Apuntes del jueves de semana 3 - 6/3/2025 - Brandon Adones Calderon Cubero

***Abstract*—En este documento, se visualizara los apuntes de la clase del 6 de marzo de IA.**

I. Repaso

Nota: Se vio el primer notebook.

Un vector es un tipo de arreglo, el cual tiene dirección y magnitud asociadas y puede ser definido en un espacio de N dimensiones. Además, se compone de un punto de origen y un punto final, donde por lo general el punto de origen es el origen de las coordenadas; sin embargo, puede ser cualquiera.

El vector está compuesto de la resta de cada una de las entradas, ejemplo:

Vector $A = (a_1, a_2, \ldots, a_n)$ y $B = (b_1, b_2, \ldots, b_n)$, donde el vector sería:

$$\mathbf{u} = (b_1, b_2, \ldots, b_n) - (a_1, a_2, \ldots, a_n)$$

Si partimos de un punto $A = (0, 0)$ y como punto $B = (4, 3)$, el vector resultante sería $\mathbf{u} = (4, 3)$. Recordar que la resta se realiza entrada por entrada.

```
print(type(dataFrameTraining["Clase"]))
```

```
<class 'pandas.core.series.Series'>
```

A. Magnitud de vector o norma

La magnitud del vector es la distancia desde el origen hasta el punto que representa el vector. Ejemplo: Distancia desde el punto $A = (0, 0)$ al punto $B = (4, 3)$.

Existen dos tipos de distancia: - La distancia Manhattan (L1) - La distancia Euclidiana (L2)

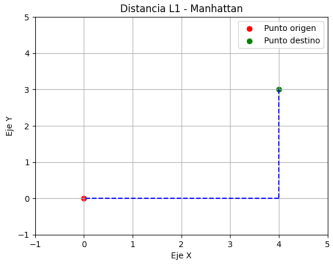
Nota: Si un vector es representado $\|U\|_1$, este representa la distancia Manhattan, mientras que si es representado $\|U\|$ o $\|U\|_2$, sería la distancia Euclidiana.

a) Distancia Manhattan: La distancia Manhattan entre dos puntos

$A = (x_1, y_1, z_1, \ldots, n_1)$ y $B = (x_2, y_2, z_2, \ldots, n_2)$ en un espacio n-dimensional se calcula con la fórmula:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Ejemplo: Puntos $A = (0, 0)$ y $B = (4, 3)$. Usando la distancia Manhattan y una gráfica donde se colocaron los puntos A y B:



Si contamos desde x del punto de origen hasta x del punto B y luego desde y del punto de origen hasta y del punto B, obtenemos que la magnitud es 7.

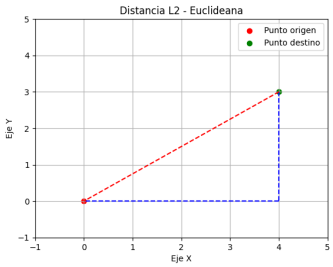
*Nota: Nos movemos entre los ejes.

b) Distancia Euclidiana: Este método obtiene la distancia más corta mediante la fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ejemplo: Puntos $A = (0, 0)$ y $B = (4, 3)$.

$$d = \sqrt{(4-0)^2 + (3-0)^2} = \sqrt{16 + 9} = \sqrt{25} = 5$$



La línea roja representa la distancia obtenida mediante el método euclidiano.

B. Vector Unidad

Un vector unidad es un vector con magnitud de 1. Se usa para indicar dirección sin considerar la longitud, ayudando a simplificar cálculos.

La fórmula para obtener el vector unidad es:

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|}$$

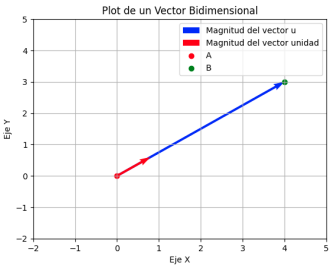
Ejemplo: Si $\mathbf{u} = (4, 3)$ y su magnitud es 5:

$$\hat{\mathbf{u}} = \left(\frac{4}{5}, \frac{3}{5}\right)$$

Verificación con la fórmula de la distancia euclidiana:

$$\sqrt{\left(\frac{4}{5}\right)^2 + \left(\frac{3}{5}\right)^2} = 1$$

Imagen de representación gráfica del vector unidad:



C. Producto Punto

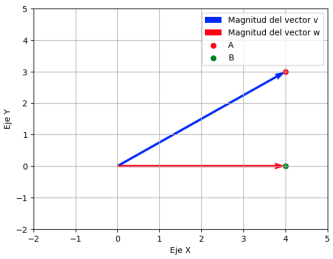
El producto punto (o escalar) es una operación entre dos vectores que da como resultado un escalar:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} u_i \cdot v_i$$

Ejemplo: Vectores $\boldsymbol{u} = (4, 3)$ y $\boldsymbol{v} = (4, 0)$:

$$\boldsymbol{u} \cdot \boldsymbol{v} = (4 \times 4) + (3 \times 0) = 16$$

Imagen representativa del producto punto:



D. Propiedad Producto Punto

Relación con el coseno del ángulo θ:

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \cdot \|\mathbf{v}\| \cdot \cos(\theta)$$

Ejemplo: $\mathbf{u} = (4, 3)$ y $\mathbf{v} = (4, 0)$ Magnitudes: $\|\mathbf{u}\| = 5$, $\|\mathbf{v}\| = 4$

$$(4, 3) \cdot (4, 0) = 5 \times 4 \times \cos(\theta)$$

Despejando:

$$\frac{16}{20} = \cos(\theta)$$

Ángulo:

$$\theta = \arccos\left(\frac{16}{20}\right) \approx 36.87^\circ$$

E. Vector Co-direccional

Dos vectores son co-direccionales si tienen la misma dirección. Existe un escalar $K \neq 0$ tal que:

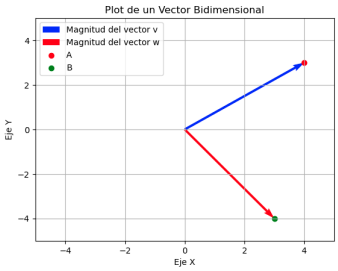
$$\boldsymbol{u} = K \cdot \boldsymbol{v}$$

Ejemplo: El vector unidad de un vector. *Nota:* El ángulo de dos vectores co-direccionales es 0.

F. Vectores Ortogonales

Dos vectores son ortogonales si el ángulo entre ellos es de 90 grados.

Ejemplo gráfico de vectores ortogonales:



II. Introducción a la programación vectorial

Existen dos formas de ejecutar programas: - CPU - GPU (más rápida por su capacidad de procesamiento vectorial).

Se usan librerías como Pandas y Numpy.

A. Introducción a Numpy

Numpy es una biblioteca para programación matemática y numérica. - Permite crear tensores y realizar operaciones vectoriales. - Alta eficiencia computacional. - Ofrece álgebra lineal y soporte para hardware especializado (GPU). - Es base para librerías como SciPy, Pandas, etc.

Librerías a importar:

```
import time
import numpy as np
import matplotlib.pyplot as plt
```

Creación de arrays:

```
# Crear un array NumPy de 1D con los
#primeros 10 números enteros.

array_1d = np.arange(10)
print("Array 1D:", array_1d)

#Crear un array que inicie en cinco.
#y termine en 10 -1

array_1d = np.arange(5,10)
print("Array 1D:", array_1d)

array_1d = np.arange(5,10)
print("Array 1D:", array_1d)

#Crear un array que inicie en
#cinco y que vaya de dos en dos.

array_1d = np.arange(5,10, 2)
print("Array 1D:", array_1d)
```

Los outputs correspondientes serían los siguientes:

```
Array 1D: [0 1 2 3 4 5 6 7 8 9]
Array 1D: [5 6 7 8 9]
Array 1D: [5 6 7 8 9]
Array 1D: [5 7 9]
```

Array de unos y ceros respectivamente:

```
array_1d = np.ones((10))
print("Array 1D:", array_1d)

array_1d = np.zeros((10))
print("Array 1D:", array_1d)
```

Los outputs correspondientes serian los siguientes:

```
Array 1D: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Array 1D: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Creación de arrays: Matrices

Por ejemplo matriz identidad , matriz de unos 4x5 y una matriz de dos dimensiones 4x5:

```
 array_1d = np.identity(3)
 print(array_1d)

 array_1d = np.ones((4, 5))
print(array_1d)

array_1d = np.ones((2,4, 5))
print(array_1d)
```

El output es el siguiente:

```
#Identidad
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

#Matriz de 1s 4x5
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]

#Matriz de 1s 4x5 de 2 dimensiones
 [[[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]

 [[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]]
```

Operaciones básicas:

*Nota: Esta librería permite las operaciones element-wise, que son aquellas por ejemplo donde tengo 2 listas, una lista es L = [1,2,3] y la otra es K = [4,5,6], entonces por ejemplo si aplico la suma y es element-wise sumaría el 1 con el 4, el 2 con el 5, el 3 con el 6 y obtendríamos el siguiente resultado, O = [5,7,9].

```
# Crear dos arrays NumPy de 1D con
#números aleatorios
lst = [2, 4, 6]
lst2 = [1, 3, 5]
# converting list to array
array_random_1 = np.array(lst)
array_random_2 = np.array(lst2)

# Realizar operaciones element-wise
suma_resultado = array_random_1 +
array_random_2
resta_resultado = array_random_1 -
array_random_2
multiplicacion_resultado = array_random_1
* array_random_2
division_resultado = array_random_1 /
array_random_2

# Imprimir los resultados de cada
#operación

print("Suma:", suma_resultado)
print("Resta:", resta_resultado)
print("Multiplicación:",
multiplicacion_resultado)
print("División:", division_resultado)
print("Elevar:", array_random_1**2)
```

Los resultados son los siguientes:

```
Suma: [ 3  7 11]
Resta: [1 1 1]
Multiplicación: [ 2 12 30]
División: [2. 1.33333333 1.2 ]
Elevar: [ 4 16 36]
```

Operaciones por dimensiones:

```
#El axis es el eje
#Si axis es = 0 es columna
#si axis es = 1 es fila
#Los resultados sin importar
#el eje, se obtienen horizontalmente
matriz_2d = np.array([[1, 2, 3],
                      [4, 5, 6],
                      [7, 8, 9]])

print("Resultado de sumar todos los
elementos de la matriz:",
np.sum(matriz_2d))
print("Resultado de sumar
los elementos de cada
columna matriz: :"
, np.sum(matriz_2d, axis=0))
# matriz_2d[:, 0] , matriz_2d[:, 1]
#, matriz_2d[:, 2]
print("Resultado de sumar los
elementos de cada fila matriz:
:", np.sum(matriz_2d, axis=1))
# matriz_2d[0, :] , matriz_2d[1,
#:] , matriz_2d[2 :]
\end{verbatin}
```

```
Los outputs son:

\begin{verbatim}
Resultado de sumar todos
los elementos de la matriz: 45

Resultado de sumar los elementos
de cada columna matriz: : [12 15 18]

Resultado de sumar los elementos
de cada fila matriz: : [ 6 15 24]
```

Funciones estadísticas:

```
    array_random = np.random.rand(100)
# Calcular la media, desviación estándar,
#valor máximo y mínimo
media = np.mean(array_random)
desviacion_estandar = np.std(array_random)
valor_maximo = np.max(array_random)
valor_minimo = np.min(array_random)

# Imprimir los resultados
#de cada operación
print("Media:", media)
print("Desviación estándar:",
desviacion_estandar)
print("Valor máximo:", valor_maximo)
print("Valor mínimo:", valor_minimo)
```

Los outputs son:

```
Media: 0.49849874449499715
Desviación estándar: 0.2847645198211511
Valor máximo: 0.9756109910102987
Valor mínimo: 0.009293305538057517
```

Comparación de programar:
Distancia euclidiana de una lista (Ciclos):

```
# Función de programación por ciclos
para calcular la distancia euclidiana
def distancia_euclidiana_por_ciclo
(vector1, vector2):
    inicio = time.time()

    resultado = 0
    # [a, b, c]
    # [1, 2, 3]
    # [(a, 1), (b, 2), (c, 3)]
    for elem1, elem2 in
    zip(vector1, vector2):
    resultado += (elem1 - elem2)**2

    resultado = np.sqrt(resultado)

    fin = time.time()
    tiempo_total = fin - inicio
    print(f"Tiempo por ciclos:
    {tiempo_total:.6f} segundos")

    return resultado
```

Distancia euclidiana de una lista (vectorial):

```
# Función de programación vectorial con
#NumPy para calcular la
#distancia euclidiana
def distancia_euclidiana_
vectorial_explicita
(vector1, vector2, printTime = True):
    inicio = time.time()

    # Elevar al cuadrado cada
    #elemento de la diferencia
    diferencia_cuadrada =
    (vector1 - vector2)**2

    # Sumar los elementos cuadrados
    suma_cuadrada =
    np.sum(diferencia_cuadrada)

    # Calcular la raíz cuadrada de la suma
    resultado = np.sqrt(suma_cuadrada)

    fin = time.time()
    tiempo_total = fin - inicio
    if printTime:
        print(f"Tiempo vectorial
        explícito:
        {tiempo_total:.6f} segundos")

    return resultado
```

Comparación de tiempos:

```
# Crear dos listas de números aleatorios
#con un tamaño de N
lista_random_1 = np.random.rand(300000)
lista_random_2 = np.random.rand(300000)

# Utilizar las funciones con las listas
#y arrays aleatorios como inputs
resultado_ciclo =
distancia_euclidiana_por_ciclo
(lista_random_1, lista_random_2)
resultado_vectorial =
distancia_euclidiana_vectorial_explicita
(lista_random_1, lista_random_2)
# Imprimir los resultados completos
print("\nResultado completo por ciclos:"
, resultado_ciclo)
print("Resultado completo vectorial:"
, resultado_vectorial)
```

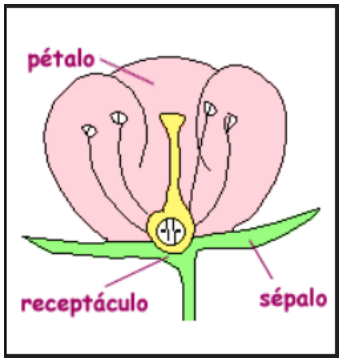
Los outputs son:

```
Tiempo por ciclos: 0.229363 segundos
Tiempo vectorial: 0.001000 segundos

Resultado completo por ciclos:
223.60713457541607
```

```
Resultado completo vectorial:
223.60713457541158

#Como se puede observar,
#el que se programo vectorialmente
#es mas rapido.
```

Ejemplo práctico de pandas:

Suponga tiene un conjunto de datos que representa flores con dos características: longitud del sépalo, ancho del pétalo que representa un conjunto de datos que puede ser utilizado como entrenamiento.



Pandas es una biblioteca que proporciona estructuras de datos y herramientas de análisis de datos.

Tomando como referencia las siguientes imágenes de código:

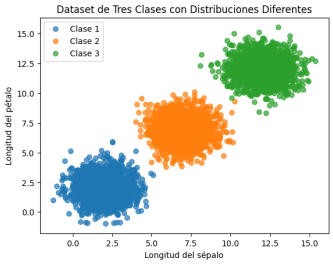
En este apartado se crea toda la lógica con respecto al desarrollo de la semilla y su visualización:

```python
import pandas as pd
def createDataset(samples, seed = 42, plot=True, isTraining = True):
    # Establecer semilla para reproducibilidad
    np.random.seed(seed) #settear la semilla y casos sean reproducibles
    #Creo clases de plantas con diferente distribuciones
    #(2,52,2,33) Uno representa petalo y otro sepalo
    clase_1 = np.random.normal(loc=2, scale=1, size=(samples//3, 2))
    clase_2 = np.random.normal(loc=7, scale=1, size=(samples//3, 2)) # cambio la media
    clase_3 = np.random.normal(loc=12, scale=1, size=(samples//3, 2)) #cambio la media
    # Etiquetas de clase representa cada uno de los features
    etiquetas_clase_1 = np.full(samples//3, 0)  # 0 para clase 1
    etiquetas_clase_2 = np.full(samples//3, 1)  # 1 para clase 2
    etiquetas_clase_3 = np.full(samples//3, 2)  # 2 para clase 3

    # Combinar las clases en un solo dataset, y stack se apilan verticalmente
    dataset = np.vstack((clase_1, clase_2, clase_3))
    etiquetas = np.concatenate((etiquetas_clase_1, etiquetas_clase_2, etiquetas_clase_3))
    if plot:
        # Visualización de las clases en un scatter plot y los estaquea porque son horizontales
        plt.scatter(clase_1[:, 0], clase_1[:, 1], label='Clase 1', alpha=0.7)
        plt.scatter(clase_2[:, 0], clase_2[:, 1], label='Clase 2', alpha=0.7)
        plt.scatter(clase_3[:, 0], clase_3[:, 1], label='Clase 3', alpha=0.7)
        plt.title('Dataset de Tres Clases con Distribuciones Diferentes')
        plt.xlabel('Longitud del sépalo')
        plt.ylabel('Longitud del pétalo')
        plt.legend()
        plt.show()
```

A partir de aquí se crea el dataframe de pandas:

```python
    #Crear un dataframe de pandas
    columnas = ['Longitud_sepalo', 'Longitud_petalo']
    df = pd.DataFrame(dataset, columns=columnas)
    if isTraining:
        # si quiero agregar df nada mas agrego la etiqueta con los datos
        df['Clase'] = etiquetas
    # Barajar el DataFrame
    df = df.sample(frac=1).reset_index(drop=True)
    return df

dataFrameTraining = createDataset(5000)
```

El resultado obtenido del código anterior es el siguiente:



Como se logra ver en el código pasado se utilizan dataframes, los cuales permiten ver la información tabular como se muestra en las siguientes imágenes:

```python
dataFrameTraining.head(5)
```

	Longitud_sepalo	Longitud_petalo	Clase
0	8.180990	7.403275	1
1	8.267652	7.288728	1
2	2.393797	0.072327	0
3	8.062657	6.624366	1
4	11.541123	10.982415	2

Head(5) muestra los primeros 5 datos, mientras que tail(5) muestra los últimos 5 datos.

```python
dataFrameTraining.tail(5)
```

	Longitud_sepalo	Longitud_petalo	Clase
4993	7.007880	6.674389	1
4994	1.160782	1.690788	0
4995	1.526161	1.985548	0
4996	2.672861	2.591814	0
4997	7.568103	5.950345	1

El método count permite contar la cantidad de datos:

```python
dataFrameTraining.count()
```

```
Longitud_sepalo    4998
Longitud_petalo    4998
Clase              4998
dtype: int64
```

Para obtener los datos de la columna de un dataframe se coloca en el nombre del dataframe y entre [] con ”” el nombre de la columna como se muestra en la siguiente imagen:

```python
#Retorna una serie de Pandas
#Una serie es un objeto unidimensional
dataFrameTraining["Clase"]
```

```
0       1
1       1
2       0
3       1
4       2
       ..
4993    1
4994    0
4995    0
4996    0
4997    1
Name: Clase, Length: 4998, dtype: int32
```

Los datos obtenidos se les conoce como series:

```python
print(type(dataFrameTraining["Clase"]))
```

```
<class 'pandas.core.series.Series'>
```

Otro método que se puede hacer es acceder al nombre de dataframe con doble [], ya que este retornará la información con las columnas correspondientes como se muestra en la imagen, además de poder agregarse más columnas:

```
# Retorna otro DataFrame pero que contiene una columna
dataFrameTraining[["Clase"]]
```

	Clase
0	1
1	1
2	0
3	1
4	2
...	...
4993	1
4994	0
4995	0
4996	0
4997	1

4998 rows × 1 columns

Otro ejemplo:

```
dataFrameTraining[["Longitud_petalo", "Clase"]]
```

	Longitud_petalo	Clase
0	7.403275	1
1	7.288728	1
2	0.072327	0
3	6.624366	1
4	10.982415	2
...	...	...
4993	6.674389	1
4994	1.690788	0
4995	1.985548	0
4996	2.591814	0
4997	5.950345	1

4998 rows × 2 columns

Como se logra observar en la siguiente imagen, devuelve el dataframe:

```
print(type(dataFrameTraining[["Clase"]]))

<class 'pandas.core.frame.DataFrame'>
```

La función describe(), permite obtener datos estadísticos del dataframe, que es útil para ser analizados o realizar ajustes en los datos.

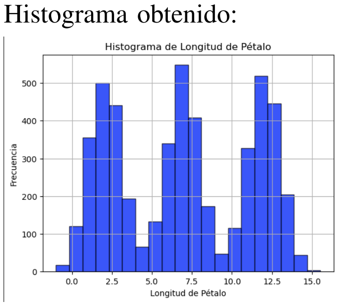
```
dataFrameTraining.describe()
```

	Longitud_sepalo	Longitud_petalo	Clase
count	4998.000000	4998.000000	4998.000000
mean	6.995514	7.000525	1.000000
std	4.204612	4.187172	0.816578
min	-1.241267	-0.991136	0.000000
25%	2.645403	2.714886	0.000000
50%	6.954242	6.968260	1.000000
75%	11.314625	11.320903	2.000000
max	15.377383	15.529055	2.000000

Además se tienen los histogramas que nos permiten ver visualmente datos importantes como la cantidad de distribuciones que se posee:

```
plt.hist(dataFrameTraining["Longitud_petalo"], bins=20, color='blue', edgecolor='black', alpha=0.7)

# Personalizar el gráfico
plt.title('Histograma de Longitud de Pétalo')
plt.xlabel('Longitud de Pétalo')
plt.ylabel('Frecuencia')
plt.grid(True)

# Mostrar el gráfico
plt.show()
```

Histograma obtenido:



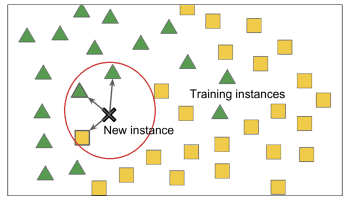
Como se logra apreciar, hay 3 distribuciones.

B. Ejemplo de KNN - K nearest neighbor

Se tiene un set de datos sin clasificar, como se muestra en el siguiente código:

```
#Tomar los indices de los elementos más cercanos
tres_indices_menores = [indice for indice, _ in nearestInstances]
tres_indices_menores

[4471, 3812, 3759]
```

La idea se basa en tomar un elemento desconocido y compararlo con todos los ademas que si conozco, como se muestra en la siguiente imagen:



Entonces creamos un código el cual nos permita almacenar los indices y las distancias.

```
newInstance = dataFrameTesting.iloc[0]
# Lista para almacenar las distancias y los índices
distancias_euclidianas = []
for i, data in dataFrameTraining.iterrows():
    distancia = distancia_euclidiana_vectorial_explicita(data[["Longitud_sepalo", "Longitud_petalo"]], newInstance, False)
    # Almacenar la distancia y el índice en la lista
    distancias_euclidianas.append((i, distancia))
#Ordenar por la segunda columna de menor a mayor
distancias_euclidianas.sort(key=lambda x: x[1])
nearestInstances = distancias_euclidianas[:3]
nearestInstances

[(4471, 0.02090716628236026),
 (3812, 0.040641022369069434),
 (3759, 0.047466766789896406)]
```

Datos almacenados en distancias euclidianas:

```
distancias_euclidianas[0]

(4471, 0.02090716628236026)

distancias_euclidianas[-1]

(102, 16.64123360701286)
```

Se toman los "vecinos" más cercanos:

```
#Tomar los indices de los elementos más cercanos
tres_indices_menores = [indice for indice, _ in nearestInstances]
tres_indices_menores

[4471, 3812, 3759]
```

Los vecinos más cercanos serían los que se muestran en pantalla:

```
print(newInstance)
dataFrameTraining.iloc[tres_indices_menores]

Longitud_sepalo    3.015425
Longitud_petalo    2.224081
Name: 0, dtype: float64
```

	Longitud_sepalo	Longitud_petalo	Clase
4471	3.030283	2.238789	0
3812	3.056057	2.223239	0
3759	3.017661	2.271495	0

K-nearest neighbor su concepto se basa en que clasifica un punto de datos según la mayoría de las clases de sus k vecinos más cercanos, basado en una métrica de distancia.

Su estructura en código se puede ver en las siguientes imágenes:

```python
from scipy.stats import mode


class KNN():
    def __init__(self, dataFrameTraining, dataFrameTesting, number_neighbors=5, classes=3):
        self.dataFrameTraining = dataFrameTraining
        self.dataFrameTesting = dataFrameTesting
        self.number_neighbors = number_neighbors
        self.neigh_ind = []
        self.classes = classes

    # (samples, features1, feature2)
    def euclidean_distance(self, a, b):
        #Ejecuta la formula para cada sample del conjunto de datos AXIS = 1
        return np.sqrt(np.sum((a-b)**2, axis=1))

    def computeDistances(self):
        # (N, len(dataFrameTraining))
        point_dist = []
        for i, data in dataFrameTesting.iterrows():
            point_dist += [self.euclidean_distance(data, dataFrameTraining[["Longitud_sepalo", "Longitud_petalo"]])]

        #Cada iteración contiene un vector de tamaño len(dataFrameTraining) con el calculo de la distancia
        for distances in point_dist:
            #Enumerate (index, value)
            enum_neigh = enumerate(distances)
            sorted_neighbor = sorted(enum_neigh, key=lambda x: x[1])[:self.number_neighbors]
            ind_list = [tupla[0] for tupla in sorted_neighbor]
            self.neigh_ind.append(ind_list)
        # Retorna los indices a los K vecinos mas cercanos
        return np.array(self.neigh_ind)

    def predict(self):
        predictions = []

        for neighbors_indices in self.neigh_ind:
            # Obtener las etiquetas de clase correspondientes a los indices de los vecinos
            neighbor_classes = self.dataFrameTraining.iloc[neighbors_indices]['Clase']

            # Calcular la moda de las clases de los vecinos
            predicted_class, _ = mode(neighbor_classes, axis=None, keepdims=True)
            predictions.append(predicted_class[0])  # Tomar la primera moda si hay empate

        return np.array(predictions)
```

Para ponerlo en funcionamiento se utiliza el siguiente código, donde los parámetros que se le pasan al constructor sería el dataframetraining (contiene etiquetas) y el DataFrame-Testing (no contiene etiquetas) y como resultado da las clases predichas:

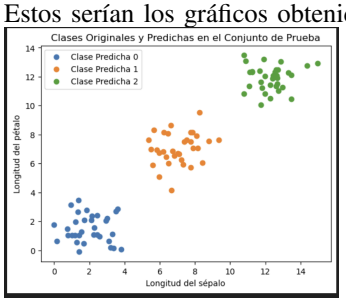
```python
classifier = KNN(dataFrameTraining, dataFrameTesting)
classifier.computeDistances()
labels_predic = classifier.predict()
dataFrameTesting['ClasePredichas'] = labels_predic
dataFrameTesting.head(5)
```

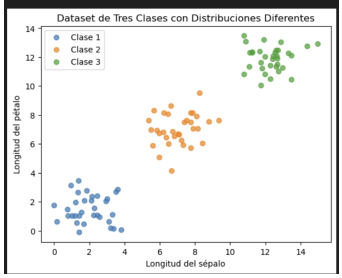
	Longitud_sepalo	Longitud_petalo	ClasePredichas
0	3.015425	2.224081	0
1	10.765395	13.470797	2
2	9.360551	7.610260	1
3	6.670456	4.167921	1
4	1.014797	1.041632	0

Con el siguiente código se busca visualizar los datos obtenidos mediante la técnica utiliza:

```python
# Visualizar las clases predichas en el conjunto de prueba
for clase in range(3):
    plt.scatter(
        dataFrameTesting[dataFrameTesting['ClasePredichas'] == clase]['Longitud_sepalo'],
        dataFrameTesting[dataFrameTesting['ClasePredichas'] == clase]['Longitud_petalo'],
        label=f'Clase Predicha {clase}',
        alpha=1
    )

plt.title('Clases Originales y Predichas en el Conjunto de Prueba')
plt.xlabel('Longitud del sépalo')
plt.ylabel('Longitud del pétalo')
plt.legend()
plt.show()

#Comprobemos si la clasificación está bien hecha.
dataFrameTestingConLabelsOriginales = createDataset(100, seed=41, plot=True, isTraining=False)
```

Estos serían los gráficos obtenidos:





Como se logra observar comparando el original con respecto al modelo, este modelo lo hizo perfecto.

III. Links a tomar en cuenta

Documentación de pandas:

- Documentación de NumPy: https://numpy.org/doc/stable/reference/generated/numpy.arange.html
- Guía de usuario de Pandas: https://pandas.pydata.org/docs/user$_guide/index.html$
- Sitio Freedium: https://freedium.cfd/

IV. Notas importantes

Se puede obtener puntos extra si se solicita al profesor ayudarlo con la traduccion del libro matematico.