M2T3. Final Report

To solve this part of the problem, implement an ML algorithm that predicts the defaulting probability based on the financial history. I tried some different approaches, but it wasn't enough. As I've been told, it was a difficult problem. In fact, the problem is still unsolved, or done in a very precarious way, with a Gradient Boost Regression poorly performing with an accuracy around 20-25%.

The general pipeline I tried was:
- **EDA** to understand the data, its distribution, and general characteristics.
- **Data Cleaning and Pre Processing**, where I got a great surprise discovering that not all the columns were relevant. In fact, from almost 25 features I ended up working with just 13.
- **ML,** I tried to use the dataset to train an algorithm to solve the problem.

As I have already sent reports of the first two steps, now I will focus on the last one.

For the last step I used another framework:

- **Phase zero:** just splinting the data into different groups of Train, Validation, and Test.
- **Phase one:** evaluate a bunch number of possible candidates. The models were fitted just out of the box, no hyperparameters were tuned. It was a kind of fast and dirty implementation. The models I tried:
    - 'Random Forest Regression', 'Linear Regression', Support vector Regression', 'Logistic Regression', 'Gradient Boosting Regressor', 'K-Neighbors Classifier', 'Decision Tree Classifier'.

I tried a mix of classification and regression models because despite the problem asked explicitly asked for a probability, it had a classification problem underlying. I have to admit that I expected some of those models to perform decently, but the highest R2 (the overall model accuracy) reached was around 0.211. A lot of models performed worst than a simple horizontal line that always predicts the same value. Anyway, I hoped that some cross-validation and hyperparameter tuning could help.

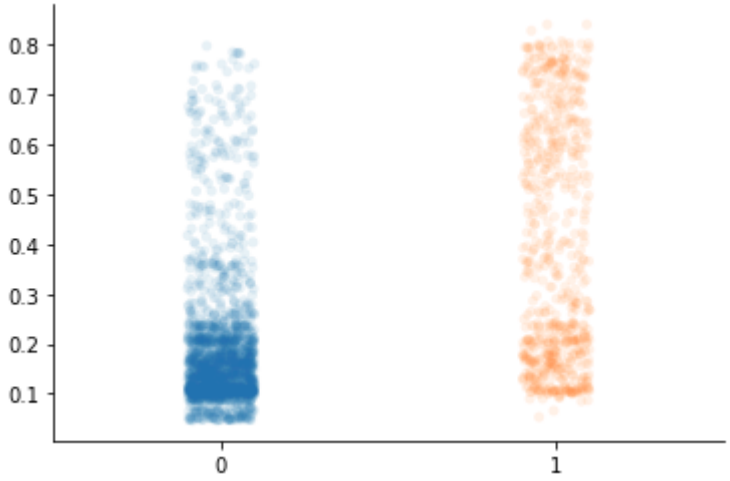
- **Phase two:** I tried all the models again using cross-validation, looking for a suitable candidate. Mostly I got the same results.
- **Phase Three:** I expected a lot from this phase, I thought it would give a 180-degree turn to the problem, but I'm getting used to the *No free lunches* of Data Science. From Phase One and Two, I selected the Gradient Boosting Regression as the best algorithm. I used the RandomiedSearchCV tool to define the best hyperparameters. Anyway, the results didn't improve too much.
- **Phase Four:** tried the tuned model, but with the test set. Similar poor results.

After the poor performance of the initial loop, I wanted to try some pre-processing of the data. I analyzed it again, plotting a heat-map of the correlations. I noticed 2 new index columns with a correlation equal to 1 (I suspect they were generated while saving the file to CSV). I removed them, also removed some columns (PAY_4,5,6) that had a moderate to low correlation with the outcome but very correlated with themselves. The result improved a bit, but still too low and imprecise.

Some recommendations and new ideas to try.

This is a problem of imbalanced data. Only 22% of the recordings included defaulting clients, naturally, the vast majority of the transactions will be legitimate, and only a very small proportion will be fraudulent. Similarly, if we are testing individuals for cancer, or spam email detectors, hardware faults detectors, Detection of oil spills from satellite images, etc, the positive rate will (hopefully) be only a small fraction of those tested. This kind of problem has some particularities that had to be taken into account.

- I divided data into train, validation, test using the tool provided by Pandas. But I didn't check the ratio of defaulting/non-defaulting of each set. If data were fully randomized and normally distributed, I think each group would receive a representative ratio of each label. I notice this aspect late, and my approach didn't take this into account.
- This plot somehow explains the situation. It's a scatter plot of the probability of defaulting, using the data of the predictions / actual values of the test set. The vast majority of dots are around the most typical situation, clients non-defaulting.



- The business question asked for the probability of default. But it had a classification problem underlying. Another possible approach could be: take this resulting probability and then define a threshold value that classifies each probability into a binary result.
  Those are some random outputs of the model. Using a threshold value, we can classify each predicted output higher than 0.25 as a possible defaulting client. Or define 3 or more groups as low, medium, and high probability.

Checking the row #: 6452 Actual value: 0 Predicted value: 0.09	Threshold e.g. 0.25	← non-default.	The algorithm predicted correctly.
Checking the row #: 27540 Actual value: 1 Predicted value: 0.11		← non-default.	The algorithm predicted incorrectly.
Checking the row #: 29128 Actual value: 0 Predicted value: 0.13		← non-default.	The algorithm predicted correctly.
Checking the row #: 3544 Actual value: 1 Predicted value: 0.30		← default.	The algorithm predicted correctly.