Perceptron Taller 2

Trabajo Corte 2 - Computación 2

Nombre: Mariana Rodríguez Pérez

Fecha: Abril 2025

En este taller se implementó desde cero un **perceptrón**, un modelo básico de redes neuronales, con el objetivo de clasificar correctamente un conjunto de datos generado artificialmente.

1. **Generación de datos**: Se crearon dos grupos de datos con formas semicirculares:
   - La **clase 1** corresponde a un semicírculo superior.
   - La **clase 2** es un semicírculo inferior desplazado hacia abajo. A cada punto se le agregó ruido para simular un entorno realista.
2. **Entrenamiento del modelo**:
   - Se inicializaron los pesos (`w`) y el sesgo (`b`) del perceptrón en cero.
   - El algoritmo recorrió los datos durante varias épocas (20), ajustando los pesos cada vez que cometía un error en la predicción, usando la regla del perceptrón:

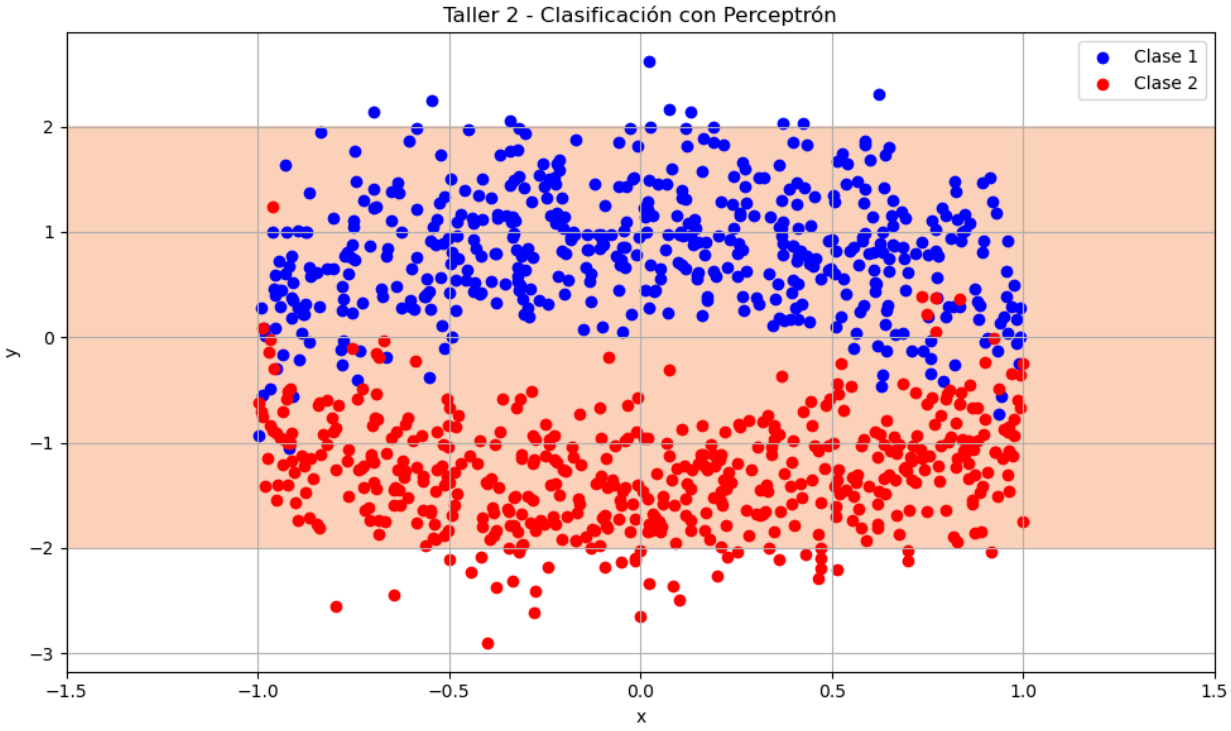
```
w = w + learning_rate * y_i * x_i
b = b + learning_rate * y_i
```

3. **Función de predicción**: Se definió una función para predecir la clase de nuevos puntos, evaluando el signo de la combinación lineal `w·x + b`.
4. **Visualización del resultado**:
   - Se construyó un mapa de calor con `matplotlib` que muestra las regiones del espacio que el modelo clasifica como clase 1 o clase 2.
   - Sobre ese mapa se graficaron los puntos reales de cada clase para comparar la clasificación con los datos originales.

```
import numpy as np
import matplotlib.pyplot as plt
import random

# 1. Generación del dataset (dos clases semicirculares con ruido)
data_1 = []
data_2 = []
random.seed(0)

for i in range(500):
```

```
    # Clase 1: semicirculo superior
    noise = 0.5
    radio = 1
    data1_random_x = (random.random() - 0.5) * radio * 2
    data1_y = (radio**2 - data1_random_x**2)**0.5 + random.gauss(0,
noise)
    data_1.append([data1_random_x, data1_y])

    # Clase 2: semicirculo inferior desplazado
    data2_random_x = (random.random() - 0.5) * radio * 2
    data2_y = -((radio**2 - data2_random_x**2)**0.5 + 0.5 +
random.gauss(0, noise))
    data_2.append([data2_random_x, data2_y])

# Conversión a arrays y etiquetas
X1 = np.array(data_1)
X2 = np.array(data_2)
X = np.vstack((X1, X2))
y = np.hstack((np.ones(len(X1)), -1 * np.ones(len(X2))))

# 2. Inicialización del perceptrón
w = np.zeros(X.shape[1])
b = 0
learning_rate = 0.01
epochs = 20

# 3. Entrenamiento
for epoch in range(epochs):
    for i in range(len(X)):
        xi = X[i]
        yi = y[i]
        if yi * (np.dot(w, xi) + b) <= 0:
            w += learning_rate * yi * xi
            b += learning_rate * yi

# 4. Función de predicción
def predict(x):
    return np.sign(np.dot(w, x) + b)

# 5. Visualización del mapa de calor de predicción
xx, yy = np.meshgrid(np.linspace(-1.5, 1.5, 300), np.linspace(-2, 2,
300))
zz = np.array([predict([x, y]) for x, y in zip(xx.ravel(),
yy.ravel())])
zz = zz.reshape(xx.shape)

plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, zz, alpha=0.4, cmap='RdYlBu')
plt.scatter(X1[:, 0], X1[:, 1], color='blue', label='Clase 1')
plt.scatter(X2[:, 0], X2[:, 1], color='red', label='Clase 2')
```

```
plt.title('Taller 2 - Clasificación con Perceptrón')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Mostrar pesos y sesgo aprendidos
print("Pesos finales:", w)
print("Sesgo:", b)
```




```
Pesos finales: [ 0.00964703 -0.00212854]
Sesgo: -0.019999999999999997
```

Resultado

El perceptrón logró encontrar una frontera de decisión que separa razonablemente bien las dos clases, a pesar del ruido en los datos. Si bien el perceptrón solo puede encontrar **fronteras lineales**, el experimento demuestra su capacidad para resolver problemas de clasificación básicos.