Desarrollo Taller 2 AI: PCA

Juan Sebastián Bravo Santacruz

I. Objetivos

1) Utilizar el método de Análisis de Componentes Principales (PCA) y analizar su funcionalidad.

II. Conjunto de datos data_3D

Utilizando el conjunto de datos data 3D (datos del punto 2 del Taller 1):

A. *Determine la transformación de $\mathbb{R}^3$ a $\mathbb{R}^2$ utilizando PCA (sobre el conjunto de entrenamiento):*

Inicialmente se realizó la importación de los módulos numpy, sklearn y matplotlib. Se importaron también 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']
```

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:

```python
# 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']

# Division 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.

Para el proceso de reducción de reducción de dimensión, inicialmente se quitó el valor medio a los conjuntos de entrenamiento tanto para la clase a como para la clase b de los datos:

```python
# Se remueve la media de cada dato del conjunto de entrenamiento:
training_a_bar = training_a - np.mean(training_a, axis=0)
training_b_bar = training_b - np.mean(training_b, axis=0)
```

Una vez obtenidos los conjuntos de entrenamiento centrados en el origen, se calculó la matriz de covarianza de estos y posteriormente se encontraron los valores y vectores propios de las matrices:

```python
# Se calcula la matriz de covarianza de los datos
A_a = np.transpose(training_a_bar)
K_a = np.cov(A_a)

A_b = np.transpose(training_b_bar)
K_b = np.cov(A_b)

# Se encuentran los vectores y valores propios de K
eig_vals_a, eig_vecs_a = np.linalg.eig(K_a)
eig_vals_b, eig_vecs_b = np.linalg.eig(K_b)

```

En este caso, al tener 3 atributos, se espera tener una matriz de covarianza de 3x3.

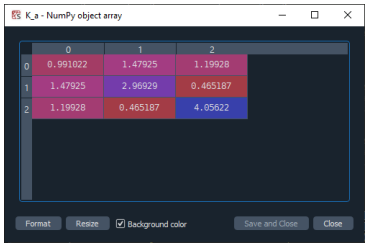

Figura 1. Matriz de covarianza clase a.

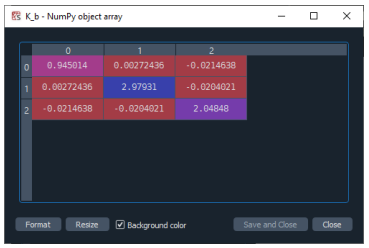

Figura 2. Matriz de covarianza clase b.

Por otra parte, se tienen 3 vectores propios con 3 componentes cada uno ya que se trata de un conjunto de datos de 3 dimensiones y 3 valores propios, uno por cada vector.

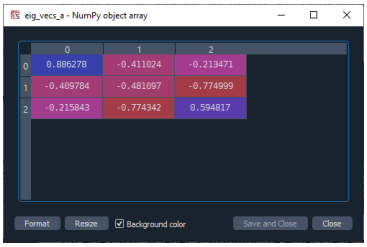

Figura 3. Vectores propios clase a.

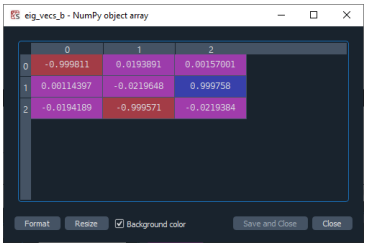

Figura 4. Vectores propios clase b.

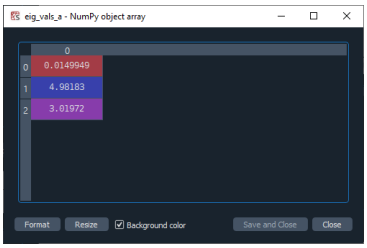

Figura 5. Valores propios clase a.

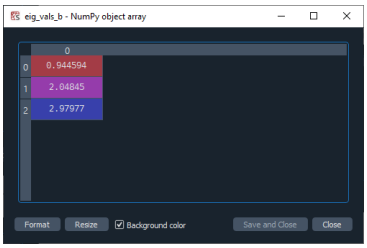

Figura 6. Valores propios clase b.

Se conservaron los 2 valores propios más grandes y sus respectivos valores propios para crear los nuevos conjuntos de entrenamiento siguiendo las siguientes ecuaciones:

$$M' = \begin{pmatrix} v_1' & v_2' & \cdots & v_m' \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{dxm}$$

$$A' = \begin{pmatrix} x_1' & x_2' & \cdots & x_m' \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{mxN}$$

$$A' = M'^{T} A$$

```
# Se conservan solo los m valores propios más grandes y sus respectivos vectores propios
# Reduccion de R3 a R2 ---> m = 2
m = 2

# Para clase a:
M_a_pri = eig_vecs_a[:,:m] # Matriz de vectores propios reducida (dxm)
A_a_pri = np.matmul(np.transpose(M_a_pri), A_a) # Matriz de transformacion con los minimos componentes
new_training_a = np.transpose(A_a_pri) # Nueva matriz de caracteristicas

# Para clase a:
M_b_pri = eig_vecs_b[:,:m] # Matriz de vectores propios reducida (dxm)
A_b_pri = np.matmul(np.transpose(M_b_pri), A_b) # Matriz de transformacion con los minimos componentes
new_training_b = np.transpose(A_b_pri) # Nueva matriz de caracteristicas

```

Resultando así 2 nuevos conjuntos de entrenamiento para cada una de las clases, con igual número de elementos, pero con habiendo reducido los atributos de 3 a 2.

B. *Implemente un clasificador Bayesiano Gaussiano sobre los datos transformados, 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:

```
'''----------------Clasificador Bayesiano Gaussiano----------------'''
# Numero de atributos
n = m

# Probabilidades a pirori
P_new_a = len(new_training_a)/(len(new_training_a) + len(new_training_b))
P_new_b = len(new_training_b)/(len(new_training_a) + len(new_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_new_a = new_training_a.mean(axis=0)
u_new_b = new_training_b.mean(axis=0)

# Matriz de covarianza:
K_new_a = np.cov(np.transpose(new_training_a))
K_new_b = np.cov(np.transpose(new_training_b))

```

Al tener ahora una reducción a 2 atributos se tiene un vector con 2 valores de media, uno por cada atributo.

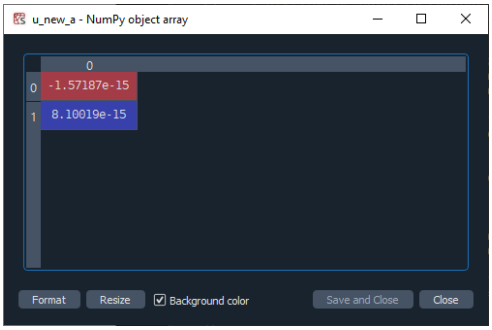

Figura 7. Vector medias para clase a.

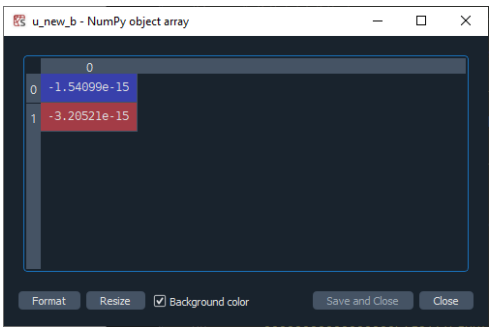

Figura 8. Vector medias para clase a.

Del valor de las medias, se puede observar cómo este es 0, debido a que el conjunto de datos de entrenamiento nuevo, se encuentra ubicado ahora en el origen.

Adicionalmente, se tienen matrices de covarianza de 2x2, debido al número de atributos:

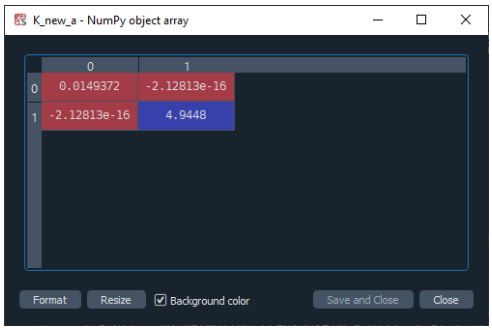

Figura 9. Matriz de covarianza clase a.

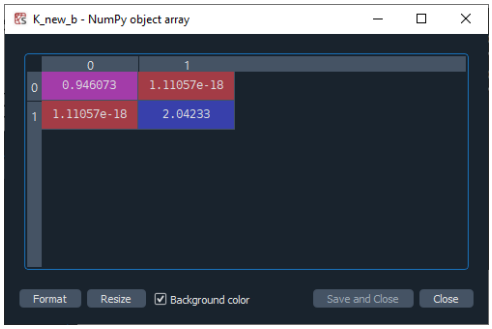

Figura 10. Matriz de covarianza clase b.

De las matrices de covarianza, se puede observar que existe una dependencia mínima entre los nuevos atributos del conjunto de datos.

ii. Determine la transformación de $\mathbb{R}^3$ a $\mathbb{R}^2$ utilizando PCA (sobre el conjunto de prueba):

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 aplicando PCA. Para esto se resto la media encontrada en el conjunto de entrenamiento y se multiplicó por la matriz M' también encontrada con el conjunto de entrenamiento.

```
# Matriz de prueba aplicando PCA

# Para clase a:
testing_a_bar = testing_a - np.mean(training_a, axis=0)
A_a_test_pri = np.matmul(np.transpose(M_a_pri), np.transpose(testing_a_bar))
new_testing_a = np.transpose(A_a_test_pri)

# Para clase b:
testing_b_bar = testing_b - np.mean(training_b, axis=0)
A_b_test_pri = np.matmul(np.transpose(M_b_pri), np.transpose(testing_b_bar))
new_testing_b = np.transpose(A_b_test_pri)

X = np.concatenate((new_testing_a, new_testing_b), axis=0)
```

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(new_testing_a),1)), np.ones((len(new_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_new_a*(1/(np.sqrt(2*(np.pi**n)*np.linalg.det(K_new_a))))*(
            np.exp(-0.5*np.matmul(np.matmul(X[i,:]-u_new_a,np.linalg.inv(K_new_a)),X[i,:]-u_new_a)))

    P_b_X = P_new_b*(1/(np.sqrt(2*(np.pi**n)*np.linalg.det(K_new_b))))*(
            np.exp(-0.5*np.matmul(np.matmul(X[i,:]-u_new_b,np.linalg.inv(K_new_b)),X[i,:]-u_new_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
```

iii. 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 scatter ():

```
# 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(dpi = 150)
plt.scatter(a_bayes[:,0], a_bayes[:,1], c='red', label='Clase a')
plt.scatter(b_bayes[:,0], b_bayes[:,1], c='blue', label='Clase b')
plt.title('Clasificacion realizada')
plt.xlabel('Atributo 1')
plt.ylabel('Atributo 2')
plt.legend()
plt.grid()
plt.show()
```

Adicionalmente, previamente se graficó el conjunto de entrenamiento nuevo para tener noción de la clasificación realizada.

```
# Visualizacion conjunto de entrenamiento
plt.figure(dpi = 150)
plt.scatter(new_training_a[:,0], new_training_a[:,1], c='red', label='Clase a')
plt.scatter(new_training_b[:,0], new_training_b[:,1], c='blue', label='Clase b')
plt.title('Conjunto de entrenamiento aplicando PCA')
plt.xlabel('Atributo 1')
plt.ylabel('Atributo 2')
plt.legend()
plt.grid()
```

Las gráficas obtenidas son las siguientes:

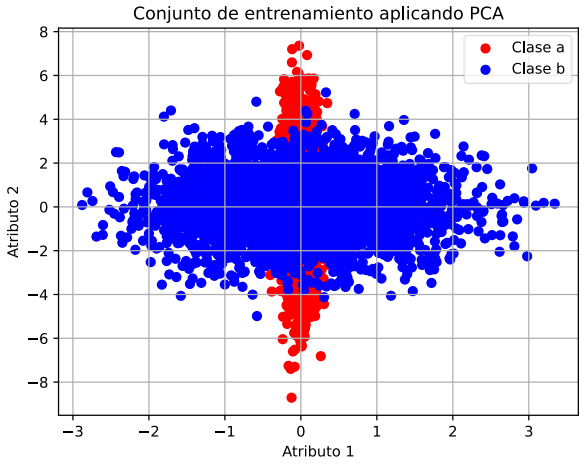

Figura 11. Conjunto de entrenamiento aplicando PCA.

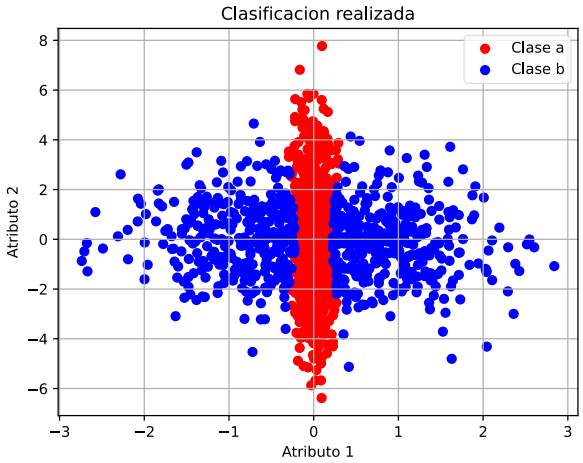

Figura 12. Conjunto de prueba clasificado con Bayes Gaussiano.

De la gráfica de clasificación, se puede apreciar que este realizó una clasificación buena que y solo presenta errores en aquellas secciones donde se superponen las clases.

iv. 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.

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

Los errores porcentuales obtenidos en 3 ejecuciones del programa fueron:

```
Error Bayes Gaussiano: 12.062%
Error Bayes Gaussiano: 11.125%
Error Bayes Gaussiano: 10.812%
```

III. Concluya sobre los resultados obtenidos de los clasificadores del Taller 1 con respecto al clasificador desarrollado en este taller.

Para realizar el proceso de comparación de los resultados con los clasificadores del Taller 1, se utilizó el mismo conjunto de entrenamiento para que el proceso de comparación sea justo. Para esto se utilizó Spyder, dejando constante la variable de training para ambos scripts.

Los errores encontrados fueron los siguientes:

Errores porcentuales		
Bayes Gaussiano	Bayes Gaussiano Naive	Bayes Gaussiano PCA
2.812 %	7.875 %	11.625 %

Partiendo de los errores de clasificación encontrados, se puede decir que si bien es cierto los clasificadores Bayesiano Gaussiano y Bayesiano Gaussiano Naive sin PCA presentan los errores más bajos, el clasificador Bayesiano Gaussiano aplicando la reducción de dimensión, presenta un rendimiento bueno y no muy alejado del Bayesiano Gaussiano Naive sin PCA.