Methods for Big Data Analytics
Not1stYet - AXA Challenge report

Robin Monnier, Nathan Rouxel
robin.monnier@eleves.enpc.fr, nathan.rouxel@polytechnique.edu

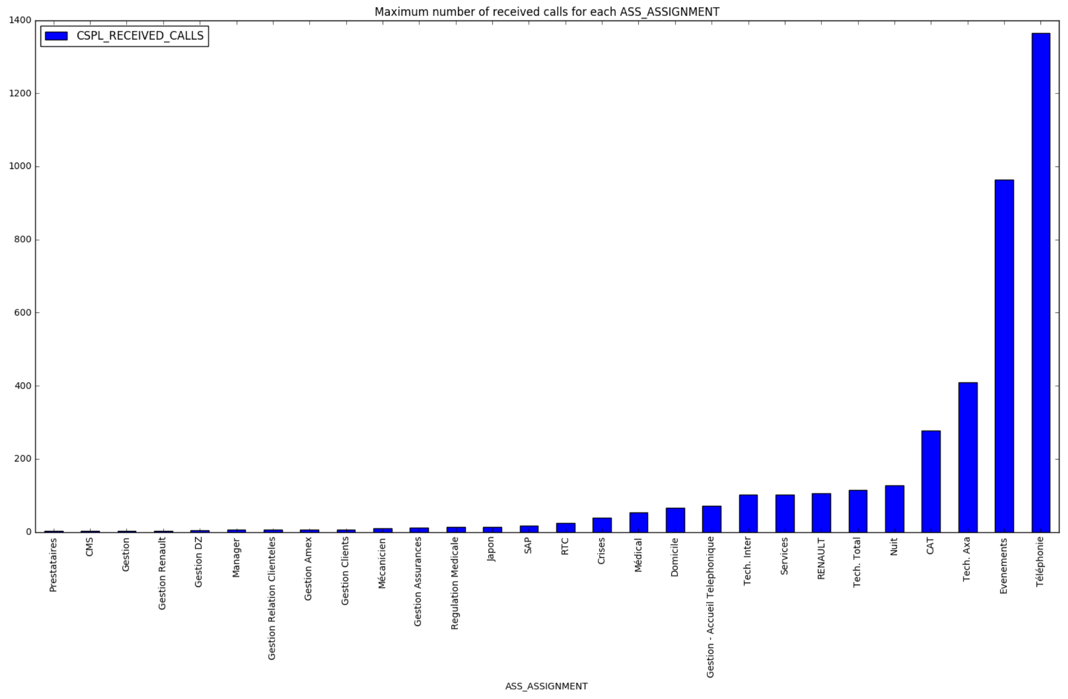
6 January 2016

Contents

1 Main conclusions of exploratory data analysis

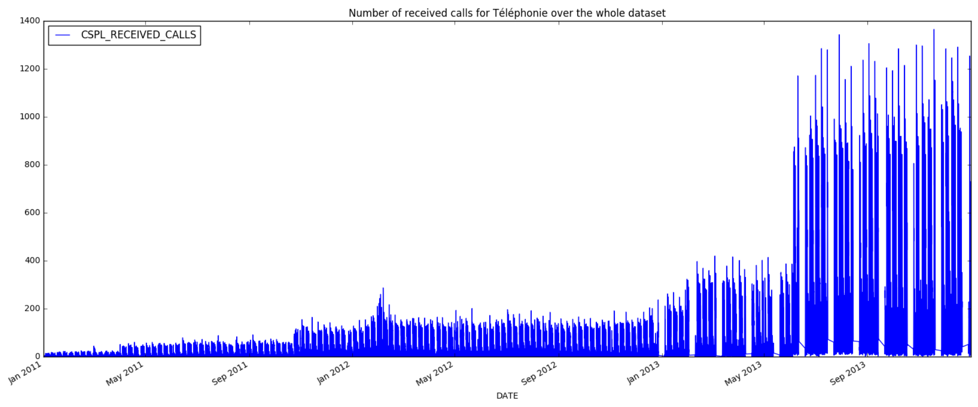
We performed a detailed exploratory data analysis in order to figure out some relevant information for feature engineering, feature selection and model design. In the following section, we will show the most useful plots we created, and discuss the insight we manage to retrieve from the data analysis.

Figure 1: Comparison of ASS_ASSIGNMENT's maximum number of received calls in a 30-minute interval



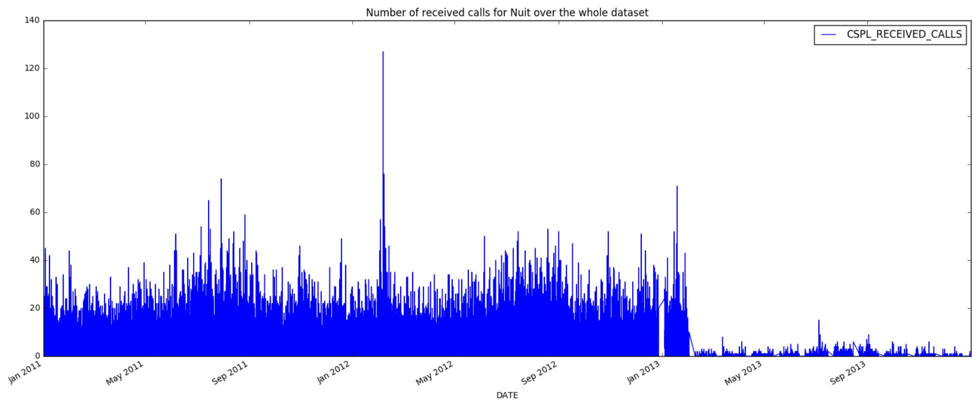
The figure 1 represents the maximum number of calls received by each ASS_ASSIGNMENT in a 30-minute time slot. As we can see, a few ASS_ASSIGNMENT are massively used by customers like Téléphonie and Tech. AXA (Evenements is in the train data but we did not have to predict anything for this assignment) whereas most of them are barely used (from Prestataire to RTC). This is important as we know that the evaluation function (LinEx) depends on the absolute error $y - \hat{y}$ made by the prediction. That's why a 10% error on Téléphonie will have a far bigger impact on the global error than a 10% error on Prestataire. This explain why we chose the maximum of calls received in a 30-minute time slot to make distinction between assignment.

Figure 2: Number of received calls for Téléphonie over the whole dataset



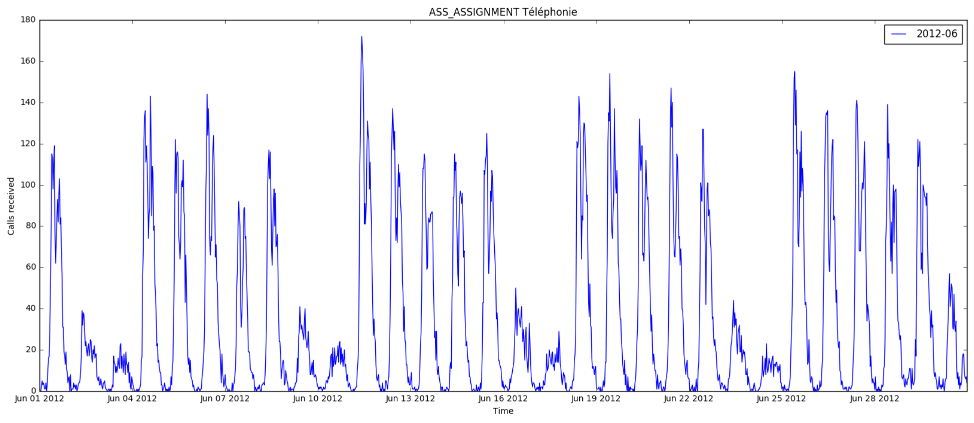
In the figure 2, we represented the number of received calls (for the Téléphonie ASS_ASSIGNMENT only) over the three years. The blank spaces in 2013 are in fact the weeks we want to predict for. In this figure, we notice at the first glance that the number of calls tends to suddenly grow from one year to another at some particular points in time. We made great use of this information in the heuristic method described in the third section.

Figure 3: Number of received calls for Nuit over the whole dataset



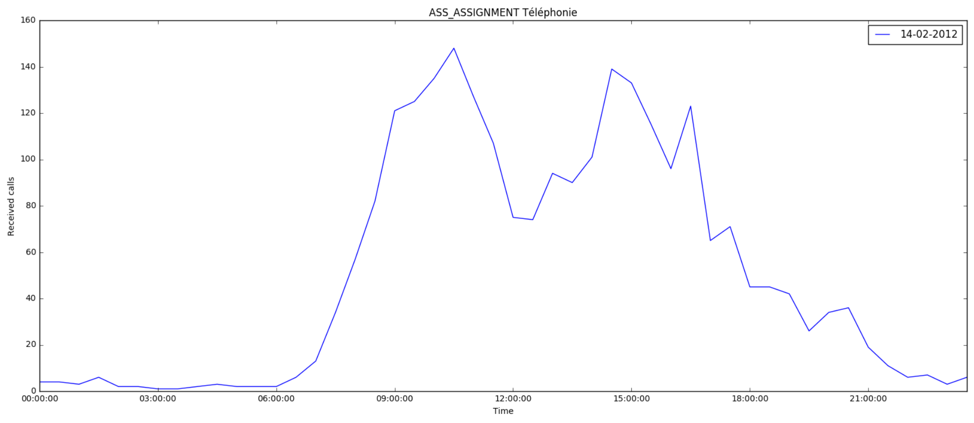
The figure 3 represents the same plot but for Nuit ASS_ASSIGNMENT. The global behaviour of the time series seems to have no obvious correlation with the one in figure 2. We represented time series for others ASS_ASSIGNMENT (focusing on the most busy one) and we find every time a assignment-specific behaviour which led us to separate the prediction and make prediction for each ASS_ASSIGNMENT, independently from the others. However, Téléphonie and Nuit seem to vary more than the others, which seems more stationary. But again, the weight of the errors on Nuit predictions in the final score is almost negligible against the one of Téléphonie.

Figure 4: Number of received calls for Téléphonie during one month



The figure 4 is a zoom in on a particular month. We notice an obvious pattern repeating itself each week. There are fewer calls during the week-end than during working days, that's why it seems relevant to us to add them as features.

Figure 5: Number of received calls for Téléphonie during one day



The figure 5 is a zoom in on a particular day. As we expected, there are almost no calls during night. So a boolean feature DAYTIME seems useful. Moreover, we notice that the number of calls is not constant over hours during daytime as there is a first decrease during lunch break and a second one from 5 PM.

2 Feature design and feature selection

From the exploratory data analysis, we concluded that there are local patterns we can clearly detect from the time series. Therefore we designed many time-related features as:

- year
- month
- day
- day_of_week
- hour
- minute
- week_end
- day_off
- not_working_day (week_end or day_off)
- daytime

We also considered using other features available in the training dataset. We could not easily use CSPL_...CALLS because they are related to the number of calls somehow (transfered calls, abandonned calls,...) and we don't have access to those features for the prediction dates. However we did study the hierarchical relations between ASS_ASSIGNMENT and other ASS_... features like ASS_DIRECTORSHIP or ASS_PARTNER. We found that each ASS_ASSIGNMENT is generally associated to only one value of ASS_... features. Because we decided to separate the predictions per ASS_ASSIGNMENT, we chose to ignore those features as they are useless for the predictions of the received calls for one ASS_ASSIGNMENT taken alone.

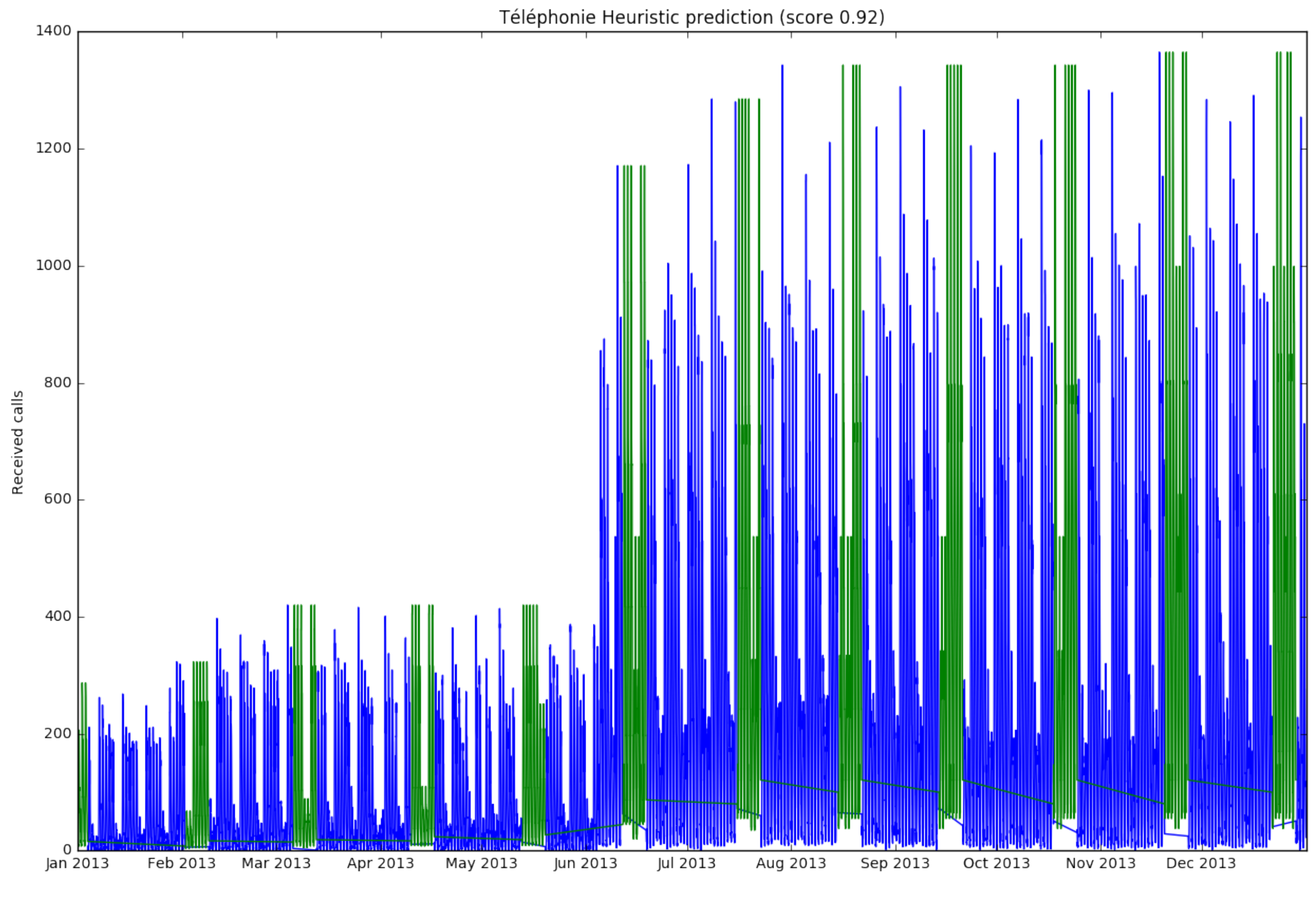
3 Final model : from quick heuristic to XGBoost algorithm

3.1 Maximum prediction heuristic

In parallel to exploratory data analysis and development for preprocessing and data loading, a quick first approach was to predict the maximum over a particular group by on temporal features. Observation was that patterns seem to repeat giving some particular time feature. Extremely fast (30 minutes of work in this way), that gives us a submission which scored 3.46. We grouped data by DAYTIME (Night/Daytime) and ASS_ASSIGNMENT and predicted the maximum CSPL_RECEIVED_CALLS in the past during a 30 min interval. For example, if we have to predict for 'Téléphonie' the number of CSPL_RECEIVED_CALLS at 12/10/2012 10:00:00.000, we predict the maximum CSPL_RECEIVED_CALLS received during daytime (10:00:00.000 is not during the night) in the data available before 12/10/2012 10:00:00.000.

Pursuing further that way, we finally got a score below 1.0 (0.92) exploiting that the data were very different from a year to another, and that they evolved every day in a similar way over hours. In addition, we also grouped by NOT_WORKING_DAY as this feature captured strong variance of the maximum of CSPL_RECEIVED_CALLS on an 30 minute time slot. The results of our best heuristic with this approach are in the figure 6, for ASS_ASSIGNMENT 'Téléphonie' (the most busy assignment) during 2013 (the year during which we have the more predictions to make). We grouped by YEAR, HOUR, NOT_WORKING_DAY and ASS_ASSIGNMENT.

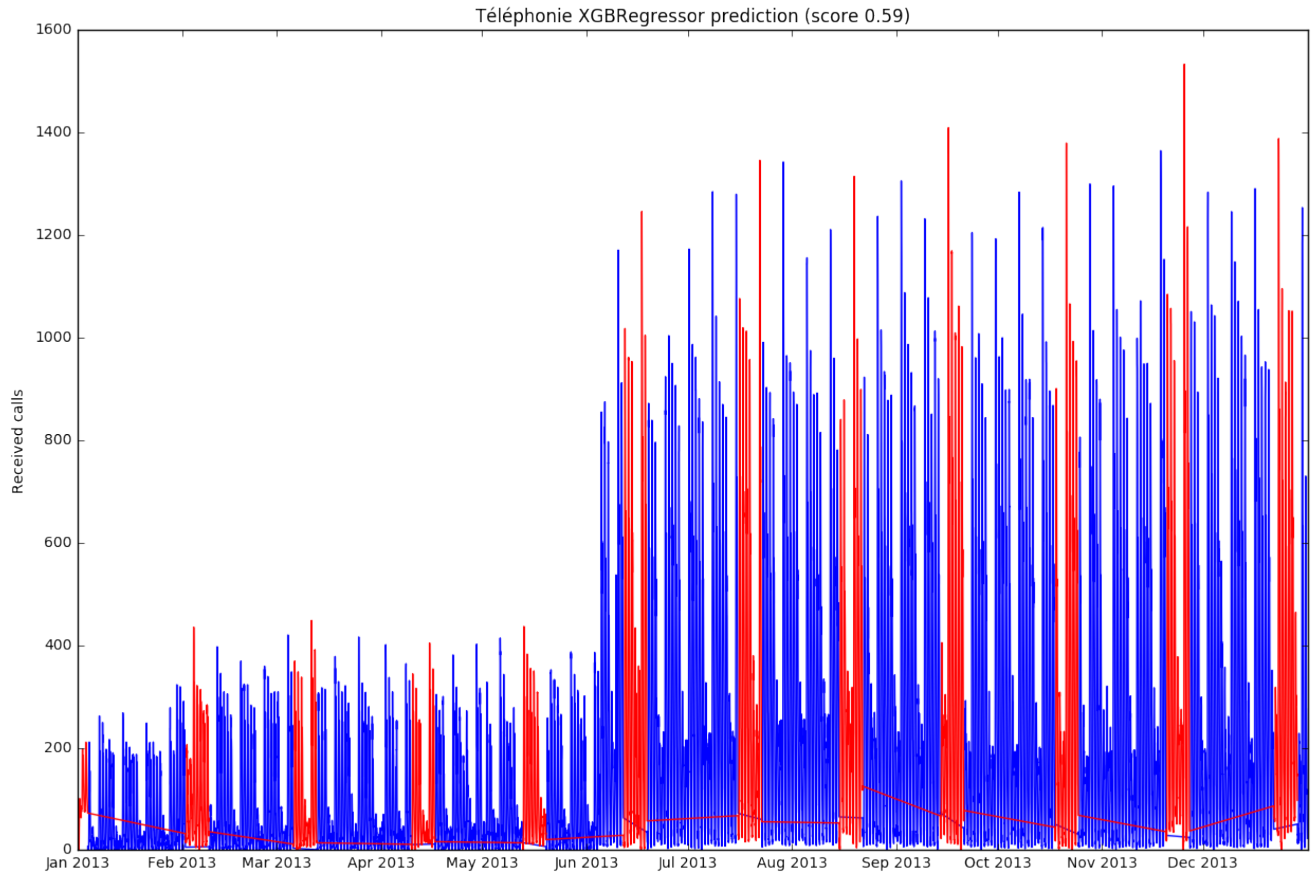
Figure 6: Heuristic prediction



3.2 XGBoost regressor

By the time we came up with the heuristic, the code was ready for machine learning approaches. As tree regressors and gradient boosting regressors were in the methods adviced in the challenge presentation, we used an XGBRegressor instance. Starting from the observation that most of the assignments had no particular impact on the score (as seen in the exploratory data analysis), we saved training time by using a model by group of assignments. We separated the 26 assignments in 5 groups, according to their maximum number of calls during daytime (or during night if this latter is bigger). We called them DUST (14 assignments with less than 23 maximum calls during daytime), DUST_2 (4 assignments with 39 to 72 maximum calls during day time), FIVE_HUNDRED (5 assignments with around 100 maximum calls during daytime), BIG_BROTHERS (2 assignments with 250 and 410 maximum calls during daytime) and TELEPHONIE (1380 maximum calls during daytime). Each group has it is own XGBRegressor instance with different parameters. The busier is the group of assignment, the more precise is the XGBRegressor and the longer it takes to train. We finally do not use dummy features, and do not include anterior predictions in the training set for future predictions (we quickly tried both, and they both led to a lower score), but these options are available in the code through a boolean. After the XGBoost predictions has been made, we up all the predictions by a coefficient (which is assignment specific), to compensate this way the symmetry between underestimation and overestimation in mean squared error which is not present in linex error. We also up all the negative predictions of XGBoost in order to make them all positive, and finally manually put to 0 all the predictions when we know the assignment is closed (some assignments are closed during holidays, or during night - we noticed it by computing the maximum number of received calls over the all train set grouping by DAYTIME, ASS_ASSIGNMENT or WORKING_DAY, ASS_ASSIGNMENT). We use a standard scaler at the beginning of the pipeline. Figure 7 shows the results of the XGBRegressor of Téléphonie in 2013.

Figure 7: XGBoost regressor prediction



This final model scores 0.58 on the leaderboard, and the whole data loading, training and predicting process take about 7 minutes to run for the 11 weeks to predict and the 26 assignments, on 4 (very) common CPU cores. The final preprocessing script takes 1 minute to run for the all training dataset (2011, 2012, 2013) (The preprocessing runs once for all). The code offers the possibility to start from a previous submission and to only change some parts of it, for example to only change CAT predictions on 2013. In this case, the algorithm is extremely fast, and can take less than 30 seconds to run.

4 Pending issues and ways of improvement

Our work on the challenge left some pending issues. Furthermore, we have a lot of ideas and ways to improve the algorithm we have.

4.1 Improve the cross validation

The cross validation raises some questions. At the end of the challenge, our cross validation is a classical KFold cross validation. We predict a random subset of the train dataset, training on the rest of the train data set. We could do it linearly in time, predicting the 2nd week learning on the 1st one, then the 3rd week learning on week 1 and 2, and so on. Averaging the results would lead to the cross validation score.

A solid cross validation would be the support for a custom grid search cross validated to better control the optimal parameters we want. The optimal could be between the best score on the leaderboard and the best score on cross validation, to assure good properties of generalisation. For the moment we mainly focused on the leaderboard score, and controlled the complexity of the model by a penalisation l2.

4.2 Improve the feature engineering

The development has already been made to add the temporal feature TIME_SINCE_EPOCH and to use dummy temporal features. We did not have time to fully exploit these and it is a way of improvement we had in mind. The intuition is that TIME_SINCE_EPOCH could help representing the dependence in time between CSPL_RECEIVED_CALLS, and dummy features could help the trees of XGBoost to select and split on a more precise set of features.

4.3 Improve XGBRegressor fundamental parameters tuning

We currently use a surrogate of linex loss for convenience and speed of development. Our XGBoost considers the problem as a linear regression with a mean squared error. The results are already quite good, but the fact that we lose the asymmetry between underestimation and overestimation of the number of received calls with mean squared error is problematic. Using a linex loss will probably allows us to get rid of the coefficients we use to multiply all the final predictions (we tried with these coefficients to move from an underestimate to an equivalent overestimate). Changing the objective could also handle the negative XGBoost predictions we observed and that we currently handle manually.

4.4 Adopt other approaches and finally use an ensemble method

Another idea we had at the beginning but which was finally too ambitious was to try to tune an autoregressive approach like ARIMA, and a recurrent neural network. It would be interesting to combine those approaches and the XGBoost one with an ensemble method. The submissions or the models could be cleverly blended (in order to use the model that makes the lowest error per assignments or per period).