Usage Instructions

On the terminal, follow these steps:

Step 1: Clone github repo & navigate to the repo root directory

Command 1: git clone https://github.com/azhan10/hiscox-test.git

Command 2: cd hiscox-test

Step 2: Create a virtual environment with Python 3

Command: virtualenv -p python3 venv

Step 3: Activate virtual environment and then install libraries

Command 1: source venv/bin/activate

Command 2: pip install -r requirements.txt

Step 4: Then compile and run the run.py in the root directory

Command: python run.py

Step 5 (optional): You can run the unit testing as well.

Command 1: cd unittest

Command 2: python <Python script>

During running, you may experience a pop-up screen of graphs; it seems that to continue, you need to cross each one. The output is also stored in the data file.

You may also experience this error: "failed to execute PosixPath('dot'), make sure the Graphviz executables are on your systems" or similar. You need to run conda install python-graphviz. It's not an error of the script; it requires a GPU (if not installed already).

More details about Graphviz are here: https://stackoverflow.com/questions/73830695/failed-to-execute-posixpathdot-make-sure-the-graphviz-executables-are-on-y

Alternatively, you can use a IDE like Spyder which doesn't require Graphviz installation.

I left a video demo as well to see it in action.

Solution Overview

The chosen approach is running the scripts through the terminal. It follows the class structure method as it’s robust and clean for production level. The run.py file produces all the data in one go. The data, model and statistics are separated into modules, and then I call them as functions in the run.py. The problem I’m solving is running it in the most efficient way possible to improve efficiency. The design is simplistic, and it’s not over-engineered. The tradeoff would be that the entire process is run in one file (you can separate them in separate run files as an improvement, and then call the files in each run stage). There are some restrictions on hyperparameter flexibility changes (the ratio in the code shows the hyperparameters to be more fixed, which is why I took this approach). Another tradeoff is that it’s coupled, but overall, the analyses are not complex. I chose the terminal approach as the project is data exploration and predictions. You can customise some analyses if you require. It doesn’t need to be complicated. Also, saving the files to a directory. I assume the user will explore hidden patterns between the target and features (assuming the data is real), change the model hyperparameters occasionally to find the best one, so there is flexibility. The irritating part is that you may need to close each diagram pop-up for it to continue compiling.

Improvements

Further improvements you can apply:

- Github workflow or similar to compile push to repo and run tests.
- Storing the model as Joblib or Pickle or docker image
- Bypass the image popup image per graph and just store them in the folder. That way it just runs without barriers.
- Maybe add more hyperparameters on some function to give more flexibility.
- Add error handling to prevent unusual (e.g. only accept XGBoost due to the non-existence hyperparameters. If you use like a Random Forest or SVM, it will break return an error).
- Add more unit testing (there are comments for other testing).