Rapport projet 1

Abderrahmane ADA

Léo XU

PART 1: Building ML Models

Data preparation

First, we did some data preparation.

We decided to keep columns that has more than 50% of missing value.

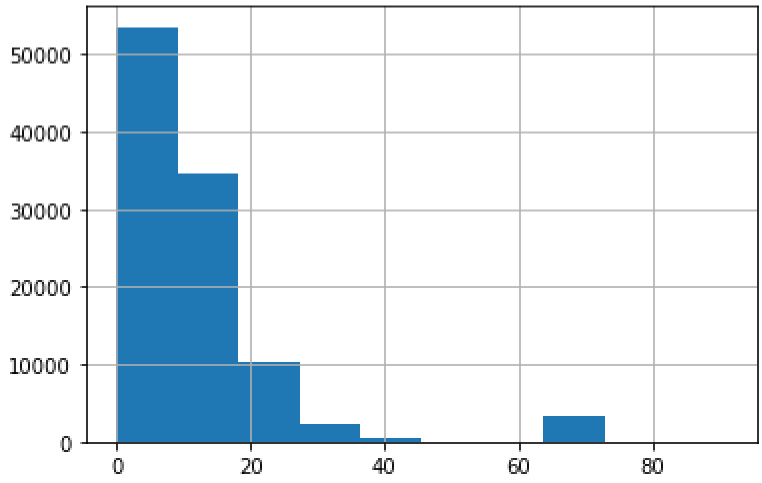
	Total	Percent
COMMONAREA_MEDI	214865	69.87
COMMONAREA_AVG	214865	69.87
COMMONAREA_MODE	214865	69.87
NONLIVINGAPARTMENTS_MODE	213514	69.43
NONLIVINGAPARTMENTS_AVG	213514	69.43

Then we looked at the columns that has the most impact on the Target columns by looking at the correlations between them and the Target columns.

Most Positive Correlations:	
REG_REGION_NOT_LIVE_REGION	0.005576
REG_REGION_NOT_WORK_REGION	0.006942
CNT_CHILDREN	0.019187
FLAG_WORK_PHONE	0.028524
LIVE_CITY_NOT_WORK_CITY	0.032518
OWN_CAR_AGE	0.037612
DAYS_REGISTRATION	0.041975
REG_CITY_NOT_LIVE_CITY	0.044395
FLAG_EMP_PHONE	0.045982
REG_CITY_NOT_WORK_CITY	0.050994
DAYS_ID_PUBLISH	0.051457
REGION_RATING_CLIENT	0.058899
REGION_RATING_CLIENT_W_CITY	0.060893
DAYS_BIRTH	0.078239
TARGET	1.000000

We decided to keep the columns that has more than 3% correlations.

After that we decided to replace the OWN_CAR_AGE missing value by replacing it by the median.



since it seems more appropriate in this case due to the outlier.

Finally, we saved our final data frame into a new csv file. We didn't do the scaling part and all the Nan replacement (reason after)

Model training

Before training our models, we used the predefined function nan_to_num() to replace the nan value of our dataframe into 0 (The missing value are found in columns like Days birth, days id publish and days registration that's why I didn't replace it by the median like our last examples) . It also turns our dataframe into a np array.

The dataset is separated in the following format:

X: all the data except TARGET

y: TARGET

We also use the StandardScaler() function to scale our features.

We used the train test split method to train our model efficiently, using 80% of our data for the training phase and 20% for the testing phase.

Now that our data are in a food format, we just train our 3 differents models and save them into sav file by using pickle.dump().

Prediction

For the prediction we just have to load our saved model by using pickle.load()

Sphinx

Sphinx is a free document generator. To install Sphinx, we just had to use: pip install Sphinx.

After that, we go to the directory of the project and use the quick start of sphinx to generate the documentation of the project. You can find it in the repository of our project.

Git

For this project we had to set-up a git environment. First, we created a repository on github.com.

Then in our local environment we created a git repository by using `git init` in a repository. We didn't create alternative branch because we didn't that it was necessary.

Upon each modification we add the file into the commit queue using `git add` and then we commit it using `git commit -m "the commit message"` and finally we set up an upstream to our online github repository by using this command `git branch --set-upstream` https://github.com/pseudo/repository

Then just have to use `git push` to push all our data into the stream.

PART 2: MLFLOW

Installation & working environment

For this part we used a virtual machine and worked in a ubuntu environment due to the misfunctioning in windows.

To install MLFLOW in our machines we just needed to type this short command in the conda environment: `pip install mlflow`

Parameters tracking

In order to track our parameters, we had to put our models in the `mlflow.start_run():` section, and then use `mlflow.logparam()` and `mlflow.logmetric()`. That will allow us to track the metrics of our models according to the parameter chosen. And `mlflow.sklearn.log_model()` is used to keep our models.

All the history will be generated inside a mlruns directory. You can either chose to look at it in your local directory and files or you can choose to use the mlflow ui by typing this command: `mlflow ui`.

Mlflow ui give us a nice interface to track and compare our parameters. We also find out that mlflow ui had to be itinialized every time if you want to be up to date as it is not updating live.

Run ID:	e466ac54bd5a4a91ab96a75836f2e92a	e69d1282bc6f4eb0bb5ef722a3f1bcd8
Run Name:		
Start Time:	2020-10-28 10:57:30	2020-10-27 16:38:35
Parameters		
n_estimators	1	20
random state	1	0
Metrics		
mae	2.564	8.14
r2	-0.035	-1.22
rmse	7.44	10.9

In this example we are comparing 2 different models.

Reusable and reproducible format

Since our code are already separated in different notebooks and structured from the previous part. It's easier to do the transition.

Instead of using a function that are waiting for arguments, we will now use systems arguments. For that we import the sys library.

```
def train(lr,estim, rand, mf,dep):
```

```
estim = int(sys.argv[1]) if len(sys.argv) > 1 else 1
lr = float(sys.argv[2]) if len(sys.argv) > 2 else 0.5
mf = int(sys.argv[3]) if len(sys.argv) > 3 else 10
dep = int(sys.argv[4]) if len(sys.argv) > 4 else 10
rand = int(sys.argv[5]) if len(sys.argv) > 5 else 5
```

After that we converted our. ipynb files into .py files so that it is executable without jupyted.

Using jupyter nbconvert –to script .ipynb

Then we have to create the MLproject which will launch our python code and the conda.yaml which contains all the requirements needed to use the code.

```
name: Project

conda_env: /home/leo/Python/MLFLOW/Gradient_Boosting/conda.yaml

entry_points:
  main:
    parameters:
      estim: {type: int, default: 5}
      lr: {type: float, default: 0.5}
      mf: {type: int, default: 10}
      dep: {type: int, default: 10}
      rand: {type: int, default: 5}
    command: "python Gradient_Boosting.py {estim} {lr} {mf} {dep} {rand}"
```

This is the MLproject for ou GradientBoosting models. As we can see here it use the environment defined in conda.yaml.

Then it juste specify how to initialize our code. We define the parameters, and then we juste launch our code with the parameters. Here the parameters are optional since it will have a default values if we don't define them.

```
name: tutorial
channels:
- defaults
dependencies:
- numpy=1.14.3
- pandas=0.22.0
- scikit-learn=0.19.1
- pip:
    - mlflow
    - xgboost
```

And that's what the conda.yaml looks like. (I purposely used the conda.yaml for the Xgboost model instead of Gradient Boosting because it has an extra pip installation requirement)

We separated our model into different directories. So in order to launch a model we just have to go to a specific directory and use the command `mlflow run .` to start the MLproject file which will execute the environment building and launch the code.

If you want to put arguments into the model you just need to use the `-P` options before each argument.

And that's the final result:

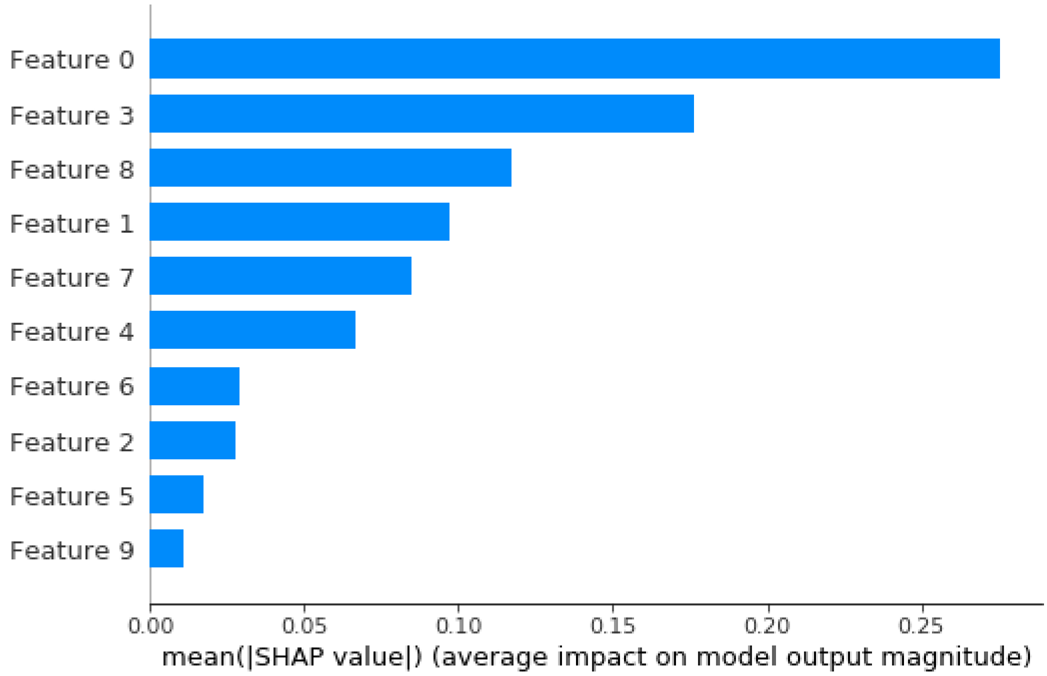
```
Gradient Boosting (gamma=0.500000, learning_rate=0.500000,max_depth=5.000000,max_d
elta_step=5.000000 lambda=1.000000, alpha=0.000000)):
Mean Absolute Error: 0.2812504798768143
Mean Squared Error: 0.07910183243093832
Root Mean Squared Error: -0.08794096369929205
```

PART 2: XAI with SHAP method

To install SHAP in our Environnement we can just use: pip install SHAP.

We'll use SHAP to explain and plot information about our Xgboost model.

To build a TreeExplainer and compute Shaplay Values we use shap.TreeExplainer(Xgboost). We plot the values for all the features and we obtain the following plot :



You can find the plot on the source code .

We can also print all the shap values with shap_values :

```
array([[ 0.07662712, -0.00513793, -0.02676886, ..., -0.02316365,
         0.07162519, -0.00718339],
       [-0.34747717, -0.50548923, -0.02025679, ...,  0.02911871,
         0.088447  , -0.00766503],
       [-0.09117541, -0.00827016, -0.00961998, ..., -0.00448823,
         0.1260188 , -0.00358319],
       ...,
       [ 0.08877876,  0.20714098,  0.0623149 , ..., -0.07455409,
         0.09227457, -0.0574662 ],
       [ 0.14378883,  0.02134109, -0.01961259, ..., -0.02363995,
         0.09885503, -0.00200997],
       [-0.31956047, -0.51236564, -0.02391518, ...,  0.01175132,
         0.11698311,  0.04776762]], dtype=float32)
```

Finally we can create a summary plot that allows to see all the points of the dataset and interprate lots of thing, here it is :