CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS (CUCEI)

DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Carrera: Ingeniería en Computación

Nombre Materia: Seminario de Solución de Problemas de IA ll

Profesor: Valdez López Julio Esteban

SECCIÓN: D02

Nombre alumno: López Arellano Ricardo David

CODIGO: 217136143

Practica 3

Fecha de entrega: 02/10/2023

CÓDIGO :

```
import numpy as np
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from sklearn.metrics import accuracy_score, mean_squared_error

class Adaline:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        self.weights = np.random.rand(X.shape[1])
        self.bias = np.random.rand()

        for _ in range(self.epochs):
            output = self.predict(X)
            error = y - output
            self.weights += self.learning_rate * X.T.dot(error)
            self.bias += self.learning_rate * error.sum()

    def predict(self, X):
        z = np.dot(X, self.weights) + self.bias
        return z

def generate_classification_data():
    np.random.seed(0)
    X = np.random.rand(100, 2)
    y = np.where(X[:, 0] + X[:, 1] > 1, 1, -1)
    return X, y

def generate_regression_data():
    np.random.seed(0)
    X = np.random.rand(50, 1) * 10
    y = 2 * X + np.random.randn(50, 1)
    return X, y

def plot_decision_boundary(X, y, adaline):
    fig, ax = plt.subplots(figsize=(6, 6))
    ax.scatter(X[y == 1][:, 0], X[y == 1][:, 1], label='Clase 1',
marker='o')
    ax.scatter(X[y == -1][:, 0], X[y == -1][:, 1], label='Clase -1',
marker='x')

    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
```

```
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))
    Z = adaline.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3)
    ax.set_xlabel('Característica 1')
    ax.set_ylabel('Característica 2')
    ax.legend()
    return fig

def plot_regression_line(X, y, adaline):
    fig, ax = plt.subplots(figsize=(6, 6))
    ax.scatter(X, y, label='Datos de Regresión')

    x_min, x_max = X.min() - 1, X.max() + 1
    xx = np.arange(x_min, x_max, 0.01).reshape(-1, 1)
    yy = adaline.predict(xx)

    ax.plot(xx, yy, color='red', label='Línea de Regresión')
    ax.set_xlabel('Característica')
    ax.set_ylabel('Etiqueta')
    ax.legend()
    return fig

def train_and_plot_classification(adaline, X, y, canvas):
    adaline.fit(X, y)
    fig = plot_decision_boundary(X, y, adaline)
    canvas.figure = fig
    canvas.draw()

def train_and_plot_regression(adaline, X, y, canvas):
    adaline.fit(X, y)
    fig = plot_regression_line(X, y, adaline)
    canvas.figure = fig
    canvas.draw()

def main():
    root = tk.Tk()
    root.title("Adaline Classifier and Regressor")

    classification_frame = tk.Frame(root)
    classification_frame.pack(side=tk.LEFT, padx=20)

    regression_frame = tk.Frame(root)
    regression_frame.pack(side=tk.RIGHT, padx=20)

    X_classification, y_classification = generate_classification_data()
```

```
    X_regression, y_regression = generate_regression_data()

    adaline_classification = Adaline(learning_rate=0.01, epochs=100)
    adaline_regression = Adaline(learning_rate=0.01, epochs=100)

    canvas_classification = FigureCanvasTkAgg(plt.figure(figsize=(6, 6)),
master=classification_frame)
    canvas_classification.get_tk_widget().pack()

    canvas_regression = FigureCanvasTkAgg(plt.figure(figsize=(6, 6)),
master=regression_frame)
    canvas_regression.get_tk_widget().pack()

    train_classification_button = tk.Button(classification_frame,
text="Entrenar Clasificación", command=lambda:
train_and_plot_classification(adaline_classification, X_classification,
y_classification, canvas_classification))
    train_classification_button.pack()

    train_regression_button = tk.Button(regression_frame, text="Entrenar
Regresión", command=lambda: train_and_plot_regression(adaline_regression,
X_regression, y_regression, canvas_regression))
    train_regression_button.pack()

    root.mainloop()

if __name__ == "__main__":
    main()
```