@hardikkgupta
Hardik Gupta

ml-e2e

Machine learning end-to-end project encompasses the entire workflow, starting from data gathering and preprocessing, through model building, and extending to advanced tasks such as implementing continuous integration and continuous delivery (CI/CD)

---

1 Model

1. **Type**: Logistic Regression (Binary Classification)
2. **Preprocessing**: Handles missing data via imputation (numerical: median, categorical: most frequent), standardizes numerical data, and one-hot encodes categorical data.
3. **Prediction Target**: Predicts whether a passenger survived the Titanic disaster (from the Titanic dataset) based on features like age, gender, class, etc.

2 Introduction

To understand machine learning end-to-end, it's essential to grasp each concept and the purpose of every associated file.

1. `app/model.pkl`: Serialized (saved) model using the `pickle` module.
2. `app/app.py`: Handles HTTP requests, loads the machine learning model, and defines the prediction endpoint.
3. `app/utils.py`: Contains helper functions for loading the model, validating input data, formatting predictions, and handling exceptions.

4. `Dockerfile`: script used to create a Docker image, which is a lightweight, portable, and self-sufficient executable package that includes everything needed to run this.

Serialized (saved) representation of a machine learning model, created using Python's `pickle` module—the standard method for serializing and deserializing data in Python. This file allows you to save trained machine learning models and load them later without needing to retrain. The `pickle` module converts objects into byte streams (serialization) and reconstructs them back into objects (deserialization). This process enables you to save models and share them without exposing the training data. Serialized files can be easily stored or transmitted over the internet because sequential (linear) formats are straightforward to read and write. Additionally, standard protocols ensure compatibility across a broad range of systems.

The file `app.py` is responsible for loading the machine learning model, assuming that the model incorporates the entire preprocessing pipeline. Upon loading, it extracts the numerical and categorical feature names that the model expects as inputs.

```
@app.route('/predict', methods = ['POST'])
```

The `/predict` endpoint is defined to handle `POST` requests, which send data from the client to the server (similar to submitting a form). The server listens for incoming requests at this endpoint, expecting a JSON payload where the features are provided in a dictionary format. The input data is then validated, processed, and converted into a `pandas.DataFrame` before being passed to the model for prediction.

```
app.run(host = '0.0.0.0', port = 5000)
```

The `host='0.0.0.0'` configuration makes the server accessible externally, allowing it to accept requests from any IP address. The `port=5000` specifies that the server will listen on port 5000.

The `logging.basicConfig()` function is used to set up the logging system, capturing log messages at the INFO level and higher, which helps in tracking and debugging the application's behavior.

3 CI/CD pipeline

The file `ci-cd.yml` defines the (Continuous Integration/Continuous Deployment) pipeline for automating the process of building, testing, and deploying the model.

1. Trigger the events: The pipeline can be triggered by two events:
   (a) `push`: Whenever code is pushed to `main` branch.
   (b) `pull`: Whenever code is pulled from `main` branch.
2. Jobs: The pipeline defines a job `build`, which runs on the latest version of Ubuntu.
3. Steps:
   (a) Checkout code: Uses the Github `actions/checkout@v3` action to checkout the repository's code.
   (b) Setup Python
   (c) Install dependencies
   (d) Set `PYTHONPATH`
   (e) Linting with `flake8` on `app/`, `tests/` and `models/`. It checks code style, syntax errors and quality assurance.
   (f) Test with `pytest`
   (g) Build Docker Image
   (h) AWS Integration: pipeline integrates with AWS for deployement and logs in AWS ECR and deploys the docker in EC2.

Appendix

A. Dockerfile

The Dockerfile allows you to package entire application with its dependencies in a portable/reproducible way ensuring consistent behaviours. `FROM python:3.12-slim` specifies the base image your container will be built from. 'slim' version is smaller in size. Sets the environment variables, working directory, `PORT` and import the required packages.

Image and Container

The term "image" essentially means a snapshot of everything your application needs to run. It is like a blueprint (static) and contains all the necessary files, dependencies and confirgurations. Container is a running instance of the image when you launch the image, it creates a container from it. A container is like a virtual machine, but lighter because it shares same OS as host.

B. Amazon Web Services

Elastic Container Registry

It is fully manager container registry service used to store, manage and deploy Docker images. ECR is mainly used in workflows where you need to manage Docker images. For instance, after building a Docker image, you push it to ECR and later pull it from ECR when deploying the image to services like EC2.

Elastic Compute Cloud

EC2 provide virtual servers in the cloud. These applications are used to run applications, deploy workloads and websites.