Desarrollo Taller 1 AI: Decisión Bayesiana

Juan Sebastián Bravo Santacruz

I. Objetivos

1) Visualizar e interpretar datos de acuerdo con la estadística descriptiva de los mismos.

2) Diseñar e implementar clasificadores Bayesianos.

II. Conjunto de datos data_2D

El conjunto de datos data_2D (ver archivo data.npy) contiene datos pertenecientes a dos clases {a, b}.

A. *Grafique los datos utilizando un color distintivo para cada clase:*

Inicialmente se realizó la importación de los datos del archivo data.npy y se separaron los conjuntos data_2D y data_3D en 2 distintas variables:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import model_selection

# Importar la informacion del archivo data
data = np.load('data.npy', allow_pickle=True)
data = data.item()

'''---------------------------- Punto 1 ----------------------------'''

# Separar los conjuntos de datos data_2D y data_3D
data_2D = data['data_2D']
data_3D = data['data_3D']
```

Una vez separados los conjuntos de datos, se extrajeron los valores de las clases a y b y se realizó la gráfica de estas dos clases usando el módulo matplotlib.

```python

# Adquirir informacion de las 2 clases del conjunto de datos data_2D
data_2D_a = data_2D['data_a']
data_2D_b = data_2D['data_b']

# Grafica de los datos con colores distintivos para cada clase
plt.scatter(data_2D_a[0], data_2D_a[1], c='red', label='Clase a')
plt.scatter(data_2D_b[0], data_2D_b[1], c='blue', label='Clase b')
plt.title('Conjunto de datos data_2D ')
plt.xlabel('Atributo 1')
plt.ylabel('Atributo 2')
plt.legend()
plt.grid()
```

En este caso, se eligió el color rojo para la clase a y el color azul para la clase b para poder distinguirlos adecuadamente en la gráfica.

La gráfica obtenida se presenta en la siguiente imagen:

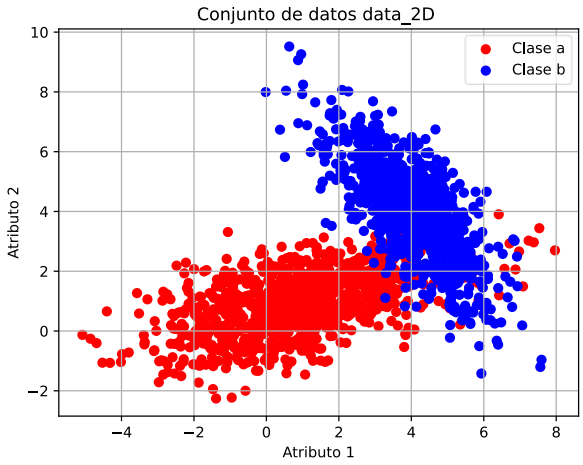

Figura 1. Conjunto de datos data_2D.

B. *Determine el centro de cada clase* {μ*a*, μ*b*}*:*

Para encontrar el centro de cada una de las clases, se utilizó la función mean() del módulo numpy:

```python
# Calculo del centro de cada clase
u_2D_a = data_2D_a.mean(axis=1)
u_2D_b = data_2D_b.mean(axis=1)
```

En este caso, al tener 2 atributos, se obtiene un vector de 2 valores de media, uno por cada atributo:

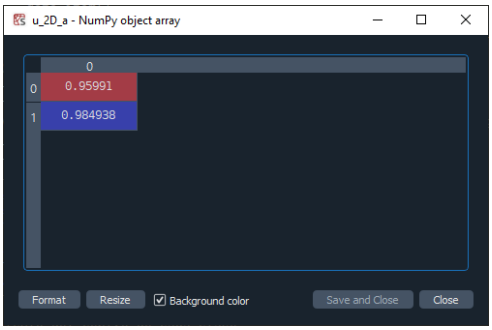

Figura 2. Vector medias para clase a.

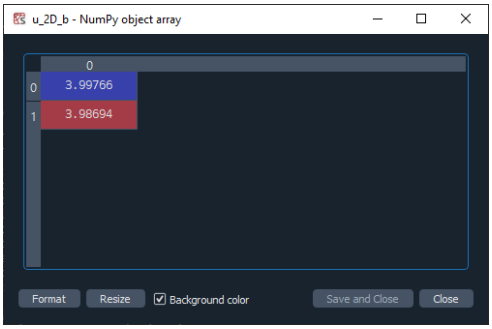

Figura 3. Vector medias para clase b.

C. *Determine las matrices de covarianza de cada clase {Ka, Kb}. ¿Qué se puede concluir?:*

Para encontrar las matrices de covarianza de cada una de las clases, se utilizó la función cov() del módulo numpy:

```
33  # Calculo de las matrices de covarianza de cada clase
34  K_2D_a = np.cov(data_2D_a)
35  K_2D_b = np.cov(data_2D_b)
```

En este caso, al tener 2 atributos, se obtiene una matriz cuadrada de 2x2:

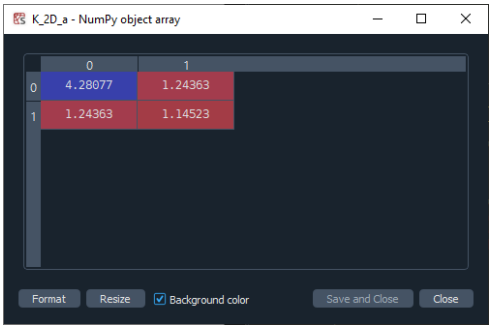

Figura 4. Matriz de covarianza clase a.

De la matriz de covarianza de la clase a, se puede concluir que los atributos no son independientes debido a que no es una matriz diagonal, y que tienen una relación directa (a medida que el atributo 1 aumenta, el atributo 2 aumenta), adicionalmente, se puede concluir que los datos varían más en el atributo 1 que en el atributo 2.

Lo anteriormente dicho, se puede comprobar al ver la gráfica de datos realizados en el inciso A. del ejercicio.

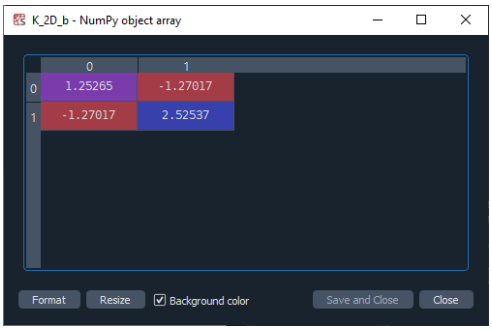

Figura 5. Matriz de covarianza clase b.

De la matriz de covarianza de la clase b, se puede concluir que los atributos no son independientes debido a que no es una matriz diagonal, y que tienen una relación inversa (a medida que el atributo 1 aumenta, el atributo 2 disminuye), adicionalmente, se puede concluir que los datos varían más en el atributo 2 que en el atributo 1.

Lo anteriormente dicho, se puede comprobar al ver la gráfica de datos realizados en el inciso A. del ejercicio.

D. *Determine y visualice el histograma de los datos.*

Se utilizó el módulo matplotlib para encontrar el histograma de los datos, haciendo uso de la función histrogram2d() y meshgrid haciendo un histograma para cada una de las clases como se muestra en el siguiente código:

```
37  # Histograma clase_a data_2D
38  fig = plt.figure()
39  ax = fig.add_subplot(projection='3d')
40  hist, xedges, yedges = np.histogram2d(data_2D_a[0], data_2D_a[1], bins=(20,20))
41  xpos, ypos = np.meshgrid(xedges[:-1]+xedges[1:], yedges[:-1]+yedges[1:])
42
43  xpos = xpos.flatten()/2.
44  ypos = ypos.flatten()/2.
45  zpos = np.zeros_like (xpos)
46
47  dx = xedges [1] - xedges [0]
48  dy = yedges [1] - yedges [0]
49  dz = hist.flatten()
50
51  ax.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average')
52
53  plt.title('Histograma clase_a data_2D')
54  plt.xlabel('Atributo 1')
55  plt.ylabel('Atributo 2')
56
57  # Histograma clase_b data_2D
58  fig = plt.figure()
59  ax = fig.add_subplot(projection='3d')
60  hist, xedges, yedges = np.histogram2d(data_2D_b[0], data_2D_b[1], bins=(20,20))
61  xpos, ypos = np.meshgrid(xedges[:-1]+xedges[1:], yedges[:-1]+yedges[1:])
62
63  xpos = xpos.flatten()/2.
64  ypos = ypos.flatten()/2.
65  zpos = np.zeros_like (xpos)
66
67  dx = xedges [1] - xedges [0]
68  dy = yedges [1] - yedges [0]
69  dz = hist.flatten()
70
71  ax.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average')
72
73  plt.title('Histograma clase_b data_2D')
74  plt.xlabel('Atributo 1')
75  plt.ylabel('Atributo 2')
76  plt.show()
77
```

Los histogramas obtenidos se presentan en las siguientes imágenes:

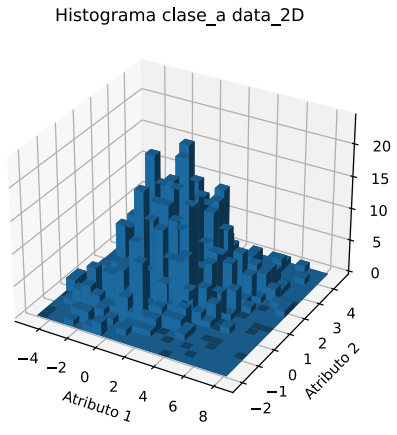

Figura 6. Histograma clase a data_2D.

En el histograma realizado para la clase a, se puede observar que la mayor frecuencia de datos se encuentra entorno al valor central de los datos muy cercano a (1,1).

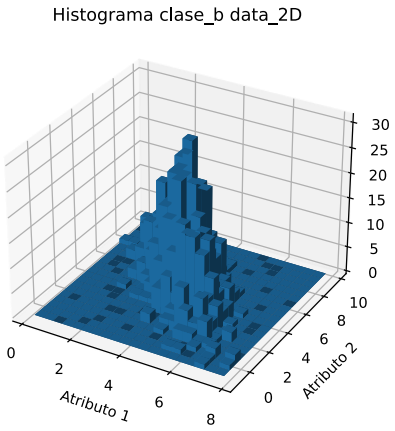

Figura 7. Histograma clase b data_2D.

En el histograma realizado para la clase b, se puede observar que la mayor frecuencia de datos se encuentra entorno al valor central de los datos muy cercano a (4,4).

III. Conjunto de datos data_3D

A. *Divida los datos aleatoriamente en conjunto de prueba (20 %) y de entrenamiento (80 %).*

La división de datos se realizó adquiriendo inicialmente los datos de cada una de las clases para el conjunto data_3D, seguidamente se utilizó la función train_test_split() del módulo sklearn como se muestra en la imagen:

```
# Adquirir información de las 2 clases del conjunto de datos data_3D
data_3D_a = data_3D['data_a']
data_3D_b = data_3D['data_b']

# División de datos en conjuntos de prueba y entrenamiento
training_a, testing_a = model_selection.train_test_split(data_3D_a, test_size=int(0.2*len(data_3D_a)), train_size=int(0.8*len(data_3D_a)))
training_b, testing_b = model_selection.train_test_split(data_3D_b, test_size=int(0.2*len(data_3D_b)), train_size=int(0.8*len(data_3D_b)))
```

En este, para el parámetro test_size, se encontró el 20% del número total de datos de cada clase y para el parámetro train_size se encontró el 80% del número total de datos de cada clase.

B. *Visualice el conjunto de entrenamiento con un color para cada clase.*

Para graficar el conjunto de entrenamiento de cada clase, se utilizó la función scatter3D pasando como parámetro las columnas de las variables de entrenamiento creadas para cada clase como se muestra en el siguiente código:

```
# Visualizacion de datos
ax = plt.axes(projection ="3d")
ax.scatter3D(training_a[:,0], training_a[:,1], training_a[:,2], c='red', label='Clase a')
ax.scatter3D(training_b[:,0], training_b[:,1], training_b[:,2], c='blue', label='Clase b')
plt.title('Conjunto de entrenamiento datos data_3D ')
ax.set_xlabel('Atributo 1')
ax.set_ylabel('Atributo 2')
ax.set_zlabel('Atributo 3')
plt.legend()
plt.grid()
plt.show()
```

La gráfica obtenida es la siguiente:

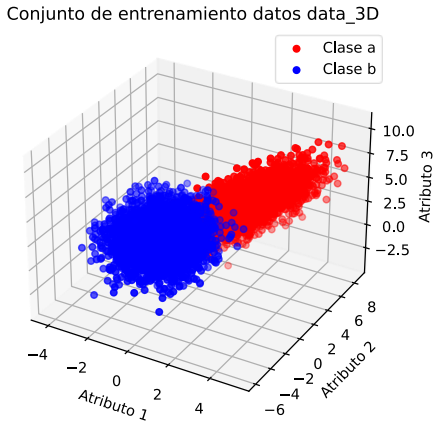

Figura 8. Conjunto de entramiento data_3D.

C. *Implemente un clasificador Bayesiano Gaussiano, y:*

i. Estime la función de verosimilitud de cada clase:

Inicialmente para la implantación de la función de verosimilitud presentada en la siguiente ecuación:

$$f_X(X|c_i)P(c_i) = \frac{P(c_i)}{\sqrt{2\pi^n|K|}} e^{\left[-\frac{1}{2}(X-\mu)^T K^{-1}(X-\mu)\right]}$$

Se encontró el número de atributos y las probabilidades a priori de cada una de las clases, en este caso, asumiendo que estaban distribuidas homogéneamente:

```
# Numero de atributos
n = np.size(data_3D_a,axis=1)

# Probabilidades a pirori
P_3D_a = len(training_a)/(len(training_a) + len(training_b))
P_3D_b = len(training_b)/(len(training_a) + len(training_b))
```

Posteriormente, se encontraron los valores centrales y las matrices de covarianza para cada una de las clases.

```
# Calculo del centro de cada clase
u_3D_a = training_a.mean(axis=0)
u_3D_b = training_b.mean(axis=0)

# Matriz de covarianza:
K_3D_a = np.cov(np.transpose(training_a))
K_3D_b = np.cov(np.transpose(training_b))
```

Al tener 3 atributos se tiene un vector con 3 valores de media, uno por cada atributo.

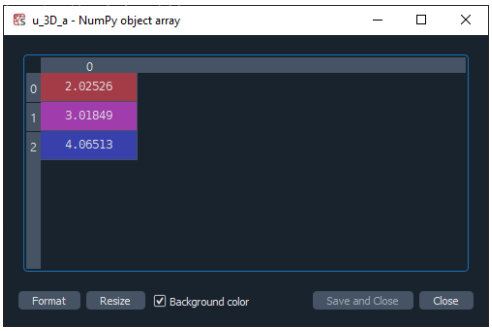

Figura 9. Vector medias para clase a.

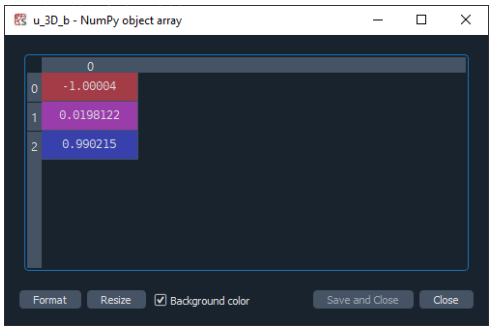

Figura 10. Vector medias para clase a.

Adicionalmente, se tienen matrices de covarianza de 3x3, debido al número de atributos:

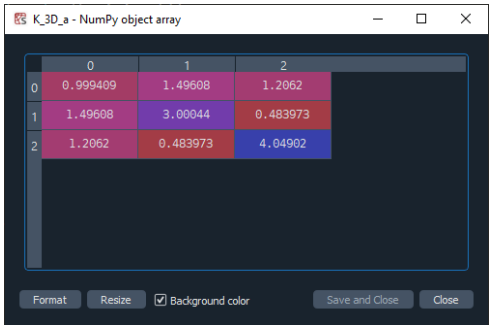

Figura 11. Matriz de covarianza clase a.

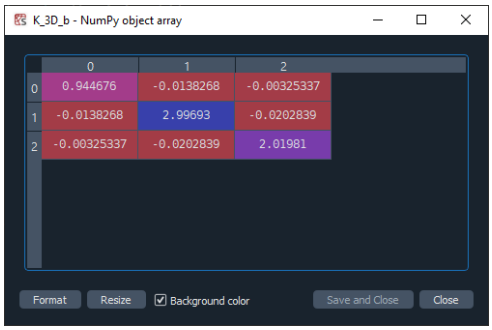

Figura 12. Matriz de covarianza clase b.

De las matrices de covarianza, se puede observar que en caso de la clase a, se tiene una dependencia entre atributos, mientras que para el caso de la clase b, se podría decir existe una dependencia mínima entre estos.

Una vez encontradas cada una de las constantes necesarias para encontrar la función, se creó una matriz de prueba que contenía los conjuntos de prueba de la clase a y b, se creó un vector con las etiquetas de clasificación reales para posteriormente calcular el error de la clasificación y finalmente un vector para almacenar la salida de la clasificación a realizar:

```
# Matriz de prueba
X = np.concatenate((testing_a, testing_b), axis=0)

# vector de comparacion
y_real = np.concatenate((np.zeros((len(testing_a),1)), np.ones((len(testing_b),1))), axis=0)

# vector con las clasificaciones
y_bayes = np.zeros((len(X),1))
```

Finalmente, se implementó la función, asignando un 0 para la clase a y un 1 para la clase b como se muestra en el siguiente código:

```
for i in range(len(X)):
    # Funcion de verosimilitud de cada clase:
    P_a_X = P_3D_a*(1/(np.sqrt(2*(np.pi**n)*np.linalg.det(K_3D_a))))*(
            np.exp(-0.5*np.matmul(np.matmul(X[i,:]-u_3D_a,np.linalg.inv(K_3D_a)),X[i,:]-u_3D_a)))

    P_b_X = P_3D_b*(1/(np.sqrt(2*(np.pi**n)*np.linalg.det(K_3D_b))))*(
            np.exp(-0.5*np.matmul(np.matmul(X[i,:]-u_3D_b,np.linalg.inv(K_3D_b)),X[i,:]-u_3D_b)))

    # Vector de probabilidades
    P = (P_a_X, P_b_X)

    # Proceso de clasificacion
    clase = P.index(max(P))

    # Clase a
    if clase==0:
        y_bayes[i] = 0

    # Clase b
    if clase==1:
        y_bayes[i] = 1
```

ii. Visualice la clasificación realizada sobre el conjunto de prueba:

Para la visualización del conjunto de prueba, inicialmente, se crearon 2 nuevas matrices para cada una de las clases en base a la clasificación realizada y seguidamente se graficó el conjunto haciendo uso de la función scatter3D():

```
# Clasificacion del conjunto de prueba
a_bayes = np.array([X[i] for i in range(len(y_bayes)) if y_bayes[i] == 0])
b_bayes = np.array([X[i] for i in range(len(y_bayes)) if y_bayes[i] == 1])

# Visualizacion conjunto de prueba clasificado
plt.figure()
ax = plt.axes(projection ="3d")
ax.scatter3D(a_bayes[:,0], a_bayes[:,1], a_bayes[:,2], c='red', label='Clase a')
ax.scatter3D(b_bayes[:,0], b_bayes[:,1], b_bayes[:,2], c='blue', label='Clase b')
plt.title('Conjunto de prueba datos data_3D clasificado con Bayes Gaussiano')
ax.set_xlabel('Atributo 1')
ax.set_ylabel('Atributo 2')
ax.set_zlabel('Atributo 3')
plt.legend()
plt.grid()
```

La gráfica obtenida es la siguiente:

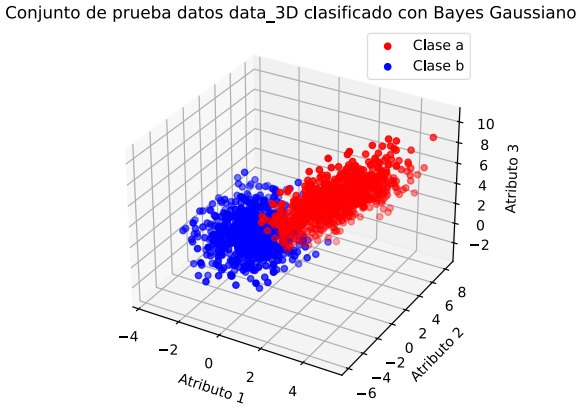

Figura 13. Conjunto de prueba clasificado con Bayes Gaussiano.

iii. Determine el error de clasificación sobre el conjunto de prueba:

Se determinó el error de clasificación, encontrando la cantidad de clasificaciones mal realizadas dividido el número de elementos clasificados.

```
# Carlculo error porcentual de la clasificacion
error_bayes = 100*sum(y_bayes != y_real)/len(y_real)
print(f"Error Bayes Gaussiano: {error_bayes[0]:.3f}%")
```

Los errores porcentuales obtenidos en 3 ejecuciones del programa fueron:

```
Error Bayes Gaussiano: 1.875%
Error Bayes Gaussiano: 2.875%
Error Bayes Gaussiano: 3.125%
```

D. *Implemente un clasificador Bayesiano Gaussiano Naive, y:*

i. Estime la función de verosimilitud de cada clase:

En este caso, la función implementada en se presenta en la siguiente ecuación:

$$f_X(X|c_i)P(c_i) = \frac{P(c_i)^n}{\sqrt{2\pi^n(\sigma_1^2 \dots \sigma_n^2)}} e^{\left[-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2\right]}$$

Se encontró el número de atributos y las probabilidades a priori de cada una de las clases, en este caso, asumiendo que estaban distribuidas homogéneamente. Estas son las mismas que en el caso anterior. Adicionalmente, se encontraron las desviaciones estándar para cada una de las clases.

```
# Probabilidades a pirori y medias iguales al clasificador anterior

# Desviaciones estandar
std_a = training_a.std(axis=0)
std_b = training_b.std(axis=0)

```

Al tener 3 atributos, se obtiene 1 desviación estándar por cada uno de los atributos:

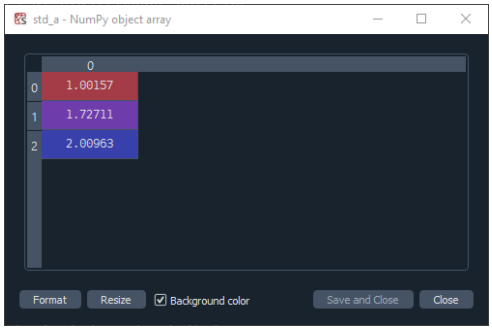

Figura 14. Desviaciones estandar clase a.

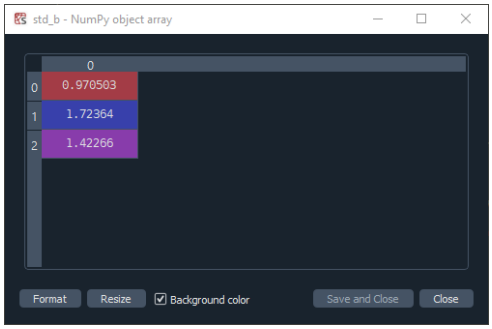

Figura 15. Desviaciones estandar clase b.

Se creó un vector con las etiquetas de clasificación reales para posteriormente calcular el error de la clasificación y finalmente un vector para almacenar la salida de la clasificación a realizar:

```
# vector de comparacion
y_real = np.concatenate((np.zeros((len(testing_a),1)), np.ones((len(testing_b),1))), axis=0)

# vector con las clasificaciones
y_bayes_naive = np.zeros((len(X),1))
```

Finalmente, se implementó la función, asignando un 0 para la clase a y un 1 para la clase b como se muestra en el siguiente código:

```
for i in range(len(X)):
    # Lista con probabilidades de atributos
    P_a_X = []
    P_b_X = []
    for j in range(n):
        # Calculo de probabilidades a posteriori de cada atributo
        P_a_X.append(P_3D_a*(1/(np.sqrt(2*(np.pi**n))*std_a[j]))*(np.exp(-0.5*(X[i,j]-u_3D_a)**2/std_a[j]**2)))
        P_b_X.append(P_3D_b*(1/(np.sqrt(2*(np.pi**n))*std_b[j]))*(np.exp(-0.5*(X[i,j]-u_3D_b)**2/std_b[j]**2)))

    # Probabilidad por clase
    P_a_X = np.prod(P_a_X)
    P_b_X = np.prod(P_b_X)

    # Vector de probabilidades
    P = (P_a_X, P_b_X)

    # Proceso de clasificacion
    clase = P.index(max(P))

    # Clase a
    if clase==0:
        y_bayes_naive[i] = 0

    # Clase b
    if clase==1:
        y_bayes_naive[i] = 1

```

ii. Visualice la clasificación realizada sobre el conjunto de prueba:

Para la visualización del conjunto de prueba, inicialmente, se crearon 2 nuevas matrices para cada una de las clases en base a la clasificación realizada y seguidamente se graficó el conjunto haciendo uso de la función scatter3D():

```
# Clasificacion del conjunto de prueba
a_bayes_naive = np.array([X[i] for i in range(len(y_bayes_naive)) if y_bayes_naive[i] == 0])
b_bayes_naive = np.array([X[i] for i in range(len(y_bayes_naive)) if y_bayes_naive[i] == 1])

# Visualizacion conjunto de prueba clasificado
plt.figure()
ax = plt.axes(projection ="3d")
ax.scatter3D(a_bayes_naive[:,0], a_bayes_naive[:,1], a_bayes_naive[:,2], c='red', label='Clase a')
ax.scatter3D(b_bayes_naive[:,0], b_bayes_naive[:,1], b_bayes_naive[:,2], c='blue', label='Clase b')
plt.title('Conjunto de prueba datos data_3D clasificado con Bayes Gaussiano Naive')
ax.set_xlabel('Atributo 1')
ax.set_ylabel('Atributo 2')
ax.set_zlabel('Atributo 3')
plt.legend()
plt.grid()
plt.show()

```

La gráfica obtenida es la siguiente:

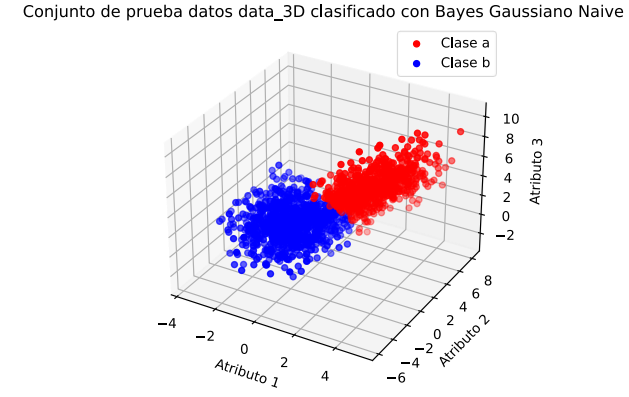

Figura 16. Conjunto de prueba clasificado con Bayes Gaussiano Naive.

iii. Determine el error de clasificación sobre el conjunto de prueba:

Se determinó el error de clasificación, encontrando la cantidad de clasificaciones mal realizadas dividido el número de elementos clasificados.

```
223  # Carlculo error porcentual de la clasificacion
224  error_bayes_naive = 100*sum(y_bayes_naive != y_real)/len(y_real)
225  print(f"Error Bayes Gaussiano Naive: {error_bayes_naive[0]:.3f}%")
```

Los errores porcentuales obtenidos en 3 ejecuciones del programa fueron:

```
Error Bayes Gaussiano Naive: 7.688%
Error Bayes Gaussiano Naive: 7.750%
Error Bayes Gaussiano Naive: 8.188%
```

IV. CONCLUYA SOBRE LOS RESULTADOS OBTENIDOS DE LOS CLASIFICADORES 2.C Y 2.D

Partiendo de los errores de clasificaciones encontrados, se puede decir que el clasificador Bayesiano Gaussiano funciona correctamente teniendo errores menores al 5%.

En el caso del clasificador a Bayesiano Gaussiano Naive, se siguen obteniendo buenos resultandos con errores menores al 10%. Este incremento en el error se debe a la aproximación que se hace al decir que los atributos son independientes entre sí, ya que como se pudo observar en la matriz de covarianza de la clase a, estos sí presentan una dependencia.

En general, se puede decir que ambos clasificadores tienen un buen rendimiento y al observar los resultados de clasificación.