EFREI PARIS

APPLICATIONS OF BIG DATA
ML PROJECT

MEDINA HADJEM

By

Karthikeyan PAVADE

And

Nayel HAMANI

28th octobre 2020

SOMMAIRE

1 Project Setup

1.1 Project configuration

Requirements

An IDE with python 3.6 at least. (Anaconda, Pycharm, Vscode,etc.)

Dataset

The project is based on the exploitation of a dataset of Home credit Risk Classification. By default the dataset can be found on the above hyperlink:

https://www.kaggle.com/c/home-credit-default-risk/data

The aim of the project is to predict whether if a client is able to repay a loan sanctionned by the bank or not. And to achieve this, we have 2 files :

- Application_train.csv : where we can find our features and the label.
- Application_test.csv : that is used to the prediction case for later case during the project.

1.2 Data processing and feature engineering

Our aim is to predict the label "Target" that we can easily find from one of the column in the csv application_train. This label can only take one of these folowing values:

- 0- Taken value if the concerned client is able to repay the loan to the bank
- 1- Taken value if the concerned client is NOT able to repay the loan to the bank

To resume our processing, first of all we saw that there we are some missing values that can be cleaned or replaced according to the type of the column.

Then we use some of features selections like pearson Correlation and so on that we explained further in documents.

Our exploartion and analysis are already well detailled in attached files such as documentation files. The purpose is to give the possiblity for a beginner to understand well our process for the data for each function and step that we used.

1.3 Models and scores

In our actual project we respected the criteria expected in the project and so we have 3 different models:

- Xgboost
- Gradient Boosting
- Random forest

During our first exploration and computation of the project, we were used to have a large amount errors and average scorings. But, with some cleaning process like missing values and normalization, we were able to pass from 75% into 90-95% of accuracy.

Something important in Machine Learning is to check if we are not overfitting. And in our case, we have to manage our data and check very often that this process is not done in our case.

Furthermore, we were used to play with our hyper-parameters like Learning rate, number of estimators, samples, etc. Thanks to that, we were able to achieve some good scores and the improvement is really good and as we know the score value depends most on the data, our model, and hyper parameters.

2 Best Pratices

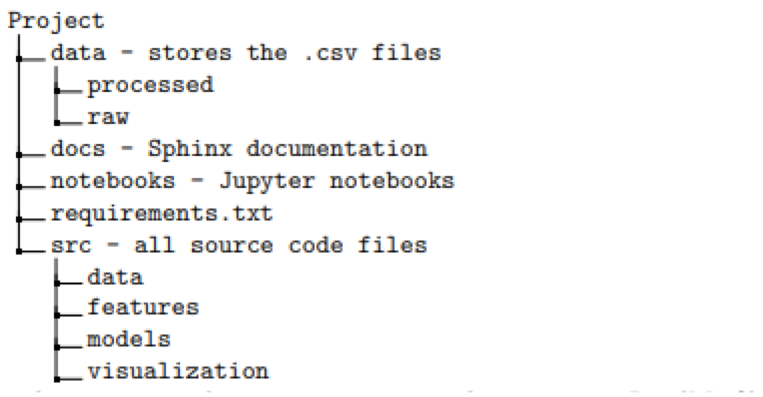
Code versioning and repository

For our case we can find our repository link:

https://github.com/Naykoh/repo-projet-bd

CookieCutter

We used it to structure our project with the folowing structure:



```
Project
 |__data - stores the .csv files
 |   |__processed
 |   |__raw
 |__docs - Sphinx documentation
 |__notebooks - Jupyter notebooks
 |__requirements.txt
 |__src - all source code files
     |__data
     |__features
     |__models
     |__visualization
```

Figure 1: structure of cookiecutter

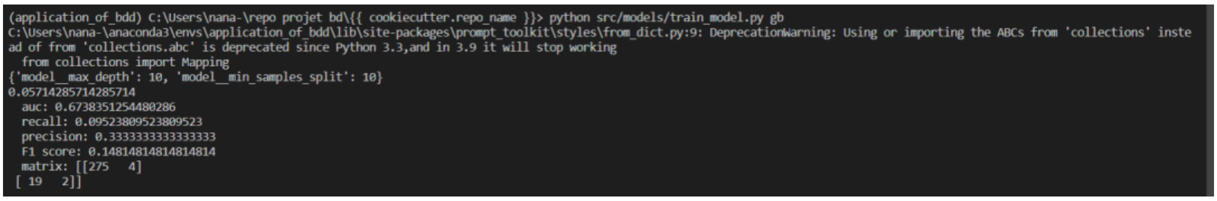
Documentation

As it was requiested and adviced, we used a lot of documentations simply user-friend using a Sphinx Documentation. Every step and function that we used are implictely explained thanks to the document corresponding to that process. From usage of scripts into the use of our notebooks.

3 MLflow

Terminal process

First of all when we compute our python script that train our model, we get the folowing output in our terminal.



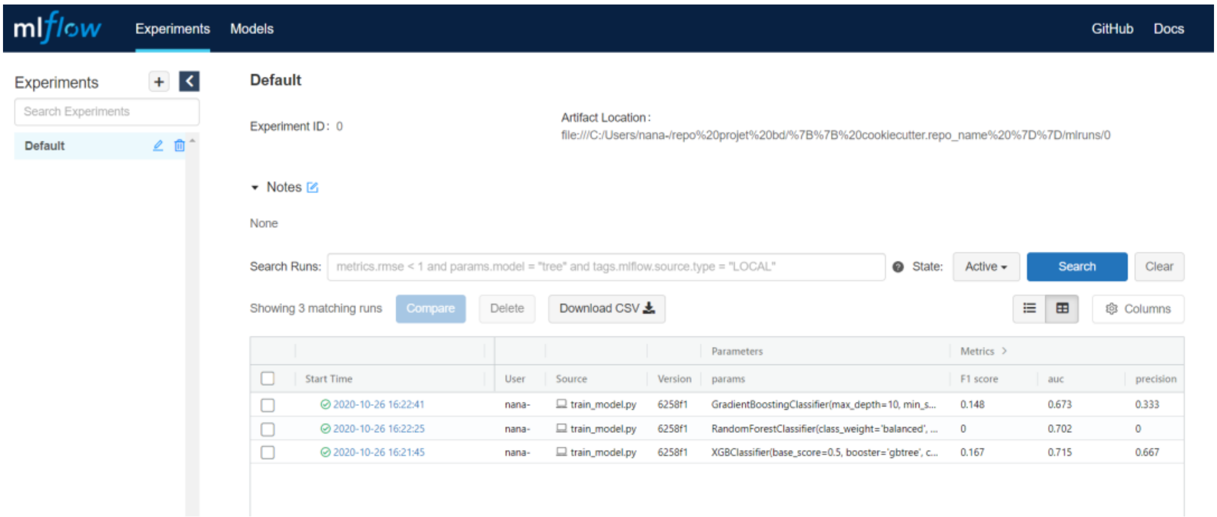
```
(application_of_bdd) C:\Users\nana-\repo projet bd\{{ cookiecutter.repo_name }}> python src/models/train_model.py gb
C:\Users\nana-\anaconda3\envs\application_of_bdd\lib\site-packages\prompt_toolkit\styles\from_dict.py:9: DeprecationWarning: Using or importing the ABCs from 'collections' inste
ad of from 'collections.abc' is deprecated since Python 3.3,and in 3.9 it will stop working
  from collections import Mapping
{'model__max_depth': 10, 'model__min_samples_split': 10}
0.05714285714285714
  auc: 0.6738351254480286
  recall: 0.09523809523809523
  precision: 0.3333333333333333
  F1 score: 0.14814814814814814
  matrix: [[275   4]
 [ 19   2]]
```

Figure 2: Script launched from terminal for Gradient boosting

In this case, we used our train model script with the gradient boosting Model. However, the process remains the same for every other models that we used during our project: Random Forest or Xgboost. For this case, we can see and accuracy of 67%. And its used to reject in the output the confusion matrix aswell.

MLflow UI

When we need to check out our results in a better interface, we can simply use the Mlflow UI interface. It can be easily been access with the localhost. And we can therefore track our results and our models.



Default

Experiment ID: 0

Artifact Location: file:///C:/Users/nana-/repo%20projet%20bd/%7B%7B%20cookiecutter.repo_name%20%7D%7D/mlruns/0

Notes

None

Search Runs: metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.type = "LOCAL"

State: Active

Showing 3 matching runs

					Parameters	Metrics		
□	Start Time	User	Source	Version	params	F1 score	auc	precision
□	2020-10-26 16:22:41	nana-	train_model.py	6258f1	GradientBoostingClassifier(max_depth=10, min_s...	0.148	0.673	0.333
□	2020-10-26 16:22:25	nana-	train_model.py	6258f1	RandomForestClassifier(class_weight='balanced', ...	0	0.702	0
□	2020-10-26 16:21:45	nana-	train_model.py	6258f1	XGBClassifier(base_score=0.5, booster='gbtree', c...	0.167	0.715	0.667

Figure 3: Mlflow Ui with parameters tunning and differents models

In the picture, we can indentify the 3 models that we ran, and the accuracy for each of them. In this case XGBClassifer seems to be the best one in term of accuracy with 71-75%.

Furthermore we can as well find out some tunning with hyper parameters like "max_depth, class_weight, learning rate, etc." that are used for each model. In our case, some of them were automatically choosed by our programm in order to compute with best parameters possible.

4 XAI with SHAP method

The purpose of this methode is to track our results and interprets them according to the human understand knowledge methods. That means that we can easily interprets our results thanks to simple illustration for example: graphs, hists, and point values, KPI. In order to kick off some idea or even conclusion.

4.1 Shap on the whole dataset

The figure above illustrates what we found onthe whole dataset with SHAP plot:

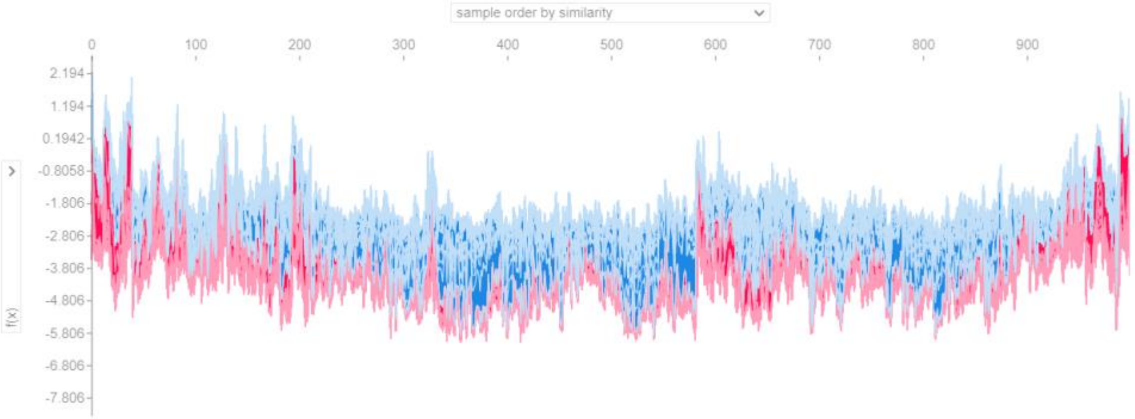
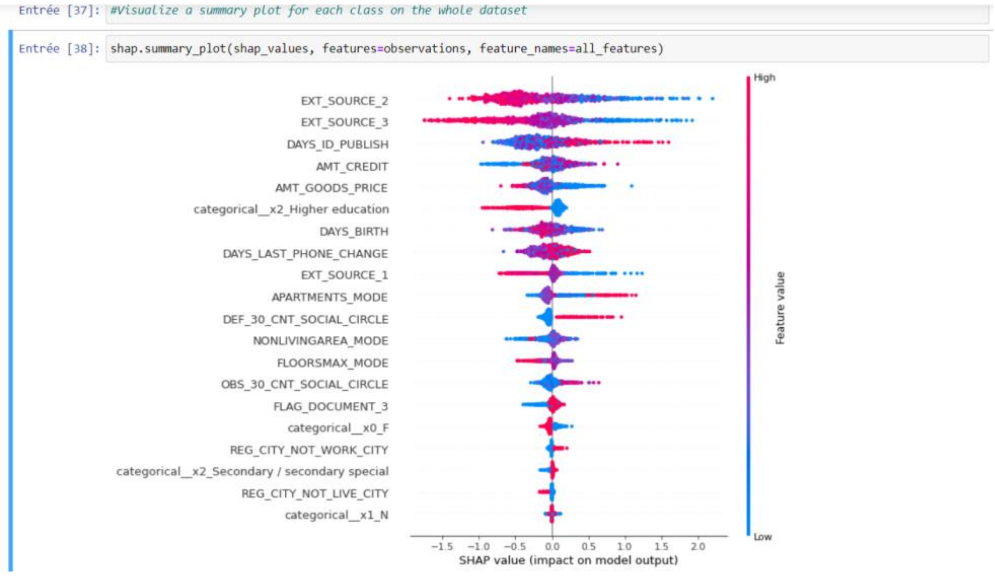


Figure 4: General dataset SHAP plot

Pavade | Hamani

4.2 SHAP summary plot



```
Entrée [37]: #Visualize a summary plot for each class on the whole dataset
```

```
Entrée [38]: shap.summary_plot(shap_values, features=observations, feature_names=all_features)
```

Figure 5: summary PLOT

To resume if we take a chosen case with the target is 0. We'll for example have some features that would have more impact on their ability to repay their loan. IT could be Days_birth, or higher_education. And it could be realiable in our sumamry plot when we are comparing the impact of the SHAP value among other features.

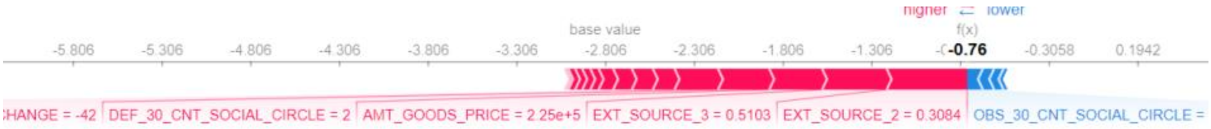


Figure 6: Single point interpretation: 0