Práctica 2: Clasificador Bayesiano

Serapio Hernández Alexis Arturo
Facultad de Ingeniería
UNAM
Ciudad de México, México
alexis.serapio@ingenieria.unam.

García López Erik
Facultad de Ingeniería
UNAM
Ciudad de México, México
erikpumas999@gmail.com

Reyes Herrera Rogelio
Facultad de Ingeniería
UNAM
Ciudad de México, México
masterfive5of@gmail.com

***Abstract*—This article is about the making of a practical document at the Faculty of Engineering at UNAM, it is about the Bayesian Classifier that its a statistical model that uses the Bayes theorem to realize classifications. This kind of classifier its based in the conditional probability and its commonly used in the machine learning and pattern recognition.**

Keywords—Basic Images Processing, Python, Pattern, Pattern Recognition, Bayes, Bayes theorem, classifier.

I. Objetivos

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

II. Introducción

Un clasificador bayesiano es un método de aprendizaje automático basado en el **teorema de bayes** que utiliza la probabilidad condicional para predecir la clase a la que pertenece una muestra.

Este tipo de clasificador asume que las características son independientes entre sí, lo que simplifica el cálculo de la probabilidad de que una muestra pertenezca a cada clase. En el contexto del procesamiento de imágenes, los clasificadores bayesianos son útiles para tareas como la detección de objetos, la segmentación de imágenes y el reconocimiento facial.

Por ejemplo, se puede utilizar un clasificador naive bayes para clasificar píxeles en imágenes, como diferenciar entre piel y no puel en imagenes de rostro, analizando los valores de color RGB y aplicando un enfoque probabilistico para determinar la clase más probable para cada píxel. Este método ha demostrado buenos resultados en aplicaciones de visión computacional gracias a su eficiencia y capacidad para manejar datos multidimensionales.

Vamos a profundizar un poco más en el **clasificador de bayes: naive bayes.**

Naive bayes es un algoritmo de aprendizaje automático supervisado que se basa en el teorema de bayes. Este teorema permite calcular la probabilidad de que un evento ocurra dado que otro evento ha ocurrido, utilizando probabilidades condicionales. En el contexto de la clasificación, se asume que las características son independientes entre sí, lo que simplifica los cálculos necesarios para determinar la clase a la que pertenece una instancia.

La fórmula básica del teorema se expresa como:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

donde P(A|B) es la probabilidad posterior, P(B|A) es la probabilidad de observar B dado A, P(A) es la probabilidad previa y P(B) es la probabilidad marginal.

Este clasificador Naive Bayes asume que cada característica contribuye de manera independiente de la probabilidad total, lo que simplifica el modelo y lo hace computacionalmente eficiente.

Un estudio propuso un método para clasificar píxeles en imágenes basándose en su color. Se utilizó análisis de componentes principales (PCA) para reducir la dimensionalidad y mejorar la representatividad de las características antes de aplicar el clasificador bayesiano.

Funciona bien con atributos categóricos y numéricos, siendo útil en escenarios donde las características son numerosas.

La suposición de que las características son independientes puede no ser válida en todos los casos, lo que podría afectar la precisión del modelo.

III. DESARROLLO PRÁCTICA 1

3.1.- Realizar el preprocesamiento de sus imágenes con un filtro gaussiano.

En el preprocesamiento se realizó la suavización de las imágenes aplicando un filtro gaussiano. Inicialmente, se intentó con la función *filters.gaussian* de skimage, pero esta opción distorsionaba los colores, lo cual no era deseado. Finalmente, se optó por usar *ImageFilter.GaussianBlur(radius=2)* de *PIL*, que funcionó mejor preservando los colores originales, lo que facilitó el

análisis posterior. Es indispensable hacer esto ya que permite eliminar parte del ruido visual en las imágenes

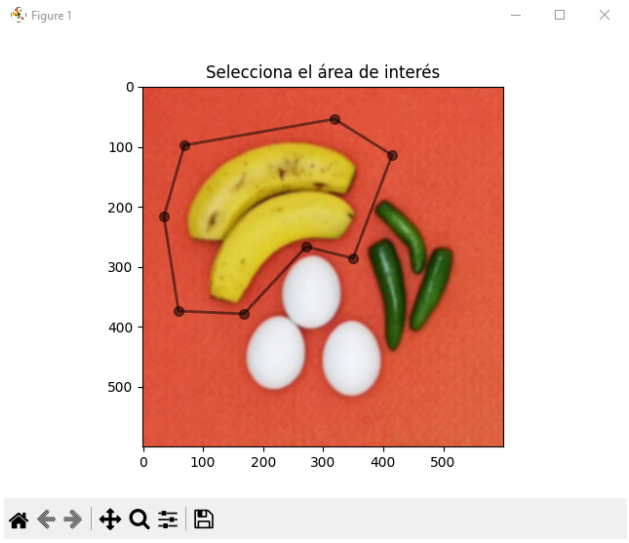
```
image_pil = Image.fromarray(image)
blurred_image = image_pil.filter(ImageFilter.GaussianBlur(radius=4))
blurred_images.append(np.array(blurred_image))
```

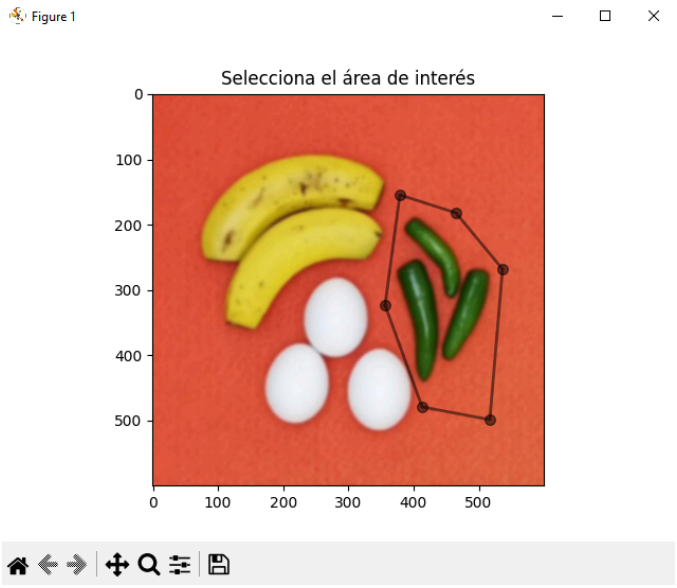
3.2.- Seleccionar sus imágenes con las regiones a clasificar.

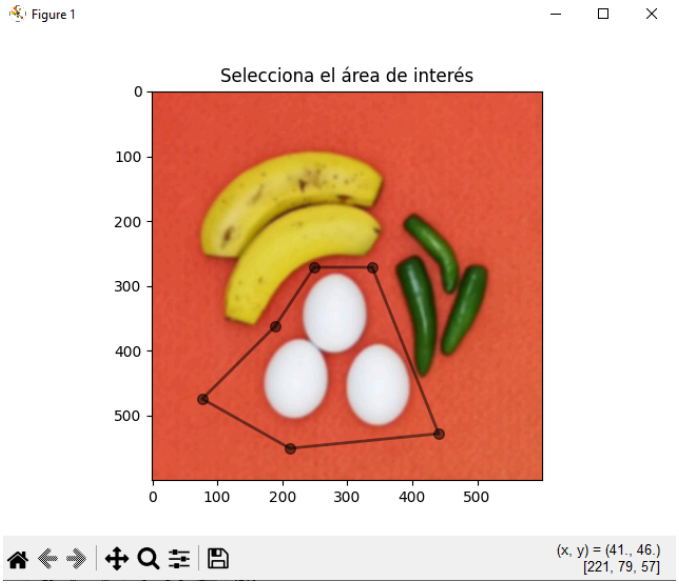
Para la selección de regiones a clasificar, se empleó una interfaz gráfica utilizando matplotlib con el selector de polígonos (PolygonSelector). Después de seleccionar los píxeles de estas regiones, se generan máscaras que asignan un valor de **1 a los píxeles seleccionados y 0 al resto**. Posteriormente, estos píxeles se procesan para eliminar aquellos que están cerca de los colores del fondo, quedándose únicamente con los píxeles relevantes para la clase seleccionada. El bucle for facilita la selección de tres clases diferentes (plátano, chile y huevo), y el fondo se procesó en un programa separado que lo elimina de la imagen

```
ax.set_title("Selecciona el área de interés")
selector = PolygonSelector(ax, on_select)
```

Ejemplos:





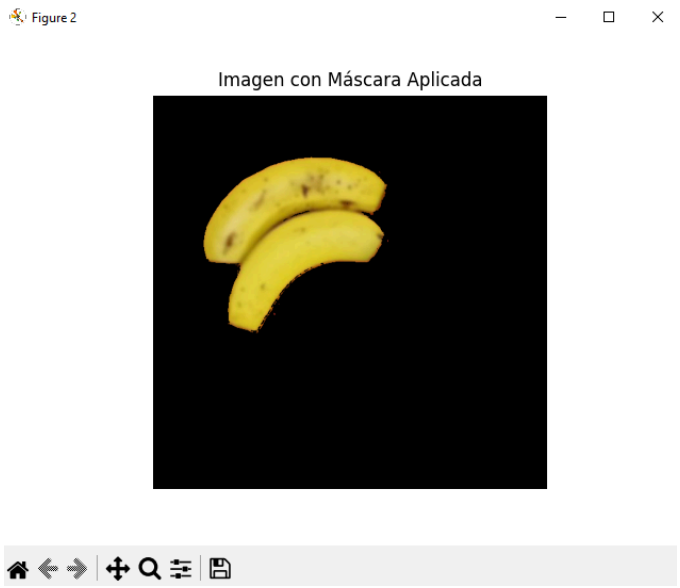


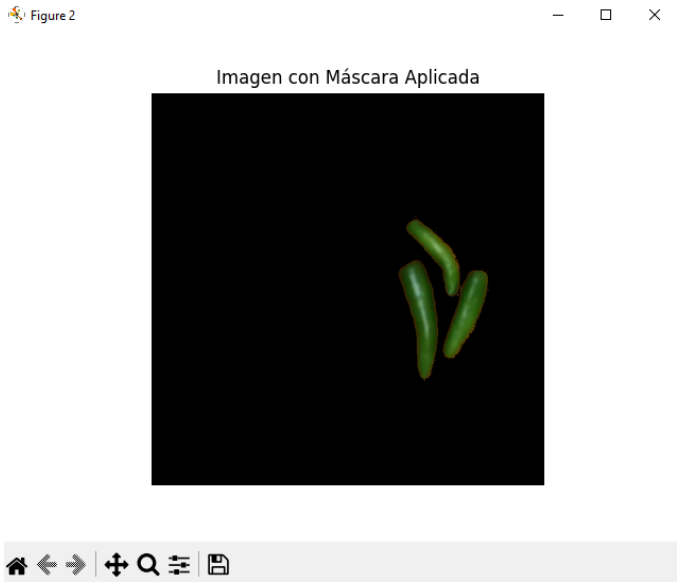
3.3.- Genere sus propias máscaras de análisis para que sólo se quede con información de cada zona.

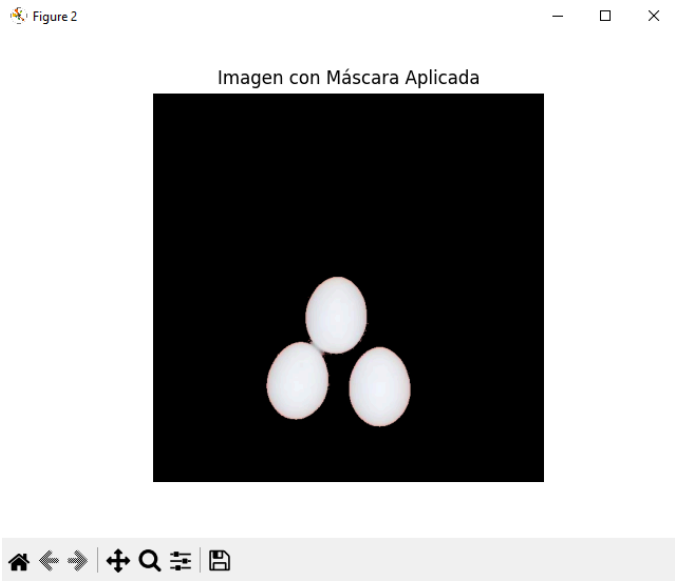
Los píxeles que coinciden con los colores del fondo son descartados, dejando únicamente los píxeles válidos de la clase seleccionada. Este proceso resulta rápido y eficaz para calcular la media y la matriz de covarianza de los píxeles seleccionados, lo cual es fundamental para el posterior cálculo de las probabilidades a priori. Durante este proceso, se acumula la cantidad de píxeles válidos, lo que permite calcular las probabilidades de fondo dividiendo el total de píxeles seleccionados entre el total de la imagen

```
distancias = np.array([np.linalg.norm(pixeles_seleccionados[:, :3] - color, axis=1) for color in colores_excluir])
mascara_validos = np.all(distancias > umbral_distancia, axis=0)
```

Ejemplos:







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

Recordando que la fórmula del clasificador bayesiano es:

$$Y_k(X) = \frac{-1}{2}(X - \boldsymbol{\mu_k})^T {S_k}^{-1}(X - \boldsymbol{\mu_k}) - \frac{1}{2} ln|S_k| + lnP(C_k)$$

El clasificador bayesiano se implementa utilizando las medias y matrices de covarianza obtenidas de las regiones seleccionadas. Los cálculos a priori se realizan sumando los píxeles válidos de cada región y dividiéndolos entre el total de píxeles de la imagen. Esto permite calcular las probabilidades de las clases seleccionadas. Con estos datos, se inicializa una nueva imagen en escala de grises donde los píxeles de las diferentes regiones se pintan en diferentes tonos de gris dependiendo de la clase a la que pertenecen

```
Y_platano = calcular_Yk(pixel, banmed, banvar, priori[0])
Y_chiles = calcular_Yk(pixel, chmed, chvar, priori[1])
Y_huevo = calcular_Yk(pixel, medias[4], medias[5], priori[2])
Y_fondo = calcular_Yk(pixel, fondomed, fondovar, priori[3])
```

3.5.- Desplegar en cada fase las imágenes y cálculos intermedios que apoyen el proceso, por ejemplo: cálculo de probabilidad de la región 1, región 2, ... hasta la región n. Mostrar los resultados también para el cálculo de la media, matriz de covarianza etc.

Para las imágenes de prueba, se recicla parte del código de la fase de entrenamiento. Se cargan las imágenes del directorio, y se les aplica nuevamente el filtro gaussiano, esta vez con un radio de 4, lo que ofreció mejores resultados en comparación con otros parámetros. Cada imagen es procesada píxel por píxel utilizando tres bucles for anidados: el primero para

cambiar la imagen, el segundo para recorrer las filas y el tercero para las columnas. Cada píxel es evaluado con base en las probabilidades calculadas para determinar a qué clase pertenece (plátano, chile, huevo o fondo) y se le asigna un valor de gris correspondiente

```
probabilidadClase = pixeles_validos / pixeles_totales
priori.append(probabilidadClase)
```

3.6.- Una vez obtenido estos valores. Clasificar los píxeles 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).

Con los resultados obtenidos, se genera una imagen clasificada en escala de grises donde el fondo es representado con un valor cercano a 0, el chile con un gris oscuro, el plátano con un gris claro y el huevo con blanco. Este proceso se despliega visualmente y se analizan los errores de clasificación, como el caso donde algunos bordes oscuros de los chiles fueron clasificados como plátano. A pesar de estos pequeños fallos, los resultados generales fueron satisfactorios, y el aumento en el radio del filtro gaussiano ayudó a reducir el ruido y mejorar la clasificación

```
if max_Yk == 0:
    imagen_gris[i, j] = 170  # Clase plátano
elif max_Yk == 1:
    imagen_gris[i, j] = 85  # Clase huevo
elif max_Yk == 2:
    imagen_gris[i, j] = 255  # Clase chiles
else:
    imagen_gris[i, j] = 0  # Fondo
```

```
Media: [211.49962699 186.68036623  53.64989262],
Covarianza:
[[429.7027794  556.71508853 267.54448441]
 [556.71508853 797.85001804 378.03544404]
 [267.54448441 378.03544404 386.44058613]]
```

```
--------------------------------------------------
Total de píxeles del fondo: 274657
[0.122875, 0.04005, 0.07413888888888889, 0.7629361111111111]
```

Aquí se observa que el fondo tiene 76% de probabilidad mientras que el chile es el menos probable con 4%.

3.7.- Cree una nueva imagen con las clases resultantes y asigna 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.

Se probaron diferentes imágenes, incluyendo algunas de entrenamiento. En ciertos casos, se observó que el algoritmo clasificó erróneamente ciertos objetos debido a su color, como ocurrió con la papa, que fue clasificada como plátano debido a su similitud en color. Estos errores sugieren que sería útil incorporar características adicionales, como la forma o la textura, para mejorar la precisión de la clasificación

```
plt.figure()
plt.imshow(imagen_gris, cmap='gray')
plt.title(f"Imagen Clasificada: {file_name}")
plt.axis('off')
plt.show()
```

Imagen Clasificada: Entrenamiento2.jpg

Imagen Clasificada: Entrenamiento3.jpg

Imagen Clasificada: Prueba2.jpg

Imagen Clasificada: Entrenamiento4.jpg

Imagen Clasificada: Prueba3.jpg

Imagen Clasificada: Prueba1.jpg

3.8.- Ahora, utilice la función del clasificador de Bayes de scikit learn. Averigüe cómo debe de utilizarlo para que pueda introducir sus datos que generó anteriormente, es posible que haya cambios. Compare sus resultados contra los anteriores.

IV. Códigos

En este apéndice, se anexará el contenido de los códigos para los ejercicios desarrollados en la presente.

```
import numpy as np
import os
from PIL import Image, ImageFilter
from skimage import io, draw
import matplotlib.pyplot as plt
from matplotlib.widgets import
PolygonSelector
#se deben seleccionar las clases en orden
de banana, chile, huevo en la figura
```

```
# Colores excluidos, lista para medias y covarianzas y datos del fondo y el platano
acumulado = 0
priori = []
medias = []
colores_excluir = [np.array([175, 50, 32]), np.array([222, 91, 61]), np.array([151, 63, 41]),
                   np.array([213, 115, 106]), np.array([202, 140, 145]),
                   np.array([255, 140, 109]), np.array([106, 38, 19]),
                  np.array([185, 79, 55]), np.array([177, 98, 91]),
                   np.array([226.38698107, 90.25534188, 63.83707265]),
                   np.array([208, 110, 39]),np.array([205, 99, 37])]
umbral_distancia = 30

fondomed      =      np.array([222.46339678, 86.90785391,  61.44530337])
fondovar    =    np.array([[125.75563668    , 51.77164678 , 47.19401452],
    [    51.77164678,       47.19693844    , 31.14250208],
    [    47.19401452,       31.14250208    , 25.32974721]])

banmed       =       np.array([212.69594469, 189.86605505 , 56.21662013])
banvar       =       np.array([[450.64225698, 564.63829575, 279.65085721],
        [564.63829575,        737.45251354, 363.45099233],
  [279.65085721,  363.45099233  ,390.5381718 ]])

chmed = np.array([52.14456739, 78.99299293, 26.28224197])
chvar       =       np.array([[522.05255134, 384.55960224, 519.81883663],
        [384.55960224,        878.12501571, 798.80866994],
        [519.81883663,        798.80866994 ,879.13549295]])

def   calcular_Yk(X,   media,   covarianza, prior):
    diff = X - media
    cov_inv = np.linalg.inv(covarianza)
     term1  =  -0.5  *  np.dot(np.dot(diff.T, cov_inv), diff)
                  term2     =     -0.5     *  np.log(np.linalg.det(covarianza))
    term3 = np.log(prior)
    Yk = term1 + term2 + term3
    return Yk

def       calcular_media_covarianza(imagen, vertices):
    global acumulado
    vertices = np.array(vertices)
      mascara  =  np.zeros(imagen.shape[:2], dtype=bool)
     rr, cc = draw.polygon(vertices[:, 1], vertices[:, 0], mascara.shape)
    mascara[rr, cc] = True
    pixeles_seleccionados = imagen[mascara]

                            distancias      = np.array([np.linalg.norm(pixeles_seleccionados[:, :3] - color, axis=1) for color in colores_excluir])
     mascara_validos = np.all(distancias > umbral_distancia, axis=0)

                        pixeles_validos     = pixeles_seleccionados[mascara_validos]

     acumulado += pixeles_validos.shape[0]
#esta variable se usa para calcular el fondo

    if pixeles_validos.size > 0:
          media = np.mean(pixeles_validos, axis=0)
                                covarianza    = np.cov(pixeles_validos, rowvar=False)
```

```
        return media, covarianza, mascara,
mascara_validos, pixeles_validos.shape[0]
    else:
        return None, None, None, None, 0

def on_select(verts):
    global vertices
    global pixeles_totales
    vertices = verts
            media,   covarianza,   mascara,
mascara_validos,     pixeles_validos     =
calcular_media_covarianza(imagen, vertices)

    if media is not None:
                print(f"Media:  {media},
\nCovarianza:\n{covarianza}")
        medias.append(media)
        medias.append(covarianza)

                          imagen_final   =
np.zeros_like(imagen)
                imagen_final[mascara]  =
imagen[mascara] * mascara_validos[:, None]
        probabilidadClase = pixeles_validos
/ pixeles_totales
        priori.append(probabilidadClase)

        plt.figure()
        plt.imshow(imagen_final)
             plt.title("Imagen con Máscara
Aplicada")
        plt.axis('off')
        plt.show()
    else:
        print("No se encontraron píxeles en
el área seleccionada.")

# Cargar la imagen con filtro gaussiano con
radio de 2
imagen = io.imread('Ent2.jpg')
filas, columnas, _ = imagen.shape
pixeles_totales    =    imagen.shape[0]    *
imagen.shape[1]
imagen_gris  =  np.zeros((filas,  columnas),
dtype=np.uint8)
```

```
for k in range(3):
    fig, ax = plt.subplots()
    ax.imshow(imagen)
      ax.set_title("Selecciona el área de
interés")
          selector   =   PolygonSelector(ax,
on_select)
    plt.show()



print("_____
_____")

pixeles_fondo = pixeles_totales - acumulado
print(f"Total   de   píxeles   del   fondo:
{pixeles_fondo}")
priori.append(pixeles_fondo               /
pixeles_totales)
print(priori)

# Listas para almacenar imágenes y sus
nombres
images = []
file_names = []
blurred_images = []

# Directorio de imágenes
images_directory = "Prueba"

# Cargar imágenes del directorio y aplicar
filtro gaussiano con radio =4
for               file_name              in
sorted(os.listdir(images_directory)):
                            file_path      =
os.path.join(images_directory, file_name)
    try:
        image = io.imread(file_path)
        images.append(image)
        file_names.append(file_name)

        image_pil = Image.fromarray(image)
                           blurred_image   =
image_pil.filter(ImageFilter.GaussianBlur(r
adius=4))
```

```
blurred_images.append(np.array(blurred_imag
e))

    except Exception as e:
        print(f"Error al cargar la imagen
{file_name}: {e}")

# Aplicar el proceso de pixel a pixel a
cada una de las imagenes preprocesadas de
la lista
for imagen, file_name in
zip(blurred_images, file_names):
    print(f"Procesando imagen desenfocada:
{file_name}")

    # Inicializar imagen de salida en
escala de grises
    filas, columnas, _ = imagen.shape
        imagen_gris = np.zeros((filas,
columnas), dtype=np.uint8)

    # Recorrer los píxeles de la imagen
    for i in range(imagen.shape[0]):  #
Filas
        for j in range(imagen.shape[1]):  #
Columnas
            pixel = imagen[i, j, :3]  #
Obtener el valor RGB del píxel

            # Probabilidades, algunos datos
se calcularon en otro programa
            Y_platano = calcular_Yk(pixel,
banmed, banvar, priori[0])
            Y_chiles = calcular_Yk(pixel,
chmed, chvar, priori[1])
            Y_huevo = calcular_Yk(pixel,
medias[4], medias[5], priori[2])
            Y_fondo = calcular_Yk(pixel,
fondomed, fondovar, priori[3])


            Y_vals = [Y_platano, Y_chiles,
Y_huevo, Y_fondo]
            max_Yk = np.argmax(Y_vals)

            if max_Yk == 0:
                imagen_gris[i, j] = 170  #
Clase plátano
            elif max_Yk == 1:
                imagen_gris[i, j] = 85  #
Clase huevo
            elif max_Yk == 2:
                imagen_gris[i, j] = 255  #
Clase chiles
            else:
                imagen_gris[i, j] = 0  #
Fondo

    # Resultado
    plt.figure()
    plt.imshow(imagen_gris, cmap='gray')
        plt.title(f"Imagen Clasificada:
{file_name}")
    plt.axis('off')
    plt.show()

print(medias)
print(medias[0][0])
print(medias[1] * 2)
```

V. CONCLUSIONES

En conclusión, esta práctica permitió implementar y comprender el funcionamiento del clasificador de Bayes aplicado a imágenes, desde el preprocesamiento hasta la clasificación de píxeles. A través del uso de filtros gaussianos, la creación de máscaras de análisis y el cálculo de probabilidades a priori, se lograron segmentar distintas regiones de una imagen con precisión.

Además, se realizó una comparación entre una implementación propia y la proporcionada por la librería scikit-learn, lo que permitió evaluar la eficiencia y precisión de ambos enfoques.

Esta práctica no solo nos hizo poner en práctica nuestros conceptos teóricos sobre el clasificador de Bayes, sino que también la pusimos en práctica evidenciando la importancia de los cálculos estadísticos como la media y la matriz de covarianza en el proceso de clasificación.

References

[1] Sucar, L. E. (2021). Bayesian classifiers. En Advances in Computer Vision and Pattern Recognition (pp. 43–69). Springer International Publishing.

[2] IBM (2024) ¿Qué son los clasificadores NaÏve Bayes? Ibm.com. https://www.ibm.com/mx-es/topics/naive-bayes

[3] Sci-Kit Team. (2021). Scikit-Learn. About Us Recuperado el 7 de octubre de 2024, de https://scikit-learn.org/stable/about.html

[4] Matplotlib Dev Team (2022) Sobre Matplotlib, ¿Quienes Somos? Recuperado el 7 de octubre de 2024 https://matplotlib.org/stable/index.html

[5] Amat J. (2020) Clustering con Python. Cienciadedatos.net. Recuperado el 7 de octubre de 2024, de https://cienciadedatos.net/documentos/py20-clustering-con-python