM4
GM5
g
r

```
print(gmm_model.weights_)
[124]  ✓ 0.3s
... [0.62031007 0.03610493 0.26272721 0.01883558 0.0620222 ]
```



The above results show that: we could increase K value appropriately to make the GMM more similar to the original data distribution, that is replace any data distribution by the combination of many GMs. **This idea is just like the Calculus(replacing the curve by the straight), isn't it?**