INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE:

MACHINE LEARNING

EVIDENCIAS EXÁMEN 3|

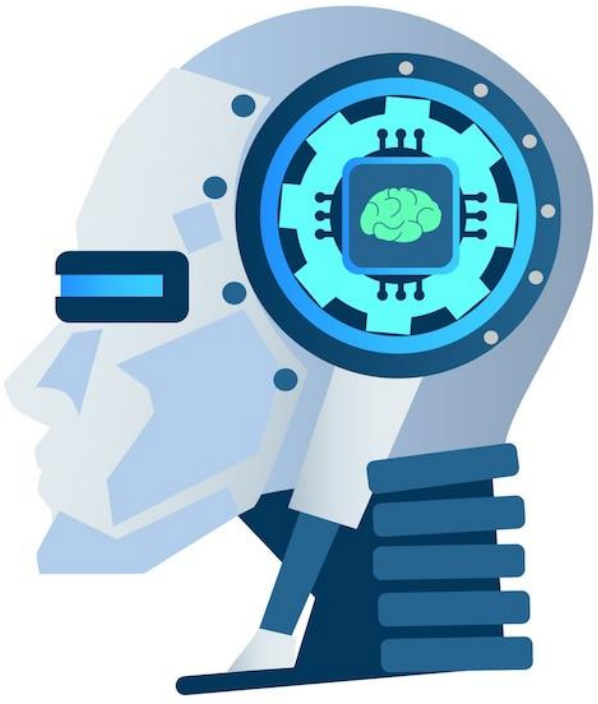
INTEGRANTES:

Hernández Hernández Roberto Isaac

Gonzalez Llamosas Noe Ramses

PROFESOR:

Ortiz Castillo Marco Antonio



FECHA DE ENTREGA: 10/01/2025

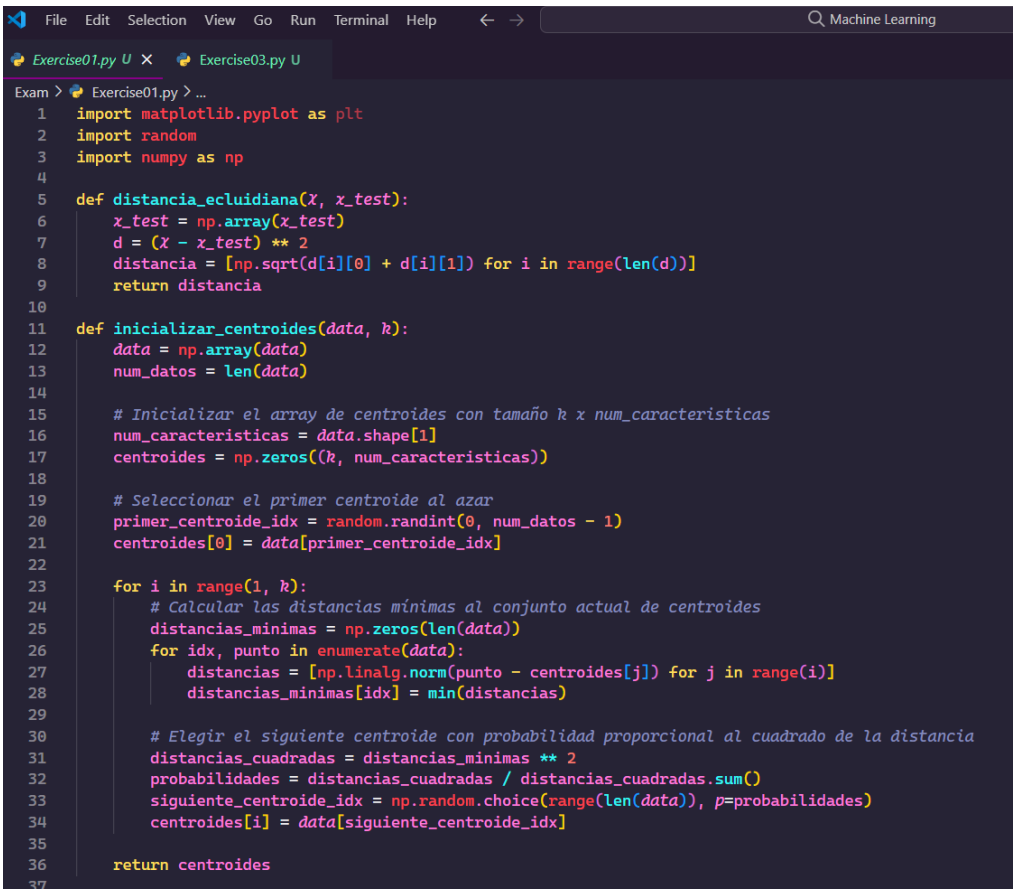
1. **Sea la siguiente matriz de características:**

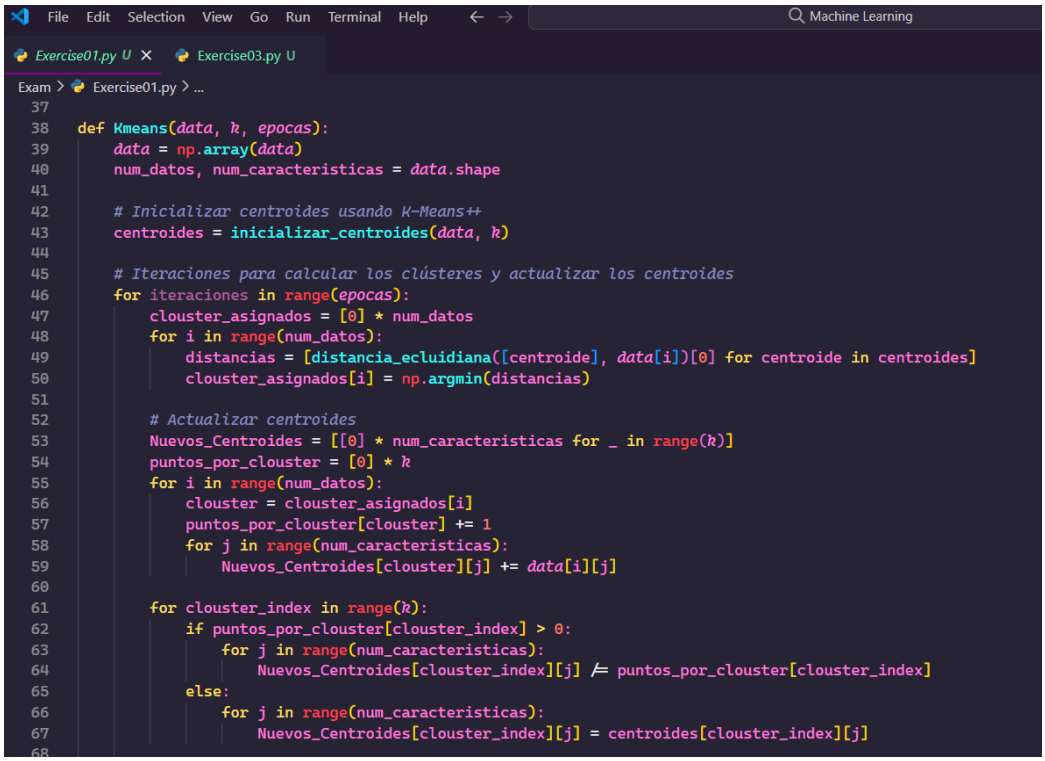
X_1	X_2	X_3	X_4	X_5	X_6	X_7
0	2	0	10	0	6	0
0	2	0	8	0	5	0
5	0	10	0	2	0	1
5	0	9	0	1	0	1.5
3	6	0	0	9	7	0
3	5	0	0	8	9	0

Realice un programa que:

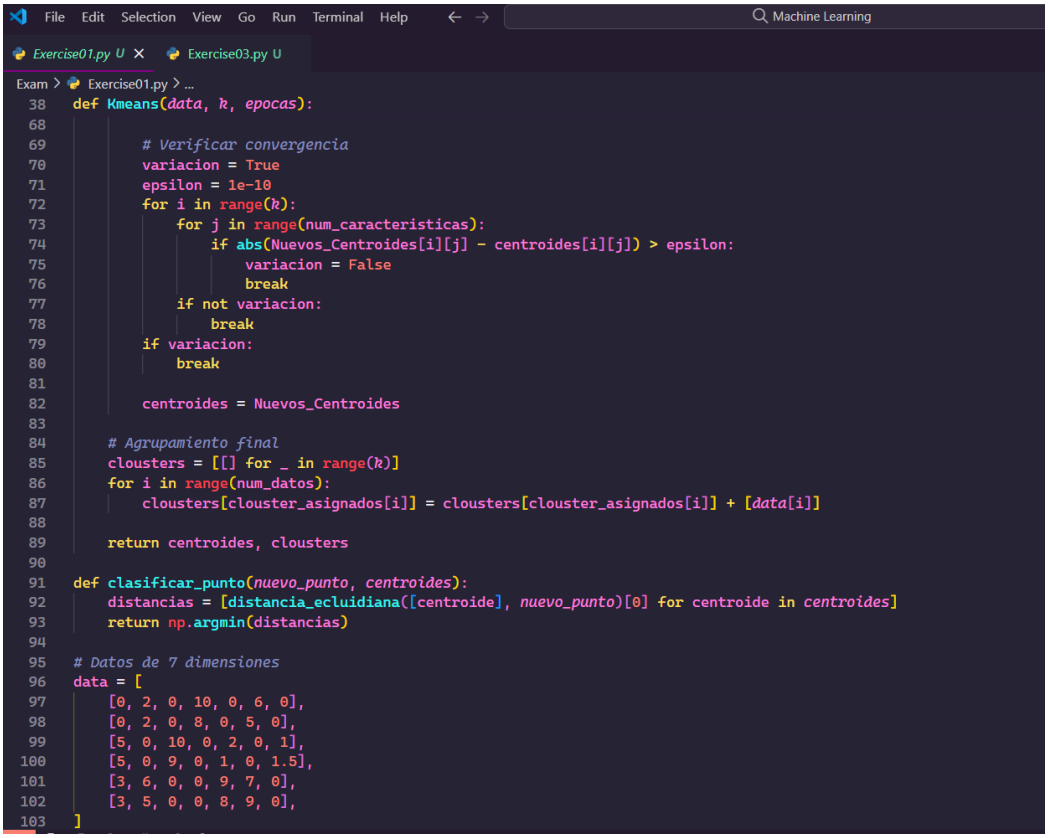
a) **Clasifique cada uno de los datos mediante el método (visto en clase) que más adecuado sea.**

b) **Clasifique [1, 0, 0, 7, 0, 3, 0]**

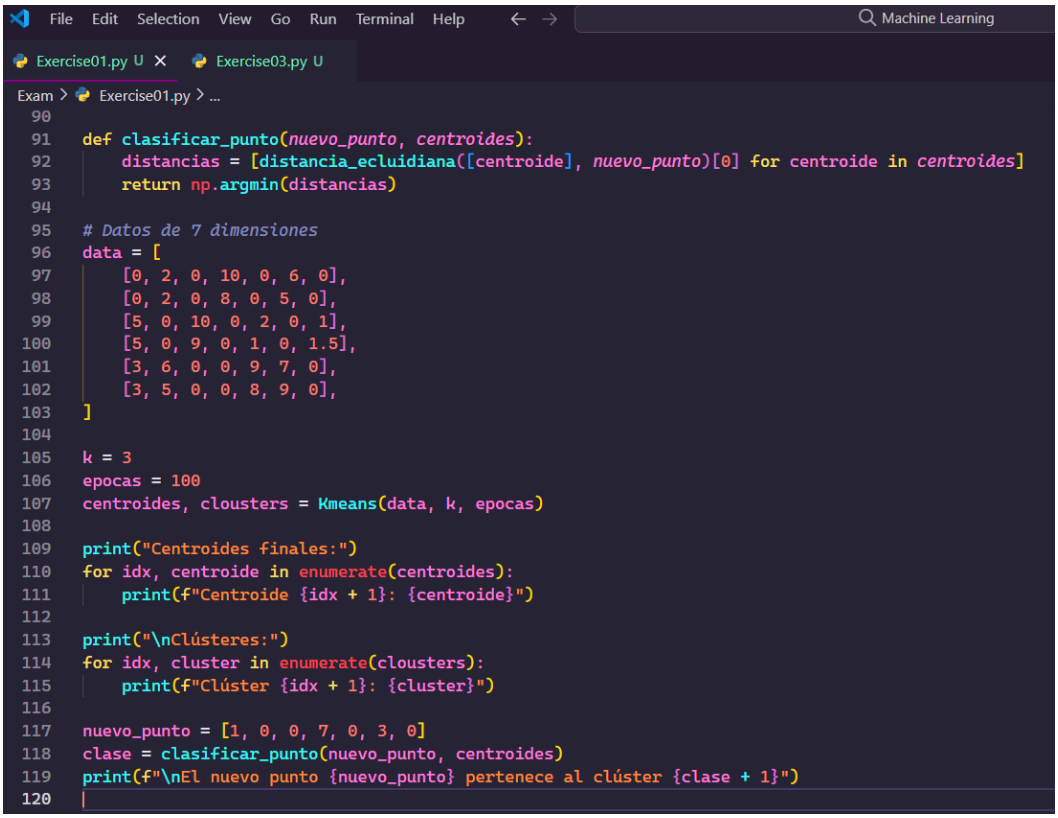




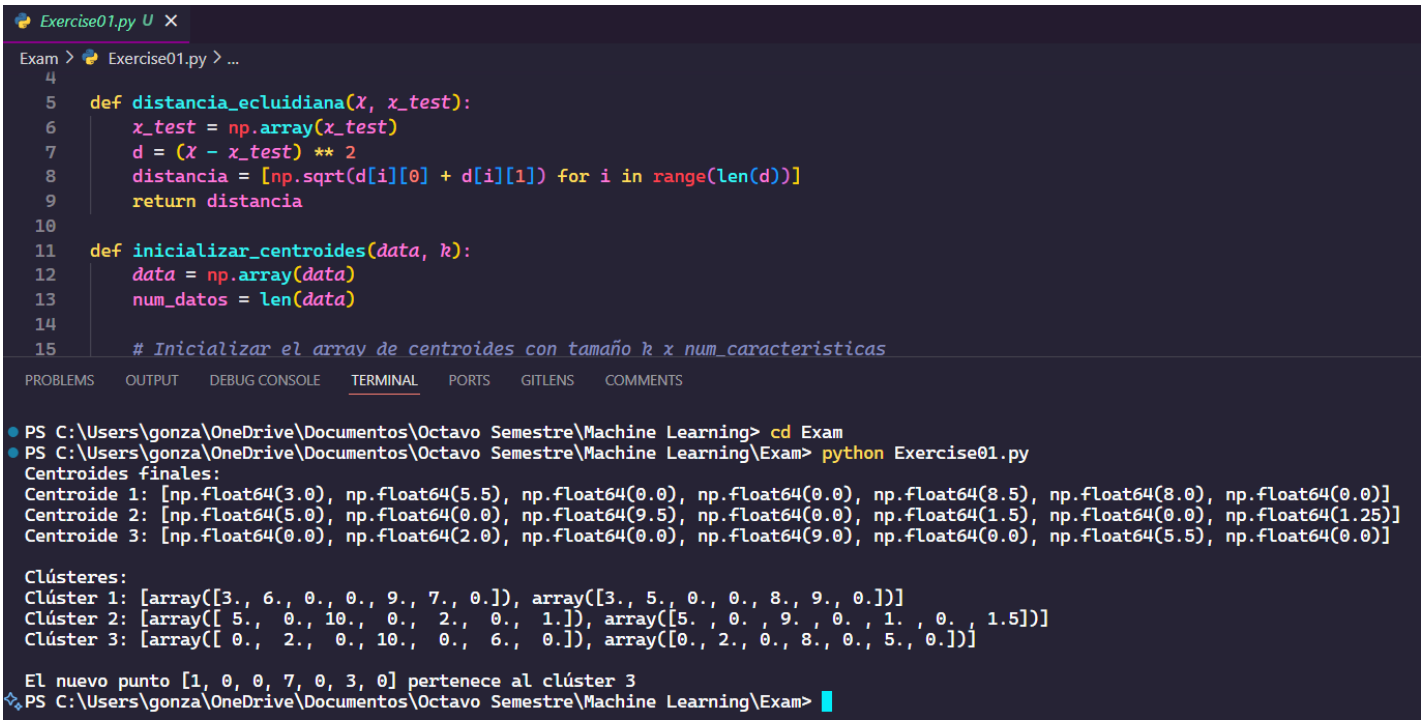

```python
def Kmeans(data, k, epocas):
    data = np.array(data)
    num_datos, num_caracteristicas = data.shape

    # Inicializar centroides usando K-Means++
    centroides = inicializar_centroides(data, k)

    # Iteraciones para calcular los clústeres y actualizar los centroides
    for iteraciones in range(epocas):
        clouster_asignados = [0] * num_datos
        for i in range(num_datos):
            distancias = [distancia_ecluidiana([centroide], data[i])[0] for centroide in centroides]
            clouster_asignados[i] = np.argmin(distancias)

        # Actualizar centroides
        Nuevos_Centroides = [[0] * num_caracteristicas for _ in range(k)]
        puntos_por_clouster = [0] * k
        for i in range(num_datos):
            clouster = clouster_asignados[i]
            puntos_por_clouster[clouster] += 1
            for j in range(num_caracteristicas):
                Nuevos_Centroides[clouster][j] += data[i][j]

        for clouster_index in range(k):
            if puntos_por_clouster[clouster_index] > 0:
                for j in range(num_caracteristicas):
                    Nuevos_Centroides[clouster_index][j] /= puntos_por_clouster[clouster_index]
            else:
                for j in range(num_caracteristicas):
                    Nuevos_Centroides[clouster_index][j] = centroides[clouster_index][j]
```




```python
def Kmeans(data, k, epocas):

        # Verificar convergencia
        variacion = True
        epsilon = 1e-10
        for i in range(k):
            for j in range(num_caracteristicas):
                if abs(Nuevos_Centroides[i][j] - centroides[i][j]) > epsilon:
                    variacion = False
                    break
            if not variacion:
                break
        if variacion:
            break

        centroides = Nuevos_Centroides

    # Agrupamiento final
    clousters = [[] for _ in range(k)]
    for i in range(num_datos):
        clousters[clouster_asignados[i]] = clousters[clouster_asignados[i]] + [data[i]]

    return centroides, clousters

def clasificar_punto(nuevo_punto, centroides):
    distancias = [distancia_ecluidiana([centroide], nuevo_punto)[0] for centroide in centroides]
    return np.argmin(distancias)

# Datos de 7 dimensiones
data = [
    [0, 2, 0, 10, 0, 6, 0],
    [0, 2, 0, 8, 0, 5, 0],
    [5, 0, 10, 0, 2, 0, 1],
    [5, 0, 9, 0, 1, 0, 1.5],
    [3, 6, 0, 0, 9, 7, 0],
    [3, 5, 0, 0, 8, 9, 0],
]
```



Resultados en terminal

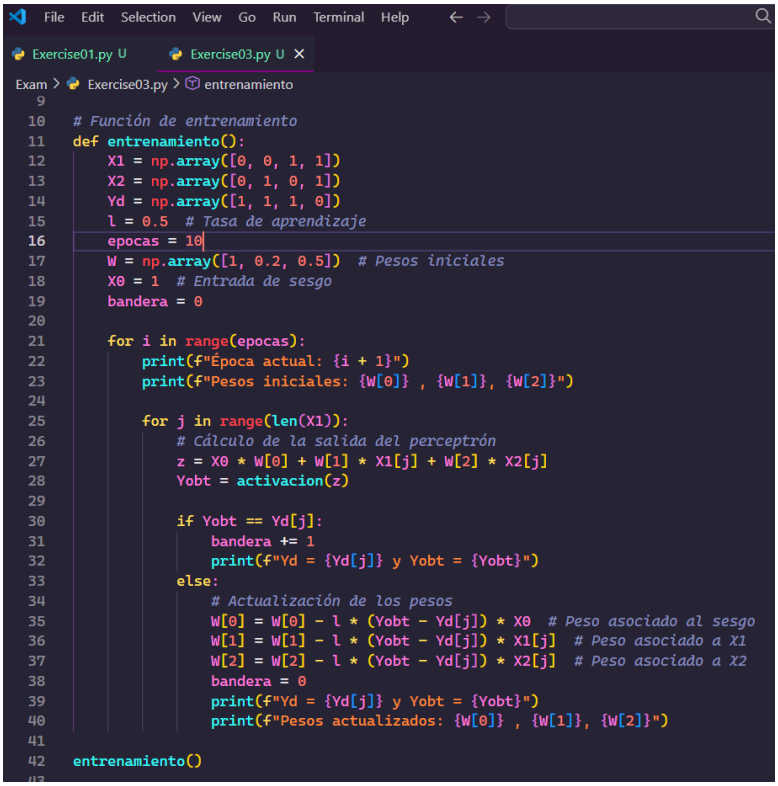


2. Diga con M.S.V. ¿Cómo sería el algoritmo para clasificar los datos mostrados? ¿Dónde y por qué se clasificaría el punto P?

Dado que las muestras forman una circunferencia en capas, la implementación de una máquina de soporte vectorial (SVM) requiere un cambio de dimensión. Esto implica que, al manejar tres tipos de características distintas, podemos representar una característica en un espacio 2D, otra en un espacio 3D y la última en un espacio 4D. De esta manera, es posible clasificar los datos de manera efectiva utilizando este algoritmo.

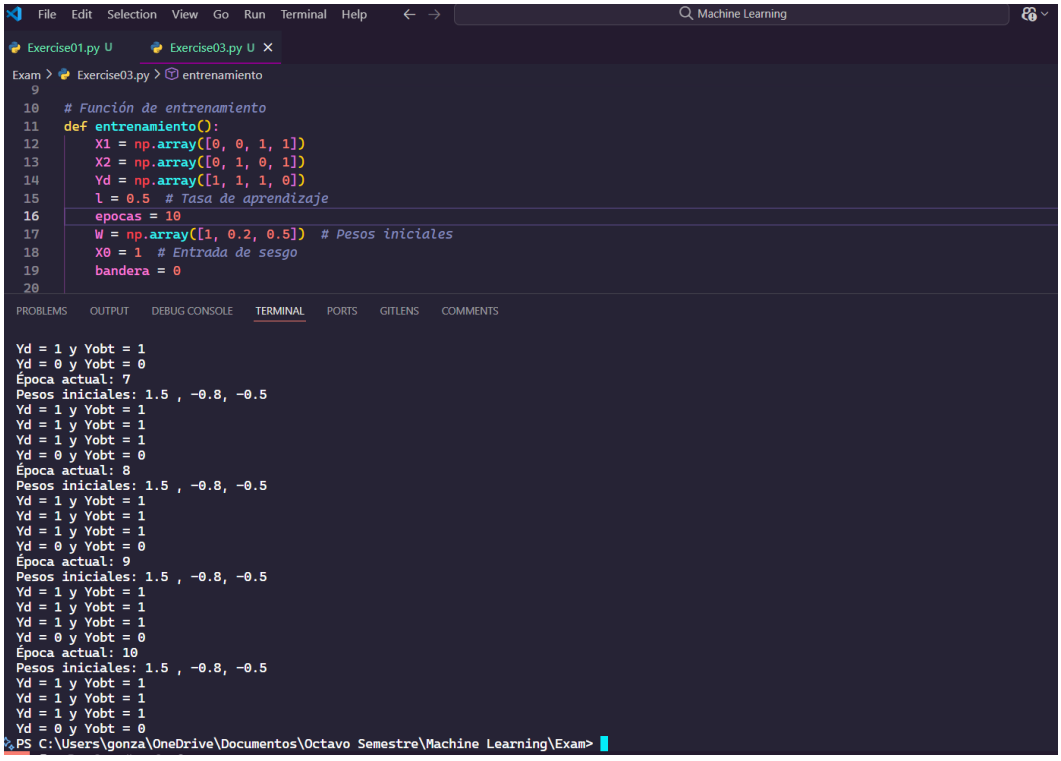
El objetivo principal de las SVM es maximizar la distancia entre los márgenes y los vectores de soporte, lo cual se logra mediante la operación de un producto punto.

3. **Considerando el perceptrón propuesto en clase, implementar el código que permite el entrenamiento de dicha neurona y la solución de alguna tabla de verdad.**



```python
# Función de entrenamiento
def entrenamiento():
    X1 = np.array([0, 0, 1, 1])
    X2 = np.array([0, 1, 0, 1])
    Yd = np.array([1, 1, 1, 0])
    l = 0.5  # Tasa de aprendizaje
    epocas = 10
    W = np.array([1, 0.2, 0.5])  # Pesos iniciales
    X0 = 1  # Entrada de sesgo
    bandera = 0

    for i in range(epocas):
        print(f"Época actual: {i + 1}")
        print(f"Pesos iniciales: {W[0]} , {W[1]}, {W[2]}")

        for j in range(len(X1)):
            # Cálculo de la salida del perceptrón
            z = X0 * W[0] + W[1] * X1[j] + W[2] * X2[j]
            Yobt = activacion(z)

            if Yobt == Yd[j]:
                bandera += 1
                print(f"Yd = {Yd[j]} y Yobt = {Yobt}")
            else:
                # Actualización de los pesos
                W[0] = W[0] - l * (Yobt - Yd[j]) * X0  # Peso asociado al sesgo
                W[1] = W[1] - l * (Yobt - Yd[j]) * X1[j]  # Peso asociado a X1
                W[2] = W[2] - l * (Yobt - Yd[j]) * X2[j]  # Peso asociado a X2
                bandera = 0
                print(f"Yd = {Yd[j]} y Yobt = {Yobt}")
                print(f"Pesos actualizados: {W[0]} , {W[1]}, {W[2]}")

entrenamiento()
```

Resultados en terminal



```
Yd = 1 y Yobt = 1
Yd = 0 y Yobt = 0
Época actual: 7
Pesos iniciales: 1.5 , -0.8, -0.5
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 0 y Yobt = 0
Época actual: 8
Pesos iniciales: 1.5 , -0.8, -0.5
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 0 y Yobt = 0
Época actual: 9
Pesos iniciales: 1.5 , -0.8, -0.5
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 0 y Yobt = 0
Época actual: 10
Pesos iniciales: 1.5 , -0.8, -0.5
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 1 y Yobt = 1
Yd = 0 y Yobt = 0
PS C:\Users\gonza\OneDrive\Documentos\Octavo Semestre\Machine Learning\Exam>
```