Porto Serguro Safe Driver Prediction

Contributors : Charles Frank, Ryan Hartford.
Project Lead : Samuel Madden[1]*

Abstract
Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones. https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/overview
Goal:
Using AWS, created S3 bucket, IAM roles for all team members and a remote Jupiter Notebook, clean / organize disjointed dataset and deployed an SVM (Support-Vector Machine), a Naive Bayes Classifier and a Decision Tree Classifier on the dataset. All code written in Python

Keywords
Data Mining — Naive Bayes — LightGBM

[1] *Computer Science, School of Informatics , Computing and Engineering, Indiana University, Bloomington, IN, USA*

Contents

1. Problem and Data Description

Porto Seguro is one of the largest auto and home insurance companies in Brazil. Because of the positive correlation between the number of annual predicted claims and cost for the ideal diver, Porto Seguro is looking to decrease innaccuracies for anticipated claims. Using a predictive model, Porto Seguro hopes to better serve their customers by more accurately pricing the car insurance they provide. The given data is split into a train and test csv file. Missing values are represented with a -1. The data is ordinal, binary, nominal, or interval. There are 59 variables and $595,212$ rows. Excluding *id* and *target*, which are not an input type, there are 57 features describing the attributes of each record. Viewing the head of the first 100 rows shows a large number of clients having not made a claim. This is indicated by a 0 classification in the target column. To better understand the distribution as a whole, a bar graph and pie chart are created.

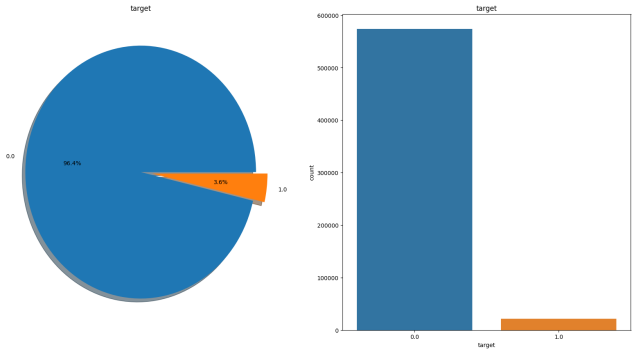

Figure 1. Target Ratio

An imbalance becomes readily apparent, showing that a very few amount of people have a claim filed. Lastly, the data contains $11,671,510$ zero instances. Though this is not enough to be categorized as a sparse dataset, it is worth mentioning.

2. Data Preprocessing & Exploratory Data Analysis

2.1 Handling Missing Values

The provided training set has dimensions of $595,212$ X 59 with a mixture of columns containing int64 and float64 numbers. Upon initial evaluation of the set, no missing values exist from .info() output. This is because NaN values are instead given the value of -1. After replacing -1 values with NaN, the missing values become apparent. Notably the columns *ps_reg_03* missing $107,772$ values, *ps_car_03_cat* missing $411,231$ values, and *ps_car_05_cat* missing $266,551$. Using the package ***missingno*** to visualize the data, columns containing missing values are quickly identified.

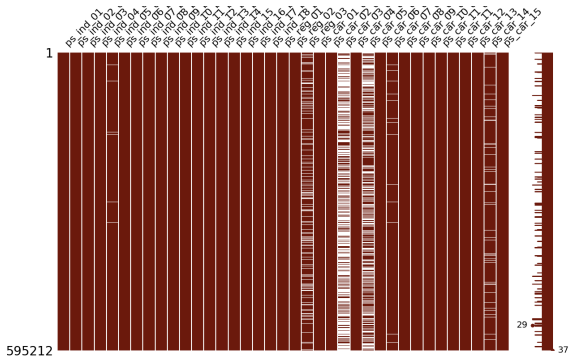

Figure 2. MissingNo representaion

Columns *ps_reg_03*, *ps_car_03_cat*, *ps_car_05_cat* are dropped due to high number of missing values. The remaining missing values are replaced with the mean of the respective column.

After reassessing the data, no NaN values are identified and the dataset is complete.

2.2 Highlighting Dependent Features

To better understand the way the large set of attributes interact with one another, a heat map of correlation is created. In the given scale, as the hue approaches red, a positive correlation is identified while deep blues show a negative correlation. The color of teal indicates neither positive or negative correlation upon comparison.

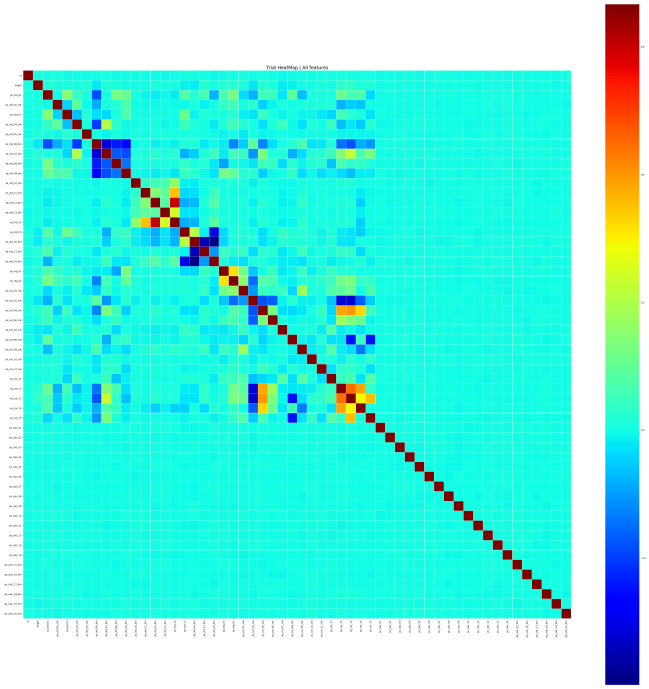

Figure 3. HeatMap of Feature Correlations

The above chart condenses a large attribute set to make feature correlation more apparent at a glance. As mentioned, all teal tiles indicate no correlation. Instead, more attention should be given to the tiles and features that do show some either a positive or negative relationship.

2.3 Scaling using Standardization

The standardization approach of scaling is used on this dataset. Normalization(min-max scaling) is declined in this instance due to its susceptibility to outliers. Using standardization, the feature columns are centered at mean 0 with standard deviation 1 meaning a Gaussian distribution. This scaling will retain information about outliers while also making the classifiers less sensitive to large deviations from the mean. The equation for standardization is as follows:

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

Where μ_x is the sample mean of a particular feature column and σ_x is the corresponding standard deviation.

Due to this transformation, the X_test will also be transformed but not fitted using the scaler.

A new heat-map is created using the standardized data with no deviation from the original output.

2.4 Baseline Performance

Two linear algorithms were chosen to establish a baseline performance after the cleaning and dropping of NaN columns and columns of little impact. The Logistic Regression model created using a subset of the training data consistently returned an accuracy score of 0.9645 when tested using a train/test split. The Linear Discriminant Analysis created from the same strategy bottomed out congruent to the Logistic model at 0.9643. Generating a first impression, linear algorithms provided accuracy scores consistently over 95%.

3. Algorithm and Methodology

Two popular algorithms, Naive Bayes and LightGBM are chosen to effectively build predictive models for the Porto Serguro dataset. The two models returning the lowest error will be presented as an ensemble and final classifier. Originally, an implementation of SVM using either linear or RBF strategies was to be implemented as well. Unfortunately, because of the quadratic nature of the runtime, lack of memory, and processing power an SVM was not realized.

3.1 Naive Bayes Algorithm

The Naive Bayes Algorithm is an Eagar, Supervised learning algorithm based on the Bayes Rule. The basic idea behind the Naive Bayes Algorithm is to predict an unknown class for a new record based on that records known observations. The naivety in the Naive Bayes Algorithm comes from assuming the likelihood of an observation is independent from the other known observations. For this paper, the Scikit-learn implementation of the Naive Bayes Algorithm was used. In

this implementation there are three assumptions for the distribution of the data: Gaussian, Multinomial, and Bernoulli. Regardless of which assumption is used, the algorithm is more or less the same in practice. For all the class values in the training data, the probability is calculated as $P(X_1, X_2, ..., X_n|Y_j = P(X_1|Y_j) * P(X_2|Y_j) * ... * P(X_n|Y_j)$. After all the classes and attributes in the training data is understood, new records are given class Y in $Y = arg\max_{Y}[P(Y)\prod_{i=1}^{n} P(X_i|Y)]$.

3.2 Decision Tree Classifiers

LightGBM Classifier This is a fast, distributed, high performance gradient boosting framework based on decision tree learning algorithms. This classifier is used to for ranking, classifying and general machine learning. LightGBM performs similar to XGBoost. The two differ in speed, and LightGBM builds trees vertically (leaf-wise) while XGBoost builds level-wise. LightGBM is slightly faster than XGBoost and pulls slightly higher accuracy. LightGBM grows on leaves with maximum delta loss and experiences sensitivity due to over-fitting on smaller data sets. This can be found in the package ***lightgbm*** in python.

Generic Decision Tree A decision tree is a standard tree that uses branches to classify a dataset. This algorithms follow a greedy search style through all possible branches and implemented top-down, meaning implementation begins at a point and works its way to the bottom.

Information gain Decision trees gain information with a decrease of entropy after splitting attributes of a dataset. Decision trees first-and-foremost build and find attributes that return the most homogeneous branches.

Entropy is used in decision trees to define the structure. Calculating the frequency of two attributes is as follows:

$$E(X,Y) = \sum_{c \in X} P(c)E(c)$$

Algorithm The algorithm begins with calculating entropy. This information is utilized to create decision nodes. The dataset is divided by the number of branches and the process is repeated. We always ensure the leaf node of a branch has entropy of 0, if this term is not met, then we will perform further splitting. Algorithm is run recursively on all non-leaf branches until all data is classified.

To run this algorithm the cross-folded data will be taken as input including a gini index, and receive the scores.

4. Experiments and Results

We have conducted some preliminary experiments using the Naive Bayes Algorithm and 2 assumptions for the distribution of the data. The assumptions tested were Gaussian and Bernoulli; one experiment for each assumption. In each trial, for each experiment, denote the best column to drop from the data by creating a model with all the other present columns, defining best as lowest model error rate. The lowest error rate achieved was 3.645% and all three experiments reached this error rate. All required different columns to be dropped to achieve this outcome. The Gaussian assumption, seen in table 1, required 14 columns to be dropped to achieve an error rate of 3.645%. The Bernoulli assumption, seen in table 2, required 1 column to be dropped to achieve an error rate of 3.645% but was able to drop an additional two columns while maintaining an error rate of 3.645%.

Column	Error Rate Without Column
ps_car_09_cat	3.618%

Table 1. Output for Gaussian Assumption. No other column drops had a positive effect on error rate

Column	Error Rate Without Column
ps_car_07_cat	3.668%
ps_car_02_cat	3.634%
ps_ind_04_cat	3.624%
ps_ind_14	3.619%.
ps_ind_17_bin	3.618%

Table 2. Output for Bernoulli Assumption

Table 1: LightGBM

	Param Set 0	Param Set 1	Param Set 2	Param Set 3	Param Set 4
Error Rate	3.36	3.3	3.63	3.63	3.62
Task	train	train	train	train	train
Boosting Type	gbdt	gbdt	gbdt	gbdt	gbdt
Objective	binary	binary	binary	binary	binary
Metric	binary_logloss	binary_logloss	binary_logloss	binary_logloss	binary_logloss
Max_dept	10	10	10	10	10
Num_leaves	15	20	10	15	15
Learning_rate	0.0785	0.0785	0.0785	0.1	0.785
Feature_fraction	0.6	0.6	0.6	0.6	0.8
Bagging_fraction	0.5	0.5	0.5	0.5	0.5
bagging_freq	2	2	2	2	2
verbose	1	1	1	1	1

Figure 4. LightGBM

5. Summary and Conclusions

When viewing the correlation of values represented by the heat-map mention in section 2.2, the group concluded that dropping the columns of *ps_calc_#* and *ps_calc_#_bin* would positively affect the error rate. However, upon feeding a trimmed version of the dataset for the model to fit, accuracy scores fell significantly across multiple implementations of

both algorithms. Clearly a non-existent correlation among features is not indicative of worthless data.

As stated in section 1 of this report, 3.6% of the clients file an insurance claim. Consistently, from both algorithms implemented, an error rate of 3.6% was returned. Clearly, the error rate returned did not take into account the gross bias created in favor of classifying all clients as 0 or not filing an insurance claim. Because of the drastic imbalance of the classes, the trained models quickly dismissed occurrences of filed claims. Known as The Accuracy Paradox, the returned scores were only reflecting the underlying class distribution.

This pitfall was unfortunately not discovered until late in the maturation of the project and was not able to be remedied in time for the deadline. This is the shortcoming of using error rate to measure performance when the training data is severally imbalanced. A better performance metric would have been a confusion matrix or an F1-score. Both of these measures take into account false positives, false negatives, true positive, and true negatives in determining the performance of the model.

In conclusion, the models created from the training set did not accurately place new client instances in an appropriate category. The models instead overwhelmingly categorized clients in a single category, thus reflecting the data distribution through the error rate. The phenomena experienced in this report is firmly planted in the minds of every group member and will be quickly tested and accounted for in all future endeavors of all involved. Our best score from Kaggle: 0.21 from the Gaussian Naive Bayes model with the aforementioned column removed.

Acknowledgments

API design for machine learning software: experiences from the scikit-learn project, Buitinck et al., 2013.