AIDEA

Aidea tool @ripik

It is a Minimalistic No-Code platform to convert your AI Ideas into Reality

Building guide

> To build the container

```
sudo docker build -t aidea .
```

> To run the container

```
sudo docker run -p 8501:8501 --restart always -it aidea
```

> IF the port comes out to be occupied after running the container and then running again after stoping then,

```
 sudo docker kill $(sudo docker ps -q)
```

General introduction

Directory structure-

```
.
├── Backend
│   ├── autoclassifier.py
│   ├── autonn.py
│   ├── autoregression.py
│   ├── autotimeseries.py
│   └── Session.py
├── dashboard.py
├── datasets
│   ├── apple.csv
│   ├── banknote.csv
│   ├── diabetes.csv
│   ├── pima.csv
│   ├── servo.data
│   ├── shampoo.csv
│   └── temp.csv
├── Dockerfile
├── Frontend
```

```
│   ├── aidea.png
│   ├── classif_dashboard.py
│   ├── load_data.py
│   ├── logo.png
│   ├── regress_dashboard.py
│   └── ts_dashboard.py
├── README.md
├── requirements.txt
└── Trained_Models
    ├── decoder.pkl
    ├── model3be2b2da-f36c-4def-a95a-627c725865f2.pkl
    ├── model3be2b2da-f36c-4def-a95a-627c725865f2.zip
    ├── modelade6f9db-b489-464f-8f29-4a5f4af7b73c.pkl
    └── modelade6f9db-b489-464f-8f29-4a5f4af7b73c.zip
```

> **Note:** All the auto files starting from auto are backend files and rest are frontend (Later this is organised properly in file directory structure)

General Programm Flow

> **Frontend**

- First of all dashboard.py is called which creates a basic fontend dashboard structure for navigation and choosing the task.
- After the task is choosen dashboard.py would redirect to the relevent dashboard from `Frontend` folder for example, `classif_dashboard.py` for classification tasks etc.
- in `classif_dashboard.py` the data is user get s an option to upload the data or to use exesting data.
- then it is analysed in the frontend file only to show relevent information like data imbalance etc.
- Other options like generate EDA and run experiment buttons are created which uses functions defined in backend.

> **Backend**

- The runexperiment button in Frontend calls the `run_experiment` function of the relevent object defined in backend for example in classfication task, it calls `autoclassifier.run_experiment()`
- The `run_experiment` method calls the `training()` function internally which returns the best model trained depending upon the current dataset, along with that it times the training process and after `training` it plots few curves for better understading of the dataset and outputs the pipeline of the model with highest accuracy.
- **Training()**- The training function first fits the dataset on different available models and compaires their accuracies using HalvingGridSearchCV from skLearn. And then it performs the hyper-parameter tuning for the best model.
- **Fit()**- The fit function have a list of the models on which the dataset should be fitted and it uses HalvingGridSearchCV() object to find the model which fits the dataset best using F1 score as metric for judging the accuracy.

- **tune()**- The tune function again searchs for the best hyperparameters based on the model with highest F1 score using HalvingGridSearchCV and saves the `best_estimator` and `best_pipeline` objects in the class.

Tech stack used

Frontend

- StreamLit and Dask.
- numpy and pandas for effecient and vectorised computation.
- PandasProfiling for generating the EDA report.
- sklearn for loading and showing availabe datasets to the user.

Backend

- SkLearn for creating model, training model, and searching best hyperparameters for the dataset.
- autokeras for finding model which fits the dataset.
- scikitplot for plotting different graphs and matrics.
- xgboost for regularizing gradient boosting framework in python.