explainable_credit_scoring

The code for the *Improving Credit Default Prediction with Explainable AI* masters project.

Please see the following step-by-step process to run the project.

Step 1: Build the database required for execution.

Follow the instructions in the following link: https://github.com/i-am-yohan/home_credit_2_postgres This will download all of the required files and upload them to a postgres database. This is not the most efficient method of doing it so feel free to only use this as a guide.

Step 2: Executing the models and submitting to Kaggle.

Execute the `Model_Execute.sh` to build the models and make the Kaggle submissions. This is done as follows:

```
./Model_Execute.sh <postgres id> <postgres password>
```

Ensure this is executed in the same folder as the Scripts sub-folder.

The `Model_Execute.sh` has the following commands:

2.1 Create directory for the Kaggle submissions

The following command creates a directory for the kaggle submissions.

```
mkdir Kaggle_Submissions
```

Given that the dataset used here is a kaggle dataset, in order to analyse the competition score, a dataset of predictions must be submitted to Kaggle in CSV format. The CSVs resulting from each model will be stored in the created sub-folder and will be submitted to Kaggle later in the process. The CSVs are created in each of the model build processes.

2.2 Install python requirements

The following command installs all of the Python packages required for the project.

```
pip3 install -r Scripts/Requirements.txt
```

This installs the python packages *only*. The required R packages are installed within the R script executed later in the process. The following python packages are installed:

```
psycopg2
statsmodels
SQLAlchemy
sklearn
pickles
XGBoost
LightGBM
argparse
progressbar
plotly
imblearn
```

Note that the `psycopg2` package is used to access and run commands on the `postgres` DB. Before executing it is probably best that the use of Postgres is set up and tested beforehand.

2.3 Data preprocessing

The data cleaning and feature engineering phase is executed as follows:

```
python3 Scripts/01_Data_Cleaning_and_Feature_Engineering.py $1 $2
```

This creates a table `abt.abt_full` with all of the observations with all of the engineered features.

The process where the train, test and kaggle submission datasets are sampled from the main table resulting from the previous command is executed as follows:

```
python3 Scripts/02_Sampling.py $1 $2
```

This creates three tables:

```
abt.abt_kaggle_submission
abt.abt_train
abt.abt_test
```

2.4 Model build

This is the section where the models are trained, validated and evaluated. As previously stated the CSVs to be submitted to kaggle are created by each individual model script. The evaluation (with the exception of the kaggle submission) is printed to the log when the model script is executed. The build by each model is as follows:

Traditional credit scorecard

The credit scorecard model is trained, validated and evaluated using the following command:

```
Rscript Scripts/03_1_Scorecard_Dev.R $1 $2
```

This is done in R due to the requirement of the `scorecard` package. This is the only model built in R with the rest being built in Python.

The model evaluation is printed to the log as follows:

Train

```
$Confusion_Matrix

     FALSE   TRUE
  0 155949  69329
  1   5892  13789

$Accuracy
[1] 0.6929241

$Precision
[1] 0.1658967

$Recall
[1] 0.700625

$False_Positive_Rate
[1] 0.3077486

$AUC
[1] 0.7621526

$F1
[1] 0.2682711

NULL
```

Test

```
$Confusion_Matrix

    FALSE  TRUE
  0 39070 17468
  1  1531  3434

$Accuracy
[1] 0.6910882

$Precision
[1] 0.1642905

$Recall
[1] 0.6916415

$False_Positive_Rate
[1] 0.3089603

$AUC
[1] 0.7571097

$F1
[1] 0.265512
```

Random forest

The Random Forest model is trained, validated and evaluated using the following command:

```
python3 Scripts/03_2_RandomForest_Dev.py $1 $2
```

The model evaluation is printed to the log as follows:

Train

```
{'Accuracy': 0.6841662116580853, 'Precision': 0.16295647093193105, 'Recall': 0.7085423944109241, 'AUC': 0.7599507847734416}
```

Test

```
 Evaluation for test set
{'Accuracy': 0.6846657886607157, 'Precision': 0.16140235603322853, 'Recall': 0.6926485397784491, 'AUC': 0.7514528901717733}
```

LightGBM

The LightGBM model is trained, validated and evaluated using the following command:

```
python3 Scripts/03_3_LightGBM_Dev.py $1 $2
```

The model evaluation is printed to the log as follows:

Train

```
{'Accuracy': 0.6982553184975021, 'Precision': 0.17241743805022425, 'Recall': 0.7251825976500477, 'AUC': 0.7829692807261011}
```

Test

```
 Evaluation for test set
{'Accuracy': 0.6960473472838723, 'Precision': 0.16932414856206945, 'Recall': 0.7079556898288016, 'AUC': 0.7738585286791403}
```

XGBoost

The XGBoost model is trained, validated and evaluated using the following command:

```
python3 Scripts/03_4_XGBoost_Dev.py $1 $2
```

This is the best performing model overall. It is because of this that the created XGBoost model is saved to a pickles file `Final_Model_XGBoost.pkl`. This will be used in the explanation phase.

The model evaluation is printed to the log as follows:

Train

```
{'Accuracy': 0.7163297902197819, 'Precision': 0.1834029907670793, 'Recall': 0.7329946014607812, 'AUC': 0.7999641421746178}
```

Test

```
 Evaluation for test set
{'Accuracy': 0.7140302099084598, 'Precision': 0.1786896095301125, 'Recall': 0.7069486404833837, 'AUC': 0.7812163228132317}
```

2.5 Submit the results to Kaggle

In this phase the external results are submitted to Kaggle using the Kaggle API. Details on how to install the Kaggle API can be found using the following link: https://github.com/Kaggle/kaggle-api . The commands are as follows:

```
~/.local/bin/kaggle competitions submit -c home-credit-default-risk -f Kaggle_Submissions/01_Scorecard.csv -m 01_Credit_Scorecard_De
~/.local/bin/kaggle competitions submit -c home-credit-default-risk -f Kaggle_Submissions/02_Random_Forest.csv -m 02_Random_Forest_D
~/.local/bin/kaggle competitions submit -c home-credit-default-risk -f Kaggle_Submissions/03_LightGBM.csv -m 03_LightGBM_Depl_Sub
~/.local/bin/kaggle competitions submit -c home-credit-default-risk -f Kaggle_Submissions/04_XGBoost.csv -m 04_XGBoost_Depl_Sub
```

The AUC scores resulting from each submission are then checked by signing into Kaggle and checking the submissions.

2.6 Explanation phase data preparation

The following command prepares data for the explanation phase:

```
python3 Scripts/04_Expl_Data_Create.py $1 $2
```

This creates the table `expl.base_table` . This contains all observations from the train, test and kaggle submission tables. Each observation has the prediction made in predicted probability, predicted credit score and predicted binary outcome format. This is the main table that will be used in the explanation phase.

Step 3: Extracting the counterfactuals.

In this phase, the counterfactual search algorithm is deployed. The following inputs to the counterfactual search alogorithm are as follows: 1. `sk_id_curr` - The observation of interest. 2. `target_score` - The target counterfactual score. 3. `lower_bound` - The lower bound score to use when searching for counterfactuals. 4. `upper_bound` - The upper bound score to use when searching for counterfactuals. 5. `n_samples` - The number of samples to take at each iteration.

The script is executed as follows:

```
python3 Scripts/05_Extract_CF.py <postgres id> <postgres password> <sk_id_curr> <target_score> <lower_bound> <upper_bound> <n_sample
```

This loads a table to the postgresDB called `expl.plot_table` . This contains the counterfactual information.

Step 4: Visualize the counterfactuals

The `06_Visualization.py` script uses the `plot_table` to create graphs to help visualize the project. It is executed as follows:

```
python3 Scripts/06_Visualization.py <postgres id> <postgres password> <Counterfactual ID>
```

The `<Counterfactual ID>` parameter is the counterfactual value which requires visualization. The `<Counterfactual ID>` is a natural number that corresponds to the `cf_id` column in the `expl.plot_table` table. These are output as plotly graphs sent to the web browser.