# Porto Seguro's Safe Driver Prediction

*Sungwon In, Seungmin Baek, Younghun Kim,*

CSCI-B365 Data Mining

kim754@iu.edu, sungin@iu.edu, seubaek@iu.edu

## Abstract

This project is about predicting if Porto Seguro's insurance holders will claim insurance next year with quantification in probabilities. Our team decided to use language R, because it is the most efficient language for data mining. There are two files; test and train data. Because those data contain lots outliers and noises, we cleaned the data first. After cleaning all the useless data, then we used algorithms such as, KNN and XGBoost, which are all supervised machine learning algorithms, to train. After this process we were able to find the final result of the prediction.

## 1. Introduction

### 1.1. Porto Seguro's Data

Porto Seguro held a machine learning competition by providing two data sets, train and test data, on Kaggle. Each data was subdivided and organized with same column name except one, the target column. The train data had the extra target column, so that it can be trained by XGBoost or K-Nearest Neighbors algorithms. Our job was to predict the target's value in test data by using the trained machine. Train and test data both break down into features id, ind, reg, car, calc, cat, and bin. And they are explained below:

**-Id**: Porto Seguro's user identification number.

**-Reg**: Indicates user's region information

**-Car**: Indicates user's car information

**-Calc**: Indicates user's calculated feature with Porto Seguro's own method

**-Cat**: Indicates categorical features

**-Bin**: Indicates binary value

There are 17 binary features, and 14 categorical features. Rest of other features are either continuous or ordinal.

Test data has 892816 rows with 58 columns and train data has 595212 rows with 59 columns because train data has target column.

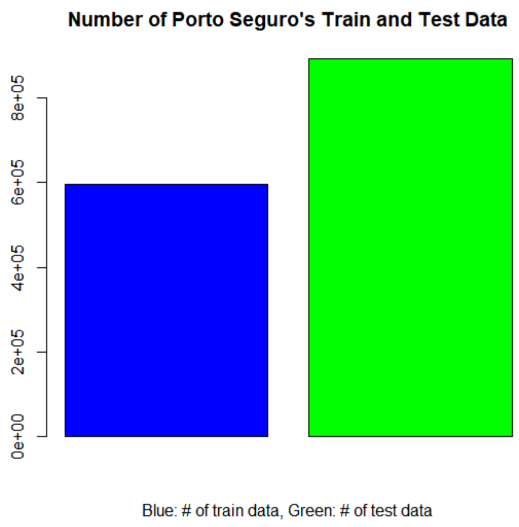


Figure 1: *Number of objects from Porto Seguro's given data*

## 2. Approach and Algorithms

### 2.1. Handling data

To train a machine with the given data, we have to handle the missing values in both data. We used the function **na.omit** to handle the NA values in data. But the number of objects didn't change, thus we found out that there are no NA values in the data.

We figured out '-1' indicates missing values in Porto Seguro's data. By using function unique, we could get the number of missing values. Train data has 470280 missing values out of 595212 objects, which means 79.0105% of the data are missing values. And test data has 706249 missing values out of 892816 objects, which is 79.10353%. Because number of missing values are excessive, we could not ignore the missing values.

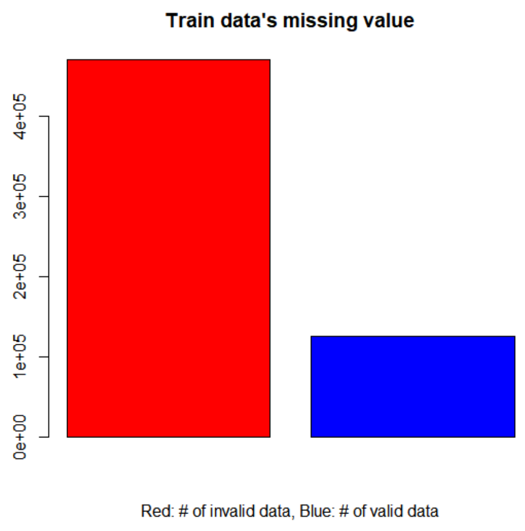


Figure 2: *Number of invalid and valid data in Train data*

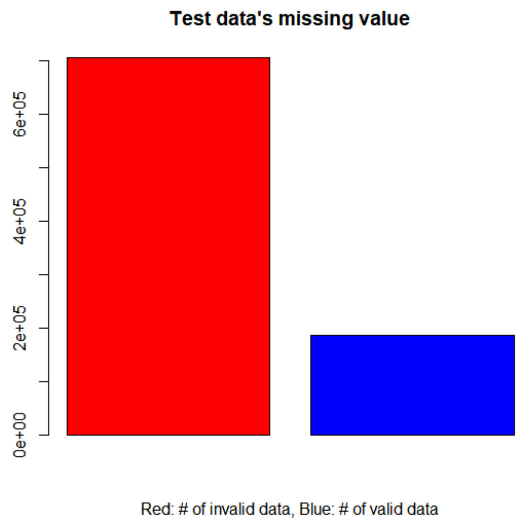


Figure 3: *Number of invalid and valid data in Test data*

We decided to substitute the missing values with the average value of each column. We compute the average value after we eliminate '-1' values, because '-1' will cause wrong average value of each column.

After we decided to substitute missing values to average value of each column, we need to make decision what columns do we need compute. Because if we compute all column's average, then it will increase the run-time a lot. We decided not to compute the binary columns, because binary columns only contain 0 and 1, and didn't have any '-1' values, we didn't compute the average values for binary columns.

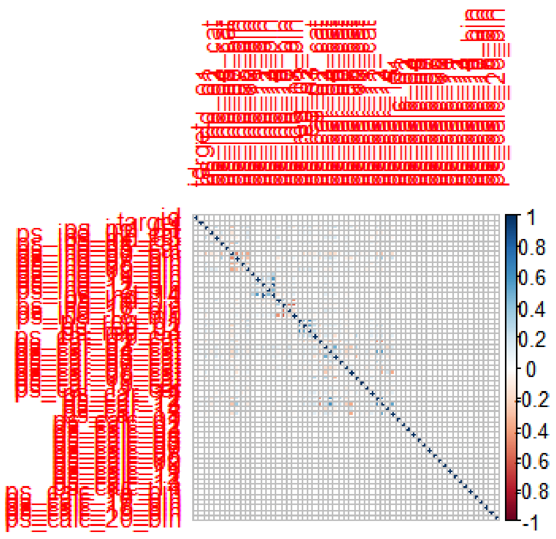


Figure 4: *Train data's correlation plot*

Most controversial problem was computing calculated features' average or not. Because figure 4 explains calculated features are mostly unrelated columns to target and given calculated features are not raw data, we thought we need to eliminate the calculated features.

However, we still want to check it is necessary or not after we see the result. So, we decided to create two different data frame variable; one for holding calculated features, and another for eliminating calculated features.

After we got the average value of each column for two different data, we created the categorical arrays which will contain the name of the columns of each data. By using categorical array, we could insert the average value to missing values

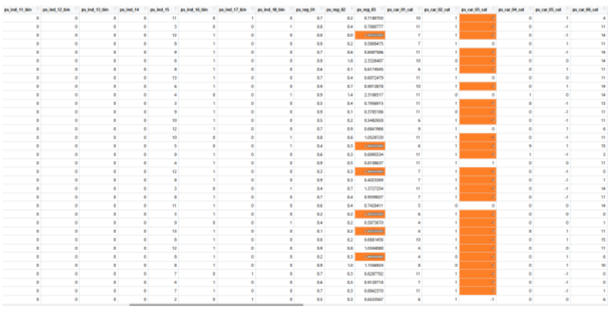
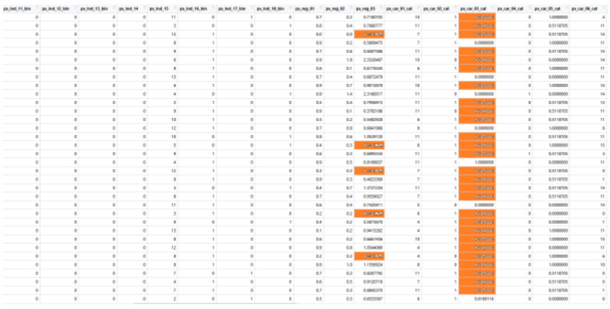


Figure 5: *Before substituted to average in train data*



. Figure 6: *After substituted to average in train data*

## 2.2. XGBoost

XGBoost or Extreme Gradient Boosting is a Greedy Function Approximation using a gradient boosting machine. It is an open-source tool which we used to predict the values for this Porto Seguro project. To understand the XGBoost we need to know about supervised learning and gradient boosting decision tree algorithm.

Supervised learning is a type of machine learning where we have training data with multiple features x_i to predict a target variable y_i.

The model of XGBoost is a tree. The tree ensemble model is a classification and regression trees. A model in supervised learning usually refers to the mathematical structure of how to make prediction.

Boosting is an ensemble technique where new models are added to correct the errors made from existing prior models. This process is repeated until no improvements are made between the models.

Gradient boosting is where new models are created that predict the errors of prior models and then conglomerated together to make the final decision model. The difference between boosting is that it minimizes the loss when adding new models.

XGBoost is a tool motivated by the formal principle of gradient boosted decision tree. The goal of this library is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate library.

We used the XGBoost library after cleaning the data. We first created a test data as a matrix, which didn't have the ID and target features. Before moving on to the next step, we needed to decide on the parameter and the best number of iterations. The parameter for this project was the ETA value or the learning rate, the ETA value is important because when we choose a good eta value it can prevent overfitting. The value is in the range from 0 to 1. In XGBoost it's usually best to keep the ETA value as low as possible. And iteration had to kept large since we wanted to test the data set as much as possible.

## 2.3. K-Nearest Neighbors

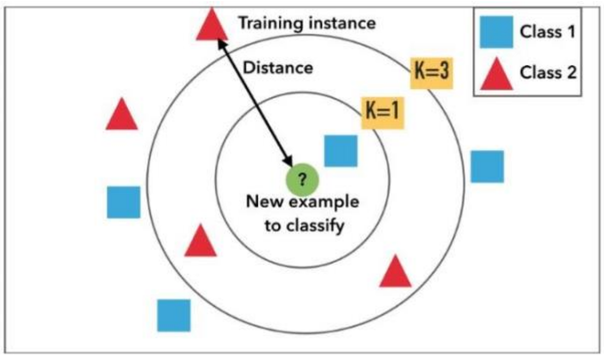


Figure 6: *Visualization of K-Nearest Neighbors Classification.*

KNN is efficient classification algorithm and predict the result with binary value 0, and 1. It calculates the distances of each objects and find the nearest neighbors of each objects. After find the nearest neighbors, it creates the centroids of each neighborhoods, and classify the data. We decide to use this algorithm to see how accurate it is.

We make our KNN code with train data, test data, and k-value as inputs. We choose calculating distance method to Euclidean.

## 3. Experiments

We need to test with different eta value, number of iteration for the XGBoost for both data included calculated features, and data eliminated calculated features. Then we will compare the best result from XGBoost and KNN result.

### 3.1. Including calculated features result

XGBoost with eta= 0.01, iteration=700 gives score 0.27444

XGBoost with eta= 0.01, iteration=800 gives score 0.27528

XGBoost with eta= 0.01, iteration=900 gives score 0.27529

XGBoost with eta= 0.03, iteration=700 gives score 0.27087

XGBoost with eta= 0.03, iteration=800 gives score 0.26940

XGBoost with eta= 0.03, iteration=900 gives score 0.26789

### 3.2. Eliminating calculated features result

XGBoost with eta= 0.01, iteration=700 gives score 0.27488

**XGBoost with eta= 0.01, iteration=800 gives score 0.27650**

XGBoost with eta= 0.01, iteration=900 gives score 0.27632

XGBoost with eta= 0.03, iteration=700 gives score 0.27059

XGBoost with eta= 0.03, iteration=800 gives score 0.27059

XGBoost with eta= 0.03, iteration=900 gives score 0.26919

### 3.3.Deciding XGBoost or KNN

We could get the result from KNN after 15 hours from when we run the code. It's running time is extremely long, and this running time give enough reason why we need to choose XGBoost rather than KNN.

KNN doesn't rate the target with the probabilities. It rates the target with binary value, 0, and 1. So we could get only the error rate from KNN. **We choose to run KNN with k value to 3, and the error rate was 0.002888613** which is highly accurate. However, because of extremely long running time, and format of the result, we decide to use XGBoost rather than KNN.

## 4. Result

From Step 3. Experiment, we found out using XGBoost algorithm gives better result than K-Nearest Neighbors algorithm. The run time of KNN was much longer than XGBoost, and also gave worse result.
And also, as we predicted on Step 2.1: handling data's Figure 4, eliminating calculated features gives better result than including calculated features.
By when experimenting with different eta value and number of iteration for XGBoost, scores increase when eta decrease from 0.03 to 0.01, and gives better result when number of iteration is between 700 and 900. So number of iteration must be proper number and eta value should be small number.
**XGBoost with eta value 0.01, iteration with 800 times, objective to "binary:logistic", booster = "gbtree" gives best score 0.27650.**
And by using **xgb.importance** function, we could rank each features' importance except calculated feature.

Most top 10 important features are "ps_car_13", "ps_reg_03", "ps_ind_05_cat", "ps_ind_03", "ps_ind_15", "ps_ind_17_bin", "ps_reg_02", "ps_reg_01", "ps_car_14", and "ps_car_01_cat".

## 5. Conclusions

We concluded our project with the 0.27650 score from Kaggle, which is near the top 50% percentile of the submission. While working on this project our team faced many challenges, the run time on the program was extremely long, we didn't have a big and fast enough processor to run the code which delayed us a lot during the experimenting process. Moreover, the KNN algorithm that we worked with took more than 14 hours to compute the result which made us difficult to test out different values. Overall, we found out that XGBoost was the go to method to work with data mining, and eta value 0.01 and iteration of 800 times gave us the best score.

## 6. Challenges

There were lots of hurdles we faced during this project. And all of the challenges were caused by huge data size. Train data has 595,212 rows with 59 columns, and test data has 892,816 rows with 58 columns. When we run the XGBoost algorithm to each of the data, it took more than 10 minutes. So it was hard to get all the scores from different eta value and iteration number.

After we score all the results from XGBoost, we faced the real huge problem. It was KNN. We could get the result from KNN after 15 hours of running time.

## 7. References

[1] Bronshtein, Adi. "A Quick Introduction to K-Nearest Neighbors Algorithm." *Medium*, Medium, 11 Apr. 2017, medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7.

[2] Brownlee, Jason. "A Gentle Introduction to XGBoost for Applied Machine Learning." *Machine Learning Mastery*, 17 Aug. 2016, machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.