

Práctica 2.

Clasificador Bayesiano.

31508811-1 Martíñón Luna Jonathan José.

31506842-1 Ortega Ibarra Jaime Jesus.

41800471-9 Tapia López José de Jesús.

I. OBJETIVO.

Clasificar imágenes con 2, 3 o 4 regiones utilizando el clasificador de Bayes.

II. INTRODUCCIÓN.

Para poder entender el Clasificador Bayesiano, primero debemos entender el concepto de clasificación, pues una clasificación consiste en asignar un objeto (instancia, dato) a una clase (categoría). Por ejemplo, podemos clasificar una imagen como paisaje, retrato, urbana, etc. Un Clasificador debe tener ciertas consideraciones.

1. Exactitud.
2. Rapidez
3. Claridad
4. Tiempo de aprendizaje

Una vez dado esto podemos abordar el Clasificado bayesiano, pues tenemos como definición inicial que El Clasificador Bayesiano es un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales, generalmente es utilizado en teorías de la probabilidad y minería de datos. Dicho clasificador asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra característica, dada la clase variable. Por ejemplo, una fruta puede ser considerada como una manzana si es roja, redonda y de alrededor de 7 cm de diámetro. Un clasificador de Bayes ingenuo considera que cada una de estas características contribuye de manera independiente a la probabilidad de que esta fruta sea una manzana, independientemente de la presencia o ausencia de las otras características. Este clasificador al tener una capacidad de generar categorías dependiendo de las características, es bastante utilizado en procesamiento y clasificación de imágenes, ya que como sabemos una imagen contiene regiones y cada una de estas regiones tienen ciertas características las cuales podemos clasificar y a su vez poder generar desde una detección de objetos en alguna imagen simple, hasta reconocimiento facial. Si C_i representa la clase i , y X es el vector de características extraído para el píxel cuya clase tiene que ser encontrada, entonces la probabilidad de que el píxel pertenezca a una clase particular está dada por

la probabilidad posterior $P(C_i|X)$. Entonces, la fórmula se basa en el teorema de Bayes:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}, \quad (1)$$

donde $P(C_i)$ es la probabilidad a priori de la clase C_i ; $P(X|C_i)$ es la probabilidad del vector de características X , dado que el píxel pertenece a la clase C_i ; y, $P(X)$ es la probabilidad total del vector de características X (es decir, $\sum_i P(X|C_i)P(C_i)$). Un clasificador bayesiano primero calcula $P(C_i|X)$ usando la ecuación anterior. Luego, el clasificador da la etiqueta C_m a un vector de características X_0 dado si $P(C_m|X_0)$ es máximo, es decir, $C_m = \operatorname{argmax}_i\{P(C_i|X)\}$. Las probabilidades anteriores $P(C_i)$, $P(X)$ y la probabilidad condicional $P(X|C_i)$ se calculan a partir de las imágenes etiquetadas; sin embargo, dado que el término $P(X)$ aparece para cualquier clase C_i , es constante para cualquier i , por lo que se omite.

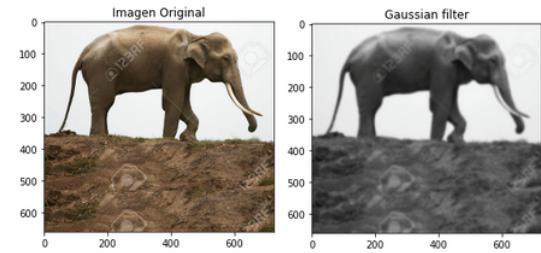
III. DESARROLLO.

Para llevar el desarrollo de dicha práctica, se divide en dos partes: *Entrenamiento* y *Prueba*, pero antes de iniciar, seleccionamos 3 imágenes, las cuales debían tener ciertas características para ser procesadas, es decir, nos fijamos en imágenes, que en nuestro caso son elefantes, en las cuales se pudiera observar distintas texturas u objetos, para posteriormente poder seleccionar dichas regiones. Por lo tanto, nuestras imágenes son las siguientes:





Una vez definidas nuestras imágenes, podemos comenzar con el desarrollo de la práctica.



```

1  imagen_3 = Image.open('elefante_3_2.png')
2  image3_gauss = ndimage.gaussian_filter(imagen_3, 3)
3  plt.imshow(imagen_3)
4  plt.title('Imagen Original')
5  plt.show()
6  plt.imshow(image3_gauss)
7  plt.title('Gaussian filter')
8  plt.show()

```

A. Entrenamiento.

- Realizar preprocessamiento de sus imágenes con un filtro gaussiano.

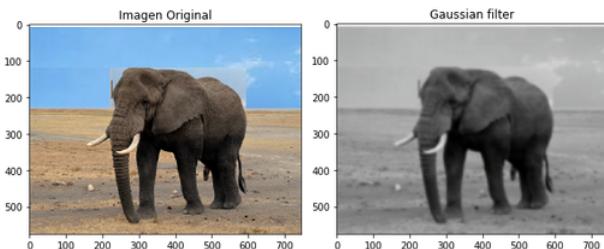
Para este punto de la práctica, hemos decidido utilizar el filtro gaussiano de la biblioteca `ndimage`, dando el siguiente código:

```

1 # Importamos las bibliotecas que usaremos
2 from scipy import misc
3 from scipy import ndimage
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import scipy
7 import cv2
8 from PIL import Image
9 imagen_1 = Image.open('elefante_1_2.png')
10 image1_gauss = ndimage.gaussian_filter(imagen_1, 3)
11 plt.imshow(imagen_1)
12 plt.title('Imagen Original')
13 plt.show()
14 plt.imshow(image1_gauss)
15 plt.title('Gaussian filter')
16 plt.show()

```

Dando así el siguiente resultado:

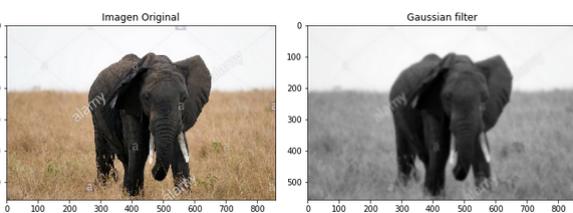


Repetimos dicho procedimiento para cada una de las imágenes seleccionadas:

```

1 imagen_2 = Image.open('elefante_2_2.png')
2 image2_gauss = ndimage.gaussian_filter(imagen_2, 3)
3 plt.imshow(imagen_2)
4 plt.title('Imagen Original')
5 plt.show()
6 plt.imshow(image2_gauss)
7 plt.title('Gaussian filter')
8 plt.show()

```



- Seleccionar sus imágenes con zonas aledañas a clasificar, verifique que tenga regiones contiguas.

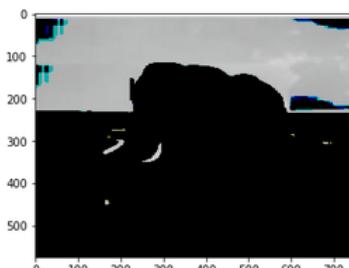
Para este punto, inicialmente identificamos mediante la propia visualización las distintas regiones que podíamos encontrar dentro de nuestras imágenes, posterior a ello comenzamos a analizar, pues cada región cuenta con ciertas características, una de ellas son los colores que la representan, por lo que decidimos generar ciertas iteraciones para poder distinguir cada una de nuestras regiones dentro de cada una de las imágenes, el código es el siguiente:

```

1 region_cielo = np.zeros((575,745,3))
2 for i in range(3):
3     for j in range(745):
4         for x in range(575):
5             if image1_gauss[x,j,i] >= 175 and
6                 image1_gauss[x,j,i] <= 255:
7                     region_cielo[x,j,i] = image1_gauss[x,
8                                     ,j,i]
9 plt.imshow(region_cielo/255)

```

Dáandonos así la primer región:



Posteriormente realizamos el mismo proceso pero con valores distintos para el resto de regiones.

```

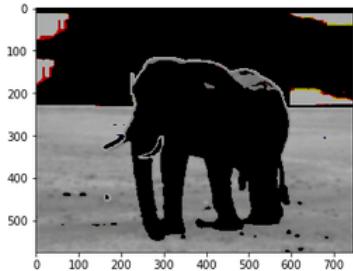
1 region_desierto = np.zeros((575,745,3))
2 for i in range(3):
3     for j in range(745):
4         for x in range(575):
5

```

```

5      if image1_gauss[x,j,i] >= 110 and
6      image1_gauss[x,j,i] <= 175:
7          region_desierto[x,j,i] =
8              image1_gauss[x,j,i]
9  plt.imshow(region_desierto/255)

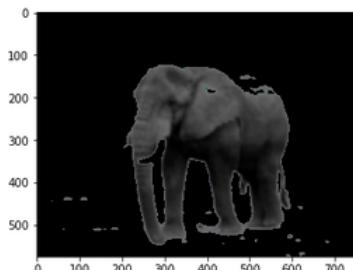
```



```

1 region_elefante = np.zeros((575,745,3))
2 for i in range(3):
3     for j in range(745):
4         for x in range(575):
5             if image1_gauss[x,j,i] >= 0 and
6                 image1_gauss[x,j,i] <= 110:
7                 region_elefante[x,j,i] =
8                     image1_gauss[x,j,i]
#Region_elefante contiene nicamente los valores
#que ilustran al animal, lo dem s son 0
#region_elefante
9 plt.imshow(region_elefante/255)

```

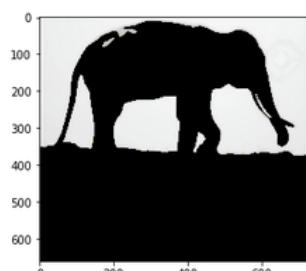


Y de igual manera repetimos para las 2 imágenes restantes:

```

1 region_cielo = np.zeros((661,726,3))
2 for i in range(3):
3     for j in range(726):
4         for x in range(661):
5             if image2_gauss[x,j,i] >= 190 and
6                 image2_gauss[x,j,i] <= 255:
7                 region_cielo[x,j,i] = image2_gauss[x,
8 ,j,i]
9 plt.imshow(region_cielo/255)

```



```

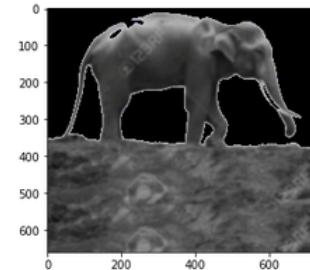
1 region_desierto = np.zeros((661,726,3))
2 for i in range(3):
3     for j in range(726):
4         for x in range(661):
5             if image2_gauss[x,j,i] >= 0 and
6                 image2_gauss[x,j,i] <= 190:

```

```

7      region_desierto[x,j,i] =
8          image2_gauss[x,j,i]
9  plt.imshow(region_desierto/255)

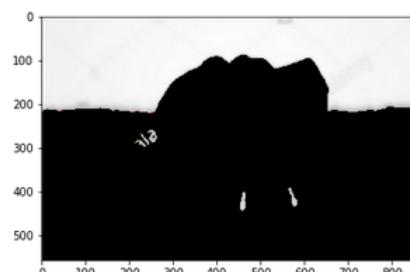
```



```

1 region_cielo = np.zeros((556,858,3))
2 for i in range(3):
3     for j in range(858):
4         for x in range(556):
5             if image3_gauss[x,j,i] >= 180 and
6                 image3_gauss[x,j,i] <= 255:
7                 region_cielo[x,j,i] = image3_gauss[x,
8 ,j,i]
9 plt.imshow(region_cielo/255)

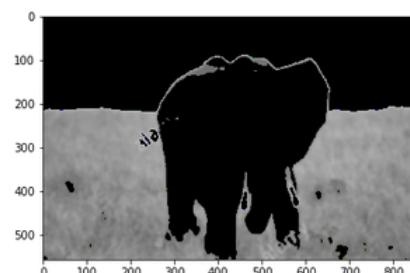
```



```

1 region_desierto = np.zeros((556,858,3))
2 for i in range(3):
3     for j in range(858):
4         for x in range(556):
5             if image3_gauss[x,j,i] >= 110 and
6                 image3_gauss[x,j,i] <= 180:
7                 region_desierto[x,j,i] =
8                     image3_gauss[x,j,i]
9 plt.imshow(region_desierto/255)

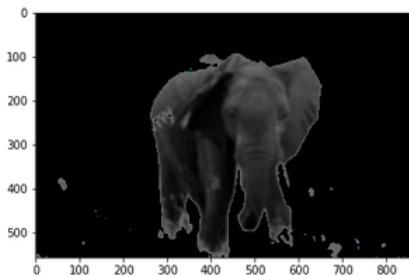
```



```

1 region_elefante = np.zeros((556,858,3))
2 for i in range(3):
3     for j in range(858):
4         for x in range(556):
5             if image3_gauss[x,j,i] >= 0 and
6                 image3_gauss[x,j,i] <= 110:
7                 region_elefante[x,j,i] =
8                     image3_gauss[x,j,i]
9 plt.imshow(region_elefante/255)

```

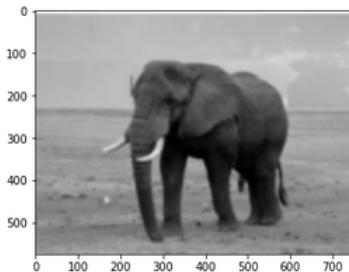


3) Genere sus propias máscaras de análisis para que sólo se quede con información de cada zona.

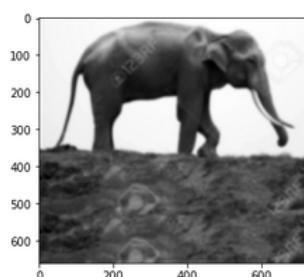
Como pudimos observar en el inciso anterior, las tres imágenes comparten tres características: el elefante, el suelo y el cielo; por lo que nuestras máscaras de análisis las hacemos sobre dichas características, y entonces son estas las clases que vamos a considerar para nuestro clasificador bayesiano.

Por lo tanto, para identificar con mejor claridad y facilitar la obtención de pixeles de cada clase, convertimos las imágenes con el filtro gaussiano a escala de grises:

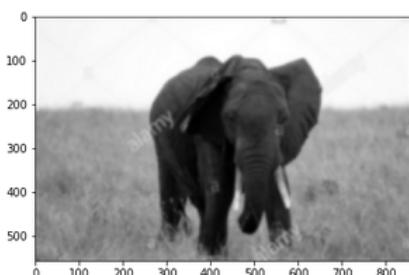
```
1 image1_gauss_gray = cv2.cvtColor(image1_gauss, cv2.
2     COLOR_BGR2GRAY)
3 plt.imshow(image1_gauss_gray, cmap = plt.cm.gray)
```



```
1 image2_gauss_gray = cv2.cvtColor(image2_gauss, cv2.
2     COLOR_BGR2GRAY)
3 plt.imshow(image2_gauss_gray, cmap = plt.cm.gray)
```

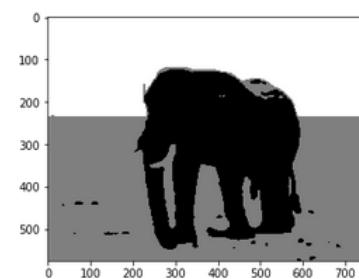


```
1 image3_gauss_gray = cv2.cvtColor(image3_gauss, cv2.
2     COLOR_BGR2GRAY)
3 plt.imshow(image3_gauss_gray, cmap = plt.cm.gray)
```



Posteriormente, realizamos varios ciclos, de tal forma que en cada imagen indicamos de color blanco a la clase 1, la cual corresponde al cielo; de color gris a la clase 2, la cual corresponde al suelo; y el color negro a la clase 3, el cual corresponde al elefante. En cada imagen, dado que difieren en sus dimensiones, fuimos ajustando a mano estos ciclos con la finalidad de obtener de la mejor manera posible las regiones (clases) en las máscaras de análisis. Para estos ajustes, nos apoyamos del cursor que obtenemos con "%matplotlib notebook", en el que podemos recorrer los pixeles de cada imagen y saber tanto su valor como su ubicación en cada una de las imágenes:

```
1 im1_blanco = 0
2 im1_gris = 0
3 im1_negro = 0
4
5 imagen1_gris = image1_gauss_gray.copy()
6
7 im1_c1 = []
8 im1_c2 = []
9 im1_c3 = []
10
11 for i in range(0,(235)):
12     for j in range(len(imagen1_gris[0])):
13         if imagen1_gris[i][j] > 150:
14             imagen1_gris[i][j] = 255
15             im1_blanco += 1
16             im1_c1.append(image1_gauss[i][j])
17
18 for i in range(len(imagen1_gris)):
19     for j in range(len(imagen1_gris[0])):
20         if imagen1_gris[i][j] <= 111:
21             imagen1_gris[i][j] = 0
22             im1_negro += 1
23             im1_c3.append(image1_gauss[i][j])
24
25         elif imagen1_gris[i][j] < 255 :
26             imagen1_gris[i][j] = 127
27             im1_gris += 1
28             im1_c2.append(image1_gauss[i][j])
29
30 #Probabilidad de pertenecer a cada una de las clases
31 # usando la medida de probabilidad clásica, es decir,
32 # esta es la probabilidad a priori de la imagen 1:
33
34 total = im1_blanco + im1_gris +im1_negro
35 # Estas son las prob. a priori de cada clase:
36
37 prob_blanco = im1_blanco/total
38 prob_gris = im1_gris/total
39 prob_negro = im1_negro/total
40
41
42 #%matplotlib notebook
43 plt.imshow(imagen1_gris, cmap = plt.cm.gray)
```



```
1 im2_blanco = 0
2 im2_gris = 0
3 im2_negro = 0
```

```

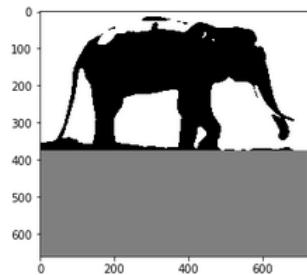
4  imagen2_gris = image2_gauss_gray.copy()
6  im2_c1 = []
8  im2_c2 = []
10 im2_c3 = []
11
12 for i in range(375,(661)):
13     for j in range(len(imagen2_gris[0])):
14         imagen2_gris[i][j] = 127
15         im2_gris += 1
16         im2_c2.append(image2_gauss[i][j])
17
18 for i in range(0,375):
19     for j in range(len(imagen2_gris[0])):
20         if imagen2_gris[i][j] < 160:
21             imagen2_gris[i][j] = 0
22             im2_negro += 1
23             im2_c3.append(image2_gauss[i][j])
24
25     else:
26         imagen2_gris[i][j] = 255
27         im2_blanco += 1
28         im2_c1.append(image2_gauss[i][j])
29
30 #Probabilidad de pertenecer a cada una de las clases
31 # usando la medida de probabilidad clasica, es decir,
32 # esta es la probabilidad a priori de la imagen 2:
33
34 total2 = im2_blanco + im2_gris +im2_negro
35 # Estas son las prob. a priori de cada clase:
36
37 prob_blanco2 = im2_blanco/total2
38 prob_gris2 = im2_gris/total2
40 prob_negro2 = im2_negro/total2

```

```

#%matplotlib notebook
plt.imshow(imagen2_gris, cmap = plt.cm.gray)

```



```

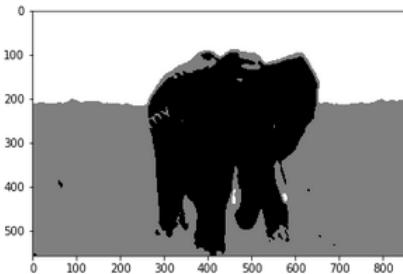
2 im3_blanco = 0
4 im3_gris = 0
6 im3_negro = 0
8
10 imagen3_gris = image3_gauss_gray.copy()
12 im3_c1 = []
14 im3_c2 = []
16 im3_c3 = []
18
20 for i in range(len(imagen3_gris)):
21     for j in range(len(imagen3_gris[0])):
22         if imagen3_gris[i][j] > 100 and imagen3_gris
23             [i][j] < 200:
24             imagen3_gris[i][j] = 127
25             im3_gris += 1
26             im3_c2.append(image3_gauss[i][j])
28
29 for i in range(len(imagen3_gris)):
30     for j in range(len(imagen3_gris[0])):
31         if imagen3_gris[i][j] >= 200:
32             imagen3_gris[i][j] = 255
33
34 im3_blanco += 1
35 im3_c1.append(image3_gauss[i][j])
37
38 elif imagen3_gris[i][j] <= 100:
39     imagen3_gris[i][j] = 0
40     im3_negro += 1
41     im3_c3.append(image3_gauss[i][j])
43
44 #Probabilidad de pertenecer a cada una de las clases
45 # usando la medida de probabilidad clasica, es decir,
46 # esta es la probabilidad a priori de la imagen 3:
47
48 total3 = im3_blanco + im3_gris +im3_negro
49 # Estas son las probabilidades a priori de cada clase:
50
51 prob_blanco3 = im3_blanco/total3
52 prob_gris3 = im3_gris/total3
53 prob_negro3 = im3_negro/total3

```

```

1 #%%matplotlib notebook
2 plt.imshow(imagen3_gris, cmap = plt.cm.gray)

```



Observamos que cada una de las máscaras son relativamente buenas. Además, al generar cada una de las máscaras de análisis de las tres imágenes, hemos utilizado listas vacías en las que en cada iteración les fuimos agregando la ubicación de los píxeles de la imagen en *RGB* con el filtro gaussiano para obtener información de los píxeles que corresponden a cada una de las clases en cada imagen. Esto es, nos basamos en las imágenes en escala de grises para facilitar la creación de las máscaras, pero realmente la información que nos interesa es el valor del píxel de cada clase de las imágenes con el filtro, y dicha información la vamos a requerir para calcular el vector de medias y las matrices de covarianzas de cada clase.

4) Implementar un clasificador bayesiano, obteniendo información a priori de las imágenes para las diferentes regiones de imagen.

Aprovechamos los ciclos anteriores para calcular las probabilidades a priori de cada clase en cada imagen. Nótese que es por eso que en cada ciclo **for** usamos contadores.

5) Desplegar en cada fase las imágenes y cálculos intermedios que apoyen el proceso, por ejemplo: cálculo de probabilidad de la región1, región 2, ... hasta la región n. Mostrar los resultados también para el cálculo de la media, matriz de covarianza etc.

Dado que son 3 clases y estamos obteniendo la información de los píxeles en *RGB*, vamos a obtener 3 vectores de medias de 3×1 y 3 matrices de covarianzas de 3×3 .

Realizamos las siguientes líneas de código:

```

print("De la mascara de la primera imagen, las 14
    probabilidades a priori de la clase 1 (blanco)
    ,\nde la clase 2 (gris) y de la clase 3 (negro)
    son respectivamente:\n", prob_blanco , prob_gris
    , prob_negro) 16
2 print("De la mascara de la segunda imagen, las 18
    probabilidades a priori de la clase 1 (blanco)
    ,\nde la clase 2 (gris) y de la clase 3 (negro)
    son respectivamente:\n", prob_blanco2 ,
    prob_gris2 , prob_negro2) 20
4 print("De la mascara de la tercera imagen, las 22
    probabilidades a priori de la clase 1 (blanco),\
        nde la clase 2 (gris) y de la clase 3 (negro)
    son respectivamente:\n", prob_blanco3 ,
    prob_gris3 , prob_negro3) 24
6 De la mascara de la primera imagen, las 26
    probabilidades a priori de la clase 1 (blanco),
    de la clase 2 (gris) y de la clase 3 (negro) son
    respectivamente: 28
10 0.3308666472133061 0.4009290925007295
    0.2682042602859644
12 De la mascara de la segunda imagen, las 30
    probabilidades a priori de la clase 1 (blanco),
    de la clase 2 (gris) y de la clase 3 (negro) son
    respectivamente: 32
14 0.3279362181851523 0.43267776096822996
    0.23938602084661773
16 De la mascara de la tercera imagen, las 34
    probabilidades a priori de la clase 1 (blanco),
    de la clase 2 (gris) y de la clase 3 (negro) son
    respectivamente: 36
20 0.2968380540322986 0.45282654994885213
    0.25033539601884924

```

Notamos que las imágenes mantienen proporciones similares de elefante, cielo y tierra.

```

2 # Aqui ya calculamos las probabilidades a priori
    finales:
4 priori_c1 = (prob_blanco + prob_blanco2 +
    prob_blanco3)/3
6 priori_c2=(prob_gris + prob_gris2 +prob_gris3)/3
priori_c3=(prob_negro + prob_negro2 + prob_negro3)/3
8 print("La probabilidad a priori de la clase 1, de la
    clase 2 y de la clase 3 son respectivamente:\n",
priori_c1 ,priori_c2 ,priori_c3) 10
12 La probabilidad a priori de la clase 1, de la clase
    2 y de la clase 3 son respectivamente: 14
14 0.31854697314358565 0.4288111344726038
    0.25264189238381046

```

En esta parte, dado que estamos trabajando con arreglos con numpy, para obtener la media y la matriz de covarianza, usamos lo métodos de **mean()** en de la biblioteca de NumPy para calcular el vector de medias y la matriz de covarianzas:

```

2 media_im1_c1 = np.mean(np.array(im1_c1), axis=0) 74
media_im1_c2 = np.mean(np.array(im1_c2), axis=0)
4 media_im1_c3 = np.mean(np.array(im1_c3), axis=0)
media_im2_c1 = np.mean(np.array(im2_c1), axis=0)
6 media_im2_c2 = np.mean(np.array(im2_c2), axis=0)
media_im2_c3 = np.mean(np.array(im2_c3), axis=0)
8 media_im3_c1 = np.mean(np.array(im3_c1), axis=0)
media_im3_c2 = np.mean(np.array(im3_c2), axis=0)
10 media_im3_c3 = np.mean(np.array(im3_c3), axis=0)
12 print("Del filtro gaussiano de la primera imagen, la 84
    media de la clase 1 (blanco),\nde la clase 2 (gris)
    y de la clase 3 (negro) son respectivamente:\n",
media_im1_c1 , media_im1_c2 , media_im1_c3)
14 print("Del filtro gaussiano de la segunda imagen, la
    media de la clase 1 (blanco),\nde la clase 2 (gris)
    y de la clase 3 (negro) son respectivamente:\n",
media_im2_c1 , media_im2_c2 , media_im2_c3)
16 print("Del filtro gaussiano de la tercera imagen,
    la media de la clase 1 (blanco),\nde la clase 2 (
        gris)
    y de la clase 3 (negro) son respectivamente:\n",
media_im3_c1 , media_im3_c2 , media_im3_c3)
18 Del filtro gaussiano de la primera imagen, la media
    de la clase 1 (blanco), de la clase 2 (gris) y de
    la clase 3 (negro) son respectivamente:
20 [185.62621089 186.28972378 186.51000106]
[144.6537194 144.63830729 144.32028321]
22 [57.54377154 57.40893187 57.23252272]
24 Del filtro gaussiano de la segunda imagen, la media
    de la clase 1 (blanco), de la clase 2 (gris) y de
    la clase 3 (negro) son respectivamente:
26 [231.22893526 231.83595557 231.77414661]
[83.92954979 83.74284324 83.58199927]
28 [77.57993698 77.56996988 77.19075889]
30 Del filtro gaussiano de la tercera imagen, la media
    de la clase 1 (blanco), de la clase 2 (gris) y de
    la clase 3 (negro) son respectivamente:
32 [244.66820615 244.79401297 244.78851179]
[143.88802426 143.84586149 143.55344413]
34 [46.43921555 46.42338933 46.24355646]
36 media_c1 = list((media_im1_c1 + media_im2_c1 +
    media_im2_c1)/3)
media_c2 = list((media_im1_c2 + media_im2_c2 +
    media_im2_c2)/3)
media_c3 = list((media_im1_c3 + media_im2_c3 +
    media_im2_c3)/3)
38 vector_medias = np.matrix([media_c1 ,media_c2 ,
    media_c3 ])
40 print("El vector de medias de la clase 1,de la clase
    2 y de la clase 3 son respectivamente:\n",
vector_medias[0],vector_medias[1],vector_medias[2])
42 El vector de medias de la clase 1,de la clase 2 y de
    la clase 3 son respectivamente:
44 [[216.02802714 216.65387831 216.68609809]]
[[104.17093966 104.04133126 103.82809391]]
[[70.90121517 70.84962388 70.5380135 ]]
46 # Elementos (pixeles) de la clase 1:
c1 = np.concatenate((im1_c1, im2_c1), axis=0)
c1 = np.concatenate((c1, im3_c1), axis=0)
c1 = np.matrix(c1)
48 # Elementos de la clase 2:
c2 = np.concatenate((im1_c2, im2_c2), axis=0)
c2 = np.concatenate((c2, im3_c2), axis=0)
c2 = np.matrix(c2)
50 # Elementos de la clase 3:
c3 = np.concatenate((im1_c3, im2_c3), axis=0)
c3 = np.concatenate((c3, im3_c3), axis=0)
c3 = np.matrix(c3)
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84

```

```

    transpose(),c1[i] - vector_medias[0])
cov1 = sum1/len(c1)
86
# matriz de covarianzas de la imagen 2:
88 for i in range(len(c2)):
    sum2 += np.dot((c2[i] - vector_medias[1]),
    transpose(),c2[i] - vector_medias[1]))
90 cov2 = sum2/len(c2)
92
# matriz de covarianzas de la imagen 3:
94 for i in range(len(c3)):
    sum3 += np.dot((c3[i] - vector_medias[2]),
    transpose(),c3[i] - vector_medias[2]))
96 cov3 = sum3/len(c3)
98 print("La matriz de covarianzas de la clase 1, de la
100 clase 2 y de la clase 3 son respectivamente:\n",
cov1,"\\n\\n",cov2,"\\n\\n",cov3)
102 La matriz de covarianzas de la clase 1, de la clase
104 2 y de la clase 3 son respectivamente:
[[852.77570431 849.19582215 847.06918462]
[849.19582215 845.8921179 843.74008163]
106 [847.06918462 843.74008163 841.68592231]]
108 [[1483.03705522 1485.91329876 1483.15526617]
[1485.91329876 1488.86994089 1486.08852647]
110 [1483.15526617 1486.08852647 1483.49837974]]
112 [[1162.95950946 1161.46133275 1160.32676961]
[1161.46133275 1160.0380855 1158.89044771]
114 [1160.32676961 1158.89044771 1157.93667121]]
116 Entonces, los valores que vas a ocupar para
    sustituir en la f rmula de la distribuci n
    normal multivariada son:
priori_c1, priori_c2, priori_c3, vector_medias,
cov1, cov2, cov3.

```

B. Prueba.

6) Una vez obtenido estos valores, clasificar los pixeles de la imagen con base en las probabilidades a priori obtenidas utilizando la aproximación gaussiana en la fórmula de Bayes(no olvide calcular la media, la matriz de covarianza y los cálculos necesarios para la clasificación).

En primer lugar, mostramos el código:

```

aux = a+b+c
Y.append(aux[0,0])
26
28
29 a = (-1/2) * (x - vector_medias[2].T).T
a *= np.linalg.inv(cov3)
a *= (x - vector_medias[2].T)
30
32 b = (-1/2) * log(np.linalg.det(cov3))
c = log(priori_c3)
33
34 aux = a+b+c
36 Y.append(aux[0,0])
38
40 return Y
42
def codifica(x):
44     if x == 1:
        return 0
46     elif x==2:
        return 128
48     elif x==3:
        return 255
50
def clasifica_pixel(name):
52
54 img = Image.open(name)
columnas, renglones = img.size
pixeles = img.load()
56
58
clas = np.zeros((columnas, renglones))
60
62
for i in range(columnas):
    for j in range(renglones):
64
        #Obtenemos el pixel a clasificar
        pixel = pixeles[i,j]
        print("Pixel:",i,j,"=",pixel)
66
68
        #Convertimos nuestro vector en columna
        x = np.array([[pixel[0]], [pixel[1]], [pixel[2]]])
69
71
        #clasificamos pixel por pixel
        clase = obten_prob(x)
72
74
        #Obtenemos la probabilidad mayor, como
        #nos da la posici n de
        # 0 ... n, nosotros no tenemos la clase
        # 0, sino 1,2 y 3, entonces
        # Simplemente le sumamos 1
        clase = clase.index(max(clase)) + 1
76
78
        if clase != 1 and clase != 3 and clase!=
2:
            print("ERROR en el pixel",pixel,"i =
",i,"j = ",j)
80
82
        #Guardamos la clase a la que pertenece
        clas[i,j] = codifica(clase)
84
86
        # Al final solo queda devolver los valores de
        # cada pixel
88
return clas.T

```

Para realizar esta sección, se hizo uso de los valores anteriormente calculados (Σ ,prior's, μ). Dentro de nuestra práctica trabajamos en la identificación de 3 regiones (Cielo, tierra, elefante), lo que nos llevó a calcular 3 probabilidades por

```

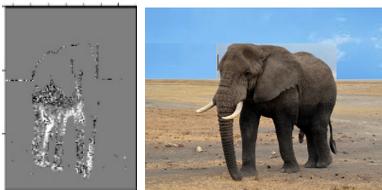
def obten_prob(x):
2
    g = []
    Y = []
4
6
    a = (-1/2) * (x - vector_medias[0].T).T
    a *= np.linalg.inv(cov1)
    a *= (x - vector_medias[0].T)
8
10
    b = (-1/2) * log(np.linalg.det(cov1))
    c = log(priori_c1)
12
14
    aux = a+b+c
16
    Y.append(aux[0,0])
18
19
    a = (-1/2) * (x - vector_medias[1].T).T
    a *= np.linalg.inv(cov2)
    a *= (x - vector_medias[1].T)
20
22
    b = (-1/2) * log(np.linalg.det(cov2))
    c = log(priori_c2)
24

```

cada una de los posibles pixeles. En un inicio optamos por la fórmula:

$$\ln(Y_k) = \ln(G_n) + \ln(P(C_k))$$

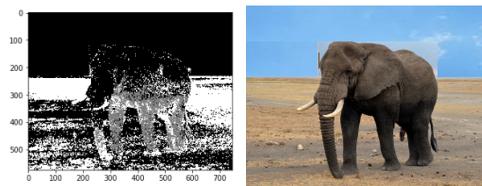
Y curiosamente nuestros desempeños fueron terribles:



Lo que se observa en la imagen anterior es la clasificación de pixeles de la imagen a color, como podemos observar no es lo mejor. Teniendo en cuenta estos resultados, optamos por cambiar la fórmula, por la última presentada en el *PDF* que se nos hizo favor de proporcionar:

$$Y_k(x) = -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k) - \ln |\Sigma_k| + \ln[P(C_k)]$$

Lo cual resultó en una notable mejora:



Como podemos observar, si bien es cierto, no define completamente los bordes, no tiene una buena *noción* de los límites, se acercó mucho más que el modelo anterior. Pensamos que la falta de precisión se debía a la falta de imágenes o que simplemente poseían muchas diferencias entre ellas. Tal vez debido a que las posturas de los elefantes en las tres imágenes es distinta, y que el cielo en dos imágenes parece estar nublado, mientras que en una está despejado.

```
# Pobando en la imagen 1
2 Class_Image_1 = clasifica_pixel('elefante_1_2.png')
3 plt.imshow(Class_Image_1, cmap='gray')

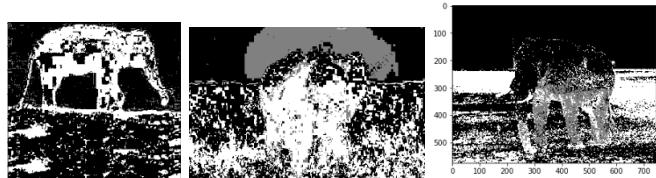
#Probando en la imagen 2
6 Class_Image_2 = clasifica_pixel('elefante_2_2.png')
7 plt.imshow(Class_Image_2, cmap='gray')

#Probando imagen 3
10 Class_Image_3 = clasifica_pixel('elefante_3_2.png')
11 plt.imshow(Class_Image_3, cmap='gray')

#Decidimos usar otra imagen, externa, para ver el
#rendimiento de nuestro clasificador
14 prueba = clasifica_pixel('prueba.jpeg')
15 plt.imshow(prueba, cmap='gray')
```

7) Cree una nueva imagen con las clases resultantes y asigne diferentes valores de gris para cada región, por ejemplo, en una imagen de 3 regiones (fondo, halo y objeto de interés) sería: 0 para el fondo, 128 para el halo y 250 para el objeto de interés. Despliegue sus resultados y verifique que tanto se acercó a lo esperado.

Para este punto aprovechamos el algoritmo de la clasificación anterior, pues decidimos que en el momento en el que se clasificara el pixel (0,128,255), se añadiese con el valor relacionado a su clasificación en una nueva matriz, para así poder mostrarla como imagen en un futuro, teniendo en cuenta que los valores se daban en escala de grises, no hubo problema en las dimensiones de la matriz, pues únicamente fue bidimensional para almacenar el valor de clasificación en la posición (i,j) de cada pixel, posteriormente procedimos a mostrar los arreglos como imágenes y obtuvimos las siguientes:



Como podemos observar, no fue muy bueno para observar los tonos grises, la única imagen que lo logró fue la de en medio y a pesar de ello cuesta trabajo identificar al elefante en el centro. Por curiosidad, pensamos en aplicar el algoritmo a una imagen que no tuviesemos en un principio, los resultados no fueron tan frustrantes, nuevamente tuvimos problemas con el gris, pero fuera de eso tuvo un desempeño regular.



8) Utilice la función del clasificador de Bayes de scikit learn. Averigüe como debe de utilizarlo para que pueda introducir sus datos que generó anteriormente, es posible que haya cambios. Compare sus resultados contra los anteriores.

De acuerdo con lo que teníamos entendido el modelo funcionaba en cuanto a que dadas ciertas características llegar a un resultado. Relativamente teníamos dudas, en cuanto a los datos a usar, finalmente decidimos recabar de cada uno de los pixeles de las 3 imágenes usadas en un inicio, obtener sus valores RGB, y, puesto que ya teníamos la clasificación, decidimos juntarlo en un dataset de la siguiente forma:

R	G	B	Class
0	255	255	0.0
1	253	253	0.0
2	250	252	0.0
3	255	255	0.0
4	255	255	0.0
...
1385304	139	104	64.0
1385305	140	105	65.0
1385306	146	111	71.0
1385307	146	111	71.0
1385308	175	150	109.0
			255.0

1385309 rows × 4 columns

Juntamos un total de 1,385,309 de datos, puesto que era el total de pixeles presentes en las 3 imágenes, era un buen número, decidimos utilizar 70% para entrenar y 30% para probar, a

decir verdad, la evaluación de nuestro modelo no hizo más que confirmar nuestras sospechas, pues era realmente pésimo para detectar *grises*, otra opción que pensamos fue tomar datos en lugar de escala RGB, en escala de grises, son menos datos y en apariencia podría no ser significativo, podría apareantar incluso un menor desempeño, sin embargo, logró levantar un poco los 2 aspectos restantes (Tampoco pudo con los *grises*).

El dataset quedó de la siguiente forma:

X	Class
0	255
1	253
2	251
3	255
4	255
...	...
1385304	109
1385305	110
1385306	116
1385307	116
1385308	152
1385309	255

1385309 rows × 2 columns

```

1 from sklearn.naive_bayes import GaussianNB
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4
5 model = GaussianNB()
6
7 #Creamos nuestras listas del dataset
8 img = Image.open('elefante_1_2.png')
9 columnas, renglones = img.size
10 pixeles = img.load()
11 Cl = Class_Image_1.T.copy()
12 R=[]
13 G=[]
14 B=[]
15 Clas = []
16 for i in range(columnas):
17     for j in range(renglones):
18
19         pixel = pixeles[i,j]
20
21         R.append(pixel[0])
22         G.append(pixel[1])
23         B.append(pixel[2])
24         Clas.append(Cl[i,j])
25
26 img = Image.open('elefante_2_2.png')
27 columnas, renglones = img.size
28 pixeles = img.load()
29 Cl = Class_Image_2.T.copy()
30 for i in range(columnas):
31     for j in range(renglones):
32
33         pixel = pixeles[i,j]
34
35         R.append(pixel[0])
36         G.append(pixel[1])
37         B.append(pixel[2])
38         Clas.append(Cl[i,j])
39
40 img = Image.open('elefante_3_2.png')
41 columnas, renglones = img.size
42 pixeles = img.load()
43 Cl = Class_Image_3.T.copy()
44 for i in range(columnas):
45     for j in range(renglones):
46
47
57
58         pixel = pixeles[i,j]
59         R.append(pixel[0])
60         G.append(pixel[1])
61         B.append(pixel[2])
62         Clas.append(Cl[i,j])
63
64 # Una vez que contamos con las listas, podemos crear
65 # un dataframe
66 data = pd.DataFrame({ 'R':R, 'G':G, 'B':B, 'Class':Clas })
67
68 #Observamos la distribución de nuestras clases
69 print(data.groupby('Class').size())
70
71 #De acuerdo con los datos obtenidos, mostramos los
72 #porcentajes de distribución
73 tot = 854078 + 113001 + 418230
74 ceros = 854078/tot
75 medio = 113001/tot
76 negro = 418230 /tot
77 print("Proporciones en los datos")
78 print("Clase 1 =",round(ceros*100,3),"%")
79 print("Clase 2 =",round(medio*100,3),"%")
80 print("Clase 3 =",round(negro*100,3),"%")
81
82 #Mostramos los datos
83 data.head(10)
84
85 #Estimamos X y Y
86 X = data.iloc[:, :3].values
87 Y = data.iloc[:, -3].values
88
89 #Creamos datos de prueba y entrenamiento
90 X_train, X_test, Y_train, Y_test = train_test_split(
91     X, y, test_size = 0.3)
92
93 #Entrenamos el modelo
94 model.fit(X_train, Y_train)
95
96 #Generamos predicciones
97 y_pred = model.predict(X_test)
98
99 #Observamos el reporte
100 from sklearn.metrics import classification_report
101 report = classification_report(Y_test, y_pred)
102 print(report)

```

Ahora, bien, aplicado a la escala de grises, los pasos son prácticamente los mismos, el código es el siguiente

```

1
2 #Creamos una función encargada de añadir valores a
3 #una lista
4 def aade_datos(name):
5     im_aux1 = Image.open(name).convert('L')
6     columnas, renglones = im_aux1.size
7     pixeles = im_aux1.load()
8
9     aux = []
10
11    for i in range(columnas):
12        for j in range(renglones):
13            aux.append(pixeles[i,j])
14
15    return aux
16
17 #Añadimos los datos
18 X = aade_datos('elefante_1_2.png')
19 X += aade_datos('elefante_2_2.png')
20 X += aade_datos('elefante_3_2.png')
21
22 #Creamos el nuevo data frame
23 new_data = pd.DataFrame({ 'X':X, 'Class':Clas })
24
25 #Verificamos la existencia de datos de 0 – 255

```

```

25 print(new_data.groupby('X').size())
27 #Seleccionamos nuestros valores X, Y
28 X = new_data.iloc[:,0].values
29 Y = new_data.iloc[:,1].values
31 #Creamos nuestros conjuntos de entrenamiento y prueba
32 X_train, X_test, Y_train, Y_test = train_test_split(
33     X, Y, test_size = 0.25)
35 #Entrenamos el modelo
36 model = GaussianNB()
37 model.fit(np.reshape(X_train, (-1, 1)), Y_train)
39 #Aplicamos el modelo
40 y_pred = model.predict(np.reshape(X_test, (-1, 1)))
41 #Generamos el reporte del modelo con escala de grises
42 report = classification_report(Y_test, y_pred)
43 print(report)

```

IV. RESULTADOS.

Finalmente pudimos obtener el desempeño de los 2 modelos utilizados (Tomando en cuenta RGB y tomando en cuenta Escala de grises):

En escala de grises

	precision	recall	f1-score	support
0.0	0.69	0.86	0.77	213513
128.0	0.00	0.00	0.00	28243
255.0	0.51	0.40	0.45	104572
accuracy			0.65	346328
macro avg	0.40	0.42	0.41	346328
weighted avg	0.58	0.65	0.61	346328

En escala RGB

	precision	recall	f1-score	support
0.0	0.68	0.65	0.66	256500
128.0	0.00	0.00	0.00	33843
255.0	0.40	0.55	0.47	125250
accuracy			0.57	415593
macro avg	0.36	0.40	0.38	415593
weighted avg	0.54	0.57	0.55	415593

V. CONCLUSIONES.

Una imagen posterior a ser procesada, también puede ser clasificada y si dicha clasificación es realmente correcta, podemos generar ciertos algoritmos tales como reconocimientos de objetos, reconocimiento facial, detección de imágenes o simplemente clasificar con respecto a la similitud entre sus características y el Clasificador Bayesiano es fundamental en dichas clasificaciones. Sin embargo es importante mencionar que se debe establecer un mínimo de datos para entrenar nuestro algoritmo, en apariencia al ojo humano resulta sencillo (en la mayoría de los casos) identificar, discriminar y clasificar

imágenes, sin embargo, esto precisamente se debe a la gran exposición de imágenes similares a las que se han encontrado, podemos ver nuestro modelo como un bebé, el cual tiene un mundo completamente desconocido y que con el paso del tiempo va captando miles de imágenes para poder así llegar a un reconocimiento, lo mismo podemos verlo a nuestro modelo, entre mayores imágenes le sean proporcionadas para aprender (entrenar) podemos llegar a obtener un buen desempeño. Por otro lado, tal vez el desempeño del clasificador no fue el mejor debido a que las máscaras de análisis las forzamos, o que la posición de los elefantes es distinta en cada imagen. Sin embargo, el clasificador logró detectar a los elefantes en algunas imágenes, y tal vez podríamos mejorar el análisis con imágenes mucho más similares, que con las que escogimos.

VI. REFERENCIAS.

- <http://lapi.fi-p.unam.mx/wp-content/uploads/Clasificador-Bayes-Distribucion-Normal.png>
- <http://www.isaet.org/images/extraimages/P1216004.pdf>
- <https://arxiv.org/pdf/1204.1631.pdf>
- <https://pybonacci.org/2012/06/07/algebra-lineal-en-python-con-numpy-i-operaciones-basicas/>
- <https://docs.scipy.org/doc/scipy/reference/generated/py.ndimage.gaussian.html>
- <https://ccc.inaoep.mx/esucar/Clases-mgp/pgm06-clasif-2012.pdf>
- <http://www.kovan.ceng.metu.edu.tr/sinan/publications/dibyenduSCIA.pdf>
- <http://lineadecodigo.com/python/concatenar-listas-en-python/>
- https://scikit-learn.org/stable/modules/naive_bayes.html