DOCKERIZED API SERVER

Introduction

The goal of this package is to provide a deployable API that predicts whether a person will survive a trip. It offers the user the choice between different models (a possible evolution would be to offer ensemble predictions for better accuracy).

Package

The code is organized as a package with a structured file architecture.

Note: here I only provide train accuracy. Not only should I provide test accuracy instead (with a simple train-test-split), but I could also offer more metrics to help users better grasp the different models' performances.

API server

The API server follows a traditional REST architecture.

At server startup, the models are trained and stored in pickle files, while their accuracy is stored in a csv file.

Here are the existing routes:

- Default model: check what the default model is and get its accuracy.
- Models list: lists all the models and their accuracy.
- Features: provides feature names and types (meant for developers).
- Predict: returns the prediction of whether someone will survive or not a trip. It can predict for many individuals at the same time, and the user is given the opportunity to choose the model they want (from the list given above).

Tests

When working on a big package it is recommended to use automated unit and integration testing. The aim is to implement unit tests for all functions and methods. This allows to refactor the code while ensuring code quality. So far, I have only implemented manual integration tests on Postman.

Docker

To make this API easy to share and to deploy, I need to dockerize it. I build an image of my API server with the minimum requirements to run it (python version, and necessary libraries). This image can then be shared and run as is on any machine.

Note: The first build can take up to 10min to install the packages.

Volume

It is recommended to use volumes in a docker app: a database specifically adapted to docker containers. This volume can be persisted and re-used among containers. Our api provides models and displays their accuracy. It is much more efficient to store those models and their accuracy across containers in a volume.

Web Deployment

The last step is to deploy the API server. I did not implement this.