Práctica Clasificador Bayesiano

Julio Pérez
Estudiante, Facultad de Ingeniería
Ciudad Universitaria, Ciudad de México, México

Mauricio Bautista
Estudiante, Facultad de Ingeniería
Ciudad Universitaria Ciudad de México, México

***Resumen*—Por medio de datos de entrenamiento generaremos un clasificador de bayes con el que generaremos un análisis de los datos y luego los compararemos según sus valores estadísticos.**

Palabras clave—Bayes, valores estadísticos, covarianza

I. Objetivo

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

II. Introducción

En esta práctica usaremos el teorema de bayes, usando loa valores estadísticos de la media y la covarianza de los datos podremos hacer predicciones de patrones en imágenes al ver las relaciones en sus canales RGB podremos ver según sus clases que tan probable es encontrarnos dicho objeto y con qué frecuencia lo esperaremos.

III. Entrenamiento

La parte de entrenamiento es lo que hicimos de tarea, pero hicimos un nuevo modelo con las bibliotecas opencv para las imágenes y el crear máscaras, numpy para los cálculos de las matrices y scipy para poder hacer los cálculos de los valores estadísticos.

```python
import cv2
import numpy as np
from scipy.stats import multivariate_normal
```

Fig.1 bibliotecas de la primera parte

Lo primero que haremos será cargar la imagen y leerla en bgr para el momento de crear las máscaras además de una copia para evitar trabajar con la original y perder datos, además de las variables donde guardaremos los datos para los cálculos.

```python
# Cargar imagen
image_path = 'Entrenamiento1.jpg'
original_img = cv2.imread(image_path)
original_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
display_img = np.copy(original_img)  # Imagen para mostrar en la interfaz

# Inicializar variables
results = []  # Almacenará (media, covarianza) para cada clase
priors = []  # Probabilidades a priori de cada clase
mask = np.zeros(original_img.shape[:2], dtype=np.uint8)
drawing = False
```

Fig.2 parte inicial del código para leer y guardar datos.

Definiremos una función para poder dibujar sobre la imagen que vamos a manipular con interrupciones del mouse y poder crear las máscaras lo más parecidas a las clases de nuestros objetos.

```python
def draw_mask(event, x, y, flags, param):
    global drawing, mask, display_img
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
    elif event == cv2.EVENT_MOUSEMOVE and drawing:
        cv2.circle(mask, (x, y), 5, (255), -1)  # Dibuja en la máscara
        cv2.circle(display_img, (x, y), 10, (0, 255, 0), -1)  # Dibuja en verde en la imagen de
visualización
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
```

Fig.3 función para crear las máscaras a mano.

Además, definiremos una función para generar nuevas mascaras para las nuevas clases por lo que pondrán a 0 la mascara y creara otra copia de la imagen.

```python
def reset_mask():
    global mask, display_img
    mask = np.zeros(original_img.shape[:2], dtype=np.uint8)
    display_img = np.copy(original_img)  # Restaura la imagen de visualización
```

Fig.4 función para restablecer la imagen para trabajar.

Finalmente, para obtener los valores lo aremos con un ciclo while infinito que terminara al dar esc y que esperara a dar enter para obtener los valores de la media y la varianza con numpy a la zona recortada y después sacara el valor a priori en relación a los pixeles en la zona recortada con el tamaño de la imagen y dará los datos en pantalla para repetir el ciclo con una nueva mascara.

```python
# Configuración de OpenCV
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_mask)

while True:
    cv2.imshow('image', display_img)
    key = cv2.waitKey(1) & 0xFF
    if key == 27:  # Esc para salir y clasificar
        cv2.destroyAllWindows()  # Asegurarse de cerrar todas las ventanas antes de clasificar
        classify_pixels()
        break
    elif key == 13:  # Enter para procesar la selección
        selected_region = original_img[mask == 255]
        if selected_region.size > 0:
            mean_colors = np.mean(selected_region.reshape(-1, 3), axis=0)
            cov_colors = np.cov(selected_region.reshape(-1, 3), rowvar=False)
            results.append((mean_colors, cov_colors))
            priors.append(np.sum(mask) / mask.size)  # A priori basado en el área seleccionada
            print(f"Clase {len(results)} registrada. Media: {mean_colors}, Covarianza: {cov_colors}")
        reset_mask()  # Preparar para nueva clase
```

Fig.5 ciclo principal para la obtención de los datos.

IV. PRUEBA 6.

Para el primer ejercicio de prueba pide que con los datos anteriores media, covarianza y valor priori de las clases clasifiquemos los pixeles, por lo que haremos una función que primero verifica si hay clases y si hay genera un modelo gaussiano normal con scipy y genera una imagen vacía y ya con los datos en dos for anidados evalúa de cada pixel de la imagen original con la funcion de densidad de probabilidad y los valores a priori dentó del modelo gaussiano cual clase es más probable que sea ese pixel y según eso lo pone de un color u otro en la imagen vacía y cuando termina le da el formato a la imagen ahora llena y la imprime en pantalla.

```python
def classify_pixels():
    if not results:
        print("No hay clases registradas para clasificar.")
        return
    gaussian_models = [multivariate_normal(mean=mean, cov=cov) for mean, cov in results]
    classified_image = np.zeros((original_img.shape[0], original_img.shape[1]), dtype=int)

    for i in range(original_img.shape[0]):
        for j in range(original_img.shape[1]):
            pixel = original_img[i, j]
            posteriors = [model.pdf(pixel) * prior for model, prior in zip(gaussian_models, priors)]
            classified_image[i, j] = np.argmax(posteriors)

    classified_image = np.uint8(classified_image * 60)
    classified_image = np.clip(classified_image, 0, 255)

    cv2.imshow('Classified Image', classified_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Fig.6 función para clasificar pixeles.

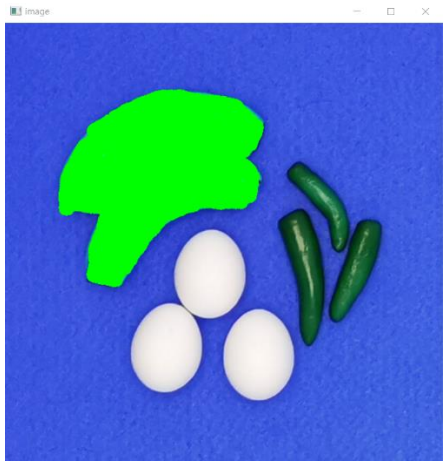

Fig.7 clase 1.

Fig.8 clase 2.

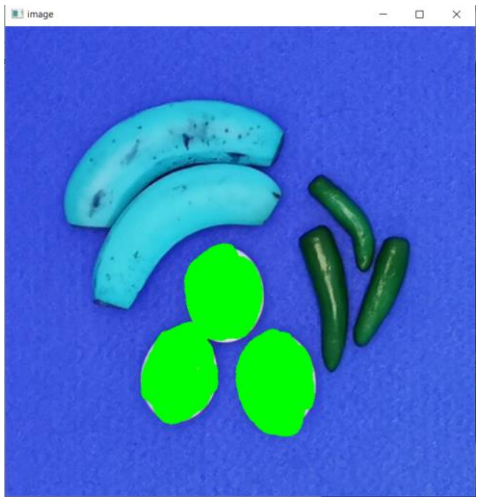

Fig.9 clase 3.

```
D:\semestre 10\patrones\practica_3>python prueba.py
Clase 1 registrada. Media: [212.73519461 189.19919979  54.04245241], Covarianza: [[493.92690458 614.82383046 286.57187744]
 [614.82383046 842.39106261 371.43785098]
 [286.57187744 371.43785098 415.13938632]]
Clase 2 registrada. Media: [56.12894177 89.38305281 27.91777907], Covarianza: [[ 818.36713946  541.42250078  660.16306477]
 [ 541.42250078 1064.16630655  853.63641239]
 [ 660.16306477  853.63641239  937.87250771]]
Clase 3 registrada. Media: [230.61272827 227.67074688 231.86221824], Covarianza: [[113.80219268 215.63183832 235.33606911]
 [215.63183832 676.49912082 741.13436993]
 [235.33606911  741.13436993 819.31396707]]
```

Fig.10 valores estadísticos.

Fig.11 pixeles clasificados según las clases.

V. PRUEBA 7

Para este ejercicio es lo mismo que en el anterior, pero con valores asignados a la imagen para ver las diferencias entonces los agregaremos sobre la matriz de resultados para ver la diferencia.

```
# Inicializar variables
results = []  # Almacenará (media, covarianza) para cada clase
priors = []  # Probabilidades a priori de cada clase
mask = np.zeros(original_img.shape[:2], dtype=np.uint8)
drawing = False

# Asignación de valores de gris para cada clase
class_values = [0, 128, 250]  # Asegúrate de tener tantos valores como clase
```

Fig.11 vector de modificación al inicio.

```
for i in range(original_img.shape[0]):
        for j in range(original_img.shape[1]):
            pixel = original_img[i, j]
            posteriors = [model.pdf(pixel) * prior for model, prior in zip(gaussian_models,
priors)]    class_index = np.argmax(posteriors)
            classified_image[i, j] = class_values[class_index]
```

Fig.12 Modificación aplicada al crear la imagen de clasificación.

```
D:\semestre 10\patrones\practica_3>python prueba2.py
Clase 1 registrada. Media: [213.96350541 190.52966881  55.1834373 ], Covarianza: [[396.86561404 503.90794494 261.8815063
2]
 [503.90794494 744.91448801 351.12608963]
 [261.88150632 351.12608963 423.98446197]]
Clase 2 registrada. Media: [56.69904692 99.73717009 33.77065005], Covarianza: [[ 704.79258861  758.64153121  794.7783344
8]
 [ 758.64153121  879.13474922  866.76019409]
 [ 794.77833448  866.76019409 1048.46981671]]
Clase 3 registrada. Media: [229.86640573 224.48278789 228.09167217], Covarianza: [[ 126.04312792  299.59409281  332.7821
1305]
 [ 299.59409281 1081.59656768 1209.61685736]
 [ 332.78211305 1209.61685736 1360.39178445]]
```

Fig.13 resultados de modificación.

Fig.14 pixeles clasificados de modificación.

Se ve que ahora los datos en la clasificación en lugar de tomar el fondo para los chiles ahora son un poco mas brilloso y lo considera para los huevos además de que algunos valores de la varianza son un poco más altos.

VI. PRUEBA 8

Para el ejercicio 8 nos pide hacer los cálculos ahora con el clasificador de bayes de la función de scikit learn y ver que tan parecidos son los resultados entre lo que hicimos y los de la función, por lo que reutilizamos el método para poder obtener las clases sin embargo los cálculos de la clasificación se harán con la función de bayes.

```
import cv2
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
```

Fig.15 bibliotecas de la prueba 8.

Para la nueva función para clasificar pixeles primero creamos el modelo gaussiano y luego se le dan los datos para entrenar, después se le da el formato a la imagen para que con los datos de predicción se le pueda dar una clasificación y se reajusta a la imagen para después darle un formato legible y poner la imagen en pantalla.

```
def classify_pixels(X_train, y_train):
    gnb = GaussianNB()
    gnb.fit(X_train, y_train)
    flat_image = img.reshape(-1, 3)
    predictions = gnb.predict(flat_image)
    classified_image = predictions.reshape(img.shape[:2])

    # Convertir classified_image a uint8 y escalar adecuadamente
    classified_image = np.uint8(classified_image)
    unique_labels = np.unique(classified_image)
    scale_factor = 255 / unique_labels.max() if unique_labels.max() != 0 else
1   classified_image = np.uint8(classified_image * scale_factor)

    cv2.imshow('Classified Image', classified_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Fig.16 función para clasificar los pixeles con scikit learn.

Para el flujo principal ahora en lugar de obtener los datos de para hacer los cálculos obtenemos los datos de entrenamiento de las imágenes para después clasificar las imágenes por lo que solo obtenemos los datos de los pixeles para cada clase particular.

```
X_train = []
y_train = []

while True:
    cv2.imshow('image', display_img)
    key = cv2.waitKey(1) & 0xFF
    if key == 27:  # Esc para salir y clasificar
        if len(X_train) > 0:
            classify_pixels(X_train, y_train)
        break
    elif key == 13:  # Enter para guardar la selección actual
        if np.any(current_mask):
            masks.append(current_mask)
            selected_pixels = img[current_mask == 255]
            X_train.extend(selected_pixels)
            y_train.extend([len(masks)] * len(selected_pixels))  # Usar el índice de la máscara como
etiqueta
            print(f"Clase {len(masks)} registrada con {len(selected_pixels)} píxeles.")
            reset_mask()  # Preparar para nueva clase
```

Fig.17 Cambio al loop principal para obtener los valores de entrenamiento de las máscaras.

```
D:\semestre 10\patrones\practica_3>python prueba3.py
Clase 1 registrada con 34970 píxeles.
Clase 2 registrada con 9551 píxeles.
Clase 3 registrada con 21546 píxeles.
```

Fig.18 registro de las clases.

Fig.19 resultado de la función scikit learn.

A pesar de que se ve más limpio con la función de scikit learn nuestros resultados no estuvieron tan lejos del como lo hace la función, aunque también es más difícil hacer qué distinga del fondo no encontramos manera efectiva de quitarlo puesto que tiene que evaluar toda la imagen, pero consideramos que los resultados son similares.

VII. CONCLUSIÓN

Consideramos que la practica nos ayudó a pensar un poco el cómo clasificar de la mejor manera posible ya que identificar los objetos y luego clasificarlos es un poco conflictivo ya que por cosas como el fondo que fue lo que no nos convenció de los resultados era muy estorboso, pero consideramos que el objetivo de la práctica que era evaluar y clasificar se cumplió al poder ver maneras de detectar los diferentes objetos y hasta cierto punto clasificarlos simplemente con lo que se ve.

VIII. REFERENCIAS

[1] "Numpy.Mean — NumPy v1.26 manual", Numpy.org. [En línea]. Disponible en: https://numpy.org/doc/stable/reference/generated/numpy.mean.html. [Consultado: 15-abr-2024].

[2] "Numpy.Cov — NumPy v1.26 manual", Numpy.org. [En línea]. Disponible en: https://numpy.org/doc/stable/reference/generated/numpy.cov.html. [Consultado: 15-abr-2024].

[3] "Scipy.Stats.Multivariate_normal — SciPy v1.13.0 manual", Scipy.org. [En línea]. Disponible en: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html. [Consultado: 15-abr-2024].

[4] "OpenCV: Drawing Functions in OpenCV", Opencv.org. [En línea]. Disponible en: https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html. [Consultado: 15-abr-2024].

[5] "Sklearn.Naive_bayes.GaussianNB", scikit-learn. [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. [Consultado: 15-abr-2024].