Applied Machine Learning Fall 2017

Semester Project: Porto Seguro's Safe Driver Prediction

Manoj Joshi[1],Manek Bahl[2],Sohan Udupi Rai[3]

Abstract

Porto Seguro is the third largest insurance company in Brazil. The company offers car insurance, residential, health, life, business. The consortium also offers auto and homeowners, pension, savings bonds and other financial services.

Currently, the car insurance modes are designed in such a way that even if a driver has been cautious for several years, he might end up paying more premium based on the current market scenario. This seems to be a little unfair on good drivers.

In this paper, we are trying to find the probability that a driver will initiate an insurance claim in the next year using the data provided by Porto Seguro. This prediction will allow Porto Seguro insurance to tailor their prices and make auto insurance more accessible to more drivers.

Keywords

XGBoost — Neural Network — Random Forest — Logistic Regression

Computer Science, School of Informatics, Computing and Engineering, Indiana University, Bloomington, IN, USA

Contents

Introduction

This paper describes the overall approach taken to predict the probability of driver initiating the insurance claim in the upcoming year. The test and train dataset was provided by Porto Seguro. We first performed data preprocessing on the dataset before running any classifying algorithms. There are few missing values in the dataset which needs to dealt in a specific way based on whether the column to which they belong is numeric or categorical data. Once the missing values are taken care of, we do feature selection to identify which features in the dataset contribute the most towards predicting the probability of initiating the insurance claim. The training data provided, is highly imbalanced as it has only 3% of 1's. We describe in this paper, the different approaches used in dealing with all the 3 aspects of data preprocessing mentioned above.

After the data preprocessing step, we run our classifying algorithms over the new reduced dataset and calculate the train and test accuracy. The results produced by each algorithm on different datasets obtained using different data preprocessing approach, have been plotted to check which combination produces the best prediction.

1. Data Preprocessing

1.1 Missing Values

Missing data (or missing values) is defined as the data value that is not stored for a variable in the observation of interest. Missing data present various problems. First, the absence of data reduces statistical power. Second, the lost data can cause bias in the estimation of parameters. Third, it can reduce the representativeness of the samples. Fourth, it may complicate the analysis of the study. Each of these distortions may threaten the validity of the trials and can lead to invalid conclusions. So, it is of utmost importance that missing values are being taken care of before we start any analysis on this dataset.

The missing values in this particular dataset are represented by -1. For each column, depending upon the type of data (discrete or continuous), we need to take different action on each of the variable to fix the missing values.

Below are the 3 approaches being taken to deal with missing values:

1. More than 20% missing values: There are 2 variables **ps_car_03_cat** and **ps_car_05_cat** which have approximately 68%and 44% missing values respectively. We have taken 20% as the threshold for missing values, if the percentage of missing values exceeds 20% , we will ignore those particular variables and remove them from any further calculations.
2. Missing values in continuous variables: The missing values in continuous variables will be replaced by the mean. While calculating the mean we need to keep in mind that the mean needs to be calculated for the variable by excluding the missing values -1. So, we replace -1 with NA and then calculate the mean of the variable and then replace NA with the calculated mean.
3. Missing values in discrete variables: For discrete variables, the missing values are replaced by the highest frequency value(mode) from that particular variable.

Once these three approaches are taken, we would have a clean dataset with no missing values and ready to be used over a model.

1.2 Imbalance in classes

In the training dataset provided, there is a total of 595212 training examples out of which only around 3 percent (21694 training examples) belong to the class with target = 1. This large imbalance in the data is a problem since classifiers are more sensitive to detecting the majority class and less sensitive to the minority class. The classification output will be biased, in many cases resulting in always predicting the majority class.

To test this, we created our prediction model over the imbalanced train dataset and used it to predict the classes for the test dataset. We found that the training error was very low, yielding an accuracy of around 99 percent, but the prediction on the test dataset was very poor, and yielded the gini coefficient of 0.142. Therefore, to improve the accuracy of our prediction model, we proceeded to reduce this imbalance in the train dataset.
Under sampling of the majority class :

1. **Using K-means (minibatch Kmeans)**:
   Let "n" be the number of training examples belonging to the class 0.
   Let "m" be the number of training examples belonging to the class 1.
   We performed K-means on the data belonging to the majority class and divided them into "m" clusters. We then replaced the "n" datapoints with the "m" cluster centroids obtained. This resulted in a balanced dataset with the majority class to minority class ratio being 1:1. We used the mini-batch K-means instead of the traditional K-means as the traditional K-means has a very bad runtime on dataset involving a large number of records being clustered into a large number of clusters.

2. **Using Random undersampling**: In this approach, instead of using clustering techniques to undersample the majority class, we choose a random sample of records from the majority class. We tried taking different sample sizes – m, 2m, 3m, 4m, 5m etc... and we found that the best training and test dataset accuracy was obtained when we kept the sample size as "2m", which made the ratio of majority to the minority class to 2:1. We got a decent test accuracy score of 0.242 on the prediction obtained using random forests model generated on the above created dataset.
   Out of the 2 techniques for undersampling mentioned above, we got better results using Random undersampling, taking sample size of 2m for the majority class.

3. **Using Synthetic Minority Oversampling Technique**: In this technique, in order to reduce the imbalance in the two classes, the minority class is oversampled. While a simple over-sampling of the minority class with replacement is the easiest way to achieve it, the SMOTE technique oversamples by creating synthetic examples instead. Synthetic samples are generated in the following way: Take the difference between the feature vector (sample) under consideration and its nearest neighbor. Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general. More general regions are now learned for the minority class samples rather than those being subsumed by the majority class samples around them. The effect is that decision trees generalize better.
   We performed SMOTE in python using the package "SMOTE" under "Imbalanced-learn", part of the "sklearn" library. After performing class balancing using SMOTE we did find significant improvement in prediction accuracy for logistic regression algorithm; however, for the other models, minority over-sampling technique produced better results. The comparison of these results has been done in detail in the "Experiments and Results" section of this paper.

1.3 Feature selection

1. **Using the significance probability values (p value) obtained by using "glm" for binomial classification in R:** We used the glm function in R to perform Logistic regression and obtained significance probability i.e. "p" value for all the features. We found that only 21 predictors have significance probability less than 0.05 and

hence are statistically significant for our classification model.

2. **Using feature importance obtained from Random Forest Classifier:** The RandomForestClassifier in the sklearn package in python creates multiple random decision trees which decrease in mean impurity at each level. The "feature_importance" method of this class assigns "importance" values to each feature. We used this to obtain the importance value for each feature of for our training dataset. There wasn't any sudden decrease in feature importance values until around 40 features. Thus, we considered the top 50 percent of the features ordered in decreasing order of importance.

Out of the 2 techniques mentioned above, we found the best training and test prediction accuracy by reducing features based on the significance probability values obtained using glm for binomial classification.

We followed another approach for feature selection which is based on Frequency and Binary Encoding. Each of the encoding technique is described below:

1. **Frequency Encoding:** For a particular column in the data, we took the frequency of each value in that column and create a new column with the frequency of the value. For instance if a data column has the values [a, b, c, b, a, b], our new column would be [2, 3, 1, 3, 2, 3] since frequency (a) = 2, frequency (b) =3, frequency (c) =1. We append the new column to the existing data frame. We applied the frequency encoding on the original dataset for the columns that ended with "cat". This was done because in dataset description of the contest, it mentioned that features ending with "cat" are categorical.
2. **Binary Encoding:** This idea is derived from one-hot-encoding with a slight modification of taking union of unique values in a particular column in both train and test set. For a particular column in the data, we took the union of values in test and the train set. We found out the maximum among these values. With the maximum value, we found the number of bits required to represent the maximum value. Then we represented each value in its binary form. We split each bits into separate features as 1 or 0. Total number of new features will be the number of bits required to represent the maximum value. This was done for all the features ending with "cat" after performing frequency encoding as described above.

   Lastly, from a kernel in the competition, we found that "calc" features were irrelevant for claims. We dropped the columns that began with "ps_calc". [1]

2. Algorithm and Methodology

2.1 XGBoost

Lets first discuss briefly about Boosting. Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher. This technique is followed for a classification problem while a similar technique is used for regression.

XGBoost (**eXtreme Gradient Boosting**) is an advanced implementation of gradient boosting algorithm. Below are few advantages of XGBoost which prompted us to use XGBoost over some other well-known classification algorithms:

1. Regularization: Standard XGBoost implementation has regularization, hence it can reduce overfitting.
2. Handling Missing values: Although, we have already handled missing values above, XGBoost has an in-built routine to handle missing values. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
3. Built-in Cross-Validation: XGBoost allows us to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

The built in package needs to be tuned according to respective datasets. There are various parameters that can be tuned to get better results. The tuning was done based on the below parameters for our dataset:

1. Eta: Eta is analogous to learning rate in other gradient descent algorithms. The default value of eta is 0.3 but we ran XGBoost for various values to get higher accuracy.
2. Max_depth: It is the maximum depth of the tree. This is used to control the over-fitting problem as higher depth will allow the model to learn relation specific to a particular sample. We ran XGBoost for max_depth 20, 5 and 3 and it turned out that using 20 we overfitted the data which gave us a very bad score. For the best score max_depth value was kept to be 5.
3. Objective: This defines the loss function to be minimized. Since we need the value of probabilities of the person to claim insurance, we would be using binary:logistic.
4. Nrounds: Using nrounds we limit the number of iterations used to reach the global minimum.

2.2 Neural Network

Using the pre-processed dataset, we decided to implement a fully connected Neural Network . The reason for using a

traditional fully connected neural network is that we wanted to feed all the features and let the network assign weights appropriately and learn the important features. The architecture of the neural network consists of multiple hidden layers which depends on the number of features in the input. The first layer of the neural network consists of "N" neurons where N - is the number of features in the input. Different data pre-processing techniques yield different number on input features. The subsequent layers of neural network consist of lesser number of neurons. The intuition behind this is that the network will learn the relationships between various variables since it is a fully connected neural network. Each of the neurons are activated by "ReLu" activation functions. The last layer of the neural network has just two neurons since we have two classes. Each of the neuron has "Softmax" as the activation function since we need probabilities for each class as the output. We scale the input features using "Standard Scaler" in Keras. The reason for scaling the parameters is to avoid "exploding gradient problem". Also, we did not implement a very deep neural network to avoid vanishing gradient problem.
The network architecture for the Neural Network using Random undersampling for class imbalance and feature selection using Logistic Regression in "glm" package in R for binomial classification is shown below:

1. The first layer has 21 neurons which is the equal to the number of features in the input. ReLu is used as the activation function.
2. The second layer has 15 neurons. Again this layer has ReLu as the activation function.
3. The third layer has 10 neurons. Again this layer has ReLu as the activation function.
4. The fourth layer has 5 neurons. Again this layer has ReLu as the activation function.
5. The last layer has 2 neurons. This layer uses softmax as the activation function to obtain the class probabilities.

The features and hyper-parameters of the network are:

1. Adam Optimizer was used. This optimizer uses momentum to converge the stochastic gradient descent faster.
2. Binary Cross Entropy was used as a loss function since it is a binary classification problem.
3. The number of epochs was set to 6000.
4. The batch size was set to 2000.

With this hyperparamter setting, the network gave an accuracy of around **68 percent** on the training data. This was expected since the data size was reduced drastically. Ideally, a neural network requires a lot of data to train.

With this training accuracy, we predicted the target variable for test data set. Upon submission, we got a Gini Coefficient of **0.243**.

Tools used:

1. Jupyter notebook
2. Keras with Tensorflow as the backend.

2.3 Random forests

In Random forests, multiple random decision trees are created using only a small random subset of the features. Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split. Using Random Forests also removes the necessity of pruning trees which would otherwise be required to prevent over-fitting of the model to the training dataset. With default values kept for the parameters like the maximum depth of the tree, maximum number of trees in the forest and maximum depth, we got a very high training accuracy of around 99 percent but very bad prediction on the test data giving us a score of 0.14 on submission. The problem was due to overfitting. Hence we tweaked the parameters of the model to get better predictions on the test data.
The parameters for the model used are:

1. n_estimators = 100. This refers to the number of decision trees in the forest. Increasing this parameter increases the accuracy of prediction as it reduces the impact of over-fitting.
2. max_features = 10. This determines how many features each tree is randomly assigned. The smaller, the less likely to overfit, but too small will start to introduce under fitting.
3. max_depth = 7. This will reduce the complexity of the learned models, lowering over fitting risk.

With the above parameters, the gave an accuracy of **67.2 percent** on the training data.

With this training accuracy, we predicted the target variable for test data set. Upon submission, we got a Gini Coefficient of **0.252**.

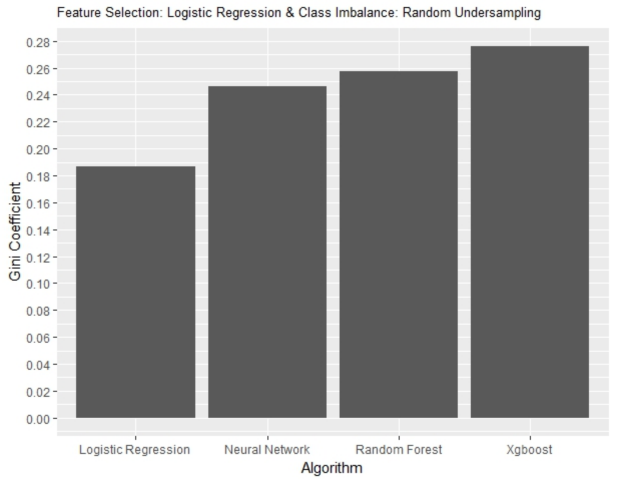
2.4 Logistic regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous numeric values, logistic regression transforms its output using the logistic sigmoid function. The function maps any real value into another value

between 0 and 1. In order to map this to a discrete class, we select a threshold value or tipping point above which we will classify values into class 1 and below which we classify values into class 2. Unlike Linear regression, we cannot use Mean least squared error (MSE) as the cost function. This is because squaring the prediction using sigmoid function, as done in MSE, results in a non-convex function with many local minimums, in which case, gradient descent may not find the optimal global minimum. Thus, in order to ensure the cost function is convex (and therefore ensure convergence to the global minimum), the cost function is transformed using the logarithm of the sigmoid function.
We performed logistic regression in R by using the built-in function "glm" for binomial classification.

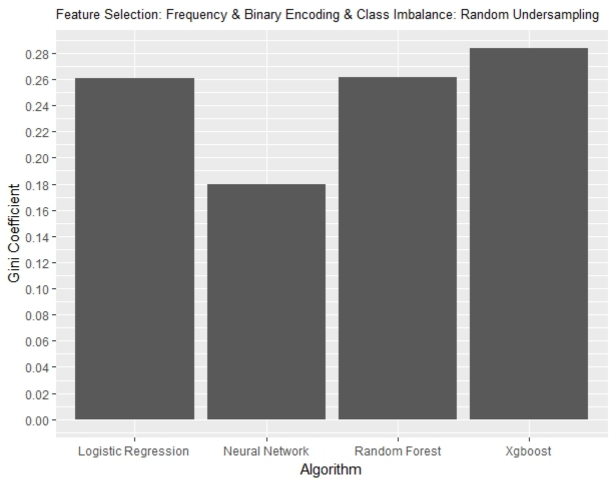
3. Experiments and Results

Firstly, we ran each of the 4 algorithms using Random undersampling for class imbalance and feature selection using Logistic Regression in "glm" package in R for binomial classification. The result of each algorithm for this data processing methodology on the test data set is shown below:



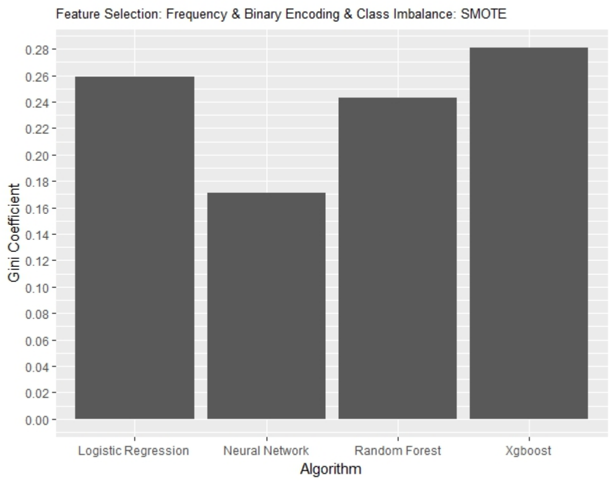
For this method, Xgboost outperformed all the other algorithms with a Gini Coefficient of around 0.27 and Logistic Regression was the worst with a Gini Coefficient around 0.19.

Secondly, we ran each of the 4 algorithms using Random undersampling for class imbalance and feature selection using Frequency and Binary Encoding. The result of each algorithm for this data processing methodology on the test data set is shown below:



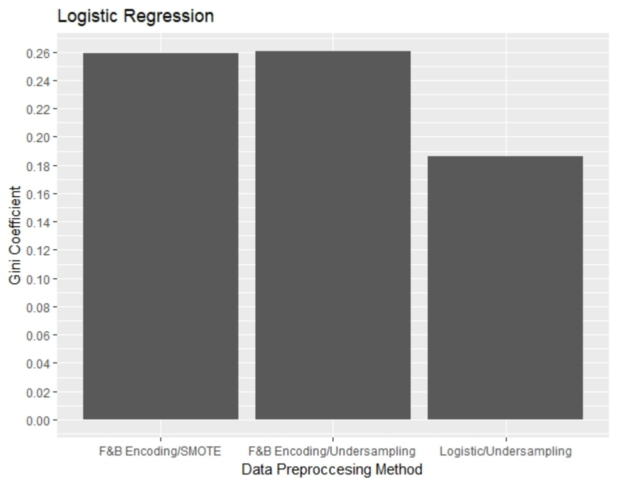
For this method, Xgboost outperformed all the other algorithms with a Gini Coefficient of around 0.28 and Neural Network was the worst with a Gini Coefficient around 0.18. Surprisingly, Logistic Regression had a big jump in its Gini Coefficient compared to the previous approach.

Lastly, we ran each of the 4 algorithms using SMOTE method for class imbalance and feature selection using Frequency and Binary encoding. The result of each algorithm for this data processing methodology on the test data set is shown below:

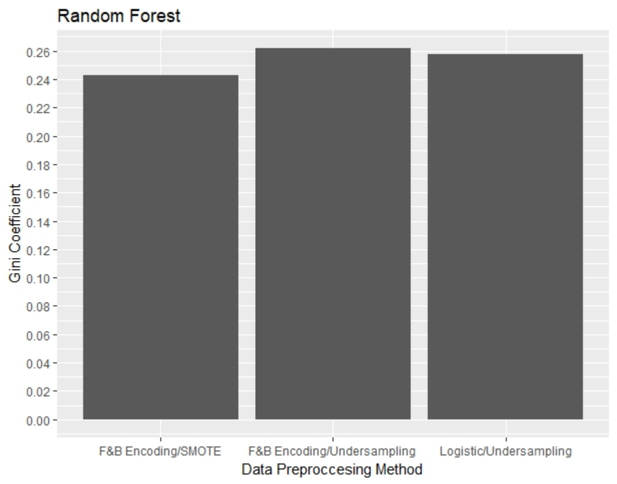


For this method, Xgboost outperformed all the other algorithms with a Gini Coefficient around 0.28 and Neural Network was the worst with a Gini Coefficient around 0.17.

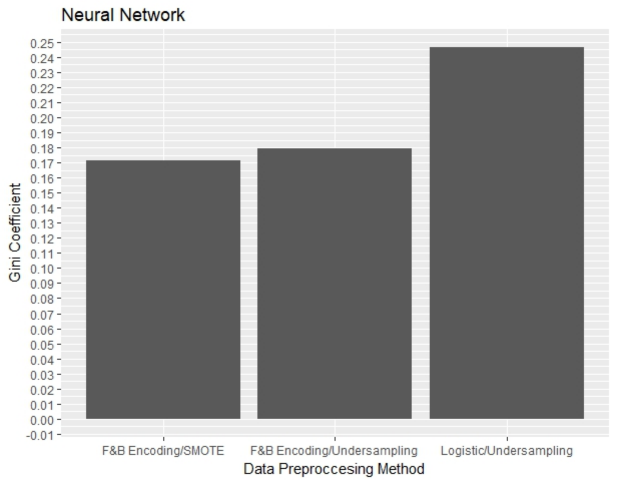
We also did 4 graphs - one for each algorithm. Each of the graphs contained 3 results based on the 3 different combinations of data processing as discussed above. The intuition behind these graphs is to understand how each data processing technique works on an algorithm on the test data set.
The graph below shows results of Logistic Regression



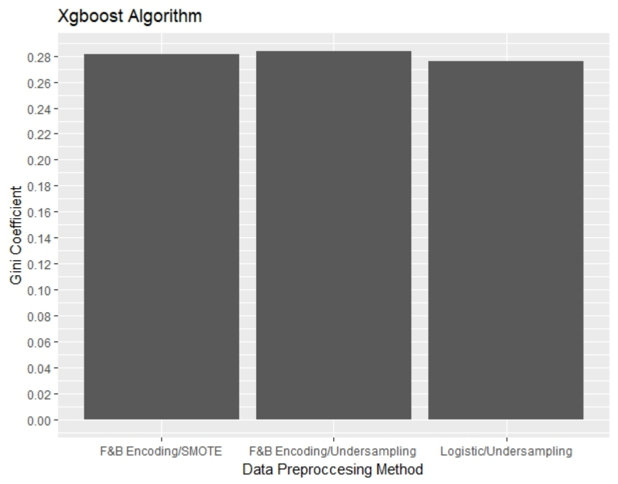
The graph below shows results of Random Forest



The graph below shows results of Neural Network



The graph below shows results of Xgboost



4. Summary and Conclusions

Looking at the graphs for the all data pre-processing approaches, Random undersampling for class imbalance and feature selection using Frequency and Binary Encoding gave us the best results. Xgboost gave the best Gini Coefficient of 0.28359 for this approach.

For Logistic Regression, Random Forest and Xgboost, Random undersampling for class imbalance and feature selection using Frequency and Binary Encoding gave us the best result with Gini Coefficients being 0.26049, 0.26175, 0.28359 respectively.
For Neural Network, Random undersampling for class imbalance and feature selection using Logistic Regression for binary classification gave the best result with Gini Coefficient being 0.2465.

To conclude, Random undersampling for class imbalance and feature selection using Frequency and Binary Encoding seems to be a better approach for a training dataset with many categorical features and high class imbalance. Among all the algorithms, Xgboost was the best for each data pre-processing method.

Acknowledgments

We would like to thank Kaggle community for sharing many kernels during the contest to help us better understand the data. We would also like to thank Dr. Jason Brownlee for sharing a blog post on Deep Learning for Binary Classification using Keras.

References

[1] The Exploratory Data Analysis kernel of "Heads or Tails". "https://www.kaggle.com/headsortails/steering-wheel-of-fortune-porto-seguro-eda"