INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

PRACTICA_LAB_2: CLASIFICADORES DE DATOS

Unidad de aprendizaje: Fundamentos de Inteligencia Artificial

Alumnos:

Flores Lara Alberto

Profesor:
Catalán Salgado Edgar armando

Grupo:
4BV1

El código presentado es una implementación de clasificación supervisada utilizando dos métodos: K-Nearest Neighbors (KNN) y Clasificación por Distancia Mínima. Está diseñado para cargar datos desde un archivo de texto plano, permitiendo al usuario seleccionar columnas específicas como atributos, ingresar un vector de prueba y luego clasificar este vector utilizando uno de los dos algoritmos mencionados. El usuario puede elegir entre distancias euclidianas o de Manhattan para calcular la cercanía entre puntos.

Importación de bibliotecas

- numpy y pandas son bibliotecas para operaciones matemáticas y manipulación de datos, respectivamente.
- math es una biblioteca estándar de Python utilizada para operaciones matemáticas.

```
- import numpy as np
- import pandas as pd
- import math
```

Distancia_euclidiana(point1, point2):

Calcula la distancia euclidiana entre dos puntos utilizando la fórmula euclidiana.

- Parámetros:
  - point1: Primer punto en forma de lista o array.
  - point2: Segundo punto en forma de lista o array.

Devuelve la distancia euclidiana entre los dos puntos.

Distancia_manhattan(point1, point2)

Calcula la distancia de Manhattan entre dos puntos.

- Parámetros:
  - point1: Primer punto en forma de lista o array.
  - point2: Segundo punto en forma de lista o array.

Devuelve la distancia de Manhattan entre los dos puntos.

```
# Función para calcular la distancia euclidiana entre dos puntos
def distancia_euclidiana(point1, point2):
    distance = 0.0
    for i in range(len(point1)):
        distance += (point1[i] - point2[i]) ** 2
    return math.sqrt(distance)

def distancia_manhattan(point1, point2):
    distance = 0.0
    for i in range(len(point1)):
        distance += (point1[i] - point2[i])
    return distance
```

Cargardatos(archivo, delimitador)

Lee un archivo de texto plano utilizando Pandas read_csv.

- Parámetros:
  - ❖ archivo: Nombre del archivo a leer.
  - ❖ delimitador: El carácter utilizado como delimitador en el archivo.

Devuelve un DataFrame de Pandas con los datos cargados desde el archivo.

```
# Función para cargar el archivo de texto plano
def cargardatos(archivo, delimitador):
    data=pd.read_csv(archivo, delimiter=delimitador)
    return data
```

Función principal (main())

1. Solicita al usuario el nombre del archivo y el carácter delimitador para cargar los datos.
2. Muestra la cantidad de filas, columnas y tipos de datos en el DataFrame.
3. Permite al usuario seleccionar columnas específicas para formar un vector de atributos.
4. Solicita valores para crear un vector de prueba.
5. Proporciona opciones para elegir entre dos tipos de clasificadores: KNN o Distancia Mínima.

```
def main():
    archivo=input("Escriba el nombre del archivo de donde obtendremos la informacion: ")
    delimitador=input("Seleccione cual es el signo delimitador del archivo: ")
    datos=cargardatos(archivo,delimitador)
    num_filas, num_columnas = datos.shape
    print(f"El DataFrame tiene {num_filas} patrones y {num_columnas} atributos.")
    tipos_de_datos = datos.dtypes
    print(tipos_de_datos)
    #Seleccionamos atributos para nuestro vector
    z=(str(input("Escriba el nombre de la columna que quiere predecir: ")))
    x = datos.drop(z, axis=1).values
    y = np.array(datos[z])
    limite_inferior_1 = int(input(f"Seleccione el limite inferior (Valores entre 0 y
{num_columnas-2}) para generar el vector de attributos: "))
    limite_superior_1 = int(input(f"Ahora el limite superior (Valores entre
{limite_inferior_1} y {num_columnas-2}): "))
    matriz_patrones = x[:, limite_inferior_1:limite_superior_1+1]
    nombres_columnas = datos.columns
    nombres_columnas_restringidos =
nombres_columnas[limite_inferior_1:limite_superior_1+1]
    vector_test=[]
    for nombre_columna in nombres_columnas_restringidos:
        dato=float(input(f"Ingrese el valor de {nombre_columna}: "))
        vector_test.append(dato)
    print(vector_test)
    opc1=0
    while(opc1!=1 and opc1!=2):
```

```
        opc1=int(input("Ingrese el tipo de clasificador que desee usar:\n 1.Clasificador
Knn 2.Clasificador distancia minima\n"))
        if(opc1==1):
            clas_knn(matriz_patrones,y, [vector_test])
        elif(opc1==2):
            class_min(matriz_patrones,y, vector_test)
        else:
            print("Seleccione una opcion correcta")
```

Clasificador KNN (clas_knn(x, y, x_test))

1. Define una clase ClasificadorKNN que implementa un clasificador K-Nearest Neighbors.
2. Permite al usuario elegir entre distancias euclidianas o de Manhattan.
3. Entrena el clasificador KNN con los datos de entrada x y las etiquetas y.
4. Clasifica el x_test proporcionado y muestra la clase predicha.

Clasificador de Distancia Mínima (class_min(x, y, x_test))

1. Define una clase ClasificadorDistanciaMinima que implementa un clasificador de Distancia Mínima.
2. Similar al clasificador KNN, permite al usuario elegir entre distancias euclidianas o de Manhattan.
3. Entrena el clasificador con los datos de entrada x y las etiquetas y.
4. Clasifica el x_test proporcionado y muestra la clase predicha.

```
def clas_knn(x,y,x_test):
    # Clasificador KNN
    class ClasificadorKNN:
        def __init__(self, n_neighbors=1):
            self.n_neighbors = n_neighbors

        def fit(self, X, y):
            self.X_train = X
            self.y_train = y

        def predict(self, X):
            met_distancia=int(input("Ingrese el tipo de distancia que desee usar:\n
1.Euclidiana 2.Manhattan\n"))
            y_pred = []
            for sample in X:
                distances = []
                for i, train_sample in enumerate(self.X_train):
                    if(met_distancia==1):
                        distance = distancia_euclidiana(sample, train_sample)
                    else:
                        distance = distancia_manhattan(sample, train_sample)
                    distances.append((distance, self.y_train[i]))
```

```
                distances.sort(key=lambda x: x[0])
                neighbors = distances[:self.n_neighbors]
                neighbor_labels = [neighbor[1] for neighbor in neighbors]
                prediction = max(set(neighbor_labels), key=neighbor_labels.count)
                y_pred.append(prediction)
            return y_pred


    # Pedir al usuario el número de vecinos a considerar
    n_neighbors_input = int(input("Introduce el número de vecinos a considerar: "))

    # Crear una instancia del clasificador KNN con el número de vecinos especificado
    knn_classifier = ClasificadorKNN(n_neighbors=n_neighbors_input)

    # Entrenar el clasificador con los datos de entrenamiento
    knn_classifier.fit(x, y)

    #Clasificar el vector
    y_pred = knn_classifier.predict(x_test)
    print(f"La clase a la que pertenece es {y_pred}")


def class_min(x,y,x_test):
    class ClasificadorDistanciaMinima:
        def fit(self, X, y):
            self.X_train = X
            self.y_train = y

        def predict(self, X):
            met_distancia=int(input("Ingrese el tipo de distancia que desee usar:\n
1.Euclidiana 2.Manhattan\n"))
            y_pred = []
            for sample in X:
                min_distance = float('inf')
                nearest_label = None
                for i, train_sample in enumerate(self.X_train):
                    if(met_distancia==1):
                        distance = distancia_euclidiana(sample, train_sample)
                    else:
                        distance = distancia_manhattan(sample, train_sample)
                    if distance < min_distance:
                        min_distance = distance
                        nearest_label = self.y_train[i]
                y_pred.append(nearest_label)
            return y_pred

    min_distance = ClasificadorDistanciaMinima()
    min_distance.fit(x, y)

    # Ajustar la entrada de x_test para que sea una lista de un solo elemento
```

```
    y_pred = min_distance.predict([x_test])
    print(f"La clase a la que pertenece es {y_pred}")

main()
```