Introducción

En la siguiente practica se hizo uso del algoritmo de K nearest neighbor (K vecinos más cercanos), también conocido como KNN o k-NN. Para poder aplicarlo durante el desarrollo de la practica primero se comprendió en que consiste y que es. El algoritmo clasifica cada dato nuevo en el grupo que corresponda, según tenga *k* vecinos más cerca de un grupo o de otro. Es decir, calcula la distancia del elemento nuevo a cada uno de los existentes, y ordena dichas distancias de menor a mayor para ir seleccionando el grupo al que pertenecer. Este grupo será, por tanto, el de mayor frecuencia con menores distancias.

Para los problemas de clasificación, se utiliza la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinados. A comparación con otros algoritmos de aprendizaje supervisado, KNN no genera un modelo fruto del aprendizaje con datos de entrenamiento, sino que el aprendizaje sucede en el mismo momento en el que se prueban los datos. Cabe señalar que el algoritmo KNN también forma parte de una familia de modelos de "aprendizaje perezoso", lo que significa que solo almacena un conjunto de datos de entrenamiento en lugar de pasar por una etapa de entrenamiento. Esto también significa que todo el cálculo ocurre cuando se realiza una clasificación o predicción. Dado que depende en gran medida de la memoria para almacenar todos sus datos de entrenamiento, también se lo denomina método de aprendizaje basado en instancias o basado en la memoria. Otro punto para considerar es que a medida que crece un conjunto de datos, KNN se vuelve cada vez más ineficiente, lo que compromete el rendimiento general del modelo. Se usa comúnmente para sistemas de recomendación simples, reconocimiento de patrones, extracción de datos, predicciones del mercado financiero, detección de intrusos y más.

El algoritmo KNN es muy sensible a:

- La variable k, de modo que con valores distintos de k podemos obtener resultados también muy distintos. Este valor suele fijarse tras un proceso de pruebas con varias instancias.
- La métrica de similitud utilizada, puesto que esta influirá fuertemente, en las relaciones de cercanía que se irán estableciendo en el proceso de construcción del algoritmo.

Su mayor debilidad es la lentitud en el proceso de clasificación puesto que su objetivo no es obtener un modelo optimizado, sino que cada instancia de prueba es comparada contra todo el juego de datos de entrenamiento y, será la bondad de los resultados lo que determinará el ajuste de aspectos del algoritmo como el propio valor *k*, el criterio de selección de instancias para formar parte del juego de datos "D" de entrenamiento o la propia métrica de medida de similitud. El objetivo del algoritmo del vecino más cercano es identificar los vecinos más cercanos de un punto de consulta dado, de modo que podamos asignar una etiqueta de clase a ese punto. En resumen, KNN tiene los siguientes pasos básicos:

- Calcular la distancia: Una opción popular es la distancia euclidiana, pero también hay otras medidas que pueden ser más adecuadas para un entorno dado e incluyen la distancia de Manhattan y Minkowski.

Distancia Euclidiana

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Distancia Manhattan

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

Distancia Minkowski

$$[\sum_{i=1}^{k}(|x_i - y_i|)^4]^{\frac{1}{4}}$$

- Encontrar sus vecinos más cercanos
- Votar por las etiquetas

Las ventajas que tiene este algoritmo son las siguientes:

- **No paramétrico.** No hace suposiciones explícitas sobre la forma funcional de los datos, evitando los peligros de la distribución subyacente de los datos.
- **Algoritmo simple.** Para explicar, comprender e interpretar.
- **Alta precisión (relativa).** Es bastante alta pero no competitiva en comparación con modelos de aprendizaje mejor supervisados.
- **Insensible a los valores atípicos.** La precisión puede verse afectada por el ruido o las características irrelevantes.

Las desventajas de este algoritmo son:

- **Basado en instancia.** El algoritmo no aprende explícitamente un modelo, en su lugar, elige memorizar las instancias de *capacitación* que se utilizan posteriormente como conocimiento para la fase de predicción. Concretamente, esto significa que solo cuando se realiza una consulta a nuestra base de datos, es decir cuando le pedimos que prediga una etiqueta dada una entrada, el algoritmo usará las instancias de entrenamiento para escupir una respuesta.
- **Computacionalmente costoso.** Porque el algoritmo almacena todos los datos de entrenamiento.
- **Requisito de memoria alta.** Almacena todos (o casi todos) los datos de entrenamiento.

Código

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

semilla = int(input("Ingresa el valor de la semilla a utilizar: "))
N = 10
np.random.seed(semilla)
clase1 = np.zeros((N,2))
clase2 = np.zeros((N,2))

#Se obtienen los 10
for i in range(0,10):
  clase1[i,0] = np.random.rand()+np.sin(np.random.rand())
  clase1[i,1] = np.random.rand()+np.cos(np.random.rand())
  clase2[i,0] = np.random.rand()+np.sin(np.random.rand())
  clase2[i,1] = np.random.rand()+np.cos(np.random.rand())

plt.scatter(clase1[:,0],clase1[:,1],marker='*',color = 'red', alpha=0.8,
label ='Clase A') #Se colocan los puntos de la clase A en el gráfico
plt.scatter(clase2[:,0],clase2[:,1],marker='s',color = 'blue', alpha=0.8,
label ='Clase B') #Se colocan los puntos de la clase B en el gráfico

x = float(input("Ingresa la coordenada en x: "))
y = float(input("Ingresa la coordenada en y: "))


nuevo_punto = [x, y] #Se crea el nuevo punto con las coordenadas ingresadas
plt.scatter(x, y, color = 'green') #Se agrega el punto al gráfico con los
otros puntos
plt.legend(loc="best") #Se colocan las etiquetas de las clases en el gráfico
plt.show() #Se muestra el gráfico con el nuevo punto y los puntos de las
clases A y B

distanciasClaseA = list() #Se crea la lista para almacenar las distancias
euclidianas que hay del punto generado hacia los puntos de la clase A
distanciasClaseB = list() #Se crea la lista para almacenar las distancias
euclidianas que hay del punto generado hacia los puntos de la clase B

"""Se calculan las distancias euclidianas entre el punto generado y los
puntos de la clase A, los resultados se agregan a
la lista de distancias de la clase A"""
for i in range(0,10):
 distanciasClaseA.append(math.dist(nuevo_punto,
[clase1[:,0][i],clase1[:,1][i]]))
```

```
"""Se calculan las distancias euclidianas entre el punto generado y los
puntos de la clase B, los resultados se agregan a
la lista de distancias de la clase B"""
for j in range(0,10):
  distanciasClaseB.append(math.dist(nuevo_punto,
[clase2[:,0][j],clase2[:,1][j]]))

distanciasTotales = dict() #Se crea un diccionario para relacionar las
distancias obtenidas con su clase correspondiente

distanciasTotales['A'] = distanciasClaseA #Se asignan las distancias de la
clase A al diccionario
distanciasTotales['B'] = distanciasClaseB #Se asignan las distancias de la
clase B al diccionario

print(f'\nDistancias correspondientes a cada clase: {distanciasTotales} \n')

listaDistancias = distanciasClaseA + distanciasClaseB #Se crea una lista con
todas las distancias calculadas (tanto de la clase A y B)
listaDistancias.sort() #Se ordenan todas las distancias obtenidas de menor a
mayor
print(f'Total de distancias de menor a mayor: {listaDistancias} \n')

"""Mediante la función obtenerClase se recibe como parámetro el valor de k
para hallar a los vecinos más cercanos, a su vez se va
registrando la clase de cada punto cercano para lograr identificar la clase
a la que pertenece el punto generado de acuerdo a la
clase mayoritaria de puntos cercanos"""
def obtenerClase(k):
    contadorA = 0 #Se inicia el contador de vecinos de la clase A
    contadorB = 0 #Se inicia el contador de vecinos de la clase B
    for i in range (0,k):
        valor = listaDistancias[i] #Se obtiene la distancia del vecino k a
analizar
        print(f'Distancia {[i+1]}: {valor}')
        if(valor in distanciasTotales['A']): #Determina si la distancia
pertenece a la clase A
            print(' Clase: A')
            contadorA+=1
        elif(valor in distanciasTotales['B']): #Determina si la distancia
pertenece a la clase B
            print(' Clase: B')
            contadorB+=1
```

```
    """Determina la clase a la que pertenece el punto generado, a partir del
recuento de todas las distancias evaluadas para cada clase"""
    if (contadorA > contadorB): #Si la clase mayoritaria es A, entonces el
dato pertenece a la clase A
        print('El valor pertenece a la clase A')
    elif(contadorB > contadorA): #Si la clase mayoritaria es B, entonces el
dato pertenece a la clase B
        print('El valor pertenece a la clase B')
    elif(contadorB == contadorA): #Si se obtuvieron la misma cantidad de
vecinos de cada clase, se asigna el dato a ambas clases
        print('El valor pertenece a la clase A y B')

for v in range(0,2):
    k = int(input("Ingresa el valor de k: ")) #Se ingresa el valor de k para
calcular a los vecinos más cercanos
    obtenerClase(k) #Se invoca a la función para determinar la clase a
partir del valor de k vecinos más cercanos
```

Desarrollo

A continuación, se presentan los resultados obtenidos para 2 semillas con 2 puntos generados en distinta posición y considerando los valores para k de 3 y 6.

Para el punto (0.5, 1.5) y semilla = 10

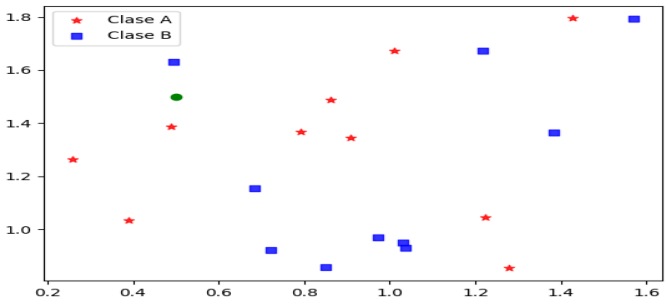

Figura 1. Gráfico con el punto en verde.

```
Ingresa el valor de la semilla a utilizar: 10
Ingresa la coordenada en x: 0.5
Ingresa la coordenada en y: 1.5

Distancias correspondientes a cada clase: {'A': [0.3212800073859273, 0.33830479487865, 0.5381212187283448, 0.3625168729466027, 0.8535114081488886, 1.011573377989447, 0.1123
0103508640198, 0.9739147774437185, 0.47971652147306165, 0.43752181575937715], 'B': [0.618459879409364, 0.13095506815565455, 0.39147385028322873, 0.7363212336969748, 0.73248
33024662077, 0.7828449318556263, 0.7091552076811768, 0.8926638638645779, 0.7641504551074497, 1.110320028000 6722]}

Total de distancias de menor a mayor: [0.11230103508640198, 0.13095506815565455, 0.3212800073859273, 0.33830479487865, 0.3625168729466027, 0.39147385028322873, 0.4375218157
5937715, 0.47971652147306165, 0.5381212187283448, 0.618459879409364, 0.7091552076811768, 0.7324833024662077, 0.7363212336969748, 0.7641504551074497, 0.7828449318556263, 0.8
535114081488886, 0.8926638638645779, 0.9739147774437185, 1.011573377989447, 1.110320028000 6722]
```

Figura 2. Listas con distancias euclidianas.

```
Ingresa el valor de k: 3
Distancia [1]: 0.11230103508640198
 Clase: A
Distancia [2]: 0.13095506815565455
 Clase: B
Distancia [3]: 0.32128000073859273
 Clase: A
El valor pertenece a la clase A
```

Figura 3. Resultados para k=3.

```
Ingresa el valor de k: 6
Distancia [1]: 0.11230103508640198
 Clase: A
Distancia [2]: 0.13095506815565455
 Clase: B
Distancia [3]: 0.32128000073859273
 Clase: A
Distancia [4]: 0.338304794878 65
 Clase: A
Distancia [5]: 0.3625168729466027
 Clase: A
Distancia [6]: 0.39147385028322873
 Clase: B
El valor pertenece a la clase A
```

Figura 4. Resultados para k=6.

Para el punto (0.9, 1.1) y semilla = 10

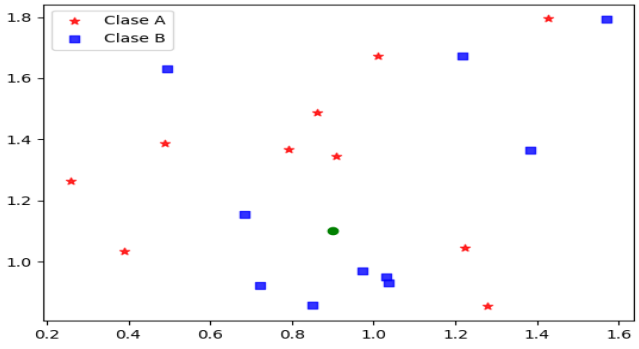

Figura 5. Gráfico con el punto en verde.

```
Ingresa el valor de la semilla a utilizar: 10
Ingresa la coordenada en x: 0.9
Ingresa la coordenada en y: 1.1

Distancias correspondientes a cada clase: {'A': [0.2872028564467218, 0.6633287182039227, 0.5835126103381044, 0.390793127030861 6, 0.32704820429279663, 0.45133069306629453, 0.5013180657215041, 0.8727302623638814, 0.5154141913772616, 0.24559490788857374], 'B': [0.251767815878029, 0.6682623301595687, 0.2225624780464605, 0.6536281056649014, 0.24778255808982483, 0.21807699429947508, 0.14805396091780967, 0.5506300991346553, 0.1988862727046545, 0.9641699996678329]}

Total de distancias de menor a mayor: [0.14805396091780967, 0.1988862727046545, 0.21807699429947508, 0.2225624780464605, 0.24559490788857374, 0.24778255808982483, 0.25176781587802 9, 0.2872028564467218, 0.32704820429279663, 0.390793127030861 6, 0.45133069306629453, 0.5013180657215041, 0.5154141913772616, 0.5506300991346553, 0.5835126103381044, 0.6536281056649014, 0.6633287182039227, 0.6682623301595687, 0.8727302623638814, 0.9641699996678329]
```

Figura 6. Listas con distancias euclidianas.

```
Ingresa el valor de k: 3
Distancia [1]: 0.14805396091780967
 Clase: B
Distancia [2]: 0.1988862727046545
 Clase: B
Distancia [3]: 0.21807699429947508
 Clase: B
El valor pertenece a la clase B
```

Figura 7. Resultados para k=3.

```
Ingresa el valor de k: 6
Distancia [1]: 0.14805396091780967
 Clase: B
Distancia [2]: 0.1988862727046545
 Clase: B
Distancia [3]: 0.21807699429947508
 Clase: B
Distancia [4]: 0.2225624780464605
 Clase: B
Distancia [5]: 0.24559490788857374
 Clase: A
Distancia [6]: 0.24778255808982483
 Clase: B
El valor pertenece a la clase B
```

Figura 8. Resultados para k=6.

Para el punto (0.8, 1) y semilla = 4

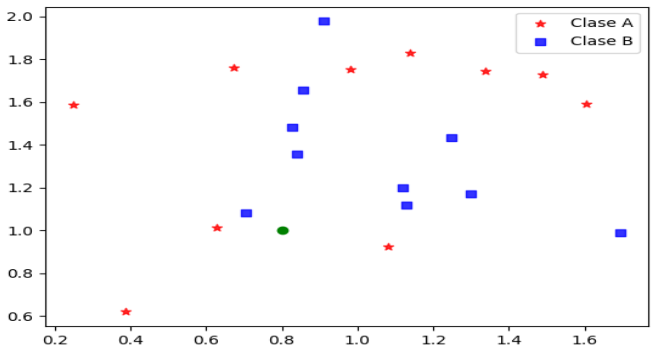

Figura 9. Gráfico con el punto en verde.

```
Ingresa el valor de la semilla a utilizar: 4
Ingresa la coordenada en x: 0.8
Ingresa la coordenada en y: 1

Distancias correspondientes a cada clase: {'A': [1.001146044326546, 0.7702486739740381, 0.5616734978950217, 0.8055032881050904, 0.17083572182832887, 0.7729511770192882, 0.8
972165991733072, 0.917976654729055, 0.9984597148820507, 0.2926953524639091], 'B': [0.9826746239513928, 0.8954285556708615, 0.6244167083867075, 0.351551899978611, 0.37587247
42362105, 0.6571586879257009, 0.4820015223810341, 0.5273368516801381, 0.36063889860470844, 0.12563751683300903]}

Total de distancias de menor a mayor: [0.12563751683300903, 0.17083572182832887, 0.2926953524639091, 0.351551899978611, 0.36063889860470844, 0.3758724742362105, 0.482001522
3810341, 0.5273368516801381, 0.5616734978950217, 0.6244167083867075, 0.6571586879257009, 0.7702486739740381, 0.7729511770192882, 0.8055032881050904, 0.8954285556708615, 0.8
972165991733072, 0.917976654729055, 0.9826746239513928, 0.9984597148820507, 1.001146044326546]
```

Figura 10. Listas con distancias euclidianas.

```
Ingresa el valor de k: 3
Distancia [1]: 0.12563751683300903
 Clase: B
Distancia [2]: 0.17083572182832887
 Clase: A
Distancia [3]: 0.2926953524639091
 Clase: A
El valor pertenece a la clase A
```

Figura 11. Resultados para k=3.

```
Ingresa el valor de k: 6
Distancia [1]: 0.12563751683300903
 Clase: B
Distancia [2]: 0.17083572182832887
 Clase: A
Distancia [3]: 0.2926953524639091
 Clase: A
Distancia [4]: 0.351551899978611
 Clase: B
Distancia [5]: 0.36063889860470844
 Clase: B
Distancia [6]: 0.3758724742362105
 Clase: B
El valor pertenece a la clase B
```

Figura 12. Resultados para k=6.

Para el punto (1.3, 1.6) y semilla = 4

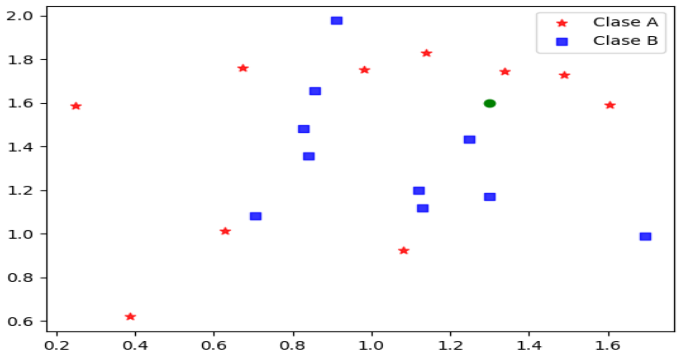

Figura 13. Gráfico con el punto en verde.

```
Ingresa el valor de la semilla a utilizar: 4
Ingresa la coordenada en x: 1.3
Ingresa la coordenada en y: 1.6

Distancias correspondientes a cada clase: {'A': [0.22684804869341915, 0.6459033865294916, 1.3397662711400318, 1.0520406114128429, 0.8908487808392334, 0.35301185649498856, 0
.28062225916617295, 0.14922345730678693, 0.30405007755806357, 0.7113866319767087], 'B': [0.5403728554153979, 0.7263579671085301, 0.17370158946945483, 0.5093519998658467, 0.
4408656315504574, 0.4461386602670736, 0.4868807296772339, 0.42808313414085564, 0.5193613368484717, 0.7875612251838773]}

Total de distancias de menor a mayor: [0.14922345730678693, 0.17370158946945483, 0.22684804869341915, 0.28062225916617295, 0.30405007755806357, 0.35301185649498856, 0.42808
313414085564, 0.4408656315504574, 0.4461386602670736, 0.4868807296772339, 0.5093519998658467, 0.5193613368484717, 0.5403728554153979, 0.6459033865294916, 0.7113866319767087
, 0.7263579671085301, 0.7875612251838773, 0.8908487808392334, 1.0520406114128429, 1.3397662711400318]
```

Figura 14. Listas con distancias euclidianas.

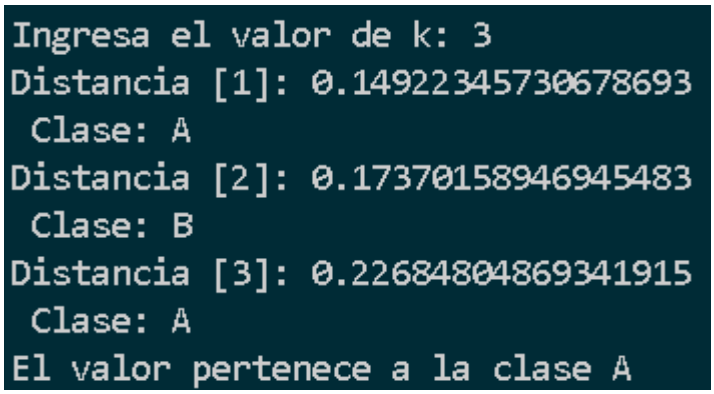

Figura 15. Resultados para k=3.

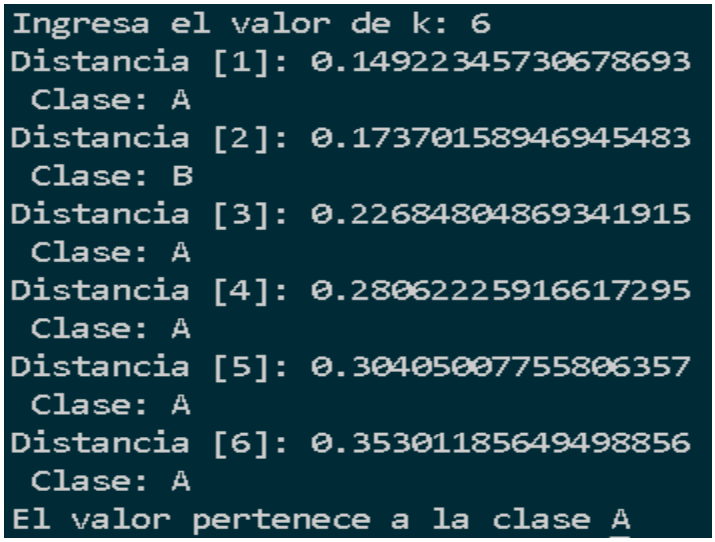

Figura 16. Resultados para k=6.

Conclusiones

Mediante el desarrollo de esta práctica pudimos comprender el funcionamiento e implementación del algoritmo KNN, el cual es muy útil para clasificar algún dato en un conjunto de clases pues por lo general realiza clasificaciones precisas. Consideramos que este algoritmo es bastante sencillo de entender ya que parte de un principio matemático que resulta fácil de aplicar, lo cual lo vuelve una opción viable para estudiar cuando se comienza a trabajar con bancos de datos. Aunque también debemos resaltar algunas desventajas que pudimos notar, la más notoria es que si queremos clasificar un dato en el que sus atributos tienen una escala mayor o menor a los de las clases, el algoritmo genera problemas al realizar dicha clasificación y llega a arrojar resultados incorrectos. Por otra parte, si los datos de las clases están muy revueltos, el algoritmo KNN tendrá problemas para realizar la correcta clasificación. Pero en general, este algoritmo es bastante práctico de utilizar, aunque es importante entender cuando no es una buena opción.

Referencias

[1] ¿Qué es el algoritmo de k vecinos más cercanos? | *IBM*. (n.d.). https://www.ibm.com/mx-es/topics/knn

[2] El algoritmo K-NN y su importancia en el modelado de datos | *Blog* | *Merkle*. (n.d.). Merkle. https://www.merkle.com/es/es/blog/algoritmo-knn-modelado-datos

[3] Gonzalez, L. (2022). K Vecinos más Cercanos – Teoría. *Aprende IA*. https://aprendeia.com/algoritmo-k-vecinos-mas-cercanos-teoria-machine-learning/