Assignment(Mlflow, Fastapi, Flask, Docker)

Qunatitative and financial modeling master
Data Science major
Mohammed VI Polytechnic University

Module Name : Cloud Native

Writer:
YASSINE Smail

Supervised by:
Prof. Fahd KALLOUBI

December 3, 2023

Contents

List of Figures

Chapter 1

Introduction

In the realm of data-driven decision-making, sentiment analysis has emerged as a pivotal application, offering valuable insights into user opinions and attitudes. This report encapsulates a machine learning project centered around the sentiment analysis of IMDb movie reviews, leveraging cutting-edge technologies and frameworks. The primary objectives of the project include data preprocessing, training multiple machine learning models, exposing the best-performing model as a REST API, and containerizing the entire application for efficient deployment and scalability.

The initial phase of the project involves meticulous data preprocessing to ensure the cleanliness and relevance of IMDb movie reviews data. This step is foundational, setting the stage for subsequent machine learning tasks. Following this, the project embarks on the development and training of five distinct machine learning models tailored for sentiment analysis. These models, trained on a diverse set of features, are designed to capture nuanced sentiments expressed in the reviews.

1.1 Report Structure :

To systematically manage the iterative process of model development and fine-tuning, the project employs MLflow, a comprehensive platform for the complete machine learning lifecycle. MLflow facilitates the tracking of model performance, versions, and parameters, ensuring a transparent and reproducible workflow. Utilizing MLflow's logging capabilities, the report highlights the systematic recording of metrics, parameters, and artifacts associated with each model iteration.

The project then advances to the serialization of the best-performing model in the ONNX format—a lightweight and interoperable standard for representing machine learning models. Additionally, the dedicated preprocessing transformations are serialized using the Transformers API and saved in pickle format, ensuring the seamless integration of preprocessing steps into the deployed model.

Moving into the deployment phase, the report presents the implementation of a REST API using FastAPI, a modern, fast, web framework for building APIs with Python 3.7+. The seri-

alized ONNX model and preprocessing transformations are leveraged to create a robust sentiment analysis API. To enhance deployment efficiency and scalability, the FastAPI application is encapsulated within a Docker container, promoting consistency across diverse computing environments.

Postman, a popular API testing tool, is employed to consume and evaluate the performance of the FastAPI sentiment analysis API. The report details the steps taken to ensure a smooth interaction between the API and external applications, validating its efficacy and reliability.

In parallel, a dedicated Flask application is developed to showcase an alternative approach to consuming the sentiment analysis API. This secondary application encapsulates the versatility of the deployed API and serves as an illustrative example of its potential integration into various applications.

Conclusively, the entire project is containerized using Docker, encapsulating both the FastAPI application and the Flask consumer application. This containerization enhances reproducibility, facilitates easy deployment across different environments, and underscores the project's commitment to best practices in modern software development.

Chapter 2

Data Preprocessing and Processing:

2.1 Dataset Description

The dataset employed in this sentiment analysis project comprises a diverse collection of IMDb movie reviews, offering a rich tapestry of opinions and expressions from a broad spectrum of moviegoers. Each review is accompanied by metadata such as user ratings, timestamps, and other relevant information, providing context for the sentiments expressed.

2.1.1 Reveiws preprocessing:

- HTML Tag Removal: The preprocessing phase begins with the systematic removal of HTML tags from the IMDb movie reviews dataset. This step ensures that the textual content remains devoid of any extraneous formatting artifacts, laying the groundwork for cleaner and more meaningful analysis.
- Punctuation Removal: Subsequent to HTML tag removal, the dataset undergoes meticulous punctuation removal. This process streamlines the textual data by eliminating unnecessary symbols, contributing to a more focused and coherent corpus for sentiment analysis.
- Stopword Elimination: Common stopwords, often carrying little semantic value, are systematically eradicated from the dataset. By discarding these frequently occurring but non-informative words, the preprocessing phase further refines the dataset, promoting model generalization and accuracy.

2.2 Model Training :

In this sentiment analysis project, a diverse set of machine learning models has been employed to effectively discern sentiments within IMDb movie reviews. The decision tree, forming a hierarchical structure through recursive partitioning, predicts sentiments by traversing the tree. The random forest, an ensemble of decision trees, enhances accuracy through collective predictions. Logistic regression, a linear model, estimates class probabilities, well-suited for binary

classification. The support vector machine (SVM), a discriminative model, constructs a hyperplane for effective class separation, especially useful in handling non-linear relationships through kernel functions. These models collectively provide a robust and versatile framework for extracting nuanced sentiments from the dynamic landscape of IMDb reviews.

2.2.1 Top model Overview :

The optimal accuracy is given by the logistic regression model :

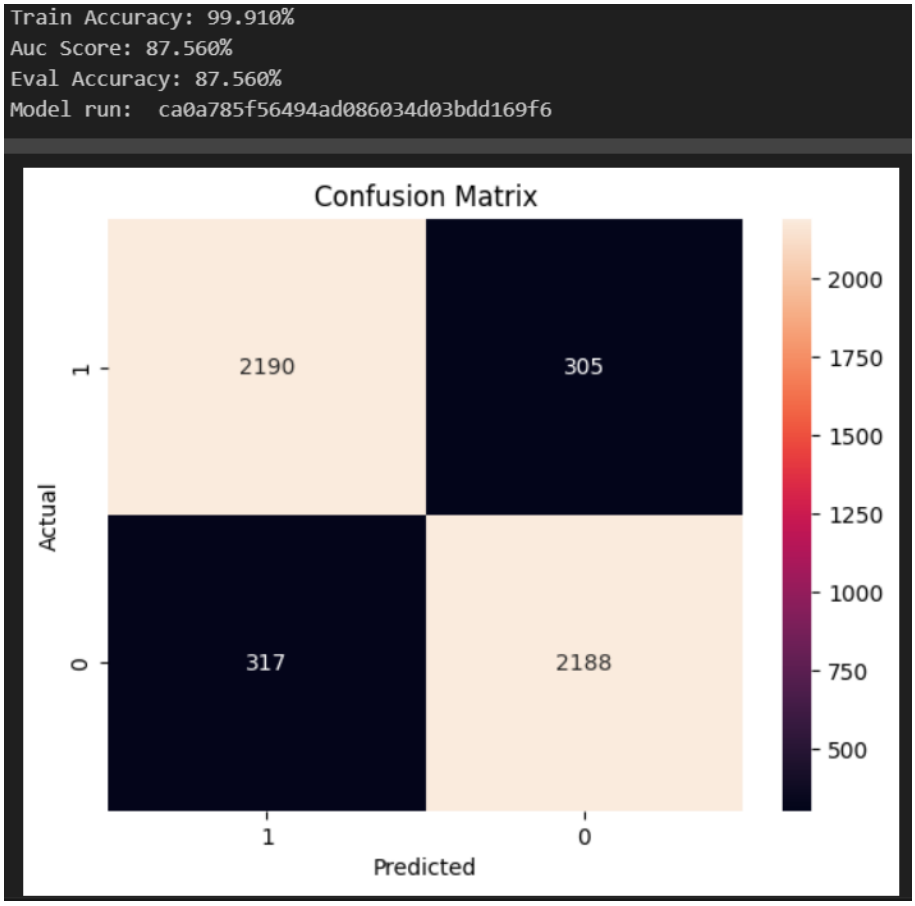

Figure. 2.1: Confusion matrix of logistic regression

Chapter 3

Mlflow Logging :

3.1 Mlflow Use :

In the context of this sentiment analysis project, MLflow serves as a crucial tool for systematic model management and performance tracking. Leveraging MLflow, each phase of model development, from preprocessing to training and evaluation, is seamlessly logged, allowing for meticulous monitoring of metrics, parameters, and artifacts. This facilitates a transparent and reproducible workflow, enabling efficient model iteration and comparison.

3.2 Models logging and comparison :

3.2.1 model logging and preprocessing logging:

As we have trained many models, each model has its own parameters and artifacts.

Below is an example of the logged parameters of a certain model :

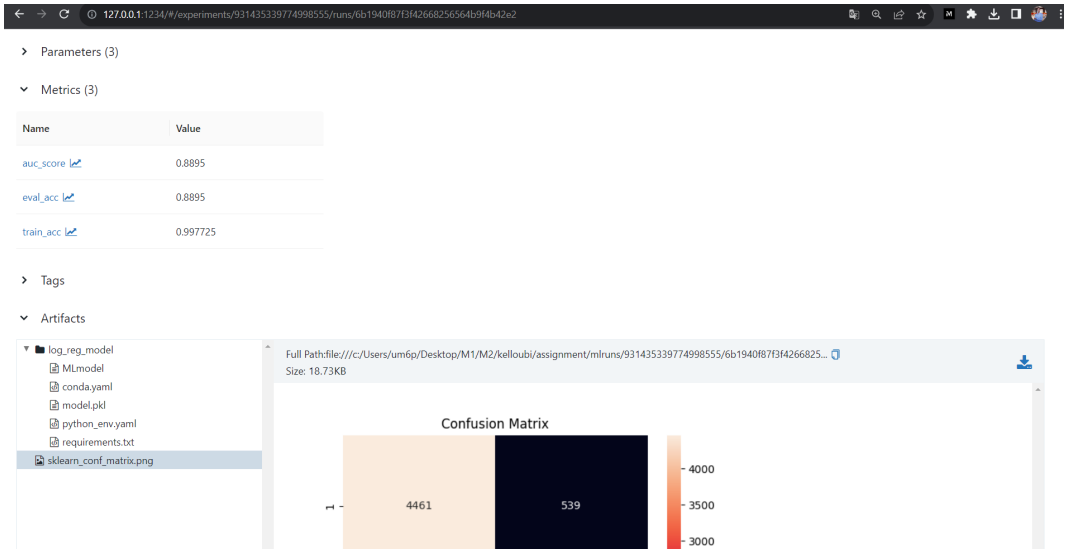

Figure. 3.1: Model artifacts

3.2.2 Models comparison through Mlflow :

Below are the models accuracies dashboard tracking, we can filter by several parameters.

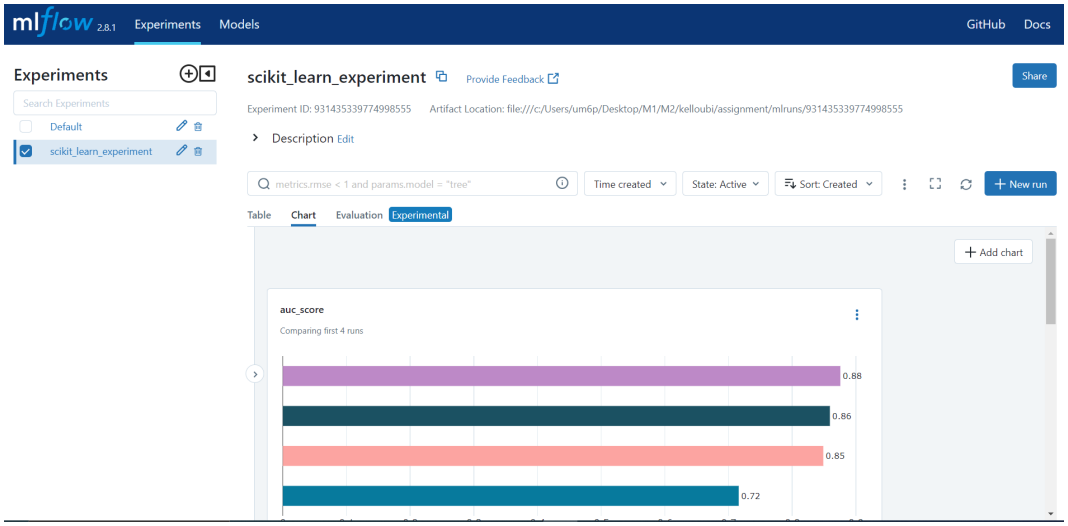

Figure. 3.2: Models comparison

3.3 Mlflow automation

3.3.1 Optimal model searching:

The search for the optimal model architecture within the MLflow framework is a pivotal aspect of this sentiment analysis project. MLflow's ability to systematically log and track the performance metrics and parameters of multiple machine learning models facilitates an efficient and informed exploration of the model space. By leveraging MLflow's experimentation capabilities, iterative model development becomes streamlined, allowing for the systematic comparison of different model configurations. This search for the optimal model involves fine-tuning hyperparameters, experimenting with varying architectures, and ultimately identifying the configuration that yields the best sentiment analysis performance. MLflow's centralized tracking system enhances the visibility and reproducibility of these experiments, ensuring that the chosen model configuration is well-informed and backed by comprehensive insights gained throughout the iterative search process.

```
# Find the best model
best_run = mlflow.search_runs(order_by=["metrics.auc_score"]).iloc[0]
best_run_id = best_run.run_id

# Retrieve the artifact URI for the 'model' directory
best_model_uri = mlflow.get_run(run_id=best_run_id).info.artifact_uri
```

```
best_model_uri
```

```
'file:///c:/Users/um6p/Desktop/M1/M2/kelloubi/assignment/mlruns/931435339774998555/08d32b1413fb43de83944a4a0372e2e6/artifacts'
```

Figure. 3.3: Optimal model searcching (Mlflow)

Chapter 4

API Building and Demo App Creation:

FastAPI stands as a modern, fast, and high-performance web framework for building APIs with Python 3.7 and above. Characterized by its exceptional speed and automatic generation of interactive API documentation, FastAPI is designed to simplify and accelerate the development of robust APIs. It leverages Python type hints for data validation and automatic API documentation generation, enhancing code clarity and reducing the likelihood of errors. Asynchronous capabilities make it well-suited for handling concurrent requests efficiently. The framework's adherence to open standards and its seamless integration with third-party tools make it a versatile choice for developers seeking both speed and productivity in API development.

4.1 Fast API building :

The fastapi app will be consumed later in building our demo app using flask and rendering a template for styling purposes.

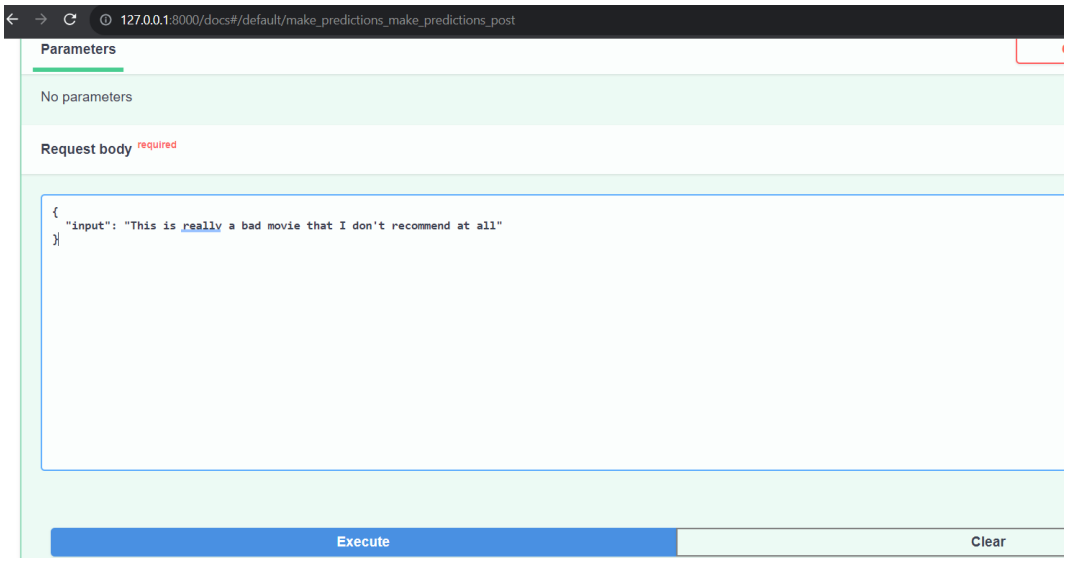

Figure. 4.1: Fastapi

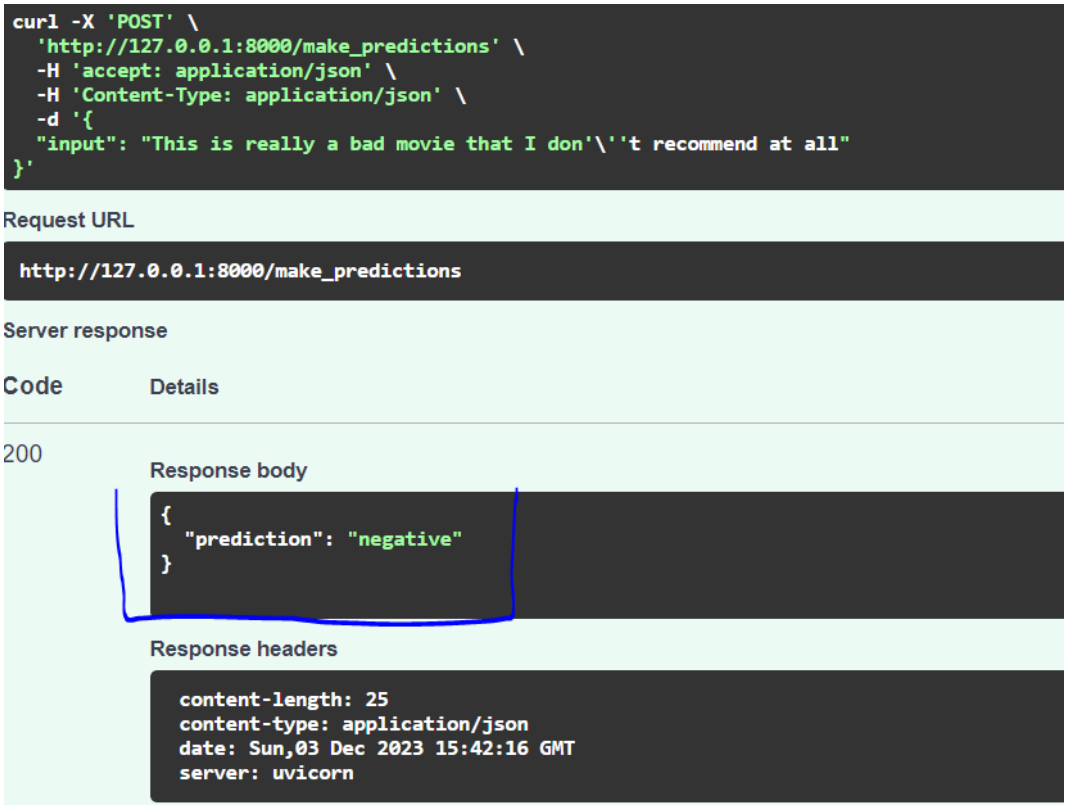

Figure. 4.2: Test and make a prediction

4.1.1 API Dockerization :

Dockerizing the API app involves encapsulating the FastAPI application within a Docker container, ensuring seamless deployment and portability. By leveraging Docker, the API and its dependencies are packaged together, creating an isolated environment that remains consistent across different systems.

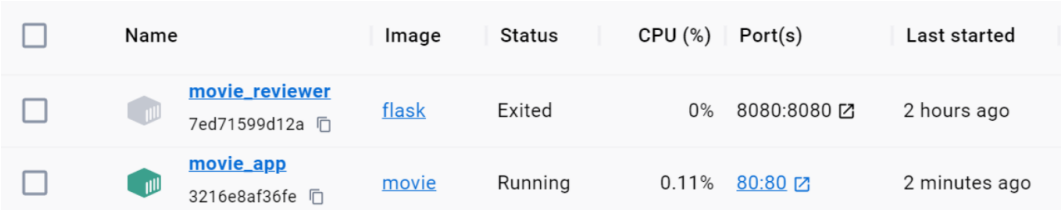

Figure. 4.3: Docker image running

4.1.2 Postman for API tracking :

Postman is a versatile and user-friendly API development and testing platform that streamlines the process of designing, testing, and documenting APIs. Recognized for its intuitive interface, Postman enables developers to send requests to APIs, inspect responses, and visualize data effortlessly. Beyond its fundamental role in testing API endpoints, Postman offers collaboration features, allowing teams to share and synchronize their API workflows. Its comprehensive set of tools includes automated testing, scripting capabilities, and the creation of detailed API documentation.

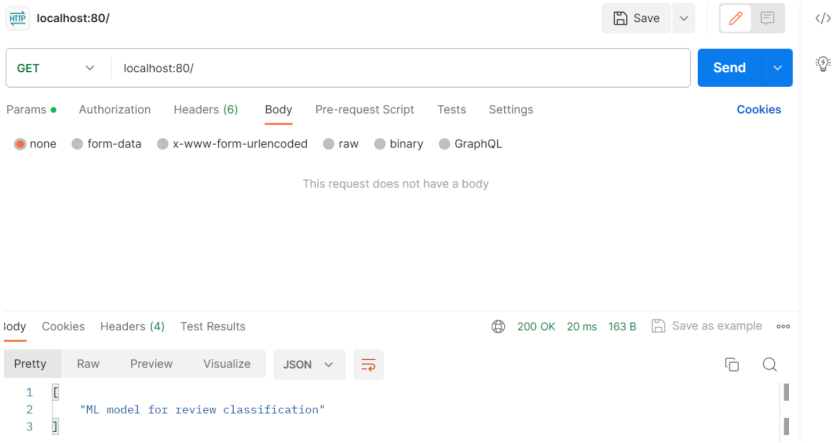

Figure. 4.4: Postman interface for our api

4.2 Flask app creation :

Flask, a micro-framework for Python, is renowned for its simplicity and flexibility in web development. Built with the philosophy of providing the essentials while allowing developers the freedom to choose additional components, Flask is often described as the "micro" framework due to its minimalistic core. Flask excels in crafting web applications quickly and efficiently, offering a straightforward routing system and a modular architecture. It supports various extensions, allowing developers to add functionalities as needed. Flask's lightweight design and extensive documentation make it an excellent choice for both beginners and experienced developers, providing a solid foundation for building web applications with Python.

```
(venv) PS C:\Users\um6p\Desktop\M1\M2\kelloubi\assignment\flask app> flask run
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Figure. 4.5: Running of the flask app

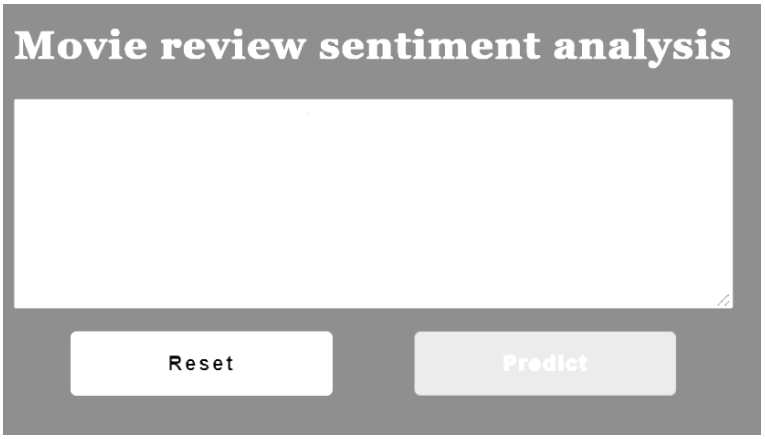

Figure. 4.6: Flask app

4.3 Parallel running using Docker-Compose :

In orchestrating the parallel deployment of both the FastAPI and Flask applications, Docker Compose serves as a pivotal tool, offering a streamlined and declarative approach. The docker-compose.yml file encapsulates the configuration for both services, defining the necessary dependencies and network configurations. Each service is specified with its respective build context and Dockerfile, ensuring the isolation and reproducibility of each application. By designating ports for each service, Docker Compose facilitates concurrent and independent access to the FastAPI and Flask applications. The depends on directive ensures a controlled startup sequence, with the Flask app waiting for the FastAPI app to be initialized before initiating its own deployment.

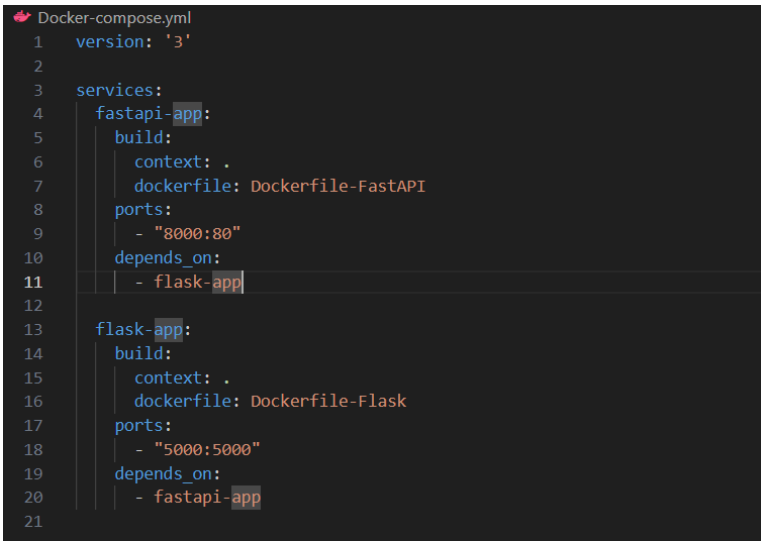

Figure. 4.7: Docker-compose configuration

As you may observe from the yml file, we have linked our flask app image to fastapi to solve the issues of locality.

Chapter 5

Conclusion :

In conclusion, this sentiment analysis project has successfully navigated the intricate landscape of IMDb movie reviews, employing a diverse array of machine learning models, and leveraging cutting-edge technologies for model deployment. The comprehensive data preprocessing ensured the quality of the dataset, laying the foundation for robust model training. MLflow played a pivotal role in systematically tracking model performance, parameters, and versions, providing a transparent and reproducible workflow throughout the machine learning lifecycle.

The selection and fine-tuning of machine learning models, including Decision Trees, Random Forests, Logistic Regression, and Support Vector Machines, underscored the project's commitment to model diversity and performance optimization. Each model brought unique strengths to the sentiment analysis task, demonstrating the importance of thoughtful model selection in achieving nuanced and accurate predictions.

The deployment of the sentiment analysis model as a FastAPI-based REST API, encapsulated within a Docker container, showcased modern practices for scalable and portable application deployment. Furthermore, the creation of a Flask application to consume the API emphasized the versatility and accessibility of the deployed sentiment analysis service.