Weekly Internship Report (22/07/24 - 26/07/24)

Introduction

This week focused on deep learning fundamentals, building and deploying a classification model, and improving API performance. The primary objectives were mastering FastAPI and Docker, creating a classification model with prediction endpoints, and performing load testing.

Tasks Completed

Task 1: Basics of DL and FastAPI

- **Description**: Studied the fundamentals of Artificial Neural Networks (ANN) and explored the basics of FastAPI. Additionally, learned about various loss functions and optimizers used with ANN.
- **Process**: Reviewed theoretical materials and documentation, and experimented with FastAPI tutorials. Studied loss functions such as MSE, MAE, Huber Loss, Binary Cross Entropy, Categorical Cross Entropy, and optimizers including Gradient Descent, Stochastic GD, Mini Batch SGD, Mini Batch SGD with Momentum, AdaGrad, AdaDelta and RMSProp, and Adam Optimizer.
- **Technologies Used**: Python, FastAPI.

Task 2: Classification Model and FastAPI Endpoint

- **Description:** Created a classification model and developed a FastAPI endpoint for making predictions.
- **Process:** Built and trained a classification model by eliminating stopwords from the training data, combined removing unnecessary columns, combining the all the text columns into a single column, handling null values and applying stemming using regular expression. Used the Logistic Regression model and TD-IDF vectorizer to train the model and then exposed it via a FastAPI endpoint.
- **Technologies Used:** Python, scikit-learn, FastAPI, re.

Task 3: Frontend Creation Using Streamlit

- **Description**: Developed a Streamlit frontend for making predictions from JSON and csv file inputs. Utilized Pydantic models for data validation and preparing response structure.
- **Process**: Implemented the user interface in Streamlit, integrated with FastAPI backend.
- **Technologies Used**: Streamlit, Pydantic, Python.

Task 4: Docker Image and Containers

- **Description**: Created Docker images and containers for the classifier to streamline deployment.
- **Process:** Initially wrote a common Dockerfile to run the FastAPI and Streamlit app and writing the necessary commands in “start.sh” and executing “start.sh” using Dockerfile, built images, and deployed containers.
- **Technologies Used**: Docker, Docker-Compose.

Task 5: Load Testing Using Locust

- **Description:** Performed load testing with Locust to assess the performance of the API endpoints and explored ways to improve Requests Per Second (RPS) with increase in number of request as well as number of concurrent users.
- **Process:** Set up Locust tests and analysed performance metrics, focusing on Ramp Up, Request Per Second, Number of concurrent Users, and the First Instance of failure for 10000 requests.
- **Technologies Used**: Locust, Python.

Task 6: BentoML Service Creation

- **Description**: Created a BentoML service for predictions and explored the use of Uvicorn with workers and Hypercorn for better performance.
- **Process**: Initially performed load tests using the Hypercorn server, then tested on the Uvicorn server with 4 workers, and finally set up the BentoML service and integrated it with FastAPI without using Docker.

Technologies Used: BentoML, Hypercorn.

Ongoing Tasks

- **BentoML Integration:**
    - **Current Status:** Integrating FastAPI with BentoML service for optimal deployment. Ongoing issues include properly mounting calling the endpoint on BentoML and ensuring seamless interaction between FastAPI and BentoML endpoints using Docker-Compose as well as executing BentoML service using Docker-Compose.

Meetings and Collaborations

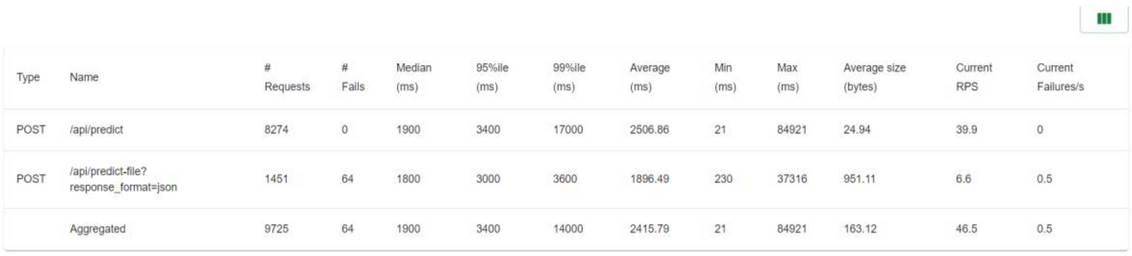
- **Daily Meetings:**
    - **Date and Time**: Monday to Friday, morning at 10:30 AM and afternoon at 3:30 PM.
    - **Agenda**: Morning meetings define the tasks for the day and follow up on the previous day's work. Afternoon meetings discuss encountered errors and think about solutions.

Learnings and Development

- **Knowledge Gained:**
    - **Technologies**: FastAPI, Docker, Docker-Compose, Locust.
    - **Cost Functions:** MSE, MAE, Huber Loss, Binary Cross Entropy, Categorical Cross Entropy, Softmax.
    - **Optimizers:** Gradient Descent, Stochastic GD, Mini Batch SGD, Mini Batch SGD with Momentum, AdaGrad, AdaDelta and RMSProp, Adam Optimizer.
    - **Coding Standards:** Familiarity with PEP8 and production-ready code structures.

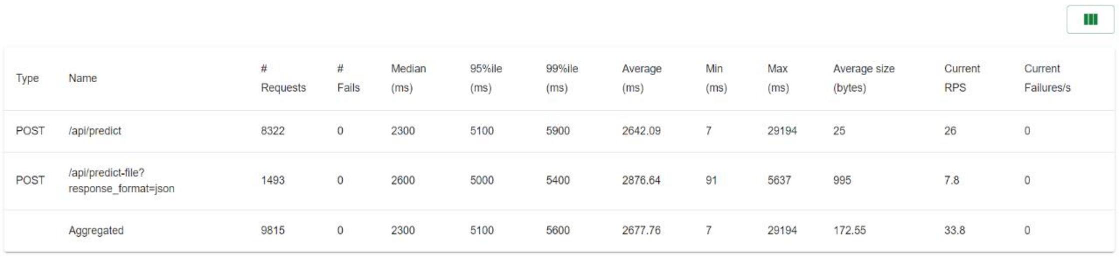
Challenges and Solutions

- **Challenge 1: Performance Issues with Uvicorn**
    - **Description:** Uvicorn supported only 200 concurrent users with an RPS of 46.5.
    - **Solution:** Switched to Hypercorn and later used multiple Uvicorn workers, achieving better performance. Load testing was performed using BentoML endpoints, achieving 500 RPS for 1000 concurrent users. Integration with FastAPI and further testing is pending.



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/api/predict	8274	0	1900	3400	17000	2506.86	21	84921	24.94	39.9	0
POST	/api/predict-file?response_format=json	1451	64	1800	3000	3600	1896.49	230	37316	951.11	6.6	0.5
	Aggregated	9725	64	1900	3400	14000	2415.79	21	84921	163.12	46.5	0.5

Results with uvicorn



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/api/predict	8322	0	2300	5100	5900	2642.09	7	29194	25	26	0
POST	/api/predict-file?response_format=json	1493	0	2600	5000	5400	2876.64	91	5637	995	7.8	0
	Aggregated	9815	0	2300	5100	5600	2677.76	7	29194	172.55	33.8	0

Results with hypercorn

- **Challenge 2: BentoML Integration without Docker-Compose**
    - **Description:** Errors with mounting FastAPI app on BentoML via **@bentoml.mount_asgi_app(app)** to reuse existing FastAPI endpoints.
    - **Solution:** Used async calls to BentoML endpoints.
- **Challenge 3: Docker Port Accessibility**
    - **Description:** Port numbers exposed in Docker were not accessible from the URL displayed in the terminal.
    - **Solution:** Configured host=0.0.0.0 in Docker, allowing access from localhost, 127.0.0.1, and the computer’s IP.
- **Challenge 4: BentoML Integration with Docker-Compose (Ongoing)**
    - **Description**: Integration using Docker-Compose faces issues when pulling the Docker image or creating a Docker image by defining a Dockerfile. The server fails to find the specified module where service.py is present, or on successful execution, BentoML serve causes VSCode to crash.

- **Solution**: Ongoing. Recent development suggests not creating a Dockerfile for BentoML and instead creating a bentoml.yaml file and containerizing the Bento created.
- **Previous Implementation**: Previously I was creating a Dockerfile and mentioning it in the docker-compose so that requirements.txt can be used to install requirements while using the latest bentoml image available.

Conclusion

This week's efforts were focused on building and deploying an end-to-end project like a classification model, optimizing performance, and integrating various technologies. The successful implementation of BentoML (direct API calls) and Docker has improved scalability and deployment efficiency. Progress on BentoML integration using Docker-Compose is ongoing, with efforts to refine the deployment setup and further enhance performance.