Porto Seguro's Safe Driver Prediction

Jimit Shah[1]*, Ritesh Tawde[2]*

Abstract
The purpose of this competition is to find out the probability with which a driver will claim insurance in the next year. This information is used to decide how much the driver will need to pay for insurance. Therefore, safe drivers would pay lesser for insurance relatively than other drivers.

Keywords
classification, xgboost, decision tree

* *Computer Science, School of Informatics and Computing, Indiana University, Bloomington, IN, USA*
[1]**Corresponding author**: jimshah@iu.edu
[2]**Corresponding author**: rtawde@iu.edu

Contents

Introduction

"Nothing ruins the thrill of buying a brand new car more quickly than seeing your new insurance bill. The sting's even more painful when you know you're a good driver. It doesn't seem fair that you have to pay so much if you've been cautious on the road for years. Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, completely agrees. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones. In this competition, challenge is to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. While Porto Seguro has used machine learning for the past 20 years, they're looking to Kaggle's machine learning community to explore new, more powerful methods. A more accurate prediction will allow them to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers."[1]

In the project we are making predictive models for predicting whether a driver will claim insurance next year or not. Initially, we start with feature engineering process through data visualization and exploration techniques. Once we complete feature engineering step, we proceed with model building using various algorithms and techniques like XGBoost, target encoding, Gini co-efficient, AUC evaluation metric, etc.

1. Data Exploration and Visualization

1.1 Data Description

On observing the features and data, following are the conclusions we can draw:

- Features that include the postfix _bin are binary feature
- Features that include the postfix _cat are categorical features
- Features without _bin or _cat postfixes are either ordinal or continuous features
- Values of -1 indicate missing value
- The target column signifies whether a claim was filed for that person or not
- The target column is highly imbalanced

Using the info() method provided by pandas DataFrame, we can observe the following:

- There are 17 binary features and 1 target binary column

- There are 10 interval or continuous features
- There are 14 nominal features and one ID column that is nominal
- There are 16 ordinal features

1.2 Dimensionality Reduction

Figures 1. and 2. are correlation heatmaps for categorical and interval features respectively.

As we can observe from the heatmaps, the correlation between variables in both the cases is low, therefore performing PCA would not yield a good result and can lead to too much information loss. Therefore, we will let our model do the heavy-lifting and we won't perform dimensionality reduction.

1.3 Handling missing data

If the feature is a continuous feature, then we can replace the missing values by feature's mean value and if the feature is a categorical feature, then we can replace the missing values by feature's mode value. Below list shows the percentage of missing values in each feature that contains missing values:

Percentage of missing data in column ps_ind_02_cat is 0.04%
Percentage of missing data in column ps_ind_04_cat is 0.01%
Percentage of missing data in column ps_ind_05_cat is 0.98%
Percentage of missing data in column ps_reg_03 is 18.11%
Percentage of missing data in column ps_car_01_cat is 0.02%
Percentage of missing data in column ps_car_02_cat is 0.0%
Percentage of missing data in column ps_car_03_cat is 69.09%
Percentage of missing data in column ps_car_05_cat is 44.78%
Percentage of missing data in column ps_car_07_cat is 1.93%
Percentage of missing data in column ps_car_09_cat is 0.1%
Percentage of missing data in column ps_car_11 is 0.0%
Percentage of missing data in column ps_car_12 is 0.0%
Percentage of missing data in column ps_car_14 is 7.16%

As we can observe, columns ps_car_03 cat and ps_car_05 cat contains 69.09% and 44.78% missing data respectively, which is very high. Therefore, we have dropped these features altogether for one of the models while trying different models and parameters tuning. For other features, we impute the missing values as discussed earlier.

Since XGBoost handles missing values intrinsically, we have replaced the missing values i.e. "-1" with NaN's for some models.

1.4 Handling imbalanced class distribution

Figure 3. shows the imbalance in the target class distribution. One of the issues with class imbalance is that if the classifier predicts the most common class without much analysis of the data, then still the accuracy will remain very high, which is clearly misleading. Also, one of the other major issues with class imbalance is that the predictive model built using conventional machine learning algorithms with such data would be highly biased and inaccurate.[2] Therefore, it is imperative that this issue be addressed appropriately. There are various techniques to handle class imbalance. Resampling is one of the techniques. Oversampling and undersampling are two simple resampling techniques. Oversampling is simply duplicating random records from minority class. This can lead to overfitting problems. Undersampling is removing random records from majority class, which can cause information loss. We have followed the technique called scale_pos_weight which is not a resampling technique but assigns more cost to records that are incorrectly classified as majority class when actually they belong to minority class. scale_pos_weight's value is passed as a parameter to XGBoost algorithm in python and can be tuned according to the requirements of the dataset.

2. Algorithm and Methodology

As discussed in data exploration and visualization section, we performed data pre-processing and feature engineering. The most important task after feature engineering is to select an algorithm which will best suit this problem and is well established, performance optimized and is widely accepted and implemented. So, we decided to go for **"Extreme Gradient Boosting"** or **XGBoost** algorithm.

2.1 Gradient Boosting[4]

Gradient boosting is one of the powerful techniques for building predictive models. Boosting is a term used to build a strong learner by combining different weak learners.

Gradient boosting builds an ensemble of trees one-by-one, then the predictions of the individuals trees are summed:

$$D(x) = d_{tree1}(x) + d_{tree2}(x) + ... \tag{1}$$

For example, if an ensemble has 2 trees, prediction of that ensemble is:

$$D(x) = d_{tree1}(x) + d_{tree2}(x) \tag{2}$$

The next tree (tree 3) in the ensemble should complement well with the existing trees and minimize the training error of the ensemble. We'll then have:

$$D(x) + d_{tree3}(x) = f(x) \tag{3}$$

Then the residual is calculated as the difference between the target and the prediction of the ensemble as :

$$R(x) = f(x) - D(x) \tag{4}$$

The above process is repeated in ensemble building.

2.2 XGBoost[5]

XGBoost[6] is an implementation of boosting trees by Tianqi Chen (Ph.D. Student at University of Washington). Reasons to choose XGBoost over other algorithms (or libraries) for this task are mainly the salient features that XGBoost offers.

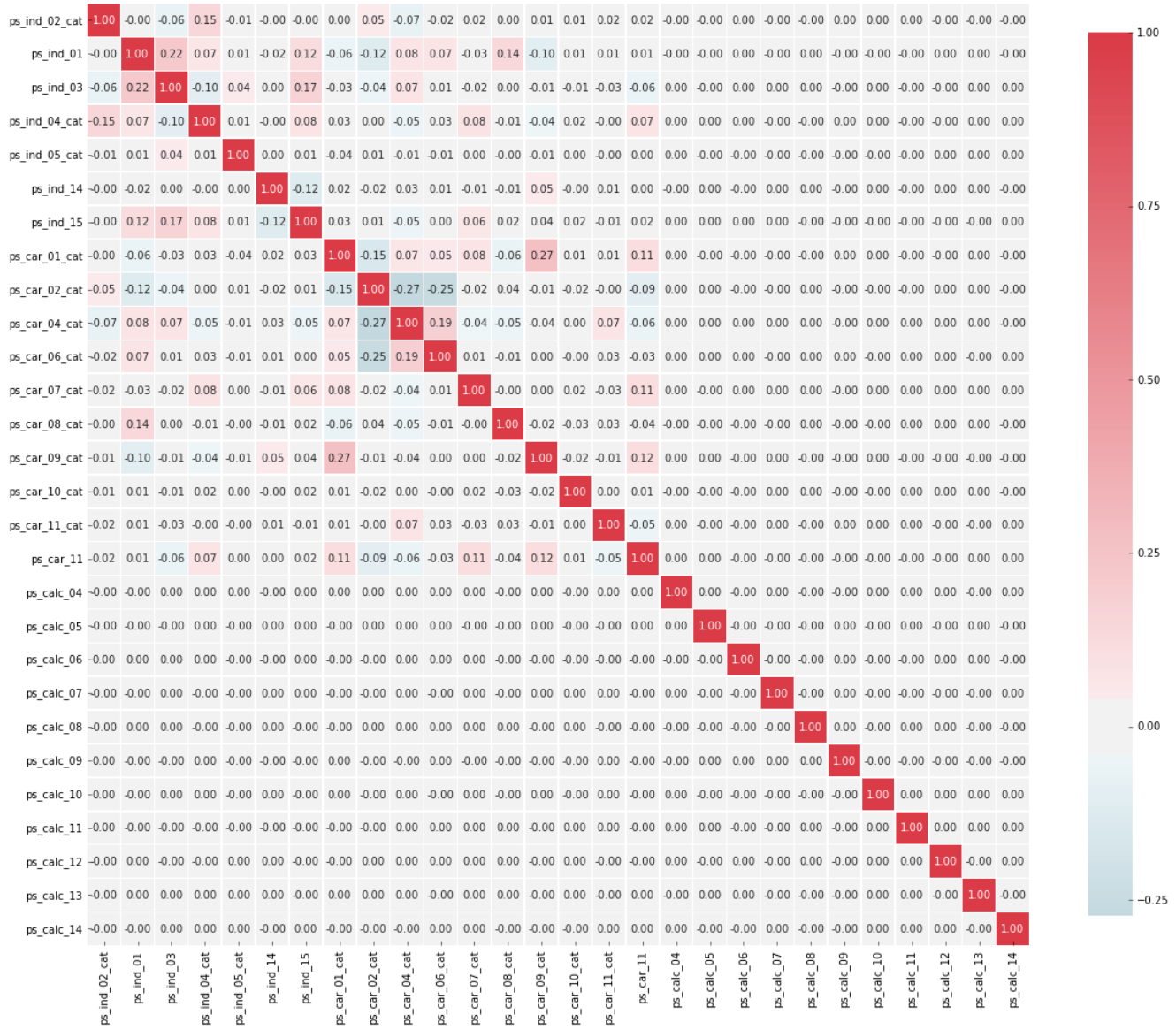

Figure 1. Correlation between Categorical features

2.2.1 XGBoost Features

- Gradient Boosting
- Stochastic Gradient Boosting
- Regularized Gradient Boosting
- Parallelization : of tree construction using multiple CPU cores
- Out-of-Core computing : for very large datasets that don't fit into memory
- Cache Optimization : of data structures and algorithms to make best use of hardware

The algorithm makes efficient use of available resources. Some key features apart from above include:

- Sparse aware : Algorithm is efficient with sparse matrices and automatically handles missing data
- Block structure : For parallel tree construction
- Continued training : To boost already fitted model on new data.

2.3 Target Encoding[7]

As observed from data exploration and visualization step, we have many categorical features within our data set, with a few features having a large number of distinct values. This may pose a challenge for classification task especially because internally, XGBoost models represent all problems as a regression predictive modeling problem that only takes numerical values as input. So, if the data is in a different form, it must be prepared into the expected format.

There are broadly two types of encoding:

Figure 2. Correlation between interval features

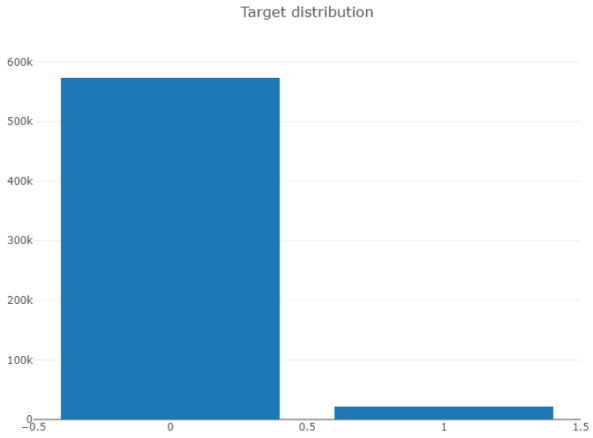

Figure 3. Class distribution

- One hot encoding : converting categorical features to numerical variables based on the distinct feature values.
- Target encoding : statistical method to convert categorical features to numeric format based on its distribution in accordance with the target variable to be predicted.

Target encoding is a preprocessing technique to map individual values of a high-cardinality categorical independent attribute to an estimate of probability or the expected value of the dependent attribute. Broadly there are three types of target encoding, but as our task is to focus on binary classification, we've considered target encoding with respect to binary target.

2.3.1 Binary target encoding

Let X represents high-cardinality categorical attribute and X_i its individual values, Y being the target attribute, and S_i representing an estimate of the probability.
If $Y = 1$ given that $X = X_i$, then

$$X_i \to S_i \cong P(Y|X = X_i) \tag{5}$$

This gives us the probability which is a desirable feature for many algorithms.

Now, it may happen that for some of the values X_i of X, there may be small samples n_i and hence the estimate of the probability given by equation above would be highly unreliable.

To handle this scenario, S_i is calculated as the combination of prior probability and posterior probability of Y given $X =$

X_i. The two probabilities are then weighted as a function of sample size given by :

$$S_i = \lambda(n_i)\frac{n_{iY}}{n_i} + (1-\lambda(n_i))\frac{n_Y}{n_{TR}} \tag{6}$$

where, n_Y : total number of cases such that $Y = 1$;
$\lambda(n_i)$: monotonically increasing function on n_i bounded between 0 and 1;
λ : smoothing/weighting factor;
Typically $\lambda(n)$ is chosen as a parametric function given by :

$$\lambda(n) = \frac{1}{1+e^{-\frac{(n-k)}{f}}} \tag{7}$$

which is a s-shaped function that assumes value 0.5 for $n = k$. Parameter f provides control over the function slope and k denotes the minimum number of samples of leaf for the estimate based on the sample in the cell.

2.4 Bayesian optimization[9]

Bayesian optimization is a technique to choose optimal hyper-parameters for a machine learning model. Bayesian optimization belongs to a class of optimization techniques called sequential model-based optimization ($SMBO$) algorithms. Given parameters x and loss function f for a model, the algorithm is roughly as follows:

1. Using previously evaluated parameters $x_{1:n}$, compute a posterior expectation of what the loss f looks like.
2. Sample the loss f for new parameters x_new, that maximizes some utility of the expectation of f. The utility specifies which regions of the domain of f are optimal to sample from.

For computing a posterior expectation, the likelihood model for samples from f considered is normal likelihood with noise and for prior probability distribution model on f we assume that the loss function f is described by a *Gaussian Process* (GP).

The utility function that is maximized for finding the next best parameter to sample f next from, is called an acquisition function. The most popular acquisition function used for Bayesian optimization is expected improvement (EI) function.

3. Models

We have built different models for XGBoost and finally chose the one with the higher Public Leader Board(LB) score on kaggle. We have used scikit-learn packages for all our model building and testing.[10]

3.1 Basic XGBoost model

We started by building basic XGBoost model. **'Target'** and **'id'** columns were removed from training and **'id'** from the test dataset.
For XGBoost, following Booster parameters were chosen.

Table 1. Boosting Parameters

objective	binary:logistic
eta	0.02
silent	True
max_depth	5
subsample	0.8
colsample_bytree	0.8
eval_metric	auc

Table 2. System Parameters

num_boosting_round	1000
early_stopping_rounds	100
maximize	True
verbose_eval	50

Apart from the above parameters, there are few more parameters to consider.

3.1.1 Parameters explanation [8]

- Objective : Custom objective function
- eta : Boosting learning rate
- max_depth : Maximum tree depth for base learners
- subsample : Subsample ratio of the training instance
- colsample_bytree : Subsample ratio of columns when constructing each tree
- metric : Evaluation metrics to be watched in cross validation step such as 'auc', 'gini'
- num_boosting_round : Specifies the number of boosting iterations
- early_stopping_rounds : Validation error needs to decrease at least every ¡early_stopping_rounds¿ to continue training. Returns a model with three additional fields : bst.best_score, bst.best_iteration and bst.best_ntree_limit
- maximize : Set to True to maximize the evaluation metric
- verbose_eval : Prints the evaluation metric every verbose_eval boosting stage.

The above parameters for basic model were chosen based on what worked the best for different competitors in Kaggle competition in different scenarios considered.

3.2 Using Stratified K-fold cross validation

Using K-fold cross validation for this task was a quintessential task with such a large data set. We opted for 5-fold cross validation. We then took the average over all folds to get us the final output for the classification task. Stratified K Fold technique is used to provide each fold a with balanced target classes that were contained in the original data set. To our surprise, this technique did not improve the score much.

Also, we tried two different types of evaluation metrics during cross-validation.

- Gini co-efficient : To measure the information gain measure during each cross-validation stage to maximize it.
- AUC : To maximize the area under the curve measure and to minimize the loss.

It was observed that **'auc'** performed better than **'Gini'** although both 'auc' and 'gini' are co related.

3.3 Using Target encoding

As explained in the Algorithms section, we implemented target encoding to make dataset more suitable to be fed to the classifiers. Using target encoding improved score by a margin and improvements were still not significant.

3.4 Using ntree_limit parameter of xgboost's predict function

This parameter limits the number of trees in the prediction. It limits to the best tree limit predicted by the xgboost model and hence gives the best number of trees for prediction.

Using this approach, score improved by a good margin, since previously there was no way to ensure where to stop and how many trees to consider.

3.5 Using Bayesian optimization

We implemented Bayesian optimization technique to find optimal hyper-parameters using scikitlearn's package bayesian_optimization. The range of parameters that the Bayesian optimization algorithm explores to get optimal set of parameters are as follows:

Table 3. Bayesian Optimization

Parameters	Range
max_depth	2 - 12
gamma	0.001 - 10.0
min_child_weight	0-20
max_delta_step	0 - 10
subsample	0.4-1.0
colsample_bytree	0.4-1.0

PS: Although we have implemented this algorithm while model building, we could not evaluate our model on the test data using Bayesian optimization's results due to time constraint.

4. Results

Each time we changed our approach or made corrections, our kaggle leader board score improved. Following table summarizes the private and public scores on kaggle respectively, for this competition :

Table 4. Result Scores

	Private Score	Public Score
Basic Model	0.27218	0.26943
K-fold CV	0.27342	0.26955
Target encoding	0.27318	0.27069
ntree_limit	0.28578	0.28208

Following graph summarizes the results :

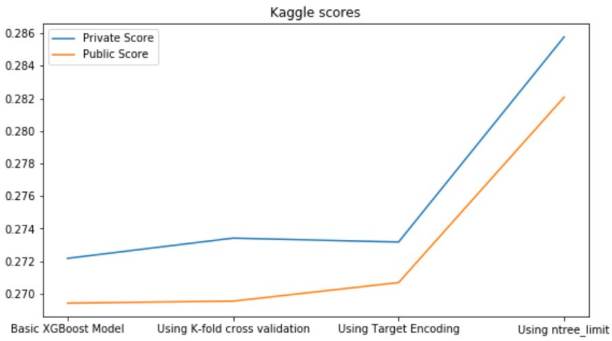

Figure 4. Kaggle scores

5. Summary and Conclusions

To improve our leader board score, we tried different models and fine tuning XGBoost's parameters. The different models we tried were: Basic XGBoost, XGBoost using Stratified K-fold CV, XGBoost using target encoding. We even tried tuning the parameters of the XGBoost algorithm like the ntree_limit to get improved results. Finally, we could obtain an optimal model out of all the models we tried and could reach a personal maximum public leader board score of 0.28208

References

[1] https://www.kaggle.com/c/porto-seguro-safe-driver-prediction

[2] https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/

[3] https://www.utdallas.edu/h̃erve/abdi-WiresCS-mfa-2013.pdf

[4] https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

[5] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[6] XGBoost: A Scalable Tree Boosting System, Tianqi Chen, Carlos Guestrin

[7] A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems
Daniele Micci-Barreca
ClearCommerce Corporation

11500 Metric Blvd.
Austin, TX 78732
daniele@clearcommerce.com

[8] http://xgboost.readthedocs.io/en/latest/python/python_api.html

[9] https://thuijskens.github.io/2016/12/29/bayesian-optimisation/

[10] http://scikit-learn.org/stable/