

Práctica 2. Clasificador bayesiano (noviembre, 2021)

1st Guerra Silva Erick Iván
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
erickivanguerra0.0@gmail.com

3rd Martínez Gutiérrez Carlos Giovanni
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
cgiovanni@comunidad.unam.mx

2nd Lázaro Martínez Abraham Josué
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
abrahamlazaro@comunidad.unam.mx

4th Castillo Sánchez Axel
Ingeniería Mecatrónica
Universidad Nacional Autónoma de México
 Ciudad de México, México
axel_castillo98@hotmail.com

Resumen—Este documento presenta los resultados obtenidos para los ejercicios planteados en la práctica para crear un clasificador Bayesiano. Se utilizaron diferentes librerías para el procesamiento de imágenes al igual que se utilizaron diferentes funciones para la creación del clasificador Bayesiano. Toda la práctica se realizó en lenguaje Python utilizando Google Collaboratory y se utilizaron librerías como Matplotlib, OpenCV, Scikit-Image, PIL y Sci-Py.

Índice de Términos—Procesamiento de Imágenes, Python, pixel, matriz, covarianza, clasificador, regiones, Bayes.

I. INTRODUCCIÓN

Clasificador Bayesiano

El concepto de clasificación tiene dos significados:

- No supervisada: dado un conjunto de datos, establecer clases o agrupaciones (clusters)
- Supervisada: dadas ciertas clases, encontrar una regla para clasificar una nueva observación dentro de las clases existentes.

Los modelos de Naive Bayes son una clase especial de algoritmos de clasificación de Aprendizaje Automático, o Machine Learning, en ellos se asume que las variables predictoras son independientes entre sí. En otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica.

Proporcionan una manera fácil de construir modelos con un comportamiento muy bueno debido a su simplicidad.

Lo consiguen proporcionando una forma de calcular la probabilidad ‘posterior’ de que ocurra un cierto evento A, dadas algunas probabilidades de eventos ‘anteriores’.[1]

Filtro Gaussiano

El filtrado gaussiano es el proceso de promedio ponderado de toda la imagen. El valor de cada píxel se obtiene mediante el promedio ponderado de sí mismo y otros valores de píxeles en la vecindad. La operación específica del filtrado gaussiano es usar una plantilla (o convolución, máscara) para escanear cada píxel en la imagen y usar la escala de grises promedio ponderada de los píxeles en la vecindad determinada por la plantilla para reemplazar el valor del píxel central de la plantilla.[2]

Media

Es el valor que se obtiene al dividir la suma de un conglomerado de números entre la cantidad de ellos.

Algunas características de la media son:

- Considera todas las puntuaciones
- El numerador de la fórmula es la cantidad de valores
- Cuando hay puntuaciones extremas, no tiene una representación exacta de la muestra.[3]

RECONOCIMIENTO DE PATRONES. PRÁCTICA 2

Covarianza

Es el valor a través del cual se refleja en qué cuantía don variables cualesquiera varían de forma conjunta respecto de sus medias aritméticas. Así, esta medida nos permite conocer cómo se comportan las variables en cuestión respecto de otras variables. Es decir, ¿qué hace la variable X cuando Y aumenta? ¿Y cuándo Y disminuye? ¿Y si Y se mantiene estable y constante?

La covarianza puede adquirir valores negativos y positivos, y además puede adquirir valores iguales a 0.[4]

Pixel

Es el elemento más pequeño de una imagen reproducida digitalmente. En un monitor o en la pantalla de un teléfono móvil se suelen alinear varios píxeles en una trama. La combinación de varios píxeles constituye una imagen rasterizada.[5]

II. OBJETIVO

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

III. RESULTADOS

A continuación, se presentan los ejercicios desarrollados en esta práctica. Antes de realizar los ejercicios se realizó la descarga de las imágenes para poder trabajar con ellas.

Para poder realizar el entrenamiento, a partir de las imágenes, primero se seleccionaron las imágenes a utilizar en esta práctica, en este caso utilizaremos las imágenes de la comida.

Entrenamiento.

1. Realizar preprocessamiento de sus imágenes con un filtro gaussiano.

En el preprocessamiento, se utilizó *OpenCV* de *cv2*, para poder aplicar un filtro Gaussiano a cada una de las imágenes que se utilizarán en el clasificador, la aplicación de este filtro sirve para que los colores de la imagen se estandarice y de esa manera no existan tantos detalles en ella.

```
img = cv2.imread(monos[0])
blur = cv2.GaussianBlur(img, (21,21), 0)
```



Fig. 1. Imagen con la aplicación de Filtro Gaussiano

Para poder aplicar el filtro Gaussiano a cada una de las imágenes de nuestro DataSet, se creó la función llamada

aplicar_filtro, la cual funciona en conjunto con la función *generar_dataset_blur*.

La primera función recibe como parámetros la imagen, la intensidad de x y la intensidad de y para poder aplicar el filtro Gaussiano a una imagen, el return de esta función es la imagen con el filtro Gaussiano ya aplicado.

```
def
aplicar_filtro(img,intensidad_x,intensidad_y=None):
:
```

Para poder aplicar el filtro a cada imagen, se utiliza la función *generar_dataset_blur*, esta función recibe como parámetros el conjunto de imágenes a aplicar el filtro, la intensidad de x y la intensidad de y, el return de es una lista de las imágenes con el filtro Gaussiano ya aplicado a cada una.

```
imagenes_filtradas=[aplicar_filtro(cv2.imread(url)
,intensidad_x, intensidad_y) for url in urls]
```

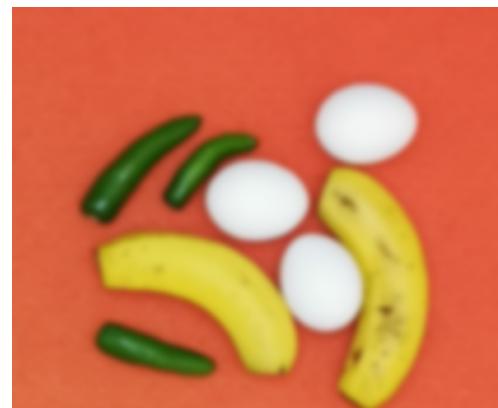


Fig. 2. Imagen 1 después de aplicar el filtro Gaussiano al DataSet

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

Seleccionamos las imágenes que deben tener regiones contiguas. En este momento el dataset ya tiene imágenes con zonas contiguas

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

Para la creación de las máscaras por clase en el DataSet, se realizaron dos métodos, en el primer método, se utilizó *K-Means* para segmentar las diferentes clases. K-Means es un algoritmo de clustering, en este caso realizamos clusters de píxeles para reducir así el número de colores en la imagen y posteriormente segmentar de manera más sencilla esta imagen con menos colores en las diferentes clases (huevos, chiles, plátanos y fondo).

Lo primero que se realizó fue importar las librerías necesarias, en este caso fue *KMeans* del módulo

RECONOCIMIENTO DE PATRONES. PRÁCTICA 2

sklearn.cluster. La primera función implementada es la función *segmentar*, esta función recibe como parámetros la imagen a segmentar y el número de colores que se desea que tenga la imagen final. En este caso fueron 5 colores, uno para cada clase y 1 uno más extra para que el algoritmo no fuera demasiado estricto y no tuviera errores al intentar clasificar. Esta función separaba los canales R, G y B de una imagen, una vez con estos canales por separado se redimensionaron los vectores y se concatenaron para formar una matriz y posteriormente aplicar el algoritmo KMeans. Se creó una nueva matriz para guardar los nuevos valores para los píxeles, posteriormente, los diferentes canales se combinaron para obtener nuevamente una imagen RGB y poder mostrarla utilizando *matplotlib*. A continuación, se anexa una imagen para mostrar los resultados después de aplicar el algoritmo K-Means a una imagen.



Fig. 3. Imagen después de aplicar el algoritmo K-Means

Como podemos observar la imagen anterior, ahora únicamente tenemos 5 colores, mismos que representan las diferentes clases en la imagen, haciendo más sencillo el proceso de segmentación por clases.

La siguiente función realizada es la función *segment_by_color*, esta función tomaría la imagen obtenida con K-Means y separaría los diferentes colores para obtener así las diferentes máscaras para entrenar nuestro modelo. En esta función lo primero que se realiza es pasar de RGB a BGR ya que la función *inRange* de OpenCV recibe una imagen BGR. Una vez con la imagen en BGR, se utiliza la función *inRange* para segmentar las imágenes por rango de colores.

Otra función importante es la función *generar_máscaras*, la cual se encarga de generar las diferentes máscaras de todas las imágenes de entrenamiento. Esta función utiliza las funciones descritas anteriormente y almacena las máscaras obtenidas en diccionarios para poder separar por clases y por imagen las máscaras generadas. A continuación, se muestran ejemplos de las máscaras generadas después de este proceso.

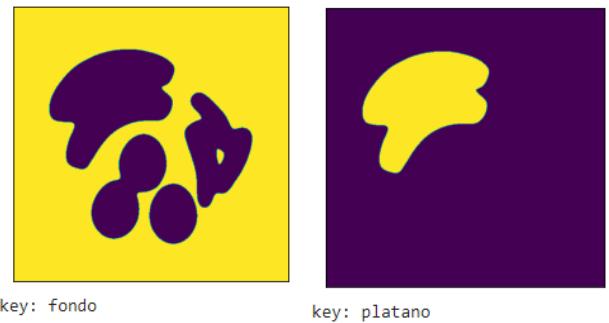


Fig. 4. Ejemplos de máscaras generadas con el primer método de segmentación

Una vez teniendo las máscaras de las diferentes clases también es posible recuperarlas en los colores originales, esto se logra multiplicando la imagen original por la máscara. A continuación, se muestran ejemplos de las máscaras obtenidas con colores originales.

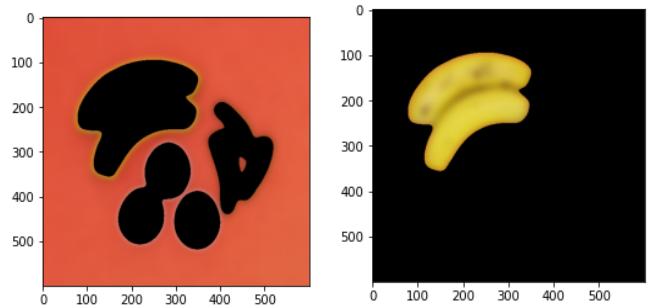


Fig. 5. Ejemplos de máscaras con color original generadas con el primer método de segmentación

Para la creación de las máscaras con el segundo método ‘A mano’, se utilizó el editor de fotografías *Photoshop*, con la ayuda de este editor, obtuvimos las clases, recortado la clase de interés y relleno con negro, las clases que no eran de interés. A continuación, se muestran ejemplos de las máscaras generadas después de este proceso.



Fig. 6. Clases de una imagen, a partir del segundo método.

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

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 la creación del clasificador Bayesiano, creamos

RECONOCIMIENTO DE PATRONES. PRÁCTICA 2

una clase llamada *Bayes*, en esta clase creamos 4 funciones para poder implementar el clasificador, las funciones se explicarán a continuación.

Carga de imágenes

Esta función nos ayuda a leer las imágenes y guardarlas en un diccionario, esto según su clase, el return de la misma es el diccionario de imágenes por clase.

Primero creamos el diccionario en donde se encontrarán las clases.

```
dict_imagenes = {k:list() for k in self.clases}
```

Posteriormente se leerán las imágenes de nuestro DataSet, para la lectura de imágenes se utiliza la función imread del módulo cv2 y el color pasa de BGR a RGB, con la función cvtColor del módulo cv2, cuando termina la conversion de color, las imágenes se guardaran en un diccionario con su respectiva clase.

```
img = cv2.imread(url)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
dict_imagenes[clase].append(img)
```

Esta funciona, también nos muestra el número de imágenes leídas para cada clase y el número total de imágenes leídas.

```
=====
CARGANDO IMAGENES
=====
Imagenes leidas para la clase platano: 4
Imagenes leidas para la clase chile: 4
Imagenes leidas para la clase fondo: 4
Imagenes leidas para la clase huevo: 4
Imagenes leidas en total: 16
```

Fig. 7. Resultado de la función *cargar_imagenes*.

Cálculo de probabilidades a priori

En esta función se calcula probabilidad a priori de la siguiente manera, primero creamos un diccionario, para poder guardar las probabilidades a priori, posteriormente creamos dos variables, las cuales nos ayudara como contadores, la primera contará el número de píxeles por clase y la segunda contará el número de píxeles totales de cada imagen, para poder contar el número de píxeles por cada clase, primero sumaremos la componente RGB de cada pixel de la siguiente manera.

```
suma_rgb = np.sum(imagen,axis=2)
```

Despues de tener la suma de la componente RGB, guardaremos en una nueva variable, la suma de la componente RGB, que sea diferente de cero, esto con el objetivo de poder

quitar de la suma los píxeles que son de color negro.

Por otra parte, para poder obtener el número de todos los píxeles, obtenemos el número de píxeles en *x* y el número de píxeles en *y*, para poder obtener el total a partir de una multiplicación.

Por último en el diccionario que creamos al inicio de nuestra función, lo ocuparemos para guardar las probabilidades a priori de cada clase, para obtener las probabilidades a priori, dividiremos la cantidad de píxeles de cada clase entre la cantidad de píxeles total de la imagen, al terminar de realizar el proceso descrito anteriormente, la función imprime la probabilidad a priori por clase, el número de píxeles totales de todas las imágenes, el total de píxeles por cada clase y por ultimo se muestra el resumen de cada una de las probabilidades de cada clase..

```
=====
CALCULANDO PROBABILIDADES A PRIORI
=====
```

```
Probabilidad a priori clase platano: 0.14619444444444443
Pixel de la clase en todas las imagenes: 210520
Pixel totales en las imágenes: 1440000
```

```
Probabilidad a priori clase chile: 0.08455416666666667
Pixel de la clase en todas las imagenes: 121758
Pixel totales en las imágenes: 1440000
```

```
Probabilidad a priori clase fondo: 0.8238534722222223
Pixel de la clase en todas las imagenes: 1186349
Pixel totales en las imágenes: 1440000
```

```
Probabilidad a priori clase huevo: 0.11321180555555556
Pixel de la clase en todas las imagenes: 163025
Pixel totales en las imágenes: 1440000
```

Resumen probabilidades:	
clase	probabilidad
platano	14.6194%
chile	8.4554%
fondo	82.3853%
huevo	11.3212%

Fig. 8. Resultado de la función *calc_probs_priori*.

Cálculo de las medias

Para poder calcular la media de todas las clases, ocupamos otra funciones para poder obtener el resultado, la primer función llamada *cal_media_imagen*, esta función nos regresa la suma de todos los píxeles en sus componentes RGB, sin contar a los píxeles negros, la siguiente función que utilizamos, fue la de *cal_media_clase*, con esta función calculamos la media de cada clase, a partir de una lista de imágenes, primero se crea un vector de tres espacios, en donde se almacenara la media, posteriormente se usara la función *cal_media_imagen*, como se mencionó anteriormente, esta función nos regresa la la suma de todos los píxeles en sus componentes RGB, sin contar a los píxeles negros al obtener la cantidad de pixeles,, los agregamos al vector creado, ademas de guardarlo en una variable llamada *cantidad_pixeles*, al finalizar este procedimiento , la función muestra el número de pixeles de cada clase, la suma de R's,

RECONOCIMIENTO DE PATRONES. PRÁCTICA 2

G's y B's de todas las imágenes y la media de cada clase, esto último se obtiene de la división de nuestro vector con la cantidad perteneciente a cada clase.

Para poder obtener la media de cada clase, se utiliza la función *calc_medias*, en donde se crea un diccionario para poder guardar los resultados de cada clase con la ayuda de la función *cal_media_clase*.

CALCULANDO MEDIA DE LAS CLASES

```
Media de la clase platano
Cantidad de pixeles pertenecientes a la clase: 210520
Suma de R's, G's y B's de todas las imágenes:
[35194646. 31027721. 10184204.]
```

Media de la clase:

```
[167.17958389 147.38609633 48.37642029]
```

Fig. 9. Media de la clase plátano

Media de la clase huevo

```
Cantidad de pixeles pertenecientes a la clase: 144770
Suma de R's, G's y B's de todas las imágenes:
[27973771. 26270664. 26101362.]
```

Media de la clase:

```
[193.22905989 181.46483387 180.29537888]
```

Fig. 10. Media de la clase huevo

Algo interesante que podemos observar en la imagen anterior es que en las medias para los canales RGB de la clase huevo obtenemos valores similares, recordando las combinaciones de colores, al tener valores parecidos en los tres canales se está representando un color blanco que corresponde al color de un huevo.

Cálculo de covarianzas

Por otra parte, también se realizó el cálculo de la covarianza, se realizaron 3 diferentes funciones para el cálculo de las covarianzas. La función *covarianza_imagen* se encarga de calcular la matriz de covarianza para una imagen. La siguiente función es la función *covarianza_clase*, esta función se encarga de calcular la covarianza de todas las imágenes que pertenecen a una clase, esta función utiliza la función anterior para calcular la covarianza de todas las imágenes que pertenecen a esa clase, algo importante a destacar es que en este punto se tiene una covarianza acumulada, por lo que se realiza una división entre el número total de pixeles para así obtener la covarianza de toda la clase. Finalmente, tenemos la función *calc_cov*, la cual se encarga de calcular la covarianza de todas las clases. A continuación, se agrega un ejemplo de matriz de covarianzas para una clase.

CALCULANDO COVARIANZA DE LAS CLASES

Covarianza de la clase platano

```
Matriz acumulada de covarianzas:
[[1.70376496e+09 1.53981193e+09 4.81993420e+08]
 [1.53981193e+09 1.41533856e+09 4.48865881e+08]
 [4.81993420e+08 4.48865881e+08 1.75410867e+08]]
```

Cantidad de pixeles pertenecientes a la clase: 210520

Covarianza de la clase:

```
[[8093.12634739 7314.32607943 2289.53743138]
 [7314.32607943 6723.05987414 2132.17690092]
 [2289.53743138 2132.17690092 833.22661481]]
```

Fig. 11. Matriz de covarianza de la clase plátano

Cálculo del inverso del determinante

Otro dato necesario para la programación del clasificador de Bayes es el inverso del determinante. Este dato se calcula en la función *calc_inversa_det*, en esta función lo primero que se realiza es calcular la matriz inversa de las matrices de covarianza y posteriormente calcular el determinante de cada una.

Prueba

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).

Una vez con los cálculos necesarios, se programan las funciones para realizar las predicciones, en este caso se programaron las funciones *predecir* y *predecir_pixel*. En la función *predecir_pixel*, se realiza la predicción de la clase a la que pertenece un solo píxel. Esto se realiza utilizando los cálculos obtenidos con las funciones anteriores aplicados en la ecuación 1, esta función retorna la clase a la que tiene mayor probabilidad de pertenecer el pixel.

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

Ecuación 1. Clasificador Bayesiano

Por otra parte, en la función *predecir*, se aplica la función *predecir_pixel* a todos los píxeles de una imagen y se asigna un tono en escala de grises a cada clase para así poder obtener una imagen coloreada según la clasificación realizada.

Para poder clasificar los píxeles de la imagen, primero se crea una lista con las imágenes de prueba a utilizar, estas imágenes, para la lectura de imágenes se utiliza la función *imread* del módulo *cv2* y el color pasa de BGR a RGB, con la función *cvtColor* del módulo *cv2*, posteriormente se les aplica el filtro Bayesiano y se guardan en un lista.

```
img = cv2.imread(url)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = aplicar_filtro(img, 51)
```

RECONOCIMIENTO DE PATRONES. PRÁCTICA 2

```
list_imagenes.append(img)
```

Después de tener en una lista las imágenes de prueba, ocupamos el método de *predecir* para generar la predicción de nuestro modelo, pasándole la imagen al método y guardando los resultados en una lista, en esta lista aparecerán la clase con un valor asignado de grises.

```
resultado = modelo.predecir(img)
list_resultados.append(resultado)

{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}
```

Fig. 12. Clases de una imagen, a partir del segundo método.

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.

Después de la realización de los dos métodos descritos anteriormente (*k-means* y *a mano*), se obtuvieron los siguientes resultados.

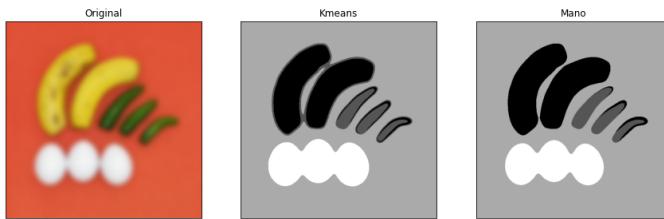


Fig. 13. Comparación entre los dos métodos, con la primera imagen de prueba.

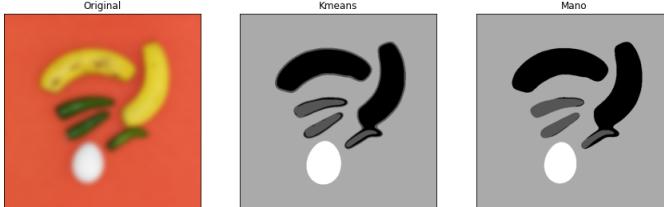


Fig. 13. Comparación entre los dos métodos, con la segunda imagen de prueba..

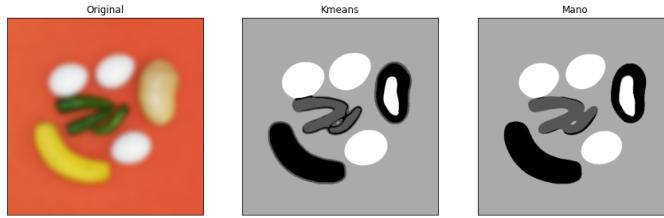


Fig. 13. Comparación entre los dos métodos, con la tercera imagen de prueba.

8. 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.

El clasificador de Naïves Bayes, por la naturaleza de su implementación, es inherentemente un modelo que se debe desarrollar en un entorno de aprendizaje supervisado, es decir, para entrenar a nuestro modelo debemos tener pares de variables X, Y: el dato o la variable de entrada y su respectiva etiqueta o *label*.

La librería scikit-learn ofrece dos funciones para realizar un clasificador bayesiano: *GaussianNB()* y *MultinomialNB()*. La primera simplemente es una clasificación binaria y la segunda es una clasificación de más de dos categorías.

Para esta práctica en específico, usamos la segunda función debido a que tenemos, por lo menos, cinco categorías (fondo, plátano, chile, huevo y papa) en las cuales debemos clasificar los pixeles.

El código de la librería para realizar el entrenamiento y la predicción es, en realidad, bastante sencillo. Únicamente es necesario exportar la librería, crear el objeto del clasificador multivariante, usar el método *fit* que recibe las variables y sus etiquetas y, finalmente, hacer la predicción con un conjunto de pruebas usando el método *predict*.

```
clf = MultinomialNB()
clf.fit(X, y)
print(clf.predict(X[2:3]))
```

La parte que resultó ser más larga y un tanto complicada fue la generación del etiquetado de cada uno de los pixeles de las imágenes de entrenamiento. Esta consistió en la generación de dos funciones principales que denominamos *generar_pixeles_clases* y *etiquetar_pixeles*.

La función *generar_pixeles_clases* se encarga de generar un arreglo que contendrá todos los pixeles de las imágenes de entrenamiento y la función *etiquetar_pixeles* simplemente se encargará de generar las etiquetas a cada uno de los pixeles del arreglo generado.

De esta manera implementamos el clasificador con la librería y, a continuación, se puede ver la comparación entre los tres clasificadores que realizamos: utilizando k-means, haciendo el clasificador bayesiano “a mano” y el de la librería.

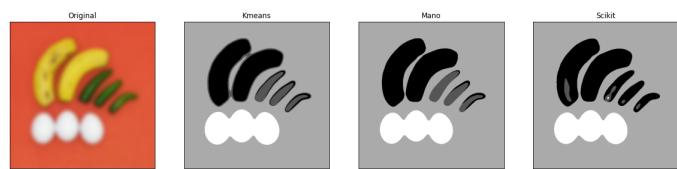


Fig. 14. Comparación entre los tres métodos, con la primera imagen de prueba.

RECONOCIMIENTO DE PATRONES. PRÁCTICA 2

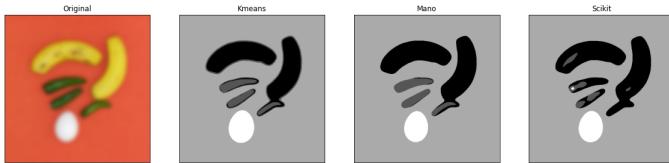


Fig. 14. Comparación entre los tres métodos, con la segunda imagen de prueba.

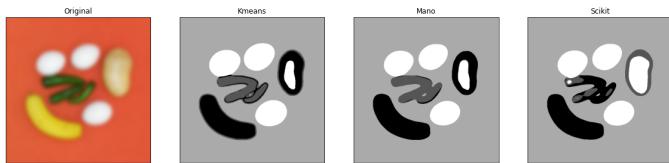


Fig. 14. Comparación entre los tres métodos, con la tercera imagen de prueba.

Podemos observar que el clasificador con las máscaras segmentadas por k means y de manera manual obtienen resultados muy similares. Para el caso del clasificador utilizando las máscaras de K-Means se obtienen algunos errores en los bordes de los chiles que se detectan como plátano. Para el caso del clasificador con las máscaras obtenidas ‘a mano’ se obtienen errores mínimos al clasificar el chile, clasificando los bordes del chile como plátano. Por otra parte, utilizando el clasificador de la librería Scikit-learn se obtienen resultados no muy buenos, obteniendo notables errores de clasificación, principalmente en los chiles que son clasificados también como plátanos e incluso con unas partes de huevo. Finalmente, podemos confirmar que nuestro clasificador es mucho mejor que el de la librería Scikit-learn, esto puede ser principalmente porque nuestro modelo fue creado especialmente para este dataset y el clasificador de la librería funciona con cualquier otro dataset.

Al agregar la papa podemos observar que está mal clasificada: se considera parcialmente como plátano y huevo. Creemos que esto sucede debido a que utilizamos un aprendizaje supervisado. Al intentar clasificar un nuevo elemento, que no estaba presente durante el entrenamiento, el clasificador lo etiqueta con base en la información que se tiene. Por lo tanto, no es posible identificar un objeto con el cual no se tuvo un previo conocimiento.

Algo interesante que observamos con la clasificación de la papa es que se aproxima a plátano y huevo ya que el color de la papa está entre ambos colores. La clasificación de este elemento es incorrecta pero creemos que esto pasaría ya que es un elemento completamente nuevo para nuestro modelo y no hay manera de clasificarlo correctamente ya que no existe una clase previamente conocida para ese elemento.

IV. CONCLUSIONES

De manera general, la práctica nos sirvió mucho para entender todo el pipeline del reconocimiento de patrones, partiendo desde el preprocesamiento hasta la clasificación, en

este caso utilizamos el algoritmo de Naive Bayes para clasificar. La práctica fue interesante ya que pudimos entender más allá de la parte matemática que habíamos revisado en clase, pudiendo implementar un clasificador haciendo uso de estos cálculos, entrenando el modelo y realizando predicciones con imágenes completamente desconocidas para el modelo.

Otra cosa importante a destacar es el uso del filtro Gaussiano en el preproceso de imágenes. El uso de este filtro nos permitió homogeneizar las imágenes, reduciendo bordes y colores para segmentar las clases de una forma más sencilla y precisa tanto por K-Means como ‘a mano’.

En la realización de este clasificador Bayesiano, se crearon dos estrategias de preproceso de las imágenes con las que trabajaría nuestro clasificador Bayesiano, creando máscaras ‘a mano’, y creando máscaras utilizando el algoritmo K-Means. Podemos concluir que la generación de máscaras ‘a mano’ tiene ligeramente mejores resultados que utilizando el algoritmo K-Means, sin embargo, con el algoritmo K-Means fue más sencillo y se ahorró mucho más tiempo; tomando en cuenta ambos factores, podemos decir que es mejor crear máscaras para el preproceso y entrenamiento de un modelo utilizando el algoritmo K-Means, siendo rápido, sin necesidad de una herramienta extra como ‘Photoshop’ y además, obteniendo muy buenos resultados al momento de clasificar.

Cabe destacar que al momento de clasificar obtenemos probabilidades mayores a 1, esto puede ser porque se están compartiendo píxeles entre clases es decir, se están tomando los mismos pixeles para diferentes clases, haciendo que se obtenga un número mayor de píxeles de los que está formada la imagen. Para evitar este problema se debería realizar una limpieza en las imágenes para excluir los píxeles por clase y así tener la cantidad de píxeles para obtener una probabilidad total de 1.

REFERENCIAS

- [1] Roman, V. (2019, 29 abril). “Algoritmos Naive Bayes: Fundamentos e Implementación”. Medium. <https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fundamentos-e-implementaci%C3%B3n-4bcb24b307f> (accessed Oct. 25, 2021).
- [2] “Filtro gaussiano OpenCV” - programador clic. (s. f.). Programador Clicl. <https://programmerclick.com/article/64011923547/> (accessed Oct. 25, 2021).
- [3] Ortega, C. (2021, 13 julio). “¿Qué es la media, la mediana y la moda?” QuestionPro. <https://www.questionpro.com/blog/es/la-media-la-mediana-y-la-moda/> (accessed Oct. 25, 2021).
- [4] Software DELSOL. (2019, 3 junio). “Covarianza”. sdelsol. <https://www.sdelsol.com/glosario/covarianza/> (accessed Oct. 25, 2021).
- [5] ¿Qué es un píxel? (2021, 7 junio). IONOS <https://www.ionos.mx/digitalguide/paginas-web/diseno-web/que-es-un-pixel/> (accessed Oct. 25, 2021).

▼ Preparativos

```
import numpy as np
from matplotlib import pyplot as plt
import os
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow
import cv2
import pandas as pd
from collections import Counter
```

Hay que copiar las imágenes en el drive para poder trabajar con ellas. Link de descarga: [carpetas de descarga](#)

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

+ Code

+ Text

▼ Entrenamiento

```
path = '/content/drive/MyDrive/Aux/'
# Listas de paths de imagenes
monos = [path+"monos/"+file for file in os.listdir(path+"monos/")]
aerea = [path+"aerea/"+file for file in os.listdir(path+"aerea/")]
comida = [path+"Comida/"+file for file in os.listdir(path+"Comida/")]
```

```
#monos_train, monos_test = monos[:-1], monos[-1]
#aerea_train, aerea_test = aerea[:-1], aerea[-1]
comida_train = [x for x in comida if "Entrenamiento" in x]
comida_test = [x for x in comida if "Prueba" in x]
```

```
# Este es random, no nos conviene tanto
# generamos los conjuntos de entrenamiento y de prueba
# monos_test, monos_train = train_test_split(monos, test_size=0.2)
# aerea_test, aerea_train = train_test_split(aerea, test_size=0.2)
# comida_test, comida_train = train_test_split(comida, test_size=0.2)
```

▼ 1 Preprocesamiento

Realizar *preprocesamiento* de sus imágenes con un filtro gaussiano

Debemos aplicar un filtro **Gaussiano** a la imagen para que los colores se estandaricen y no existan tantos detalles

▼ Con PIL

```
from PIL import Image, ImageFilter

image = Image.open(monos[0])
```

```

# se aplica un filtro gaussiano predefinido
blur = image.filter(ImageFilter.GaussianBlur)
# mostrando la imagen
plt.figure(figsize=(15,15))
plt.subplot(121),plt.imshow(image),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Filtrada')
plt.xticks([]), plt.yticks([])

# https://www.geeksforgeeks.org/apply-a-gauss-filter-to-an-image-with-python/

```

(([], <a list of 0 Text major ticklabel objects>),
 ([], <a list of 0 Text major ticklabel objects>))

Original



Filtrada



▼ Con OpenCV

```

import cv2

img = cv2.imread(monos[0])

blur = cv2.GaussianBlur(img,(21,21),0)

plt.figure(figsize=(15,15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Filtrada')
plt.xticks([]), plt.yticks([])
plt.show()

```

#https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html



▼ Aplicación al dataset

Ocupamos OpenCV por que las demás partes ocupan OpenCV para la modificación de la imagen

```
def aplicar_filtro(img, intensidad_x, intensidad_y=None):
    '''Aplica cierta intensidad de filtro gaussiano

    return imagen con el filtro gaussiano aplicado
    ...
    if intensidad_y==None: intensidad_y = intensidad_x
    if intensidad_x%2!=1: intensidad_x += 1
    if intensidad_y%2!=1: intensidad_y += 1

    post = cv2.GaussianBlur(img,(intensidad_x,intensidad_y),0)
    return post

def generar_dataset_blur(urls, intensidad_x, intensidad_y=None):
    '''Genera una lista de imágenes a las que se les aplico el filtro
    gaussiano

    return lista imágenes filtradas
    ...

    imágenes_filtradas = [ aplicar_filtro(cv2.imread(url), intensidad_x, intensidad_y)
                           for url in urls]

    return imágenes_filtradas

#monos.blur_train = generar_dataset_blur(monos_train, 21)
#comida.blur_train = generar_dataset_blur(comida_train, 21)
#aerea.blur_train = generar_dataset_blur(aerea_train, 21)

cv2_imshow(comida.blur_train[1])

import matplotlib.image as pltim

for i in range(len(comida.blur_train)):
    print(comida_train[i])
    imagen = cv2.cvtColor(comida.blur_train[i], cv2.COLOR_BGR2RGB)
    pltim.imsave("/content/drive/MyDrive/Aux/ComidaBlur/"
                +comida_train[i].split("/")[-1],imagen)
    plt.imshow(imagen)
    plt.show()
```

2 Selección de imágenes

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

Seleccionamos las imágenes que deben tener regiones contiguas. En este momento el dataset ya tiene imágenes con zonas contiguas

▼ 3 Generar máscaras por clase

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

Generaremos las máscaras para identificar las clases en nuestro dataset

▼ Por K-Means

Aplicamos K-Means para segmentar imágenes en colores y posteriormente aplicar una máscara, seleccionando las diferentes clases en la imagen.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from PIL import Image, ImageFilter

def segmentar(img, n=5):
    I1 = np.asarray(img,dtype=np.float32)/255

    # Extraemos cada canal por separado
    R = I1[:, :, 0]
    G = I1[:, :, 1]
    B = I1[:, :, 2]

    # Aplanamos los canales, para que sean vectores en vez de
    # matrices
    XR = R.reshape((-1, 1))
    XG = G.reshape((-1, 1))
    XB = B.reshape((-1, 1))

    # concatenamos los vectores para generar una matriz de
    # (width*height,3)
    X = np.concatenate((XR,XG,XB),axis=1)

    # aplicamos el algoritmo de kmeans con n grupos (colores)
    n = 5
    k_means = KMeans(n_clusters=n)
    k_means.fit(X)

    # Obtenemos los centros de los grupos, y las etiquetas de estos
    # las etiquetas nos indican a que indice de grupos (definidos por
    # los centroides) pertenece cada pixel
    centroides = k_means.cluster_centers_
    etiquetas = k_means.labels_

    # Creamos un nuevo arreglo indicando el nuevo valor para cada pixel
    # en cada una de sus 3 componentes
    m = XR.shape
    for i in range(m[0]):
        XR[i] = centroides[etiquetas[i]][0]
        XG[i] = centroides[etiquetas[i]][1]
        XB[i] = centroides[etiquetas[i]][2]
    # redimensionamos para que sean matrices
    XR.shape = R.shape
    XG.shape = G.shape
    XB.shape = B.shape
    #
```

```

XR = XR[:, :, np.newaxis]
XG = XG[:, :, np.newaxis]
XB = XB[:, :, np.newaxis]

# concatenamos las componentes rgb
Y = np.concatenate((XR,XG,XB),axis=2)

# imprimimos la imagen original y la segmentada
plt.imshow(img)
plt.xticks([]), plt.yticks([])
plt.show()
plt.imshow(Y)
plt.xticks([]), plt.yticks([])
plt.show()

return Y, centroides

```

```

def segment_by_color(img_bgr, r_a, r_b):
    #
    # TODO:
    # - Change color space from RGB to HSV.
    #   Check online documentation for cv2.cvtColor function
    # - Determine the pixels whose color is in the color range of the ball.
    #   Check online documentation for cv2.inRange
    # - Calculate the centroid of all pixels in the given color range (ball position).
    #   Check online documentation for cv2.findNonZero and cv2.mean
    # - Calculate the centroid of the segmented region in the cartesian space
    #   using the point cloud 'points'. Use numpy array notation to process the point cloud data.
    #   Example: 'points[240,320][1]' gets the 'y' value of the point corresponding to
    #   the pixel in the center of the image.
    # Return a tuple of the form [img_c, img_r, x, y, z] where:
    # [img_c, img_r] is the centroid of the segmented region in image coordinates.
    # [x,y,z] is the centroid of the segmented region in cartesian coordinate.
    #

    img_bin = cv2.inRange(img_bgr, r_a, r_b)
    plt.imshow(img_bin)
    plt.xticks([]), plt.yticks([])
    plt.show()
    #plt.imshow(cv2.cvtColor(img_bin, cv2.COLOR_BGR2RGB))
    #img_bin = cv2.inRange(img_bgr, r_a, r_b)
    #cv2_imshow(img_bin)

    #indices = cv2.findNonZero(img_bin)
    #[img_x, img_y, a, b] = cv2.mean(indices)
    #print([img_x, img_y])

    ...
    counter = 0
    x = 0
    y = 0
    z = 0
    for [[c,r]] in indices:
        xt = points[r,c][0]
        yt = points[r,c][1]
        zt = points[r,c][2]
        if math.isnan(xt) or math.isnan(yt) or math.isnan(zt):
            continue
        [x,y,z, counter] = [x + xt, y + yt, z + zt, counter+1]
        x = x/counter if counter > 0 else 0

```

```

y = y/counter if counter > 0 else 0
z = z/counter if counter > 0 else 0'''

return img_bin

def mostrar_grupos(img, colores, keys):
    dic_colores = {k:list() for k in keys}
    dic_masks = {k:list() for k in keys}

    for color in colores:
        mask = segment_by_color(img, color, color)
        clase = input("key: ")
        dic_colores[clase].append(color)
        dic_masks[clase].append(mask)

    return dic_colores, dic_masks

def generar_mascaras(keys, dataset):

    diccionarios_keys = list()
    diccionarios_masks = list()
    agrupadas = list()

    for i in range(len(dataset)):
        # Se aplica Kmeans
        Y, centroides = segmentar(cv2.cvtColor(dataset[i], cv2.COLOR_BGR2RGB))
        agrupadas.append(Y)

        # Se generan las máscaras
        conjuntos, masks = mostrar_grupos(Y, centroides, keys)
        diccionarios_keys.append(conjuntos)
        diccionarios_masks.append(masks)

    # Se juntan los diccionarios
    full_dict_ranges = {k:list() for k in keys}
    for dic in diccionarios_keys:
        for key in dic.keys():
            full_dict_ranges[key].extend(dic[key])

    # Se juntan las máscaras que sean de la misma clase de la
    # misma imagen
    for dict_clases in diccionarios_masks:
        for key in dict_clases:
            if len(dict_clases[key]) > 1:
                new_mask = np.zeros(dict_clases[key][0].shape)
                for array in dict_clases[key]:
                    new_mask += array
                dict_clases[key] = new_mask
            else:
                dict_clases[key] = dict_clases[key][0]

    rangos = {k:None for k in keys}
    for key in full_dict_ranges.keys():
        Aux = np.array(full_dict_ranges[key])
        rangos[key] = [np.min(Aux, axis=0), np.max(Aux, axis=0)]

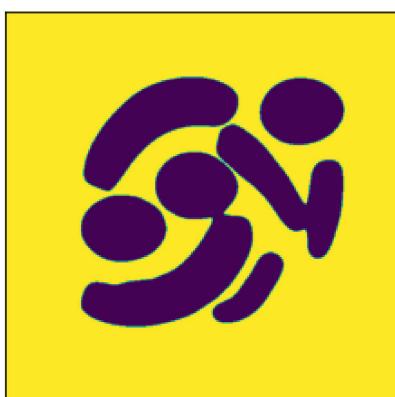
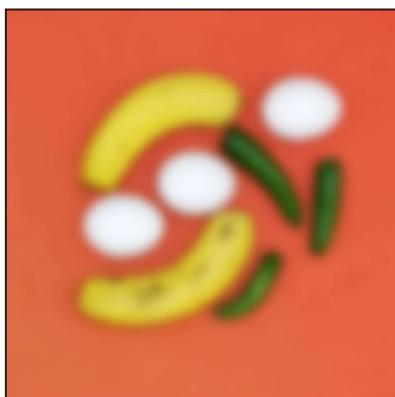
    return diccionarios_masks, rangos, agrupadas

```

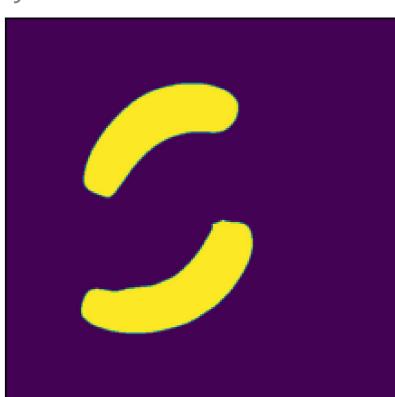
```

keys = ["huevo", "fondo", "platano", "chile"]
diccionarios_masks, rangos, agrupadas = generar_mascaras(keys, comida.blur_train)

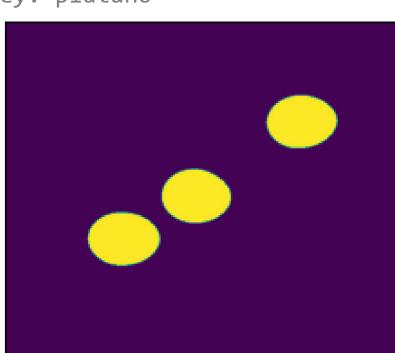
```

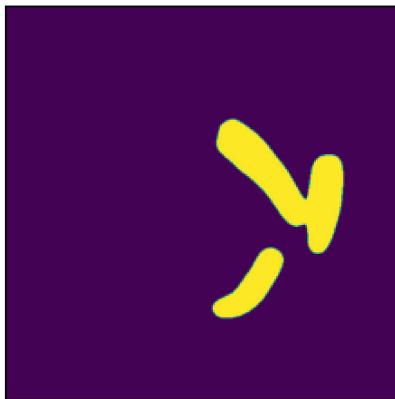
key: fondo



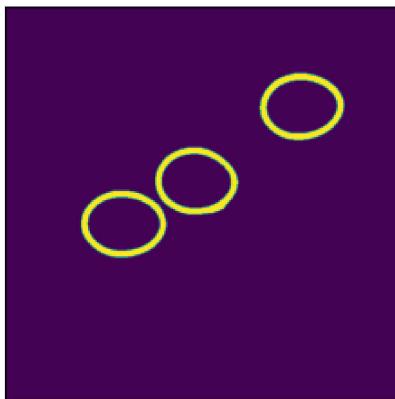
key: platano



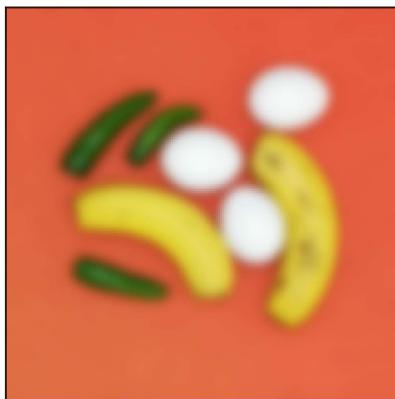
key: huevo



key: chile

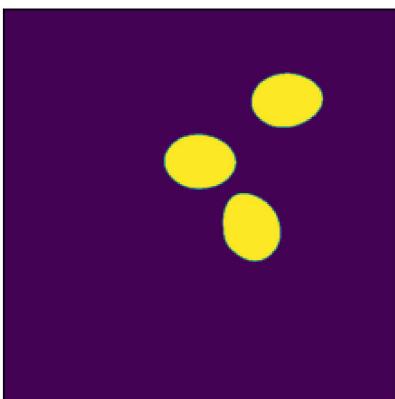


key: huevo

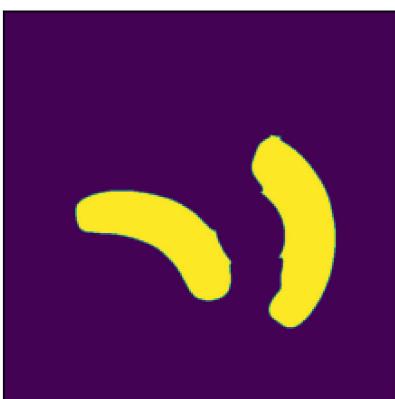




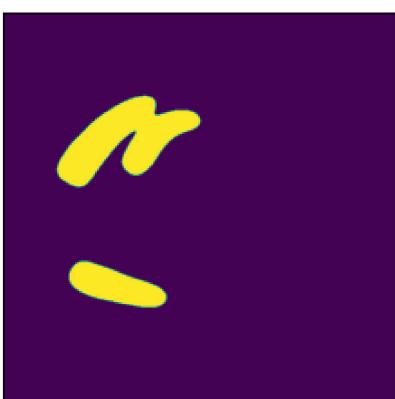
key: fondo



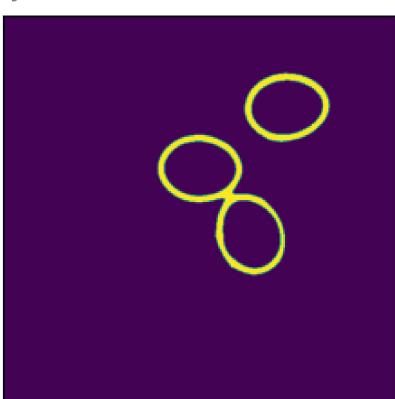
key: huevo



key: platano

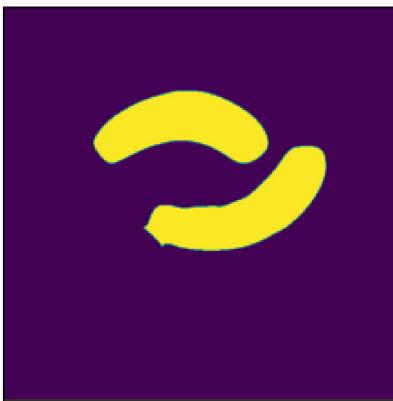


key: chile

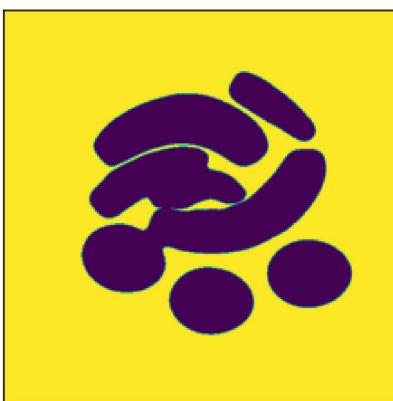


key: huevo

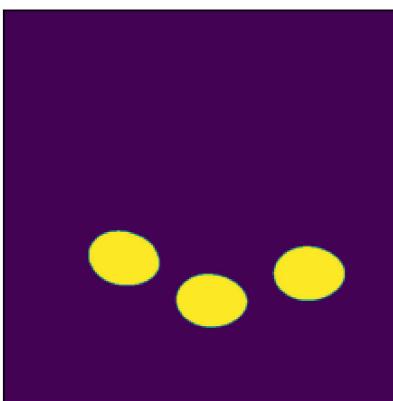




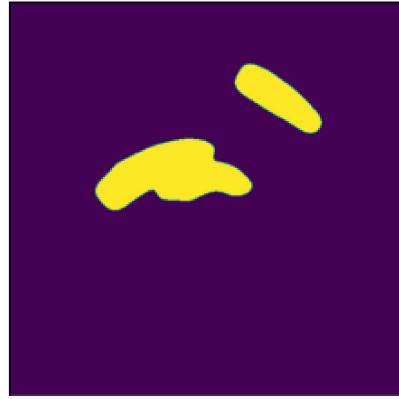
key: platano



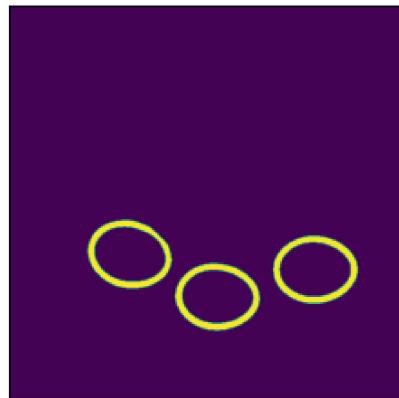
key: fondo



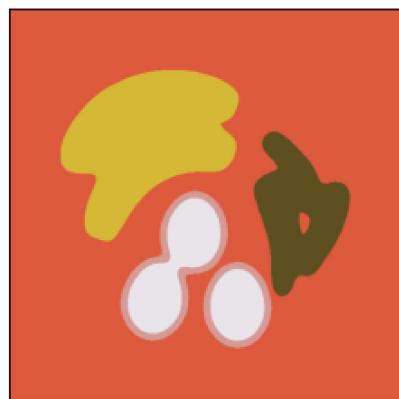
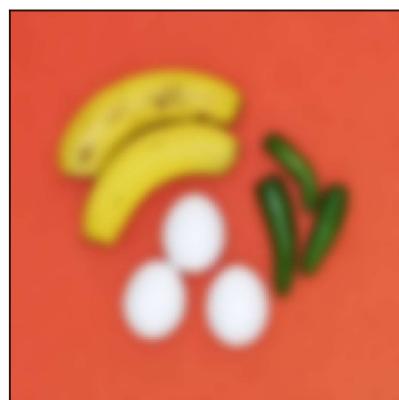
key: hueyo



key: chile



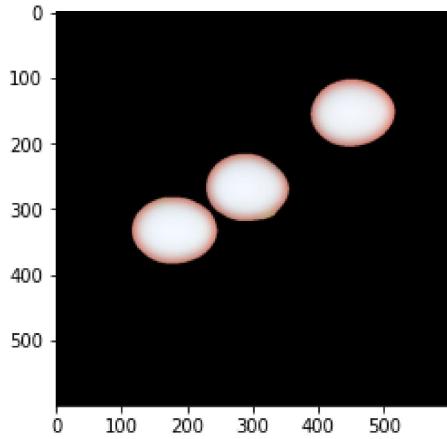
key: huevo



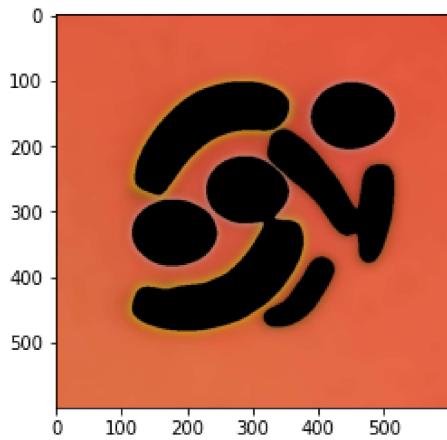
```
import matplotlib.image as pltim  
  
for i in range(len(comida.blur_train)):
```

```
print(comida_train[i])
for key in keys:
    mascara1 = diccionarios_masks[i][key]
    imagen = cv2.cvtColor(comida.blur_train[i], cv2.COLOR_BGR2RGB)
    #print(imagen.shape)
    mascara1.shape = (600,600,1)
    #print(mascara1.shape)
    mascara1 = mascara1//255
    res = imagen*mascara1
    res = res.astype('uint8')
    print(res.shape)
    pltim.imsave("/content/drive/MyDrive/Aux/ComidaMascara/"
        +comida_train[i].split("/")[-1][:-4]+"_"+key+".jpg",
        res)
plt.imshow(res)
plt.show()
```

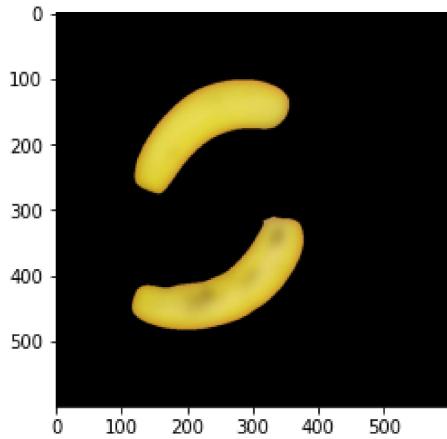
/content/drive/MyDrive/Aux/Comida/Entrenamiento2.jpg
(600, 600, 3)



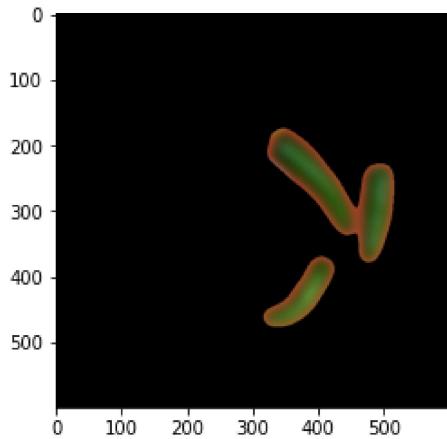
(600, 600, 3)



(600, 600, 3)

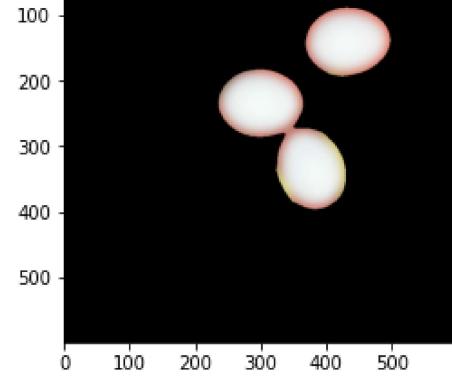


(600, 600, 3)

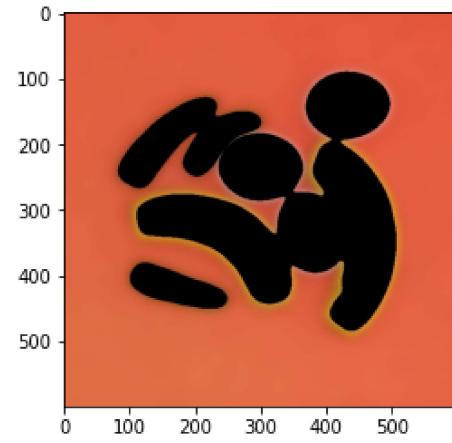


/content/drive/MyDrive/Aux/Comida/Entrenamiento3.jpg
(600, 600, 3)

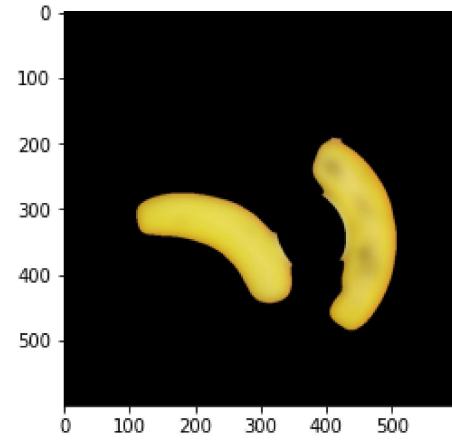




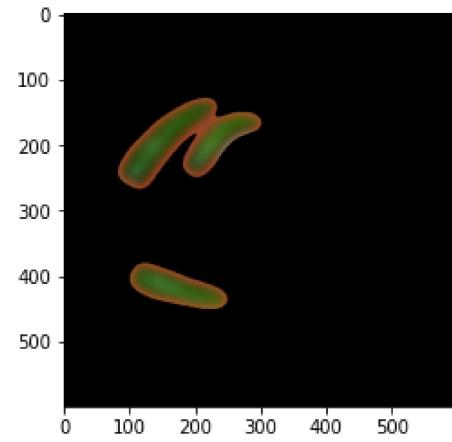
(600, 600, 3)



(600, 600, 3)

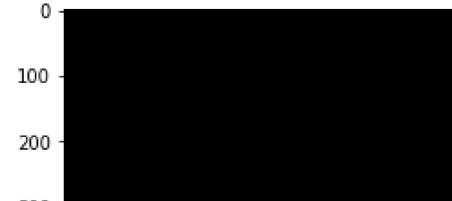


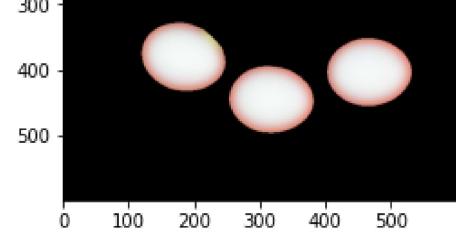
(600, 600, 3)



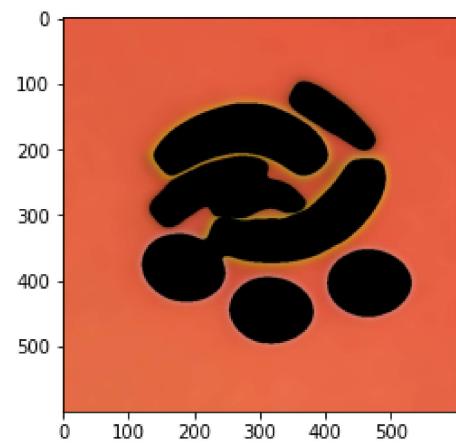
/content/drive/MyDrive/Aux/Comida/Entrenamiento4.jpg

(600, 600, 3)

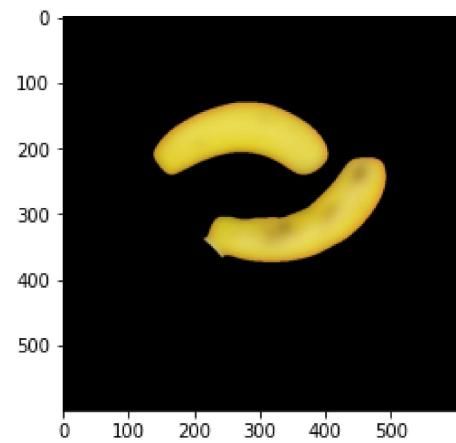




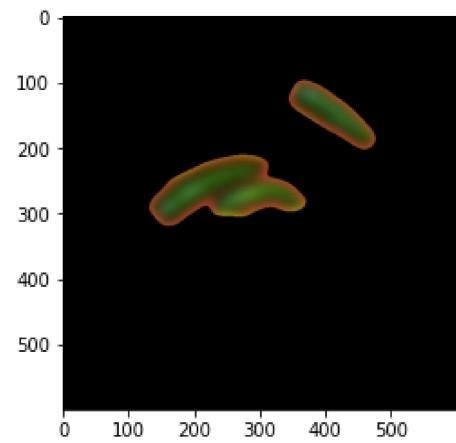
(600, 600, 3)



(600, 600, 3)

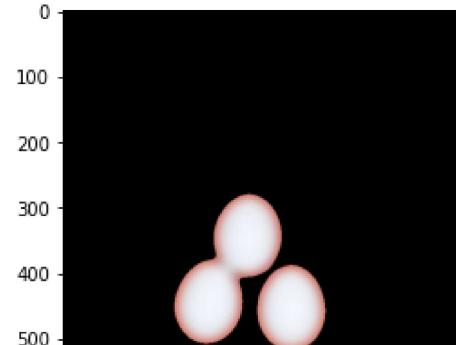


(600, 600, 3)



/content/drive/MyDrive/Aux/Comida/Entrenamiento1.jpg

(600, 600, 3)



A mano

Segmentamos con la siguiente herramienta

4 Generar Clasificador

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

```
def aplicar_filtro(img, intensidad_x, intensidad_y=None):
    '''Aplica cierta intensidad de filtro gaussiano

    return imagen con el filtro gaussiano aplicado
    '''
    if intensidad_y==None: intensidad_y = intensidad_x
    if intensidad_x%2!=1: intensidad_x += 1
    if intensidad_y%2!=1: intensidad_y += 1

    post = cv2.GaussianBlur(img,(intensidad_x,intensidad_y),0)
    return post

class Bayes():
    def __init__(self,dict_urls,clases):

        self.clases = list(clases)
        self.dict_urls = dict_urls

        self.imagenes = self.cargar_imagenes()
        print()
        self.prob_priori = self.calc_probs_priori()
        print()
        self.med_clases = self.calc_medias()
        print()
        self.cov_clases = self.calc_cov()
        print()
        self.dic_determinantes, self.dic_inversas = self.calc_inversa_det()

    def cargar_imagenes(self):
        '''Lee las imágenes y las guarda en un diccionario por clase

        return diccionario de imágenes por clase
        '''
        print("====")
        print("          CARGANDO IMAGENES")
        print("====")

        # se crea un diccionario nuevo con las clases indicadas
        dict_imagenes = {k:list() for k in self.clases}
        # para cada clase
        for clase in self.clases:
            # para cada imagen indicada
            for url in self.dict_urls[clase]:
                # se lee la imagen, se trasforma a RGB y se agrega a la lista de
                # imágenes de la clase
                img = cv2.imread(url)
```

```

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    dict_imagenes[clase].append(img)
    print("\tImagenes leidas para la clase "+clase+":"+len(dict_imagenes[clase]))

print("\n\tImagenes leídas en total:",sum([len(dict_imagenes[x]) for x in self.clases]))
return dict_imagenes

def calc_probs_priori(self):
    '''Calcula las probabilidades a priori de las clases'''

    print("====")
    print("      CALCULANDO PROBABILIDADES A PRIORI")
    print("====")

# Diccionario para guardar las probabilidades por clase
dic_probs = dict()

# por cada clase
for clase in self.clases:

    # Creamos contadores para determinar el número de pixeles total de todas
    # las imágenes y la cantidad de pixeles que pertenecen a la clase
    # de todas las imágenes
    cant_pixeles_clase = 0
    cant_pixeles_total = 0

    # para cada imagen de la clase
    for imagen in self.imagenes[clase]:
        # Sumamos en el eje 3, así sumamos la componente RGB de cada pixel
        suma_rgb = np.sum(imagen, axis=2)
        # Filtramos, solo contamos los pixeles cuya suma RGB sea mayor a 0
        cantidad = np.sum(suma_rgb != 0)
        # agregamos las cantidades de número de pixeles de la clase y
        # numero de pixeles totales
        cant_pixeles_clase += cantidad
        cant_pixeles_total += imagen.shape[0]*imagen.shape[1]

    # Calculamos la probabilidad de la clase con respecto a todos los pixeles
    # de todas las imágenes
    dic_probs[clase] = cant_pixeles_clase/cant_pixeles_total
    print("\n\tProbabilidad a priori clase "+clase+":"+cant_pixeles_clase/cant_pixeles_total)
    print("\tPixel de la clase en todas las imágenes:",cant_pixeles_clase)
    print("\tPixel totales en las imágenes:",cant_pixeles_total)

    print("\n\tResumen probabilidades:")
    print("\t\tclase\t\tprobabilidad",sep="")
    for k,v in dic_probs.items():
        print("\t\t",k," \t\t",round(v*100,4),"%",sep="")

return dic_probs

def cal_media_imagen(self, array):
    '''Genera la suma y cuenta los pixeles diferentes a negro de una
    imagen

Parámetros
array: imagen RGB

return suma de todos los pixeles en sus componentes RGB y cantidad de
pixeles diferentes al negro'''
```

```

# Se suma en el eje 0 y 1 para tener un vector que represente la suma de
# todos los R, todos los G y todos los B
suma = np.sum(array,axis=(0,1))
# Se suma en el eje 2, para tener sumas R+G+B y contar cuantos son
# diferentes a 0 (negro)
suma_rgb = np.sum(array,axis=2)
cantidad = np.sum(suma_rgb != 0)

return suma, cantidad

def cal_media_clase(self, lista_imagenes):
    '''Calcula la media por clase

Parámetro
lista_imagenes: Una lista de imágenes que pertenecen a una clase K

return media de la lista de imágenes'''

# Se genera un vector de 3 espacios
media = np.zeros(3)
cantidad_pixeles = 0

# para cada imagen
for imagen in lista_imagenes:
    # se calcula la suma de RGB para la imagen y la cantidad de pixeles
    # de la clase
    media_aux, cant_aux = self.cal_media_imagen(imagen)
    media += media_aux
    cantidad_pixeles += cant_aux

print("\t\tCantidad de pixeles pertenecientes a la clase:",cantidad_pixeles)
print("\t\tSuma de R's, G's y B's de todas las imágenes:\n\t\t",media)
print("\t\tMedia de la clase:\n\t\t",media/cantidad_pixeles)
return media/cantidad_pixeles

def calc_medias(self):
    '''Calcula la media de todas las clase

return diccionario de medias por clase'''
print("====")
print("      CALCULANDO MEDIA DE LAS CLASES")
print("====")

# diccionario nuevo
dict_media = dict()

# para cada clase
for clase in self.clases:
    print("\n\tMedia de la clase "+clase)
    dict_media[clase] = self.cal_media_clase(self.imagenes[clase])

return dict_media

def covarianza_imagen(self, imagen, media_p):
    '''Determina la matriz de covarianza de una imagen

Parámetros
imagen: imagen de la que se calculará la covarianza
media_p: media de la clase a la que pertenece la imagen

```

```

return matriz de covarianza acumulada y el número de pixeles que se
pertenece a la clase en la imagen
...

# cambia la forma de la media para que salga una matriz
media = media_p.reshape((-1,1))
# inicializamos variables auxiliares
w,h,p = imagen.shape
cov = np.zeros((3,3))

# para cada fila y columna
for i in range(h):
    for j in range(w):
        # si el pixel no es negro
        if np.sum(imagen[i,j,:]) != 0:
            # se redimensiona el pixel
            pixel = imagen[i,j,:].reshape((-1,1))
            # se suma el valor del pixel menos la media por la transpuesta de la
            # misma resta
            cov = cov + (pixel - media)@(pixel - media).T

# Se determina la cantidad de pixeles diferentes de negro
suma_rgb = np.sum(imagen, axis=2)
cantidad = np.sum(suma_rgb != 0)

return cov, cantidad

def covarianza_clase(self, lista_imagenes, media):
    '''Calcula la matriz de covarianza para una clase

    Parametros
    lista_imagenes: Lista de imágenes que pertenecen a las clase

    return covarianza de la clase'''

    # Inicia la covarianza de la clase como una matriz de 3x3
    cov_clase = np.zeros((3,3))
    # Inicializa la cantidad de pixeles que pertenecen a la clase
    cantidad_clase = 0
    # Transforma todas la imagenes para que sean tratadas como flotantes
    imagenes_float = [img.astype("float64") for img in lista_imagenes]

    # para cada imagen con números flotantes
    for imagen in imagenes_float:
        # se calcula la covarianza de la imagen y se calcula la cantidad de
        # pixeles que representan a la clase en la imagen
        cov_aux, cant_aux = self.covarianza_imagen(imagen, media)
        # Se suman los valores de la covarianza y el de la cantidad de pixeles
        cov_clase += cov_aux
        cantidad_clase += cant_aux

    # Se determina la matriz de covarianza dividiendo toda la suma entre la
    # cantidad de pixeles
    cov_clase_final = cov_clase/cantidad_clase
    print("\t\tMatriz acumulada de covarianzas:\n", cov_clase)
    print("\t\tCantidad de pixeles pertenecientes a la clase:", cantidad_clase)
    print("\t\tCovarianza de la clase:\n", cov_clase_final)
    return cov_clase_final

def calc_cov(self):
    ''' Proxy para calcular las covarianzas

```

```

return diccionario de covarianzas''''
print("=====")
print("    CALCULANDO COVARIANZA DE LAS CLASES")
print("=====")
# Diccionario nuevo
dict_cov = dict()

# para cada clase, se calcula la covarianza y la media, se agregan a los
# diccionarios
for clase in self.clases:
    print("\n\tCovarianza de la clase "+clase)
    dict_cov[clase] = self.covarianza_clase(self.imagenes[clase],
                                              self.med_clases[clase])

return dict_cov

def calc_inversa_det(self):
    '''Calcula la inversa de las matrices de covarianza y el determinante de
estas'''
    # genera matrices de covarianzas inversas
    print("=====")
    print("    CALCULANDO INVERSA DE COVARIANZAS")
    print("=====")
    dic_determinantes = {clase:np.linalg.det(self.cov_clases[clase]) for clase in self.clases}
    print("=====")
    print("    CALCULANDO DETERMINANTE DE COVARIANZAS")
    print("=====")
    dic_inversas = {clase:np.linalg.inv(self.cov_clases[clase]) for clase in self.clases}

    return dic_determinantes, dic_inversas

def predecir_pixel(self, pixel_p):

    X = pixel_p.reshape((-1,1))

    dict_probs = {k:0 for k in self.clases}

    for clase in self.clases:

        media = self.med_clases[clase].reshape((-1,1))

        resta = X - media

        S_1 = self.dic_inversas[clase]

        P = self.prob_priori[clase]*100

        ln_S = np.log(self.dic_determinantes[clase])

        Y = ( (-1/2) * (resta.T @ S_1 @ resta) ) - ( (1/2) * ln_S ) + np.log(P)

        dict_probs[clase] = Y.flatten()

    orden = sorted(dict_probs.items(), key=lambda x: x[1], reverse=True)
    mayor = orden[0]
    #print(orden)
    #print(mayor[0])

    return mayor[0]

```

```

def predecir(self, imagen):

    img = imagen.astype("float64")

    # Se calculan las escalas de grises
    if len(self.clases)>1:
        escala = 255/(len(self.clases)-1)
        escalas = {self.clases[i]:i*escala for i in range(len(self.clases))}

    else:
        escala = 255
        escalas = {self.clases[0]:escala}

    w,h,p = img.shape
    print(escalas)
    resultado = list()
    for i in range(h):
        fila = list()
        for j in range(w):
            fila.append(escalas[self.predecir_pixel(img[i,j,:])])
        conteo = Counter(fila)
        #print(conteo)

        resultado.append(fila)

    return resultado

```

▼ 5 Desplegar calculos

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.

Se crea un objeto de tipo Bayes y se muestran los calculos

▼ Kmeans

```

diccionario = {
    "platano": [
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_platano.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_platano.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_platano.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_platano.jpg"
    ],
    "chile": [
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_chile.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_chile.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_chile.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_chile.jpg"
    ],
    "fondo": [
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_fondo.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_fondo.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_fondo.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_fondo.jpg"
    ]
}

```

```
[  
    "huevo": [  
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_huevo.jpg",  
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_huevo.jpg",  
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_huevo.jpg",  
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_huevo.jpg",  
    ]  
}  
modelo = Bayes(clases=diccionario.keys(),dict_urls=diccionario)
```

```
=====  
          CARGANDO IMAGENES  
=====  
Imagenes leidas para la clase platano: 4  
Imagenes leidas para la clase chile: 4  
Imagenes leidas para la clase fondo: 4  
Imagenes leidas para la clase huevo: 4  
  
Imagenes leídas en total: 16  
=====  
          CALCULANDO PROBABILIDADES A PRIORI  
=====  
  
Probabilidad a priori clase platano: 0.1461944444444443  
Pixelés de la clase en todas las imágenes: 210520  
Pixelés totales en las imágenes: 1440000  
  
Probabilidad a priori clase chile: 0.08455416666666667  
Pixelés de la clase en todas las imágenes: 121758  
Pixelés totales en las imágenes: 1440000  
  
Probabilidad a priori clase fondo: 0.8238534722222223  
Pixelés de la clase en todas las imágenes: 1186349  
Pixelés totales en las imágenes: 1440000  
  
Probabilidad a priori clase huevo: 0.1132118055555556  
Pixelés de la clase en todas las imágenes: 163025  
Pixelés totales en las imágenes: 1440000  
  
Resumen probabilidades:  
clase      probabilidad  
platano    14.6194%  
chile      8.4554%  
fondo      82.3853%  
huevo      11.3212%  
=====  
          CALCULANDO MEDIA DE LAS CLASES  
=====  
  
Media de la clase platano  
    Cantidad de pixelés pertenecientes a la clase: 210520  
    Suma de R's, G's y B's de todas las imágenes:  
    [35194646. 31027721. 10184204.]  
    Media de la clase:  
    [167.17958389 147.38609633 48.37642029]  
  
Media de la clase chile  
    Cantidad de pixelés pertenecientes a la clase: 121758  
    Suma de R's, G's y B's de todas las imágenes:  
    [8024883. 6976827. 2712292.]  
    Media de la clase:  
    [65.90846597 57.30076874 22.27608863]  
  
Media de la clase fondo  
    Cantidad de pixelés pertenecientes a la clase: 1186349
```

▼ A mano

```
diccionario = {
    "platano": [
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 1/Platano.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 2/Platano.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 3/Platano.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 4/Platano.jpg"
    ],
    "chile": [
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 1/Chiles.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 2/Chiles.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 3/Chiles.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 4/Chiles.jpg",
    ],
    "fondo": [
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 1/Fondo.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 2/Fondo.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 3/Fondo.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 4/Fondo.jpg",
    ],
    "huevo": [
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 1/Huevos.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 2/Huevos.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 3/Huevos.jpg",
        "/content/drive/MyDrive/Aux/ManualComida/Entrenamiento 4/Huevos.jpg",
    ]
}
modelo_mano = Bayes(clases=diccionario.keys(),dict_urls=diccionario)
```

```
=====
CARGANDO IMAGENES
=====
Imagenes leidas para la clase platano: 4
Imagenes leidas para la clase chile: 4
Imagenes leidas para la clase fondo: 4
Imagenes leidas para la clase huevo: 4

Imagenes leídas en total: 16

=====
CALCULANDO PROBABILIDADES A PRIORI
=====

Probabilidad a priori clase platano: 0.1506722222222224
Pixelles de la clase en todas las imagenes: 216968
Pixelles totales en las imágenes: 1440000

Probabilidad a priori clase chile: 0.06874236111111111
Pixelles de la clase en todas las imagenes: 98989
Pixelles totales en las imágenes: 1440000

Probabilidad a priori clase fondo: 0.8431805555555556
Pixelles de la clase en todas las imagenes: 1214180
Pixelles totales en las imágenes: 1440000

Probabilidad a priori clase huevo: 0.1005347222222222
Pixelles de la clase en todas las imagenes: 144770
Pixelles totales en las imágenes: 1440000
```

```

Resumen probabilidades:
clase           probabilidad
platano        15.0672%
chile          6.8742%
fondo          84.3181%
huevo          10.0535%


=====
CALCULANDO MEDIA DE LAS CLASES
=====

Media de la clase platano
    Cantidad de pixeles pertenecientes a la clase: 216968
    Suma de R's, G's y B's de todas las imágenes:
    [40400515. 34695029. 10929519.]
    Media de la clase:
    [186.20494727 159.90850725 50.37387541]

Media de la clase chile
    Cantidad de pixeles pertenecientes a la clase: 98989
    Suma de R's, G's y B's de todas las imágenes:
    [6269123. 6240431. 2076189.]
    Media de la clase:
    [63.33151158 63.04166119 20.9739365]

Media de la clase fondo
    Cantidad de pixeles pertenecientes a la clase: 1214180
    Suma de R's, G's y B's de todas las imágenes:
    [2 489783328+0j 1 08846075e+0j 7 30760600e+0j]

```

▼ Pruebas

▼ 6 Clasificación de imágenes

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).

Con el objeto tipo Bayes, ocupamos el método de "predecir" para generar la predicción de nuestro modelo.

```

list_url = [
    "/content/drive/MyDrive/Aux/Comida/Prueba1.jpg",
    "/content/drive/MyDrive/Aux/Comida/Prueba2.jpg",
    "/content/drive/MyDrive/Aux/Comida/Prueba3.jpg"
]
list_imagenes = list()
for url in list_url:
    img = cv2.imread(url)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = aplicar_filtro(img, 51)
    list_imagenes.append(img)

```

▼ Kmeans

```

list_resultados = list()
for img in list_imagenes:
    resultado = modelo.predecir(img)
    list_resultados.append(resultado)

```

```
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}  
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}  
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}
```

▼ A Mano

```
list_resultados_mano = list()  
for img in list_imagenes:  
    resultado = modelo_mano.predecir(img)  
    list_resultados_mano.append(resultado)  
  
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}  
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}  
{'platano': 0.0, 'chile': 85.0, 'fondo': 170.0, 'huevo': 255.0}
```

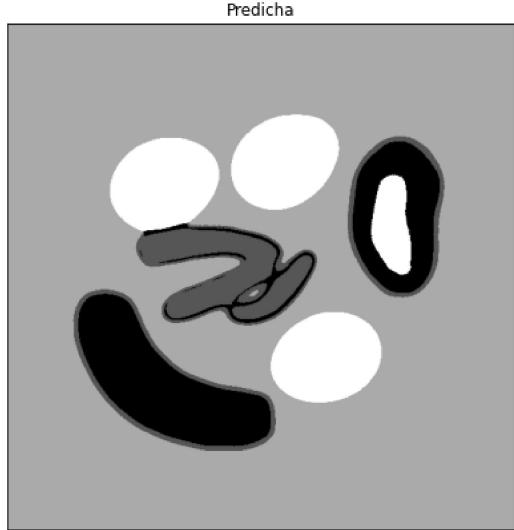
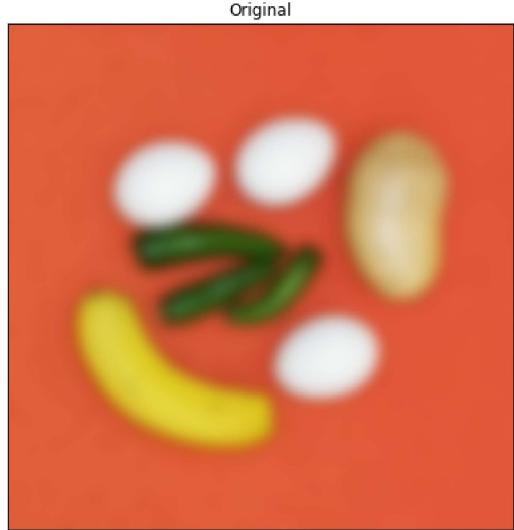
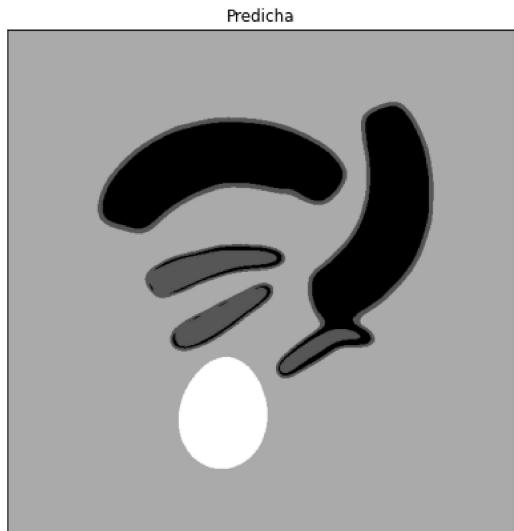
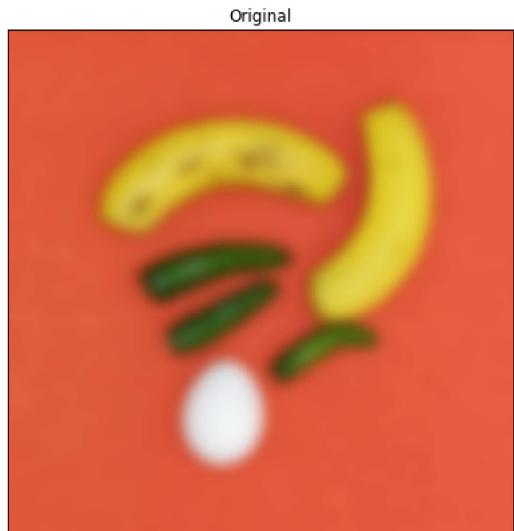
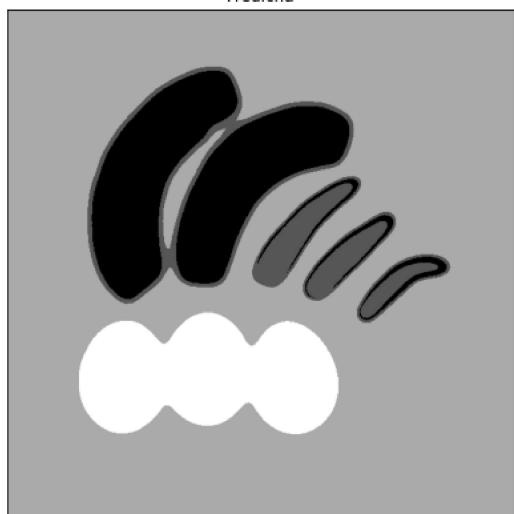
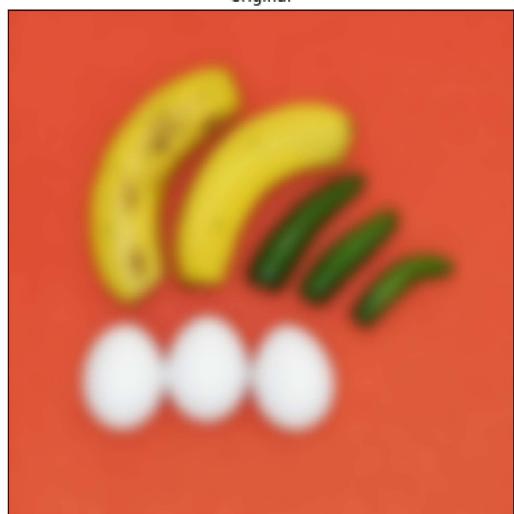
▼ 7 Mostrar resultados gráficos

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.

Grafica la imagen resultante

▼ Kmeans

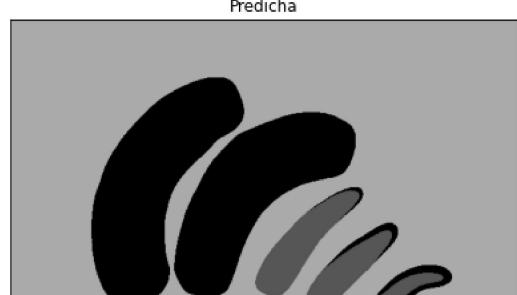
```
for resultado,img in zip(list_resultados,list_imagenes):  
    plt.figure(figsize=(15,15))  
    plt.subplot(121)  
    plt.imshow(img)  
    plt.title('Original')  
    plt.xticks([]), plt.yticks([])  
    plt.subplot(122)  
    plt.imshow(resultado,cmap='gray')  
    plt.title('Predicha')  
    plt.xticks([]), plt.yticks([])  
    plt.show()
```



▼ A mano

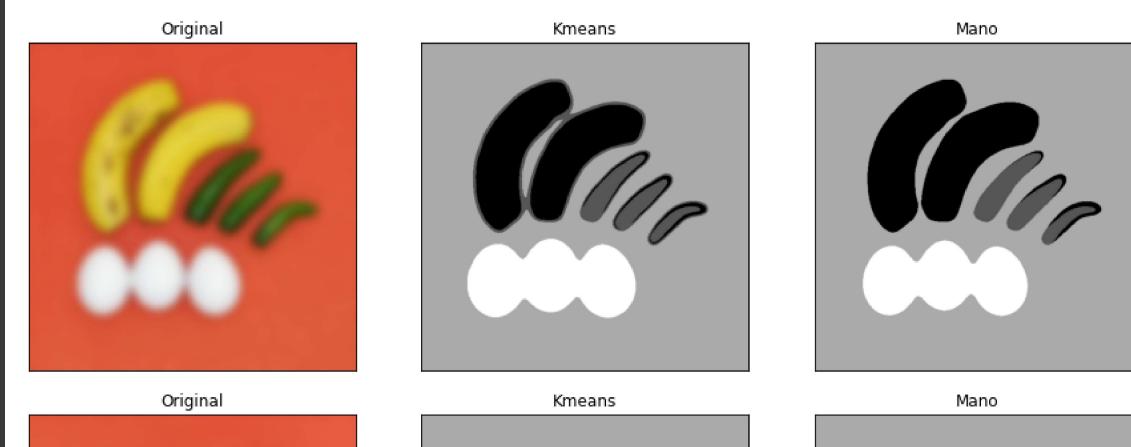
```
for resultado,img in zip(list_resultados_mano,list_imagenes):
    plt.figure(figsize=(15,15))
    plt.subplot(121)
    plt.imshow(img)
    plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122)
    plt.imshow(resultado,cmap='gray')
    plt.title('Predicha')
```

```
plt.xticks([]), plt.yticks([])
plt.show()
```



▼ Comparación

```
for resultado_m,resultado_k,img in zip(list_resultados_mano,list_resultados,list_imagenes):
    plt.figure(figsize=(15,15))
    plt.subplot(131)
    plt.imshow(img)
    plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(132)
    plt.imshow(resultado_k,cmap='gray')
    plt.title('Kmeans')
    plt.xticks([]), plt.yticks([])
    plt.subplot(133)
    plt.imshow(resultado_m,cmap='gray')
    plt.title('Mano')
    plt.xticks([]), plt.yticks([])
    plt.show()
```



▼ 8 Comparación con scikit-learn

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.

Grafica la imagen resultante

```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
from sklearn.naive_bayes import MultinomialNB

def cargar_imagenes_clases(dict_url_train):
    dict_imagenes_train = dict()

    for clase, lista_urls in dict_url_train.items():
        lista_imagenes = list()

        for url in lista_urls:
            img = cv2.imread(url)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = aplicar_filtro(img, 51)
            lista_imagenes.append(img)

        dict_imagenes_train[clase] = lista_imagenes

    return dict_imagenes_train

def arreglo_pixeles(img):
    lista_pixeles = list()

    w,h,p = img.shape

    for i in range(w):
        for j in range(h):
            lista_pixeles.append(img[i,j,:])

    return np.array(lista_pixeles)

def filtrar_negros(lista_pixeles):
    pixeles_filtrados = list()

    for pixel in lista_pixeles:
```

```

if np.sum(pixel) != 0:
    pixeles_filtrados.append(pixel)

return np.array(pixeles_filtrados)

def generar_pixeles_clases(dict_imagenes_train):
    pixeles_clases = dict()

    for clase, lista_imagenes in dict_imagenes_train.items():
        bandera = True
        pixeles_imagen = None

        for imagen in lista_imagenes:
            pixeles_planos = arreglo_pixeles(imagen)
            pixeles_filtrados = filtrar_negros(pixeles_planos)

            if bandera:
                pixeles_imagen = pixeles_filtrados
                bandera = False
            else:
                pixeles_imagen = np.concatenate((pixeles_imagen, pixeles_filtrados))

        pixeles_clases[clase] = pixeles_imagen

    return pixeles_clases

def etiquetar_pixeles(dict_imagenes_filtrados):
    lista_pixeles_etiquetados = list()
    diccionario_clases = dict()
    i = 0

    for clase, lista_pixeles in dict_imagenes_filtrados.items():
        diccionario_clases[i] = 85*i
        for j in range(lista_pixeles.shape[0]):
            lista_pixeles_etiquetados.append((i,lista_pixeles[j,:]))

        i+=1

    return lista_pixeles_etiquetados, diccionario_clases

dict_url_train = {
    "platano": [
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_platano.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_platano.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_platano.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_platano.jpg"
    ],
    "chile": [
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_chile.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_chile.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_chile.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_chile.jpg"
    ],
    "fondo": [
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_fondo.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_fondo.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_fondo.jpg",
        "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_fondo.jpg"
    ]
}

```

```
],
"huevo": [
    "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento1_huevo.jpg",
    "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento2_huevo.jpg",
    "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento3_huevo.jpg",
    "/content/drive/MyDrive/Aux/ComidaMascara/Entrenamiento4_huevo.jpg",
]
}
```

```
dict_imagenes_train = cargar_imagenes_clases(dict_url_train)
dict_imagenes_filtrados = generar_pixeles_clases(dict_imagenes_train)
pixeles_etiquetados, diccionario_clases = etiquetar_pixeles(dict_imagenes_filtrados)
```

```
np.random.shuffle(pixeles_etiquetados)
```

```
list_url = [
    "/content/drive/MyDrive/Aux/Comida/Prueba1.jpg",
    "/content/drive/MyDrive/Aux/Comida/Prueba2.jpg",
    "/content/drive/MyDrive/Aux/Comida/Prueba3.jpg"
]
list_imagenes = list()
for url in list_url:
    img = cv2.imread(url)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = aplicar_filtro(img, 51)
    list_imagenes.append(img)
```

```
from sklearn.naive_bayes import MultinomialNB

X = np.array([x[1] for x in pixeles_etiquetados])
y = np.array([x[0] for x in pixeles_etiquetados])

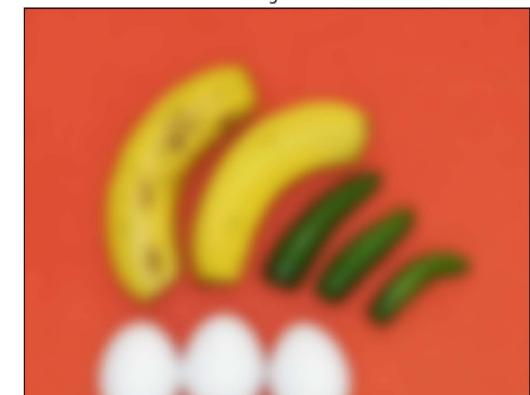
clf = MultinomialNB()
clf.fit(X, y)
print(clf.predict(X[2:3])) # [2]
```

```
[2]
```

```
imagenes_predecidas = list()
for imagen in list_imagenes:
    w,h,p = imagen.shape
    new_imagen = list()
    for row in range(w):
        new_row = list()
        for column in range(h):
            pixel = np.array([imagen[row,column,:]])
            valor = clf.predict(pixel)
            mapeo = diccionario_clases[valor[0]]
            new_row.append(mapeo)
        new_imagen.append(new_row)
    imagenes_predecidas.append([imagen,new_imagen])
```

```
for par in imagenes_predecidas:
    resultado, img = par[1], par[0]
    plt.figure(figsize=(15,15))
    plt.subplot(121)
    plt.imshow(img)
```

```
plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122)
plt.imshow(resultado,cmap='gray')
plt.title('Predicha')
plt.xticks([]), plt.yticks([])
plt.show()
```



```
for resultado_m,resultado_k,img,par in zip(list_resultados_mano,list_resultados,list_imagenes,imagenes_predecidas):
    scik = par[1]
    plt.figure(figsize=(20,20))
    plt.subplot(141)
    plt.imshow(img)
    plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(142)
    plt.imshow(resultado_k,cmap='gray')
    plt.title('Kmeans')
    plt.xticks([]), plt.yticks([])
    plt.subplot(143)
    plt.imshow(resultado_m,cmap='gray')
    plt.title('Mano')
    plt.xticks([]), plt.yticks([])
    plt.subplot(144)
    plt.imshow(scik,cmap='gray')
    plt.title('Scikit')
    plt.xticks([]), plt.yticks([])
    plt.show()
```

