K-Nearest Neighbors (KNN) Model for Penguin Species Classification

NAME- Ankita Satapathy
ROLL NO- 22053400

1. Introduction

This project implements a K-Nearest Neighbors (KNN) model to classify penguin species based on features such as bill length, bill depth, flipper length, and body mass. The model is built using Python and includes a FastAPI-based backend along with an HTML front-end for user interaction.

2. Files in the Submission

i. **KNN3400.ipynb**: Implements a KNN classification model using custom logic in Jupyter Notebook and also contains the model training process.
ii. **KNN3400.py**: Python script containing the KNN implementation for backend integration. Generated in the vscode terminal using command-

```
PS C:\Users\KIIT\Desktop\22053400\3) KNN> jupyter nbconvert --to script KNN3400.ipynb
```

iii. **main.py**: Defines the FastAPI server to handle prediction requests.
iv. **custom_knn_penguins_model.pkl**: The trained model serialized using pickle.
v. **index.html**: Front-end UI for user input and displaying predictions.

3. Installation and Setup

i. Prerequisites :- Required Python packages: Numpy, scikit-learn, fastapi, pickle, uvicorn.

   - NumPy (*numpy)* is used for numerical computations and matrix operations in model training.
   - Scikit-learn (*sklearn*) is used for loading the dataset and provides ML utilities.
   - FastAPI (*fastapi*) is used to create a web API for serving predictions.
   - Pickle (*pickle*) is used to save and load the trained model for reuse.
   - Uvicorn (*uvicorn*) is used to run the FastAPI server asynchronously.

ii. Installing Dependencies:- The following command is used in vscode terminal to install dependencies

```
PS C:\Users\KIIT\Desktop\22053400\3) KNN> pip install fastapi uvicorn numpy pandas scikit-learn
```

4. Model Implementation

The CustomKNN class in KNN3400.py implements the KNN algorithm: Computes Euclidean distances between the test point and all training points, Selects the k nearest neighbors based on distance and Uses majority voting to classify the test instance.

```python
class CustomKNN:
    def __init__(self, k=5):
        self.k = k
        self.X_train = None
        self.y_train = None

    def fit(self, X_train, y_train):
        """Store training data"""
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        """Predict the class for each test sample"""
        predictions = [self._predict(x) for x in X_test]
        return np.array(predictions)

    def _predict(self, x):
        """Helper function to predict a single sample"""
        distances = [np.linalg.norm(x - x_train) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_labels).most_common(1)
        return most_common[0][0]
```

5. Training the Model

- The dataset used is the Penguins dataset from Seaborn.
- Features include bill length, bill depth, flipper length, and body mass.
- The model is trained on 80% of the data and tested on 20%.
- The trained model is serialized and saved as custom_knn_penguins_model.pkl using Pickle.

6. API Implementation

The FastAPI-based backend (main.py) loads the trained model and provides an endpoint to make predictions:

i. API Setup

```python
app = FastAPI()
```

ii. CORS Configuration- To allow front-end requests:

```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)
```

iii. Loading the Trained Model

```python
with open("custom_knn_penguins_model.pkl", "rb") as f:
    model = pickle.load(f)
```

iv. Defining the API Endpoint

```python
@app.post("/predict/")
async def predict(data: InputData):
    features = np.array(data.features).reshape(1, -1)
    prediction = model.predict(features)[0]
    return {"prediction": prediction}

```

7. Running the Application (FastAPI Server)

The following command is used in vscode to start the API server:

```
PS C:\Users\KIIT\Desktop\22053400\3) KNN> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\KIIT\\Desktop\\22053400\\3) KNN']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [19272] using WatchFiles
```

8. Front-End Implementation

The index.html file provides a simple UI for users to input feature values and get predictions using the API. It has inline CSS and JavaScript. The JavaScript function sends a request to the FastAPI backend.

```javascript
async function predict() {
    let features = [];
    for (let i = 0; i < 4; i++) {
        let value = document.getElementById("feature" + i).value;
        if (value === "" || isNaN(value)) {
            document.getElementById("result").innerText = "⚠️ Please enter valid numbers!";
            return;
        }
        features.push(parseFloat(value));
    }
    try {
        let response = await fetch("http://127.0.0.1:8000/predict/", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ features: features })
        });
        if (!response.ok) {
            throw new Error("🚨 Server error! Try again.");
        }
        let data = await response.json();
        document.getElementById("result").innerText = "🎉 Predicted Penguin Species: " + data.prediction;
    } catch (error) {
        console.error("Error:", error);
        document.getElementById("result").innerText = "❌ Error: " + error.message;
    }
}
```

9. Conclusion

This project successfully implements a K-Nearest Neighbors (KNN) classifier to predict penguin species based on feature inputs. The model is integrated with a FastAPI backend. NumPy is used for numerical computations, Scikit-learn for dataset handling, and Pickle for saving the trained model. The application is deployed using Uvicorn, providing a seamless experience for users.

The frontend layout features a centered card with a white background, rounded corners, and a shadow effect. At the top, a bold blue header displays the title *"Penguin Species Classifier", followed by a brief description. Below, a user input section allows entry of bill length, bill depth, flipper length, and body mass, along with a Predict button that returns the predicted species.