Module 4: TDD

Technical Description Document - Push Notifications DS

Context

We are building a new product to enable the sales team to target customers to be sent a push notification to nudge them to buy a certain product that has been previousle selected by them.

This tool is fractioned in 3 parts:

- Frontend: being carried out by the FE team.
- Backend engineering: all the processes for having the data available and serve it to the frontend. Engineering team is on charge of this.
- Data science: development of a predictive model that suits the sales team need. This is covered below.

Goal

Developing a machine learning model that, given a user and a product, predicts if the user would purchase it if they were buying with us at that point in time.

We should only be focusing on purchases of at least 5 items, since it is a requirement coming from the sales team. They expect from us to ship a Proof Of Concept (POC) in a week.

Data

We will be using our groceries dataset *feature_frame_20210304.csv*. This is a well known dataset by now since we have being working on it over the latest several weeks. For this reason, this project skips the Exploratory Data Analysis (EDA) phase as it is should already be available in previous reports.

The only tweak required in the dataset is filtering orders to keep only those with at least 5 items, since they are our target.

Approach

Milestone 1: exploration phase

Given that we have a clear understanding of our data, we just jump directly on building the predictive model. Firstly, we filter the data to only those orders with 5 items or more to build a dataset to work with.

Secondly, we seek to find the best model so we seek to explore different linear and non-linear models with different parametrisations and regularisations.

The outcome is expected to be a report/notebook/documentation on what worked and what did not and whys Most importantly, we need to have a final model selected to move to milestone 2.

Milestone 2: MVP code

Using the outcomes of milestone 1 MVP ready code must be generated to share with the engineering team for deployment. This code will be run within an API developed by the engineering team. For easing their work we will be alrady generating 2 .py files:

1. A fit code, that receives the model parametrisation, loads the data, trains the model and saves it to disk. The data retrieval should be done in another function for modularity as in the future we may want to read from different sources. Since we are not implementing any ML engineering frameworks, the default model name will be "push_yyyy_mm_dd" where yyyy_mm_dd is the training date. The output of the function is a dictionary as below containing the model name and model path. Feel free to expect further fields in the input dictionary, but always document and have default values if necessary.

```
    def handler_fit(event, _):
        model_parametrisation = event["model_parametrisation"]
        [your code here]
        return {
            "statusCode": "200",
            "body": json.dumps(
                {"model_path": [your_model_stored_path],
                }
            ),
     }
```

1. An inference function predict(x), that receives a json with a field "users" that contains:

```
    {
        "user_id": {"feature 1": feature value, "feature 2": feature value, ...},
        "user_id2": {"feature 1": feature value, "feature 2": feature value, ...}.
    }
```

Output should be a json with a field "body" with fields "prediction":

```
    {"user_id": prediction, "user_id2": prediction ...}
```

You are again free to make any changes you deem necessary, but always document. Base of the method:

```
    def handler_predict(event, _):
        data_to_predict = pd.DataFrame.from_dict(json.loads(event["users"]))
        [your code here]
        return {
            "statusCode": "200",
            "body": json.dumps(
                {
                 "prediction": {dict of predictions}
                }
            ),
      }
```