Practica 2
Comparación de Métodos de Partición de Datos con Perceptrón Simple

---

Universidad de Guadalajara, Centro Universitario de Ciencias exactas e Ingenierías.

Alumno:
Cárdenas Pérez Calvin Cristopher Código: 214798706

Carrera:
Ingeniería En Computación (INCO)

Materia:
Seminario de Solución de Problemas de Inteligencia artificial II

Maestro:
Dr. Diego Oliva

Ciclo:
2024 A

Sección:
D05

Fecha de Entrega:
lunes 04 de marzo de 2024

Introducción

En el ámbito del aprendizaje automático, la correcta partición de los datos de entrenamiento y prueba es fundamental para evaluar el rendimiento de los modelos.

En esta práctica, exploraremos diferentes métodos de partición de datos y su impacto en la precisión de un modelo de clasificación. Al comparar y analizar estas técnicas, podremos comprender mejor cómo influyen en la capacidad predictiva de nuestros modelos. Este análisis nos proporcionará una visión más completa sobre la selección de métodos de partición adecuados para conjuntos de datos específicos y escenarios de aplicación.

Desarrollo

Importaciones de bibliotecas:

NumPy: Es una biblioteca de Python que proporciona soporte para arreglos y matrices de gran tamaño, junto con una amplia colección de funciones matemáticas para operar en estos arreglos.

Matplotlib. Pyplot: Es una colección de funciones de estilo de comando que hacen que Matplotlib funcione como MATLAB. Cada función de pyplot realiza algún cambio en una figura: por ejemplo, crea una figura, crea un área de trazado en una figura, traza algunas líneas en un área de trazado, decora la trama con etiquetas, etc.

train_test_split, StratifiedShuffleSplit, ShuffleSplit: Son funciones de scikit-learn para dividir conjuntos de datos en subconjuntos de entrenamiento y prueba.

os: Es un módulo que proporciona una manera portátil de usar la funcionalidad dependiente del sistema operativo.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, ShuffleSplit
import os
```

Clase perceptrón

Esta clase representa un perceptrón simple con una inicialización que toma el número de entradas y una tasa de aprendizaje opcional como argumentos.

```
# Clase de tu perceptrón
class PerceptronSimple:
    def __init__(self, num_inputs, learning_rate=0.1):
        self.weights = np.random.rand(num_inputs)
        self.learning_rate = learning_rate

    def train(self, X, y, max_epochs=100):
        epoch = 0
        while epoch < max_epochs:
            error_count = 0
            for i in range(len(X)):
                prediction = self.predict(X[i])
                error = y[i] - prediction
                if error != 0:
                    self.weights += self.learning_rate * error * X[i]
                    error_count += 1
            if error_count == 0:
                print("Entrenamiento completado en la época", epoch)
                break
            epoch += 1
        print("Entrenamiento finalizado")
```

Método para leer archivos

```python
# Método para leer los datos del archivo CSV
def read_data(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
    X = []
    y = []
    for line in lines:
        data = line.strip().split(',')
        X.append([float(x) for x in data[:-1]])
        y.append(int(data[-1]))
    return np.array(X), np.array(y)
```

Este método se utiliza para leer los datos de un archivo CSV. Toma el nombre del archivo como entrada y devuelve los datos en forma de matrices NumPy.

Métodos de partición

Estos métodos son diferentes formas de dividir los datos en conjuntos de entrenamiento y prueba. Cada uno toma los datos de entrada X, las etiquetas y y un porcentaje de datos de entrenamiento como entrada, y devuelve las divisiones correspondientes.

```python
# Métodos de partición de datos
def train_test_split_random(X, y, train_percentage):
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=train_percentag
    return X_train, X_test, y_train, y_test

def train_test_split_stratified(X, y, train_percentage):
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=train_percentag
    return X_train, X_test, y_train, y_test

def stratified_shuffle_split(X, y, train_percentage):
    ss = StratifiedShuffleSplit(n_splits=1, train_size=train_percentage / 100, random_st
    train_index, test_index = next(ss.split(X, y))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    return X_train, X_test, y_train, y_test

def shuffle_split(X, y, train_percentage):
    ss = ShuffleSplit(n_splits=1, train_size=train_percentage / 100, random_state=42)
    train_index, test_index = next(ss.split(X))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    return X_train, X_test, y_train, y_test

def random_sampling(X, y, train_percentage):
    # Selecciona aleatoriamente el conjunto de entrenamiento con el porcentaje especific
    indices = np.random.choice(len(X), size=int(train_percentage / 100 * len(X)), replac
    X_train, y_train = X[indices], y[indices]
    # Los datos restantes son para pruebas
    X_test, y_test = np.delete(X, indices, axis=0), np.delete(y, indices)
    return X_train, X_test, y_train, y_test
```

Función plot_decision_boundary:

```python
# Función para visualizar la línea de separación
def plot_decision_boundary(X, y, perceptron, technique_name):
    plt.scatter(X[:,0], X[:,1], c=y)
    plt.xlabel('X1', fontsize=10)
    plt.ylabel('X2', fontsize=10)
    plt.title(technique_name, fontsize=12)
    x_values = np.linspace(-1.5, 1.5, 100)
    y_values = -(perceptron.weights[0] * x_values) / perceptron.weights[1]
    plt.plot(x_values, y_values, label='Recta de separación')
    plt.legend()
    plt.show()
```

Esta función se utiliza para trazar la línea de decisión aprendida por el perceptrón en un diagrama de dispersión de los datos de entrada.

Función principal main:

```python
def main():
    # Definición de los datasets y técnicas de partición
    datasets = ["spheres1d10.csv", "spheres2d50.csv", "spheres2d70.csv"]
    techniques = {
        "spheres1d10.csv": [train_test_split_random, stratified_shuffle_split],
        "spheres2d50.csv": [train_test_split_stratified, shuffle_split],
        "spheres2d70.csv": [random_sampling]
    }

    dataset_results = {}  # Almacenar resultados para cada dataset
    overall_results = {}  # Almacenar resultados de todas las técnicas

    # Iteración sobre cada dataset
    for dataset in datasets:
        dataset_name = os.path.basename(dataset)  # Obtener solo el nombre del archivo
        print(f"Dataset: {dataset_name}")

        # Captura del porcentaje de datos para entrenamiento
        while True:
            train_percentage_str = input("Ingrese el porcentaje de datos destinados para ent
```

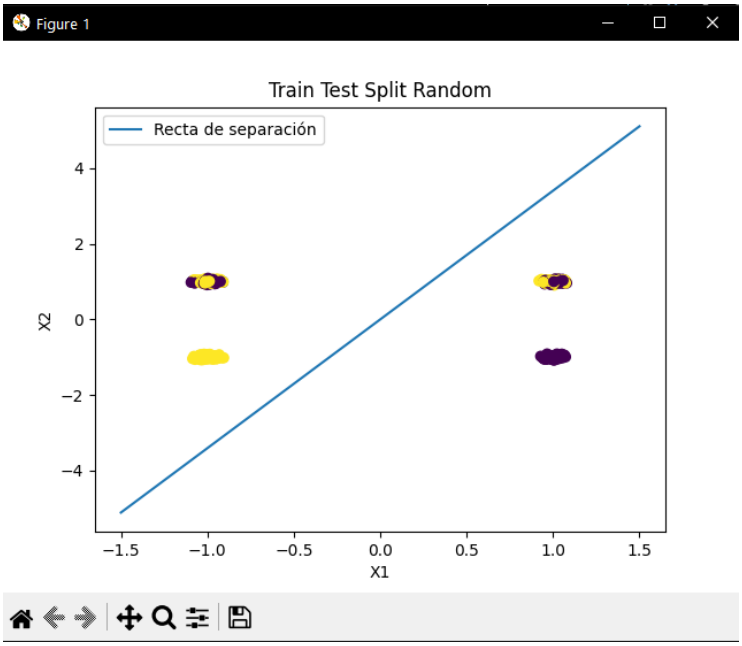
Esta función es donde ocurre la mayor parte de la lógica del programa. Maneja la iteración sobre cada conjunto de datos, la división de los datos utilizando diferentes métodos, el entrenamiento del perceptrón y la evaluación del rendimiento del modelo.

Resultados

Por último tenemos los resultados, al principio nos pide que ingresemos el porcentaje de cada dataset y nos ira mostrando cada grafica y al final la comparación de la precisión de todas las particiones, la salida de las 5 particiones de los 3 datasets, 1 dataset con 2 particiones, otro con 2 y el ultimo con una.

```
Dataset: spheres1d10.csv
Ingrese el porcentaje de datos destinados para entrenamiento (0-100): 60
Técnica: train_test_split_random
Entrenamiento finalizado
Precisión: 0.4525

Técnica: stratified_shuffle_split
Entrenamiento finalizado
Precisión: 0.405

Dataset: spheres2d50.csv
Ingrese el porcentaje de datos destinados para entrenamiento (0-100): 60
Técnica: train_test_split_stratified
Entrenamiento finalizado
Precisión: 0.507

Técnica: shuffle_split
Entrenamiento finalizado
Precisión: 0.517

Dataset: spheres2d70.csv
Ingrese el porcentaje de datos destinados para entrenamiento (0-100): 60
Técnica: random_sampling
Entrenamiento finalizado
Precisión: 0.4945
```

Archivo 1



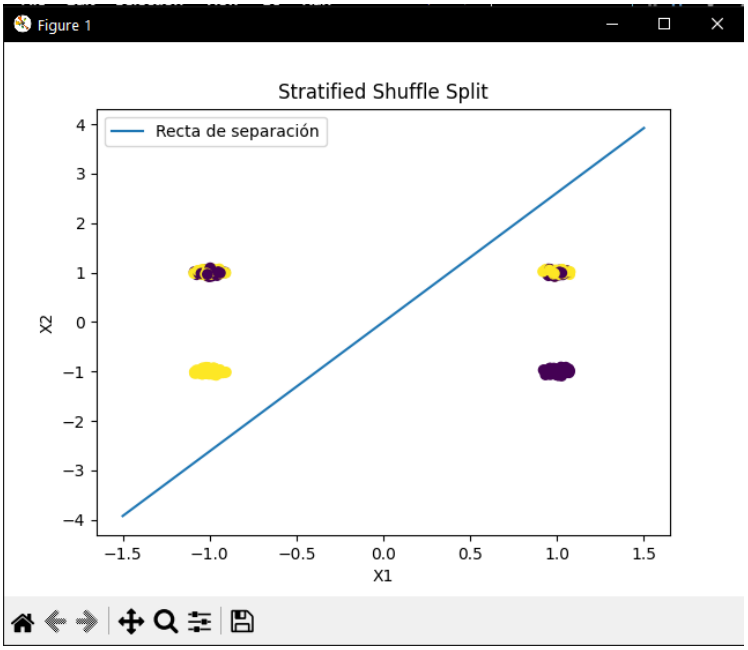


Figure 1
Stratified Shuffle Split
Recta de separación
X2
X1
4
3
2
1
0
−1
−2
−3
−4
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

Archivo 2

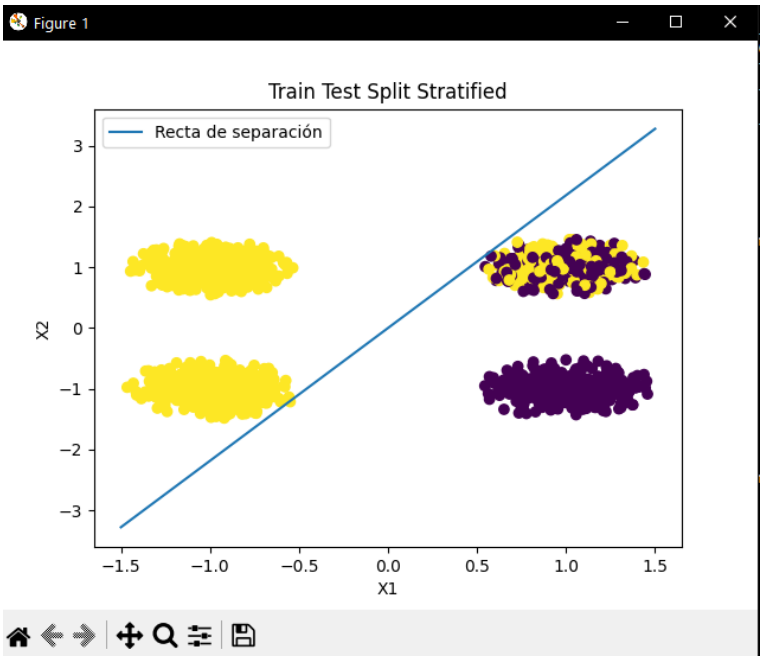
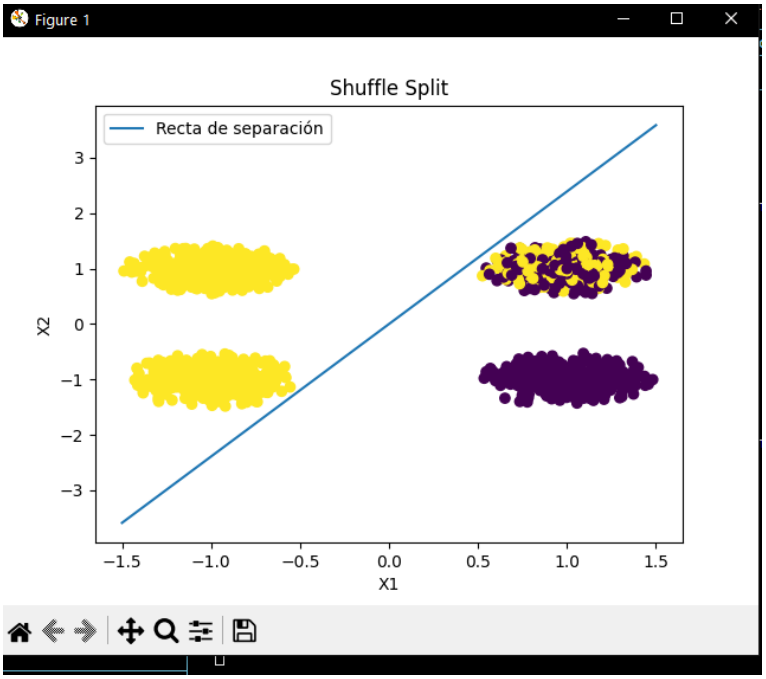
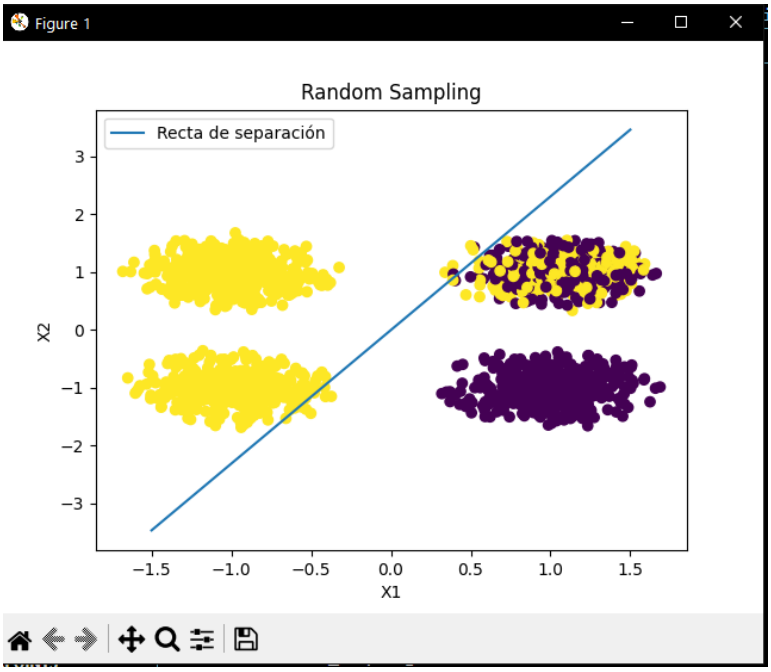


Figure 1
Train Test Split Stratified
Recta de separación
X2
X1
3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5



Archivo 3



Comparaciones en graficas

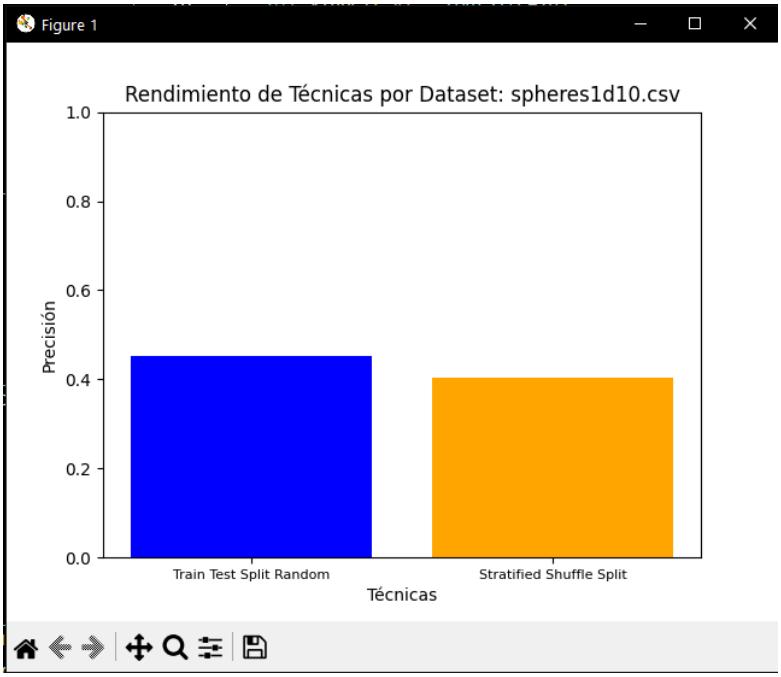


Figure 1
Rendimiento de Técnicas por Dataset: spheres1d10.csv
1.0
0.8
0.6
0.4
0.2
0.0
Precisión
Train Test Split Random
Stratified Shuffle Split
Técnicas

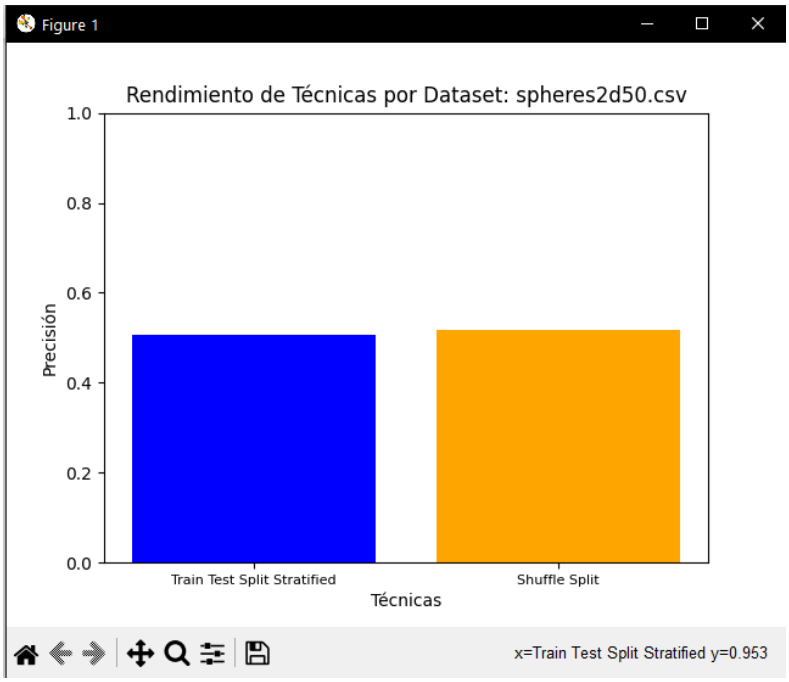


Figure 1
Rendimiento de Técnicas por Dataset: spheres2d50.csv
1.0
0.8
0.6
0.4
0.2
0.0
Precisión
Train Test Split Stratified
Shuffle Split
Técnicas
x=Train Test Split Stratified y=0.953

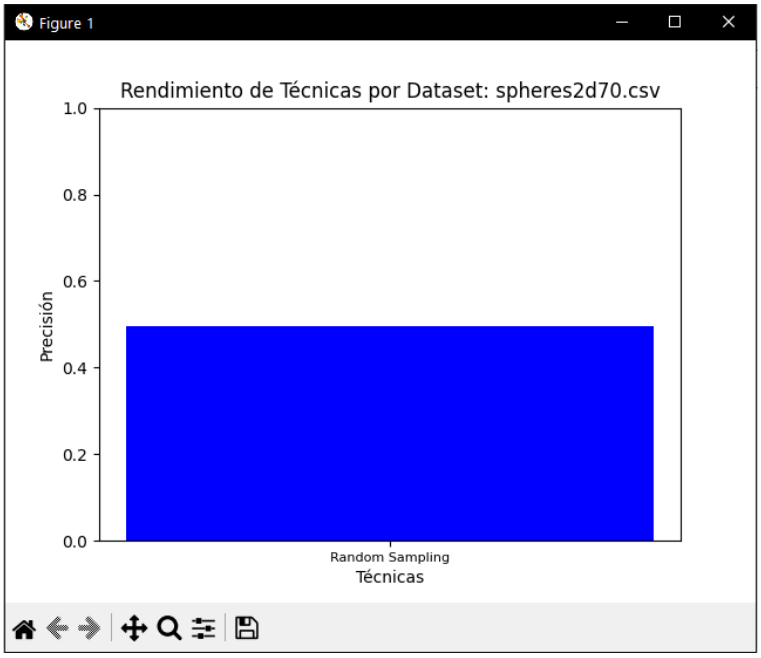


Figure 1
Rendimiento de Técnicas por Dataset: spheres2d70.csv
1.0
0.8
0.6
0.4
0.2
0.0
Precisión
Random Sampling
Técnicas

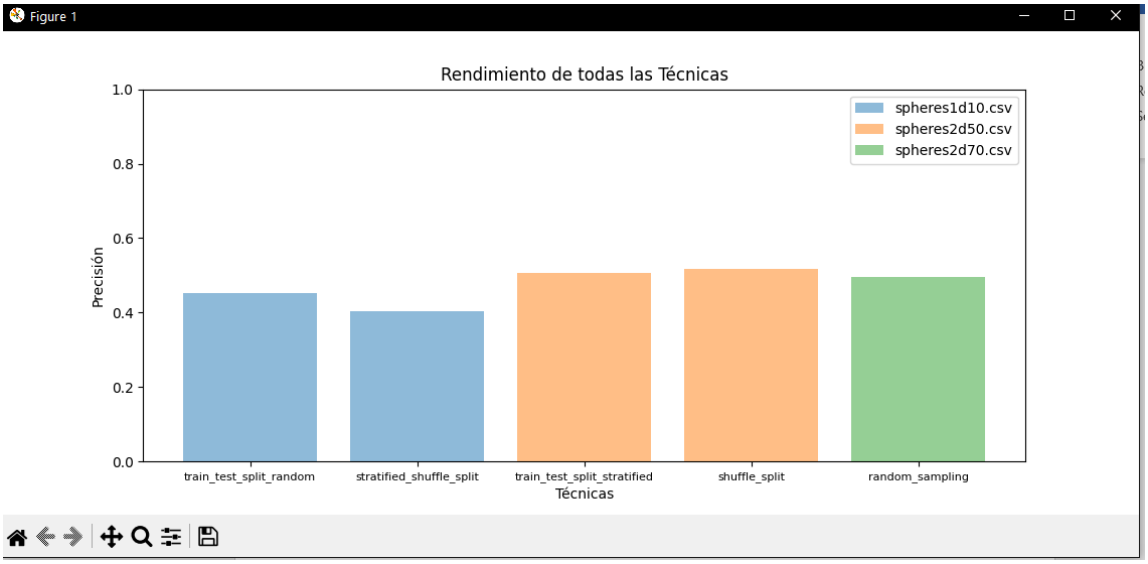


Figure 1
Rendimiento de todas las Técnicas
spheres1d10.csv
spheres2d50.csv
spheres2d70.csv
1.0
0.8
0.6
0.4
0.2
0.0
Precisión
train_test_split_random
stratified_shuffle_split
train_test_split_stratified
shuffle_split
random_sampling
Técnicas

Conclusión

Estuvo entretenida la practica pero muy laboriosa y difícil la verdad si me costó mucho trabajo y más que el perceptrón simple, a lado de esta práctica el perceptrón está muy fácil pero siempre es bueno aprender cosas nuevas.

Además, explore diferentes técnicas de partición de datos y entrenamiento de un perceptrón simple. Utilizamos métodos como la partición aleatoria, la partición estratificada y la partición estratificada con mezcla aleatoria para evaluar el rendimiento del modelo en diferentes conjuntos de datos.

Tambien pude ver cómo la elección del método de partición puede afectar el rendimiento del modelo, y cómo el perceptrón simple puede aprender a separar eficazmente clases en conjuntos de datos de diferentes dimensiones. Este análisis me ayudo a comprender cómo seleccionar y aplicar métodos de partición de datos para entrenar modelos de aprendizaje automático.