Tarea 7

October 27, 2025

Tarea Perceptrón - Redes Neuronales Artificiales Clasificación de pureza de petróleo usando algoritmo supervisado de Hebb

- *Maria Alejandra Bonilla Diaz - 20251595002*
- *Alvaro Alejandro Zarabanda Gutierrez – 20251595006*
- *Youssef Alejandro Ortiz Vargas – 20251595004*

A partir del análisis de un proceso de destilación de petróleo se observó que determinado producto podría ser clasificado en dos clases de pureza (C1 y C2), mediante la medición de tres variables (x , x , x) que representan algunas de las propiedades fisicoquímicas del petróleo.

[13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate
import warnings
warnings.filterwarnings('ignore')
```

Datos de entrenamiento Conjunto de datos del Anexo con 30 muestras de entrenamiento con 3 características cada una.

[14]:
```
# Conjunto de entrenamiento del Anexo
datos_entrenamiento = np.array([
    [-0.6508, 0.1097, 4.0009, -1.0000],
    [-1.4492, 0.8896, 4.4005, -1.0000],
    [2.0850, 0.6876, 12.0710, -1.0000],
    [0.2626, 1.1476, 7.7985, 1.0000],
    [0.6418, 1.0234, 7.0427, 1.0000],
    [0.2569, 0.6730, 8.3265, -1.0000],
    [1.1155, 0.6043, 7.4446, 1.0000],
    [0.0914, 0.3399, 7.0677, -1.0000],
    [0.0121, 0.5256, 4.6316, 1.0000],
    [-0.0429, 0.4660, 5.4323, 1.0000],
    [0.4340, 0.6870, 8.2287, -1.0000],
    [0.2735, 1.0287, 7.1934, 1.0000],
    [0.4839, 0.4851, 7.4850, -1.0000],
    [0.4089, -0.1267, 5.5019, -1.0000],
    [1.4391, 0.1614, 8.5843, -1.0000],
    [-0.9115, -0.1973, 2.1962, -1.0000],
```

```
    [0.3654, 1.0475, 7.4858, 1.0000],
    [0.2144, 0.7515, 7.1699, 1.0000],
    [0.2013, 1.0014, 6.5489, 1.0000],
    [0.6483, 0.2183, 5.8991, 1.0000],
    [-0.1147, 0.2242, 7.2435, -1.0000],
    [-0.7970, 0.8795, 3.8762, 1.0000],
    [-1.0625, 0.6366, 2.4707, 1.0000],
    [0.5307, 0.1285, 5.6883, 1.0000],
    [-1.2200, 0.7777, 1.7252, 1.0000],
    [0.3957, 0.1076, 5.6623, -1.0000],
    [-0.1013, 0.5989, 7.1812, -1.0000],
    [2.4482, 0.9455, 11.2095, 1.0000],
    [2.0149, 0.6192, 10.9263, -1.0000],
    [0.2012, 0.2611, 5.4631, 1.0000]
])

# Separar características y etiquetas
X_train = datos_entrenamiento[:, :3]  # Primeras 3 columnas: x1, x2, x3
y_train = datos_entrenamiento[:, 3]   # Última columna: clase (1 o -1)

print(f"Muestras de entrenamiento: {len(X_train)}")
print(f"Variables de entrada: {X_train.shape[1]}")
print(f"Clases: {np.unique(y_train)}")

# Mostrar primeras 5 muestras
print("\nPrimeras 5 muestras:")
print("x1\t\tx2\t\tx3\t\tClase")
for i in range(5):
    print(f"{X_train[i,0]:.4f}\t\t{X_train[i,1]:.4f}\t\t{X_train[i,2]:.
 ↪4f}\t\t{int(y_train[i])}")
```

```
Muestras de entrenamiento: 30
Variables de entrada: 3
Clases: [-1.  1.]

Primeras 5 muestras:
x1              x2              x3              Clase
-0.6508         0.1097          4.0009          -1
-1.4492         0.8896          4.4005          -1
2.0850          0.6876          12.0710         -1
0.2626          1.1476          7.7985          1
0.6418          1.0234          7.0427          1
```

Implementación del Perceptrón con algoritmo supervisado de Hebb

El algoritmo supervisado de Hebb actualiza los pesos según la regla: **w(n+1) = w(n) + * (d - y) * x**

Donde: - = tasa de aprendizaje (0.01) - d = salida deseada - y = salida actual del perceptrón - x

= vector de entrada (incluyendo bias)

[15]:
```
class PerceptronClasificador:
    def __init__(self, factor_aprendizaje=0.01, seed_valor=None):
        self.factor_aprendizaje = factor_aprendizaje
        self.vector_pesos = None
        self.pesos_origen = None
        self.iteraciones_totales = 0
        self.seed_valor = seed_valor
        self.bitacora_aprendizaje = []
        self.ha_convergido = False

    def funcion_signo(self, valor_entrada):
        """Función de activación signo bipolar: retorna 1 si x >= 0, caso␣
↪contrario -1"""
        return np.where(valor_entrada >= 0, 1, -1)

    def proceso_entrenamiento(self, caracteristicas_X, etiquetas_y,␣
↪max_iteraciones=5000, mostrar_progreso=False):
        """
        Ejecuta el entrenamiento del perceptrón empleando la regla de Hebb␣
↪supervisada
        Regla de actualización: pesos_nuevos = pesos_antiguos +   * error *␣
↪entrada
        """
        # Configuración de semilla aleatoria si es proporcionada
        if self.seed_valor is not None:
            np.random.seed(self.seed_valor)

        total_muestras, total_atributos = caracteristicas_X.shape

        # Añadir término bias (x0 = -1) como primera columna
        X_expandido = np.c_[-np.ones((total_muestras, 1)), caracteristicas_X]

        # Inicialización aleatoria de pesos en rango [0, 1]
        self.vector_pesos = np.random.random(total_atributos + 1)
        self.pesos_origen = self.vector_pesos.copy()

        if mostrar_progreso:
            print(f"Configuración inicial de pesos: {self.vector_pesos}")

        # Ciclo de entrenamiento por iteraciones
        for iteracion_actual in range(max_iteraciones):
            errores_acumulados = 0
            registro_errores = []

            for idx_muestra in range(total_muestras):
```

```
                # Computar suma ponderada (net input)
                suma_ponderada = np.dot(X_expandido[idx_muestra], self.
↪vector_pesos)

                # Aplicar función de activación
                salida_predicha = self.funcion_signo(suma_ponderada)

                # Calcular discrepancia
                discrepancia = etiquetas_y[idx_muestra] - salida_predicha
                registro_errores.append(abs(discrepancia))

                # Actualizar pesos según regla de Hebb supervisada
                if discrepancia != 0:
                    self.vector_pesos += self.factor_aprendizaje * discrepancia␣
↪* X_expandido[idx_muestra]
                    errores_acumulados += abs(discrepancia)

            # Registrar estadísticas de la iteración
            exactitud_iteracion = 1 - (errores_acumulados / (2 *␣
↪total_muestras))
            self.bitacora_aprendizaje.append({
                'iteracion': iteracion_actual + 1,
                'errores_totales': errores_acumulados,
                'exactitud': exactitud_iteracion,
                'pesos_estado': self.vector_pesos.copy()
            })

            if mostrar_progreso and (iteracion_actual + 1) % 20 == 0:
                print(f"Iteración {iteracion_actual + 1}: Errores =␣
↪{errores_acumulados}, Exactitud = {exactitud_iteracion:.3f}")

            # Condición de terminación: ausencia de errores
            if errores_acumulados == 0:
                self.iteraciones_totales = iteracion_actual + 1
                self.ha_convergido = True
                if mostrar_progreso:
                    print(f"¡Convergencia lograda en iteración␣
↪{iteracion_actual + 1}!")
                break
        else:
            self.iteraciones_totales = max_iteraciones
            self.ha_convergido = False
            if mostrar_progreso:
                print(f"Alerta: Límite de iteraciones alcanzado␣
↪({max_iteraciones})")
```

```
        return self

    def realizar_prediccion(self, caracteristicas_nuevas):
        """Generar predicciones para nuevos patrones de entrada"""
        # Incorporar bias
        X_expandido_nuevas = np.c_[-np.ones((caracteristicas_nuevas.shape[0],␣
↪1)), caracteristicas_nuevas]
        suma_ponderada = np.dot(X_expandido_nuevas, self.vector_pesos)
        return self.funcion_signo(suma_ponderada)

    def extraer_pesos_iniciales(self):
        """Obtiene pesos iniciales en formato de diccionario estructurado"""
        return {
            'peso_bias': self.pesos_origen[0],
            'peso_x1': self.pesos_origen[1],
            'peso_x2': self.pesos_origen[2],
            'peso_x3': self.pesos_origen[3]
        }

    def extraer_pesos_finales(self):
        """Obtiene pesos finales en formato de diccionario estructurado"""
        return {
            'peso_bias': self.vector_pesos[0],
            'peso_x1': self.vector_pesos[1],
            'peso_x2': self.vector_pesos[2],
            'peso_x3': self.vector_pesos[3]
        }
```

1 Ejecutar 5 entrenamientos con diferentes inicializaciones

[16]:
```
# Ejecutar serie de 5 entrenamientos con configuraciones aleatorias distintas
registro_experimentos = []
conjunto_clasificadores = []

print("EJECUCIÓN DE SERIE DE 5 EXPERIMENTOS CON PERCEPTRÓN")
print("=" * 70)

for num_experimento in range(5):
    print(f"\n--- EXPERIMENTO E{num_experimento+1} ---")

    # Instanciar clasificador con semilla única para cada experimento
    clasificador_actual = PerceptronClasificador(factor_aprendizaje=0.01,␣
↪seed_valor=num_experimento*202204206)

    # Ejecutar proceso de entrenamiento
```

```
    clasificador_actual.proceso_entrenamiento(X_train, y_train,␣
 ↪max_iteraciones=1000, mostrar_progreso=True)

    # Almacenar clasificador entrenado
    conjunto_clasificadores.append(clasificador_actual)

    # Compilar información del experimento
    info_experimento = {
        'Experimento': f"E{num_experimento+1}",
        'peso_bias_inicial': clasificador_actual.pesos_origen[0],
        'peso_x1_inicial': clasificador_actual.pesos_origen[1],
        'peso_x2_inicial': clasificador_actual.pesos_origen[2],
        'peso_x3_inicial': clasificador_actual.pesos_origen[3],
        'peso_bias_final': clasificador_actual.vector_pesos[0],
        'peso_x1_final': clasificador_actual.vector_pesos[1],
        'peso_x2_final': clasificador_actual.vector_pesos[2],
        'peso_x3_final': clasificador_actual.vector_pesos[3],
        'Iteraciones': clasificador_actual.iteraciones_totales
    }

    registro_experimentos.append(info_experimento)

    print(f"Configuración inicial: bias={clasificador_actual.pesos_origen[0]:.
 ↪4f}, x1={clasificador_actual.pesos_origen[1]:.4f}, x2={clasificador_actual.
 ↪pesos_origen[2]:.4f}, x3={clasificador_actual.pesos_origen[3]:.4f}")
    print(f"Configuración final:   bias={clasificador_actual.vector_pesos[0]:.
 ↪4f}, x1={clasificador_actual.vector_pesos[1]:.4f}, x2={clasificador_actual.
 ↪vector_pesos[2]:.4f}, x3={clasificador_actual.vector_pesos[3]:.4f}")
    print(f"Iteraciones requeridas: {clasificador_actual.iteraciones_totales}")

print(f"\n{'=' * 70}")
print("FINALIZACIÓN DE TODOS LOS EXPERIMENTOS")
```

```
EJECUCIÓN DE SERIE DE 5 EXPERIMENTOS CON PERCEPTRÓN
======================================================================

--- EXPERIMENTO E1 ---
Configuración inicial de pesos: [0.5488135  0.71518937 0.60276338 0.54488318]
Iteración 20: Errores = 18.0, Exactitud = 0.700
Iteración 40: Errores = 22.0, Exactitud = 0.633
Iteración 60: Errores = 14.0, Exactitud = 0.767
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 20.0, Exactitud = 0.667
Iteración 120: Errores = 20.0, Exactitud = 0.667
Iteración 140: Errores = 10.0, Exactitud = 0.833
Iteración 160: Errores = 10.0, Exactitud = 0.833
Iteración 180: Errores = 14.0, Exactitud = 0.767
```

```
Iteración 200: Errores = 6.0, Exactitud = 0.900
Iteración 220: Errores = 2.0, Exactitud = 0.967
Iteración 240: Errores = 16.0, Exactitud = 0.733
Iteración 260: Errores = 12.0, Exactitud = 0.800
Iteración 280: Errores = 12.0, Exactitud = 0.800
Iteración 300: Errores = 12.0, Exactitud = 0.800
Iteración 320: Errores = 16.0, Exactitud = 0.733
Iteración 340: Errores = 12.0, Exactitud = 0.800
Iteración 360: Errores = 4.0, Exactitud = 0.933
Iteración 380: Errores = 12.0, Exactitud = 0.800
¡Convergencia lograda en iteración 389!
Configuración inicial: bias=0.5488, x1=0.7152, x2=0.6028, x3=0.5449
Configuración final:   bias=-3.0312, x1=1.5207, x2=2.4595, x3=-0.7255
Iteraciones requeridas: 389

--- EXPERIMENTO E2 ---
Configuración inicial de pesos: [0.24780909 0.64451682 0.87244454 0.3331428 ]
Iteración 20: Errores = 18.0, Exactitud = 0.700
Iteración 40: Errores = 18.0, Exactitud = 0.700
Iteración 60: Errores = 14.0, Exactitud = 0.767
Iteración 80: Errores = 18.0, Exactitud = 0.700
Iteración 100: Errores = 14.0, Exactitud = 0.767
Iteración 120: Errores = 12.0, Exactitud = 0.800
Iteración 140: Errores = 14.0, Exactitud = 0.767
Iteración 160: Errores = 10.0, Exactitud = 0.833
Iteración 180: Errores = 6.0, Exactitud = 0.900
Iteración 200: Errores = 12.0, Exactitud = 0.800
Iteración 220: Errores = 12.0, Exactitud = 0.800
Iteración 240: Errores = 12.0, Exactitud = 0.800
Iteración 260: Errores = 12.0, Exactitud = 0.800
Iteración 280: Errores = 6.0, Exactitud = 0.900
Iteración 300: Errores = 2.0, Exactitud = 0.967
Iteración 320: Errores = 12.0, Exactitud = 0.800
Iteración 340: Errores = 0, Exactitud = 1.000
¡Convergencia lograda en iteración 340!
Configuración inicial: bias=0.2478, x1=0.6445, x2=0.8724, x3=0.3331
Configuración final:   bias=-2.9122, x1=1.4346, x2=2.3957, x3=-0.6790
Iteraciones requeridas: 340

--- EXPERIMENTO E3 ---
Configuración inicial de pesos: [0.24611793 0.1147657  0.7727844  0.68787168]
Iteración 20: Errores = 20.0, Exactitud = 0.667
Iteración 40: Errores = 18.0, Exactitud = 0.700
Iteración 60: Errores = 18.0, Exactitud = 0.700
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 14.0, Exactitud = 0.767
Iteración 120: Errores = 14.0, Exactitud = 0.767
Iteración 300: Errores = 2.0, Exactitud = 0.967
```

```
Iteración 320: Errores = 12.0, Exactitud = 0.800
Iteración 340: Errores = 0, Exactitud = 1.000
¡Convergencia lograda en iteración 340!
Configuración inicial: bias=0.2478, x1=0.6445, x2=0.8724, x3=0.3331
Configuración final:   bias=-2.9122, x1=1.4346, x2=2.3957, x3=-0.6790
Iteraciones requeridas: 340

--- EXPERIMENTO E3 ---
Configuración inicial de pesos: [0.24611793 0.1147657  0.7727844  0.68787168]
Iteración 20: Errores = 20.0, Exactitud = 0.667
Iteración 40: Errores = 18.0, Exactitud = 0.700
Iteración 60: Errores = 18.0, Exactitud = 0.700
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 14.0, Exactitud = 0.767
Iteración 120: Errores = 14.0, Exactitud = 0.767
Iteración 140: Errores = 12.0, Exactitud = 0.800
Iteración 160: Errores = 10.0, Exactitud = 0.833
Iteración 180: Errores = 6.0, Exactitud = 0.900
Iteración 200: Errores = 6.0, Exactitud = 0.900
Iteración 220: Errores = 6.0, Exactitud = 0.900
Iteración 240: Errores = 12.0, Exactitud = 0.800
Iteración 260: Errores = 12.0, Exactitud = 0.800
Iteración 280: Errores = 4.0, Exactitud = 0.933
Iteración 300: Errores = 6.0, Exactitud = 0.900
Iteración 320: Errores = 6.0, Exactitud = 0.900
Iteración 340: Errores = 4.0, Exactitud = 0.933
Iteración 360: Errores = 8.0, Exactitud = 0.867
Iteración 380: Errores = 4.0, Exactitud = 0.933
Iteración 400: Errores = 12.0, Exactitud = 0.800
Iteración 420: Errores = 10.0, Exactitud = 0.833
Iteración 440: Errores = 0, Exactitud = 1.000
¡Convergencia lograda en iteración 440!
Configuración inicial: bias=0.2461, x1=0.1148, x2=0.7728, x3=0.6879
Configuración final:   bias=-3.1539, x1=1.6037, x2=2.5744, x3=-0.7554
Iteraciones requeridas: 440

--- EXPERIMENTO E4 ---
Configuración inicial de pesos: [0.23662799 0.50156557 0.04607647 0.84609767]
Iteración 20: Errores = 16.0, Exactitud = 0.733
Iteración 40: Errores = 18.0, Exactitud = 0.700
Iteración 60: Errores = 14.0, Exactitud = 0.767
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 16.0, Exactitud = 0.733
Iteración 120: Errores = 16.0, Exactitud = 0.733
Iteración 140: Errores = 10.0, Exactitud = 0.833
Iteración 160: Errores = 10.0, Exactitud = 0.833
Iteración 180: Errores = 14.0, Exactitud = 0.767
Iteración 200: Errores = 10.0, Exactitud = 0.833
```

```
Iteración 220: Errores = 12.0, Exactitud = 0.800
Iteración 240: Errores = 12.0, Exactitud = 0.800
Iteración 260: Errores = 6.0, Exactitud = 0.900
Iteración 280: Errores = 12.0, Exactitud = 0.800
Iteración 300: Errores = 4.0, Exactitud = 0.933
Iteración 320: Errores = 4.0, Exactitud = 0.933
Iteración 340: Errores = 4.0, Exactitud = 0.933
¡Convergencia lograda en iteración 357!
Configuración inicial: bias=0.2366, x1=0.5016, x2=0.0461, x3=0.8461
Configuración final:   bias=-2.9834, x1=1.4616, x2=2.4393, x3=-0.7145
Iteraciones requeridas: 357

--- EXPERIMENTO E5 ---
Configuración inicial de pesos: [0.99557752 0.08719167 0.38904648 0.63798315]
Iteración 20: Errores = 32.0, Exactitud = 0.467
Iteración 40: Errores = 22.0, Exactitud = 0.633
Iteración 60: Errores = 18.0, Exactitud = 0.700
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 16.0, Exactitud = 0.733
Iteración 120: Errores = 16.0, Exactitud = 0.733
Iteración 140: Errores = 14.0, Exactitud = 0.767
Iteración 160: Errores = 14.0, Exactitud = 0.767
Iteración 180: Errores = 10.0, Exactitud = 0.833
Iteración 200: Errores = 6.0, Exactitud = 0.900
Iteración 220: Errores = 8.0, Exactitud = 0.867
Iteración 140: Errores = 12.0, Exactitud = 0.800
Iteración 160: Errores = 10.0, Exactitud = 0.833
Iteración 180: Errores = 6.0, Exactitud = 0.900
Iteración 200: Errores = 6.0, Exactitud = 0.900
Iteración 220: Errores = 6.0, Exactitud = 0.900
Iteración 240: Errores = 12.0, Exactitud = 0.800
Iteración 260: Errores = 12.0, Exactitud = 0.800
Iteración 280: Errores = 4.0, Exactitud = 0.933
Iteración 300: Errores = 6.0, Exactitud = 0.900
Iteración 320: Errores = 6.0, Exactitud = 0.900
Iteración 340: Errores = 4.0, Exactitud = 0.933
Iteración 360: Errores = 8.0, Exactitud = 0.867
Iteración 380: Errores = 4.0, Exactitud = 0.933
Iteración 400: Errores = 12.0, Exactitud = 0.800
Iteración 420: Errores = 10.0, Exactitud = 0.833
Iteración 440: Errores = 0, Exactitud = 1.000
¡Convergencia lograda en iteración 440!
Configuración inicial: bias=0.2461, x1=0.1148, x2=0.7728, x3=0.6879
Configuración final:   bias=-3.1539, x1=1.6037, x2=2.5744, x3=-0.7554
Iteraciones requeridas: 440

--- EXPERIMENTO E4 ---
Configuración inicial de pesos: [0.23662799 0.50156557 0.04607647 0.84609767]
```

```
Iteración 20: Errores = 16.0, Exactitud = 0.733
Iteración 40: Errores = 18.0, Exactitud = 0.700
Iteración 60: Errores = 14.0, Exactitud = 0.767
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 16.0, Exactitud = 0.733
Iteración 120: Errores = 16.0, Exactitud = 0.733
Iteración 140: Errores = 10.0, Exactitud = 0.833
Iteración 160: Errores = 10.0, Exactitud = 0.833
Iteración 180: Errores = 14.0, Exactitud = 0.767
Iteración 200: Errores = 10.0, Exactitud = 0.833
Iteración 220: Errores = 12.0, Exactitud = 0.800
Iteración 240: Errores = 12.0, Exactitud = 0.800
Iteración 260: Errores = 6.0, Exactitud = 0.900
Iteración 280: Errores = 12.0, Exactitud = 0.800
Iteración 300: Errores = 4.0, Exactitud = 0.933
Iteración 320: Errores = 4.0, Exactitud = 0.933
Iteración 340: Errores = 4.0, Exactitud = 0.933
¡Convergencia lograda en iteración 357!
Configuración inicial: bias=0.2366, x1=0.5016, x2=0.0461, x3=0.8461
Configuración final:   bias=-2.9834, x1=1.4616, x2=2.4393, x3=-0.7145
Iteraciones requeridas: 357

--- EXPERIMENTO E5 ---
Configuración inicial de pesos: [0.99557752 0.08719167 0.38904648 0.63798315]
Iteración 20: Errores = 32.0, Exactitud = 0.467
Iteración 40: Errores = 22.0, Exactitud = 0.633
Iteración 60: Errores = 18.0, Exactitud = 0.700
Iteración 80: Errores = 14.0, Exactitud = 0.767
Iteración 100: Errores = 16.0, Exactitud = 0.733
Iteración 120: Errores = 16.0, Exactitud = 0.733
Iteración 140: Errores = 14.0, Exactitud = 0.767
Iteración 160: Errores = 14.0, Exactitud = 0.767
Iteración 180: Errores = 10.0, Exactitud = 0.833
Iteración 200: Errores = 6.0, Exactitud = 0.900
Iteración 220: Errores = 8.0, Exactitud = 0.867
Iteración 240: Errores = 6.0, Exactitud = 0.900
Iteración 260: Errores = 12.0, Exactitud = 0.800
Iteración 280: Errores = 12.0, Exactitud = 0.800
Iteración 300: Errores = 12.0, Exactitud = 0.800
Iteración 320: Errores = 12.0, Exactitud = 0.800
Iteración 340: Errores = 12.0, Exactitud = 0.800
Iteración 360: Errores = 4.0, Exactitud = 0.933
Iteración 380: Errores = 12.0, Exactitud = 0.800
Iteración 400: Errores = 6.0, Exactitud = 0.900
Iteración 420: Errores = 4.0, Exactitud = 0.933
Iteración 440: Errores = 0, Exactitud = 1.000
¡Convergencia lograda en iteración 440!
Configuración inicial: bias=0.9956, x1=0.0872, x2=0.3890, x3=0.6380
```

```
Configuración final:   bias=-3.1044, x1=1.5736, x2=2.4833, x3=-0.7392
Iteraciones requeridas: 440

======================================================================
FINALIZACIÓN DE TODOS LOS EXPERIMENTOS
Iteración 240: Errores = 6.0, Exactitud = 0.900
Iteración 260: Errores = 12.0, Exactitud = 0.800
Iteración 280: Errores = 12.0, Exactitud = 0.800
Iteración 300: Errores = 12.0, Exactitud = 0.800
Iteración 320: Errores = 12.0, Exactitud = 0.800
Iteración 340: Errores = 12.0, Exactitud = 0.800
Iteración 360: Errores = 4.0, Exactitud = 0.933
Iteración 380: Errores = 12.0, Exactitud = 0.800
Iteración 400: Errores = 6.0, Exactitud = 0.900
Iteración 420: Errores = 4.0, Exactitud = 0.933
Iteración 440: Errores = 0, Exactitud = 1.000
¡Convergencia lograda en iteración 440!
Configuración inicial: bias=0.9956, x1=0.0872, x2=0.3890, x3=0.6380
Configuración final:   bias=-3.1044, x1=1.5736, x2=2.4833, x3=-0.7392
Iteraciones requeridas: 440

======================================================================
FINALIZACIÓN DE TODOS LOS EXPERIMENTOS
```

2 Tabla de resultados del entrenamiento

[17]:
```python
# Generar tabla estructurada con resultados experimentales
dataframe_experimentos = pd.DataFrame(registro_experimentos)

# Formatear datos para presentación tabular
matriz_resultados = []
for _, fila_datos in dataframe_experimentos.iterrows():
    matriz_resultados.append([
        fila_datos['Experimento'],
        f"{fila_datos['peso_bias_inicial']:.4f}",␣
 ↪f"{fila_datos['peso_x1_inicial']:.4f}",
        f"{fila_datos['peso_x2_inicial']:.4f}",␣
 ↪f"{fila_datos['peso_x3_inicial']:.4f}",
        f"{fila_datos['peso_bias_final']:.4f}", f"{fila_datos['peso_x1_final']:.
 ↪4f}",
        f"{fila_datos['peso_x2_final']:.4f}", f"{fila_datos['peso_x3_final']:.
 ↪4f}",
        fila_datos['Iteraciones']
    ])

encabezados_tabla = ['Experimento',
           'bias inicial', 'x  inicial', 'x  inicial', 'x  inicial',
```

```
            'bias final', 'x  final', 'x  final', 'x  final',
            'Iteraciones']

print("MATRIZ DE RESULTADOS EXPERIMENTALES")
print("=" * 130)
print(tabulate(matriz_resultados, headers=encabezados_tabla, tablefmt='grid'))

# Estadísticas descriptivas
lista_iteraciones = [exp['Iteraciones'] for exp in registro_experimentos]
print(f"Iteraciones mínimas: {min(lista_iteraciones)}")
print(f"Iteraciones máximas: {max(lista_iteraciones)}")
print(f"Promedio de iteraciones: {np.mean(lista_iteraciones):.1f}")
```

```
MATRIZ DE RESULTADOS EXPERIMENTALES
==================================================================================
================================================
+--------------+---------------+-------------+-------------+-------------+-
------------+-----------+-----------+-----------+---------------+
| Experimento  |   bias inicial |   x  inicial |   x  inicial |   x  inicial |
bias final |   x  final |   x  final |   x  final |   Iteraciones |
+==============+===============+=============+=============+=============+=
============+===========+===========+===========+===============+
| E1           |         0.5488 |       0.7152 |       0.6028 |       0.5449 |
-3.0312 |     1.5207 |     2.4595 |    -0.7255 |           389 |
+--------------+---------------+-------------+-------------+-------------+-
------------+-----------+-----------+-----------+---------------+
| E2           |         0.2478 |       0.6445 |       0.8724 |       0.3331 |
-2.9122 |     1.4346 |     2.3957 |    -0.679  |           340 |
+--------------+---------------+-------------+-------------+-------------+-
------------+-----------+-----------+-----------+---------------+
| E3           |         0.2461 |       0.1148 |       0.7728 |       0.6879 |
-3.1539 |     1.6037 |     2.5744 |    -0.7554 |           440 |
+--------------+---------------+-------------+-------------+-------------+-
------------+-----------+-----------+-----------+---------------+
| E4           |         0.2366 |       0.5016 |       0.0461 |       0.8461 |
-2.9834 |     1.4616 |     2.4393 |    -0.7145 |           357 |
+--------------+---------------+-------------+-------------+-------------+-
------------+-----------+-----------+-----------+---------------+
| E5           |         0.9956 |       0.0872 |       0.389  |       0.638  |
-3.1044 |     1.5736 |     2.4833 |    -0.7392 |           440 |
+--------------+---------------+-------------+-------------+-------------+-
------------+-----------+-----------+-----------+---------------+
Iteraciones mínimas: 340
Iteraciones máximas: 440
Promedio de iteraciones: 393.2
```

3 Clasificación de nuevas muestras

Utilizar cada modelo entrenado para clasificar las 10 nuevas muestras de petróleo.

[18]:
```
patrones_evaluacion = np.array([
    [-0.3565, 0.0620, 5.9891],
    [-0.7842, 1.1267, 5.5912],
    [0.3012, 0.5611, 5.8234],
    [0.7757, 1.0648, 8.0677],
    [0.1570, 0.8028, 6.3040],
    [-0.7014, 1.0316, 3.6005],
    [0.3748, 0.1536, 6.1537],
    [-0.6920, 0.9404, 4.4058],
    [-1.3970, 0.7141, 4.9263],
    [-1.8842, -0.2805, 1.2548]
])

print("PATRONES DE EVALUACIÓN SIN ETIQUETAS:")
print("=" * 60)
for idx_patron, patron in enumerate(patrones_evaluacion):
    print(f"Patrón {idx_patron+1}: x ={patron[0]:.4f}, x ={patron[1]:.4f},␣
 ↪x ={patron[2]:.4f}")

# Aplicar cada clasificador entrenado a los patrones de evaluación
conjunto_predicciones = []
for idx_clasificador, clasificador in enumerate(conjunto_clasificadores):
    etiquetas_predichas = clasificador.realizar_prediccion(patrones_evaluacion)
    conjunto_predicciones.append(etiquetas_predichas)
    print(f"\nEtiquetas E{idx_clasificador+1}: {etiquetas_predichas}")

# Construir matriz de resultados de clasificación
matriz_clasificacion = []
for idx_patron in range(len(patrones_evaluacion)):
    fila_patron = [idx_patron+1]
    fila_patron.extend([f"{patrones_evaluacion[idx_patron, j]:.4f}" for j in␣
 ↪range(3)])
    fila_patron.extend([int(conjunto_predicciones[j][idx_patron]) for j in␣
 ↪range(5)])
    matriz_clasificacion.append(fila_patron)

encabezados_clasificacion = ['Patrón', 'x ', 'x ', 'x ',
                             'E1', 'E2', 'E3', 'E4', 'E5']

print(f"\n{'='*90}")
print("RESULTADOS CLASIFICACION DE MUESTRAS")
print("="*90)
```

```
print(tabulate(matriz_clasificacion, headers=encabezados_clasificacion,␣
 ↪tablefmt='grid'))
```

```
PATRONES DE EVALUACIÓN SIN ETIQUETAS:
============================================================
Patrón 1: x =-0.3565, x =0.0620, x =5.9891
Patrón 2: x =-0.7842, x =1.1267, x =5.5912
Patrón 3: x =0.3012, x =0.5611, x =5.8234
Patrón 4: x =0.7757, x =1.0648, x =8.0677
Patrón 5: x =0.1570, x =0.8028, x =6.3040
Patrón 6: x =-0.7014, x =1.0316, x =3.6005
Patrón 7: x =0.3748, x =0.1536, x =6.1537
Patrón 8: x =-0.6920, x =0.9404, x =4.4058
Patrón 9: x =-1.3970, x =0.7141, x =4.9263
Patrón 10: x =-1.8842, x =-0.2805, x =1.2548

Etiquetas E1: [-1  1  1  1  1  1 -1  1 -1 -1]

Etiquetas E2: [-1  1  1  1  1  1 -1  1 -1 -1]

Etiquetas E3: [-1  1  1  1  1  1 -1  1 -1 -1]

Etiquetas E4: [-1  1  1  1  1  1 -1  1 -1 -1]

Etiquetas E5: [-1  1  1  1  1  1 -1  1 -1 -1]

================================================================================
==========
RESULTADOS CLASIFICACION DE MUESTRAS
================================================================================
==========
+----------+---------+---------+--------+------+------+------+------+------+
|   Patrón |       x |       x |      x |   E1 |   E2 |   E3 |   E4 |   E5 |
+==========+=========+=========+========+======+======+======+======+======+
|        1 | -0.3565 |  0.062  | 5.9891 |   -1 |   -1 |   -1 |   -1 |   -1 |
+----------+---------+---------+--------+------+------+------+------+------+
|        2 | -0.7842 |  1.1267 | 5.5912 |    1 |    1 |    1 |    1 |    1 |
+----------+---------+---------+--------+------+------+------+------+------+
|        3 |  0.3012 |  0.5611 | 5.8234 |    1 |    1 |    1 |    1 |    1 |
+----------+---------+---------+--------+------+------+------+------+------+
|        4 |  0.7757 |  1.0648 | 8.0677 |    1 |    1 |    1 |    1 |    1 |
+----------+---------+---------+--------+------+------+------+------+------+
|        5 |  0.157  |  0.8028 | 6.304  |    1 |    1 |    1 |    1 |    1 |
+----------+---------+---------+--------+------+------+------+------+------+
|        6 | -0.7014 |  1.0316 | 3.6005 |    1 |    1 |    1 |    1 |    1 |
+----------+---------+---------+--------+------+------+------+------+------+
|        7 |  0.3748 |  0.1536 | 6.1537 |   -1 |   -1 |   -1 |   -1 |   -1 |
+----------+---------+---------+--------+------+------+------+------+------+
```

8	-0.692	0.9404	4.4058	1	1	1	1	1
9	-1.397	0.7141	4.9263	-1	-1	-1	-1	-1
10	-1.8842	-0.2805	1.2548	-1	-1	-1	-1	-1

4 Análisis de la Variabilidad en el Número de Épocas

La diversidad en el número de iteraciones observada en cada experimento se debe a múltiples factores intrínsecos del algoritmo:

4.1 Factores Determinantes

4.1.1 Configuración Estocástica Inicial

Cada experimento inicia con vectores de pesos generados aleatoriamente en el intervalo $[0, 1]$, creando puntos de partida distintos en el espacio de parámetros del clasificador.

4.1.2 Proximidad a la Solución Óptima

La velocidad de convergencia está directamente relacionada con la distancia euclidiana entre la configuración inicial y cualquier hiperplano separador válido.

4.1.3 Topología del Espacio de Búsqueda

El algoritmo navega por el espacio de pesos buscando un hiperplano que separe linealmente las clases y la trayectoria específica depende del punto inicial y puede requerir diferentes cantidades de ajustes según la regla de actualización de Hebb

4.1.4 Dinámicas de Actualización Secuencial

Aunque el orden de presentación de patrones es constante, las actualizaciones de pesos crean estados intermedios únicos que influyen en iteraciones posteriores.

5 Principal limitación del Perceptrón

Análisis de la limitación fundamental del perceptrón en problemas de clasificación.

Limitación: Dependencia de separabilidad lineal

La **restricción fundamental del perceptrón** radica en su capacidad exclusiva para resolver problemas que exhiben **separabilidad lineal**:

Un conjunto de datos presenta separabilidad lineal cuando existe al menos un hiperplano (recta en 2D, plano en 3D, hiperplano en n-D) capaz de dividir perfectamente las clases sin errores de clasificación.

Ejemplos:

1. **Problemas intratables**: • Función XOR (ejemplo paradigmático no linealmente separable) • Datasets con clases entrelazadas o superpuestas • Patrones que requieren fronteras de decisión curvilíneas
2. **Restricciones geométricas**: • Fronteras de decisión exclusivamente Rectilineas • Incapacidad para modelar relaciones no lineales entre atributos • Imposibilidad de manejar regiones de clase múltiples y disjuntas
3. **Teorema de convergencia del perceptrón**:
   - Datos linealmente separables → Convergencia Garantizada
   - Datos no linealmente separables → Oscilación Infinita (sin convergencia)