

Práctica 2. Clasificador Bayesiano

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Reconocimiento de Patrones (757-1)

Basile Álvarez Andrés José

Keller Ascencio Rodolfo Andrés

Shen Shuai

*

Resumen—A lo largo de este documento se presentarán los resultados y análisis de nuestra segunda práctica de la materia de Reconocimiento de Patrones, donde se estará trabajando con la clasificación de objetos en imágenes a partir de un clasificador Bayesiano implementado desde cero en el lenguaje de programación Python, evaluando su precisión y capacidad de clasificación. El objetivo principal de esta práctica busca lograr clasificar imágenes con 2, 3 o 4 regiones utilizando un clasificador Bayesiano.

I. INTRODUCCIÓN

Los métodos de Bayes ingenuo representan un conjunto de algoritmos de aprendizaje supervisado que se basan en la aplicación del teorema de Bayes, asumiendo la *ingenua* condición de que las características son independientes entre sí, facilitando el cálculo de las probabilidades de cada clase. Los clasificadores Bayesianos son utilizados en gran medida para aplicaciones de clasificación de texto, minería de datos e incluso para el reconocimiento de objetos en imágenes (como se verá a lo largo de esta práctica), buscando categorizar una muestra en una o varias clases. Sin embargo, a diferencia de los clasificadores discriminantes (como la regresión logística), el clasificador Bayesiano no aprende cuáles son las características más importantes para distinguir entre las clases. [1]

La formulación del clasificador Bayesiano se basa en utilizar la regla de Bayes para calcular la probabilidad *a posteriori* de la clase dados los atributos, como se muestra a continuación: [2]

$$P(C_k|x_1, \dots, x_n) = \frac{P(C_k)P(x_1, \dots, x_n|C_k)}{P(x_1, \dots, x_n)} \quad (1)$$

El clasificador Bayesiano, entonces, consiste en asignar un objeto descrito por un conjunto de atributos o características x_1, \dots, x_n , a una de las k clases posibles, C_1, \dots, C_k , tal que la probabilidad de la clase dados los atributos se maximice:

$$\text{ArgMax}(P(C_k|x_1, \dots, x_n)) \quad (2)$$

Debido a que el denominador de (1) no varía para las diferentes clases, se puede simplificar la fórmula considerándolo como una constante:

$$P(C_k|x_1, \dots, x_n) = \alpha P(C_k)P(x_1, \dots, x_n|C_k) \quad (3)$$

El clasificador Bayesiano se basa en la suposición de que todos los atributos o características son independientes dada la clase, o sea que cada x_i es condicionalmente independiente de los demás atributos dada la clase:

$$P(x_i|x_j, C) = P(x_i|C_k), \forall i \neq j \quad (4)$$

Por lo que podemos reescribir (1) como:

$$P(C_k|x_1, \dots, x_n) = \frac{1}{Z} \prod_{i=1}^n P(C_k)P(x_i|C_k) \quad (5)$$

I-A. Conjunto de datos utilizado

Para la realización de esta práctica, se decidió trabajar con dos conjuntos de datos diferentes. El primero de ellos un conjunto con cuatro imágenes de entrenamiento y tres de prueba en donde aparecen, en distintas posiciones, dos plátanos, tres huevos y tres chiles sobre un fondo rojo. Todas las imágenes en este conjunto de datos se encuentran en formato de imagen JPG, con un tamaño de 600x600px.

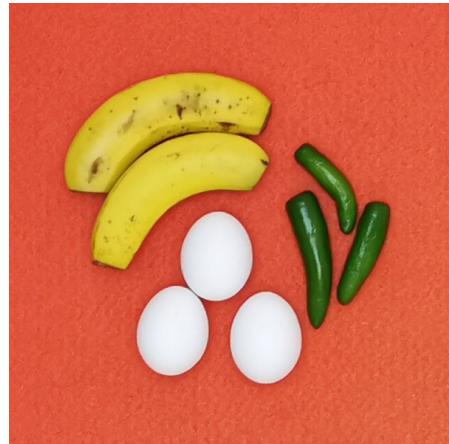


Figura 1: Ejemplo de imagen en el conjunto de datos de alimentos.

El segundo conjunto de datos cuenta con ocho imágenes de entrenamiento y tres imágenes de prueba donde aparece un mono dentro de un ambiente tipo selva. Todas las imágenes de este conjunto de datos se encuentran en formato JPG, pero con tamaños de anchura y altura de imagen variables.



Figura 2: Ejemplo de imagen en el conjunto de datos de monos.

I-B. Herramientas utilizadas

Para el desarrollo de esta práctica, se decidió trabajar con el lenguaje de programación Python, implementando el clasificador Bayesiano de manera manual (sin utilizar bibliotecas externas que ya lo implementen) y creando una interfaz gráfica de usuario (GUI) para la interacción de éste con la aplicación. Los específicos de la implementación utilizando el lenguaje Python serán vistos a lo largo del desarrollo y resultados de la práctica, así como en el apartado VI de este documento.

II. OBJETIVO

El objetivo principal de esta práctica se enfoca en lograr clasificar imágenes con 2, 3 o 4 regiones utilizando un clasificador Bayesiano.

III. DESARROLLO

Como fue mencionado anteriormente, la implementación del clasificador Bayesiano se realizó para clasificar imágenes de dos conjuntos de datos distintos: el primero de ellos de alimentos, en donde se buscaba clasificar huevos, plátanos y chiles en imágenes donde éstos se encuentran en distintas posiciones sobre un fondo rojo; el segundo conjunto donde se implementó el clasificador Bayesiano consiste en imágenes de monos en la selva, donde el propósito es determinar la localización dentro de la imagen en la cual se encuentra el mono y el espacio que representa el fondo (en este caso el fondo es vegetación diversa).

III-A. Creación de la interfaz gráfica de usuario

Se optó por crear una interfaz gráfica de usuario para hacer más amigable la carga de imágenes al modelo y la obtención de las máscaras de manera manual, siendo que éstas se pueden conseguir directamente en la interfaz gráfica. Los específicos de la creación y funcionamiento de esta interfaz están fuera del alcance de este documento, por lo que únicamente nos limitamos a presentar una imagen de ella.

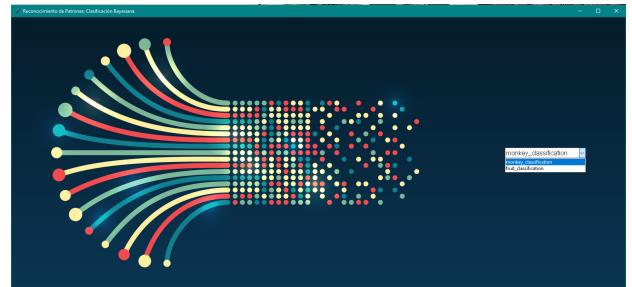


Figura 3: Interfaz gráfica construida para esta práctica, donde primeramente se permite al usuario seleccionar el conjunto de datos a utilizar, y posteriormente se le pide hacer la segregación manual para obtener las máscaras de cada imagen de entrenamiento.

III-B. Preprocesamiento de las imágenes

Para poder trabajar de manera correcta con el clasificador Bayesiano, es importante realizar un preprocesamiento de las imágenes. Dicho preprocesamiento consta, primeramente, de asegurarse que todas las imágenes tengan el mismo tamaño. Por lo tanto, se decidió trabajar únicamente con tamaños de imagen de 600x600px en ambos conjuntos de datos (las imágenes de alimentos originales ya contaban con ese tamaño, entonces únicamente se redimensionaron las imágenes de monos).

Una vez que tenemos las imágenes en el tamaño adecuado, es necesario aplicar un filtro Gaussiano a cada una de ellas. El filtro Gaussiano permite suavizar una imagen y reducir el ruido en la misma. Funciona al aplicar un kernel gaussiano a cada pixel de la imagen, borrando los detalles finos de la imagen, a través de la utilización de la "distribución de campana", donde se toma que los valores cercanos al centro del kernel tienen un peso mayor que los valores alejados al centro. [3]

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

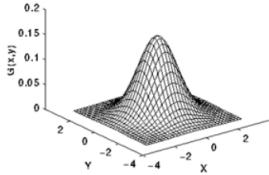


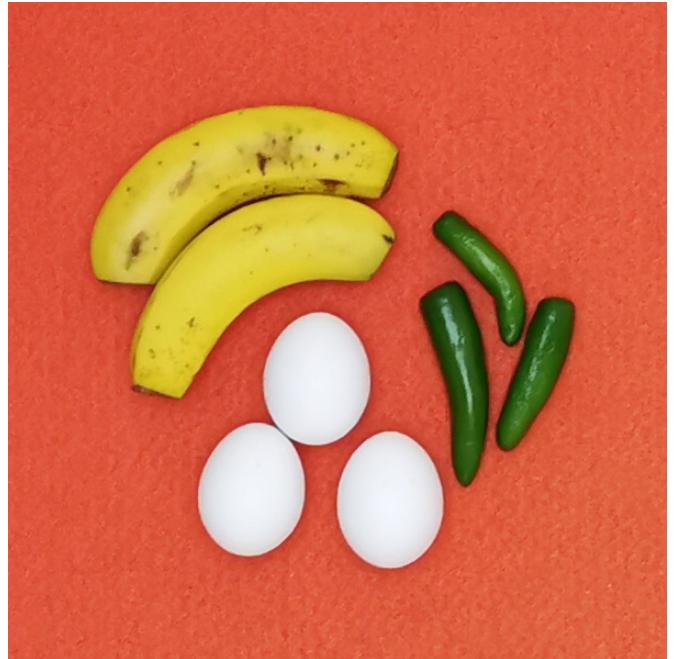
Figura 4: Al trabajar con imágenes, tenemos que usar la función Gaussiana de dos dimensiones. La imagen muestra una representación de la distribución Gaussiana 2D con media $(0,0)$ y desviación estándar igual a 1.

En el flujo normal de trabajo de nuestra aplicación, únicamente aplicamos el filtro Gaussiano a aquellas imágenes que el usuario seleccionó como de entrenamiento, y, una vez suavizadas las imágenes, solicitamos al usuario que manualmente seleccione los objetos de cada una de las clases que componen la imagen. Para las imágenes de alimentos, contamos con cuatro clases que componen la totalidad de la imagen: la clase "plátano", la clase "huevo", la clase chilez la clase "fondo"(almacenadas con los números 0, 1, 2 y 3, respectivamente, en el programa). Para las imágenes de monos, contamos con tres clases: la clase "mono", la clase "haloz la clase "fondo"(almacenadas con los números 0, 1 y 2, respectivamente).

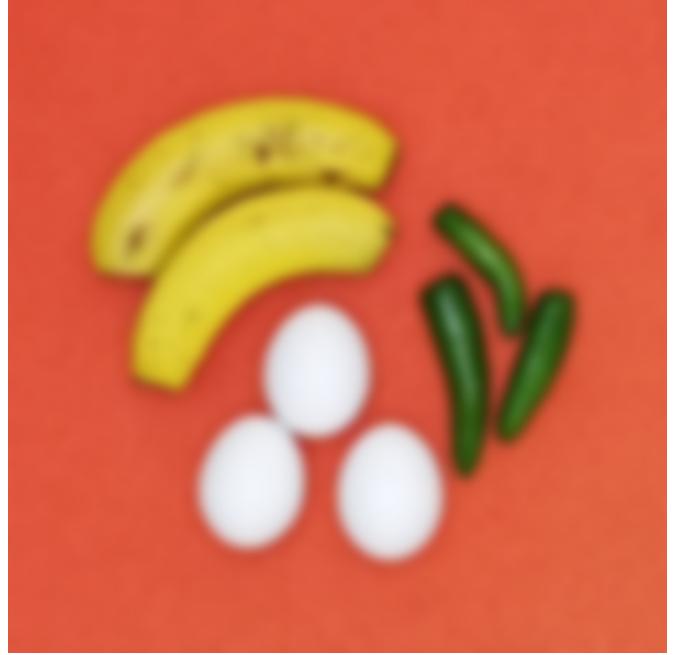
III-C. Probabilidades a priori, matriz de medias y matriz de covarianzas

Las clases seleccionadas manualmente por el usuario utilizando el cursor son almacenadas en una imagen donde se diferencia cada una de ellas con un color primario distinto (y el fondo se distingue por ser de color negro). La imagen compuesta por todas las clases seleccionadas por el usuario es utilizada posteriormente para calcular las probabilidades *a priori* de cada una de las clases. Dicha probabilidad de una clase puede ser descrita como la probabilidad de que, si tomamos un pixel cualquiera en una imagen de prueba, éste pertenezca a esa clase. La probabilidad para cada una de las clases es calculada al contar la cantidad de píxeles pertenecientes a cada una de las clases y dividirlo entre el total de píxeles de la imagen.

Una vez calculadas las probabilidades a priori, debemos de calcular la matriz de medias y de covarianzas de las clases seleccionadas manualmente. La matriz de medias se obtiene al tomar todos los objetos pertenecientes a una clase y calcular la media de colores (R, G, B) de los píxeles. Este proceso es relativamente fácil de realizar debido a que al momento de que el usuario selecciona manualmente los objetos, cada uno de los objetos de cada una de las clases se almacena como una imagen diferente de la cual podemos obtener los valores R, G, B . De la misma manera, podemos obtener la matriz de covarianzas para cada una de las clases (así como la matriz



(a)



(b)

Figura 5: Imagen de entrenamiento original (a) e imagen después de aplicarle el filtro Gaussiano (b)

de covarianzas inversa, utilizando los métodos de la biblioteca *linalg* de Python).

Teniendo el número de clases a clasificar, las probabilidades a priori de cada una de ellas, las matrices de medias, covarianzas y las matrices inversas de covarianzas, podemos aplicar la función discriminante de Bayes [4] para presentarle al clasificador imágenes nuevas y obtener la clasificación de cada una de las clases. Tomemos como ejemplo una ejecución cualquiera de nuestro flujo de trabajo para imágenes de alimentos (cabe mencionar que todos los resultados y procesos intermedios de clasificación Bayesiana se almacenan en el archivo "BayesianClassifierResults.txt"):

Calculando las probabilidades a priori de cada una de las clases en las imágenes de alimentos (recordar que C_0 : Plátanos, C_1 : Huevos, C_2 : Chiles, C_3 : Fondo):

$$P(C_0) = 0.0507$$

$$P(C_1) = 0.0117$$

$$P(C_2) = 0.0218$$

$$P(C_3) = 0.9156$$

Las matrices de medias para cada clase resultaron:

$$\mu_{C_0} = \begin{pmatrix} 213,4901 \\ 188,2360 \\ 56,1023 \end{pmatrix}$$

$$\mu_{C_1} = \begin{pmatrix} 229,3286 \\ 220,7344 \\ 224,0123 \end{pmatrix}$$

$$\mu_{C_2} = \begin{pmatrix} 71,1741 \\ 88,2690 \\ 29,7143 \end{pmatrix}$$

$$\mu_{C_3} = \begin{pmatrix} 217,3101 \\ 90,6550 \\ 61,9778 \end{pmatrix}$$

Las matrices de covarianzas se calculan con:

$$S = \frac{1}{N} \sum_{i=1}^N (X_i - \mu)(X_i - \mu)^T \quad (6)$$

$$S_{C_0} = \begin{pmatrix} 209,8491 & 312,3662 & 151,2092 \\ 312,6662 & 577,0521 & 262,9658 \\ 151,2092 & 262,9558 & 289,2896 \end{pmatrix}$$

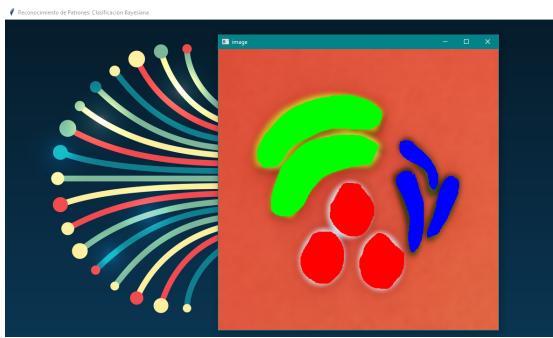
$$S_{C_1} = \begin{pmatrix} 105,3470 & 302,0659 & 337,1032 \\ 302,0659 & 1025,0239 & 1149,6592 \\ 337,1032 & 1149,6592 & 1291,3835 \end{pmatrix}$$

$$S_{C_2} = \begin{pmatrix} 897,1727 & -24,6277 & 122,2998 \\ -24,6277 & 373,4618 & 216,6380 \\ 122,2998 & 216,6380 & 230,6213 \end{pmatrix}$$

$$S_{C_3} = \begin{pmatrix} 419,4818 & 66,5563 & 143,6488 \\ 66,5563 & 384,9191 & 230,7914 \\ 143,6488 & 230,7914 & 361,3926 \end{pmatrix}$$

Y aplicamos el discriminante de Bayes para obtener las nuevas clasificaciones:

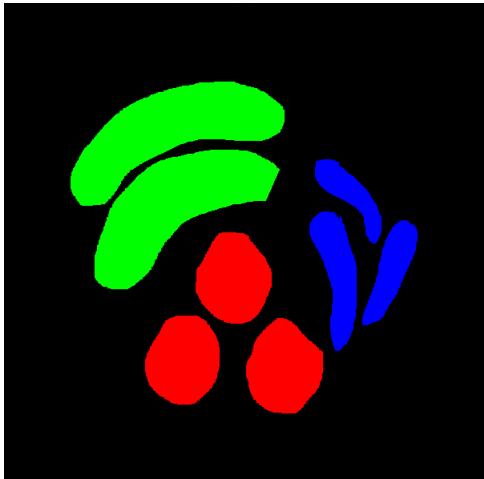
$$Y_k(X) = \frac{-1}{2}(X - \mu_k)^T S_k^{-1}(X - \mu_k) - \frac{1}{2} \ln|S_k| + \ln P(C_k) \quad (7)$$



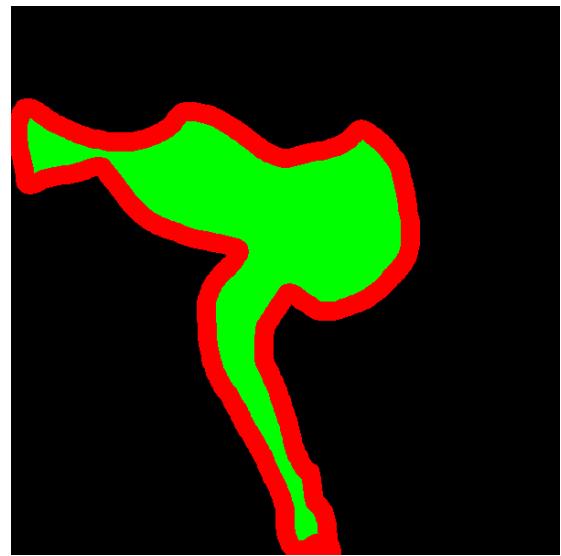
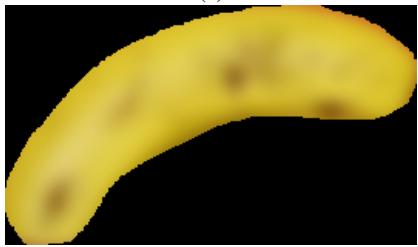
(a)



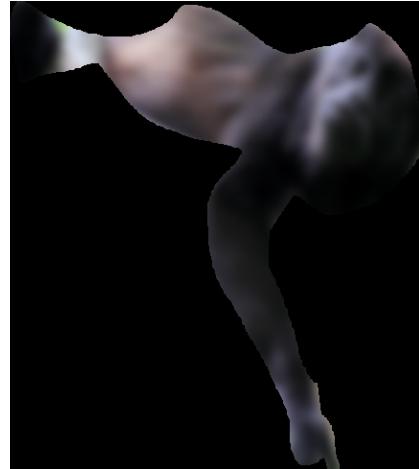
(b)



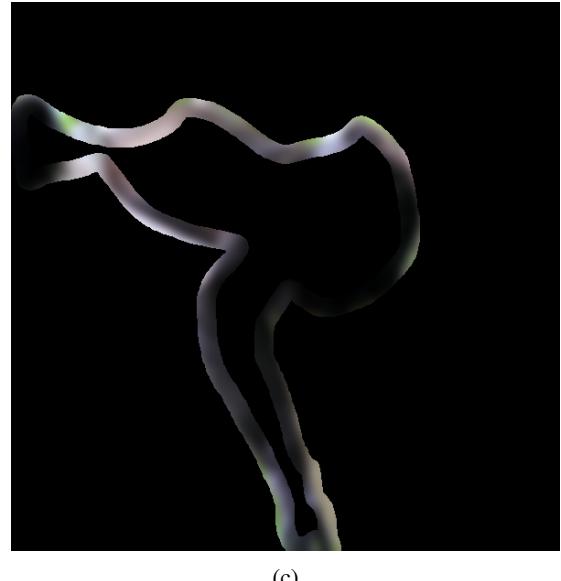
(c)



(a)



(b)



(c)

Figura 6: Herramienta gráfica creada para seleccionar manualmente los objetos de cada clase en el entrenamiento y así crear las máscaras de imagen (a) y (b), e imagen resultado de segregar manualmente las clases (c). De las máscaras obtenidas en la imagen anterior, podemos obtener los objetos seleccionados para su posterior análisis (d).

Figura 7: Se realiza un proceso similar para los monos, obteniendo la máscara de la imagen para las clases mono y halo (a), la clase mono sola (b) y la clase halo sola (c).

IV. RESULTADOS

Una vez aplicado el discriminante de Bayes para clasificar los objetos en las imágenes de prueba, generamos nuevas imágenes mostrando los resultados alcanzados.

Para los resultados de la clasificación de objetos en el conjunto de datos de alimentos, a partir de la predicción hecha por el clasificador Bayesiano, coloreamos cada uno de los pixeles de una imagen de 600x600px según la clase a la que pertenece. Coloreamos de rojo todos aquellos pixeles que fueron clasificados por nuestro clasificador como pertenecientes a la clase "fondo"; coloreamos de amarillo todos aquellos pixeles clasificados como pertenecientes a la clase "plátano"; de blanco los pertenecientes a la clase "huevo"; y de verde los pertenecientes a la clase "chile".

Similarmente, para las imágenes de monos generamos nuevas imágenes con los resultados de la clasificación, en donde coloreamos cada uno de los pixeles según el valor de clasificación dado por el clasificador, obteniendo una nueva imagen de 600x600px con pixeles coloreados en gris oscuro para la clase "fondo", gris para la clase "mono" blanco para la clase "halo".

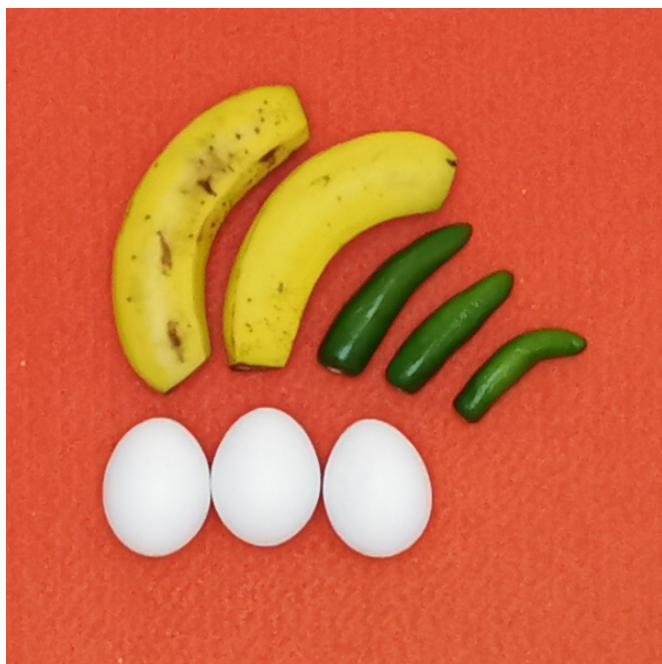
A partir de las imágenes generadas, rápidamente logramos observar que la clasificación de cada uno de los objetos fue realizada correctamente, en su mayoría. Por ejemplo, para el caso de las imágenes de alimentos, existen algunos pixeles coloreados de verde (clase chile), dentro del objeto de clase "plátano". Esto se debe a que, al tener manchas oscuras, ciertos pixeles de la clase plátano pueden aparentar ser un chile (por su semejanza de color con esta otra clase). Además, observamos la ineeficacia al momento de clasificar un objeto totalmente nuevo (la papa en la tercera imagen de prueba). Esta dificultad para clasificar nuevos objetos se debe a que el método de clasificación Bayesiana únicamente va a calcular probabilidades de cada clase y a discriminar basándose en dichas probabilidades y en las medias de color de las clases. La papa, al tener un color amarillento y un reflejo blanco, es clasificada en parte como clase "plátano" en parte como clase "huevo".

Para las imágenes de prueba del conjunto de datos de monos, observamos que prácticamente la totalidad de las clases en las imágenes fueron clasificadas de forma correcta. No obstante, existen algunas áreas que, principalmente debido a su color, son clasificadas como pertenecientes a una clase distinta (ver por ejemplo la figura 11, en donde se muestra que el dorso café del mono fue clasificado como clase "fondo" debido a que su color es más cercano al color promedio del fondo; ocurriendo algo similar en la figura 13).

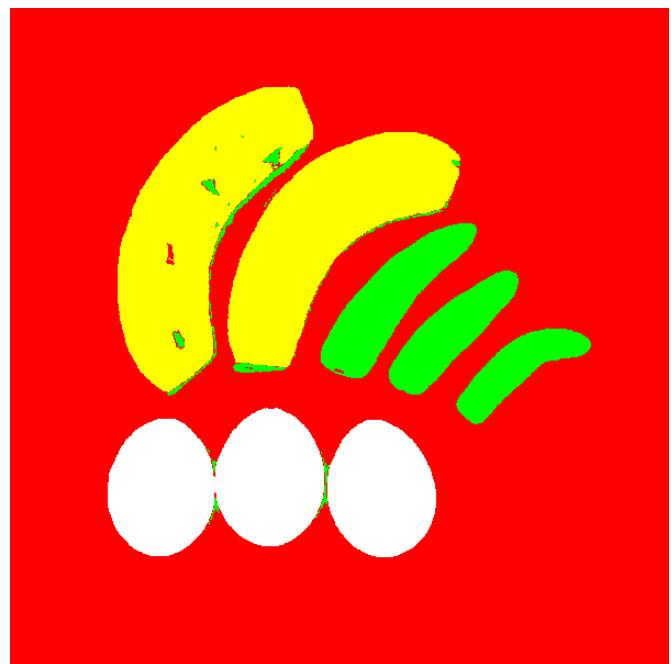
Nuestro modelo tuvo una precisión del 85.48 %, una exactitud de 91.77 %, y un *recall* de 80.23 %.

Valores reales

Predicción	Valores reales			Total
	Positive	Negative	Total	
Positive	71,153	12,083	83,236	
Negative	17,532	259,232	276,764	
Total	88,685	271,315	720,000	

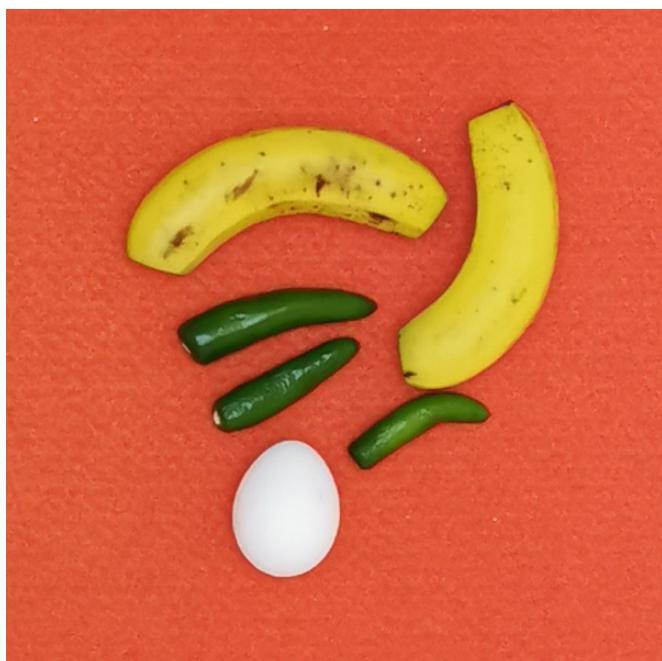


(a)



(b)

Figura 8: Imagen de prueba (a) e imagen resultado del clasificador Bayesiano (b), para el conjunto de datos de alimentos.



(a)

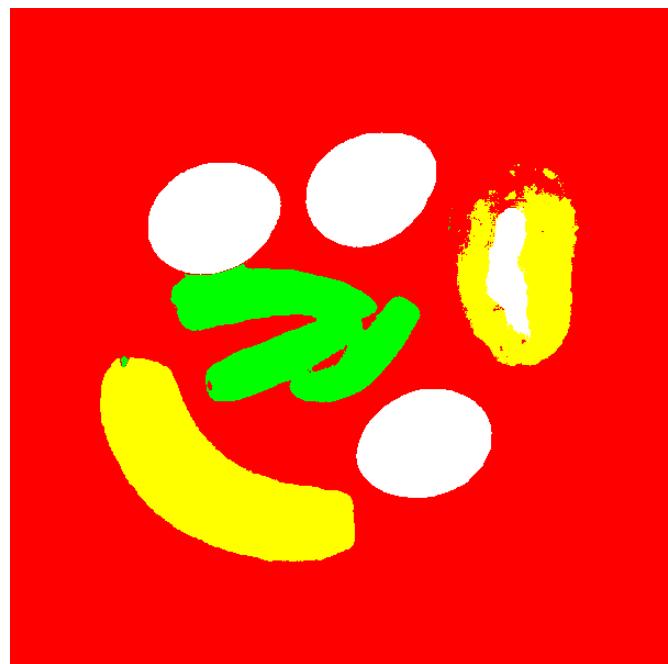


(b)

Figura 9: Imagen de prueba (a) e imagen resultado del clasificador Bayesiano (b), para el conjunto de datos de alimentos.

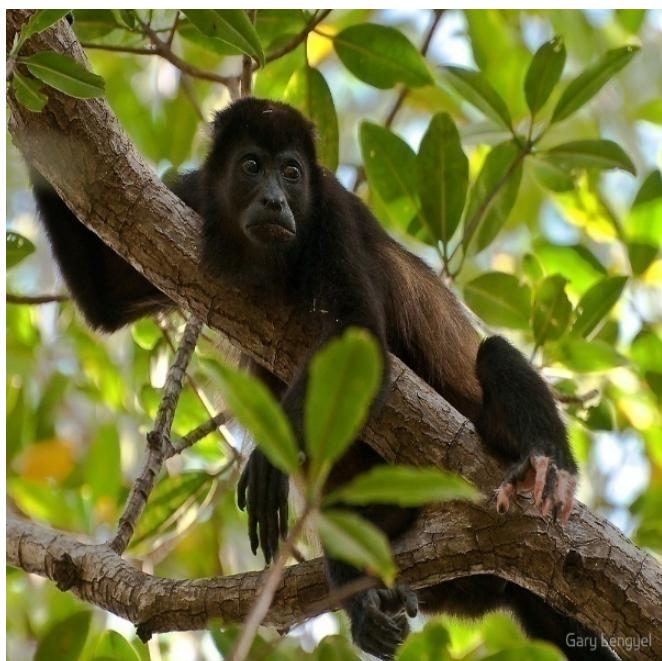


(a)

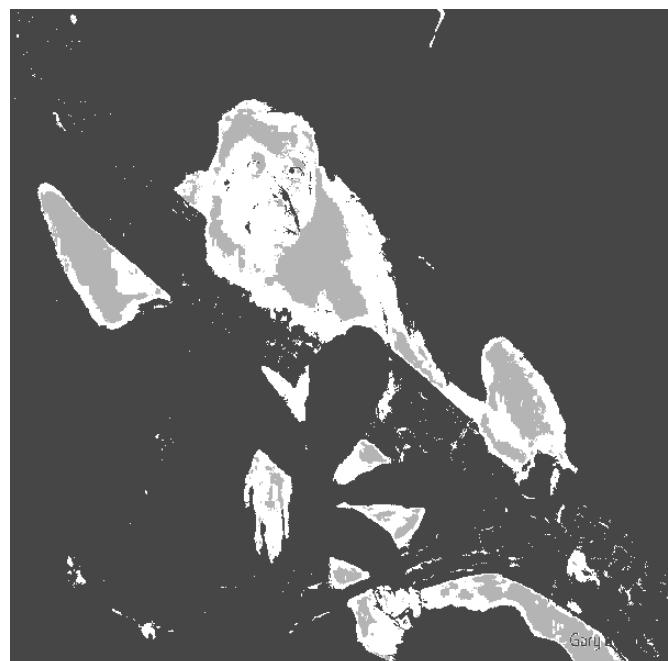


(b)

Figura 10: Imagen de prueba (a) e imagen resultado del clasificador Bayesiano (b), para el conjunto de datos de alimentos.

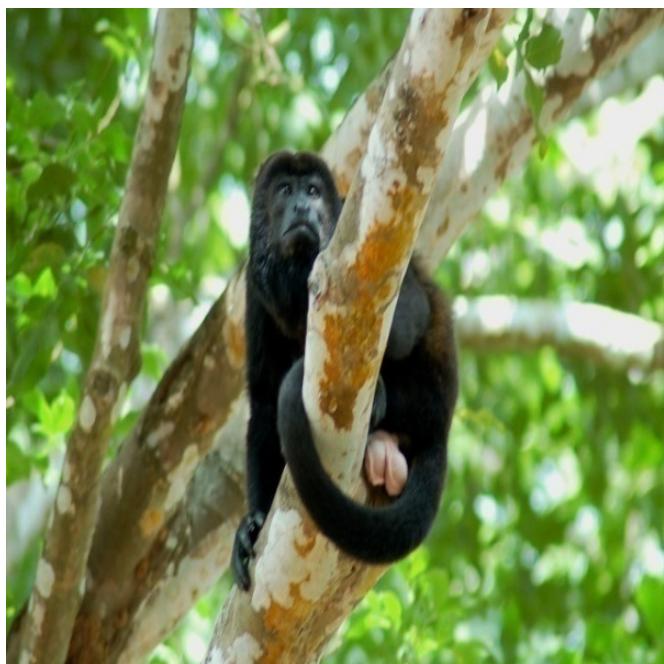


(a)



(b)

Figura 11: Imagen de prueba (a) e imagen resultado del clasificador Bayesiano (b), para el conjunto de datos de monos.

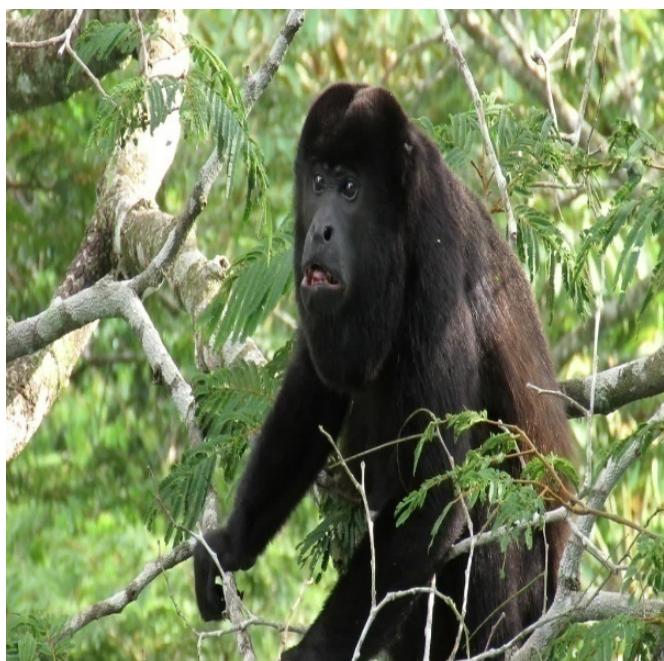


(a)

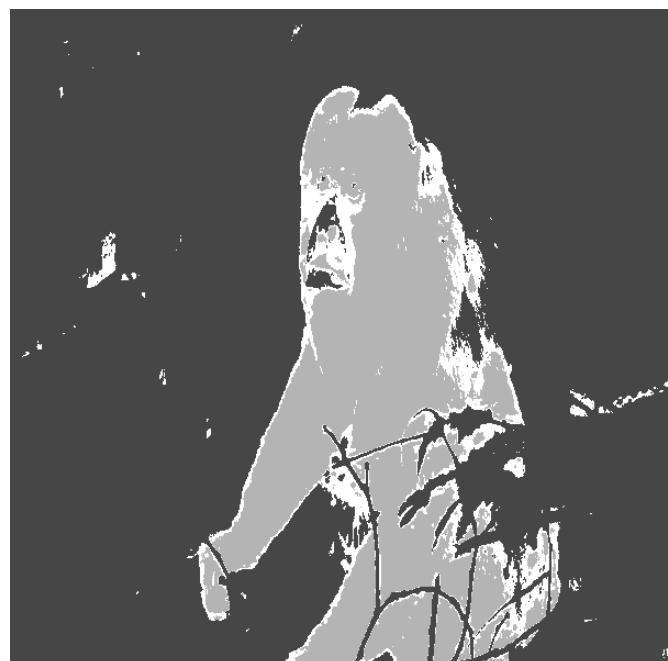


(b)

Figura 12: Imagen de prueba (a) e imagen resultado del clasificador Bayesiano (b), para el conjunto de datos de monos.



(a)



(b)

Figura 13: Imagen de prueba (a) e imagen resultado del clasificador Bayesiano (b), para el conjunto de datos de monos.

V. CLASIFICADOR DE BAYES SCIKIT LEARN

Tras haber realizado nuestra propia implementación para la clasificación de objetos dentro de las imágenes, se trabajó con la clasificación de Bayes del paquete Scikit Learn, para que de esta manera pudiéramos realizar comparaciones entre ambos modelos, trabajando con el grupo de datos relacionado con las clases frutas.

En primera instancia fue necesario cargar las imágenes de las diferentes clases que ya habían sido segmentadas previamente. De estas imágenes sabemos que hay cuatro clases: plátanos, huevos, chiles y fondo. Una vez cargadas las imágenes de cada clase, convertimos los píxeles de estas imágenes en un dataframe con tres columnas: R, G y B. Además, agregamos una cuarta columna para indicar a qué clase pertenece cada pixel.

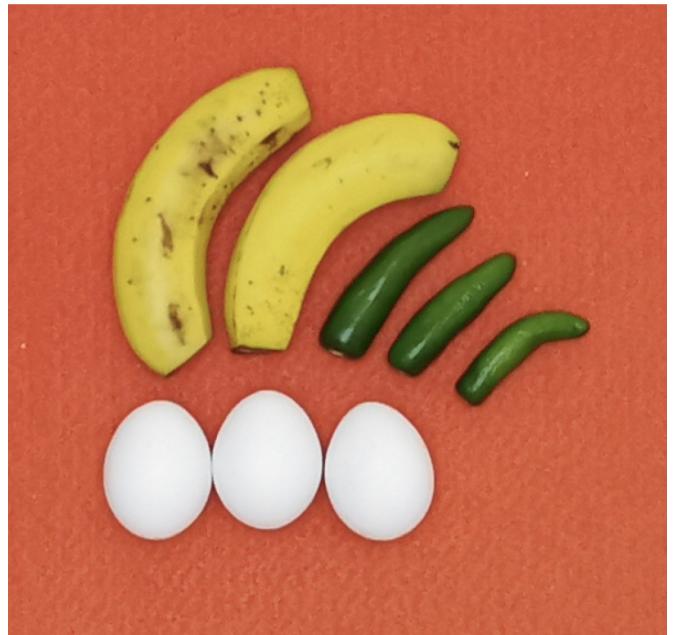
Se realizó una limpieza de los datos, eliminando los píxeles de color negro, es decir, (0,0,0).

Una vez que preprocesamos los datos, procedimos a dividirlos en dos grupos: el 70 % para entrenar el modelo y el 30 % para evaluar su rendimiento. Como resultado, obtuvimos las siguientes métricas de clasificación:

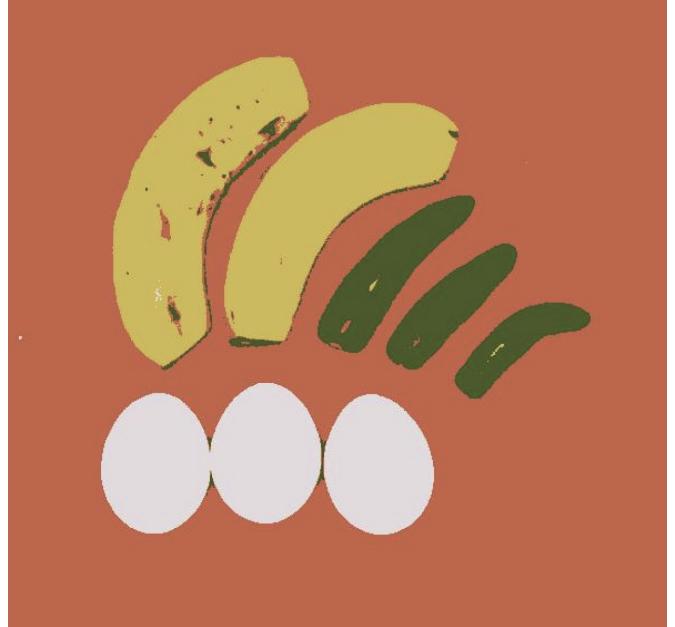
- Precisión: 86.08 %
- Sensibilidad (Recall): 95.83 %
- Exactitud: 95.38 %
- F1-score: 90.08 %

De esta manera podemos apreciar una comparación con respecto al modelo que implementamos, donde el clasificador de Scikit Learn tuvo una precisión mayor por 0.6 puntos porcentuales, una exactitud mayor por 3.61 puntos porcentuales y una sensibilidad mayor por 15.6 puntos porcentuales, siendo estos dos últimos significativamente mayores a los obtenidos a nuestro modelo implementado de forma manual, sin embargo, consideraremos que el hecho de haber implementado un modelo propio fue un gran logro.

Finalmente, probamos el funcionamiento del modelo utilizando una nueva imagen de prueba donde recorrimos cada píxel de la imagen de prueba y utilizamos nuestro modelo de clasificación para asignar una clase a cada uno de ellos. Guardamos los resultados de la clasificación en una lista y, finalmente, reconstruimos la imagen utilizando los resultados obtenidos.



(a)



(b)

Figura 14: Imagen de entrenamiento original (a) e imagen después de ser reconstruida tras aplicar el clasificador de Scikit Learn (b)

VI. CÓDIGO

Durante el desarrollo de esta práctica, se utilizó Git y Github como herramientas de gestión de versiones del código, por lo que el código completo podrá encontrarse como un repositorio abierto en el siguiente link: Github Repository Link. No obstante, se presentan algunos elementos importantes del código a continuación:

VI-A. Funciones para obtener la probabilidad a priori, la matriz de medias, covarianzas y realizar la predicción con el discriminante Bayesiano

```
def apriori_probability(monkey_fruit, num_classes):
    # A priori probability

    if monkey_fruit == 1:
        path = "OutputMonkeys/"
        num_classes = num_classes

        # Create an array to store the count of pixels for each class
        class_counts = np.zeros(num_classes)

        # Loop through each training image mask
        for filename in os.listdir(path):
            if "All_halo_Masks_" in filename:
                # Load the image
                img = cv2.imread(os.path.join(path, filename))

                # Extract the mask of each color
                blue_mask = ((img[:, :, 2] == 255).astype(int))      # halo
                green_mask = ((img[:, :, 1] == 255).astype(int))     # monkey
                black_mask = ((img[:, :, -1] == 0).astype(int))      # background

                class_counts[0] += np.sum(blue_mask)
                class_counts[1] += np.sum(green_mask)
                class_counts[2] += np.sum(black_mask)
                print("class counts, ", class_counts[0], class_counts[1])

        total_pixels = np.sum(class_counts)
        class_probs = class_counts / total_pixels

        # print("Prior probability of blue class (halo):", class_probs[0])
        # print("Prior probability of green class (monkey):", class_probs[1])
        # print("Prior probability of black class (background):", class_probs[2])

    return class_probs

elif monkey_fruit == 2:
    path = "OutputFruits/"
    num_classes = num_classes

    # Create an array to store the count of pixels for each class
    class_counts = np.zeros(num_classes)

    # Loop through each training image mask
    for filename in os.listdir(path):
        if "All_Masks_" in filename:
            # Load the image
            img = cv2.imread(os.path.join(path, filename))

            # Extract the mask of each color
            red_mask = ((img[:, :, 0] == 255).astype(int))/3      # eggs
            blue_mask = ((img[:, :, 2] == 255).astype(int))/3      # chillies
            green_mask = ((img[:, :, 1] == 255).astype(int))/2     # bananas
            black_mask = ((img[:, :, -1] == 0).astype(int))/1      # background

            class_counts[0] += np.sum(red_mask)
            class_counts[1] += np.sum(blue_mask)
            class_counts[2] += np.sum(green_mask)
            class_counts[3] += np.sum(black_mask)

    total_pixels = np.sum(class_counts)
    class_probs = class_counts / total_pixels

    print("Prior probability of red class (eggs):", class_probs[0])
```

```

print("Prior probability of blue class (chillies):", class_probs[1])
print("Prior probability of green class (bananas):", class_probs[2])
print("Prior probability of black class (background):", class_probs[3])

return class_probs

def mean_cov_matrix(monkey_fruit, num_classes, apriori):

    if monkey_fruit == 1:
        file_pattern = "OutputMonkeys/All_halo_Masks_*.png"
        file_list = glob.glob(file_pattern)

        x = 0
        for image in file_list:
            image1 = cv2.imread(image)
            image2 = cv2.imread(f"Monos/Entrenamiento{x+1}.png")
            x+=1
            mask = cv2.inRange(image1, (0,0,0), (0,0,0))

            new = np.zeros_like(image2)
            new[mask!=0] = image2[mask!=0]
            cv2.imwrite(f"OutputMonkeys/Class_3_Background_{x}.png", new)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

    # Obtaining the mean matrix for each class

    file_pattern = "OutputMonkeys/Class_*_*"
    file_list = glob.glob(file_pattern)

    c1 = []
    c2 = []
    c3 = []
    for image in file_list:
        if "Class_1" in image:
            c1.append(image)
        elif "Class_2" in image:
            c2.append(image)
        elif "Class_3" in image:
            c3.append(image)

    num_images_per_class = [c1, c2, c3]
    print("NUM IMAGES PER CLAS: ",num_images_per_class)
    n_classes = num_classes

    RGB_class_list_covs = []
    RGB_class_list_means = []
    x=0
    for class_list in num_images_per_class:
        # print(class_list)
        df_rgb_means = []
        for image in class_list:
            image = image.replace("\\", "/")
            image = cv2.imread(image)
            image_array = np.array(image)
            reshaped_array = image_array.reshape(-1,3)
            reshaped_array = reshaped_array[:,[2,1,0]]
            df = pd.DataFrame(reshaped_array, columns=['R', 'G', 'B'])
            df_rgb_means.append(df)

        df_rgb_means = pd.concat(df_rgb_means)
        df_rgb_means = df_rgb_means.loc[(df_rgb_means==0).all(axis=1)]      # Delete rows with '0' in R,
        # G and B

        x+=1

        # print(f"\n\nClase {x} RGB DataFrame:\n", df_rgb_means)
        # print(f"\n\nClase {x} mean matrix: \n", df_rgb_means.mean())
        # print(f"\n\nClase {x} cov matrix: \n", df_rgb_means.cov())

        rgb_means = df_rgb_means.to_numpy()
        cov_matrix = np.cov(rgb_means, rowvar=False)

```

```

mean_matrix = np.mean(rgb_means, axis=0)

RGB_class_list_covs.append(cov_matrix)
RGB_class_list_means.append(mean_matrix)

# print("Covs: \n\n", RGB_class_list_covs)
# print("\n\nMeans: \n\n", RGB_class_list_means)

# Covariance Matrix

det_covs = np.array([np.linalg.det(RGB_class_list_covs[k]) for k in range(n_classes)])

covs_inv = np.zeros_like(RGB_class_list_covs)
for k in range(n_classes):
    covs_inv[k] = np.linalg.inv(np.matrix(RGB_class_list_covs[k]))

# print("\n\ndet-covs", det_covs, "\n\n\n")
# print("inverse-cov", covs_inv, "\n\n\n")

return RGB_class_list_means, RGB_class_list_covs, det_covs, covs_inv

if monkey_fruit == 2:

    # Obtaining the images of only the background, to get the mean of background class.

    file_pattern = "OutputFruits/All_Masks_"
    file_list = glob.glob(file_pattern)

    x = 0
    for image in file_list:
        image1 = cv2.imread(image)
        image2 = cv2.imread(f"ImagenesEntrenamiento/Entrenamiento{x+1}.png")
        x+=1
        mask = cv2.inRange(image1, (0,0,0), (0,0,0))

        new = np.zeros_like(image2)
        new[mask!=0] = image2[mask!=0]
        cv2.imwrite(f"OutputFruits/Class_4_Background_{x}.png", new)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    # Obtaining the mean matrix for each class

    file_pattern = "OutputFruits/Class_*_*"
    file_list = glob.glob(file_pattern)

    c1 = []
    c2 = []
    c3 = []
    c4 = []
    for image in file_list:
        if "Class_1" in image:
            c1.append(image)
        elif "Class_2" in image:
            c2.append(image)
        elif "Class_3" in image:
            c3.append(image)
        elif "Class_4" in image:
            c4.append(image)

    num_images_per_class = [c1, c2, c3, c4]
    n_classes = num_classes

    RGB_class_list_covs = []
    RGB_class_list_means = []
    x=0
    for class_list in num_images_per_class:
        print(class_list)
        df_rgb_means = []
        for image in class_list:

```

```

image = image.replace("\\\\", "/")
image = cv2.imread(image)
image_array = np.array(image)
reshaped_array = image_array.reshape(-1,3)
reshaped_array = reshaped_array[:,[2,1,0]]
df = pd.DataFrame(reshaped_array, columns=['R', 'G', 'B'])
df_rgb_means.append(df)

df_rgb_means = pd.concat(df_rgb_means)
df_rgb_means = df_rgb_means.loc[~(df_rgb_means==0).all(axis=1)]    # Delete rows with '0' in R,
                                                               ↳ G and B

x+=1

print(f"\n\nClase {x} RGB DataFrame:\n",df_rgb_means)
print(f"\nClass {x} mean matrix: \n", df_rgb_means.mean())
print(f"\nClass {x} cov matrix: \n", df_rgb_means.cov())

rgb_means = df_rgb_means.to_numpy()

cov_matrix = np.cov(rgb_means, rowvar=False)
mean_matrix = np.mean(rgb_means, axis=0)

RGB_class_list_covs.append(cov_matrix)
RGB_class_list_means.append(mean_matrix)

print("Covs: \n\n", RGB_class_list_covs)
print("\n\nMeans: \n\n", RGB_class_list_means)

# Covariance Matrix

det_covs = np.array([np.linalg.det(RGB_class_list_covs[k]) for k in range(n_classes)])

covs_inv = np.zeros_like(RGB_class_list_covs)
for k in range(n_classes):
    covs_inv[k] = np.linalg.inv(np.matrix(RGB_class_list_covs[k]))

print("\ndet-covs", det_covs," \n\n")
print("inverse-cov", covs_inv, "\n\n")

return RGB_class_list_means, RGB_class_list_covs, det_covs, covs_inv

def bayes_disc(vector, mean, inverse_cov, cov_det, class_P):
    step = np.matrix(vector-mean)
    inverse_cov = np.matrix(inverse_cov)
    step2 = np.matmul(step, inverse_cov)
    step3 = (step2.dot(step.T))[0,0].item()
    bayes_disc = -(1.0/2.0)*step3-(1.0/2.0)*np.log(cov_det)+np.log(class_P)
    return bayes_disc.item()

def predict(image_data, model):
    image_data_shape = image_data.shape
    prediction = np.empty_like(image_data[:, :, 0]).astype(np.uint8)
    discr = np.empty(len(model['classes'])).astype(np.float64)
    for i in range(image_data_shape[0]):
        for j in range(image_data_shape[1]):
            for k in model['classes']:
                vector=image_data[i,j]
                mean=model['mean'][k]
                cov=model['cov'][k]
                inverse_cov=model['inverse_cov'][k]
                cov_det=model['cov_det'][k]
                class_P=model['apriori'][k]
                discr[k] = bayes_disc(vector, mean, inverse_cov, cov_det, class_P)
            prediction[i,j] = discr.argmax()

    return prediction

```

VI-B. Código de la interfaz gráfica

```
class ImageEditor(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)

        self.master = master
        self.master.title("Reconocimiento de Patrones: Clasificación Bayesiana")
        self.width = 1543
        self.height = 679
        self.screen_width = root.winfo_screenwidth()
        self.screen_height = root.winfo_screenheight()
        self.x = (self.screen_width / 2) - (self.width / 2)
        self.y = (self.screen_height / 2) - (self.height / 2)
        self.master.geometry("%dx%d+%d+%d" % (self.width, self.height, self.x, self.y))
#self.master.geometry("1200x800")
        self.master.configure()
        self.pack()

        self.bg_image = tk.PhotoImage(file="Extras/background3.png")
        self.bg_label = tk.Label(root, image=self.bg_image)
        self.bg_label.place(x=0, y=0, relwidth=1, relheight=1)

        self.program_var = tk.StringVar()
        self.program_combobox = ttk.Combobox(root, textvariable=self.program_var, font=("MS Sans Serif",
        ↵ 12))
        self.program_combobox['values'] = ("monkey_classification", "fruit_classification")
        self.program_combobox.pack(side="right", padx=100, pady=280)

# self.open_button = tk.Button(root, text="Open Files", font=("MS Sans Serif", 12),
        ↵ command=self.process_fruit_image)
        self.program_var.trace_add("write", lambda *args: self.update_button())

        self.images = []
        self.canvases = []
        self.num_classes = None
        self.filenames = []

# function to update the button visibility based on the selected program
def update_button(self):
    if self.program_var.get() == "monkey_classification":
        # self.open_button.pack(side='right', pady=20)
        self.process_monkey_image()

    elif self.program_var.get() == "fruit_classification":
        # self.open_button.pack(side='right', pady=20)
        self.process_fruit_image()

    else:
        self.open_button.pack_forget()

def process_fruit_image(self):
    self.filenames = filedialog.askopenfilenames()
    i = 0
    x = 0
    for filename in self.filenames:
        image = Tools.GaussianFilter(filename)

        Tools2.Fruit_Mask(filename.rstrip(".jpg")+".png", i)

        i += 1

    Classification.BayesRGB(2)

def process_monkey_image(self):
    self.filenames = filedialog.askopenfilenames()
    i = 0
    x = 0
    for filename in self.filenames:
        image = Tools.GaussianFilter(filename)
        Tools2.Monkey_Mask(filename.rstrip(".jpg")+".png", i)
```

```

        i += 1
Classification.BayesRGB(1)

root = tk.Tk()
app = ImageEditor(master=root)
app.mainloop()

```

VI-C. Código para aplicar el filtro Gaussiano a las imágenes y para la obtención manual de máscaras en las imágenes de frutas.

```

def GaussianFilter(image):
    image_obj = Image.open(image)
    newsize = (600,600)
    image_obj = image_obj.resize(newsize)
    image_obj = image_obj.filter(ImageFilter.GaussianBlur(radius=5))
    image_obj.save(image.rstrip(".jpg") + ".png")
    return image_obj

NUM_CLASSES = 3

# Define the colors for each class
COLORS = [
    (0, 255, 0), # Green
    (0, 0, 255), # Red
    (255, 0, 0)
]

]

# Define the variables for the user drawing
drawing = False
points = []
contours = [[] for _ in range(NUM_CLASSES)]
class_num = 0

def draw(event, x, y, flags, param):
    global drawing, points

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        points = [(x, y)]
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            points.append((x, y))
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        points.append((x, y))
        contours[class_num].append(np.array(points))
        points = []

def Fruit_Mask(filename, image_num):
    global class_num, contours

    # Load the image
    img = cv2.imread(filename)

    # Create a black mask for each class
    masks = [np.zeros(img.shape[:2], np.uint8) for _ in range(NUM_CLASSES)]

    class_num = 0
    contours = [[] for _ in range(NUM_CLASSES)]
    # Display the image and let the user draw contours
    cv2.namedWindow('image')
    cv2.setMouseCallback('image', draw)
    while True:
        # Draw the contours for the current class
        mask = masks[class_num]
        for contour in contours[class_num]:
            cv2.drawContours(mask, [contour], -1, 255, -1)

        # Display the image with the contours drawn
        # masked_img = cv2.bitwise_and(img, img, mask=mask)
        # cv2.imshow('image', masked_img)

```

```

img_with_contours = img.copy()
for i, mask in enumerate(masks):
    color = COLORS[i]
    img_with_contours[mask > 0] = color
cv2.imshow('image', img_with_contours)

# Handle user input
key = cv2.waitKey(1)
if key == ord('q'):
    break
elif key == ord('c'):
    # Clear the current class
    contours[class_num] = []
    masks[class_num] = np.zeros(img.shape[:2], np.uint8)
elif key == ord('n'):
    # Switch to the next class
    class_num = (class_num + 1) % NUM_CLASSES
elif key == ord('p'):
    # Switch to the previous class
    class_num = (class_num - 1) % NUM_CLASSES
elif key == ord('r'):
    img = cv2.imread(filename)
    points = []
    contours = []
elif key == ord('s'):
    # Save the results
    output = np.zeros(img.shape, dtype=np.uint8)
    for i, mask in enumerate(masks):
        color = COLORS[i]
        output += cv2.bitwise_and(img, img, mask=mask)
        output[mask > 0] = color
    cv2.imwrite(f'OutputFruits/All_Masks_{image_num+1}.png', output)

    # Save the results
    for i, contour_list in enumerate(contours):
        color = COLORS[i]
        for j, contour in enumerate(contour_list):
            mask = np.zeros(img.shape[:2], np.uint8)
            cv2.drawContours(mask, [contour], -1, 255, -1)
            masked_img = cv2.bitwise_and(img, img, mask=mask)
            x, y, w, h = cv2.boundingRect(contour)
            cropped_img = masked_img[y:y+h, x:x+w]
            cv2.imwrite(f"OutputFruits/Class_{i+1}_Contour_{j+1}_{image_num+1}.png", cropped_img)

# Clean up
cv2.destroyAllWindows()

```

VI-D. Código para el procesamiento de imágenes y uso del clasificador de Bayes de Scikit Learn.

```

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from PIL import Image
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import pandas as pd
import cv2
#####
#####Platanos#####
#####
imagenP1 = Image.open('OutputFruits/Class_1_Contour_1_1.png')
imagenP2 = Image.open('OutputFruits/Class_1_Contour_2_1.png')
pixelesP1 = list(imagenP1.getdata())
pixelesP2 = list(imagenP2.getdata())
# Convierte la lista de tuplas en un DataFrame de pandas
columnas = ['R', 'G', 'B']
df1 = pd.DataFrame(pixelesP1, columns=columnas)
df1 = df1.loc[~(df1 == 0).all(axis=1)]
df2 = pd.DataFrame(pixelesP2, columns=columnas)
df2 = df2.loc[~(df2 == 0).all(axis=1)]
# Une los dos DataFrames de forma vertical

```

```

df_platanos = pd.concat([df1, df2], ignore_index=True)
#Agrega una columna
df_platanos['objeto'] = 'platano'
#print(df_platanos)

#####
#####Huevos#####
#####
imagenH1 = Image.open('OutputFruits/Class_2_Contour_1_1.png')
imagenH2 = Image.open('OutputFruits/Class_2_Contour_2_1.png')
imagenH3 = Image.open('OutputFruits/Class_2_Contour_3_1.png')
pixelesH1 = list(imagenH1.getdata())
pixelesH2 = list(imagenH2.getdata())
pixelesH3 = list(imagenH3.getdata())
# Convierte la lista de tuplas en un DataFrame de pandas
columnas = ['R', 'G', 'B']
dfH1 = pd.DataFrame(pixelesH1, columns=columnas)
dfH1 = dfH1.loc[~(dfH1 == 0).all(axis=1)]
dfH2 = pd.DataFrame(pixelesH2, columns=columnas)
dfH2 = dfH2.loc[~(dfH2 == 0).all(axis=1)]
dfH3 = pd.DataFrame(pixelesH3, columns=columnas)
dfH3 = dfH3.loc[~(dfH3 == 0).all(axis=1)]
# Une los dos DataFrames de forma vertical
df_huevos = pd.concat([dfH1, dfH2, dfH3], ignore_index=True)
#Agrega una columna
df_huevos['objeto'] = 'huevo'
#print(df_huevos)

#####
#####Chiless#####
#####
imagenC1 = Image.open('OutputFruits/Class_3_Contour_1_1.png')
imagenC2 = Image.open('OutputFruits/Class_3_Contour_2_1.png')
imagenC3 = Image.open('OutputFruits/Class_3_Contour_3_1.png')
pixelesC1 = list(imagenC1.getdata())
pixelesC2 = list(imagenC2.getdata())
pixelesC3 = list(imagenC3.getdata())
# Convierte la lista de tuplas en un DataFrame de pandas
columnas = ['R', 'G', 'B']
dfC1 = pd.DataFrame(pixelesC1, columns=columnas)
dfC1 = dfC1.loc[~(dfC1 == 0).all(axis=1)]
dfC2 = pd.DataFrame(pixelesC2, columns=columnas)
dfC2 = dfC2.loc[~(dfC2 == 0).all(axis=1)]
dfC3 = pd.DataFrame(pixelesC3, columns=columnas)
dfC3 = dfC3.loc[~(dfC3 == 0).all(axis=1)]
# Une los dos DataFrames de forma vertical
df_chiles = pd.concat([dfC1, dfC2, dfC3], ignore_index=True)
#Agrega una columna
df_chiles['objeto'] = 'chile'
#print(df_chiles)

#####
#####Fondo#####
#####
imagenF1 = Image.open('OutputFruits/Class_4_Background_1.png')
pixelesF1 = list(imagenF1.getdata())
# Convierte la lista de tuplas en un DataFrame de pandas
columnas = ['R', 'G', 'B']
dfF1 = pd.DataFrame(pixelesF1, columns=columnas)
df_fondo = dfF1.loc[~(dfF1 == 0).all(axis=1)]
df_fondo.reset_index(drop=True, inplace=True)
#Agrega una columna
df_fondo['objeto'] = 'fondo'
#print(df_fondo)

#Une todos los dataframes
df_entrenamiento = pd.concat([df_platanos, df_huevos, df_chiles, df_fondo], ignore_index=True)
#print(df_entrenamiento)

#####
#Medias#

```

```

#####
# Filtra las filas que corresponden a la clase 'platano'
platano_df = df_entrenamiento.loc[df_entrenamiento['objeto'] == 'platano']
rgb_df_P = platano_df.iloc[:, :-1]
mediaP = tuple(rgb_df_P.mean().tolist())
mediaP = tuple(int(valor) for valor in mediaP)
print(mediaP)

# Filtra las filas que corresponden a la clase 'Huevo'
huevo_df = df_entrenamiento.loc[df_entrenamiento['objeto'] == 'huevo']
rgb_df_H = huevo_df.iloc[:, :-1]
mediaH = tuple(rgb_df_H.mean().tolist())
mediaH = tuple(int(valor) for valor in mediaH)
print(mediaH)

# Filtra las filas que corresponden a la clase 'Chile'
chile_df = df_entrenamiento.loc[df_entrenamiento['objeto'] == 'chile']
rgb_df_C = chile_df.iloc[:, :-1]
mediaC = tuple(rgb_df_C.mean().tolist())
mediaC = tuple(int(valor) for valor in mediaC)
print(mediaC)

# Filtra las filas que corresponden a la clase 'fondo'
fondo_df = df_entrenamiento.loc[df_entrenamiento['objeto'] == 'fondo']
rgb_df_F = fondo_df.iloc[:, :-1]
mediaF = tuple(rgb_df_F.mean().tolist())
mediaF = tuple(int(valor) for valor in mediaF)
print(mediaF)

#Entrenamiento
X = df_entrenamiento.iloc[:, :-1]
y = df_entrenamiento.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Crea una instancia de la clase GaussianNB
clf = GaussianNB()
# Entrena el modelo
clf.fit(X_train, y_train)

# Obtiene las predicciones del conjunto de pruebas
y_pred = clf.predict(X_test)

precision = precision_score(y_test, y_pred, average='macro')

# calcula la sensibilidad
recall = recall_score(y_test, y_pred, average='macro')

# calcula la exactitud
accuracy = accuracy_score(y_test, y_pred)

# calcula el F1-score
f1 = f1_score(y_test, y_pred, average='macro')

print("Precisión:", precision)
print("Sensibilidad:", recall)
print("Exactitud:", accuracy)
print("F1-score:", f1)

#####
# Clasifica un solo pixel#
#####
# pixel = pd.DataFrame({'R': [212], 'G': [184], 'B': [54]})
# prediccion = clf.predict(pixel)

img = cv2.imread('Prueba1_fruits.jpg')
color_list = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        # Obtener los valores RGB del píxel actual
        b, g, r = img[i,j]

        # Usa el modelo clasificador para predecir la clase del píxel actual
        pixel = pd.DataFrame({'R': [r], 'G': [g], 'B': [b]})
```

```
clase_predicha = clf.predict(pixel)

# Agrega una tupla con el color correspondiente a la lista de tuplas
if clase_predicha == 'platano':
    color_list.append(mediaP) # amarillo
elif clase_predicha == 'chile':
    color_list.append(mediac) # verde
elif clase_predicha == 'huevo':
    color_list.append(mediaH) # blanco
else:
    color_list.append(mediaF) # rojo

# Crea una nueva imagen con los colores clasificados
#nueva_img = np.array(color_list).reshape(img.shape)
print("Ya con la lista")
# Muestra la imagen clasificada
imagen = Image.new('RGB', (img.shape[0], img.shape[1]))
imagen.putdata(color_list)
imagen.show()
imagen.save('imagen_nueva_sklean.jpg')
```

VII. CONCLUSIONES

A lo largo de esta práctica, se tuvo como principal objetivo el lograr clasificar imágenes con 2, 3, o 4 regiones utilizando un clasificador Bayesiano. Con tal fin en mente, optamos por realizar una aplicación gráfica que permitiera al usuario realizar las máscaras de las imágenes manualmente seleccionando cada una de las clases que componen a las imágenes en los dos conjuntos de datos.

Sin lugar a dudas, la práctica fue muy útil para comprender de mejor manera el funcionamiento de los clasificadores Bayesianos, que, si bien son fáciles de implementar y en su mayoría arrojaron muy buenos resultados para clasificación de imágenes, pueden llegar a sufrir cuando existen problemas de clasificación con clases con atributos de color muy parecidos (poca distancia intraclase).

Además, la práctica sirvió como un gran ejercicio sobre manejo de imágenes en Python. Creemos que una de las principales dificultades a la que nos enfrentamos durante el desarrollo de la práctica fue el manejo de las imágenes para introducirlo como entrada en el modelo de clasificación Bayesiana. No obstante, haber trabajado con este tipo de datos, y haber logrado configurar nuestro clasificador Bayesiano nos permitirá encarar situaciones similares en un futuro, sirviendo incluso como práctica para poder implementar modelos más complejos como podrían ser redes neuronales para procesamiento de imágenes.

En este sentido, creemos que el objetivo de la práctica se cumplió en su totalidad, pues además de haber desarrollado nuestro propio programa y modelo de clasificación logramos realizar una comparativa de nuestro modelo contra modelos de clasificación bayesiana existentes, como lo es el modelo de Bayes de Scikit Learn. Con respecto a las métricas, pudimos apreciar grados de precisión similares, sin embargo, el modelo de Scikit Learn tuvo una sensibilidad y exactitud mayor. Pese a esto, nos encontramos sumamente satisfechos con los resultados obtenidos en la práctica, sobre todo, con nuestra implementación del modelo de clasificación de Bayes.

REFERENCIAS

- [1] IBM, “What are naive bayes classifiers?. <https://www.ibm.com/topics/naive-bayes>,” 2023.
- [2] Scikit-Learn, “Naive bayes. https://scikit-learn.org/stable/modules/naive_bayes.html,” 2023.
- [3] Auckland, “Gaussian filtering. <https://www.cs.auckland.ac.nz/courses>,” 2010.
- [4] O. M. Jimena, “Bayesian inference. laboratorio avanzado de procesamiento de imágenes.”