Dockerized Sentiment Analysis API: Group Report

Introduction

In this project, we implemented a machine learning (ML) sentiment analysis pipeline using the HuggingFace model, `cardiffnlp/twitter-roberta-base-sentiment-latest`. The pipeline was wrapped with a FastAPI application and containerized using Docker. This project aimed to demonstrate the advantages of containerization, such as reproducibility, portability, and scalability, for deploying ML applications. We successfully developed, tested, and pushed the Dockerized application to Docker Hub.

1. Pipeline Review and Enhancement

Review of the Original Pipeline

- **Model**: `cardiffnlp/twitter-roberta-base-sentiment-latest` was selected for its robustness in analyzing sentiment from short text inputs such as tweets.
- **Input**: Text data.
- **Output**: Sentiment labels (`positive`, `neutral`, `negative`) and their associated probabilities.
- **Dependencies**:
    - `transformers` (HuggingFace library).
    - `torch` (PyTorch for model computation).

Enhancements for Containerization

1. **Code Modularization**:
    - Separated logic into modular functions:
        - `analyze_sentiment`: Encapsulates the inference pipeline.
        - API routes are well-defined for root endpoint (`/`) and prediction (`/predict`).
2. **Configuration Management**:
    - Dynamic model loading via HuggingFace, allowing for model reuse and updates without modifying the core application.
3. **Error Handling**:
    - Incorporated robust error handling using FastAPI's `HTTPException`.
    - Ensured informative error messages for users during pipeline failures.

2. Containerization with Docker

Dockerfile

A Dockerfile was written to encapsulate the ML pipeline and API application. Key elements included:

1. **Base Image**: Used `python:3.9-slim` for its lightweight nature.
2. **Dependency Installation**:
    - Used `pip` to install `requirements.txt`, ensuring all libraries were encapsulated within the container:
        - `fastapi`
        - `uvicorn`
        - `transformers`
        - `torch`
3. **Environment Setup**:
    - The working directory was set to `/app`.
    - The application files (`app.py`, `requirements.txt`) were copied into the container.
4. **Exposing API Port**:
    - Port `6000` was exposed to allow communication with the API.

Dockerfile:

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the application files
COPY app.py /app/
COPY requirements.txt /app/

# Install the dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose the API port
EXPOSE 6000

# Command to run the application
```

```
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "6000"]
```

3: Testing and Validation

Testing Process

1. **Dockerized Testing**:

Built and ran the Docker container locally:

```
docker build -t cc-p3 .
docker run -p 6000:6000 cc-p3
```

Tested the API using Postman: accessed the API via `http://localhost:6000` and confirmed consistent behavior.

Request:

```
POST Request: "http://localhost:6000/predict"
Request Body: {"text": "I love this movie!"}
```

Response:

```
{
    "result": [
        {
            "label": "positive",
            "score": 0.9847092628479004
        }
    ]
}
```

Request:

```
POST Request: "http://localhost:6000/predict"
Request Body: {"text": "I hate this movie!"}
```

Response:

```
{
    "result": [
        {
            "label": "negative",
            "score": 0.9314882755279541
        }
    ]
```

```
}
```
Request:
```
POST Request: "http://localhost:6000/predict"
Request Body: {"text": "Today is Monday."}
```
Response:
```
{
    "result": [
        {
            "label": "neutral",
            "score": 0.7061721682548523
        }
    ]
}
```

2. **Cross-Platform Validation**:
    - Tested the Docker container on multiple platforms (Windows, macOS, and Linux) to ensure environment consistency.

Key Observations

- The encapsulation of dependencies and environment setup ensured reproducibility across different systems.
- The `/predict` endpoint demonstrated accurate and fast inference times.

4. Pushing to Docker Hub

Process

1. **Docker Image Tagging**:

Tagged the Docker image with a unique identifier for Docker Hub:
```
docker tag cc-p3 annoyingpanda/cc-p3
```

2. **Pushing to Docker Hub**:

Pushed the image to Docker Hub:
```
docker push annoyingpanda/cc-p3
```

3. **Verification**:

Verified that the image was available on Docker Hub and tested pulling and running the image on another system:

```
docker pull annoyingpanda/cc-p3:latest
docker run -p 8000:6000 annoyingpanda/cc-p3:latest
```

4. **DockerHub Repo**: https://hub.docker.com/repository/docker/annoyingpanda/cc-p3/general

5. Conclusion

This project successfully demonstrated the development and deployment of a containerized ML pipeline using FastAPI and Docker. Key achievements include:

- **Reproducibility**: The Dockerized environment ensured consistent behavior across platforms.
- **Scalability**: The containerized API can be scaled easily using orchestration tools like Kubernetes.
- **Portability**: The lightweight Docker image enabled seamless sharing and deployment.

6. Future Improvements

1. **API Enhancements**:
    - Add more endpoints, such as batch sentiment analysis.
    - Incorporate rate-limiting for better performance under heavy load.
2. **Deployment**:
    - Deploy the Dockerized API to cloud platforms like AWS ECS, Azure, or Google Cloud Run.
3. **CI/CD Integration**:
    - Automate Docker image building, testing, and pushing using GitHub Actions or GitLab CI/CD pipelines.

7. Reference:

- https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest