ML Model Inference with FastAPI - Class Assignment

Assignment Overview

Time Limit: 1.5 hours
Objective: Create a FastAPI application that serves a machine learning model for inference

Learning Goals

- Understand how to deploy ML models as web APIs
- Practice with FastAPI framework
- Learn about model serialisation and loading
- Implement proper API endpoints for ML inference

Assignment Requirements

Step 1: Problem Selection (15 minutes)

Choose ONE of these pre-defined problems to keep scope manageable:

Option A: Iris Flower Classification - https://www.kaggle.com/datasets/uciml/iris

- Predict iris species (setosa, versicolor, virginica)
- Input: 4 numerical features (sepal length, sepal width, petal length, petal width)
- Dataset: Built-in sklearn iris dataset

Option B: House Price Prediction - https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

- Predict house prices based on features
- Input: Square footage, bedrooms, bathrooms, age
- Dataset: Generate synthetic data or use the Boston housing

Option C: Text Sentiment Analysis - https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset

- Classify text as positive/negative sentiment
- Input: Text string
- Use simple vectorisation + logistic regression

Step 2: Model Development (30 minutes)

1. **Load/Create Dataset**
    - Use the provided dataset for your chosen problem
    - Split into train/test sets
2. **Train Model**
    - Use simple algorithms (LogisticRegression, RandomForestClassifier, LinearRegression)
    - Fit the model on the training data
    - Evaluate on the test set
3. **Save Model**
    - Use `joblib` or `pickle` to serialise your trained model
    - Save any preprocessing objects (scalers, encoders) if used

Step 3: FastAPI Implementation (40 minutes)

Create a FastAPI application with the following endpoints:

1. **GET `/`** - Health check endpoint
2. **POST `/predict`** - Main prediction endpoint
3. **GET `/model-info`** - Return model metadata

Required Features:

- Input validation using Pydantic models
- Proper error handling
- JSON response format
- Load the saved model on startup

Step 4: Testing & Documentation (15 minutes)

1. Test endpoints using FastAPI's automatic docs (`/docs`)
2. Provide at least 2 example requests
3. Document any assumptions or limitations

Deliverables

Create a repository on GitHub and include the link in the report.

1. `main.py` - FastAPI application
2. `model.pkl` - Saved ML model
3. `requirements.txt` - Dependencies
4. `README.md` - Brief documentation with:
    - Problem description
    - Model choice justification
    - API usage examples
    - How to run the application

Starter Code Template

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import joblib
import numpy as np
from typing import List

# Load your trained model
# model = joblib.load("model.pkl")

app = FastAPI(title="ML Model API", description="API for ML model inference")

# Define input schema
class PredictionInput(BaseModel):
    # Define your input fields here
    feature1: float
    feature2: float
    # Add more fields as needed

class PredictionOutput(BaseModel):
    prediction: float  # or str for classification
    confidence: float = None

@app.get("/")
def health_check():
    return {"status": "healthy", "message": "ML Model API is running"}

@app.post("/predict", response_model=PredictionOutput)
def predict(input_data: PredictionInput):
    try:
        # Convert input to model format
        # features = np.array([[input_data.feature1, input_data.feature2, ...]])

        # Make prediction
        # prediction = model.predict(features)[0]

        # Return prediction
        return PredictionOutput(prediction=prediction)

    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

@app.get("/model-info")
def model_info():
    return {
        "model_type": "Your model type",
        "problem_type": "classification/regression",
```

```
    "features": ["list", "of", "feature", "names"]
}
```

Grading Criteria (100 points)

- **Model Implementation (25 points)**
    - Appropriate model choice for the problem
    - Proper training and evaluation
    - Model saved correctly
- **FastAPI Implementation (35 points)**
    - All required endpoints are working
    - Proper Pydantic models for input validation
    - Error handling implemented
    - Model loading on startup
- **Code Quality (20 points)**
    - Clean, readable code
    - Proper structure and organisation
    - Comments were appropriate
- **Documentation (10 points)**
    - Clear README with usage examples
    - API documentation through FastAPI docs
- **Testing (10 points)**
    - Demonstrated testing through `/docs`
    - Valid example requests provided

Tips for Success

1. **Start Simple:** Choose the iris classification problem if unsure - it's the most straightforward
2. **Use Basic Models:** LogisticRegression or RandomForest are sufficient
3. **Test Early:** Use `/docs` endpoint to test your API as you build
4. **Handle Errors:** Add try-catch blocks around model predictions
5. **Keep Dependencies Minimal:** Only install what you need

Common Issues to Avoid

- Forgetting to load the model in the FastAPI app
- Not handling input validation properly
- Overly complex models that take too long to train
- Missing error handling for invalid inputs
- Not testing the API endpoints

Bonus Points (Optional)

- Add logging to your API
- Implement batch prediction endpoint
- Add model confidence scores
- Create a simple HTML form for testing

Getting Started

1. Install dependencies: `pip install fastapi uvicorn scikit-learn pandas numpy joblib`
2. Choose your problem and start with model development
3. Save your model and build the FastAPI app
4. Test using: `uvicorn main:app --reload`
5. Visit `http://localhost:8000/docs` to test your API

Good luck!