

Credit scoring

Build a model

Deploy the model through an API

Dashboard

Julien Le Boucher

09-27-2023

Context and missions



"Prêt à Depenser" is a financial company which goal is to broaden the financial inclusion for the unbanked population by providing a safe borrowing experience.

In that sense, financial, statutory and behavioral data were acquired and I was asked to:

- build a classifier which accept or reject a credit application.
- make the model inferences available through an API.
- provide a dashboard which:
 - displays the model decision;
 - assists customer relation managers in transparency by explaining the main factors that influenced the model decision;
 - facilitates navigation through customers' data.

Constraints

→ Embrace an MLops approach :

- reproducibility, tracking and registration of models with MLflow
- code versioning with git & GitHub.
- implement a CI/CD pipeline with tests(Unittest) and GitHub actions.

→ Deploy both the API and the dashboard.

→ Test evidently as a tool to detect data drift when the model is in production.

→ Document the steps explored to build the classifier.

Plan

1. Modeling approach and results.
2. CI/CD pipeline.
3. Data drift.
4. Dashboard demonstration.

Modeling approach and results

Predictors: Features engineering + data preparation

- Feature engineering using this [GitHub repository](#) (adding lots of ratios and aggregations).

→ 735 features for ~ 356 000 individuals.

problem: lots of nulls and +/- inf values.

- Need to handle those values → function `load_split_clip_scale_and_impute_data()` which:
 - filter very sparse individuals (> 400 null values).
 - sample the population (optional)
 - split into train and test sets.
 - impute categorical data to the nearest neighbor
 - one-hot encode categorical data (optional)
 - filter inf values by clipping based on min and max values found in the train set (optional)
 - scale with some options: "minmax", "standard" or None
 - impute with some options: "knn", "zero", "median"

Note : Ideally, this pre-processing should be part of the pipeline with the model, but some operations are time consuming. We can save a lot of time pre-computing rather than doing so for each iteration in the cross-validation scheme (although that implies data leakage).

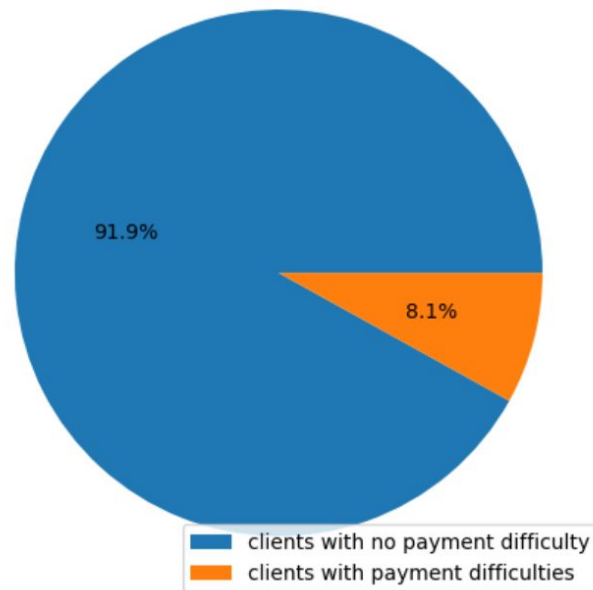
Target distribution

→ Imbalance. Ratio ~ 11.5.

Problem: classifiers tend to predict the dominant class.

3 techniques used to address that:

- **Weight classes errors**: increase the impact of errors made on the minority class when evaluating and decrease the ones made on the majority class (it usually is a classifier hyperparameter).
- **Resample the training set**: Oversample or generate plausible sample for the minority class (SMOTE) and undersample the majority class to get closer to class balance.
- **Threshold moving**: find the best value to decide the cutoff class from the probability prediction which defaults to 0.5



Model evaluation and Business considerations

Definitions

False Negative: someone wrongly seen as capable of repaying.

False Positive: someone wrongly seen as not capable of repaying.

Assumption :

"loss of income due to a false negative is 5 times bigger than the loss induced by a false positive."

Derived metric

$$LoI = \frac{5FN + FP}{N_{indiv}}$$

FN: number of false negative. FP: number of false positive. N_{indiv}: number of individual.

We want to minimize this metric, which can be seen as the mean loss due to the model prediction errors per individual.

Best model search

- Tested model type:
 - logistic regression (Lasso and Ridge type)
 - SVC (rbf and polynomial)
 - random forest classifier
 - xgboost and lightgbm classifier.
- Uniform pre-processing on a sample of individuals (2000) :
 - one-hot encoding: ON
 - clipping: ON
 - scaling: minmax between 0 and 1
 - imputation: knn (k=2)
- Best hyperparameters search with hyperopt (50 iterations per model).
 - minimize the **Lol** function.
 - evaluate a model predicting probabilities using a cross-validation scheme with 5 folds while making use of the threshold moving technique.
- metrics and time tracking with MLflow.
Lol, AUC, f-2.

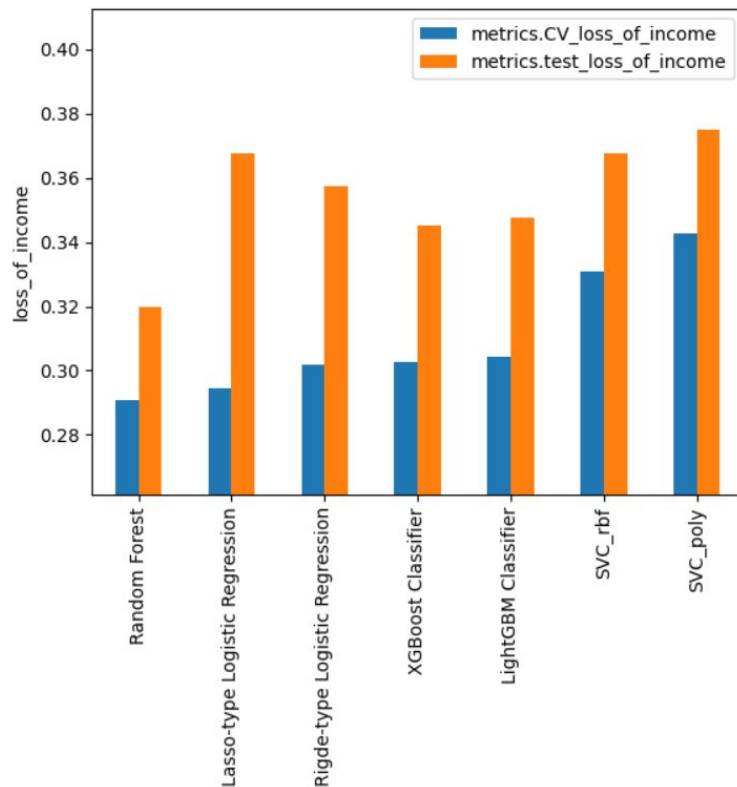
Results (loss of income *LoI*)

→ exploration via MLflow UI

	metrics.CV_loss_of_income	metrics.test_loss_of_income
parent_run_name		
Lasso-type Logistic Regression	0.294375	0.3675
LightGBM Classifier	0.304375	0.3475
Random Forest	0.290625	0.3200
Rigde-type Logistic Regression	0.301875	0.3575
SVC_poly	0.342500	0.3750
SVC_rbf	0.330625	0.3675
XGBoost Classifier	0.302500	0.3450

Best score per model type (smaller is better)

- Random forest classifier is the winner.
- Linear classifiers score decently but do not generalize well.
- SVC do not fit the problem.
- XGBoost and lightGBM offer a decent generalization, but are not as good as the random forest.
- From the appendix, you can see results from the AUC and f-2 perspective. The Random Forest and lightGBM are the top-2 models regarding the 3 metrics.



Selecting the best pre-processing for the Random Forest classifier

- SMOTE and one-hot encoder effect. 1500 individuals / 100 hyperopt iterations per pipeline:

	metrics.CV_AUC	metrics.CV_loss_of_income	metrics.CV_f2
parent_run_name			
SMOTE pipeline with Random Forest	0.723628	0.300000	0.466667
SMOTE pipeline, ohe categorical, with Random Forest	0.727545	0.299167	0.482456
ohe categorical with Random Forest	0.736479	0.283333	0.494327

- Varying imputation and scaling methods. 1500 individuals / 100 hyperopt iterations per classifier :

scaling_method	imputation_method	metrics.CV_AUC	metrics.CV_loss_of_income	metrics.CV_f2
minmax	knn	0.751201	0.278333	0.523349
	median	0.747347	0.264167	0.525862
	zero	0.732510	0.258333	0.508475
standard	knn	0.752525	0.281667	0.513866
	median	0.735382	0.262500	0.510949
	zero	0.730186	0.263333	0.502793

→ **SMOTE worsen the results.**

→ For the first time, the custom metric behaves differently.

AUC and f2 suggest knn is better and the loss_of_income encourage to use minmax with zero imputation.¹¹

Training random forest classifier on the whole dataset with the best pre-processing configuration

- minmax scaling [0,1]
- zero imputation
- clipping values
- no SMOTE
- categorical features one-hot encoded

Best random forest scores

metrics.CV_AUC	metrics.CV_loss_of_income	metrics.CV_f2
0.698053	0.332687	0.431447

lower than expected



Hyperparameters of the best models (top-5)

params.n_estimators	params.max_depth	params.min_samples_leaf	params.min_samples_split	params.class_weight
204	18	4	5	None
200	18	5	6	None
232	16	5	4	None
229	17	5	4	None
237	17	5	5	None

Interesting to note that the class_weight hyperparameter is not used either.
Threshold moving is really taking care of the class imbalance by itself.

Comparing with lightgbm (whole dataset, no pre-processing)

Best random forest scores

metrics.CV_AUC	metrics.CV_loss_of_income	metrics.CV_f2
0.698053	0.332687	0.431447

Best lightgbm scores

metrics.CV_AUC	metrics.CV_loss_of_income	metrics.CV_f2
0.721945	0.314635	0.460757

On the whole dataset, **lightgbm is more convincing**.
This is the model chosen to pursue the project.

lightgbm best hyper-parameters

	value
colsample_bytree	0.7886281999030231
reg_lambda	0.5957468159776326
subsample	0.8679759773737233
is_unbalance	False
subsample_for_bin	140000
reg_alpha	0.8631398118506094
learning_rate	0.01752285342870759
num_leaves	54
boosting_type	goss
min_child_samples	230
max_depth	14
min_split_gain	0.0402532222541853

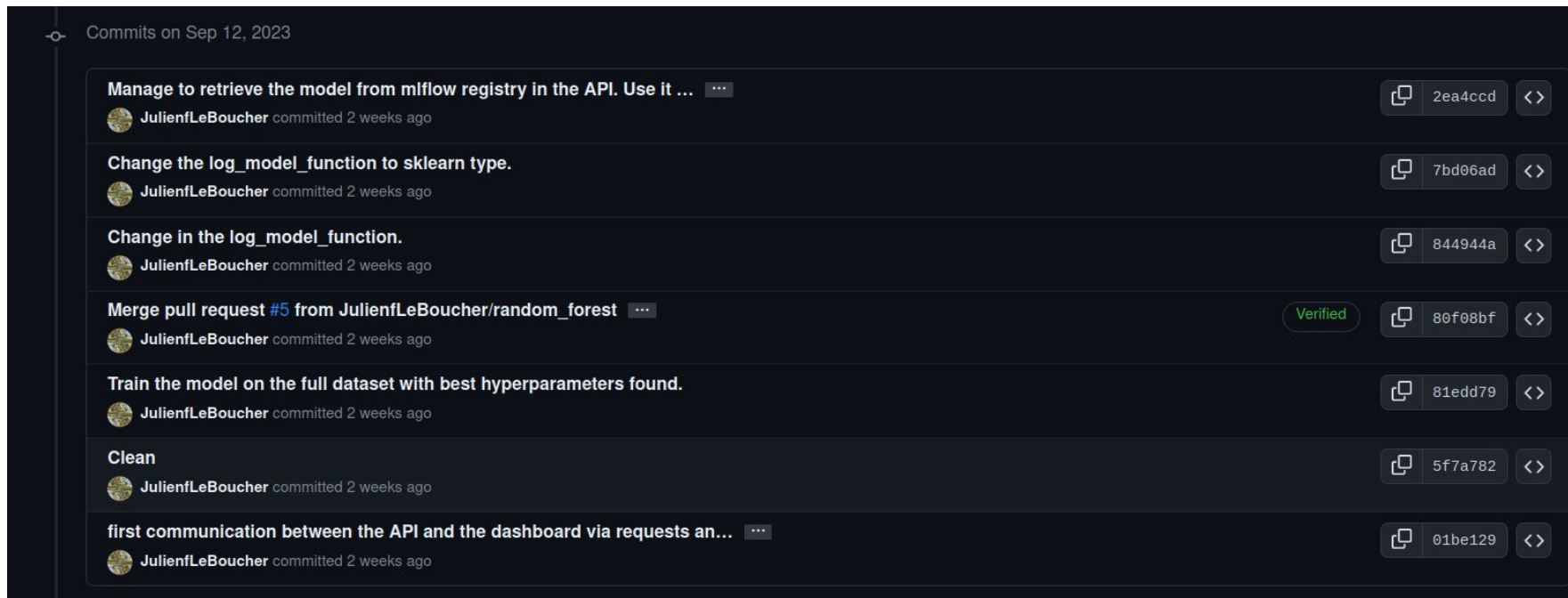
neither using class weights.

CI/CD pipeline

Code versioning.

My GitHub repository

Some commits:



A screenshot of a GitHub repository's commit history for the date September 12, 2023. The interface is dark-themed. It shows a list of seven commits by user JulienfLeBoucher, all committed 2 weeks ago. Each commit entry includes a title, a commit hash, and a 'Verified' badge. The commit titles are: 'Manage to retrieve the model from mlflow registry in the API. Use it ...', 'Change the log_model_function to sklearn type.', 'Change in the log_model_function.', 'Merge pull request #5 from JulienfLeBoucher/random_forest', 'Train the model on the full dataset with best hyperparameters found.', 'Clean', and 'first communication between the API and the dashboard via requests an...'. The commit hashes are 2ea4ccd, 7bd06ad, 844944a, 80f08bf, 81edd79, 5f7a782, and 01be129 respectively. Each entry also has a copy icon and a compare icon.

Commits on Sep 12, 2023

- Manage to retrieve the model from mlflow registry in the API. Use it ...** 2ea4ccd
- Change the log_model_function to sklearn type.** 7bd06ad
- Change in the log_model_function.** 844944a
- Merge pull request #5 from JulienfLeBoucher/random_forest** 80f08bf
- Train the model on the full dataset with best hyperparameters found.** 81edd79
- Clean** 5f7a782
- first communication between the API and the dashboard via requests an...** 01be129

Tests made with unittest

```
1  import my_app
2  import api_utils
3  import unittest
4  import numpy as np
5
6  def inv_logit(p):
7      return np.exp(p) / (1 + np.exp(p))
8
9  class MyTestCase(unittest.TestCase):
10
11      def test_customer_and_shap_values_correspondence(self):
12          """
13          Test if the get_index() function enables to extract the
14          shap values related to the right customer.
15
16          This is true when the sum of the shap values + base value
17          are related to the probability predicted by the model.
18
19          Here, equality is tested with a relative tolerance.
20
21          For robustness, it is tested on several customers.
22          """
23          rtol = 1e-3
24          # Pick three customers.
25          customer_ids = [400991, 191921, 353368]
26          # Prepare vector to store and compare.
27          model_proba = np.zeros(len(customer_ids))
```

```
28      shap_proba = np.zeros_like(model_proba)
29      for n, customer_id in enumerate(customer_ids):
30          # Get the customer probability predicted by the model
31          model_proba[n] = api_utils.get_customer_proba(
32              my_app.model, my_app.features, customer_id
33          )
34          # Get the index (row number of the shap values to retrieve)
35          idx = api_utils.get_index(customer_id, my_app.features)
36          # Get the customer shap explanation according that index
37          customer_exp = my_app.exp[idx]
38          # Compute the sum of the shap values on the log-odds scale
39          shap_sum = customer_exp.base_values + np.sum(customer_exp.values)
40          # Convert to probability
41          shap_proba[n] = inv_logit(shap_sum)
42          # Check both vectors are almost equal.
43          self.assertTrue(
44              np.allclose(
45                  model_proba,
46                  shap_proba,
47                  rtol=rtol,
48              )
49          )
```


Using a workflow in Github Actions to build, test and deploy the API to the EC2 instance

✓ Build Test and Deploy API to EC2 instance #24

Re-run all jobs

...

Summary

Jobs

✓ build_test_and_deploy

Run details

Usage

Workflow file

build_test_and_deploy

succeeded 1 minute ago in 57s

Search logs

> ✓ Set up job

2s

> ✓ Run actions/checkout@v3

2s

> ✓ Set up Python 3.11 and cache it

3s

> ✓ Install dependencies

39s

> ✓ Test with unittest

5s

1 ▶ Run python -m unittest discover api

11 ..

12 -----

13 Ran 2 tests in 0.450s

14

15 OK

> ✓ Deploy to EC2 through ssh

1s

> ✓ Post Set up Python 3.11 and cache it

0s

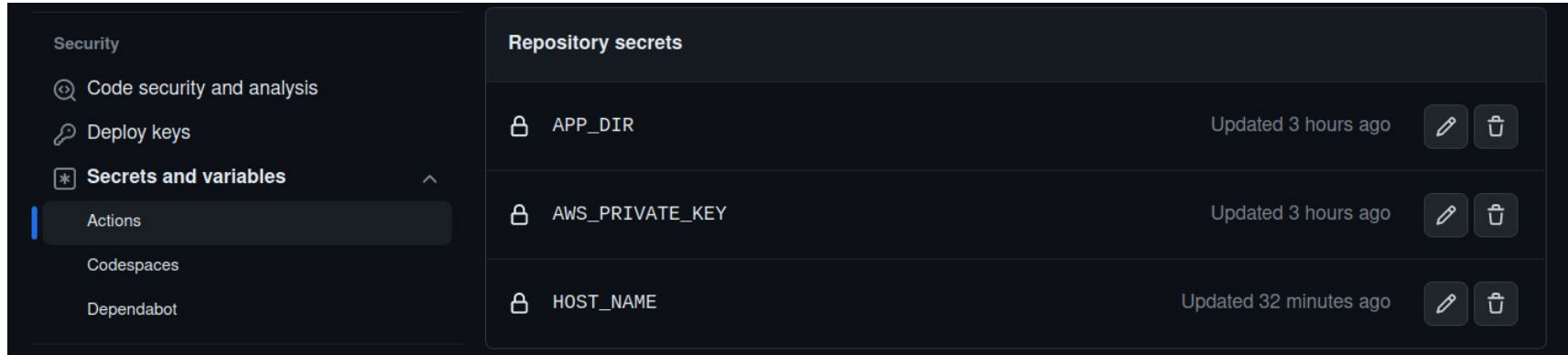
> ✓ Post Run actions/checkout@v3

0s

> ✓ Complete job

0s

Using Github secrets to secure the connection to the EC2 instance



Data drit

Data drift with Evidently

- Compare and measure main features distribution changes between 2 datasets.
- Provide a detection algorithm to automatically test data drift (could be integrated in the CI/CD pipeline).

Dataset Drift

Dataset Drift is NOT detected. Dataset drift detection threshold is 0.5

27	0	0.0
Columns	Drifted Columns	Share of Drifted Columns

Results comparing the train set and the test set for the main features obtained from the lightgbm feature_importances_ attribute and from shap values.

Show interactive html summary

Dashboard demonstration

Show deployed dashboard.

Thanks for your
attention.

Appendix

API and dashboard initial Deployments on AWS

For each application:

EC2 instance

(AMI linux amazon, t2.micro, with inbound rules to open ad hoc ssh and http ports.)

1. Connect via ssh.
2. Install git and some development tools.
3. Clone github pyenv project to create a virtualenv.
4. Clone the application code from my github repository
5. pip install requirements in the virtualenv.
6. Run the application.

API – main endpoints

- `/`: Welcome!
- `/prediction/`: list of the available customer ids for prediction
- `/prediction/<int:customer_id>`: information about the customer (probability, id, class)
- `/model_info`: information about the model used underneath.
- `/global_shap/`: base64 encoded image of shap summary plot
- `/local_shap/<int:customer_id>`: base64 encoded image of shap waterfall plot.

f-beta (f-2)

$$F_{\beta} = \frac{(1 + \beta) \times TP}{(1 + \beta) \times TP + \beta \times FN + FP}$$