**Assignment 5**

**Name:** Snehal Laxmikant Yelwande  
**Class:** TY A CSE(AI)  
**Roll No:** 63  
**Batch:** P-2  
**PRN:** 22311760

**Problem Statement**

Build and expose REST APIs using **FastAPI** and containerize the application using **Docker**.

**Theory**

**Need of REST APIs in ML**

Machine Learning models are often trained offline but need to be deployed so applications can consume them in real time. REST APIs allow:

* Serving predictions to web, mobile, or enterprise apps.
* Standard communication via HTTP (GET, POST).
* Scalability and modularity in deployment.

**FastAPI**

FastAPI is a modern, high-performance web framework for building APIs with Python.  
**Features:**

* Auto-generates interactive Swagger UI.
* Asynchronous support for fast execution.
* Easy model serving with Pydantic validation.

**Docker**

Docker enables containerization of applications, ensuring:

* Portability across environments.
* Lightweight deployment.
* Dependency isolation.

**Workflow**

1. Train ML model and save it.
2. Build FastAPI app to serve predictions.
3. Test endpoints locally.
4. Containerize app with Docker.

**Executions**

**Step 1 – Train and Save Model**

**File: train\_model.py**

import joblib

from sklearn.datasets import load\_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

# Load dataset

iris = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

iris.data, iris.target, test\_size=0.2, random\_state=42

)

# Train model

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Save model

joblib.dump(model, "iris\_model.pkl")

print("✅ Model saved as iris\_model.pkl")

**Step 2 – Create FastAPI App**

**File: main.py**

from fastapi import FastAPI

import joblib

import numpy as np

from pydantic import BaseModel

# Load trained model

model = joblib.load("iris\_model.pkl")

# Define API input schema

class IrisInput(BaseModel):

sepal\_length: float

sepal\_width: float

petal\_length: float

petal\_width: float

# Initialize app

app = FastAPI(title="Iris Classifier API", version="1.0")

@app.get("/")

def home():

return {"message": "Welcome to the Iris Classifier API"}

@app.post("/predict")

def predict(data: IrisInput):

features = np.array([[data.sepal\_length, data.sepal\_width,

data.petal\_length, data.petal\_width]])

prediction = model.predict(features)[0]

return {"prediction": int(prediction)}

**Step 3 – Run API with Uvicorn**

uvicorn main:app --reload --host 0.0.0.0 --port 8000

**Test in Browser:**

* Swagger UI: http://127.0.0.1:8000/docs
* Example request (POST /predict):

{

"sepal\_length": 6.1,

"sepal\_width": 2.8,

"petal\_length": 4.7,

"petal\_width": 1.2

}

Output:

{"prediction": 1}

**Step 4 – Dockerize the Application**

**File: Dockerfile**

# Base image

FROM python:3.9

# Set working directory

WORKDIR /app

# Copy requirements and install

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copy project files

COPY . .

# Expose port

EXPOSE 8000

# Run API

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

**File: requirements.txt**

fastapi

uvicorn

scikit-learn

joblib

numpy

**Step 5 – Build and Run Docker Container**

# Build image

docker build -t iris-api .

# Run container

docker run -p 8000:8000 iris-api

**Output**

* FastAPI app successfully started at http://127.0.0.1:8000.
* Swagger docs auto-generated.
* Predictions returned correctly.
* Docker containerized API accessible at http://localhost:8000.

**Conclusion**

Using **FastAPI**, we exposed a trained ML model as a REST API that accepts JSON input and returns predictions. Containerizing with **Docker** made the API portable, reproducible, and ready for deployment in any environment. This demonstrates a production-ready ML model deployment workflow.

**Link to GitHub Repository:** [**MLOPS\_Assignments**](https://github.com/SnehalLY/MLOPS_Assignments)