Centro Universitario de Ciencias Exactas e Ingenierías

Ingeniería en Computación
Práctica 1 ejercicio 2

PRESENTA:

Ramirez Gutierrez Hugo Vladimir
Código: 220287144
Materia:

Seminario de Solución de problemas de Inteligencia Artificial 2

Docente: Diego Campos Pena

Índice:

Introducción:

Un perceptrón es una unidad computacional que procesa información y realiza decisiones binarias. Se puede pensar en un perceptrón como una neurona artificial que toma múltiples entradas, las procesa mediante una función matemática y produce una salida.

Desarrollo:

1. Importaciones de Bibliotecas

El código importa las siguientes bibliotecas:

- numpy para operaciones numéricas.
- random para la generación de números aleatorios.
- matplotlib.pyplot para la visualización de datos.
- sklearn.metrics para calcular métricas de evaluación del modelo.
- sklearn.model_selection para dividir los datos en conjuntos de entrenamiento y prueba, y para la validación cruzada.
- mpl_toolkits.mplot3d para gráficos tridimensionales.

2. Funciones Definidas

El código define varias funciones que realizan tareas específicas:

- step_function: Implementa una función de activación.
- predict: Realiza la predicción utilizando un perceptrón.
- read_data: Lee los datos de un archivo CSV.
- train_perceptron: Entrenar un perceptrón utilizando el algoritmo de aprendizaje del perceptrón.
- test_perceptron: Prueba el perceptrón entrenado.
- Funciones de partición de datos: random_partition, stratified_partition, temporal_partition, kfold_partition, y feature_group_partition, que dividen los datos en conjuntos de entrenamiento y prueba utilizando diferentes estrategias de partición.
- plot_3d_dataset: Visualiza los datos en un gráfico tridimensional.
- main: La función principal que orquesta todo el flujo del programa.

Código:

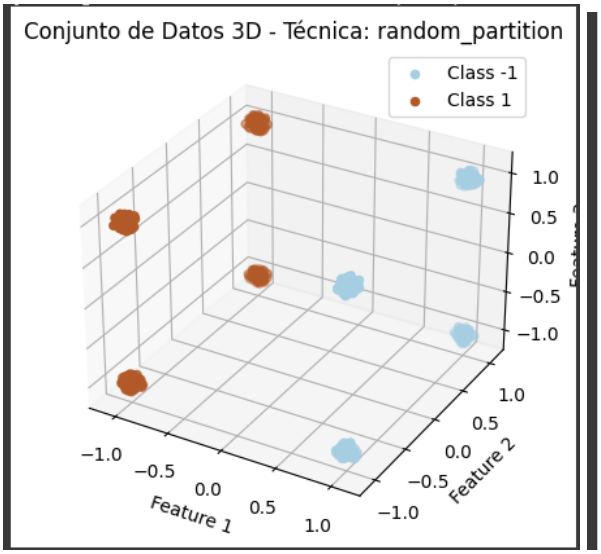
```
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from sklearn.model_selection import train_test_split, KFold
from mpl_toolkits.mplot3d import Axes3D

def step_function(summation):
    return np.sign(summation)

def predict(input_data, weights, bias):
    return step_function(np.dot(input_data, weights) + bias)

def read_data(file_path):
    data = np.genfromtxt(file_path, delimiter=',')
    inputs = data[:, :-1]
    outputs = data[:, -1]
    return inputs, outputs

def train_perceptron(inputs, outputs, learning_rate, max_epochs,
convergence_criterion):
    weights = np.random.rand(inputs.shape[1])
    bias = np.random.rand()
    epochs = 0
    while epochs < max_epochs:
        has_converged = True
        for input_pattern, target_output in zip(inputs, outputs):
            predicted_output = np.dot(weights, input_pattern) +
bias
            error = target_output - predicted_output
            if np.any(np.abs(error) > convergence_criterion):

                has_converged = False
                weights += learning_rate * error * input_pattern
                bias += learning_rate * error
        if has_converged:
            break
        epochs += 1
    return weights, bias

def test_perceptron(inputs, weights, bias):
```

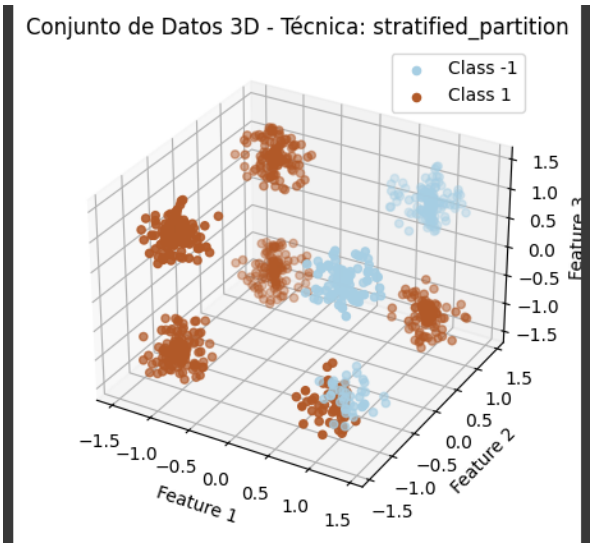
```
    return np.vectorize(step_function)(np.dot(inputs, weights) +
bias)

def random_partition(inputs, outputs, train_ratio):
    indices = np.random.permutation(len(inputs))
    train_size = int(train_ratio * len(inputs))
    train_indices, test_indices = indices[:train_size],
indices[train_size:]
    return inputs[train_indices], outputs[train_indices],
inputs[test_indices], outputs[test_indices]

def stratified_partition(inputs, outputs, train_ratio):
    inputs_train, inputs_test, outputs_train, outputs_test =
train_test_split(inputs, outputs, train_size=train_ratio,
stratify=outputs)
    return inputs_train, outputs_train, inputs_test, outputs_test

def temporal_partition(inputs, outputs, train_ratio):
    train_size = int(train_ratio * len(inputs))
    return inputs[:train_size], outputs[:train_size],
inputs[train_size:], outputs[train_size:]

def kfold_partition(inputs, outputs, train_ratio, n_splits=5):
    kf = KFold(n_splits=n_splits, shuffle=True)
    train_indices, test_indices = next(kf.split(inputs))
    return inputs[train_indices], outputs[train_indices],
inputs[test_indices], outputs[test_indices]

def feature_group_partition(inputs, outputs, train_ratio):
    train_size = int(train_ratio * len(inputs))
    return inputs[:train_size], outputs[:train_size],
inputs[train_size:], outputs[train_size:]

def plot_3d_dataset(inputs, outputs, title="Dataset"):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    unique_classes = np.unique(outputs)
    colors = plt.cm.Paired(np.linspace(0, 1,
len(unique_classes)))
    for class_label, color in zip(unique_classes, colors):
        class_indices = np.where(outputs == class_label)
        ax.scatter(*inputs[class_indices].T, label=f'Class
{int(class_label)}', c=[color])
```

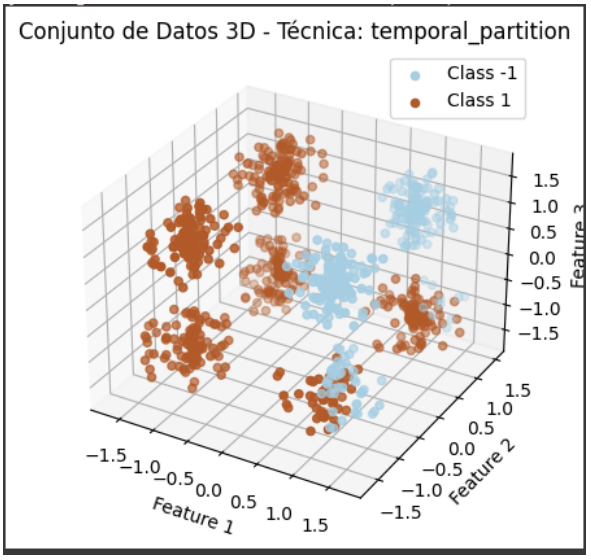
```
    ax.set_title(title)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")
    ax.set_zlabel("Feature 3")
    ax.legend()
    plt.show()

def main():
    datasets = ['/content/spheres2d10.csv',
'/content/spheres2d50.csv', '/content/spheres2d70.csv']
    additional_datasets = random.sample(datasets, 2)
    datasets += additional_datasets

    for i, dataset in enumerate(datasets):
        inputs, outputs = read_data(dataset)
        print(f'\nDataset {i + 1}: {dataset}')

        train_ratio = float(input("Ingrese el valor de
train_ratio (por ejemplo, 0.8 para un 80%): "))

        techniques_to_test = [random_partition,
stratified_partition, temporal_partition, kfold_partition,
feature_group_partition]

        print("Técnicas disponibles:")
        for idx, technique in enumerate(techniques_to_test,
start=1):
            print(f"{idx}. {technique.__name__}")

        selected_techniques = [techniques_to_test[int(choice) -
1] for choice in input("Seleccione las técnicas de partición (por
ejemplo, 1, 2, 3, 4, 5): ").split()]

        for selected_technique in selected_techniques:
            inputs_train, outputs_train, inputs_test,
outputs_test = selected_technique(inputs, outputs,
train_ratio=train_ratio)
            trained_weights, trained_bias =
train_perceptron(inputs_train, outputs_train, learning_rate=0.1,
max_epochs=30, convergence_criterion=0.01)
            predicted_outputs = test_perceptron(inputs_test,
trained_weights, trained_bias)
```

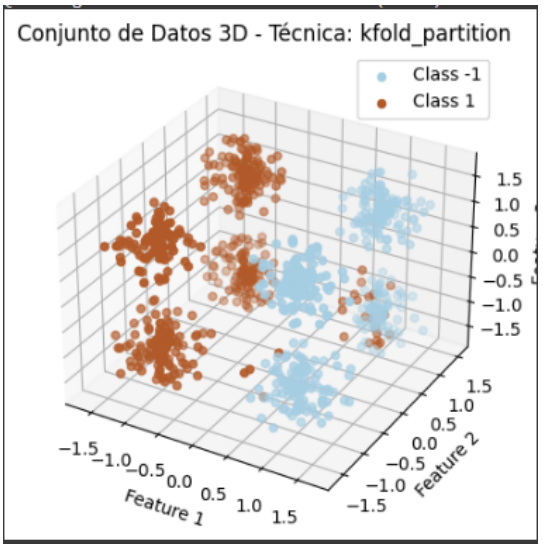
```
            accuracy = accuracy_score(outputs_test,
predicted_outputs)
            precision = precision_score(outputs_test,
predicted_outputs, average='weighted')
            recall = recall_score(outputs_test,
predicted_outputs, average='weighted')
            f1 = f1_score(outputs_test, predicted_outputs,
average='weighted')
            print(f'Técnica: {selected_technique.__name__} -
Dataset: {dataset}')
            print(f'Accuracy: {accuracy}\nPrecision:
{precision}\nRecall: {recall}\nF1 Score: {f1}\n')
            plot_choice = input("¿Desea graficar los resultados
en 3D? (Sí/No): ").lower()
            if plot_choice == "si" or plot_choice == "sí":
                plot_3d_dataset(inputs_test, predicted_outputs,
title=f"Conjunto de Datos 3D - Técnica:
{selected_technique.__name__}")

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

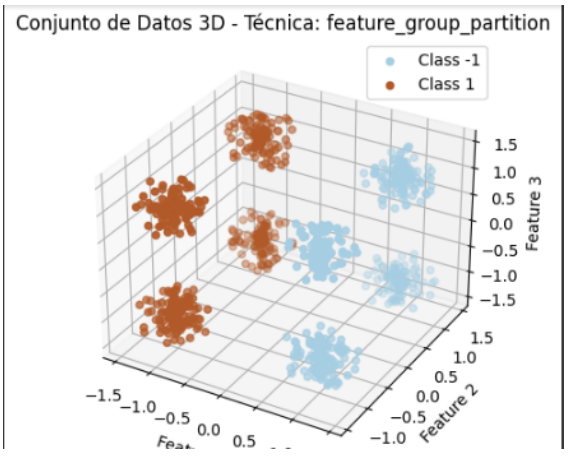
Resultados:











Conclusión:

Los perceptrones son modelos básicos pero poderosos de aprendizaje automático. Aunque tienen limitaciones en términos de la complejidad de los problemas que pueden resolver, proporcionan una introducción valiosa al mundo del aprendizaje automático y sirven como punto de partida para comprender conceptos más avanzados en inteligencia artificial.