

MACHINE LEARNING ENGINEER CHALLENGE

By Daniel Baena

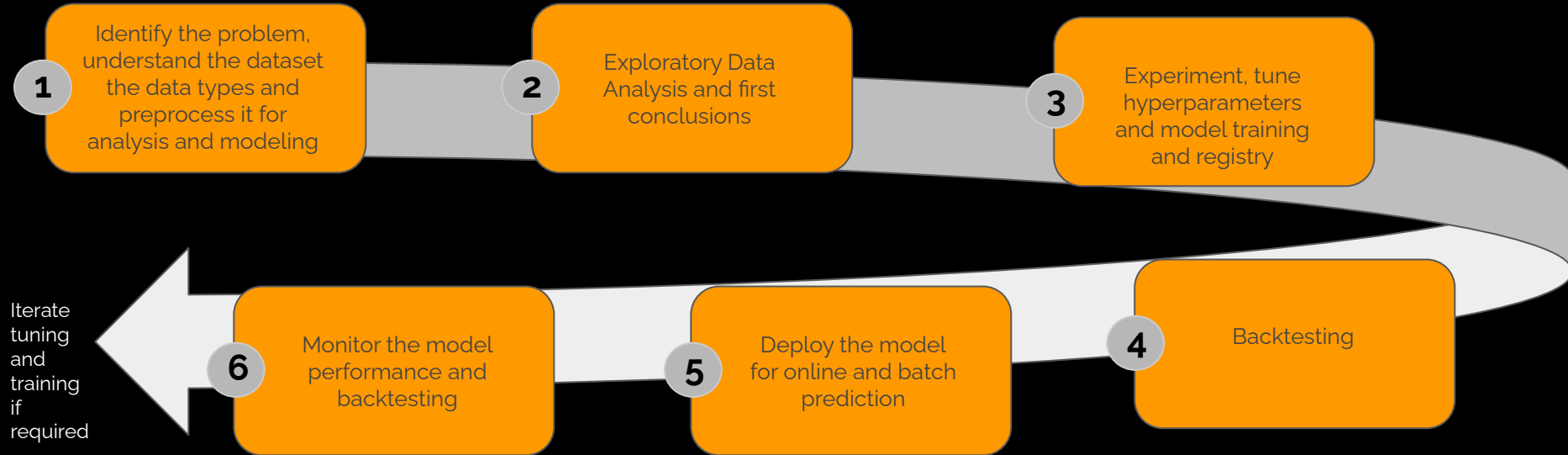


Content

1. Model structure
2. Understanding & Preprocessing
3. EDA
4. Training & model registry
5. Backtesting
6. Deployment
7. Monitoring

Model Structure - MLOps Cycle

The model is structured into 6 main processes: **Understand the problem**, preprocessing, EDA, Training, **Model registry and Backtesting**, **Model serving or deployment**, **Monitoring**.



Understanding & preprocessing

Files: `./train/utlils/preprocess.py` `./train/preprocessing`

This process consist into clean and process the raw initial dataset. One challenge of the dataset is the structure and the type of the variables, specifically:

Fecha comes in string format with type `"%d/%m/%Y"`.

Monto comes separated by comma and in string format.

Dcto comes separated by comma and in string format.

Cashback comes separated by comma and in string format.

Dispositivo is a string column with dictionary values separated by semicolons, which is very unusual.

Tipo_tc comes with characters in portuguese, which are changed to unicode.

The file script `./train/utlils/preprocess.py` clean and adjust variables for use in training.

After cleaning the variables, they are classified by **numerical**, **categorical** and **boolean**, applying an **imputer** with **mean** for **numerical** and filling with **missing** the **categorical** ones that are empty.

EDA

Files: `./train/eda.py`

It is used the library "SweetViz" for the EDA, which contains an option to include a correlation analysis in the button "Associations" at the top of the [website](#).

As conclusions we can find mainly:

TIPO_TC: indicates if the transaction is physical or virtual. There are more physical fraudulent transactions than virtual (**579** vs **231**), but the proportion is the same, **3%** of the total for both.

IS_PRIME: no prime users are more, and although the proportion of fraudsters are equal for both populations (3%), no prime users tend to have more fraudulent transactions.

DCTO: it is notable that transactions with a discount higher than 0% discount tend to present a bigger amount of fraudulent transactions. 69 % of the transactions do not have discounts a have zero fraud cases.

CASHBACK: fraudulent transactions tend to keep sigly constant with low and higher amounts of cashbacks.

For correlations, we can find particularly that **CASHBACK** is highly correlated with **MONTO** (0.81 Pearson correlation coefficient). The other variables do not show a significant positive nor negative correlation.

Training & model registry

Files: `./train/train.py` `./train/utils/models.py` `./train/utils/figures.py`

Model training

An XGBoost classifier is used for the training process that covers the entire pipeline from pre-processing to model registry. This process is all executed from the `./train/train.py` file and is structured this way for one reason.

In a production and continuous monitoring environment, the ideal is to facilitate retraining so that the container that contains the model pushes the new retrained model and the new metrics of the experiment.

Model registry

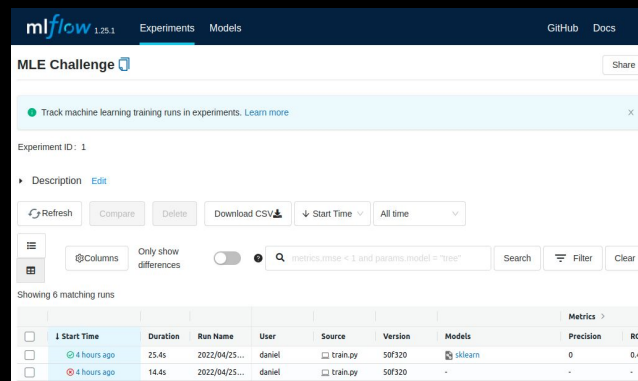
This time I am using **MLFlow** as model registry and experiment tracking. To start the MLflow server, activate the virtual environment and run **mlflow ui** in the terminal.

Experiment replicability

Follow the instructions in the repository to replicate the experiment.

Next steps

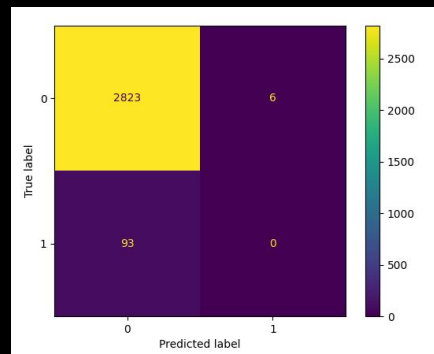
To reduce the variance of the test set, we could increase the train dataset, improve the imbalanced dataset by implementing SMOTE, implement regularization techniques and include other metrics to measure the model performance (ks, log loss, accuracy).



The screenshot shows the MLflow web interface with the 'Experiments' tab selected. It displays 'Experiment ID: 1' with a description and various action buttons like 'Refresh', 'Compare', 'Delete', 'Download CSV', and 'Start Time'. Below this, there's a table showing 6 matching runs. The first two runs are visible:

	Start Time	Duration	Run Name	User	Source	Version	Models	Precision	ROC
<input type="checkbox"/>	4 hours ago	25.4s	2022/04/25...	daniel	train.py	50f320	sklearn	0	0.47
<input type="checkbox"/>	4 hours ago	14.4s	2022/04/25...	daniel	train.py	50f320	-	-	-

MLflow interface



Confusion Matrix
optimizing F2 score

'Threshold': 0.952
'Precision': 0.0,
'Recall': 0.0,
'ROC AUC': 0.471

The file
`./metrics/test_set_m
etrics.json` contains
other metrics
suggesting other
thresholds.

Backtesting

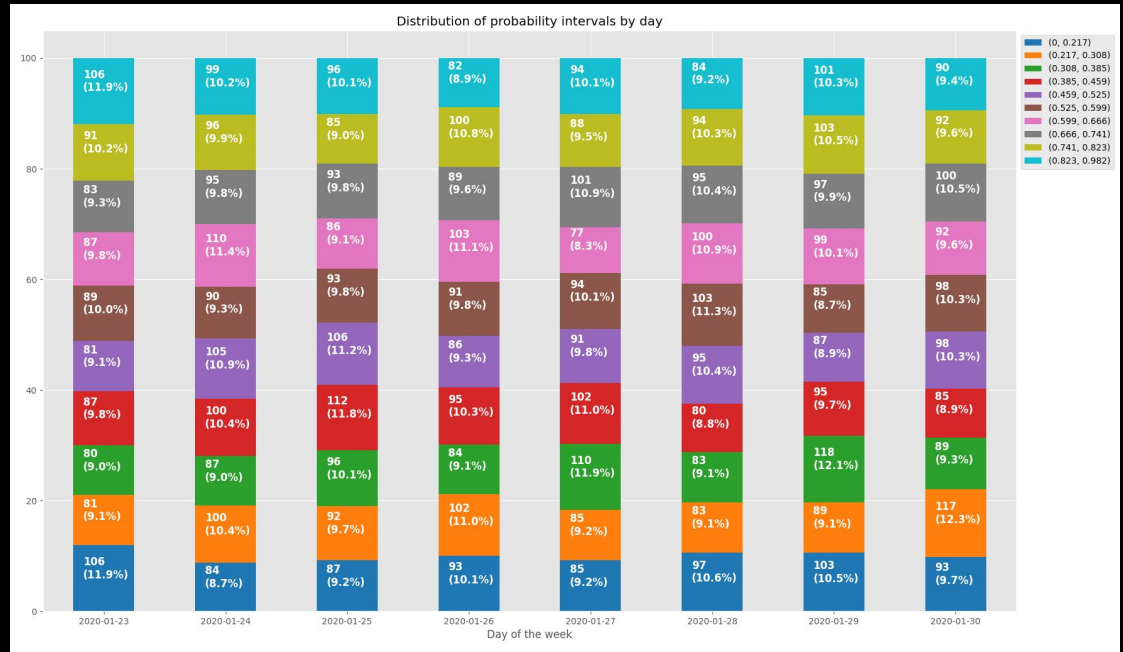
Files: `./test/batch.py`

Before deploying a model into production, it's important to do a backtesting and understand how the model performs in recent data and live data.

In this case, before splitting the data into train and test sets, I separate and store the the transactions of the last week in a file called `backtesting.csv`

Result:

Running the backtesting, it can be noticed that our model through the week is stable and the percentiles of the scores keep constant across new population .



Suggestions: Deploy the model using a threshold of **0.90**. According to the test confusion matrix in the metrics/img, we would have a low precision and low recall (0.03), but we could iterate through it and block fewer transactions with a high threshold.

Model deployment

Real time:

The model uses a Dockerfile that can be deployed in any cloud.

Currently the model is deployed in a cluster of **AWS App Runner**, which by default create a Docker container with **Python 3.8**, copy all the files that are not in the dockerignore, install the requirements, test the health endpoint ("/") and deploy the container.

In a real world application environment, I suggest the following framework:

- Include **logging** to your application and check them using services as **LogDNA**.

- Test your model in a **dev and staging environment** before production deployment.

- Deploy the container in a **Kubernetes** environment.

- Protect and connect the model with other microservices using an authentication service such as **Amazon Cognito**.

- Monitor the container resources (CPU and RAM) using services as **Splunk**.

- Evaluate the code quality and coverage using **Sonar Qube**.

- Setup alerts of the microservice in services as **Opsgenie**.

Test the model deployed in the following website hosted in AWS: [link](#)

Batch:

The current model also run predictions in batch using the file `./tests/batch.py`. For deployment, I suggest to use an orchestrator tool such as **Apache Airflow** with a K8s operator running the model container.



Model monitoring & retraining

Retraining

In this framework, the models both in batch and real time will need to have two pipelines (as it is in the current model code) for scoring and training.

The continuous training pipeline will run **MLflow** in parallel inside a **docker container**, updating the **new model weights (binary files)** into an **S3 bucket** and setting alerts with **slack or email webhooks** to send the model **pdf report**.

With the report, we could take the decision of accept or reject the pull request of the new model version

Monitoring

Model monitoring can be live or in batch. Usually, it requires a considerable amount of data to identify changes in distributions. In this sense, I suggest a daily, weekly and monthly monitoring batch system that computes metrics such as: Kullback-Leibler divergence, Kolmogorov-Smirnov test, CSI (Characteristic Stability Index) and PSI (Population Stability Index). The current model code already contains all the datasets required for model and variables monitoring.

