Methods for Big Data Analytics
AXA Data Challenge - Team Bokeyy

Antoine Isnardy	antoine.isnardy@ensae-paristech.fr
Benjamin Muller	benjamin.muller@ensae-paristech.fr
Alexandre Sevin	alexandre.sevin@ensae-paristech.fr

This short report presents the main guidelines that helped us score .39 on the public leaderboard.

1 Code

The whole code is available at https://github.com/antisrdy/axadatachallenge. If one wanted to play with models, please open a terminal, `cd axadatachallenge` and `cat README.md`.

2 Deep dive into the data

This crucial step was about discovering which data was at our disposal, and what to predict. We made a bunch of small analysis, mainly plotting data from different perspectives, in order to try to draw first lessons from them. Below are some important steps we took.

2.1 Assignments disparities

Very quickly, huge differences among the 26 assignments have been noticed, especially about the value to be predicted (CSPL RECEIVED CALLS later abbreviated with CRC). As displayed on Figures 1 and 2, for *Crises*, the value is almost always 0. On the contrary, for *Téléphonie*, values can be very large. Moreover, it seems that there are several cycles and breaks into the data as it is roughly drawn with bold blue lines for *Téléphonie*. Consequently, the prediction task for *Crises* looks in the end very different than what it does for *Téléphonie*.
Those two Figures are just two examples among the 26 assignments, but they are symptomatic of what is happening into the data. Furthermore, the number of predictions to be made was not even the same for each assignment, varying from 120 to 4028.

This led us to group data by assignment. Further, we even decided to focus only on CRC; meaning we started working with univariate time series. It looks like a big step since we dropped all variables but CRC. However, and for a first approach, it made sense, since most additional variables were administrative ones. Consequently, one may easily understand that these variables might affect less the number of calls than the seasonality.

Meanwhile, we chose to aggregate the COD by summing, as the loss function fosters overestimation (see Subsection 4.2 for details). This process is summarized in `./code/groupby_assignment.py`. It is a standalone script that can be launched from the root directory.

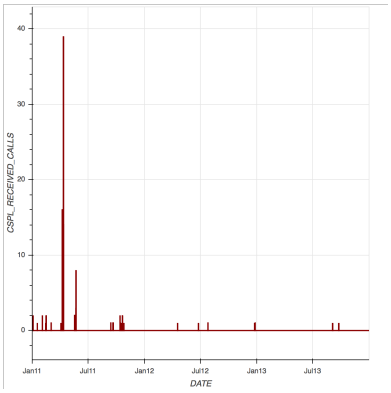

Figure 1: *Crises*

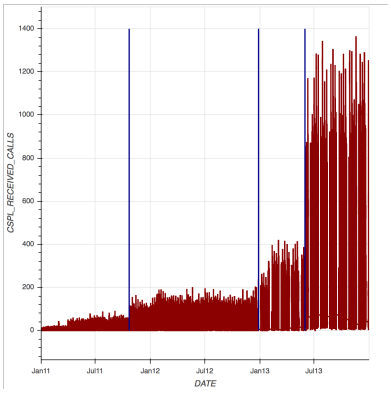

Figure 2: *Téléphonie*

2.2 Missing values

Next step was to handle missing values, as all the series were partially incomplete. It is important to note that incompleteness did not follow the same pattern among series, meaning that each one needed a special treatment. This supports the split made above.

An attempt - which proved its effectiveness - was then to fill missing values with the mean between the previous and the next date (if existing). It is summarized in `./code/treat_missing_values.py`. As above, it is a standalone script that can be launched from the root directory. At this point, we were then almost ready to try some modelizations.

Data discovery is in the end summed up in a few paragraphs. However, it took us a large amount of time to figure out all these simple remarks.

3 Data Science pipeline

This section aims at presenting the data science pipeline we set up to test different models. It is inspired by the code of A. Gramfort and B. Kegl we used during the DataCamp course. Even if it took quite a while to adapt it, it turns out it was in the end pretty useful to run fast cross-validations and focus on models.

3.1 Pipeline

This script, located in `./code/pipeline.py` handled everything related to cross-validation. Particularly, it correctly splitted the data into train - validation - test sets in a coherent way, mixing data from the beginning of the time serie with data from the end. This script helped us getting quick insights on models we were training, with outputs that looked like[1]:

```
length of training array: 27072 half hours = 80 weeks
length of test array: 6143 half hours = 18 weeks
length of common block: 13536 half_hours = 40 weeks
length of validation block: 13536 half_hours = 40 weeks
length of each cv block: 3384 half_hours = 10 weeks
train RMSE = 1.038; valid RMSE = 1.645; test RMSE = 1.979
```

[1]Note that a single cross-validation is displayed here. That's because at some point, thanks to a better knowledge of the data, we decided to cut others off.

```
mean train RMSE = 1.038 +- 0.0
mean valid RMSE = 1.645 +- 0.0
mean test RMSE = 1.979 +- 0.0
```

This pipeline stood for our main tool to **evaluate performances of our models**.

3.2 Feature extraction and regressor

These two categories of scripts, namely `./code/ts_feature_extractor_XX.py` and `./code/regressor_XX.py` were the real playground. Indeed, they stood for the real take-off runway, since it is in these 2 scripts that most of modelization experiments were conducted:

- Feature extractor: transforming raw time series to feed a regression method.
- Regression: learning methods and predictions.

3.3 Submission

This script, located in `./code/submission.py` aimed at building the submission file from predictions issued by every assignment.

4 Models and results

4.1 The failure of classical regression methods

Lasso regression was our first guess. As it outperformed in recent competitions involving time series prediction, we thought it might be a relevant idea. It was not. For a reason that would drive the rest of our challenge: **the special form of the loss**.
Indeed, we did not take into account the loss function at all while doing linear regression with Lasso for obvious reason. It was silly, and could not have ended well. And it did not, since our first submissions were around 1600, including 26 Lasso regressions trainings on basic features such as last dates available for prediction.

4.2 It is all about the evaluation loss

This led us to take a step back, and really look at the loss function. As a reminder, it is of the form:

$$\text{LinExp}(y, \hat{y}) = e^{\alpha(y-\hat{y})} - \alpha(y - \hat{y}) - 1$$

As mentioned in the guidelines, it gives a relatively higher penalty to underestimating the number of calls. Indeed, it is exponential in underestimation, and linear in overestimation. For instance, an overestimation of 100 calls leads to an error of 9, while an underestimation of 100 calls leads to an error of 22000(!). This asymetry clearly differs from traditional MSE, and fosters - as suggested - overestimating models.

4.3 Back to basics: minimizing loss function

Next idea was to come back to simpler ideas that looked more appropriated to this original loss. Why couldn't we **minimize the mean of LinExp** (+ possibly a penalization term). So that's what we tried. We mainly stuck to this approach until almost the end of the challenge. Nevertheless, we utilized it in several ways, depending on the assignment.

Feature engineering

For all the assignments, we kept only the past two weeks for the prediction. Moreover, we separated the data between weekdays for *CAT*, in order to fit one model per weekday.

Regressor

The regressor simply solved:

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \text{LinExp}(y_i, w^T x_i)$$

On a MacBook Pro with 16Go of RAM, it took an average of less than 30s to run the model for each assignment.

4.4 Holes in training data lead to underestimating

First results on cross-validation (see Section 3.1 for more details on cross-validation) and submissions proved we were doing things right, as our scores significantly decreased to go under 1 on the public leaderboard. Note by the way that we tried to penalize the problem, adding $\lambda \|w\|_1$ to the previous optimization problem, but it was not of a great help.
But because of holes in training data, models tended to underestimate true values in some cases. We decided then to **set up a voting system** between two predictors, hoping that it might fight against underestimation in these special cases, and improve our predictions in general.

4.5 Making things robust: XGBoost

XGBoost is often cited as a performing tool for data challenges. That's why we tried to utilize it. Note that similarly to the first method discussed above, the special loss of this problem needed to be taken into account while building trees. Consequently, it was important to specify the custom loss. Next, we averaged results of the first model introduced above, with those of the current boosting method for each assignment but *Téléphonie*.

Ensemble methods proved their efficiency as well, since they boosted our performances and reduced underestimation. Figure 3 displays for instance predictions for *Tech. Inter*. We got decent results which tended to stick to reality since red and blue curves were almost always the same. The main downside of this method is that it took a little more time to train: around 1 minute per assignment. The final and total process is summarized into the below section.

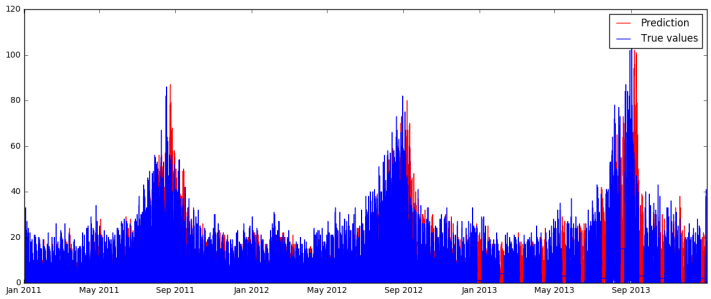

Figure 3: Predictions for *Tech. Inter*

4.6 Final process

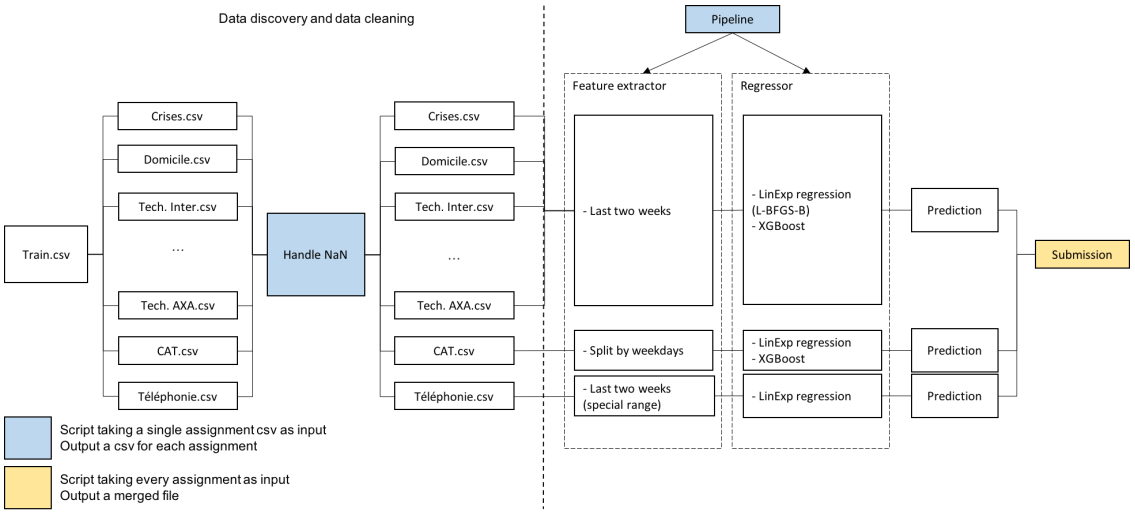

Figure 4: Final data science pipeline

5 Conclusion

Our final models helped us to score top 10 on the public leaderboard. It looks like a decent score. We are quite happy with the pipeline we set up, and models we built.

However, we realized at some point that the public leaderboard rated only a small portion of the global test set. Among other things, it means that we will probably be penalized for overestimating a little as displayed on Figures 5 and 6. It is still better than underestimating, but unfortunately, we could not come up with a better solution, especially for *Téléphonie*, which was in the end the most difficult assignment to predict.

Furthermore, it might have been a relevant idea maybe to take other variables into account, not necessarily the ones given in the dataset, but also external ones, like well-known future (or not) important events that migh affect the number of calls, weather conditions, etc.

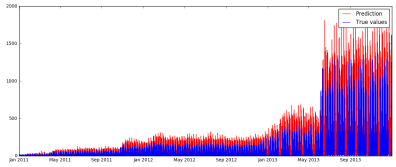

Figure 5: Predictions for *Téléphonie*

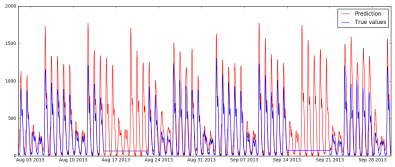

Figure 6: Zoom on *Téléphonie*'s preds