Data Gandalf's Installation Guide

Main Web Application

The main web application for Data Gandalf is containerized using Docker. While the various parts can be run independently, it's simplest to run the whole system using Docker and Docker Compose.

System Requirements

- Docker

Configuration

The system can be configured using the following environment variables. These environment variables should not need changing if the system is deployed with Docker Compose as the default values are properly configured for Docker Compose:

- `BACKEND_ADDRESS`
  - Default Value: `http://backend:8080`
- `DATABASE_ADDRESS`
  - Default Value: `database`

Additional configuration can be done in the backend/backend/config.py file. Specifically, the `MODEL_PATH` value can be adjusted to the desired path of the serialized model. By default, it should already point to a valid model.

Deployment

The system can be deployed simply with `docker compose up -d` from the same folder as the `compose.yaml` file. The web application can be reached by navigating to localhost:3000. Interacting with the web application tests the frontend, backend, and database parts of the main web application.

Data Subsystem

The data subsystem is not user facing but allows for administrators to add more datasets for the end user to interact with and discover.

Data Subsystem Dependency Installation

The data subsystem has it's own set of dependencies that need to be installed. A version of Python 3 must be installed on the system before proceeding. From the top level folder, running the following command should install all the needed dependencies.

```
python3 -m pip install -r data_system/requirements.txt
```

Configuring the Data Subsystem

The constant values in data_system/main.py can be modified as desired.

Running the Data Subsystem

The system can be run from the data_system folder.

```
python3 main.py
```

Database Dump

In order to get the data out of the database into a SQL file, PostgreSQL and all related CLI are installed (pg_dump, psql) as well as PGAdmin must be installed. The data can then be dumped with the following command.

```
pg_dump -U <username> <database_to_dump> > <dump_filename>.sql
```

Model Subsystem

The model subsystem trains the recommendation model based on pulled data from the data subsystem in order to generate a serialized model. This system is also not user facing but for administrator use.

Model Subsystem Dependency Installation

The dependency requirements for the model subsystem are the same as for the backend of the main web application. Python 3 must be installed on the system for the following to work. From the top level folder, running the following command should install all the needed dependencies.

```
python3 -m pip install -r backend/requirements.txt
```

Model Subsystem Configuration

1. Navigate to the model_system directory. Install and activate the same Python environment as the backend.
2. Add the directory to your `PYTHONPATH`.

   ```
   export PYTHONPATH=$PYTHONPATH:$(pwd)
   ```

3. Ensure that you have an active PostgreSQL database running. Run this command from the `model_system` directory to load the `pg_dump.sql` file to the database as a table named `Dataset`.

   ```
   psql -U postgres -f ../data_system/data_storage/pg_dump.sql
      {database_name}
   ```

4. Configure `training/config.py`:
   - Path where serialized model will be saved
   - Database settings
   - Table column names (Default settings match pg_dump.sql)
   - Feature weights

Running the Model Subsystem

```
python3 -m training.model_training
```