Naive Bayes Classifier for Flower Species Prediction

NAME: Ankita Satapathy
ROLL NO: 22053400

1. Introduction

This project implements a Naive Bayes classifier to predict flower species based on their characteristics (sepal length, sepal width, petal length, and petal width). The model is built using Python and the scikit-learn library, with a custom Naive Bayes implementation. The application provides an API using FastAPI and a front-end interface for user interaction.

2. Files in the Submission

i. **NB3400.ipynb**: Jupyter Notebook containing the implementation and training of the Naive Bayes classifier.
ii. **NB3400.py**: Python script implementing the Naive Bayes classifier and training process. Generated in vscode terminal using the command-

```
PS C:\Users\KIIT\Desktop\22053400\4) Naive Bayes> jupyter nbconvert --to script NB3400.ipynb
```

iii. **main.py**: Defines the FastAPI server to handle prediction requests.
iv. **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 (numpy)**: Used for numerical computations.
- **Pandas (pandas)**: Used for data manipulation.
- **Scikit-learn (sklearn)**: Provides ML utilities and dataset handling.
- **FastAPI (fastapi)**: Creates a web API for serving predictions.
- **Pickle (pickle)**: Saves and loads the trained model.
- **Uvicorn (uvicorn)**: Runs the FastAPI server asynchronously.

ii. Installing Dependencies
Run the following command in the terminal to install the required dependencies:

```
PS C:\Users\KIIT\Desktop\22053400\4) Naive Bayes> pip install fastapi uvicorn numpy pandas scikit-learn
```

4. Model Implementation

The NaiveBayes class in NB3400.py implements a Naive Bayes classifier. It calculates the likelihood of each class based on the Gaussian probability distribution and selects the class with the highest posterior probability.

```
class NaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.var = {}
        self.priors = {}
        for c in self.classes:
            X_c = X[y == c]
            self.mean[c] = X_c.mean(axis=0)
            self.var[c] = X_c.var(axis=0) + 1e-9
            self.priors[c] = X_c.shape[0] / X.shape[0]
    def _pdf(self, class_idx, x):
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
    def predict(self, X):
        posteriors = []
        for x in X:
            class_probs = {}
            for c in self.classes:
                prior = np.log(self.priors[c])
                likelihood = np.sum(np.log(self._pdf(c, x)))
                class_probs[c] = prior + likelihood
            posteriors.append(max(class_probs, key=class_probs.get))
        return np.array(posteriors)
```

5. Training the Model

- The dataset used is the Iris Dataset from sklearn.datasets.
- The model is trained using four features:Sepal length, Sepal width, Petal length and Petal width
- The dataset is split into training and testing sets.
- The Naive Bayes model is trained and saved using pickle for later use.

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_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

iii. Loading the Trained Model

```python
with open("model.pkl", "wb") as file:
    pickle.dump(model, file)
```

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)
    return {"prediction": int(prediction[0])}
```

7. Running the Application (FastAPI Server)

Run the following command to start the FastAPI server-

```
PS C:\Users\KIIT\Desktop\22053400\4) Naive Bayes> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\KIIT\\Desktop\\22053400\\4) Naive Bayes']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [8460] using WatchFiles
INFO:     Started server process [17112]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

8. Front-End Implementation

The index.html file provides a user-friendly interface for inputting flower features and making predictions. It uses inline CSS and JavaScript to send user input to the FastAPI backend and display the predicted species.

```javascript
async function predict() {
    let features = [];
    let featureIds = ["sepal_length", "sepal_width", "petal_length", "petal_width"];
    for (let id of featureIds) {
        let value = document.getElementById(id).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();
        let classNames = ["🌸 Setosa", "🌿 Versicolor", "🌺 Virginica"];
        let className = classNames[data.prediction];
        document.getElementById("result").innerText = "🎉 Predicted Flower Species: " + className;
    } catch (error) {
        console.error("Error:", error);
        document.getElementById("result").innerText = "❌ Error: " + error.message;
    }
}
```

9. Conclusion

This project successfully implements a Naive Bayes classifier for predicting flower species. The model is integrated with a FastAPI backend and deployed using Uvicorn. It uses NumPy and Scikit-learn for computation and dataset handling, and Pickle for model persistence. The front-end allows users to interact with the model in a simple and intuitive manner.