Titanic Prediction Service

Overview

The Titanic Prediction Service predicts survival probabilities using passenger features. It offers both synchronous and asynchronous API endpoints built with FastAPI. The backend consists of a pre-trained LightGBM model for predictions, and the service supports containerized deployment using Docker.

You can download and contribute to this project and it is available at this github repository.
https://github.com/dilanHewawitharana/Titanic_Prediction_Service

Project Structure

```
- TITANIC_PREDICTION_SERVICE/
    - train_model/
        - model_training.ipynb
        - titanic/
            - test.csv
            - train.csv
    - app/
        - model/
            - Best_titanic_model.pkl
            - inference.py
        - routes/
            - async_routes.py
            - sync_routes.py
    - main.py
    - Dockerfile
    - requirements.txt
```

How to setup and run

How to run service as dockerize container

- Clone the repository in github

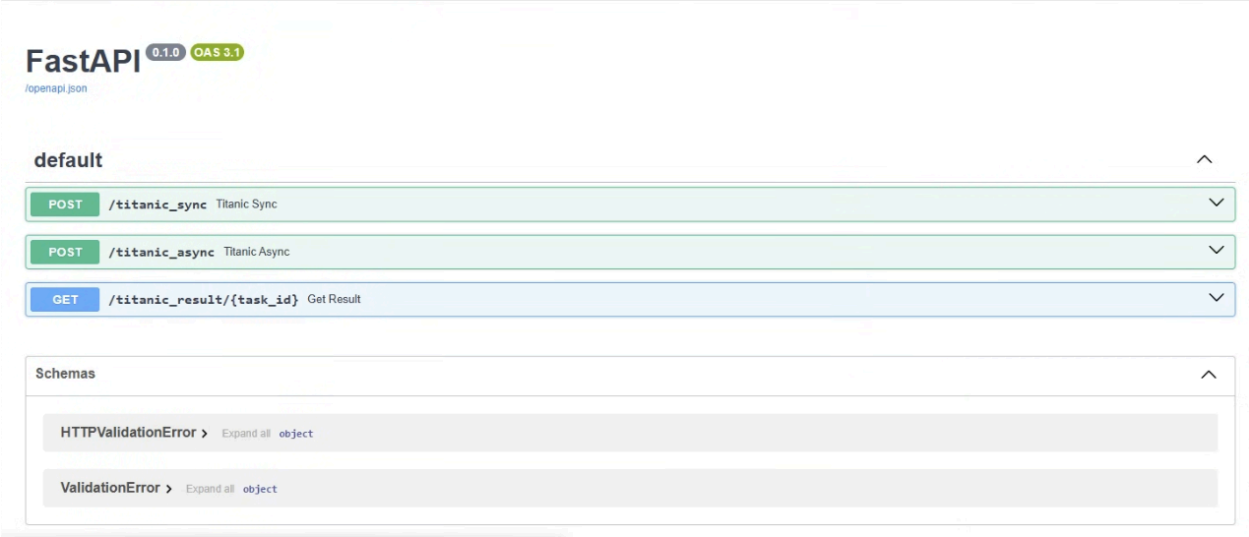
```
git clone https://github.com/dilanHewawitharana/Titanic_Prediction_Service.git

cd Titanic_Prediction_Service
```

- Build the Docker Image
  `docker build -t titanic-prediction .`

- Run the Container
  `docker run -p 8000:8000 titanic-prediction`

- FastAPI interface
  Navigate to: http://127.0.0.1:8000/docs



How to run service locally (for development)

- Clone the repository in github
  `git clone`
https://github.com/dilanHewawitharana/Titanic_Prediction_Service.git

- Install dependencies
  `cd Titanic_Prediction_Service`
  `pip install -r requirements.txt`

- Running the Application Locally
  `uvicorn main:app --host 0.0.0.0 --port 8000 --reload`

- FastAPI interface
  Navigate to: http://127.0.0.1:8000/docs

API Endpoints

Synchronous Prediction API

- Endpoint : `POST /titanic_sync`
- Request Body : `{\"data\": [3, \"male\", 22.0, 7.25, 0, 0, \"S\"]}`
- Response : `{"prediction": 0, "probability": 0.34}`
- Example : `curl -X POST http://127.0.0.1:8000/titanic_sync -H "Content-Type: application/json" -d "{\"data\": [3, \"male\", 22.0, 7.25, 0, 0, \"S\"]}"`

Asynchronous Prediction API

- Endpoint : `POST /titanic_async`
- Request Body : `{\"data\": [3, \"male\", 22.0, 7.25, 0, 0, \"S\"]}`
- Response : `{"task_id": "123e4567-e89b-12d3-a456-426614174000"}`
- Example: `curl -X POST http://127.0.0.1:8000/titanic_async -H "Content-Type: application/json" -d "{\"data\": [3, \"male\", 22.0, 7.25, 0, 0, \"S\"]}"`

Get Async Result

- Endpoint : `GET /titanic_result/{task_id}`
- Example : `curl -X GET http://127.0.0.1:8000/titanic_result/b778ade6-f91e-4ab7-bac5-474a82ef0617`

Model Training

LightGBM model used as the pre-trained models to train the model. Sklearn library used as the machine learning library to train models. The process includes data loading, preprocessing, feature engineering, model training, evaluation, and saving the trained model.

Data source : https://www.kaggle.com/c/titanic

Model training steps explanation

1. Import Libraries

Import necessary libraries such as pandas, scikit-learn, lightgbm, joblib..etc for data manipulation, preprocessing, model training, and saving the model.

2. Load the Dataset

Load the Titanic dataset from a CSV file named train.csv located in the titanic directory. Display the first few rows of the dataset to understand its structure. Analyze the data and identify missing data.

3. Handle Missing Values

Use SimpleImputer to fill missing values in the Age columns with the mean value of the column.

4. Drop Unnecessary Columns

Remove columns that are not relevant for prediction, such as PassengerId, Name, Ticket, and Cabin.

5. Encode Categorical Variables

Convert categorical columns Sex and Embarked into numerical values using LabelEncoder.

6. Feature Engineering

Create a new feature FamilySize by summing SibSp, Parch, and 1 (to include the passenger themselves).

7. Normalize Numerical Features

Scale numerical features Age, Fare, and FamilySize using StandardScaler to ensure they have a mean of 0 and a standard deviation of 1.

8. Split the Data

Separate the features (X) from the target variable (Survived) and Split the data into training and testing sets using an 80-20 split ratio.

9. Initialize and Train the LightGBM Model

Initialize a LightGBM classifier and Train the model using the training data.

10. Make Predictions

Use the trained model to predict survival on the test set.

11. Evaluate the Model

Calculate the accuracy of the model by comparing the predicted values with the actual values and print the model accuracy.

12. Confusion matrix

The confusion matrix helps to understand the performance of your classification model by showing the true positives, true negatives, false positives, and false negatives.

13. ROC Curve

The ROC curve helps to visualize the trade-off between the true positive rate and the false positive rate. It's useful for understanding the model's performance across different thresholds.

14. Save the Trained Model

Save the trained LightGBM model to a file named best_titanic_model.pkl.

Conclusion

This project successfully implements a Titanic passenger survival prediction API with both synchronous and asynchronous endpoints. It is containerized with Docker for ease of deployment and scalability.