Práctica 2: Clasificador Bayesiano

Reconocimiento de Patrones - 0757

Facultad de Ingeniería
Universidad Nacional Autónoma de México

Ceballos Equihua C. N.
Ingeniería en Computación
Facultad de Ingeniería, UNAM
Ciudad de México, México
ceballos.equihua@gmail.com

Muñoz Marbán J.
Ingeniería en Mecatrónica
Facultad de Ingeniería, UNAM
Ciudad de México, México
juanmzmb@gmail.com

Murrieta Villegas A.
Ingeniería en Computación
Facultad de Ingeniería, UNAM
Ciudad de México, México
alfonsomvmx@gmail.com

Solano González F. J.
Ingeniería en Mecatrónica
Facultad de Ingeniería, UNAM
Ciudad de México, México
felipe.solano.gos@gmail.com

Profesores:
Dr. Boris Escalante Ramírez
Dra. Olveres Montiel Jimena
I.I.M.A.S. - UNAM

Resumen— **Esta práctica consiste en desarrollar un clasificador de Bayes para un número definido de regiones, utilizando imágenes de comida y de monos. Se obtuvieron los vectores de características utilizando los valores RGB de cada vector y los estadísticos muestrales para aplicar la fórmula de Bayes.**

I. Objetivo

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

II. Introducción

Naive Bayes es una técnica de clasificación estadística basada en el Teorema de Bayes. Es uno de los algoritmos de aprendizaje supervisado más simples. El clasificador Naive Bayes es el algoritmo rápido, preciso y confiable. Los clasificadores Naive Bayes tienen alta precisión y velocidad en grandes conjuntos de datos.

El clasificador Naive Bayes asume que el efecto de una característica particular en una clase es independiente de otras características. Por ejemplo, un solicitante de préstamo es deseable o no dependiendo de sus ingresos, historial de préstamos y transacciones anteriores, edad y ubicación. Incluso si estas características son interdependientes, estas características aún se consideran de forma independiente. Esta suposición simplifica el cálculo y por eso se considera ingenua. Esta suposición se llama independencia condicional de clase.

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

P(H) es la probabilidad ***a priori***, la forma de introducir conocimiento previo sobre los valores que puede tomar la hipótesis.

P(D|H) es el ***likelihood*** de una hipótesis H dados los datos D, es decir, la probabilidad de obtener D dado que H es verdadera.

P(D) es el ***likelihood marginal*** o ***evidencia***, es la probabilidad de observar los datos D promediado sobre todas las posibles hipótesis H.

P(H|D) es el ***a posteriori***, la distribución de probabilidad final para la hipótesis. Es la consecuencia lógica de haber usado un conjunto de datos, un ***likelihood*** y un ***a priori***.

Teorema de Bayes en términos simples

$$Posterior = \frac{Probabilidad\ x\ Anterior}{Evidencia}$$

La fórmula nos indica la probabilidad de que una hipótesis **H** sea verdadera si algún evento **D** ha sucedido. Esto es importante dado que, normalmente obtenemos la *probabilidad de los efectos dadas las causas*, pero el *teorema de Bayes* nos indica la probabilidad de las *causas* dados los *efectos*.

Por ejemplo, podemos saber cuál es el porcentaje de pacientes con *gripe* que tienen *fiebre*, pero lo que realmente queremos saber es la probabilidad de que un paciente con *fiebre* tenga *gripe*.

Aplicación del clasificador Bayesiano en las imágenes consiste en la información que nos dan los píxeles. Para la clasificación se calculan estadísticos para los píxeles de las zonas de interés, estos estadísticos nos dan información que podemos ocupar para comparar con otras imágenes. El procedimiento es el siguiente:

- Preprocesamiento de las imágenes.
- Procesamiento de máscaras.
- Obtención de regiones.
- Obtención de características.
- Clasificación.

III. Desarrollo

La práctica consiste en clasificar algunos conjuntos de imágenes. En este caso, se utilizaron los conjuntos de comida y de monos.

En primer lugar, se procesaron las imágenes de forma externa, para lo cual se utilizó Photoshop. Con esta herramienta, se aplicó el filtro gaussiano y se realizaron las máscaras, utilizando un fondo negro para denotar las zonas a ignorar en cada máscara.

En el caso de la comida, se utilizaron máscaras más definidas, mientras que en el de los monos, las máscaras se hicieron más suaves, especialmente por el pelo de los monos, que está menos definido con respecto al fondo.

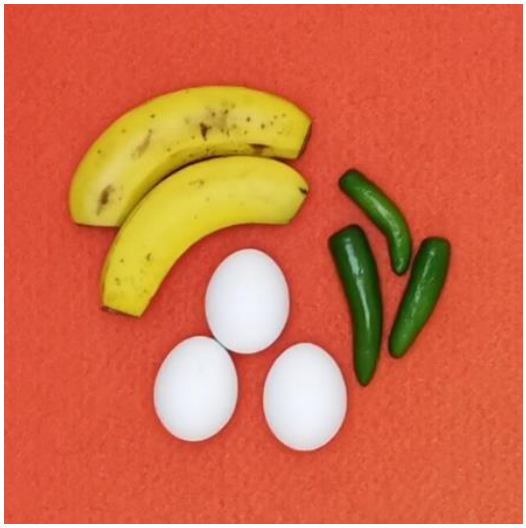


Fig. 1 Ejemplo de imagen original de entrenamiento de comida.

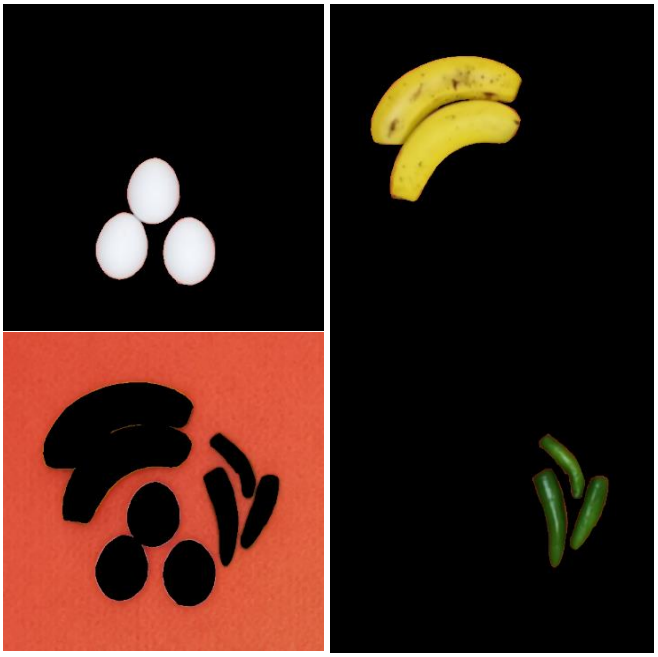


Fig. 2 Máscaras correspondientes a la imagen de la Fig. 1.

Fig. 3 Ejemplo de imagen original de entrenamiento de monos.

Fig. 4 Máscaras correspondientes a la imagen de la Fig. 3.

Se obtiene una máscara a cada imagen para reconocer sobre que píxeles existe una posible categorización. Para esto se define una función que obtiene dicha máscara.

```
# Generates an array per mask; 0s mean the pixel is masked, values different than 0 mean the pixel will be processed
def getMask(imageList):

  maskList = list()
  for image in imageList: # For each image
    img = Image.open(image)
    imgR = np.asarray(img.split()[0]) # Takes R values
    imgG = np.asarray(img.split()[1]) # Takes G values
    imgB = np.asarray(img.split()[2]) # Takes B values
    imgMask = np.zeros_like(imgR) # Blank image for mask
    for i in range(0, len(imgR)): # For each column
      imgMask[i] = imgR[i] + imgG[i] + imgB[i] # Adds the R, G and B values per element
    maskList.append(imgMask)

  return maskList
```

Fig. 5 Función para obtener máscara

Para cada imagen de entrenamiento se obtiene la máscara.

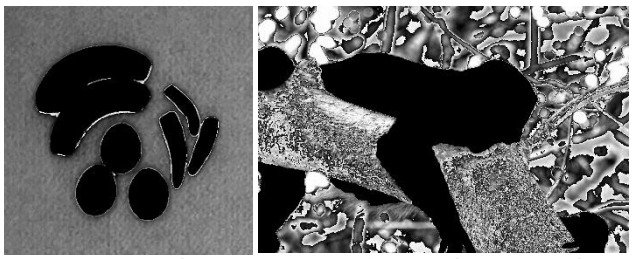


Fig. 6 Obteniendo máscara donde se identifican los píxeles de interés a clasificar.

Nótese que la máscara obtenida muestra los píxeles en negro donde no existe categorización, y los píxeles diferentes de negro son la región de interés.

Es una forma sencilla de identificar los píxeles que nos interesan por cada región en sus diferentes eventos. Después, se obtienen las direcciones o posiciones de los píxeles diferentes de 0, es decir, que no seas negros. Y a estas posiciones o direcciones se les asigna su color de la imagen sin preprocesamiento. Esto se aplicó para todos los eventos de una región y se guarda en una matriz.

```
def featVector(images, masks):

  featVector = np.array([])
  tempComp = np.array([])

  for i in range (0, len(images)):
    image = np.asarray(Image.open(images[i]))
    mask = np.asarray(np.nonzero(masks[i])) # We get an array of indices of non-zero values, which we'll use to get appropiate pixels to get
    colours = image[mask[0], mask[1], :] # We get all colour values for the indicated range; mask[0] indicates column, mask[1] indicates row
    featArray = colours # One row per pixel, 3 features per row (r, g, b)
    if i == 0:
      featVector = featArray
    else:
      featVector = np.vstack((featVector, featArray)) # Add all rows from each image

  return featVector
```

Fig. 7 Obtención de los valores RGB de los píxeles de interés.

Después de obtener los vectores de características, compuestos por los valores RGB de cada píxel, se obtiene el número de eventos totales y se muestra cómo cada renglón contiene los valores RGB asociados a cada píxel de cada categorización.

```
trainBackground_Food = featVector(originalList_Food, masks_Food_Background) # Feature vector
trainBananas_Food = featVector(originalList_Food, masks_Food_Bananas)
trainEggs_Food = featVector(originalList_Food, masks_Food_Eggs)
trainChilies_Food = featVector(originalList_Food, masks_Food_Chilies)

n_samples_Food = trainBackground_Food.shape[0] + trainBananas_Food.shape[0] + trainEggs_Food.shape[0] + trainChilies_Food.shape[0]

print(n_samples_Food) # Total of events
print(trainBackground_Food.shape[1]) # Total features
```

```
1439878
3
```

Fig. 8 Obtención del número de eventos.

Para la obtención de los valores estadísticos, se crea una función que obtiene aquéllos de interés (media, matriz de covarianza y su inversa, determinante de dicha matriz y probabilidad a priori de cada región) para la construcción del modelo y la obtención de regiones.

```
def statistics(trainData, n_samples):

  mean = trainData.mean(axis = 0) # Mean of each feature
  sigma = np.cov(trainData.T) # Covariance matrix
  sigma_det = np.linalg.det(sigma) # Determinant of covariance matrix, to use on Bayes rule
  sigma_inv = np.linalg.inv(sigma) # Inverse of covariance matrix, to use on Bayes rule
  priori = trainData.shape[0] / n_samples # A priori probability, obtained dividing number of samples of a class by total number of samples

  return(mean, sigma_det, sigma_inv, priori)
```

Fig. 9 Función para obtener datos estadísticos.

Para el clasificador bayesiano se utilizó la siguiente formula:

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

Fig. 10 Fórmula del clasificador bayesiano.

El cual se modeló en código para cada uno de sus términos de la fórmula. Esta función nos da la mayor probabilidad y su clase respectivamente.

```
def obtainClass(classes, entry): # Dictionary with classes as keys and statistics as values, and data to classify

  probabilities = dict()

  for class_i, statistic in classes.items(): # For each class
    factor = 1/2 * np.log(statistic[1])
    exponential = -1/2 * np.dot((entry - statistic[0].T) @ statistic[2], entry - statistic[0])
    priori = np.log(statistic[3])
    probabilities[class_i] = exponential - factor + priori # Apply naive Bayes formula

  max_probability = max(probabilities, key = probabilities.get) # Get class with highest probability

  return max_probability
```

Fig. 11 Fórmula del clasificador bayesiano.

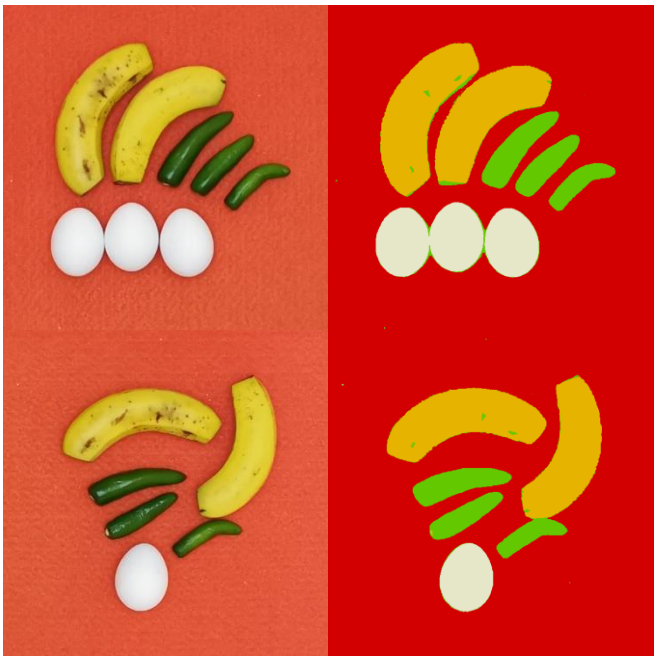
Para ver el funcionamiento de nuestro clasificador, creamos una función que nos crea un lienzo del tamaño de la imagen a clasificar donde ocupamos la función de *obtainClass*, que le corresponde al cálculo de cuál región es más probable que le pertenezca y la pinta de un color dependiendo de dicha región asignada.

```
# Obtain regions identified by colours
def regions(classes, image, colours): # Receives classes dictionary, entry image and colours dictionary

  entryImg = np.asarray(image)
  shape = entryImg.shape[:2]
  shape = shape[::-1] # Fixes shape
  blankImg = Image.new('RGB', shape) # Creates new blank image with same shape as entry image
  arrayImg = np.array(blankImg) # Converts said image to array, for ease of use

  # For each pixel:
  for i in range(blankImg.size[1]):
    for j in range(blankImg.size[0]):
      featVector = np.array([entryImg[i, j, 0], entryImg[i, j, 1], entryImg[i, j, 2]]) # Creates feature vector with RGB values
      highClass = obtainClass(classes, featVector) # Obtains class with highest probability
      arrayImg[i, j] = np.array(colours[highClass]) # Puts respective colour on final image

  regions = Image.fromarray(arrayImg) # Converts array back to image

  return regions
```

Fig. 12 Funcionamiento de nuestro clasificador bayesiano.

IV. Resultados

Después de aplicar el clasificador se observa las siguientes clasificaciones para las imágenes con alimentos:



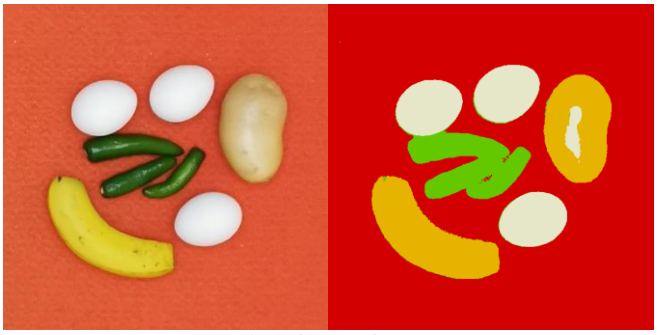


Fig. 13 Clasificación de imágenes de comida

Nótese que, para los valores oscuros, los asocia más al color de los chiles, por eso en la clasificación de los plátanos se observa algunos píxeles de color verde.

En el caso especial de la papa, debido a que no existe dicha categoría, el color claro lo asocia a un huevo y el color oscuro a un plátano.

Ahora, se aplica el clasificador a las otras imágenes:

Fig. 14 Clasificación de imágenes de monos.

Se observa que, para este caso, debido a que sólo existen dos categorías, a los monos le asocia los valores oscuros, al fondo los valores más claros por eso se observan que en algunos troncos o ramas donde hay sombra los clasifica como mono, mientras que a las zonas claras de los monos las clasifica como fondo.

V. Conclusión

La realización de la práctica resultó bastante enriquecedora para comprender en su totalidad el funcionamiento de un clasificador bayesiano. Además, entender el flujo adecuado para realizar un proceso de clasificación, sin importar la naturaleza o tipo de datos con los que se trabaje. El enfoque utilizado en la práctica es profundo, se estudia desde la construcción del modelo hasta su implementación en un caso real, si bien se sabe que existen librerías asociadas a diferentes lenguajes de programación que cuentan con el modelo ya establecido, es importante conocer desde lo básico para poder aventurarnos a utilizar mejores modelos o clasificadores.

El clasificador bayesiano, aunque es antiguo, es una herramienta potente y sencilla de implementar. A pesar de los avances en la Ciencia de Datos, sigue demostrando ser un buen clasificador. Cabe destacar que tiene buenas características como: tiene un costo de cálculo muy bajo, adaptable a un gran conjunto de datos, funciona de mejor manera con variables discretas, se puede utilizar para problemas multiclase, etc.

Por último y como bien se mencionó previamente, el caso de la papa es sin duda un claro ejemplo de lo que sucede en un sistema entrenado que no contempla clases o categorías que no ha visto previamente, realmente es aquí donde podemos observar uno de los grandes limites que tiene los modelos que no son entrenados por reforzamiento, sino que dependen directamente del “labeling” o etiquetado de datos.

Referencias

[1] J. Brownlee, «Naive Bayes Classifier From Scratch in Python,» 2019. [En línea]. Available: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/. [Último acceso: 28 Octubre 2021].

[2] S. Ray, «6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R,» 2017. [En línea]. Available: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/. [Último acceso: 29 Octubre 2021].

[3] N. Janakiev, «Understanding the Covariance Matrix,» 2018. [En línea]. Available: https://datascienceplus.com/understanding-the-covariance-matrix/. [Último acceso: 28 Octubre 2021].

[4] R. Gandhi, «Naive Bayes Classifier,» Towards Data Science, 2018. [En línea]. Available: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c. [Último acceso: 29 Octubre 2021].

[5] NumPy, «NumPy v1.21 Manual,» s. f. [En línea]. Available: https://numpy.org/doc/stable/index.html. [Último acceso: 28 Octubre 2021].