A Lucky Tractor

3rd Ranking Solution to *Asia Actuarial Analytics Challenge 2017*

October 2017

Name: Xiaodong DENG (Manulife Insurance Singapore)

Email: xd.deng.r@gmail.com

Website: seekingQED.com

Competition: Asia Actuarial Analytics Challenge 2017 (link)

0. Preface

You may find the ideas used and the pipeline built in this study are quite straightforward, even too simple to some readers. No special trick, no trump card, no magic external data introduced. The situation is more or less like a tractor competing with a Ferrari. When everything is in better shape (like more observations are available, more balanced positive-negative ratio, more clear-cut boundaries, etc.), a perfectly tuned model may outperform with big margin, just like a Ferrari can reach 100 MPH within a few seconds while the tractor just starts to move. However, a tractor may still stand a chance to win if they're both running on a mud road rather than highway. In some applications, especially the industry ones, robustness and simplicity should be considered as one part of the model performance as well, in order to prepare for mud roads.

That being said, some steps in my model building pipeline may or may not be the actual best practice. Randomness was somehow involved in the final ranking of this competition, given the difficulty to be more creative in this study, like introducing external data or building new feature. That's also why I would more like to call my model "**a lucky tractor**".

This documentation will mainly discuss about the model itself. For background and more information about the problem itself, please refer to the competition homepage https://www.kaggle.com/c/asia-actuarial-analytics-challenge-2017.

1. Background

Academic/professional Background

Education
M.Sc in Mathematics, National University of Singapore, Singapore
B.Sc in Applied Mathematics, Beijing Forestry University, China

Working Experience
2017 May - Present: Data Analytics Function, Manulife Insurance Singapore
2016 April - 2017 April: Data Analytics Specialist, AXA Insurance Singapore
2015 January - 2016 March: Research Assistant, National University Health System

Any prior experience that helped you succeed in this competition?

My quantitative education background, as well as my working experience in the insurance industry, did help me understand the problem faster. Some projects I worked on were more or less similar to the problem of this competition as well.

How much time did you spend on the competition?

I spent my off time of a few weeks on this competition. Later I realised the room for further improvement was relatively small so I stopped further tuning (my last submission was 4 months prior to the deadline).

2. Summary

Python was used for this study as the main tool, with *Numpy* and *Pandas* for data processing, *matplotlib* for visualisation, *Scikit-Learn* and *XGBoost* for the modelling (not surprisingly). Since there were quite a few hyper-parameters to tune, a simple greedy algorithm was implemented to determine the optimal hyper-parameter combination (section 4.1), which took about one and a half hours (of course to understand and fully explore the problem took much longer). After the hyper-parameters were determined, the model fitting and predicting took a small amount of time. A majority of the most important features are those economic indicators, like "Euribor3m", No_Employed", "Monthly_CCI", "Emp_Var_Rate", together with a few client profiling features, like “Age” and “Job” (illustrated in detail in section 5.1).

3. Features Selection/Engineering

3.1 Feature Selection

Careful feature selection was skipped.

Some methods, like *Stepwise*, was tried to select features, but the corresponding submissions didn’t give better performance. I would like to learn from other solutions if they have any good ideas for this part.

3.2 Data Transformation

To address a few data quality issues, the rules below were applied on both Train and Test data.

- If **NoContacts_PrevCampaigns** == 0, **Pdays** = 999
- If **NoContacts_PrevCampaigns** == 0, **Prev_Outcome** = 'nonexistent'
- If **Pdays** == 999, **Prev_Outcome** = ‘nonexistent'.

In addition, we have some (unordered) categorical features which need to be processed before we can dump them into the model. One way is to convert them into dummy variables. For example,

	ID	Age	Gender
0	1	35	F
1	2	40	M
2	3	21	F

	ID	Age	Gender_F	Gender_M
0	1	35	1	0
1	2	40	0	1
2	3	21	1	0

Another way for categorical features transformation is to order the feature classes according to the proportion falling in outcome class 1. Then we split this feature as if it were an ordered predictor. It can be shown that this gives the optimal split, in terms of cross-entropy or Gini index [1]. We need to note that the transformation statistics should be obtained from Train data, then applied on Train data and any later data, say Test data. For example, if Males' proportion falling in outcome class 1 is higher than that of Females, we will transform "Male" into 1 and "Female" into 0.

	Outcome	Predictor
0	0	A
1	0	B
2	1	A
3	0	C
4	1	B
5	1	A

	Predictor Class	Proportion Falling into Outcome Class 1
0	C	0
1	B	1/2
2	A	2/3

```
{'A': 2, 'B': 1, 'C': 0}
```

The second way was used in this study.

3.3 External Data

Given the original data was altered in an unknown way, it became difficult to introduce external data in this study (like macroeconomic indicators other than those provided in the data). A few other participants also mentioned this.

4. Training Methods

4.1 Hyperparameter Tuning

Hyper-parameter tuning is essential for Boosting algorithms. For XGBoost, there are a few hyper-parameters we need to pay attention to [2]:

- **eta** (learning rate): step size shrinkage used in update to prevent overfitting.
- **max_depth**: maximum depth of a tree. Increasing this value will make the model more complex / likely to be overfitting
- **subsample**: subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
- **gamma**: minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be.
- **colsample_bylevel**: subsample ratio of columns for each split, in each level. This is a concept from *Random Forest* [3]. It can help de-correlate the trees, in order to make result less variable and hence more reliable [4].
- **scale_pos_weight**: Control the balance of positive and negative weights, useful for unbalanced classes.

The objective would be to determine the optimal hyper-parameter combination $\widehat{P}$,

$$\widehat{P} = argmax < modelperformance >$$

Normally we can use **Grid Search** to determine which hyper-parameter combination gives the best cross-validation metric on Train data, so that we can apply the same combination on Test data to make prediction. But this may be quite time-consuming. Of course we can also use **Random Search** which can help reduce the time needed, but the training process is still expected to be long [5].

To address this issue, "greedy algorithm" idea was used instead. From an initial hyper-parameter combination (normally using the default value for each parameter), different values (given in advance) will be tried for each hyper-parameter and the one producing best cross-validation metric on Train data will be chosen. This process will be run until the hyper-parameter combination doesn't change anymore (algorithm is described as below). But we only run the iteration for limited times here due to computing power & time restriction.

Algorithm

Given hyper-parameters P_i ($i = 1,...,n$. n is the number of hyper-parameters to tune). For each P_i, we have a vector of a few candidate values, C_i. In addition, we have a hyper-parameter combination $\widehat{P}$ containing initial values.

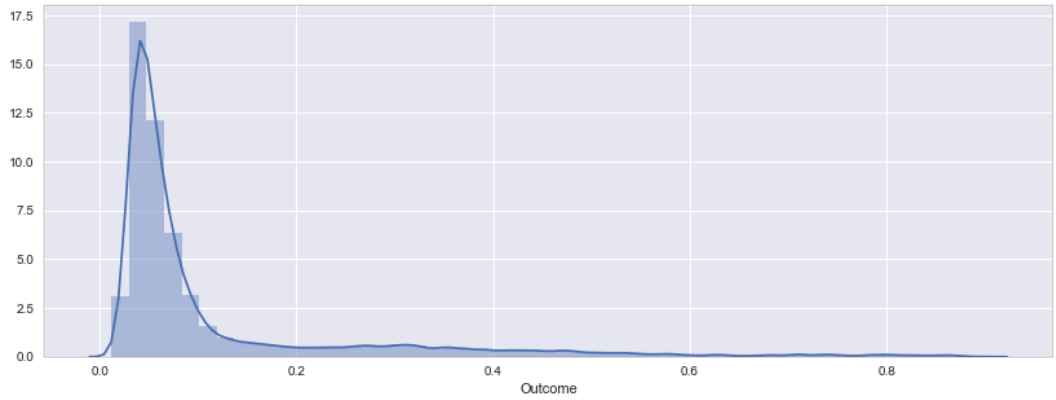
Step 1:
For i in 1 to n:
Use different values in C_i to update the P_i in $\widehat{P}$ (keeps others unchanged), the one that outputs the best cross-validation metric will be used to update the P_i in $\widehat{P}$ before go to next iteration.

Step 2:
Check if $\widehat{P}$ after Step 1 is identical to that before Step 1:
If Yes: Stop.
If No: Go to Step 1.

Through this process, we can also estimate how many boosting iterations we should have (parameter **num_boost_round**) later when working on Test data.

4.2 Model Fitting & Prediction

After hyper-parameter tuning, the rest was quite straightforward.



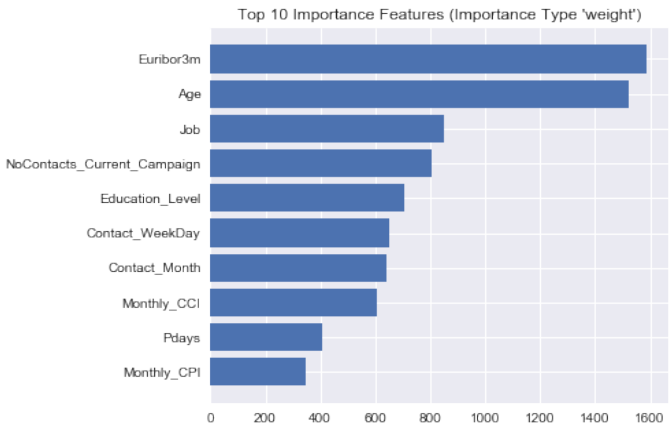
Distribution of Output

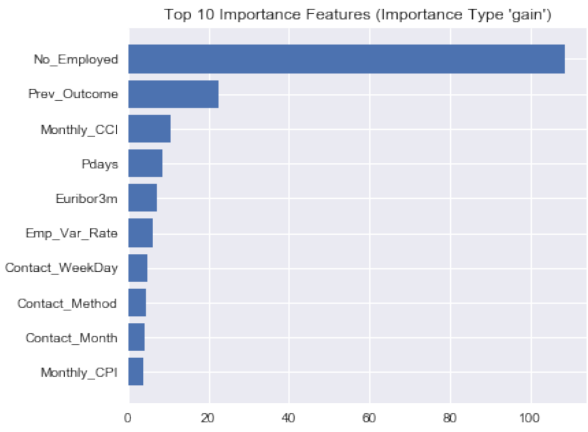
5. Findings

5.1 Important Features

The most important features were mainly from two categories, Economic Indicators and Client Profiling, among which the former ones played a bigger role. This was more or less against my presumed intuition. This may be due to the economic environment's influence on people's propensity to buy financial products, especially given the original data before altering were collected from May 2008 to November 2010, which was just during & soon after the Financial crisis of 2007-2008 [6].

Top 10 Features by "weight"			Top 10 Features by "gain"		
Feature	**weight**	**gain**	**Feature**	**weight**	**gain**
Euribor3m	1589	7.183052	**No_Employed**	217	108.532030
Age	1524	3.350808	**Prev_Outcome**	84	22.538321
Job	849	3.487242	**Monthly_CCI**	606	10.618265
NoContacts_Current_Campaign	803	3.628028	**Pdays**	407	8.521599
Education_Level	704	3.431314	**Euribor3m**	1589	7.183052
Contact_WeekDay	650	4.705276	**Emp_Var_Rate**	182	6.222029
Contact_Month	639	4.175492	**Contact_WeekDay**	650	4.705276
Monthly_CCI	606	10.618265	**Contact_Method**	189	4.327922
Pdays	407	8.521599	**Contact_Month**	639	4.175492
Monthly_CPI	349	3.776362	**Monthly_CPI**	349	3.776362



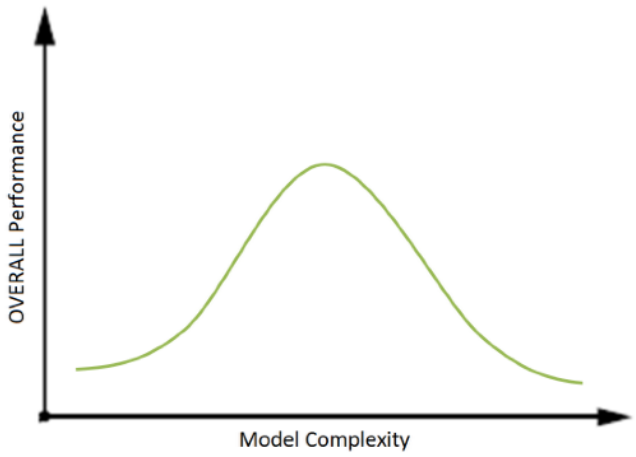


5.2 Importance of Model Simplicity

A lesson I learnt from this competition, as well as from my daily works, is the importance of reducing model complexity. There are a few advantages of a simpler model:

- Shorter time to build the model
- Less effort required for tuning (less hyper-parameter)
- Easier to understand & interpret
- Lower chance to overfit

Meanwhile, the performance doesn't have to be much worse. For example, my another submission using Random Forest got 0.795+ Private Score, which looks good enough to me, while the model building process was much simpler compared with using XGBoost. This may seem less important in a Kaggle competition scenario, given we have a few months to try and even 0.001 AUC increment may result in a big difference in the final ranking. But imagine we're doing a real-world project: tight timeline, much bigger data, collaborators without any technical background, etc. In such scenario, we'll definitely appreciate model simplicity.



References

[1] Chapter 9.2.4, "Other Issues - Categorical Predictors", Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning, Second Edition, Springer.

[2] XGBoost Parameters, https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

[3] Chen, Tianqi, and Carlos Guestrin. "XGBoost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

[4] James, Gareth, et al. An introduction to statistical learning. Vol. 112. New York: springer, 2013.

[5] Tuning the hyper-parameters of an estimator, http://scikit-learn.org/stable/modules/grid_search.html

[6] UCI Machine Learning Repository, Bank Marketing Data Set, https://archive.ics.uci.edu/ml/datasets/bank+marketing

Appendix

A1. Model Execution Time

- Hardware (CPU spec, number of CPU cores, memory)?
  MacBook Pro (Early 2015); 2.7 GHz Intel Core i5, 4 virtual cores; 16GB RAM.

- How long does it take to train your model?
  About one and a half hours (mainly for tuning the hyper-parameters).

- How long does it take to generate predictions using your model?
  About one minute was spent for training on whole Train data, as well as making predictions for Test data.

A2. Dependencies

A2.1 Software
- Python 2.7.13 | Anaconda custom (x86_64)
- Jupyter Notebook (4.3.0)

A2.2 Libraries
- numpy (1.13.0)
- pandas (0.20.2)
- scikit-learn (0.18.1)
- xgboost (0.6)
- matplotlib (2.0.0)
- seaborn (0.7.1)

A2.3 Operating System
- Mac OS Sierra (Version 10.12)

A3. Codes

Please refer to GitHub repository https://github.com/XD-DENG/solution-asia-actuarial-analytics-challenge-2017 .

Note: you may notice the hyper-parameter combination produced by step-1 is different from that used in step-2. That's because I didn't bother to set random seed for step-1 script given its exploration nature. We can't perfectly reproduce the result of step-1. But the random seed was set for step-2 so current script can exactly reproduce my submission.