Coordinación de
Educación Abierta y a Distancia
VICERRECTORADO ACADÉMICO

Unach
UNIVERSIDAD NACIONAL DE CHIMBORAZO Online

PROGRAMACIÓN 2

Actividad Autónoma 2

Unidad 1: Manipulación y Análisis de Datos con NumPy

Tema 1: Fundamentos de NumPy y Arreglos Numéricos

FACULTAD DE
Ingeniería

Nombres: Lyndon Andrés Castro Vaca

Fecha: 25-05-2025

Carrera: Ciencia de Datos e Inteligencia Artificial

Periodo académico: 2025

Semestre: 2do “B”

Objetivo de la actividad: Diseñar una solución orientada a objetos en Python que implemente desde cero el algoritmo K-Nearest Neighbors (KNN) para clasificar ejemplares del dataset penguins, analizando el efecto de distintos valores de k sobre la asignación de especie.

Recursos o temas que debe haber estudiado antes de hacer la actividad:

• Fundamentos de POO en Python: Abstracción, encapsulamiento, herencia y polimorfismo.

• Manipulación de datos con NumPy

• Buenas prácticas de código: modularidad, getters/setters y docstrings.

Formato de entrega: Jupyter Notebook (.ipynb) — cargado en GitHub Classroom o Sigea

Instrucciones:

Escenario: Clasificación de Pingüinos Antárticos

Un instituto de biología necesita un clasificador que, a partir de mediciones morfológicas de pingüinos, prediga su especie (Adelie, Chinstrap o Gentoo).

1. **Diseño de la solución**

Requerimiento	Detalle
Encapsulamiento	Atributos privados, getters y setters donde sea necesario.
Abstracción	Clase abstracta KNNBase con la firma de los métodos esenciales.
Herencia & Polimorfismo	Subclase concreta KNNClassifier que implementa la lógica específica.
Sobrecarga de operadores	__eq__: Dos ejemplares son iguales si tienen exactamente las mismas medidas. __add__: Permite combinar dos listas de ejemplares en un único conjunto de entrenamiento.

Nota: No se permiten bibliotecas que implementen KNN (por ej. scikit-learn). Solo debes predecir la especie; no calcular métricas de desempeño (accuracy, F1, etc.).

```
1. import math
2. from collections import Counter
3. from abc import ABC, abstractmethod
4.
5. # Defino la clase Penguin para representar un pingüino con sus características
   morfológicas y especie.
```

```
6.  class Penguin:
7.      def __init__(self, bill_length, bill_depth, flipper_length, body_mass,
    species=None):
8.          # Inicializo los atributos privados del pingüino.
9.          self._bill_length = bill_length
10.         self._bill_depth = bill_depth
11.         self._flipper_length = flipper_length
12.         self._body_mass = body_mass
13.         self._species = species
14.
15.     # Defino getters para acceder a los atributos de forma controlada.
16.     @property
17.     def bill_length(self):
18.         return self._bill_length
19.
20.     @property
21.     def bill_depth(self):
22.         return self._bill_depth
23.
24.     @property
25.     def flipper_length(self):
26.         return self._flipper_length
27.
28.     @property
29.     def body_mass(self):
30.         return self._body_mass
31.
32.     @property
33.     def species(self):
34.         return self._species
35.
36.     # Defino setters para modificar los atributos si es necesario.
37.     @bill_length.setter
38.     def bill_length(self, value):
39.         self._bill_length = value
40.
41.     @bill_depth.setter
42.     def bill_depth(self, value):
43.         self._bill_depth = value
44.
45.     @flipper_length.setter
46.     def flipper_length(self, value):
47.         self._flipper_length = value
48.
49.     @body_mass.setter
50.     def body_mass(self, value):
51.         self._body_mass = value
52.
53.     @species.setter
```

```
54.     def species(self, value):
55.         self._species = value
56.
57.     # Sobrecargo el operador de igualdad para comparar pingüinos por sus medidas.
58.     def __eq__(self, other):
59.         if not isinstance(other, Penguin):
60.             return False
61.         return (self.bill_length == other.bill_length and
62.                 self.bill_depth == other.bill_depth and
63.                 self.flipper_length == other.flipper_length and
64.                 self.body_mass == other.body_mass)
65.
66.     # Defino cómo se imprime un pingüino.
67.     def __str__(self):
68.         return f"Penguin(bill_length={self.bill_length}, bill_depth={self.bill_depth},
    flipper_length={self.flipper_length}, body_mass={self.body_mass},
    species={self.species})"
69.
70. # Defino una clase base abstracta para clasificadores KNN.
71. class KNNBase(ABC):
72.     @abstractmethod
73.     def fit(self, training_data):
74.         pass
75.
76.     @abstractmethod
77.     def predict(self, new_data, k=3):
78.         pass
79.
80. # Implemento el clasificador KNN heredando de la clase base.
81. class KNNClassifier(KNNBase):
82.     def __init__(self):
83.         # Inicializo la lista donde guardaré los datos de entrenamiento.
84.         self._training_data = []
85.
86.     # Guardo los datos de entrenamiento que me pasan.
87.     def fit(self, training_data):
88.         self._training_data = training_data
89.
90.     # Predigo la especie de nuevos pingüinos usando el método KNN.
91.     def predict(self, new_data, k=3):
92.         # Verifico que haya datos de entrenamiento.
93.         if not self._training_data:
94.             raise ValueError("El clasificador no ha sido entrenado. Llame al método fit
    primero.")
95.
96.         # Verifico que k sea un número válido.
97.         if k <= 0:
98.             raise ValueError("k debe ser un entero positivo.")
99.
```

```
        predictions = []
        # Para cada pingüino nuevo, calculo la distancia a todos los de entrenamiento.
        for item in new_data:
            distances = []
            for train_item in self._training_data:
                # Calculo la distancia euclidiana entre el nuevo pingüino y cada pingüino de entrenamiento.
                distance = math.sqrt(
                    (item.bill_length - train_item.bill_length)**2 +
                    (item.bill_depth - train_item.bill_depth)**2 +
                    (item.flipper_length - train_item.flipper_length)**2 +
                    (item.body_mass - train_item.body_mass)**2
                )
                # Guardo la distancia junto con la especie del pingüino de entrenamiento.
                distances.append((distance, train_item.species))

            # Ordeno las distancias y selecciono las k más pequeñas (los vecinos más cercanos).
            distances.sort(key=lambda x: x[0])
            k_nearest = distances[:k]

            # Extraigo las especies de los k vecinos más cercanos.
            species_list = [species for (_, species) in k_nearest]

            # Uso Counter para encontrar la especie más común entre los vecinos.
            most_common = Counter(species_list).most_common(1)
            predicted_species = most_common[0][0] if most_common else None

            # Agrego la predicción a la lista de resultados.
            predictions.append(predicted_species)

        return predictions

    # Sobrecargo el operador + para combinar clasificadores y sus datos de entrenamiento.
    def __add__(self, other):
        if not isinstance(other, KNNClassifier):
            raise TypeError("Solo se pueden combinar instancias de KNNClassifier")

        new_classifier = KNNClassifier()
        combined_data = self._training_data + other._training_data
        new_classifier.fit(combined_data)
        return new_classifier

# Ejemplo de uso
if __name__ == "__main__":
    # Creo una lista de pingüinos de entrenamiento, cada uno con sus medidas y especie.
    training_data = [
```

```
        Penguin(39.1, 18.7, 181, 3750, "Adelie"),
        Penguin(39.5, 17.4, 186, 3800, "Adelie"),
        Penguin(40.3, 18.0, 195, 3250, "Adelie"),
        Penguin(46.1, 18.2, 197, 4375, "Chinstrap"),
        Penguin(46.5, 17.9, 196, 3500, "Chinstrap"),
        Penguin(47.6, 18.3, 195, 3850, "Chinstrap"),
        Penguin(49.1, 19.5, 210, 4300, "Gentoo"),
        Penguin(50.0, 20.2, 218, 5700, "Gentoo"),
        Penguin(50.5, 19.8, 215, 5200, "Gentoo")
    ]

    # Creo una lista de pingüinos de prueba (sin especie).
    test_data = [
        Penguin(38.5, 18.2, 180, 3700),
        Penguin(47.0, 18.1, 195, 4000),
        Penguin(49.5, 19.0, 212, 4800)
    ]

    # Instancio el clasificador y lo entreno con los datos de entrenamiento.
    classifier = KNNClassifier()
    classifier.fit(training_data)

    # Hago predicciones sobre los pingüinos de prueba usando k=3 vecinos.
    predictions = classifier.predict(test_data, k=3)

    # Muestro los resultados de las predicciones.
    for i, (penguin, prediction) in enumerate(zip(test_data, predictions)):
        print(f"Pingüino {i+1}:")
        print(f"  Medidas: bill_length={penguin.bill_length},
    bill_depth={penguin.bill_depth}, flipper_length={penguin.flipper_length},
    body_mass={penguin.body_mass}")
        print(f"  Especie predicha: {prediction}\n")

    # Demuestro cómo puedo combinar clasificadores usando el operador +.
    classifier2 = KNNClassifier()
    classifier2.fit(training_data[:3])  # Solo primeros 3 pingüinos

    combined_classifier = classifier + classifier2
    print(f"Número total de ejemplos de entrenamiento en el clasificador combinado:
    {len(combined_classifier._training_data)}")

```

Prueba de escritorio

```
Pingüino 1:
  Medidas: bill_length=38.5, bill_depth=18.2, flipper_length=180, body_mass=3700
  Especie predicha: Adelie

Pingüino 2:
  Medidas: bill_length=47.0, bill_depth=18.1, flipper_length=195, body_mass=4000
  Especie predicha: Adelie

Pingüino 3:
  Medidas: bill_length=49.5, bill_depth=19.0, flipper_length=212, body_mass=4800
  Especie predicha: Gentoo

Número total de ejemplos de entrenamiento en el clasificador combinado: 12
PS C:\Users\andre\OneDrive\Documentos\carteta Py>
```

2. Implementación mínima exigida

Método	Descripción
fit(X, y)	Almacena las observaciones de entrenamiento.
distance(p1, p2)	Devuelve la **distancia Euclidiana** entre dos vectores.
predict(X_new, k=3)	Para cada vector en X_new, retorna la especie según mayoría de los k vecinos más cercanos.
__repr__	Representación legible de un ejemplar.

Prueba la clase con tres valores de k (ej.: 1, 3 y 5) y muestra las predicciones resultantes en pantalla o en una tabla.

```python
import math
from collections import Counter
from abc import ABC, abstractmethod

# Defino la clase Penguin para representar a cada pingüino con sus características.
class Penguin:
    def __init__(self, bill_length, bill_depth, flipper_length, body_mass, species=None):
        # Cuando creo un pingüino, guardo sus medidas y su especie (si la conozco).
        self._bill_length = bill_length
        self._bill_depth = bill_depth
        self._flipper_length = flipper_length
        self._body_mass = body_mass
        self._species = species

```

```
    # Proporciono getters para acceder a cada atributo de forma controlada.
    @property
    def bill_length(self):
        return self._bill_length

    @property
    def bill_depth(self):
        return self._bill_depth

    @property
    def flipper_length(self):
        return self._flipper_length

    @property
    def body_mass(self):
        return self._body_mass

    @property
    def species(self):
        return self._species

    # Defino cómo quiero que se vea el pingüino cuando lo imprimo o lo inspecciono.
    def __repr__(self):
        features = [
            f"bill_length={self.bill_length}",
            f"bill_depth={self.bill_depth}",
            f"flipper_length={self.flipper_length}",
            f"body_mass={self.body_mass}"
        ]
        if self.species:
            features.append(f"species='{self.species}'")
        return f"Penguin({', '.join(features)})"

# Ahora creo el clasificador KNN.
class KNNClassifier:
    def __init__(self):
        # Inicializo las listas donde guardaré los datos de entrenamiento y sus etiquetas.
        self._X = []  # Aquí guardo los pingüinos (características)
        self._y = []  # Aquí guardo las especies (etiquetas)

    def fit(self, X, y):
        """Guardo las observaciones de entrenamiento."""
        # Me aseguro de que haya la misma cantidad de muestras y etiquetas.
        if len(X) != len(y):
            raise ValueError("X e y deben tener la misma longitud")
        self._X = X
        self._y = y

    def distance(self, p1, p2):
```

```
64.         """Calculo la distancia Euclidiana entre dos pingüinos."""
65.         return math.sqrt(
66.             (p1.bill_length - p2.bill_length)**2 +
67.             (p1.bill_depth - p2.bill_depth)**2 +
68.             (p1.flipper_length - p2.flipper_length)**2 +
69.             (p1.body_mass - p2.body_mass)**2
70.         )
71.
72.     def predict(self, X_new, k=3):
73.         """
74.         Para cada pingüino en X_new, predigo su especie usando la mayoría de los k vecinos
    más cercanos.
75.         """
76.         # Me aseguro de que el clasificador ya fue entrenado.
77.         if not self._X:
78.             raise ValueError("El clasificador no ha sido entrenado. Llame al método fit
    primero.")
79.
80.         # Me aseguro de que k sea válido.
81.         if k <= 0 or k > len(self._X):
82.             raise ValueError("k debe ser un entero positivo menor o igual al número de
    muestras de entrenamiento.")
83.
84.         predictions = []
85.         # Para cada pingüino nuevo, hago lo siguiente:
86.         for new_penguin in X_new:
87.             # Calculo la distancia de este pingüino a todos los de entrenamiento.
88.             distances = [
89.                 (self.distance(new_penguin, train_penguin), label)
90.                 for train_penguin, label in zip(self._X, self._y)
91.             ]
92.
93.             # Ordeno las distancias y selecciono las k más cercanas.
94.             distances.sort(key=lambda x: x[0])
95.             k_nearest_labels = [label for (_, label) in distances[:k]]
96.
97.             # Veo cuál especie es la más común entre los vecinos.
98.             most_common = Counter(k_nearest_labels).most_common(1)
99.             predicted_species = most_common[0][0] if most_common else None
100.
101.            # Guardo la predicción.
102.            predictions.append(predicted_species)
103.
104.        return predictions
105.
106.    # Defino cómo quiero que se vea el clasificador cuando lo imprimo.
107.    def __repr__(self):
108.        return f"KNNClassifier(training_samples={len(self._X)})"
109.
```

```python
# Ejemplo de uso con diferentes valores de k
if __name__ == "__main__":
    # Creo los datos de entrenamiento: una lista de pingüinos y sus especies.
    X_train = [
        Penguin(39.1, 18.7, 181, 3750),
        Penguin(39.5, 17.4, 186, 3800),
        Penguin(40.3, 18.0, 195, 3250),
        Penguin(46.1, 18.2, 197, 4375),
        Penguin(46.5, 17.9, 196, 3500),
        Penguin(47.6, 18.3, 195, 3850),
        Penguin(49.1, 19.5, 210, 4300),
        Penguin(50.0, 20.2, 218, 5700),
        Penguin(50.5, 19.8, 215, 5200)
    ]
    y_train = [
        "Adelie", "Adelie", "Adelie",
        "Chinstrap", "Chinstrap", "Chinstrap",
        "Gentoo", "Gentoo", "Gentoo"
    ]

    # Creo los datos de prueba (sin especie, porque quiero predecirla).
    X_test = [
        Penguin(38.5, 18.2, 180, 3700),
        Penguin(47.0, 18.1, 195, 4000),
        Penguin(49.5, 19.0, 212, 4800)
    ]

    # Instancio el clasificador y lo entreno con los datos de entrenamiento.
    classifier = KNNClassifier()
    classifier.fit(X_train, y_train)

    print("Clasificador entrenado:")
    print(classifier)
    print("\nDatos de prueba:")
    for penguin in X_test:
        print(penguin)

    # Pruebo el clasificador con diferentes valores de k.
    for k in [1, 3, 5]:
        print(f"\nPredicciones con k={k}:")
        predictions = classifier.predict(X_test, k=k)
        for i, (penguin, pred) in enumerate(zip(X_test, predictions)):
            print(f"  Pingüino {i+1}: {pred}")
```

Prueba de escritorio

```
Clasificador entrenado:
KNNClassifier(training_samples=9)

Datos de prueba:
Penguin(bill_length=38.5, bill_depth=18.2, flipper_length=180, body_mass=3700)
Penguin(bill_length=47.0, bill_depth=18.1, flipper_length=195, body_mass=4000)
Penguin(bill_length=49.5, bill_depth=19.0, flipper_length=212, body_mass=4800)

Predicciones con k=1:
  Pingüino 1: Adelie
  Pingüino 2: Chinstrap
  Pingüino 3: Gentoo

Predicciones con k=3:
  Pingüino 1: Adelie
  Pingüino 2: Adelie
  Pingüino 3: Gentoo

Predicciones con k=5:
  Pingüino 1: Adelie
  Pingüino 2: Chinstrap
  Pingüino 3: Gentoo
PS C:\Users\andre\OneDrive\Documentos\carteta Py>
```

Reto adicional (1 – 2 pto. extra) (Parcial)

Implementa un método plot_neighbors(x_new, k) que, para un ejemplar nuevo, grafique los k vecinos más cercanos junto con su especie en un plano 2D (tras proyección PCA).

```python
import math
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Defino la clase Penguin para representar a cada pingüino con sus características.
class Penguin:
    def __init__(self, bill_length, bill_depth, flipper_length, body_mass, species=None):
        # Cuando creo un pingüino, guardo sus medidas y su especie (si la conozco).
        self._bill_length = bill_length
        self._bill_depth = bill_depth
        self._flipper_length = flipper_length
        self._body_mass = body_mass
        self._species = species

    # Proporciono getters para acceder a las características de forma controlada.
    @property
    def bill_length(self):
        return self._bill_length

    @property
    def bill_depth(self):
```

```
        return self._bill_depth

    @property
    def flipper_length(self):
        return self._flipper_length

    @property
    def body_mass(self):
        return self._body_mass

    @property
    def species(self):
        return self._species

    # Devuelvo las características como lista para facilitar el uso de PCA.
    def get_features(self):
        return [self.bill_length, self.bill_depth, self.flipper_length, self.body_mass]

    # Defino cómo quiero que se vea el pingüino cuando lo imprimo.
    def __repr__(self):
        features = [
            f"bill_length={self.bill_length}",
            f"bill_depth={self.bill_depth}",
            f"flipper_length={self.flipper_length}",
            f"body_mass={self.body_mass}"
        ]
        if self.species:
            features.append(f"species='{self.species}'")
        return f"Penguin({', '.join(features)})"

# Ahora creo el clasificador KNN.
class KNNClassifier:
    def __init__(self):
        # Inicializo las listas donde guardaré los datos de entrenamiento y sus
etiquetas.
        self._X = []  # Aquí guardo los pingüinos (características)
        self._y = []  # Aquí guardo las especies (etiquetas)

    def fit(self, X, y):
        """Guardo las observaciones de entrenamiento."""
        # Me aseguro de que haya la misma cantidad de muestras y etiquetas.
        if len(X) != len(y):
            raise ValueError("X e y deben tener la misma longitud")
        self._X = X
        self._y = y

    def distance(self, p1, p2):
        """Calculo la distancia Euclidiana entre dos pingüinos."""
        return math.sqrt(
```

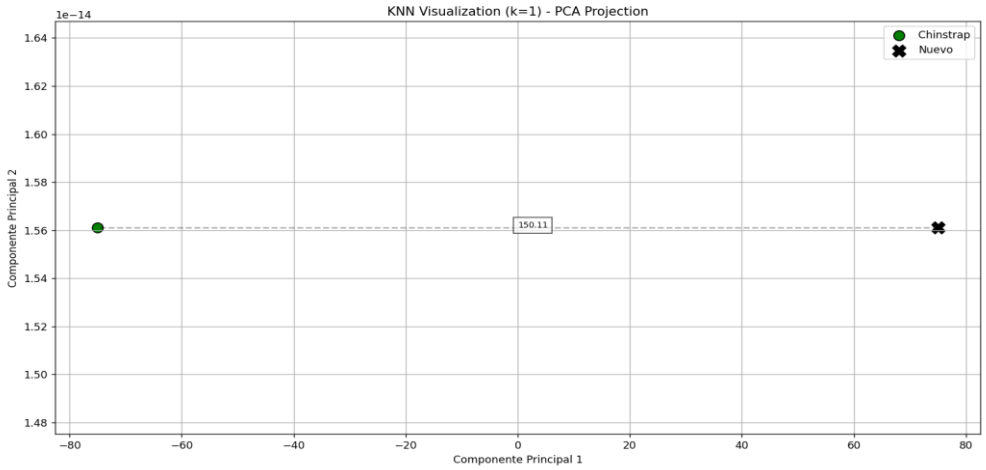
```
            (p1.bill_length - p2.bill_length)**2 +
            (p1.bill_depth - p2.bill_depth)**2 +
            (p1.flipper_length - p2.flipper_length)**2 +
            (p1.body_mass - p2.body_mass)**2
        )

    def predict(self, X_new, k=3):
        """
        Para cada pingüino en X_new, predigo su especie usando la mayoría de los k vecinos más cercanos.
        """
        # Me aseguro de que el clasificador ya fue entrenado.
        if not self._X:
            raise ValueError("El clasificador no ha sido entrenado. Llame al método fit primero.")

        # Me aseguro de que k sea válido.
        if k <= 0 or k > len(self._X):
            raise ValueError("k debe ser un entero positivo menor o igual al número de muestras de entrenamiento.")

        predictions = []
        # Para cada pingüino nuevo, hago lo siguiente:
        for new_penguin in X_new:
            # Calculo la distancia de este pingüino a todos los de entrenamiento.
            distances = [
                (self.distance(new_penguin, train_penguin), label)
                for train_penguin, label in zip(self._X, self._y)
            ]

            # Ordeno las distancias y selecciono las k más cercanas.
            distances.sort(key=lambda x: x[0])
            k_nearest_labels = [label for (_, label) in distances[:k]]

            # Veo cuál especie es la más común entre los vecinos.
            most_common = Counter(k_nearest_labels).most_common(1)
            predicted_species = most_common[0][0] if most_common else None

            # Guardo la predicción.
            predictions.append(predicted_species)

        return predictions

    def plot_neighbors(self, x_new, k=3):
        """
        Grafico los k vecinos más cercanos junto con el nuevo ejemplar
        en un plano 2D usando PCA para la proyección.
        """
        # Me aseguro de que el clasificador ya fue entrenado.
```

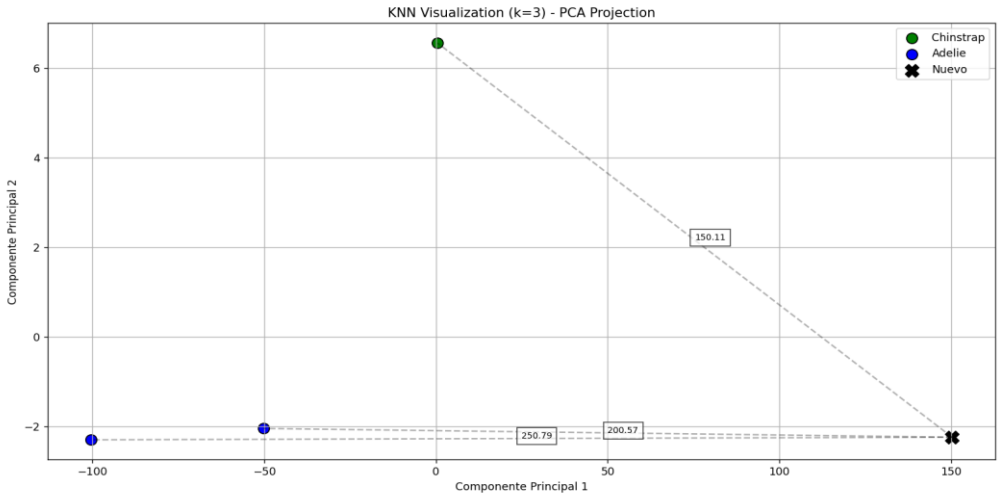
```
        if not self._X:
            raise ValueError("El clasificador no ha sido entrenado. Llame al método fit primero.")

        # Me aseguro de que k sea válido.
        if k <= 0 or k > len(self._X):
            raise ValueError("k debe ser un entero positivo menor o igual al número de muestras de entrenamiento.")

        # Calculo las distancias del nuevo pingüino a todos los de entrenamiento.
        distances = [
            (self.distance(x_new, train_penguin), train_penguin, label)
            for train_penguin, label in zip(self._X, self._y)
        ]
        distances.sort(key=lambda x: x[0])
        k_nearest = distances[:k]

        # Preparo los datos para PCA: los vecinos y el nuevo punto.
        neighbors = [item[1] for item in k_nearest]
        neighbor_labels = [item[2] for item in k_nearest]
        all_penguins = neighbors + [x_new]
        all_labels = neighbor_labels + ["Nuevo"]

        # Obtengo la matriz de características.
        features = [p.get_features() for p in all_penguins]

        # Aplico PCA para reducir a 2 dimensiones.
        pca = PCA(n_components=2)
        reduced = pca.fit_transform(features)

        # Separo las coordenadas.
        x = reduced[:, 0]
        y = reduced[:, 1]

        # Creo el gráfico.
        plt.figure(figsize=(10, 6))

        # Defino colores para cada especie.
        color_map = {
            "Adelie": "blue",
            "Chinstrap": "green",
            "Gentoo": "red",
            "Nuevo": "black"
        }

        # Dibujo cada punto.
        for i, (xi, yi) in enumerate(zip(x, y)):
            species = all_labels[i]
            color = color_map.get(species, "gray")
```

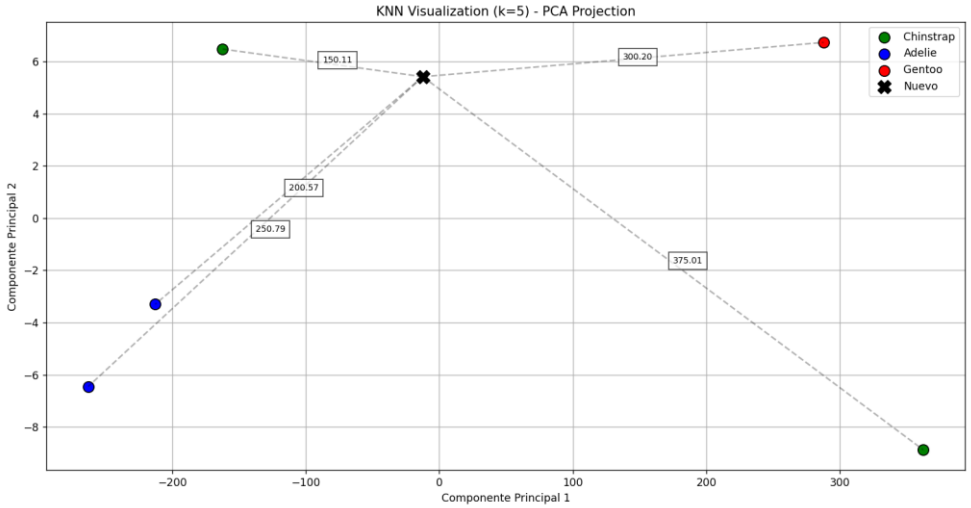
```
164.             marker = "o" if species != "Nuevo" else "X"
165.             size = 100 if species != "Nuevo" else 150
166.             plt.scatter(xi, yi, c=color, label=species if i == all_labels.index(species)
    else "",
167.                         marker=marker, s=size, edgecolors='black')
168.
169.         # Dibujo líneas entre el nuevo punto y sus vecinos.
170.         new_point = reduced[-1]
171.         for neighbor_point in reduced[:-1]:
172.             plt.plot([new_point[0], neighbor_point[0]],
173.                      [new_point[1], neighbor_point[1]],
174.                      'k--', alpha=0.3)
175.
176.         plt.title(f'KNN Visualization (k={k}) - PCA Projection')
177.         plt.xlabel('Componente Principal 1')
178.         plt.ylabel('Componente Principal 2')
179.         plt.legend()
180.         plt.grid(True)
181.
182.         # Muestro la distancia en cada línea.
183.         for i, (dist, penguin, label) in enumerate(k_nearest):
184.             plt.text((reduced[i][0] + new_point[0])/2,
185.                      (reduced[i][1] + new_point[1])/2,
186.                      f"{dist:.2f}", fontsize=8,
187.                      bbox=dict(facecolor='white', alpha=0.7))
188.
189.         plt.show()
190.
191.     # Defino cómo quiero que se vea el clasificador al imprimirlo.
192.     def __repr__(self):
193.         return f"KNNClassifier(training_samples={len(self._X)})"
194.
195.# Ejemplo de uso
196.if __name__ == "__main__":
197.     # Creo los datos de entrenamiento: una lista de pingüinos y sus especies.
198.     X_train = [
199.         Penguin(39.1, 18.7, 181, 3750),
200.         Penguin(39.5, 17.4, 186, 3800),
201.         Penguin(40.3, 18.0, 195, 3250),
202.         Penguin(46.1, 18.2, 197, 4375),
203.         Penguin(46.5, 17.9, 196, 3500),
204.         Penguin(47.6, 18.3, 195, 3850),
205.         Penguin(49.1, 19.5, 210, 4300),
206.         Penguin(50.0, 20.2, 218, 5700),
207.         Penguin(50.5, 19.8, 215, 5200)
208.     ]
209.     y_train = [
210.         "Adelie", "Adelie", "Adelie",
211.         "Chinstrap", "Chinstrap", "Chinstrap",
```

```
        "Gentoo", "Gentoo", "Gentoo"
    ]

    # Instancio el clasificador y lo entreno con los datos de entrenamiento.
    classifier = KNNClassifier()
    classifier.fit(X_train, y_train)

    # Defino un nuevo pingüino para visualizar sus vecinos.
    new_penguin = Penguin(45.0, 18.5, 200, 4000)

    # Visualizo los vecinos para diferentes valores de k.
    for k in [1, 3, 5]:
        classifier.plot_neighbors(new_penguin, k=k)
```

Prueba de escritor






KNN Visualization (k=5) - PCA Projection
Chinstrap
Adelie
Gentoo
Nuevo
150.11
300.20
200.57
250.79
375.01
Componente Principal 2
Componente Principal 1
6
4
2
0
−2
−4
−6
−8
−200
−100
0
100
200
300