Using Gradient Boosting Tree algorithms on Porto-Seguro's safe drive competition

Tuan-Binh Nguyen[*]

Abstract

We propose a method for tackling the problem of predicting the probability that an insurance claim would happen given some specific data of a car driver. The method utilizes feature (variables) extraction technique and Gradient Boosting Algorithm, combines by ensemble learning to achieve the result in top 35% of the Kaggle's private leaderboard.

1 Introduction

Porto Seguro is one of Brazil's largest auto and homeowner insurance companies. It constantly faces inaccuracies in car insurance claim, which raise the cost of insurance for good drivers and reduce the price for bad ones. The company therefore opens an online competition on Kaggle [1], a popular data science challenge platform with the goal of improving its prediction algorithm. Participants were challenged to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. A more accurate prediction will allow the company to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers.

In this report, we describe our approaches for tackling the classification problem. The essential part of our solution is to analyze the dataset, eliminate the irrelevant features by comparing their correlations with labels. Finally we apply an ensemble model combined of gradient boosting tree algorithms with cross-validation to produce prediction result.

This report is organized as followed: section §2 provide description on the dataset, section §3 describes our feature engineering method, section §4 gives explanation on how we choose Gradient Boosting Tree algorithm and detailed implementation and §5 is the concluding remarks.

All of the work, including data analysis, data visualization, training classification model was done using Python and iPython notebook. Training process, in particular, was done on a remote machine provided by Google Cloud Platform[1] in order to obtain quickest results possible.

[*] Paris-Saclay University & UEVE (tuan-binh.nguyen@ens.univ-evry.fr)

[1] https://cloud.google.com/

2 Problem settings & Dataset

The classification problem posed by Porto Seguro is a binary supervised learning task. In the context of statistical learning theory [2], this can be defined as a problem of inferring a labeled training dataset (X, Y) where $X \in \mathbf{R}^{n \times p}$ and $Y \in \mathbf{R}^n$ are matrix of features and vector of labels respectively, with $y_i \in \{0, 1\}$ is one element of Y. After the learning model is fitted on training set, it is used on the test set to predict labels. To test performance of the learning model, the predicted labels are often compared with true label using some particular benchmark, often termed *loss function*.

The organizers provided a collection of personalized data related to car driver, separated to `train.csv` and `test.csv` as train dataset and test dataset respectively. The number of observations n and is 595212, number of features p is 59 for train dataset. Interestingly, in the test data there are 892816 observations, more than 1.5 times of the train data. From the author's experience, the opposite is usually observed: train data is much larger than test data.

The benchmark used for evaluation is Normalized Gini Coefficient [1]. Details on this benchmark can be found in Appendix section.

Some important observations on the Porto Seguro's datasets:

- Features are encoded under generic names (for example `ps_car_01`, `ps_reg_02`) that give no specific indication on the meaning of those features. The organizers have done this on purpose to protect their customer's privacy. However, doing feature engineering with this dataset will be harder than usual.
- Features that belong to similar groupings are tagged as such in the feature names (e.g., `_ind`, `_reg`, `_car`, `_calc`).
- Feature names include the post-fix `_bin` to indicate binary features and `_cat` to indicate categorical features.
- The `target` columns signifies whether or not a claim was filed for that policy holder, which correlates with the label $y_i \in \{0, 1\}$ of an observation.
- Values of -1 indicate that the feature was missing from the observation, or `NaN` value as normally seen in many datasets.

Based on these observation, in the next section we will implement some data visualizations and feature engineering techniques.

3 Data analysis & Feature Engineering

3.1 Data analysis

Knowing the properties of the dataset is extremely important step. We observe there exists imbalance in the number of target label in train dataset, with just 21694 number of 1s, and 573518 number of 0s label (4% and 96% respectively). Moreover, the data exhibits a large number of missing values, with a total of 846458 NaN cells in train data and 1270295 NaN cells in test data. Therefore, eliminating null values before doing model fitting is impossible.

Continuous variables are examined. Figure 1 shows the correlation matrix of continuous variables (variables which are of the type float64). Most variables shows no correlation with each others, besides the triplet {ps_reg_01, ps_reg_02, ps_reg_03} and {ps_car_12, ps_car_13, ps_car_15}. In our final model only 3 of the variables in these lists are kept.

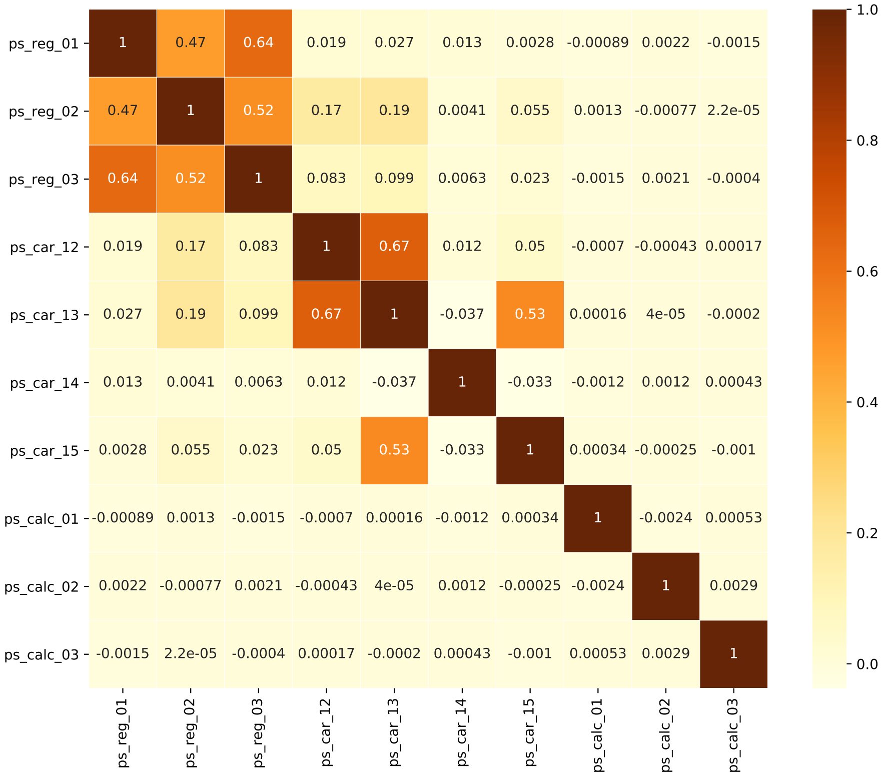

Figure 1: Correlation matrix of continuous features (Pearson method)

Reapplying this technique with integer variable (type int32) results in Figure 2, which shows that variable ps_ind_10_ind to ps_ind_14_ind are highly correlated.

When counting the number of 0s and 1s in binary variables, another interesting finding is revealed: binary features from ps_ind_10_bin to ps_ind_13_bin have almost no 1 value. This strongly indicates their small impact on predicting value of target labels, and indeed is confirmed below.

The most important part in analyzing variables might be investigating which variables (features) have the most explaining power. Details of the method is well-documented in [3]. Essentially, train data is fitted with random forest algorithm using Python library scikit-learn and its Python class RandomForestClassifier. After the model is fitted, its feature_importances is called to return list of variables with their importance score. Utilizing this method result in Figure 4.

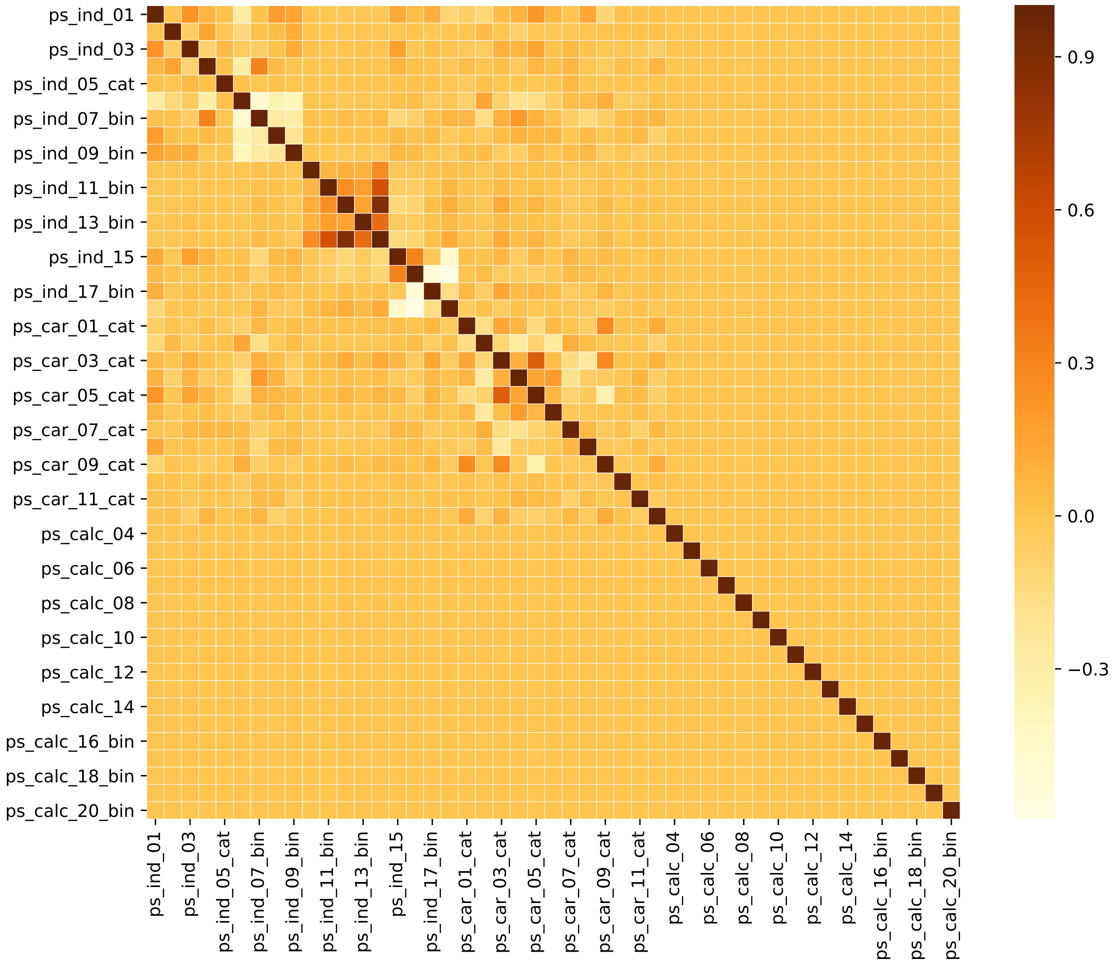

Figure 2: Correlation matrix of integer features (Pearson method)

3.2 Feature engineering

3.2.1 Eliminating unimportant features

The size of train data is big, hence it is crucial to reduce the dimension of the dataset. Based on analysis in Figure 4, top 35 variables which have highest feature important score are kept. This decreases the number of variables by 22.

3.2.2 Dealing with missing values

As mentioned in Data Analysis section, it is impossible to exclude observations that have missing variables (NaN value). One could possibly either set NaN to some extreme value (e.g. -1, 10^6) or use the value that represented the property of the particular variable ($max(X_i)$, $median(X_i)$). In this case, we choose to keep the default value -1 to represent NaN.

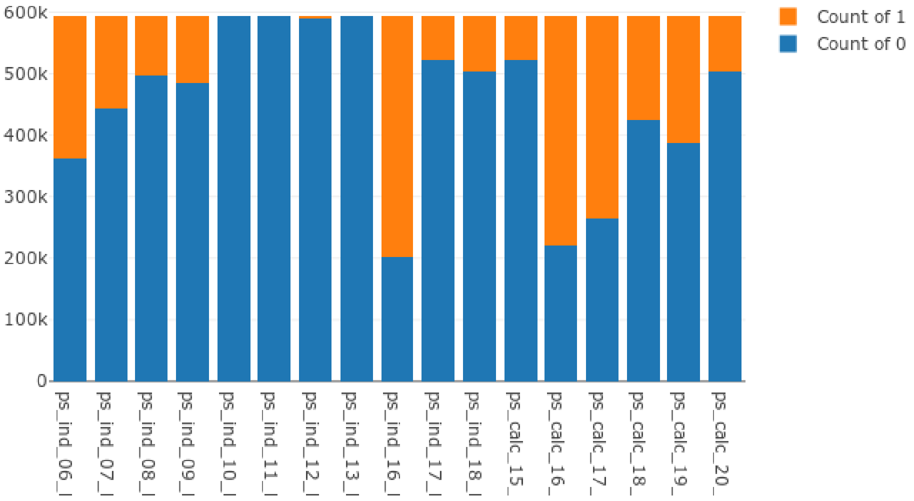

Figure 3: Count of number of 0 and 1 in binary features

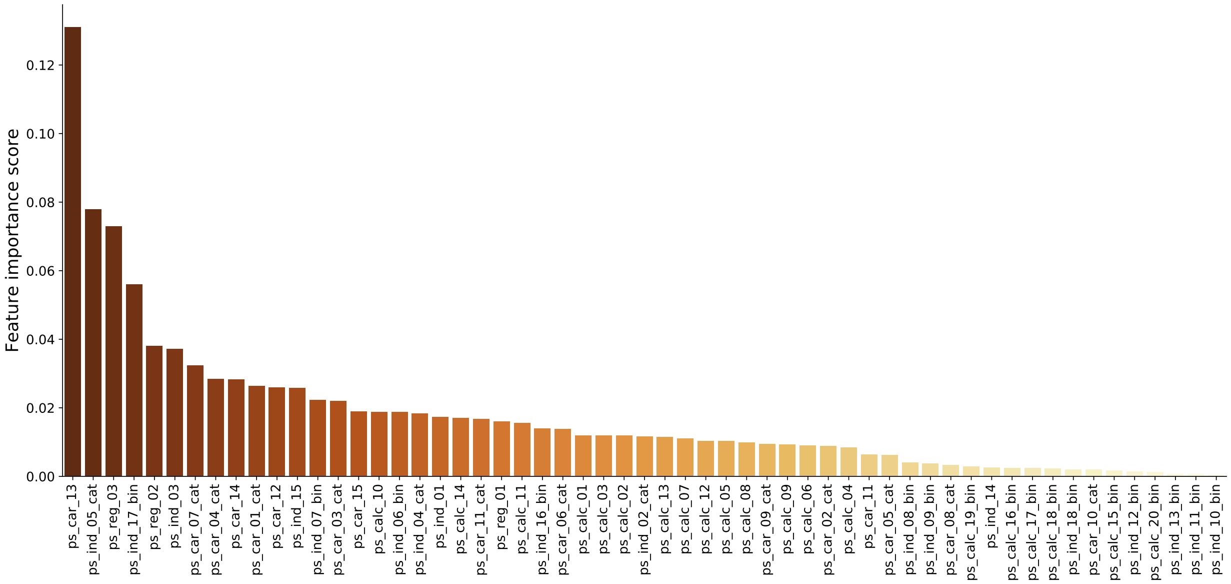

Figure 4: Variable important ranking using Random Forest Classifier

3.2.3 Likelihood encoding

The central idea behind likelihood (or target) encoding technique is to estimate likelihood target variable given a particular value of a categorical variable. It is summarized in the equation:

$$Pr(y = 1 \mid x = \texttt{catValue}) \quad = \frac{\sum \mathbb{1}_{y=1}(x_i = \texttt{catValue})}{n_{\text{observation}}} \tag{1}$$

We create a corresponding Python code for target encoding all categorical variables. The pseudocode is described in Algorithm 1.

4 Model selection & Implementation

4.1 Gradient Boosting Tree

Gradient Boosting Tree (GBT) is our selected method for this classification problem. GBT is robust in the sense that it does not require much data preparation (like normalization of the data) besides encoding all variables to numeric value. A more important reasons is our 2 GBT algorithms(XGBoost[2] and LightGBM[3]) are extremely fast. In a competition where time is limited (2 months), the power of quick prototyping ideas and implementations is a key factor. Gradient Boosting Tree is also popular in data science competitions because of this reason.

4.2 Implementation

We implement a 5-fold cross-validation process for fitting XGBoost and LightGBM tree. The prediction data from these 2 models then was fitted by a Logistic Classifier acted as the ensembler to produce final prediction.

Algorithm 2 in Appendix area is the pseudocode of final ensemble model. Running this model on a Google Cloud cluster with 16 CPU cores and 12GB RAM takes over 2 hours. The task that costs most in term of running time and memory is target encoding (Algorithm 1), mainly because our code is not able to enable parallel processing for the computation and consequently, it utilizes only 1 CPU core out of 16 available.

5 Result & Conclusion

Combining target encoding categorical feature and an ensemble of 2 Gradient Boosting Classification Tree algorithms (XGBoost and LightGBM), we have successfully scores 0.283 on Kaggle's Public Leader board and 0.288 on Private Leaderboard. This result keeps us in top 35% on the final leaderboard of the competition [4].

[2] https://github.com/dmlc/xgboost

[3] https://github.com/Microsoft/LightGBM

[4] https://www.kaggle.com/btnguyen

An interesting fact is that Kaggle offered participant to choose their 2 results as a final submission for the Private Leaderboard. In our case, the 2 final selections, which uses 5-fold CV, are actually not the best ones in final leaderboard. The 10-fold CV ensemble model that performed worse on Public leaderboard has highest final leaderboard score.

Extra: We made a small tutorial for Master 2 DSSAF's classmates, available on our website [4].

Acknowledgments

We would like to express our gratitude to Professor Agathe Guilloux, who provides us a wonderful opportunity to deal with real world data science task. We have successfully learned and applied plenty of good data science and machine learning practices to solve the challenge.

References

[1] Porto Seguro. Porto seguro's safe driver prediction. `https://www.kaggle.com/c/porto-seguro-safe-driver-prediction`, 2017.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer, Feb 2009.

[3] Scikit learn library documentation. Feature importances with forests of trees. `http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html`.

[4] Tuan-Binh Nguyen. Tutorial on using xgboost. `https://tbng.github.io/2017/11/09/xgboost-example.html`, 2017.

Appendix

A Algorithm

Algorithm 1 Target encoding categorical variables

```
Input: categoricalArray, labelArray, $k$, $l$
data ← join (categoricalArray, labelArray)
targetEncodedArray ← Empty Array
categoricalValue ← unique values of categoricalArray
split data into $k$ parts
for each fold[i] in $k$ folds do
    valueEncoded ← 0
    fold[-i] ← join remaining $k - 1$ folds except $i$
    split fold[-i] into $l$ parts
    for each fold[j] in new $l$ parts do
        for each value in categoricalValue do
            valueEncoded ← valueEncoded + $\frac{Pr(label \mid value)}{l}$ using (1)
        end for
    end for
    valueEncodedFold[i] ← join all valueEncoded
end for
targetEncodedArray ← join $k$ parts of valueEncodedFold[i]
return targetEncodedArray
```

Algorithm 2 Ensemble classifier (XGBoost + LightGBM + Logistic Regression)

```
Input: trainData, testData, $k$
objective ← Normalized Gini Index
split trainData, testData to $k$ folds
for each fold[i] in $k$ folds do
    run Algorithm 1
    fit XGBoost on fold[-i] using fold[i] to maximize objective
    fit lightGBM on fold[-i] using fold[i] to maximize objective
    predictXGB ← XGBoost(testData)
    predictLGBM ← lightGBM(testData)
end for
predictXGB ← $\frac{\texttt{predictXGB}}{k}$
predictLGBM ← $\frac{\texttt{predictLGBM}}{k}$
ensemblePrediction ← logisticRegression(predictXGB, predictLGBM)
return ensemblePrediction
```