KNN Model Integrated with Website using FastAPI

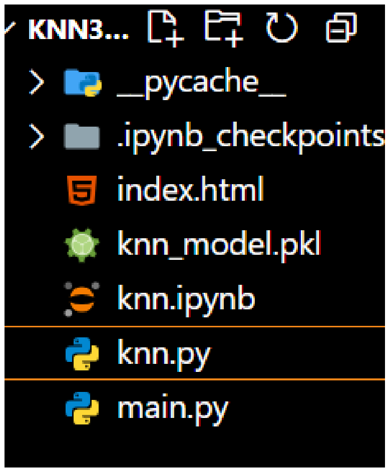
Name: Barsha Baibhabi
Roll No: 22053416

1. Overview

This project implements a K-Nearest Neighbors (KNN) model trained on the Iris dataset and integrates it with a FastAPI backend and a simple HTML frontend. The user inputs feature values through the frontend, which sends the data to the backend for prediction using the trained model.

Files in the Project:

1. **knn.ipynb** - Jupyter Notebook for training the KNN model and saving it as a pickle file.
2. **main.py** - FastAPI backend to load the trained model and provide a prediction endpoint.
3. **index.html** - Frontend with inline CSS & JavaScript for user input and displaying predictions.
4. **knn.py** - TPython script version of the Jupyter Notebook.
5. **Knn_model.pkl -** Saved Naive Bayes model.



---

2. Installation & Setup

Prerequisites:

Ensure you have Python installed. Install the required dependencies using:

```
pip install fastapi uvicorn scikit-learn numpy pandas
```

Running the Backend:

1. Train the model using **KNN_Model.ipynb** and generate `knn_model.pkl`.
2. Run the FastAPI backend:

   ```
   uvicorn main:app --reload
   ```

3. The API will be accessible at: `http://127.0.0.1:8000`

---

3. Training the Model & Generating Pickle File

The KNN_Model.ipynb notebook:

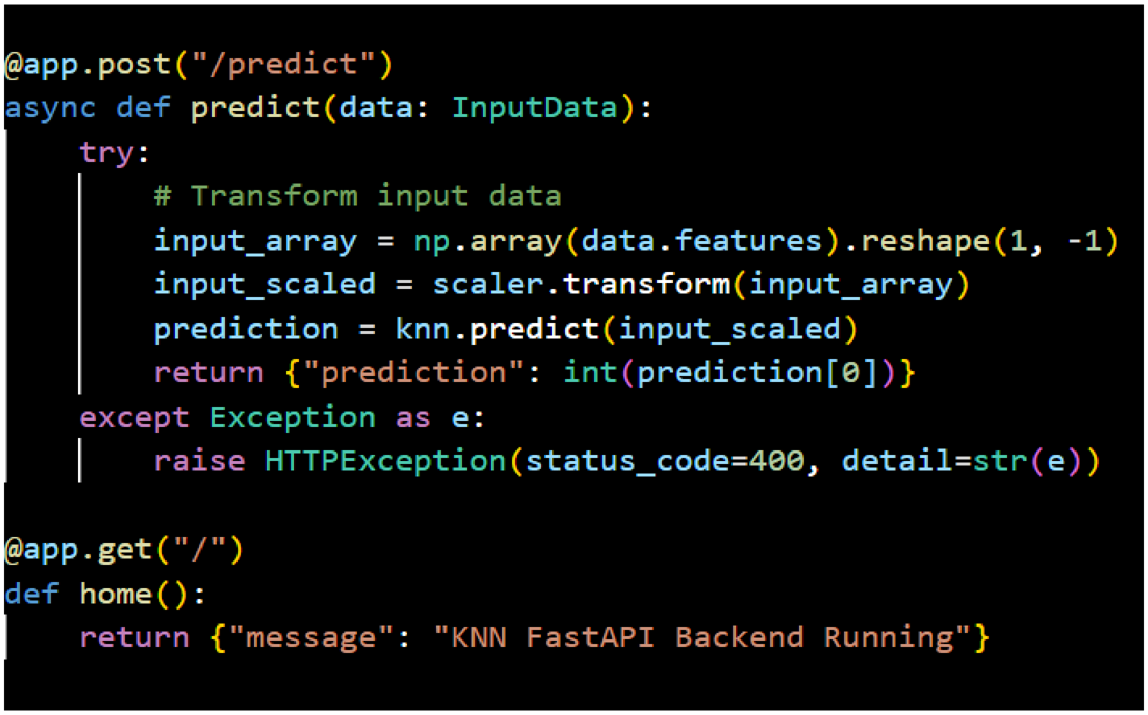
- Loads the Iris dataset from `sklearn`.
- Splits the data into training and test sets.
- Standardizes the features using `StandardScaler`.
- Trains a K-Nearest Neighbors (KNN) classifier.
- Saves the trained model and scaler to a `knn_model.pkl` file using `pickle`.

---

4. FastAPI Backend (`main.py`)

The **main.py** script:

- Loads `knn_model.pkl`.
- Defines a `/predict` endpoint that accepts input features and returns a prediction.



```python
@app.post("/predict")
async def predict(data: InputData):
    try:
        # Transform input data
        input_array = np.array(data.features).reshape(1, -1)
        input_scaled = scaler.transform(input_array)
        prediction = knn.predict(input_scaled)
        return {"prediction": int(prediction[0])}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

@app.get("/")
def home():
    return {"message": "KNN FastAPI Backend Running"}
```

- Uses `Pydantic` for input validation.

Run it using: `uvicorn main:app --reload`

---

5. Frontend (`index.html`)

The **index.html** file:

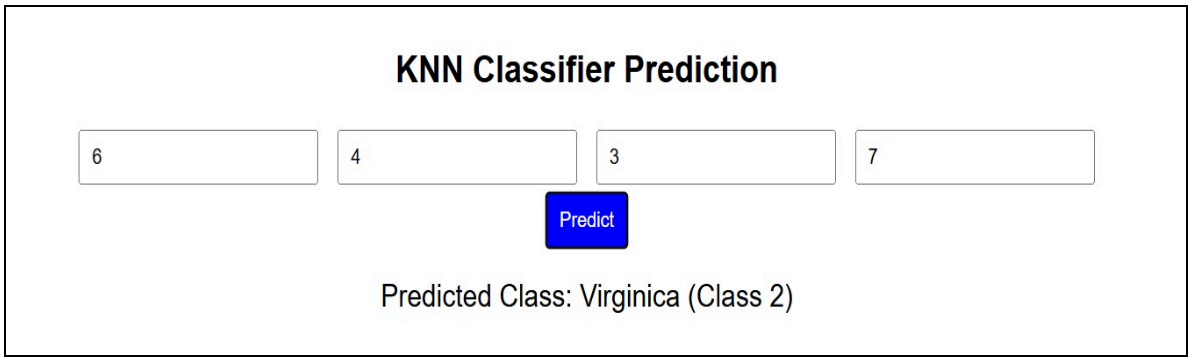
- Provides an input form for users to enter feature values.
- Sends a request to the FastAPI backend.

Code Sniped

```javascript
let response = await fetch("http://127.0.0.1:8000/predict", {

               method: "POST",

               headers: { "Content-Type": "application/json" },

               body: JSON.stringify({ features: features })
```

```
            });

            let result = await response.json();

            let prediction = result.prediction;

            const classNames = {

                0: "Setosa",

                1: "Versicolor",

                2: "Virginica"

            };

            document.getElementById("result").innerText = "Predicted
Class: " + classNames[prediction] + " (Class " + prediction + ")";

        });
```

- Displays the predicted class name (Setosa, Versicolor, or Virginica).



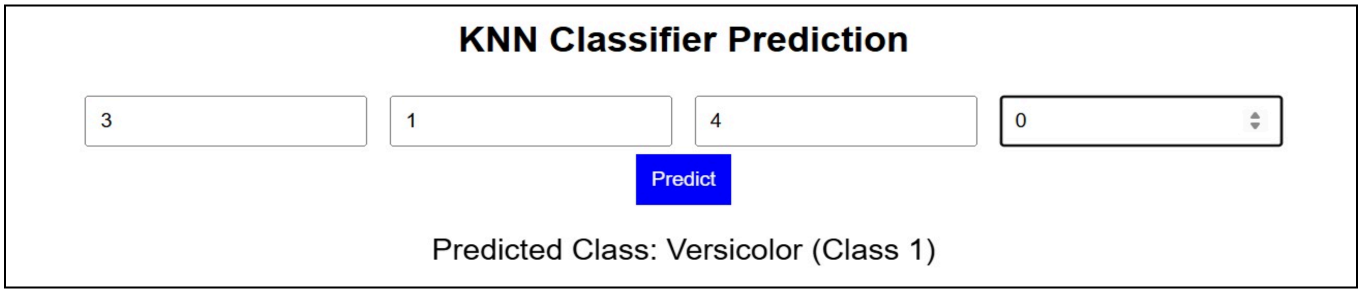
Predicted Class: Virginica (Class 2)

---

6. Testing the Integration

1. Start the FastAPI backend.

2. Open `index.html` in a browser.
3. Enter feature values and click "Predict"



KNN Classifier Prediction

Predicted Class: Versicolor (Class 1)

---

7. Conclusion

This project demonstrates a KNN classification model integrated with a FastAPI backend and a simple frontend. The setup allows easy interaction between the trained model and users through a web interface. It can be extended to other datasets and models as needed.