In [12]:
```
#JULIAN GIRALDO CARDONA COMPUTACION BLANDA CUBA
```

In [13]:
```
# PROCESAMIENTO DIGITAL

# Se importan las librería numpy y las funciones de preprocesamiento

import numpy as np
from sklearn import preprocessing

# Datos de prueba
input_data = np.array([[-5.1, 4.9, -3.3],
[1.2, -7.9, 3.2],
[3.9, 0.4, -2.1],
[-9.3, 1.9, 3.5]])
print(input_data)
```

```
[[-5.1  4.9 -3.3]
 [ 1.2 -7.9  3.2]
 [ 3.9  0.4 -2.1]
 [-9.3  1.9  3.5]]
```

In [14]:
```
# Binarizar los datos

data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\nDatos binarizados:\n", data_binarized)
```

```
Datos binarizados:
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

In [15]:
```
# Imprimir la media y la desviación estándar

print("\nANTES:")
print(input_data)
print("Media =", input_data.mean(axis=0))
print("Desviación estándar =", input_data.std(axis=0))
```

```
ANTES:
[[-5.1  4.9 -3.3]
 [ 1.2 -7.9  3.2]
 [ 3.9  0.4 -2.1]
 [-9.3  1.9  3.5]]
Media = [-2.325 -0.175  0.325]
Desviación estándar = [5.18477338 4.74519494 3.0564481 ]
```

In [17]:
```
# Remover la media

data_scaled = preprocessing.scale(input_data)
print(input_data)
print("\nDESPUÉS:")
print("Media =", data_scaled.mean(axis=0))
print("Desviación estándar =", data_scaled.std(axis=0))
```

```
[[-5.1  4.9 -3.3]
 [ 1.2 -7.9  3.2]
 [ 3.9  0.4 -2.1]
 [-9.3  1.9  3.5]]

DESPUÉS:
Media = [-5.55111512e-17 -4.16333634e-17 -5.55111512e-17]
Desviación estándar = [1. 1. 1.]
```

In [19]:
```
# Escalamiento Min Max

data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max escalamiento de datos:\n", data_scaled_minmax)
```

```
Min max escalamiento de datos:
 [[0.31818182 1.         0.        ]
 [0.79545455 0.         0.95588235]
 [1.         0.6484375  0.17647059]
 [0.         0.765625   1.        ]]
```

In [20]:
```
# Normalización de datos

data_normalized_l1 = preprocessing.normalize(input_data,
norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data,
norm='l2')
print("\nL1 dato normalizado:\n", data_normalized_l1)
print("\nL2 dato normalizado:\n", data_normalized_l2)
```

```
L1 dato normalizado:
 [[-0.38345865  0.36842105 -0.2481203 ]
 [ 0.09756098 -0.64227642  0.2601626 ]
 [ 0.609375    0.0625     -0.328125  ]
 [-0.63265306  0.1292517   0.23809524]]

L2 dato normalizado:
 [[-0.65347033  0.62784405 -0.42283375]
 [ 0.13941241 -0.9177984   0.37176644]
 [ 0.87690281  0.08993875 -0.47217844]
 [-0.9192614   0.18780609  0.34595859]]
```

In [28]:
```
# Manejo de etiquetas
import numpy as np
from sklearn import preprocessing
# Se definen algunas etiquetas simples
input_labels = ['red', 'black', 'green', 'green', 'yellow', 'yellow',
'white']
# Se crea un codificador de etiquetas y se ajustan las etiquetas

encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

# Se imprime el mapeo entre palabras y números

print("\nMapeo de etiquetas:")
for i, item in enumerate(encoder.classes_):
    print(item, '-->', i)

# Codificar un conjunto de etiquetas con el codificador

test_labels = ['yellow', 'red', 'white']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))

# Decodificar un conjunto de valores usando el codificador

encoded_values = [3, 1, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list(decoded_list))
```

```
Mapeo de etiquetas:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels = ['yellow', 'red', 'white']
Encoded values = [4, 2, 3]

Encoded values = [3, 1, 4, 1]
Decoded labels = ['white', 'green', 'yellow', 'green']
```