ECOLE POLYTECHNIQUE

INF554: Machine Learning I

AXA Data Challenge

Group:
loveandwai
Author:
DING Wensi
ZHANG ZEKUN
Mail:
wensi.ding@polytechnique.edu
zekun.zhang@polytechnique.edu

December 9, 2016

1 Introduction

1.1 Project Description

The specific project is also an AXA data challenge. Its purpose is to apply data mining and machine learning techniques for the development of an inbound call forecasting system. The forecasting system should be able to predict the number of incoming calls for the AXA call center in France, on a per "half-hour" time slot basis. The prediction is for seven (7) days ahead in time. More specifically, based on the history of the incoming calls up to a specific time stamp (we cannot use data/features that corresponds to future time slots), the proposed model should be able to predict the number of the calls, received seven (7) days later. In this way, the problem can be seen as a regression problem where the goal is to design a model that achieves to predict the incoming calls of the AXA call center with high accuracy.

1.2 Summary of Pipeline

Our pipeline of this project could be summarized as follows:

- *Data pre-processing*: After loading the data, a pre-processing task will be done to transform the data into an appropriate format. Our main task here is to select only a part of the data that is useful to us.
- *Feature engineering - add new features*: there are many features lying beneath the given data. Since the granularity of given data set is very small, we could consider creating larger time window for received data calls.
  - same time but seven days before current day
  - same weekday of one week before
  - average value for last week
  - .etc
- *Feature engineering - dimensionality reduction*: The next step involves the feature engineering task, i.e., how to select a subset of the features that will be used in the learning task (feature selection). We tried to apply dimensionality reduction algorithm in order to improve the performance of the algorithms. With PCA method, we observed that most of the selected features are mutually independent. Moreover, since the number of our final features (19) is small compared to the size of data set ($> 40,000$), doing feature selection and reduction is not really necessary in this condition.
- *Learning algorithm*: The next step of the pipeline involves the selection of the appropriate learning (i.e., regression) algorithm for the problem. Since the behavior of each *ASS-Assignment* is rather different, we applied two algorithms to each of them: *linear regression* and *gradient boosting tree*. After comparing the error of test data set for each *ASS-Assignment*, we choose the better one between the two algorithms.
- *Evaluation*: The final result of our `submission.txt` on the *Leaderboard Platform* is 1.0003. However, we discovered that if we multiply our final submission file by a factor greater than 1, for example, 1.2 or 1.5, the score on *Leaderboard Platform* will be smaller. This is because the *exp* loss function unreasonably over-encouraged the higher prediction. So we suggest to combine the normal *MSE error* with the given *EXP error*, to give a more reasonable prediction, which is slightly higher than real value, but still staying in real value's neighborhood.

2 Pipeline of data pre-processing

2.1 Data global view

First of all, we spent 2-3 hours investigating the given `train_2011_2012_2013.csv`. According to the description of this file, some of the attributes are about the description of phone calls, like their duration of time, or which steps they have gone through before finish, etc. Therefore, we simply neglect those attributes in the first step of pre-processing.

Since our prediction is about `CSPL_RECEIVED_CALLS`, we would like to have a general look about this attribute, according to daily bases and `ASS_ASSIGNMENT`.

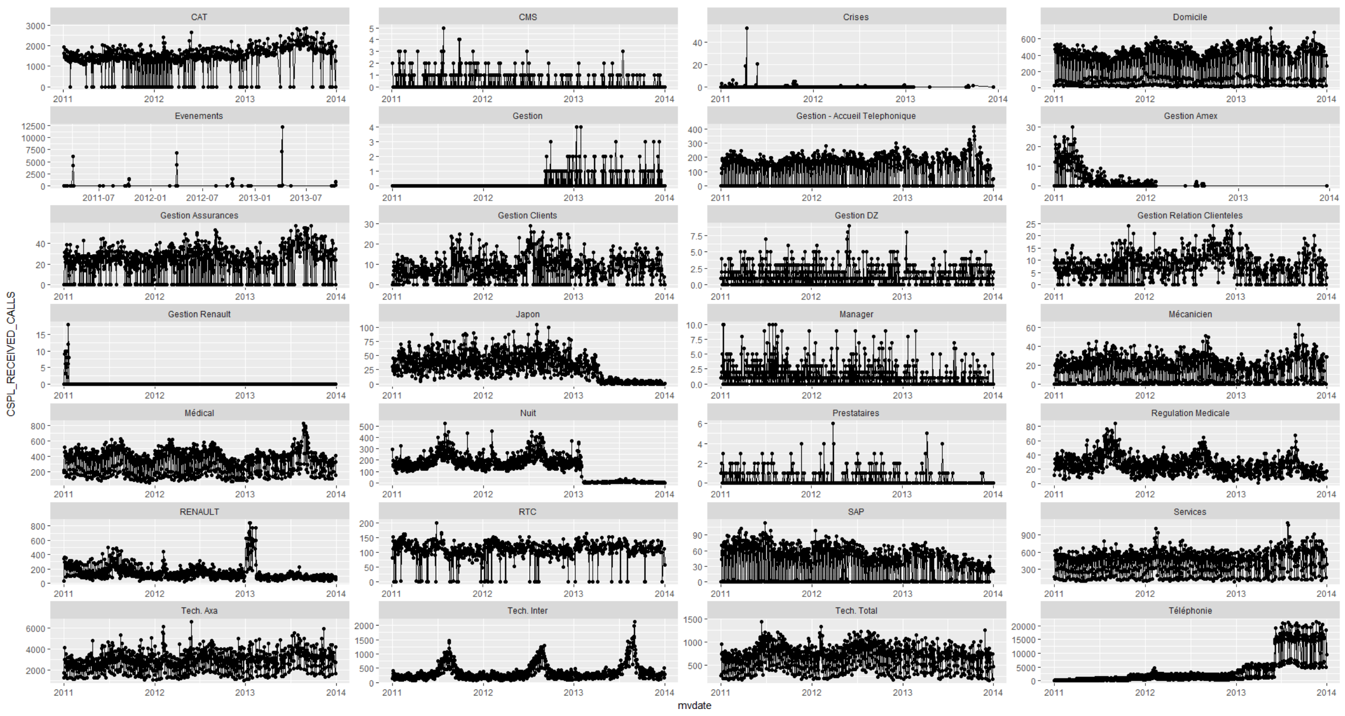


Figure 1: Daily calls for all ASS Assignment

We could easily observe four main features for the variation of `CSPL_RECEIVED_CALLS` in each `ASS_ASSIGNMENT`.

- Different `ASS_ASSIGNMENT` varies a lot on daily `CSPL_RECEIVED_CALLS`, from 10 to 20,000.
- Most `ASS_ASSIGNMENT` are below 100 `CSPL_RECEIVED_CALLS` per day. Therefore, prediction errors for these `ASS_ASSIGNMENT`s won't become too high. And they could use a same model to train their individual parameters and the final results could still be acceptable.
- Some `ASS_ASSIGNMENT` have high `CSPL_RECEIVED_CALLS` per day (> 500), but they follow a regular pattern over time. So predicting their values will be relatively easy.
- Two `ASS_ASSIGNMENT` have more special behaviors than others: `CAT` and `TELEPHONIE`. For `CAT`, it could have abrupt zero point among high values(2000 per day). For `TELEPHONIE`, in the middle of year 2013, it has a sudden increase of calls (from 2,000 to 15,000 per day). Therefore, we need to try different models and extract special features for these two important `ASS_ASSIGNMENT`s.

2.2 feature engineering I: specific features' extraction

From `train_2011_2012_2013.csv`, we extracted four attributes that we think more important: `DATE`, `DAY_WE_DS`, `ASS_ASSIGNMENT` and `CSPL_RECEIVED_CALLS`, into a new file named `databrute.csv`

Our main idea is to construct a complete table to store all possible values of `DATE`, `DAY_WE_DS`, `ASS_ASSIGNMENT`, leaving `CSPL_RECEIVED_CALLS` temporarily as `NA`. So in general, we have

$$N_{NumberOfAllDays} \times 48_{EveryHalfHourInOneDay} \times M_{NumberOfAllAssignments}$$

rows of data, using `expand.grid` command. The new generated table is stored in `rawdata`

Then we went through existing data `databrute.csv`, while filling corresponding `CSPL_RECEIVED_CALLS` in `rawdata`.

After that, our `rawdata` still has some `NA`s for attribute `CSPL_RECEIVED_CALLS`. These `NA`s could be classified into two categories:

- The data to be predicted in the `submission.txt`
- No record for this `ASS_ASSIGNMENT` at this `DATE`

Thus, by comparison with `submission.txt`, we mark the rows needing to be predicted in `submission.txt` with *true* `NA`. As for the `NA`s created because of the absence of the specific data, we have considered two methods:

- Replace the `NA` by the mean of its neighborhood. This method is more standard and secure. But we could have the problem of irregularity of neighborhood: for example, for dates near the prediction dates, we could only get the data before/after to calculate the mean.
- Replace the `NA` by 0. This is more simple and straightforward. We have talked about this problem with the professor, but with unconfirmed response, this method is practiced and our final prediction is acceptable. Therefore, looking backward, we consider this assumption as efficient.

2.3 feature engineering II: features' generation

Our next step is to discover more attributes based on the given data. There are three important insights that determine our final choices of generated features:

1. We begin with generating all the features concerning time. Here we list them as follows :
    - `mymonth`, original value 1-12, divided by 12
    - `myday`, original value 1-31, divided by 31
    - `DAY_WE_DS`, original value 1-7, divided by 7
    - `hourindex`, original value transformed into 1-48 corresponding to $0:00$-$23:30$, divided by 48
2. Since the original granularity is too small: every half hour, we would like to get more general or global attributes, for example, weekly mean of last week, or monthly mean of last month, etc. Thus, we add three more features:

- `calls_same_hour_before_seven_days` (half hour received calls of one week before)
- `calls_same_weekday_before_seven_days` (total calls received for total day of one week before)
- `daily_mean_calls_last_week`(total calls of last week divided by the number of last week's days)

Of course, these three features are all normalized by extracting the minimum value and then being divided by the range. We didn't take last month's average into account, because of adding this feature will reduce our training data set's size (first month will not be included in training data).

- We name this training data set as *training data v1*.

3. Since we have observed clear periodic features from the global view picture shown above, and also the apparent periodic behavior in one week as shown below, we would like to introduce this periodicity into our model. Therefore, after the normalization process described as before for each attribute, we expand it by multiplying π and then taking it's `cos()` and `sin()` values. In this way, we change 4 time features to 8 time features, adding up to 11 features in total.

   Besides, thanks to the Kaggle Challenge for predicting electricity load(*Some Practical Applications of Neural Networks in the Electricity Industry*, Philip David Brierley), we have decided to use `cos()` and `sin()` trick to deal with all periodic features.

   - We name this training data set as *training data v2*.

4. Another intuition comes from the observation of one day's calls for `CAT`. The whole day's variation follows the similar tendency for all days. Therefore, we considered about adding higher order term for `hourindex`, i.e., up to order 4 polynomials in features. Similarly, we extend this polynomial feature for `mymonth`, `myday`, `DAY_WE_DS`. In the end, we have $4 \times 4 + 3 = 19$ features in total.

   In later time, we added two more features for `Telephonie`. Because we observed the abrupt increase in mid 2013. One parameter is whether the prediction date is before or after that sudden change point. The other parameter is taking year $2011, 2012, 2013$ as new features.

   However, these two features are added to adjust the bad predicting performance for `Telephonie`, and only applies for data in 2011/2012/2013. Thus, it has very little ability to predict another abrupt change in the future like year 2014/2015...

   - We name this training data set as *training data v3*.

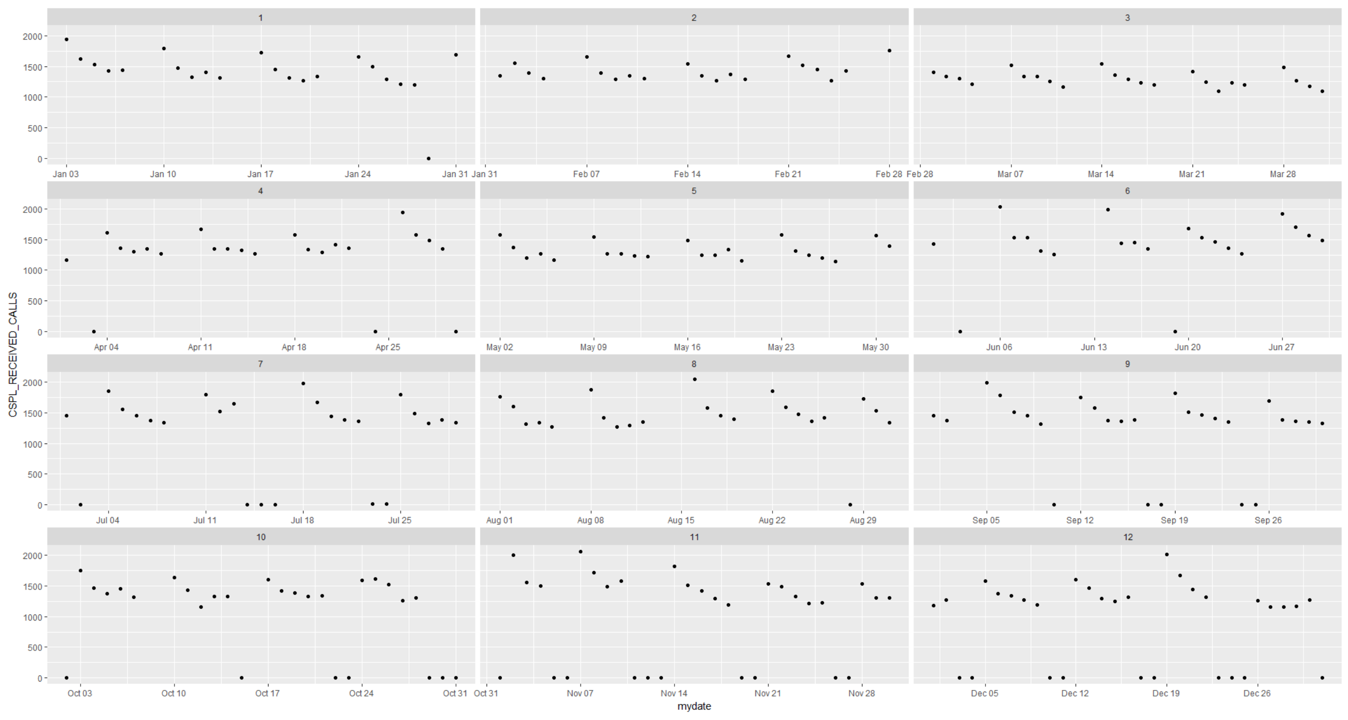


Figure 2: CAT calls' variation in one year

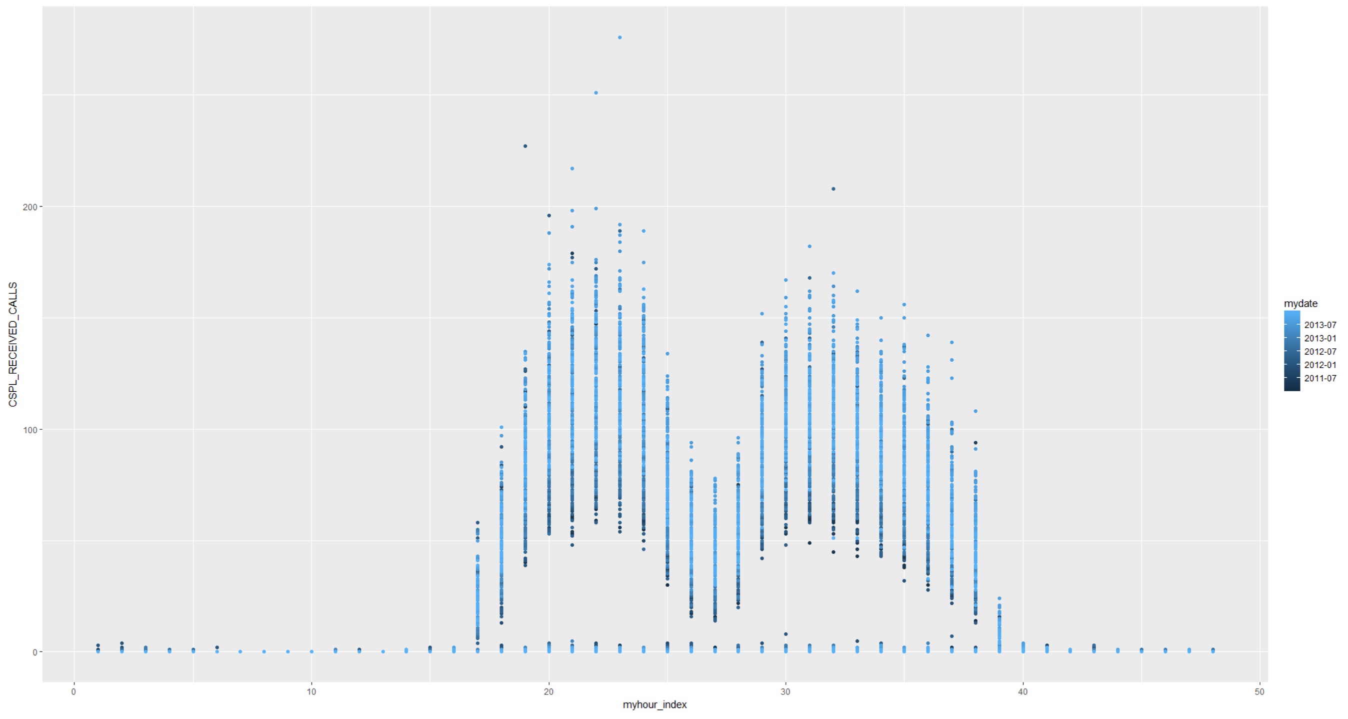
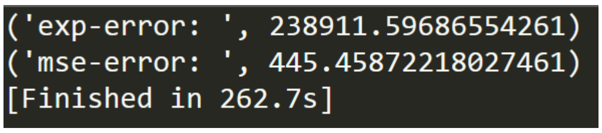


Figure 3: CAT calls' variation in one day

3 Choice of Algorithms

3.1 SVR with kernel

Firstly, we chose *Support Vector Regression* because of the application of this algorithm for predicting electricity load. We implemented this algorithm with *linearKernel* and *gaussienKernel*, using the given loss function. However, in practice this algorithm doesn't work well because recalculating all distances between all points($> 40,000$ points) with kernel function and doing gradient descent without special optimization for large dimensions all cost too much time. Instead, we used `scikit-learn` package with *epsilon* error to give a general idea about its performance. The final result is as follows:



```
('exp-error: ', 238911.59686554261)
('mse-error: ', 445.45872218027461)
[Finished in 262.7s]
```

Figure 4: SVR with MSE loss function, *training data v2*

We could draw the conclusion that *SVR with kernel* is not efficient in training the data set, both in accuracy and duration of time if we don't change the loss function to our self-defined loss function. Although we can easily implement the svr algorithm by ourselves with our self-defined loss function. But without the appropriate optimization of gradient descent, the training process can take extremely long time. And we didn't succeed in finding a library of svm which provides a self-defined loss function and at the same time offers optimization of gradient descent. So we left this approach for future research.

3.2 Linear Regression

Next we considered much more simpler model: linear regression model with given loss function. We used `scipy.optimize` package to calculate the best parameter for *training data v1/v2/v3*. Instead of simply optimizing for the given exponential loss function, we also looked at the normal *mse error* to assure that the prediction stays in the neighborhood of real value. Our final result is shown in the following two pictures:

From the two pictures, we could draw the conclusion that

1. `CAT`, `Tech.Axa` and `Telephonie` are main causes of error for prediction.
2. Other `ASS_ASSIGNMENT`'s errors are very small, both *exp-error* and *mse-error*, meaning our choice of features has done a good job in predicting their values.
3. We need other algorithms to make better predictions for `CAT`, `Tech.Axa` and `Telephonie`.

3.3 Gradient Boosting Regression

We applied *Gradient Boosting Regression* because of following reasons:

- Gradient Boosting algorithm is recommended in the given introduction of project.
- Tree boosting algorithm is more powerful to deal with sudden rise or fall in data set. This is because tree ensemble model uses additive functions to predict the output. And each additive function takes the form like step-wise function.
- Unlike `scikit-learn`'s `gradient boosting regressor`, tree boosting algorithm could be implemented in XGBoost with customized loss function. Thus it will be efficient and convenient to train and predict.

To learn the set of functions used in thee model, we minimized the following *regularized* objective.

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

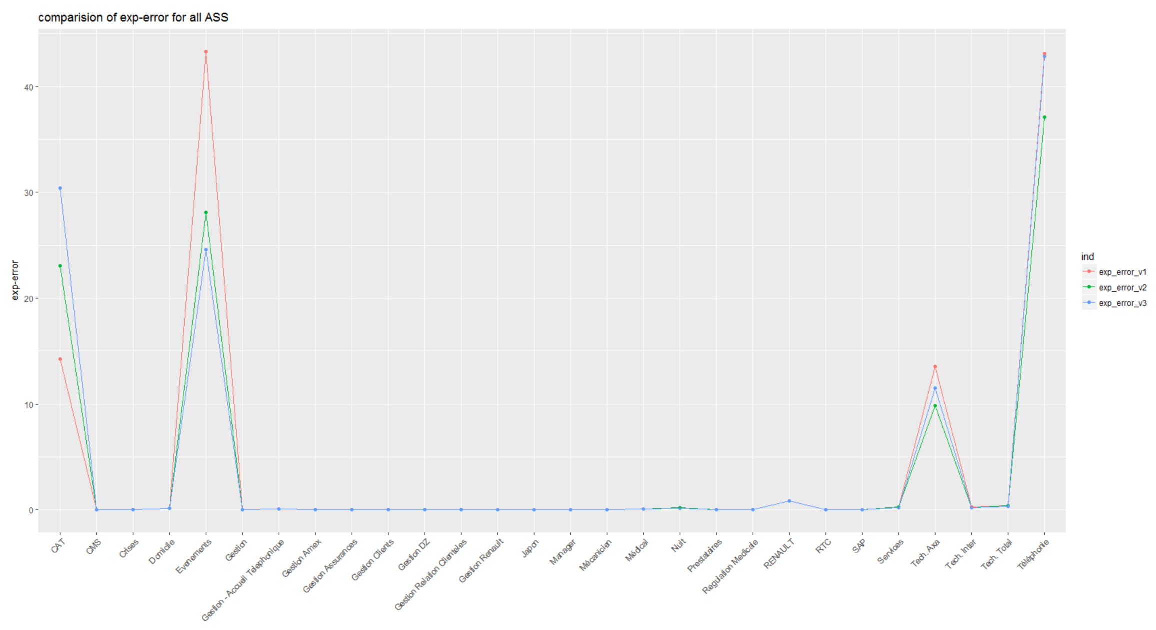


Figure 5: exp-error, linear regression, given exp loss function

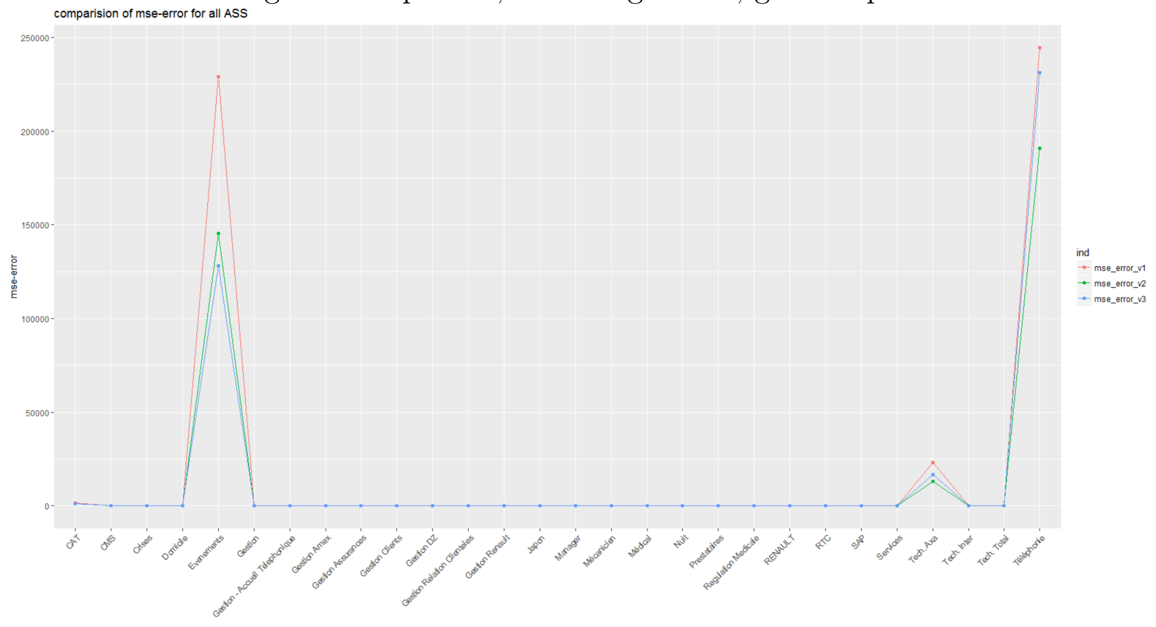


Figure 6: mse-error, linear regression, given exp loss function

where

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$$

Here l is loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term Ω penalizes the complexity of the model (i.e. the regression tree functions, in terms of tree depths or size of leaves).

Sometimes, when it's difficult to find the minimum of loss function with an analytic form (like the given loss function), we could expand loss function into Taylor series to order 2. That's why we need to give derivative and second derivative in *gradient boosting tree*.

Our training result for different number of features (*training data v1/v2/v3*) is as follows:

We could easily observe that *exp-error* and *mse-error* have been greatly reduced for `CAT`, `Tech.Axa` and `Telephonie`

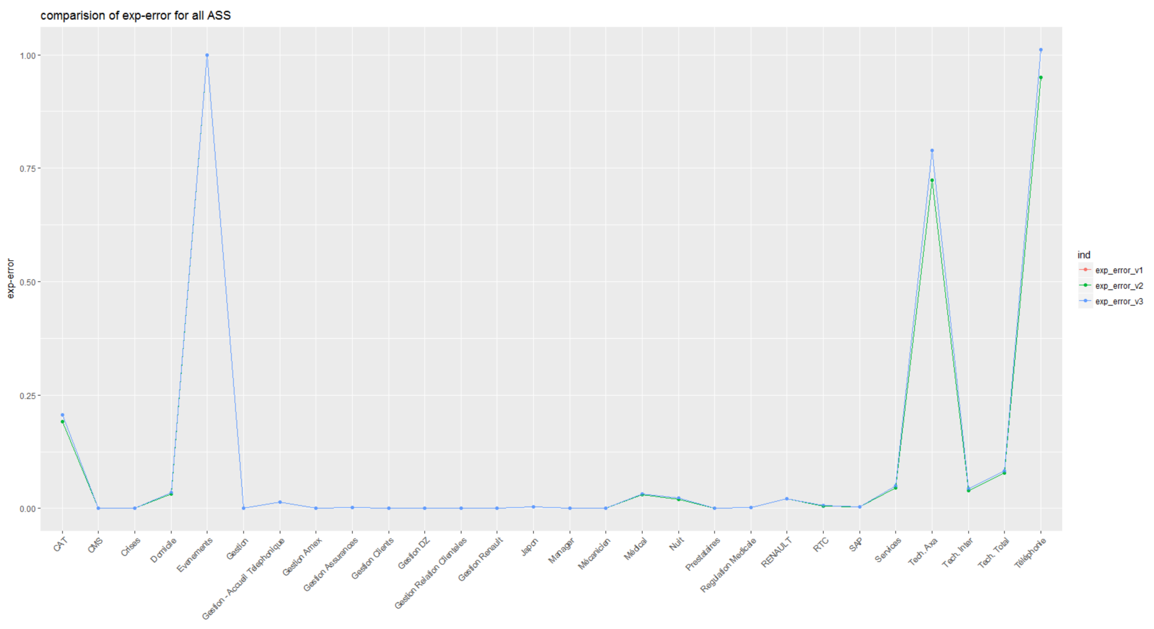


Figure 7: exp-error, boosting regression, given exp loss function

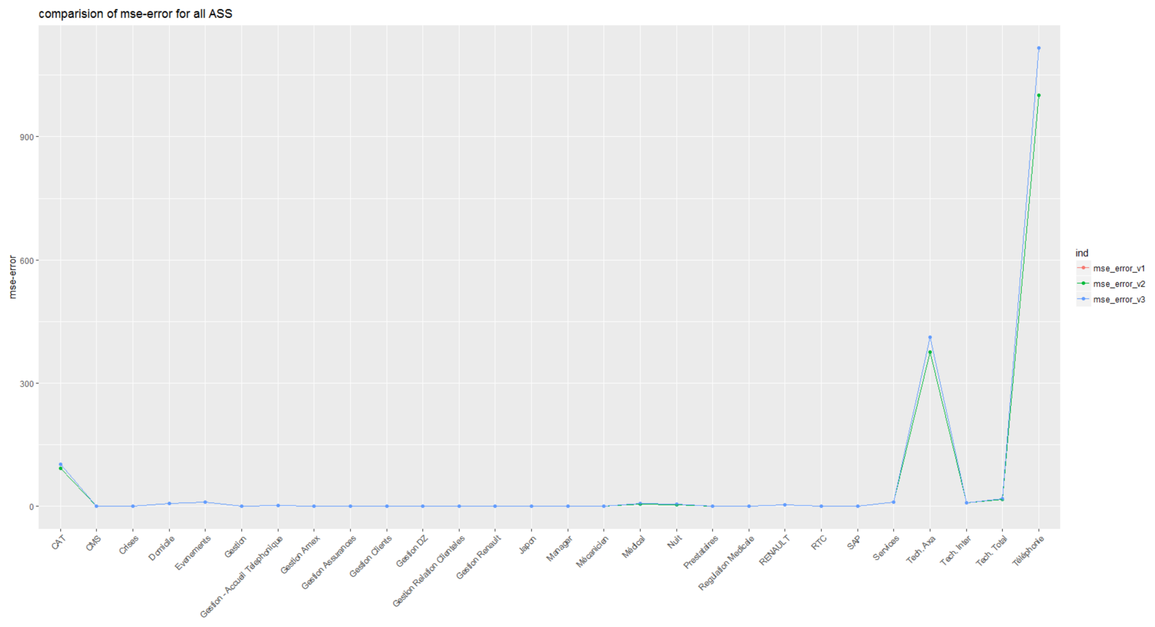


Figure 8: mse-error, boosting regression, given exp loss function

Evaluation

To evaluate our model, we used cross-validation method to determine global parameters in our model. In practice, we chose the function `RandomizedSearchCV` provided by the `sklearn` `grid_search` package.

For the linear regression model, we only have one global parameter : the regularization parameter. So we did the random search for this parameter for each kind of data set with different features. The final result shows that when regularization parameter goes to zero, we gets the best cross-validation test. We think this is due to our limited number of features($10-20$ features for $> 40,000$ data), regularization parameter almost has no impact on loss function.

For gradient boosting tree algorithm, we couldn't utilize this function of `sklearn`. Because the import training data is stored in different form for our XGBoost (DMatrix) and for `sklearn`. So we tested different sets variables manually, and chose a better set of global parameters, using still cross-validation method.

The best score on leaderboard is achieved by using the prediction of *gradient boosting tree* for `CAT` department, the prediction of *linear regression* using the data version3 with two added features for `Telephonie` department and the prediction of *linear regression* using the data version3 for other `ASS_ASSIGNMENT`.

Conclusion

We have investigated this data-predicting problem in several aspects. Firstly, by observation of periodic behaviors of history data set, we created new features of *sin* and *cos* for every time feature (hour, weekday, day, month). Secondly we expand these features by their high-order corresponding features, so as to better fit daily, weekly and monthly distribution. Last but not least, we use different algorithms for different *ASS-Assignment*, such that our global performance is improved.

In the end, we obtain score of 0.9 on leaderboard, with processing time less than 1 minute on $i5-4210H$ CPU. The overall performance of our algorithms is good enough to put into practice.

References

[1] Tianqi Chen, Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System.* arXiv:1603.02754v3, 10 Jun 2016

[2] Philip David Brierley *Some Practical Applications of Neural Networks in the Electricity Industry.* 1998