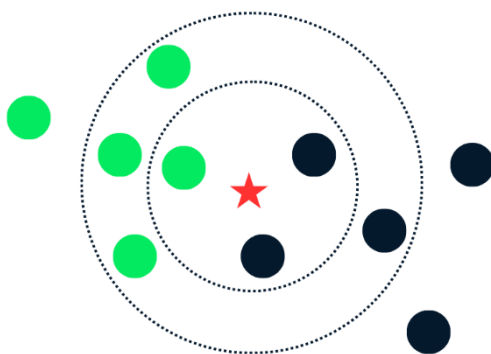




**Centro Universitario de Ciencias Exactas e
Ingenierías**
Universidad de Guadalajara



Práctica 5: kNN
Aprendizaje Máquina



Alumno: Samuel David Pérez Brambila

Código: 222966286

Profesora: Karla Ávila Cárdenas

Sección: D01

Fecha de Entrega: 13 de Octubre de 2024

Introducción

El algoritmo de k vecinos más cercanos, también conocido como KNN o k-NN, es “un clasificador de aprendizaje supervisado no paramétrico, que utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual.” (IBM, s.f.)

Si bien se puede usar para problemas de regresión o clasificación, generalmente se usa como un algoritmo de clasificación, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro. Vale la pena 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.

Y bien, ¿cuál es la clave para el algoritmo KNN? De acuerdo con Elastic (s.f.), la clave es determinar la distancia entre el punto de búsqueda y los otros puntos de datos. Determinar las métricas de distancia posibilita los límites de decisión. Estos límites crean diferentes regiones de puntos de datos. Existen distintos métodos que se usan para calcular la distancia:

- La distancia euclidiana es la medición de distancia más común, la cual mide una línea recta entre el punto de búsqueda y el otro punto que se está midiendo.
- La distancia Manhattan también es una medición de distancia popular, la cual mide el valor absoluto entre dos puntos. Se representa en una cuadrícula y, con frecuencia, se la llama geometría del taxi; ¿cómo se llega del punto A (tu punto de búsqueda) al punto B (el punto que se está midiendo)?
- La distancia de Minkowski es una generalización de las métricas de distancia euclidiana o Manhattan, la cual permite la creación de otras métricas de distancia. Se calcula en un espacio de vectores normalizado. En la distancia de Minkowski, p es el parámetro que define el tipo de distancia usado en el cálculo. Si $p=1$, se usa la distancia Manhattan. Si $p=2$, se usa la distancia euclidiana.
- La distancia de Hamming, también conocida como métrica de superposición, es una técnica usada con vectores de cadena o booleanos para identificar los sitios en los que los vectores no coinciden. En otras palabras, mide la distancia entre dos cadenas de igual longitud. Es particularmente útil para los códigos de corrección de errores y detección de errores.

Otro punto que considerar en este algoritmo es el elegir el mejor valor k (número de vecinos más cercanos considerado), según Elastic (s.f) se debe experimentar con algunos valores para encontrar el valor k que genere las predicciones más exactas con la menor cantidad de errores. Determinar el mejor valor es un acto de balanceo:

- Los valores k bajos hacen predicciones inestables.

Tomando como referencia este ejemplo: un punto de búsqueda está rodeado por dos puntos verdes y un triángulo rojo. Si $k=1$ y el punto más cercano al punto de búsqueda es uno de los puntos verdes, el algoritmo predecirá incorrectamente un punto verde como el resultado de la búsqueda. Los valores k bajos tienen variación alta (el modelo se ajusta demasiado a los datos de entrenamiento), complejidad alta y sesgo bajo (el modelo es lo suficientemente complejo como para ajustarse bien a los datos de entrenamiento).

- Los valores k altos son ruidosos.

Un valor k más alto aumentará la precisión de las predicciones dado que hay más números a partir de los cuales calcular los modos o promedios. Sin embargo, si el valor k es demasiado alto, probablemente dará como resultado una variación baja, complejidad baja y sesgo alto (el modelo no es lo suficientemente complejo para ajustarse bien a los datos de entrenamiento).

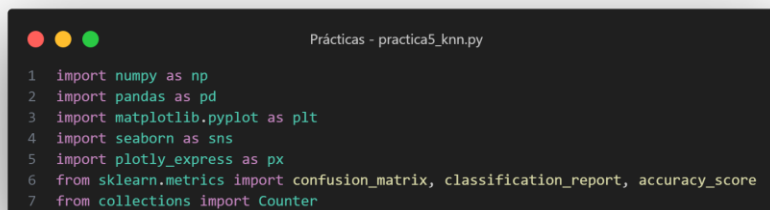
Lo ideal es encontrar un valor k que se encuentre entre variación alta y sesgo alto. También se recomienda elegir un número impar para k , a fin de evitar empates en el análisis de clasificación. El valor k correcto también es relativo al set de datos. Para elegir ese valor, se puede intentar encontrar la raíz cuadrada de N , donde N es el número de puntos de datos en el set de datos de entrenamiento.

En esta práctica, se implementará el algoritmo KNN en el conjunto de datos Penguins de Seaborn, una elección adecuada debido a la simplicidad y estructura clara del dataset para realizar clasificaciones. Para el análisis y visualización, se utilizarán Python y sus bibliotecas populares, como Matplotlib para las visualizaciones, SciKit Learn para realizar la evaluación al modelo y Numpy para el manejo eficiente de los datos.

Contenido de la Actividad

El presente código implementa el algoritmo de k vecinos más cercanos (KNN) sobre el dataset "Penguins" de Seaborn. En este caso, se utilizan las características "bill_length_mm" (que mide la longitud del pico en milímetros, relacionada con el tamaño del pico del pingüino), "bill_depth_mm" (que mide la profundidad del pico en milímetros, asociada al grosor del pico), "flipper_length_mm" (que mide la longitud de la aleta en milímetros y está relacionada con el tamaño del pingüino), y "body_mass_g" (que mide la masa corporal en gramos y está asociada al peso del pingüino) como variables independientes. Estas características permiten predecir la especie del pingüino, la cual es la variable dependiente. El algoritmo KNN clasifica cada pingüino basándose en las características de sus vecinos más cercanos, seleccionados en función de las distancias calculadas, para determinar la especie a la que pertenece. La implementación también explora diferentes valores de k para encontrar el más adecuado, logrando un equilibrio entre la variación y el sesgo en la clasificación.

Importación de bibliotecas

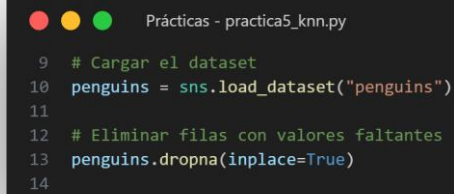


```
Prácticas - practica5_knn.py
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import plotly.express as px
6 from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
7 from collections import Counter
```

Se importan las bibliotecas necesarias:

- Numpy: Para realizar operaciones numéricas y manipulación de arrays.
- Pandas: Para gestionar datos en forma de DataFrames.
- Matplotlib y Seaborn: Para visualización de datos.
- Plotly Express: Para visualizaciones interactivas en 3D.
- Scikit-learn: Para métricas de evaluación del modelo como matriz de confusión y reporte de clasificación.
- Counter: Para contar la frecuencia de los vecinos más cercanos en el algoritmo KNN.

Carga y preprocesamiento del dataset

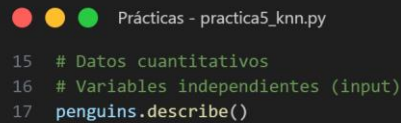


```
Prácticas - practica5_knn.py

9 # Cargar el dataset
10 penguins = sns.load_dataset("penguins")
11
12 # Eliminar filas con valores faltantes
13 penguins.dropna(inplace=True)
14
```

- Se carga el dataset de pingüinos usando seaborn.
- Se eliminan las filas con valores faltantes (NaN) para evitar errores en el análisis.

Análisis de datos cuantitativos

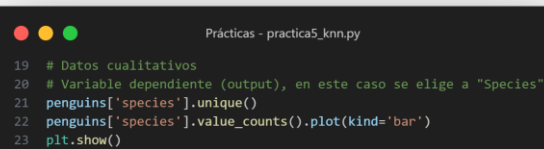


```
Prácticas - practica5_knn.py

15 # Datos cuantitativos
16 # Variables independientes (input)
17 penguins.describe()
```

- describe() genera estadísticas descriptivas (como media, desviación estándar, etc.) para las variables numéricas del dataset, en pocas palabras se utiliza principalmente para visualizar cuáles son las variables o datos numéricos del dataset.

Análisis de datos cualitativos



```
Prácticas - practica5_knn.py

19 # Datos cualitativos
20 # Variable dependiente (output), en este caso se elige a "Species"
21 penguins['species'].unique()
22 penguins['species'].value_counts().plot(kind='bar')
23 plt.show()
```

- Se examinan los valores únicos de la variable objetivo species (especies de pingüinos).
- value_counts() cuenta cuántas veces aparece cada especie en el dataset y lo grafica en un gráfico de barras.
- plt.show() sirve para asegurar que se muestre la ventana con el gráfico de barras generado.

Preprocesamiento de datos (conversión de categorías a valores numéricos)

1. Conversión de la variable 'sex'

```
Prácticas - practica5_knn.py
25 # Convertir la variable 'sex' a valores numéricos
26 penguins['sex'] = penguins['sex'].map({'Male': 0, 'Female': 1})
```

La columna sex (sexo) se convierte a valores numéricos: 0 para 'Male' y 1 para 'Female'. Esto es necesario para que el algoritmo pueda procesar esta variable categórica.

2. Conversión de la variable 'island' usando codificación de etiquetas

```
Prácticas - practica5_knn.py
28 # Convertir la variable 'island' usando codificación de etiquetas
29 penguins['island'] = penguins['island'].astype('category').cat.codes
```

La columna island (isla) se convierte a valores numéricos asignando un código único a cada isla mediante cat.codes. Esto también es parte del preprocesamiento para manejar variables categóricas.

Mapa de correlación

```
Prácticas - practica5_knn.py
31 # Mapa de correlación
32 penguins_numeric = penguins.select_dtypes(include=[np.number])
33 sns.heatmap(penguins_numeric.corr(), square=True, annot=True)
34 plt.title('Mapa de correlación')
35 plt.show()
```

- Se seleccionan las columnas numéricas con `select_dtypes(include=[np.number])`, lo cual ya incluye las variables 'sex' e 'island' ya que ya se convirtieron a numérico asignando un número identificador por el dato contenido, por ejemplo con sex si es male se le asigna 0 y female 1.
- Se genera un heatmap (mapa de calor) para visualizar la correlación entre variables numéricas. La correlación alta indica una relación fuerte entre variables, lo cual puede ser útil para el análisis de características. Es así, que podemos decidir cuáles son las 4 variables más importantes.

Visualización 3D de las 4 variables más importantes

```
Prácticas - practica5_knn.py
37 # Graficar las 4 variables independientes más importantes
38 fig = px.scatter_3d(penguins,
39                     x="body_mass_g",
40                     y="flipper_length_mm",
41                     z="bill_length_mm",
42                     size="bill_depth_mm",      # Profundidad del pico (cuarta variable representada por el tamaño de los puntos)
43                     color="species",          # Colorear según la especie del pingüino
44                     color_discrete_map = {"Joly": "blue", "Bergeron": "violet", "Coderre": "pink"})
45
46 fig.show()
```

Se utiliza Plotly Express para crear un gráfico interactivo en 3D con las siguientes características:

- x: masa corporal del pingüino.
- y: longitud de las aletas.
- z: longitud del pico.
- size: el tamaño de los puntos representa la profundidad del pico.
- color: los puntos se colorean según la especie de pingüino.
- La opción `color_discrete_map` define los colores para cada especie.

Implementación del algoritmo KNN

1. Clase KNN

```
Prácticas - practica5_knn.py
48 class KNN:
49     def __init__(self, k=3):
50         self.k = k
```

Se define una clase KNN con un parámetro `k` que representa el número de vecinos más cercanos.

2. Función de distancia euclidiana

```
Prácticas - practica5_knn.py
52 # Función para calcular la distancia euclidiana
53 def euclidean_distance(self, x1, x2):
54     return np.sqrt(np.sum((x1 - x2)**2))
```

Calcula la distancia euclidiana entre dos puntos, la cual mide la cercanía entre ellos en el espacio n-dimensional.

3. Método fit para entrenar el modelo

```
Prácticas - practica5_knn.py

56 # Método para entrenar el modelo
57 def fit(self, X_train, y_train):
58     self.X_train = X_train
59     self.y_train = y_train
```

Almacena los datos de entrenamiento X_train y y_train en la clase.

4. Método predict para realizar predicciones

```
Prácticas - practica5_knn.py

61 # Método para realizar predicciones
62 def predict(self, X_test):
63     predictions = []
64     for test_point in X_test:
65         # Calcular la distancia entre el punto de prueba y todos los puntos de entrenamiento
66         distances = [self.euclidean_distance(test_point, x_train) for x_train in self.X_train]
67         # Ordenar los índices de las distancias más cercanas
68         k_indices = np.argsort(distances)[:self.k]
69         # Obtener las etiquetas de los vecinos más cercanos
70         k_nearest_labels = [self.y_train[i] for i in k_indices]
71         # Asignar la clase más común entre los vecinos más cercanos
72         most_common = Counter(k_nearest_labels).most_common(1)[0][0]
73         predictions.append(most_common)
74     return predictions
```

- Para cada punto de prueba, calcula las distancias euclidianas a todos los puntos de entrenamiento.
- Encuentra los k vecinos más cercanos (menores distancias) y asigna la etiqueta más común entre ellos como predicción.

División de datos de entrenamiento y prueba

```
Prácticas - practica5_knn.py

76 # Función para dividir los datos de entrenamiento y prueba
77 def train_test_split(self, X, y, test_size=0.3, random_state=42):
78     if random_state is not None:
79         np.random.seed(random_state)
80     # Mezclar los datos aleatoriamente
81     indices = np.random.permutation(len(X))
82     test_size = int(len(X) * test_size)
83     # Dividir los datos
84     test_indices = indices[:test_size]
85     train_indices = indices[test_size:]
86     return X[train_indices], X[test_indices], y[train_indices], y[test_indices]
```

Divide los datos en entrenamiento y prueba usando un test_size del 30%. También mezcla los datos aleatoriamente antes de dividirlos.

Búsqueda del mejor valor de k

```
Prácticas - practica5_knn.py

88 # Función para encontrar el mejor número de vecinos (k)
89 def find_best_k(self, X_train, y_train, X_test, y_test, max_k=15):
90     train_accuracy = []
91     test_accuracy = []
92     for k in range(1, max_k + 1):
93         self.k = k
94         # Entrenar el modelo con los datos de entrenamiento
95         self.fit(X_train, y_train)
96
97         # Predecir para los datos de entrenamiento y prueba
98         y_train_pred = self.predict(X_train) # Solo X_train
99         y_test_pred = self.predict(X_test)   # Solo X_test
100
101         # Calcular la precisión para los datos de entrenamiento y prueba
102         train_acc = accuracy_score(y_train, y_train_pred)
103         test_acc = accuracy_score(y_test, y_test_pred)
104
105         train_accuracy.append(train_acc)
106         test_accuracy.append(test_acc)
107
108     # Graficar la precisión en función de k
109     plt.plot(range(1, max_k + 1), train_accuracy, label='Training Accuracy')
110     plt.plot(range(1, max_k + 1), test_accuracy, label='Test Accuracy')
111     plt.xlabel('n_neighbors')
112     plt.ylabel('Accuracy')
113     plt.title('Accuracy vs. n_neighbors')
114     plt.legend()
115     plt.show()
```

Prueba diferentes valores de k (de 1 a 15) para encontrar el número óptimo de vecinos, calculando la precisión en cada iteración (`accuracy_score`). Graficando la precisión en el conjunto de entrenamiento y prueba para cada valor de k, para así poder visualizar de manera gráfica (`plt.plot()`) cuál sería el mejor valor de k.

Validación cruzada

```
Prácticas - practica5_knn.py

117 # Función para realizar validación cruzada
118 def cross_validate(self, X, y, folds=5):
119     fold_size = len(X) // folds
120     scores = []
121     for i in range(folds):
122         # Crear los conjuntos de entrenamiento y prueba para cada pliegue
123         X_train = np.concatenate([X[:i * fold_size], X[(i + 1) * fold_size:]], axis=0)
124         y_train = np.concatenate([y[:i * fold_size], y[(i + 1) * fold_size:]], axis=0)
125         X_test = X[i * fold_size:(i + 1) * fold_size]
126         y_test = y[i * fold_size:(i + 1) * fold_size]
127
128         # Entrenar el modelo con los datos de entrenamiento
129         self.fit(X_train, y_train)
130
131         # Realizar predicciones sobre los datos de prueba
132         y_pred = self.predict(X_test)
133
134         # Calcular la precisión y añadirla a la lista de puntuaciones
135         score = accuracy_score(y_test, y_pred)
136         scores.append(score)
137
138     # Mostrar las precisiones por pliegue y la media
139     print(f'Precisión por cada iteración: {scores}')
140     print(f'Precisión promedio: {np.mean(scores)}')
```

Realiza validación cruzada dividiendo los datos en 5 pliegues (folds) para evaluar la precisión del modelo.

Matriz de confusión

```
Prácticas - practica5_knn.py
142 # Función para graficar la matriz de confusión
143 def plot_confusion_matrix(self, y_true, y_pred):
144     cm = confusion_matrix(y_true, y_pred)
145     plt.figure(figsize=(6, 4))
146     sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', cbar=False,
147                xticklabels=np.unique(y_true), yticklabels=np.unique(y_true))
148     plt.xlabel('Predicho')
149     plt.ylabel('Verdadero')
150     plt.title('Matriz de Confusión')
151     plt.show()
```

Crea una matriz de confusión para evaluar visualmente el rendimiento del modelo, mostrando qué tan bien clasificó cada clase.

Carga de datos

```
Prácticas - practica5_knn.py
153 # Cargar los datos
154 X = penguins[['body_mass_g', 'flipper_length_mm', 'bill_length_mm', 'bill_depth_mm']].values
155 y = penguins['species'].values
```

- X: Aquí se seleccionan las columnas que contienen las variables independientes del conjunto de datos de los pingüinos. Estas variables son `body_mass_g`, `flipper_length_mm`, `bill_length_mm`, y `bill_depth_mm`. El método `.values` convierte estas columnas en un arreglo de NumPy, que es el formato requerido para las operaciones matemáticas en el algoritmo KNN.
- y: Esta variable representa la clase de salida, es decir, la especie de los pingüinos (`species`). También se convierte en un arreglo de NumPy con `.values`, lo que facilita el uso en el modelo.

División de datos e instanciación de KNN con un valor k de 13

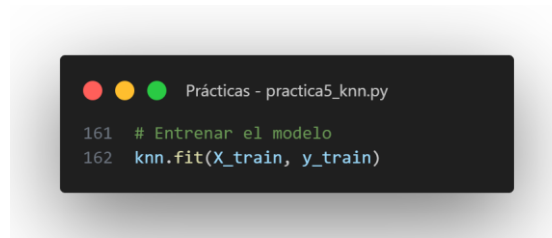
```
Prácticas - practica5_knn.py
157 # Dividir los datos en entrenamiento y prueba
158 knn = KNN(k=13)
159 X_train, X_test, y_train, y_test = knn.train_test_split(X, y)
```

- Instancia de KNN: Se crea una instancia de la clase KNN con `k=13`, lo que indica que se usarán los 13 vecinos más cercanos para las predicciones, este

valor se tomó en base al gráfico de búsqueda del mejor k (que se ve más adelante).

- División de Datos: Se llama al método `train_test_split` de la clase KNN para dividir el conjunto de datos en entrenamiento y prueba. Este método mezcla aleatoriamente los datos y divide el 70% para el entrenamiento y el 30% para la prueba.

Entrenamiento del modelo



```
Prácticas - practica5_knn.py
161 # Entrenar el modelo
162 knn.fit(X_train, y_train)
```

Se llama al método `fit`, pasando `X_train` y `y_train`. Esto almacena los datos de entrenamiento dentro de la instancia de KNN para que puedan ser utilizados posteriormente durante la predicción.

Predicciones



```
Prácticas - practica5_knn.py
164 # Realizar predicciones
165 y_pred = knn.predict(X_test)
```

Se llama al método `predict` con el conjunto de prueba `X_test`. Este método calcula las distancias entre cada punto en `X_test` y los puntos en `X_train`, determina los vecinos más cercanos y asigna la clase más común entre ellos a `y_pred`.

Validación cruzada con K-folds

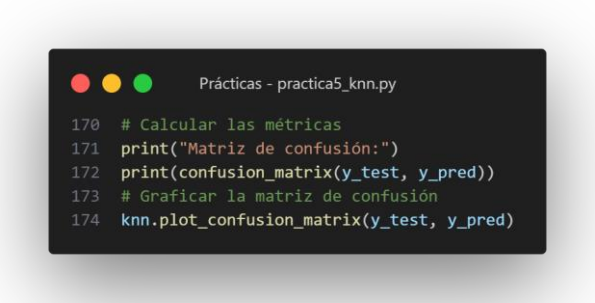


```
Prácticas - practica5_knn.py
167 # Realizar validación cruzada
168 knn.cross_validate(X, y, folds=5)
```

Se llama al método `cross_validate`, que divide el conjunto completo en 5 pliegues (folds) para evaluar el modelo. En cada iteración, el modelo se entrena con 4 pliegues y se prueba con el pliegue restante. Esto ayuda a asegurar que el modelo

se desempeñe bien en datos no vistos y proporciona una evaluación más confiable de su rendimiento.

Cálculo y visualización de métricas




```
Prácticas - practica5_knn.py

170 # Calcular las métricas
171 print("Matriz de confusión:")
172 print(confusion_matrix(y_test, y_pred))
173 # Graficar la matriz de confusión
174 knn.plot_confusion_matrix(y_test, y_pred)
```

- Matriz de Confusión: Se imprime la matriz de confusión que muestra el rendimiento del modelo al comparar las predicciones (`y_pred`) con las etiquetas verdaderas (`y_test`). La matriz de confusión proporciona una visión clara de cuántas predicciones fueron correctas y cuántas incorrectas para cada clase.
- Gráfico de la Matriz de Confusión: Se llama al método `plot_confusion_matrix` para graficar la matriz de confusión usando seaborn. Esto ayuda a visualizar mejor cómo el modelo clasifica cada especie de pingüino.

Informe de clasificación (precision, recall y accuracy)



```
Prácticas - practica5_knn.py

176 print("\nClassification Report:")
177 print(classification_report(y_test, y_pred))
```

Se imprime un informe de clasificación que incluye métricas como precisión, recall y F1-score para cada clase (especie de pingüino), además del accuracy (precisión) del modelo. Estas métricas son útiles para evaluar el rendimiento del modelo de manera más detallada, especialmente en problemas de clasificación donde las clases pueden estar desbalanceadas.

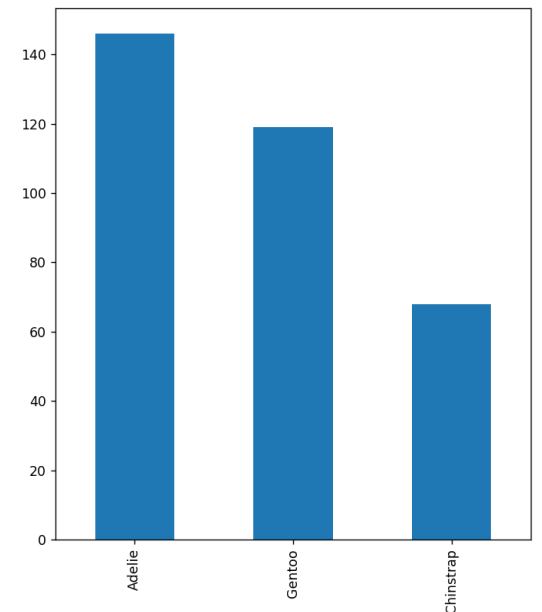
Llamada al método de buscar el mejor k

```
Prácticas - practica5_knn.py
179 # Encontrar el mejor número de vecinos (k)
180 knn.find_best_k(X_train, y_train, X_test, y_test)
```

Se llama al método `find_best_k`, que evalúa diferentes valores de `k` para encontrar el que produce la mejor precisión en los datos de prueba. Este método genera un gráfico que muestra cómo varía la precisión en función del número de vecinos considerados. Esta optimización es crucial para asegurar que el modelo KNN no sea ni demasiado simple ni demasiado complejo.

Es así que obtenemos los siguientes resultados:

Veces que aparece cada especie en el dataset representado en gráfico de barras



Mapa de correlación

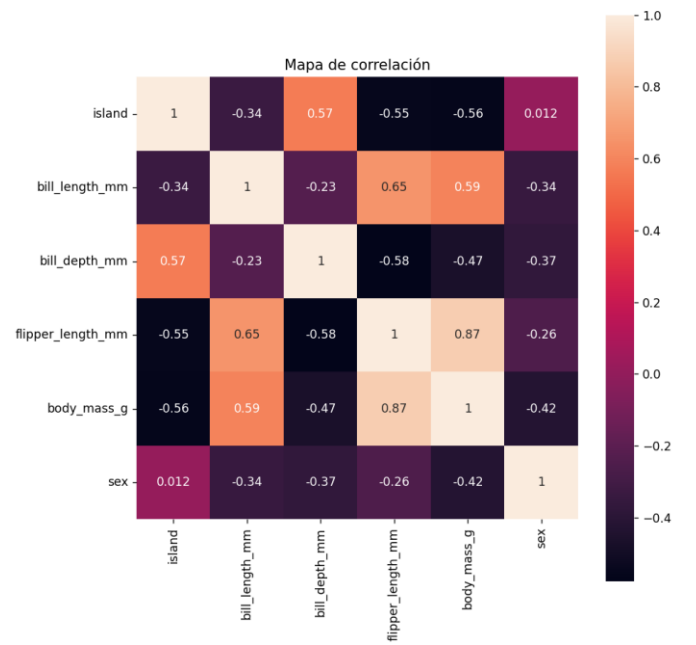
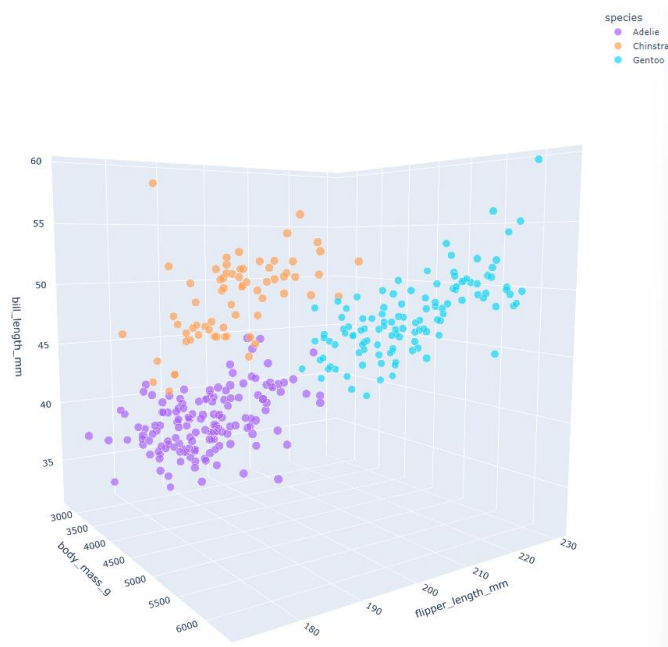


Gráfico 3D de las variables más importantes



Matriz de confusión (gráfico)

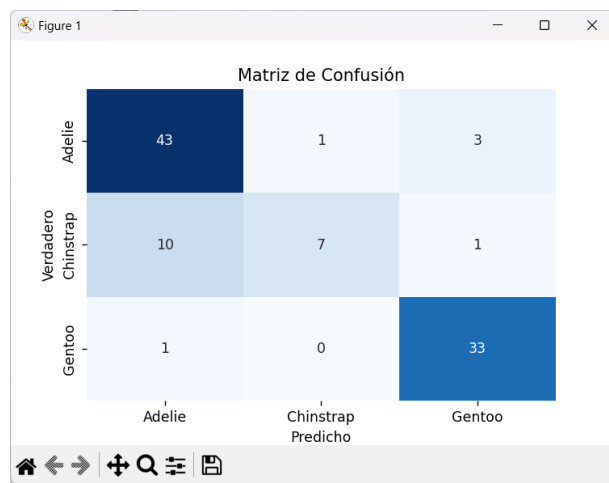
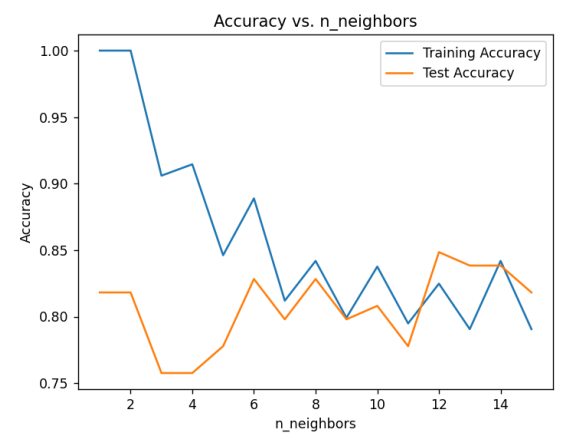


Gráfico de búsqueda del mejor k



Matriz de confusión e informe de clasificación con precision, recall, f1-score de cada clase, así como accuracy del modelo

```
Matriz de confusión:
[[43  1  3]
 [10  7  1]
 [ 1  0 33]]

Classification Report:
              precision    recall  f1-score   support

   Adelie       0.80      0.91      0.85         47
  Chinstrap     0.88      0.39      0.54         18
    Gentoo     0.89      0.97      0.93         34

 accuracy              0.84         99
  macro avg       0.85      0.76      0.77         99
 weighted avg     0.84      0.84      0.82         99
```

Conclusiones

Al finalizar la práctica de clasificación utilizando el algoritmo K-Nearest Neighbors (KNN) en Python, se puede afirmar que es una herramienta eficaz y accesible para resolver problemas de clasificación multicategórica.

Durante el proceso, se subrayó la importancia de una adecuada preparación de los datos, como la eliminación de valores faltantes y la conversión de variables categóricas a un formato numérico, estas etapas son fundamentales para garantizar que el modelo funcione de manera óptima. Además, se destacó la necesidad de seleccionar métricas de evaluación apropiadas, como la matriz de confusión y el informe de clasificación, que proporcionaron una visión clara del desempeño del modelo en términos de verdaderos positivos, falsos positivos y otras métricas relevantes como la exactitud (accuracy), precisión (precision) y recall.

La capacidad de realizar validación cruzada con k-folds y ajustar el número de vecinos (k) permitió optimizar el modelo y evitar problemas de sobreajuste, lo que es crucial para mejorar la generalización de KNN en datos no vistos. También se evidenció la utilidad de visualizar el rendimiento a través de gráficos, lo que facilitó la comprensión de cómo el modelo clasifica cada especie de pingüino en el conjunto de datos.

En resumen, esta práctica no solo ha proporcionado experiencia en la implementación del algoritmo KNN, sino que también ha enriquecido la comprensión sobre cómo los algoritmos de aprendizaje automático pueden ser aplicados efectivamente utilizando herramientas o librerías en Python. La experiencia ha sido valiosa al integrar teoría y práctica, estableciendo una base sólida para trabajos futuros en el ámbito de la ciencia de datos y el aprendizaje automático.

Referencias

- Elastic. (s.f.). *¿Qué es kNN?*. Elastic. Recuperado el 03 de Octubre de 2024 de: <https://www.elastic.co/es/what-is/knn>
- IBM. (s.f.). *¿Qué es KNN?*. IBM. Recuperado el 03 de Octubre de 2024 de: <https://www.ibm.com/mx-es/topics/knn>