Deploying an Iris Classifier with FastAPI

Presented by Mohammed Mubarak, July 21, 2025

What is FastAPI?

FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It's designed for quick development and robustness.

Asynchronous Capabilities

Handles multiple requests concurrently using Python's **async/await** syntax, ensuring high throughput for demanding applications.

Automatic Docs

Generates interactive API documentation (Swagger UI and ReDoc) automatically, simplifying testing and development workflows.

Data Validation

Leverages Pydantic for data validation, serialization, and deserialization, ensuring robust and error-free data handling.

Think of FastAPI as an efficient waiter: it takes orders (requests), processes them quickly, and delivers predictions without delay.

Why FastAPI for ML Deployment?

FastAPI is an ideal choice for deploying machine learning models due to its inherent strengths, which align perfectly with the demands of ML inference services.

High Performance & Scalability

FastAPI's asynchronous nature allows it to handle numerous concurrent user requests efficiently, making it well-suited for high-traffic ML inference.

Production Readiness

Supports essential tools like **Docker** for containerisation and **AWS** for cloud deployment, facilitating a smooth transition from development to production environments.

Seamless ML Integration

It easily integrates with popular ML libraries like Scikit-learn (e.g., Random Forest), PyTorch, and TensorFlow, allowing you to load and use pre-trained models directly within your API endpoints.

Simplified Testing with Swagger UI

The automatically generated Swagger UI provides an interactive interface for testing API endpoints, making it simple even for developers new to API interactions.

The Iris Classification Problem

Our client requires an API to classify Iris flower species based on their physical measurements. This is a classic machine learning problem ideal for demonstrating API deployment.

Dataset Size	150 samples of Iris flowers
Features	• Sepal Length (cm) • Sepal Width (cm) • Petal Length (cm) • Petal Width (cm)
Target Classes	• Iris setosa • Iris versicolor • Iris virginica
Objective	Build an API to classify Iris species from measurements

Machine Learning Model: Random Forest

For the Iris classification, we employed a Random Forest Classifier, a robust ensemble learning method known for its accuracy and ability to handle various data types.

Model Training

The Random Forest model was trained on the Iris dataset, leveraging all four features (sepal length, sepal width, petal length, petal width) to learn the patterns distinguishing each species.

Model Persistence

After training, the final, fitted model was saved as a Pickle (.pkl) file. This allows for efficient loading and reuse of the trained model in the FastAPI application without retraining.

Key Code Snippet:

```
model = RandomForestClassifier(n_estimators=100, random_state=42)model.fit(X, y)joblib.dump(model, 'model.pkl')
```

The random_state ensures reproducibility of the results.

FastAPI Setup for Prediction

Setting up the FastAPI application involved a few straightforward steps to create a web service capable of serving our trained Iris classification model.

1

Step 1: Installation

Installed FastAPI and Uvicorn (an ASGI server) using pip to set up the development environment.

```
pip install fastapi uvicorn
```

2

Step 2: Model Loading

The pre-trained model.pkl was loaded into memory when the FastAPI application starts, making it ready for inference.

```
model =
joblib.load('model.pkl')
```

3

Step 3: Prediction Endpoint

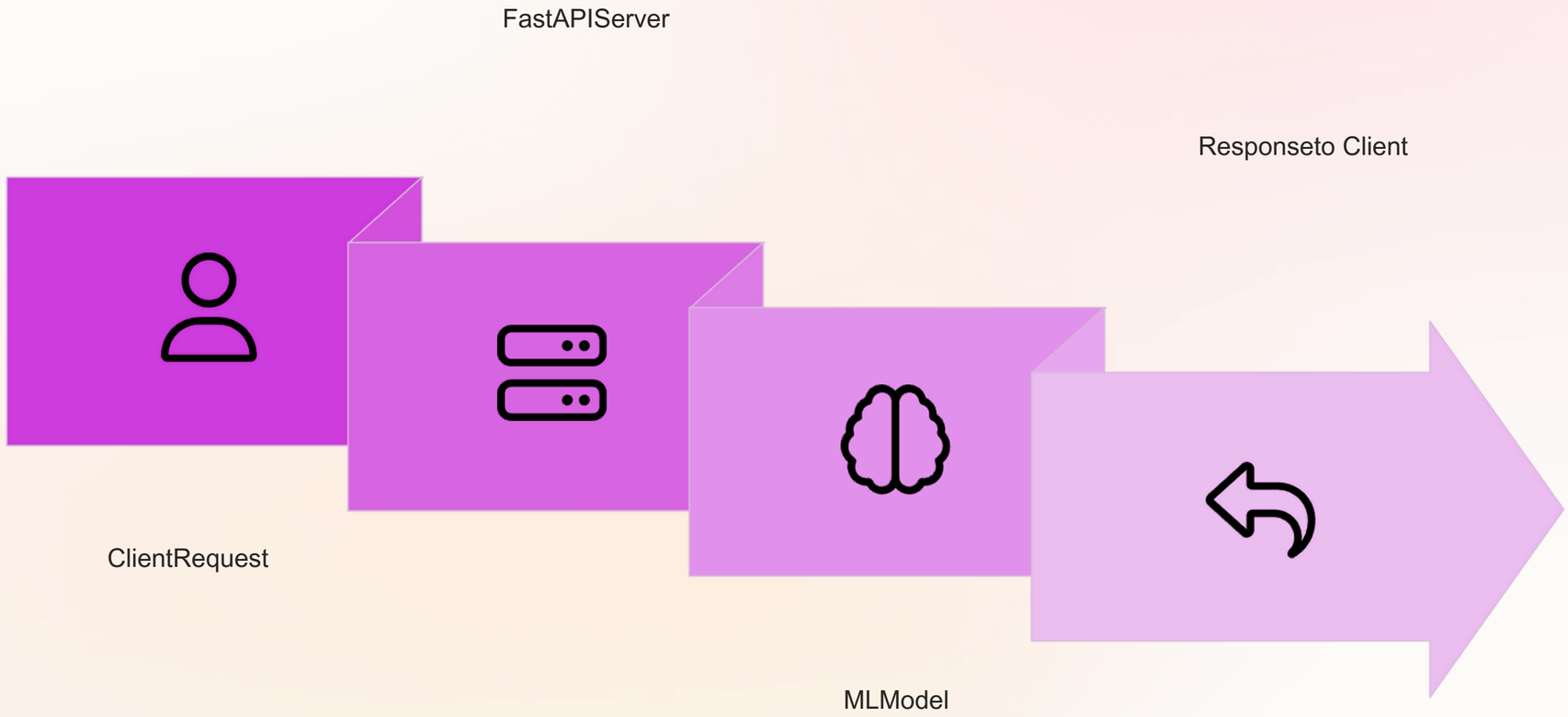
A /predict endpoint was created using @app.post to accept input data and return predictions.

```
@app.post("/predict")async
def predict(iris: IrisInput):
# Prediction logic here
```

The IrisInput class, defined with Pydantic, ensures that incoming data adheres to the expected format (sepal length/width, petal length/width).

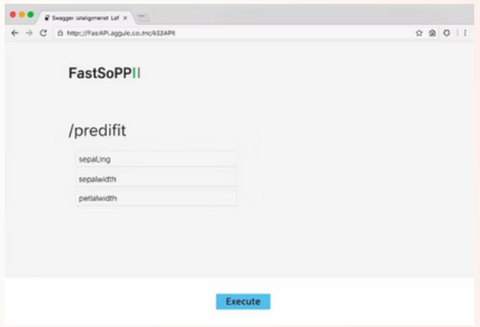
API Workflow for Iris Classification

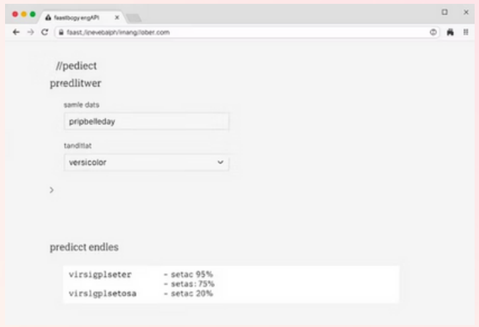
The API workflow for the Iris classifier is designed for simplicity and efficiency, enabling seamless interaction between the client and the ML model.



Testing the API with Swagger UI

Swagger UI provides an intuitive web interface for testing API endpoints directly from your browser, making development and debugging highly efficient.





Sample Test Input:

```
{  "sepal_length": 6.7,  "sepal_width": 3.0,  "petal_length": 5.2,  "petal_width": 2.3}
```

Expected API Output:

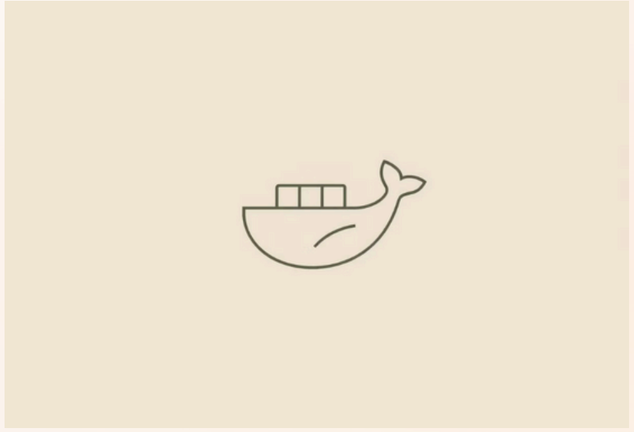
```
{ "species": "virginica", "probabilities": { "setosa": 0.0, "versicolor": 0.0, "virginica": 1.0 }}
```

Best Practices for Production Deployment

To ensure the Iris classifier API is robust, scalable, and maintainable in a production environment, several best practices should be adopted.

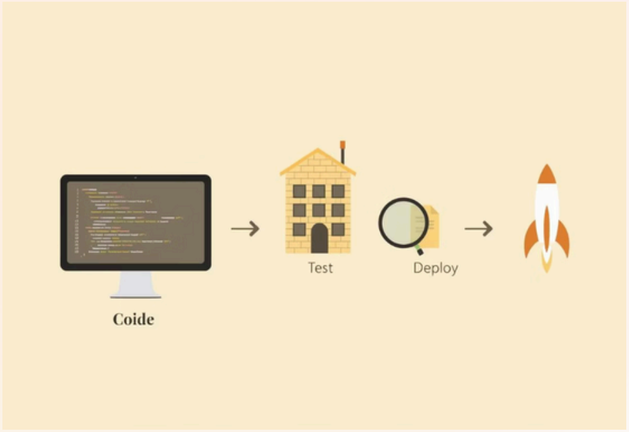
Containerisation with Docker

Encapsulate the FastAPI application and all its dependencies within a Docker container. This ensures consistent operation across different environments, from development to production.



Continuous Integration/Continuous Deployment (CI/CD)

Automate the build, test, and deployment process. CI/CD pipelines ensure that updates to the API are deployed reliably and efficiently, minimising downtime and manual errors.



API Monitoring with Uptrace

Implement robust monitoring solutions like **Uptrace** to track API performance, identify errors, and gain insights into usage patterns. This proactive monitoring helps ensure API reliability

Adhering to these practices makes the API not only functional but also **reliable**, **scalable**, and **easy to manage** in real-world scenarios.

Summary: Key Takeaways

We have successfully deployed an Iris classifier API using FastAPI, demonstrating a practical approach to bringing machine learning models into production environments.

Building & Saving ML Models

Understanding how to train and persist machine learning models (e.g., Random Forest saved as .pkl) is fundamental for deployment.

Creating APIs with FastAPI

FastAPI provides a fast, modern, and intuitive framework for developing high-performance web APIs with Python.

Testing with Swagger UI

Leveraging automatically generated interactive documentation (Swagger UI) simplifies API testing and validation.

These skills are directly transferable to a wide range of real-world machine learning projects, from predictive analytics to natural language processing. The journey from model development to a deployed, accessible API is a critical step in delivering tangible value from data science initiatives.