Machine Learning Engineer Intern Assignment

Instructions

- This assignment should take approximately 8 hours to complete.
- You can submit this assignment even though you have not completed every requirement.
- Usage of LLM for assistance is acceptable. But we expect you to understand the code submitted, whether it is written by you or the LLM.
- To simulate a large data size and maintain the fairness of this assignment, the resource consumption of every program running together at the same time for each part of the assignment must not exceed these quotas.
    - CPU: 4 cores
    - Memory: 4 GiB
    - No GPU

Context

Suppose you are a Machine Learning Engineer of a restaurant recommendation project in LINE MAN Wongnai. Your data scientist colleagues have developed a machine learning model using PyTorch library to predict the probability of a user clicking any given restaurant. A user ID and candidate restaurant IDs will be provided in each recommendation request. Then the candidate restaurants are sorted either by the click probability in descending order or by displacement between the user and restaurants in ascending order.

Here is the list of attachments for this assignment

- `01_generate_dataset.py`: Generate synthetic dataset for this assignment.
- `02_train_model.py`: Model training program provided by data science team.
- `03_evaluate.py`: Evaluate the model using test dataset
- `04_inference.py`: Example usage of the model to generate restaurant recommendation.
- `requirements.txt`: Dependencies for the provided programs.
- `Dockerfile`:
- `data/model.pt`: Example model artifact.

Prerequisites

- Install Python 3.11 and dependencies specified in `requirements.txt`
- Execute `01_generate_dataset.py` to generate the dataset.
    - You can change the variables `TRAIN_NUM_FILES` and `TEST_NUM_FILES` to make the generated data fit your machine's memory. By increasing them, the number of rows per file will decrease and so will the memory usage. The default configuration consumes memory within 1GiB.
- The generated files will be written in `data/` directory
    - `train/train_*.parquet`: Training dataset indicating if a user clicks a restaurant.
    - `test/test_*.parquet`: Test dataset indicating if a user clicks a restaurant.
    - `user_features.parquet`: User preference features.
    - `restaurant_features.parquet`: Restaurant characteristic features.
    - `requests.parquet`: Generated API requests for performance testing.

Tasks

Part 1: Model Training

The original model training program (`02_train_model.py`) was developed on a high resource machine for development. The deployment resources are limited due to other services' usage.

1. Please identify the issues and optimize the program. Here are the constraints that **must not be changed** to simulate the large size of the model.
    - Every record of the data must be used for each epoch
    - Model architecture and optimizer must be the same
    - Epochs: 5
    - Learning rate: 1E-8
    - Maximum time per epoch: 60 seconds
    - Minimum accuracy: 0.7
2. Please write a performance report of the optimized training program with some evidences and answer the following questions
    - What are the issues?
    - How do you identify those issues?

Note: `02_train_model.py` requires a large amount of memory (17GB). You are advised to optimize this script first before running.

Part 2: Model Serving

Please implement an HTTP API server for the online model inference given that the user and restaurant data need to be retrieved from a **database server** for every request to allow feature update. You can decide the database server as you see fit to the use case. The server must be able to tolerate the load of **30 requests per second for 1 minute the 95th percentile of response time within 100 milliseconds** given request parameters (`requests.parquet`). The example of recommendation list generation is provided in `04_inference.py`.

HTTP API Specification

- Endpoint: `/recommend/<user_id>` (e.g. `/recommend/u00000`)
- Method: `POST`
- Request body
    - `candidate_restaurant_ids`: List of candidate restaurant IDs
    - `latitude`: User's latitude
    - `longitude`: User's longitude
    - `size`: Number of recommended restaurants (default: 20)
    - `max_dist`: Max **geodesic or great circle displacement** in meters between the user and restaurants. Restaurants further than this are considered irrelevant. (default: 5000) (Optional hint: H3 index)

    - `sort_dist`: Flag to sort restaurants by the displacement.
        - `sort_dist=true`: Sort restaurants by `displacement` field
        - `sort_dist=false` or not provided: Sort restaurants by `score` field
- Response: recommended restaurants in JSON format
    - `restaurants`: Recommended restaurants that are composed of
        - `id`: Restaurant ID
        - `score`: Click probability returned from the model
        - `displacement`: Geodesic or great circle displacement

Example


```
{
    "restaurants": [
        {
            "id": 2,
            "score": 0.2,
            "displacement": 800

        },
        {
            "id": 1,
            "score": 0.7,
            "displacement": 1000
        }
}
```

Submission Guideline

Please submit the work as a zip file (.zip extension) containing

- **`recommender`**: a directory containing code of part 1 and 2
    - Training and serving programs
    - **`requirements.txt`**: Python dependencies for executing the programs
    - **`Dockerfile`**: for building a docker image to execute the programs
    - **`docker-compose.yml`**: for starting API and database server
    - **`perf_test`**: a folder containing performance testing code
    - **`README.md`**: an instruction to execute the server
    - Scripts to start every service. Can be shell script or docker compose
    - Do **not** include any files under `data/` directory
- **`submission.pdf`**: Log how you approach the problems and the results.This is not to evaluate writing quality. Be concise and honest. Can be either Thai or English.
    - The performance report of part 1 and 2 must be included in this file.

Scoring Criteria

- Correctness and efficiency of the implementation
- Code structure and readability
- Fulfillment and detail of the design
- Task prioritization

Data Dictionary

- `train/train_*.parquet` and `test/test_*.parquet`
  - `user_id` (integer)
  - `restaurant_id` (integer)
  - `click` (integer)
    - 1; if the user clicks the restaurant
    - 0; otherwise
- `user_features.parquet`
  - `user_id` (integer)
  - `f00` - `f29` (float)
    - User preference features
- `restaurant_features.parquet`
  - `restaurant_id` (integer)
  - `f00` - `f09` (float)
    - Restaurant characteristic features
- `requests.parquet`
  - `user_id` (integer)
  - `latitude` (float)
  - `longitude` (float)
  - `candidate_restaurant_ids` (list of integer)
  - `size` (integer)
  - `max_dist` (integer)
  - `sort_dist` (boolean)