Trabajo práctico 1: Bayes Ingenuo

Ph. D. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Ingeniería en Computación, Programa de Ciencias de Datos,
PAttern Recongition and MAchine Learning Group (PARMA-Group)

26 de octubre de 2022

Fecha de entrega: Domingo 23 de Octubre

Entrega: Un archivo .zip con el código fuente LaTeX o Lyx, el pdf, y un notebook Jupyter, debidamente documentado, con una función definida por ejercicio. A través del TEC-digital.

Modo de trabajo: Grupos de 3 personas.

Integrantes: Dennis Luna Acuña, Graciela Rivera Picado, Luis Enrique Vargas Porras

Resumen

En el presente trabajo práctico se introduce la implementación de redes bayesianas.

1. Implementación de la clasificación multi-clase con Bayes ingenuo (100 puntos)

1. Para el presente ejercicio, se implementará la clasificación de dígitos manualmente escritos en 10 clases (dígitos decimales del 0 al 9). La Figura 1 muestra algunas observaciones del conjunto de datos. El código provisto lee las imágenes del conjunto de datos, y los transforma a matrices binarias de $28 \times 28 = 784$ pixeles. El objetivo de su equipo de desarrollo es utilizar el teorema de Bayes para construir un modelo conocido como Bayes ingenuo, el cual permita estimar la clase a la que pertenece una nueva observación.

2. En el material del curso, se discute el algoritmo de Bayes ingenuo, el cual tiene por objetivo estimar la **probabilidad posterior** de que una observación (imagen en este caso) $\overrightarrow{m} \in \mathbb{N}^{784}$ pertenezca a una clase k como:

$$p\left(t = k|\overrightarrow{m}_i\right)$$

Para aproximarla, se utiliza el teorema de Bayes, el cual luego de desarrollar y simplificar la expresión de tal probabilidad posterior, se concluye

Figura 1: Muestra del conjunto de datos a utilizar.

que esta es proporcional a la multiplicación de la probabilidad a priori de $p\left(t=k\right)$ y la verosimilitud de un pixel (blanco: 1 o negro: 0) $p\left(m_i|t=k\right)$:

$$p\left(t=k|\overrightarrow{m}\right) \propto \prod_{i=0}^{D} p\left(m_i|t=k\right) p\left(t=k\right).$$

La verosimilitud del pixel $p\left(m_i|t=k\right)$ se implementa como la probablidad de que $p\left(m_i=0|t=k\right)$ en caso de que ese pixel i de la observación a evaluar en el modelo sea negro (0),en caso contrario $p\left(m_i=1|t=k\right)$.

a) **(15 puntos)** Implemente el cálculo de las probabilidades a priori $p\left(t\right)$ para las $K=10$ clases en el conjunto de datos de entrenamiento. Realice tal calculo dentro de la funcion *train_model.*

-

Solución: La implementación fue realizada dentro de la función train model, donde se toma como muestra el dataset (expresada en forma de tensor) para cada uno de los formatos de las observaciones, es decir a nivel de escala de grises y de forma binarizada. El resultado final es un tensor unidimensional con 10 posiciones (1 por clase) dando como resultado un 10 % de la probabilidad ya que se cuenta con 60 imágenes por clase. En este punto específico solo se toma como probabilidad marginal el caso binarizado de las observaciones.

A nivel de código en la figura 2 se puede apreciar el detalle de la implementación, donde cálculo de la probabilidad marginal esta dada por $p\left(t=k\right)=\frac{CantImagenesXclase}{N}$

```python
def train_model(train_data_tensor_bin, train_data_tensor_gray, labels_bin, labels_gray, num_classes = 10):

    first_tensor = True
    p_t_tensor_acc = None
    p_m_1_given_k_acc = None
    p_m_0_given_k_acc = None
    mu_given_k_acc = None
    sigma_given_k_acc = None


    for k in range(num_classes):
        # Filtra train_data_tensor por clase para dataset binarizado y en escala de grises
        train_data_tensor_bin_per_k = train_data_tensor_bin[:, labels_bin == k].type(torch.int64)
        train_data_tensor_gray_per_k = train_data_tensor_gray[:, labels_gray == k].type(torch.float64)


        # D = cantidad de filas (784 pixeles)
        # N = cantidad de columnas (600 imágenes)
        D, N = train_data_tensor_bin.shape

        # Estimacion de probabilidad a priori
        p_t_tensor = torch.tensor([train_data_tensor_bin_per_k.shape[1] / N])
```

Inicialización de parámetros

Cálculo Probabilidad Priori

Figura 2: Implementación para el cálculo de probabilidad a priori o marginal

La función train model fue implementada para que posea un alcance de no solo retornar la probabilidad a priori si no de retornar valores como la lista que contiene para cada una de sus posiciones (2 en total), arreglos de tensores con los datos de *p_m_0_given_k* y *p_m_1_given_k*. También se retornan *sigma_given_k_acc* y *mu_given_k_acc*. Dichos parámetros son utilizados en los enfoques que aplican un modelo paramétrico que se discuten más adelante.
En la figura 3 se obtiene el cálculo de la probabilidad a priori:

```python
px,mu_given_k_acc,sigma_given_k_acc,p_t_tensor_acc =
train_model(train_data_tensor_bin, train_data_tensor_gray, labels_training, labels_training, 10)

print(p_t_tensor_acc)
```

Se llama función train model

```
tensor([0.1000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000,
        0.1000])
```

Resultado Obtenido probabilidad a Priori

Figura 3: Resultado del cálculo de la probabilidad a Priori

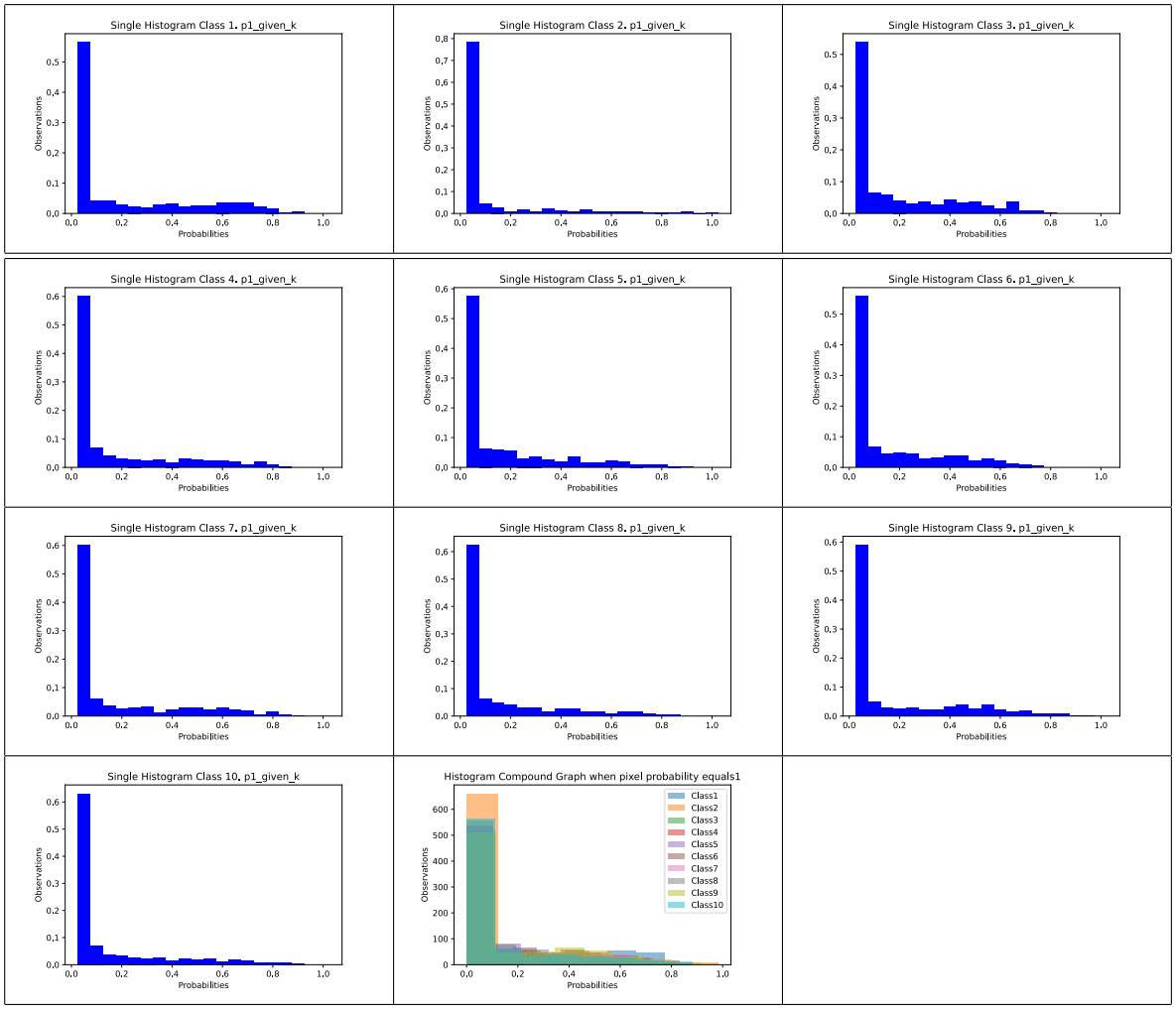
3. Para evaluar la verosimilitud $p(m_i|t)$, es necesario estimar las densidades $p(m_i = 0|t)$ y $p(m_i = 1|t), p(m_i = 2|t), \ldots, p(m_i = 255|t)$ para todos los pixeles $i = 1, \ldots 768$ pixeles.

   a) **(10 puntos extra) Enfoque basado en histogramas**: Para ello se le sugiere crear la matriz *p_m_1_given_k* con 768 filas (una por pixel) y 10 columnas (una por clase), por lo que entonces cada columna corresponde a la densidad de cada pixel. **Para realizar este calculo solo se le permite usar un ciclo *for*, con una iteración por clase k, como máximo**.

   -

   Solución: En un enfoque basado en histogramas se necesitó de la utilidad de la graficación de histogramas singulares por clases y de

un histograma compuesto. Lo primero es tabular el resultado para el caso de los histogramas singulares dada la matriz *p_m_0_given_k* y su histograma compuesto obteniendo los siguientes resultados:



Cuadro 1: Histogramas singulares para el caso *p_m_1_given_k* e Histograma Compuesto

-

Con base a los resultados en el cuadro 1 se tienen las siguientes conclusiones:

1) Las probabilidades graficadas con base al modelo basado en histogramas se resumen en la frecuencia que aparecen probabilidades en el trayecto del eje x. En este caso el histograma com-

puesto nos puede dar una idea inicial de cual sería un serio candidato a la clase predominante. En los resultados obtenidos se puede notar que la clase candidata predominante corresponde a la clase 2 por tener la mayor probabilidad.

2) El uso de histogramas como medio para determinar cual clase posee dominio según la frecuencia de probabilidades graficadas, no es tan seguro o robusto para el estimado de una clase dominante.
3) El método se sustenta en conclusiones visuales las cuales están sujetos a cantidad de clases y cantidad de muestras. El aumento de estos 2 factores significaría la inviabilidad del método.
4) La graficación de histogramas singulares nos pueden dar pistas de cual clase posee mayor frecuencia de aparición de probabilidades a lo largo de el eje x, sin embargo no es eficiente sino se tiene disponible un histograma compuesto para ver su comportamiento en comparación con el resto de clases.

b) **(15 puntos) Enfoque basado en un modelo paramétrico:** Su equipo decide probar un enfoque paramétrico donde se ajuste un modelo paramétrico como una función de densidad Gaussiana (en este caso con parámetros μ y σ). Implemente la estimación de $p\left(m_i|t\right)$ usando un modelo Gaussiano para los pixeles en cada clase.

1) Explique los cambios en cuanto a la representación propuesta en el enfoque basado en histogramas. Qué ventaja tiene el enfoque basado en el modelo paramétrico en cuanto la representación?
-
Solución: En cuanto a la representación propuesta por histogramas es un enfoque que se limita únicamente a la parte visual, haciendo que el alcance del método tenga una eficiencia menor como se pudo apreciar dentro de las conclusiones anteriores, en particular la 4. Una de las ventajas que tiene el enfoque Gaussiano es que tras utilizar los parámetros de μ y σ se logra formar una campana Gaussiana, haciendo que no se penalice de forma estricta las observaciones, ya que en el enfoque de histogramas las observaciones que no caben en las cubetas son usualmente asignadas con valor de cero, que en el caso de hacer una corrección de Laplace seria casi cero. Otra ventaja del enfoque del modelo paramétrico radica en el uso de puntajes en cada clase a través de una ecuación de una forma mas precisa, y con criterio para seleccionar la mayor de todas las probabilidades.
A nivel de implementación la figura 4 muestra a nivel de código el uso de un modelo paramétrico basado en la densidad de la función Gaussiana. Nótese el uso de puntajes para determinar la clase predominante, en este caso como resultado se obtuvo que la clase predominante corresponde a la número 1 (Índice 1 del lista). En comparativa con el modelo con enfoque de histo-

gramas se obtuvo el mismo resultado.

Figura 4: Implementación de modelo con enfoque paramétrico. Función densidad Gaussiana

c) **(10 puntos)** Implemente los dos puntos anteriores en la función *train_model* y retorne una lista con las dos matrices (*[[p_m_0_given_k, p_m_1_given_k]]*), junto con el arreglo de probabilidades a priori para todas las clases.

-

Solución: Para este punto la implementación esta dada en la figura 5 que se ilustra a continuación:

```python
def train_model(train_data_tensor_bin, train_data_tensor_gray, labels_bin, labels_gray, num_classes = 10):

    first_tensor = True
    p_t_tensor_acc = None
    p_m_1_given_k_acc = None
    p_m_0_given_k_acc = None
    mu_given_k_acc = None
    sigma_given_k_acc = None

    for k in range(num_classes):
        # Filtra train_data_tensor por clase para dataset binarizado y en escala de grises
        train_data_tensor_bin_per_k = train_data_tensor_bin[:, labels_bin == k].type(torch.int64)
        train_data_tensor_gray_per_k = train_data_tensor_gray[:, labels_gray == k].type(torch.float64)


        # D = cantidad de filas (784 pixeles)
        # N = cantidad de columnas (600 imágenes)
        D, N = train_data_tensor_bin.shape

        # Estimacion de probabilidad a priori
        p_t_tensor = torch.tensor([train_data_tensor_bin_per_k.shape[1] / N])



        # Estimacion de verosimilitud para cada pixel por clase
        p_m_1_given_k = ((train_data_tensor_bin_per_k == 1).sum(dim=1)/train_data_tensor_bin_per_k.shape[1])+1e-12


        # Transforma tensor de verosimilitud a una sola columna
        p_m_1_given_k = p_m_1_given_k.view(p_m_1_given_k.shape[0], -1)


        # Calculo de media y desviacion estandar
        mu_given_k = torch.mean(train_data_tensor_gray_per_k, dim=1)
        sigma_given_k = torch.std(train_data_tensor_gray_per_k, dim=1)+1e-12


        # Transforma tensor de mu y sigma a una sola columna
        mu_given_k = mu_given_k.view(mu_given_k.shape[0], -1)
        sigma_given_k = sigma_given_k.view(sigma_given_k.shape[0], -1)

        # Estimacion de funcion de densidad de probabilidad

        if(first_tensor):
            first_tensor = False
            p_m_1_given_k_acc = p_m_1_given_k
            p_t_tensor_acc = p_t_tensor
            mu_given_k_acc = mu_given_k
            sigma_given_k_acc = sigma_given_k
            #gaussian_given_k_acc = gaussian_given_k
        else:
            p_m_1_given_k_acc = torch.cat((p_m_1_given_k_acc, p_m_1_given_k), 1)
            p_t_tensor_acc = torch.cat((p_t_tensor_acc, p_t_tensor), 0)
            mu_given_k_acc = torch.cat((mu_given_k_acc, mu_given_k), 1)
            sigma_given_k_acc = torch.cat((sigma_given_k_acc, sigma_given_k), 1)
            #gaussian_given_k_acc = torch.cat((gaussian_given_k_acc, gaussian_given_k), 1)

    # Saca complemento de p_m_1_given_k_acc
    p_m_0_given_k_acc = 1 - p_m_1_given_k_acc

    return (list([p_m_0_given_k_acc, p_m_1_given_k_acc]), mu_given_k_acc, sigma_given_k_acc, p_t_tensor_acc)
```

Inicializar parámetros

Evalúa si es el primer tensor para realizar el acumulado de probabilidad

Se obtiene el complemento

Retorno Valores Lista Prob

Figura 5: Bloque de implementación de la lista con las 2 matrices

d) **(10 puntos)** Implemente la función *test_model(input_torch, p_m_pix_val_given_k, p_t_tensor, num_classes = 10)* la cual realice la estimación de a cual clase pertenece una observación contenida en el vector *input_torch,* para un modelo representado en *p_m_pix_val_given_k, p_t_tensor* (obtenido en el paso anterior). Para ello, el enfoque de Bayes ingenuo estima la función de densidad posterior como sigue:

$$p\left(t=k|\overrightarrow{m}\right) \propto \prod_{i=0}^{D} p\left(m_i|t=k\right) p\left(t=k\right).$$

La clase estimada a la que pertenece la observación $\overrightarrow{m}$ corresponde

entonces a la clase k con mayor probabilidad posterior $p\left(t=k|\overrightarrow{m}\right)$

1) Implemente una funcion *test_model* por cada variante de Bayes Ingenuo.

Solución: Para esta sección se establece *test_model* para las dos variantes del modelo. Su lógica debe ser distinta, ya que la forma de calcular la probabilidad varia una de otra. A continuación se muestra la lógica de programación, y su estructura. Ambas funciones se usan para el apartado anterior, y la explicación de la funcionalidad de cada línea de comando se explican en las figuras siguientes:

3. Test Bernoulli Model Function

```
:
def test_model_bernoulli(input_torch, p_m_pix_val_given_k, p_t_tensor, num_classes = 10):
    #assumes that the input comes in a row
    input_torch_single_column = input_torch.view(input_torch.shape[0], -1)

    p_t_k_given_m = []
    for k in range(num_classes):
        # Extraer probabilidades de acuerdo al input_torch nuevo para cuando pixel es 0
        #print(p_m_pix_val_given_k[0][:,k].shape)
        #print(p_m_pix_val_given_k[0][:,k].unsqueeze(dim=1).shape)
        #print(input_torch_single_column.shape)
        p_m_0_extracted = torch.log(p_m_pix_val_given_k[0][:,k].unsqueeze(dim=1)[input_torch_single_column == 0])
        #p_m_0_extracted = torch.log(p_m_pix_val_given_k[0][:,k][input_torch[k,:] == 0])
        # Estima log para todas las probabilidades extraidas
        p_m_0_estimated = p_m_0_extracted.sum(dim=0)
        # Extraer probabilidades de acuerdo al input_torch nuevo para cuando pixel es 1
        p_m_1_extracted = torch.log(p_m_pix_val_given_k[1][:,k].unsqueeze(dim=1)[input_torch_single_column == 1])
        #p_m_1_extracted = torch.log(p_m_pix_val_given_k[1][:,k][input_torch[k,:] == 1])
        # Estima log para todas las probabilidades extraidas
        p_m_1_estimated = p_m_1_extracted.sum(dim=0)
        # Estimacion de la probabilidad posterior
        p_t_k_given_m.append(p_m_0_estimated + p_m_1_estimated + torch.log(p_t_tensor[k]))

    scores_classes = p_t_k_given_m
    #predicted_label = max(p_t_k_given_m)
    predicted_label = scores_classes.index(max(scores_classes))
    return (predicted_label, scores_classes)#ocupo retorna la clase

new_image_ber = load_single_image(img_index = 65, mode="bin") #Numero 1
test_model_bernoulli(new_image_ber, p_m_pix_val_given_k, p_t_tensor, num_classes = 10)

```

Figura 6: test_model para Bernoulli

4. Test Gaussian Model Function

```python
def test_model_gaussian(input_torch, mu, sigma, p_t_tensor, num_classes = 10):
    #assumes that the input comes in a row
    #input_torch_reshaped = input_torch.view(input_torch.shape[0], -1)
    #print(input_torch_reshaped.shape)

    # Estimacion de la probabilidad de verisimilitud para todas las clases
    #p_gaussian_estimated = evaluate_gaussian(mu, sigma, input_torch_reshaped)+1e-12
    #print(p_gaussian_estimated.shape)

    p_posterior_all = []
    for k in range(num_classes):
        # Estimacion de la probabilidad posterior
        mu_row = mu[:,k].reshape(1, mu.shape[0]).squeeze()
        sigma_row = sigma[:,k].reshape(1, sigma.shape[0]).squeeze()
        p_gaussian_estimated = mvn.logpdf(input_torch, mean=mu_row, cov=sigma_row, allow_singular=True) + torch.log
        p_posterior_all.append(p_gaussian_estimated)
        #p_posterior_all.append(p_gaussian_estimated[:, k].sum(dim=0) + torch.log(p_t_tensor[k]))

    scores_classes = p_posterior_all
    predicted_label = scores_classes.index(max(scores_classes))
    #predicted_label = max(scores_classes)

    return(predicted_label, scores_classes)

new_image_gaus = load_single_image(img_index = 65, mode="gray") #Numero 1

test_model_gaussian(new_image_gaus, mu_all, sigma_all, p_t_tensor, num_classes = 10)
```

Figura 7: test_model para Gaussian

4. **(10 puntos)** Implemente la función *test_model_batch(test_set, labels, p_m_pix_val_given_k, p_t_tensor)* la cual calcule y retorne la tasa de aciertos para un conjunto de observaciones, basado en la función anteriormente implementada *test_model*.

    a) Implemente una funcion *test_model_batch* por cada variante de Bayes Ingenuo.

    Solución: Para este enunciado se utiliza dos funciones para el cálculo de la tasa de aciertos para ambas variantes de Bernoulli y Gaussiana utilizando la misma lógica de programación, pero variando la llamada del algoritmo a utilizar, ya sea *test_model_bernoulli* o *test_model_gaussian*. A continuación se muestra las funciones descritas:

6.Test Bernoulli Model batch Function: Accuracy

```python
def test_Bernoulli_model_batch(test_set, labels, p_m_pix_val_given_k, p_t_tensor):

    number_observations = test_set.shape[0]
    number_correct_predictions = 0
    for observation_id in range(number_observations):
        current_observation = test_set[observation_id]
        predicted_label,scores_classes = test_model_bernoulli(current_observation, p_m_pix_val_given_k, p_t_tensor, num_classes = 10)
        if (predicted_label == labels[observation_id]): #poner la clase label= clase
            number_correct_predictions += 1

    accuracy = number_correct_predictions/number_observations
    print(accuracy)
    return accuracy
```

Figura 8: Test_model_batch Función para Bernoulli

5.Test Gaussian Model batch Function: Accuracy

```
def test_Gaussian_model_batch(test_set, labels, mu_tensor, sigma_tensor, p_t_tensor):

    number_observations = test_set.shape[0]
    number_correct_predictions = 0
    for observation_id in range(number_observations):
        current_observation = test_set[observation_id]
        predicted_label,scores_classes = test_model_gaussian(current_observation, mu_tensor, sigma_tensor, p_t_tensor, num_classes = 10)
        if (predicted_label == labels[observation_id]): #poner la clase label= clase
            number_correct_predictions += 1

    accuracy = number_correct_predictions/number_observations
    print(accuracy)
    return accuracy
```

Figura 9: Test_model_batch Función para Gaussian

2. Prueba del modelo

1. **(10 puntos)** Entrene el modelo con el conjunto de observaciones contenido en la carpeta train, y reporte la tasa de aciertos al utilizar la función anteriormente implementada *test_model_batch* (se espera que la tasa de aciertos sea mayor a 80 %). Verifique y comente los resultados. Es posible que observe valores nulos en el resultado de evaluar la función posterior a través de la función *test_model* la cual implementa la ecuación:

$$p\left(t=k|\overrightarrow{m}\right) \propto \prod_{i=0}^{D} p\left(m_i|t=k\right) p\left(t=k\right).$$

Si observa valores de 0 o nulos en la evaluación de la función, argumente el porqué puede deberse este comportamiento. ¿Cómo se puede corregir el problema detectado, según las herramientas matemáticas estudiadas en clase? Implemente tal enfoque y compruebe los resultados.

a) Compruebe lo anterior usando el enfoque basado en la binarizacion (func. densidad binomial), histogramas y el basado en el modelo Gaussiano. Comente y compare los resultados.

Solución: En esta sección se entrena el modelo con el 100 % de datos y se utiliza la función comentada anteriormente de *test_model_batch* para poder probar la tasa de aciertos. Se puede observar un comportamiento único, y es que cuando se aplica la ecuación manual del gaussiano se observa una cantidad de valores de 0 y nulos, debido a que puede que la varianza sea cero, y esto da como resultado la indefinición de la ecuación al dividir entre cero, por lo que se realiza dos tratamientos/correcciones vistos en clase: El primero es sustituir el valor de varianza por un valor de epsilon, cuando ésta es cero. Y en la segunda, se realiza la corrección de Laplace, ya que según la definición se aumenta el recuento de la variable con cero a un valor pequeño en el numerador, para que la probabilidad

Cuadro 2: Tasa de aciertos con 100 % de Datos de entrenamiento

Enfoque Gaussiano	Enfoque Bernoulli
TA = 83.83 %	TA=91.50 %

total no se convierta en cero o se indefina. La mayoría de las implementaciones del modelo Bayes Ingenuos aceptan esta forma de corrección o una equivalente como parámetro. Por otro lado, cuando se aplica el mvn de la librería de sklearn se puede observar que estos valores 0 y nulos no se observan en estimaciones de las matrices de probabilidades. Lo anterior fue comprobado por un testeo que se realiza con los datos de estudio.

En cuanto a la tasa de aciertos, los resultados que se tienen al entrenar el modelo son los siguientes:

Se puede observar que el modelo con una mayor tasa de aciertos es el enfoque de Bernoulli con un 91,5 %, el cuál también tiene una mayor velocidad computacional (el resultado, computa más rápido que el resto).

2. **(20 puntos)** Particione los datos de forma aleatoria con 70 % de las observaciones para entrenamiento y 30 % para prueba (a partir de la carpeta *train*). Calcule la tasa de aciertos para 10 corridas (**idealmente 30**), cada una con una partición de entrenamiento y otra de prueba distintas. Reporte los resultados de las corridas en una tabla, además de la media y desviación estándar de la tasa de aciertos para las 10 corridas. Para realizar las particiones puede usar la libreria *sklearn.*

   a) Compruebe lo anterior usando el enfoque basado en histogramas, binarizado y el basado en el modelo Gaussiano. Comente y compare los resultados.

   Solución: Se particiona los datos usando la librería de *sklearn* y a continuación tiene el siguiente resultados para 10 corridas de la tasa de aciertos, con su respectiva media y desviación estándar:

Cuadro 3: Tasas de aciertos para 10 corridas usando particiones de entrenamiento y prueba distintas

Corrida	1	2	3	4	5	6	7	8	9	10	μ	σ
Gaussiano	69.44	66.11	58.33	65.56	56.67	66.11	68.89	67.22	67.78	68.33	65.44	4.39
Bernoulli	81.11	78.33	77.22	80.00	68.33	75.56	77.78	77.78	81.11	80.56	77.78	3.80
μ	75.27	72.22	67.77	72.78	62.50	70.83	73.33	72.50	74.45	74.45	-	-
σ	8.25	8.64	13.36	10.21	8.24	6.68	6.29	7.47	9.43	8.65	-	-

De acuerdo a los resultados mostrados en la tabla 3 se determina que el enfoque con mayor tasa de aciertos es el de Bernoulli, el cual también es el que tiene menor desviación estándar. Algunas observaciones dadas

durante el desarrollo de ambos enfoques es que el Bernoulli converge bastante rápido durante su ejecución, mientras que en el Gaussiano el tiempo de ejecución es considerablemente mayor, esto debido a que para el cálculo de la probabilidad de la función de densidad se utilizó un algoritmo multivariable donde se estima la matriz de covarianza de Σ, lo cual computacionalmente puede ser exigente dependiendo si el conjunto de datos contiene muchos features y observaciones en sí.

Adicionalmente, en el enfoque de Bernoulli se realiza una binarización del conjunto de datos, lo cual simplifica bastante la obtención de las probabilidades de verosimilitud para cada pixel (con valor de '0' o '1') para cada una de las clases. Otro consideración a contemplar es que para el enfoque Gaussiano, dicho algoritmo depende de 2 parámetros: la media y varianza; en donde la varianza puede necesitar en ocasiones de una corrección de Laplace para que cuando sea cero, se sustituya por un valor que tienda a cero y así no se indefina la probabilidad global.