# Binary Classification of Imbalanced data from Bosch Production Line

**P-1: Xi Yang Liang Dong Yeojin Kim**

## 1 Introduction

The big data analysis is one of the challenging problems in modern manufacturing industries [1], since the target datasets mostly have complex inheritances, which are large-scale, imbalanced [2], high-dimensional, with different types of features, and including lots of missing values. In this report, we focus on processing the imbalanced data with different types of features, and exploring a best strategy to process a large dataset. For this study, we used Bosch Production Lines dataset from the Kaggle competition, which has the properties described above. The remainder of this report is as follows: Section 2 describes the brief characteristics of the used dataset, Section 3 presents the machine learning procedure and methods to tackle this problem, Section 4 accounts the experiments and the results. Finally, in Section 5 we will discuss the contributions and limitations of our study.

## 2 Data

The data is from an ongoing Kaggle competition: Bosch Production Line Performance [3]. The data represents measurements of parts as they move through Bosch's production lines and it is a binary classification problem.

The raw data has three types of features: numerical, categorical, and date. The number of features for each type is 968, 2140, and 1157, respectively, and the total number of features is 4265. The total number of instances is 118347 in the training data, and 1183748 in the testing data. In the training dataset, the positive instances are 6879, and the negatives are 1176868, and so its ratio of positives and negatives is 1:171 in the training dataset. It is noted that the ground truth of testing data is not open to public so that we could only check the prediction results by submitting the labels to Kaggle site. The total size of raw data is large(15.42GB) and cannot directly fit into a single machine. We will do data exploration and feature reduction to fit important features into a single memory with 16GB memory.

Table 1: Description for Bosch Training Dataset

| Training Data | | | | Testing Data | | | |
|---|---|---|---|---|---|---|---|
| Data Types | Features | Instances | Size ($GB$) | Data Types | Features | Instances | Size ($GB$) |
| Numeric | 986 | 118347 | 2.14 | Numeric | 986 | 118348 | 2.14 |
| Categorical | 2140 | 118347 | 2.68 | Categorical | 2140 | 118348 | 2.68 |
| Date | 1157 | 118347 | 2.89 | Date | 1157 | 118348 | 2.89 |
| Total | 4265 | 118347 | 7.71 | Total | 4265 | 118348 | 7.71 |

## 3 Method

### 3.1 Data Analysis Framework

Our whole data analysis framework is delineated on Figure 1. The inputs are three kinds of data, numeric, categorical, and date. We followed this framework: 1) the preprocessing includes missing

data imputation, feature extraction, and feature merge. 2) Then we applied Xgboost as classifier with the merged features. 3) To build the model and optimize the hyper-parameters, we plan to do sampling. 4) Lastly, we need to input our testing into the classifier and get the predictive labels. The results will be submitted to Kaggle to evaluate the performance of our classifier.

Following the above two procedures, we will then train the machine learning models with gradient boosting tree through Xgboost, and then compare its performance with other decision models, including Linear SVM and SAS Viya.

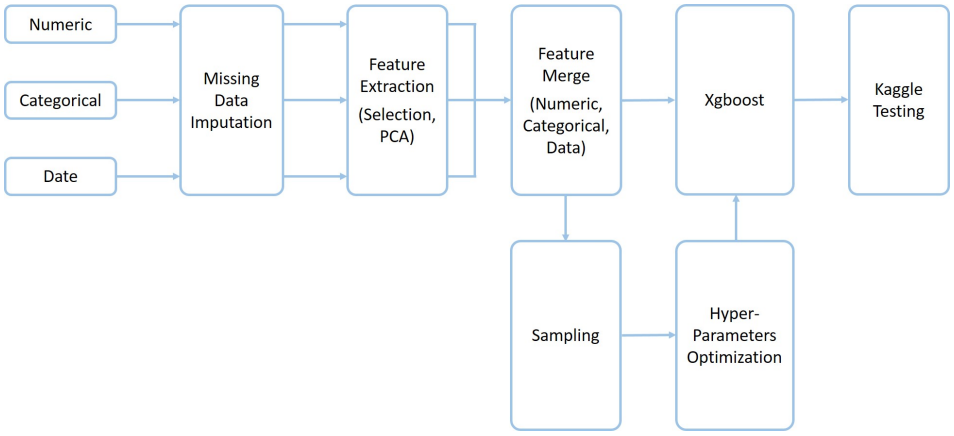


Figure 1: Data Analysis Framework

### 3.2 Missing Data Imputation

There are lots of missing data in our dataset. The general approaches to handle missing data include three categories: deletion, imputation, using decision model handling missing data. Even though our adopted classification model, XGBoost can inherently handle missing values, we imputed extreme values to missing values to apply a dimensionality reduction method, PCA. In this preprocessing step, we set -1 for missing values in categorical data because there is no value with -1, and set -1.1 for missing values in numeric data because its value range is equal or greater than -1 and equal or less than 1.

### 3.3 Feature Extraction

The dimension of dataset is so high, which is total 4265 from numeric, categorical, and date. In order to reduce the dimension, we discarded features with less than 500 instances in categorical dataset, and also discarded features with redundant values in the same product lines in date dataset. Then we applied PCA to the numeric, categorical, and date dateset. The numbers of reduced features are 198, 22, 53, for numeric, categorical, date, respectively. Thus after feature extraction, we got total 263 dimension.

### 3.4 Sampling

For the future work, we will do sampling for the hyper-parameter optimization. To efficiently search the parameter spaces, we need to make the data size much smaller to fit into a single machine with 16GB memory. There are several sampling strategies which we can apply to the large dataset, such as SMOTE[4] and Borderline-SMOTE[5].

### 3.5 Gradient boosting tree

We choose Xgboost as our decision model with the following two reasons: 1) Comparing to traditional models, e.g. SVM and Logistic Classification, models based on tree-ensemble, e.g. Xgboost, are easier to control. Specifically, when the model is under-fitting, for traditional methods, we could increase feature size to improve the accuracy, and for the Xgboost, we could add more trees.

Comparing these two processing ways, number of trees is much controllable than the number of features, which would easily lead to over-fitting.

2) Tree-based methods like Xgboost could better handle noise occurred in practical problems, e.g. the Bosch production line, than traditional models. Also in practical applications, we have diverse features collected from different sources. For traditional methods, it's hard to define the similarity between features, e.g. kernel function in SVM, while the tree-based method like Xgboost is not sensitive to different characteristics among features.

Gradient boosting tree is achieving wide attention in Kaggle competitions. The basic idea of gradient boosting tree is to fit a base tree learner to pseudo-residuals.

$$r_{im} = -[\frac{\delta F(y_i, F(x_i))}{\delta F(x_i)}]_{F(x)=F_{m-1}(x))}, i = 1, ...n$$

Then computer $r_m$ by solving the following one-dimensional optimization problem:

$$\sum_{i=1}^{n} L(y_i, F_{m-1}(x) + rh_m(x_i))$$

There are many versions of gradient boosting tree, such as: 1) Scikit Learn gradient boosting tree: implemented in python and integrated in scikit learn package, relatively slow; 2) SAS Viya: implemented in C and MPI, developed by SAS, supports python and lua API. It only support equal binning now; 3) xgboost(`https://github.com/dmlc/xgboost`): C language, open sourced and supply good support with python wrapper and R wrapper; 4) FastBDT(`https://github.com/thomaskeck/FastBDT`): C language, open sourced and currently only suitable for classification problem; 5) LightGBM(`https://github.com/Microsoft/LightGBM`): C language, developed by Microsoft and open sourced recently, it is said to be much faster than xgboost and can achieve the same accuracy.

In this project, we mainly focus on xgboost and SAS Viya.

### 3.6 The magic feature

Some kagglers report that there is magic feature in the competition which will improve the performance in public board from 0.3 to 0.4. The magic feature comes from id and there is a debate if it is a leak. The leak is something that was introduced during data manipulation, either by data owners or by Kaggle. In essence, the leak is an unintentional tipping of the final outcome by those who know the outcome. We've had plenty of leaks that satisfy this definition. In this competition, the id column has leak information and we will use the information.

### 3.7 Grid Search

For gradient boosting tree, we need to do hyperparameter optimization for tree number, learning rate and other parameters. We tune the hyperparameters of a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set. In this case performance is evaluated by MCC. The traditional way of performing hyperparameter optimization is grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm.

### 3.8 SAS Viya python wrapper

SAS Viya is the next-generation high-performance and visualization architecture from the leader in analysis. It supplies python interface for gradient boosting tree, but it doesn't support grid search and cross validation APIs. To solve this problem, we wrote python modules for SAS Viya following scikit-learn architecture.

## 4 Experiment

In our experiment, the highest mcc score we got in public board now is 0.322, whereas the current highest mcc score in public board is about 0.51. We employed magic feature, grid search and other techniques in the experiments. These techniques are discussed in the following sections.

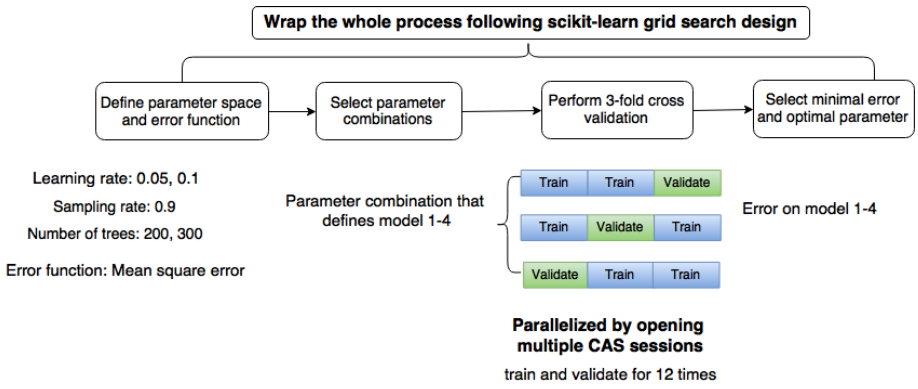


Figure 2: The Procedure of SAS Viya python wrapper

### 4.1 Optimizing probabilities for best MCC

The criteria for the competition is Matthews correlation coefficient, the equation to calculate it is as following. We will yield the threshold that yields the best MCC score.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

The steps are as following: ordering distinct probabilities, computing TP/TN when assuming the threshold for each distinct probability, computing the MCC, returning the probability row with the maximum MCC.

### 4.2 Equal Binning vs. Quantile Binning

Single machine version of Xgboost uses exact greedy algorithm that searches overall possible candidates, this can be desirable for deeper trees, and is usually what the user want. But this will slow the training speed when the number of observations is very large. So we use quantile binning of xgboost for this competition. We find that some features are normally distributed and others are quite skewed, the distribution of some features are shown in figure 3 and figure 4. That's the reason xgboost outperforms SAS Viya, SAS Viya only supports equal binning and cannot handle skewed feature well.

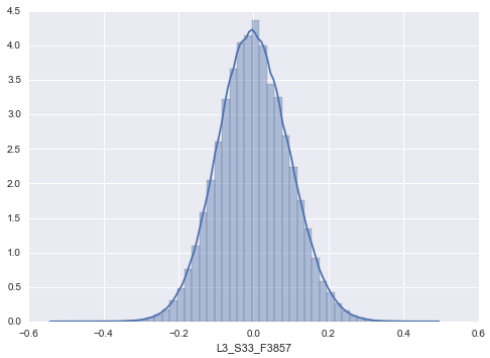


Figure 3: normal distributed feature

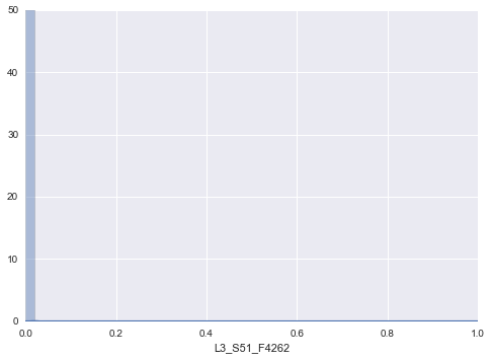


Figure 4: skewed feature

### 4.3 Grid search

We did stratified 2-fold cross validation in the grid search because the data is big and imbalanced. We fix learning rate as 0.1, we only tune tree depth and number of trees in current experiments.

### 4.4 Results

At this point, the best result regarding xgboost is 0.322, while Linear SVM, which is the baseline method has 0.04179 as the best mcc, and SAS viya has 0.19848 as the best result.

## 5 Discussion

The main difficulty in our project is to fit the data into a single machine and do hyperparameter optimization. SAS Viya can handle the data easily but the gradient boosting tree in SAS Viya cannot handle skewed feature. We had access to a machine with 128GB memory, but the grid search tuning is too time consuming. So in the next step we will do parameter tuning in sampling data.

## 6 Reference

[1] Choudhary, A.K., Harding, J.A. & Tiwari, M.K. (2009) Data mining in manufacturing: a review based on the kind of knowledge. *Journal of Intelligent Manufacturing*, **20**:501-521.

[2] He, H., & Garcia, E. (2009) Learning from Imbalanced Data. *IEEE Transactions of Knowledge and Data Engineering*, **21**(9):1263-1284

[3] https://www.kaggle.com/c/bosch-production-line-performance

[4] Chawla, N.V., Japkowicz, N. & Kotcz, A. (2002) SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, **6**(1):321-357.

[5] Han, H., Wang, W.Y., & Mao, B.H. (2005) Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In D.S. Huang, X.-P. Zhang, G.-B. Huang (Eds.),*International Conference on Intelligent Computing*, Part I, LNCS 3644, pp. 878-887. Springer-Verlag.

[6] Tianqi Chen, & Carlos Guestrin. (2016) XGBoost: A Scalable Tree Boosting System. *KDD.*

[7] Keck T. (2016) A speed-optimized and cache-friendly implementation of stochastic gradient-boosted decision trees for multivariate classification *arXiv preprint arXiv*: 1609.06119.

## 7 Appendix

### 7.1 Teammate and Work Division

Xi & Yeojin: Data exploration and feature reduction, sampling

Liang: Hyperparameter optimization, SAS Viya

### 7.2 Changes

One of our members, Weijie Zhou, left the team, because she wanted to study with smaller size of dataset.