Data Analysis and Machine Learning on Porto Seguro's Safe Drive Prediction

Jiamin Wang (NUID:001289959)
INFO 7245, Spring 2018, Northeastern University

1. Abstract

This study attempts to help Porto Seguro to predict the probability that a driver will initial an auto insurance claim in the next year. My project in the notebook dives into the data analysis and modeling for the Porto Seguro Kaggle competition in the Python environment. I used Pandas and associated libraries for most of the data manipulation. Furthermore, I used Scikit-learn and Tensorflow for the modeling. In data analysis part, I visualized different features to see how they relate to the target variable, explored multi-parameter interactions, and performed some feature engineering. For data modeling part, I have done a comparative analysis by implementing different machine learning and deep learning models. These models including Logistic Regression, Random Forests, Decision Tree, Support Vector Machine and Recurrent Neural Network. The result is that the prediction performance of machine learning models in this research is generally not very satisfactory, but the deep learning model with the appropriate parameters can greatly improve the accuracy of prediction.

2. Introduction

Porto Seguro is one of Brazil's largest auto and homeowner insurance companies. Improving the accuracy of insurance claims benefits both customers and insurance companies. Inaccuracies in car insurance company's claim predictions raise insurance costs for good drivers and lower the price for risky ones. Better predictions increase car-ownership accessibility for safer drivers and allow car insurance companies to charge fair prices to all customers. What's more, better predictions can also lead to improved profits for insurance companies.

In this research, I am challenged to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year, with the purpose of providing a fairer insurance cost based on an individual's driving habits.

3. Dataset and Features

The training dataset contains 595212 labeled records, one per client. Each record consists of 57 features with unknown meaning, one client ID, and one target variable indicated whether the customer filed an insurance claim. In the train and test data, features that belong to similar groupings are tagged as such in the feature names (e.g., “ind”, “reg”, “car”, “calc”). In addition, feature names include the postfix “bin” to indicate binary features and “cat” to indicate categorical features. Features without these designations are either continuous or ordinal. And values of “-1” indicate that the feature was missing from the observation. After analyzing, it can be distinguished that 17 of the 57 features are binary, 14 are categorical and the others are either continuous or ordinal. The target is binary and the dataset is strongly imbalanced: 3.65% is 1, the rest is 0.

4. Preprocessing

4.1 Columns Correlations

After classifying 57 features into four types of variable: binary variable, categorical variable, ordinal variable and numeric variable, I analysis the columns correlations for each of these variables. The following heat maps: figure 4.1, figure 4.2, figure 4.3 and figure 4.4 present the correlation matrices of the variables according to their types.

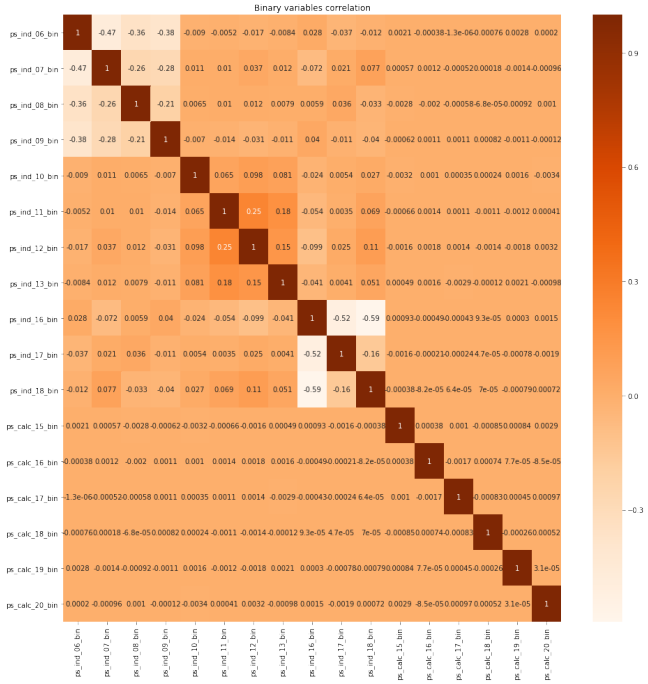

Figure 4.1 Binary variables correlation

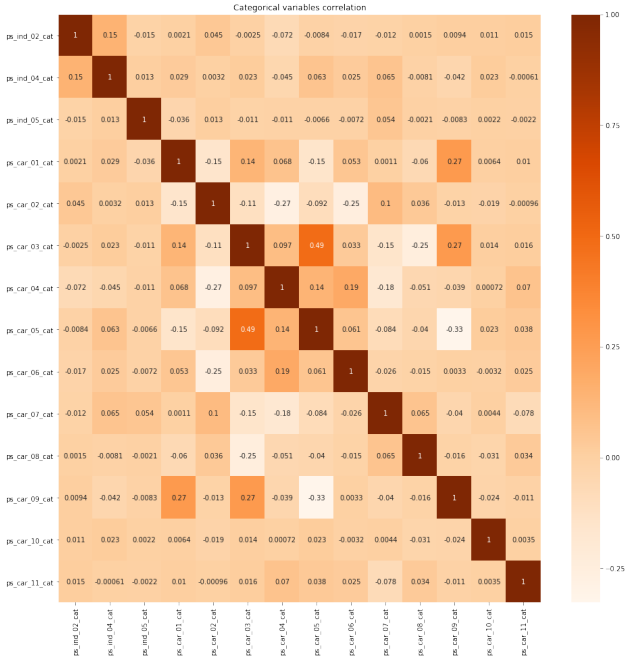

Figure 4.2 Categorical variables correlation

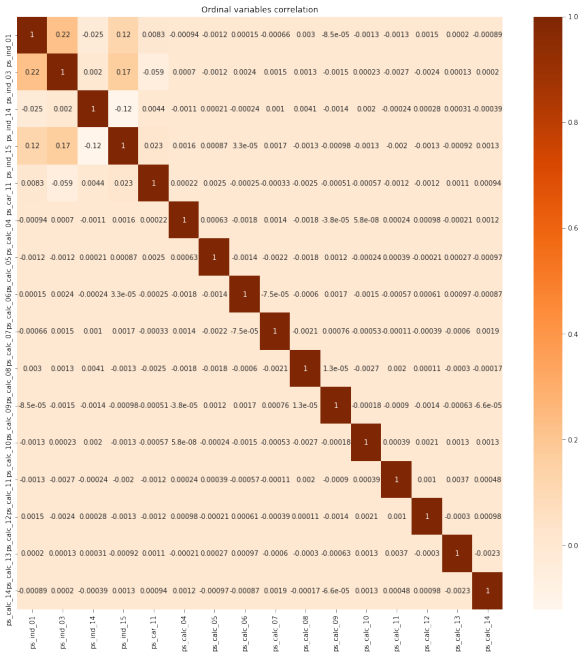

Figure 4.3 Ordinal variables correlation

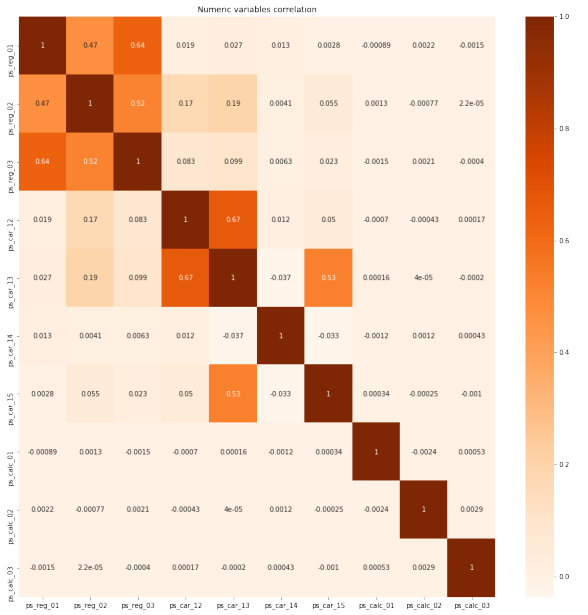

Figure 4.4 Numeric variable correlation

From the above figures, if the coefficient is higher than 0.5, I will choose them as highly correlated columns. The correlated columns are "ps_reg_01" and "ps_reg_03", "ps_reg_02" and "ps_reg_03", "ps_car_12" and "ps_car_13", "ps_car_13" and "ps_car_15". From the first plot we can notice that binary features are not correlated. However, they may have other characteristics which make us think they are not relevant. For the further exploration, I checked the 0/1 ratio of each binary feature. Figure 4.5 shows the 0/1 ratio of all binary features. It counts the total number of 0s and 1s in the target column for each binary variable. From this visualization, we can see that the distribution of 0 and 1 is extremely uneven for "ps_ind_10_bin", "ps_ind_11_bin", " ps_ind_12_bin" and "ps_ind_13_bin". Thus, it can be assumed that they are not relevant features since these features don't have enough observations with 1s to tell us anything. And these features will be deleted later.

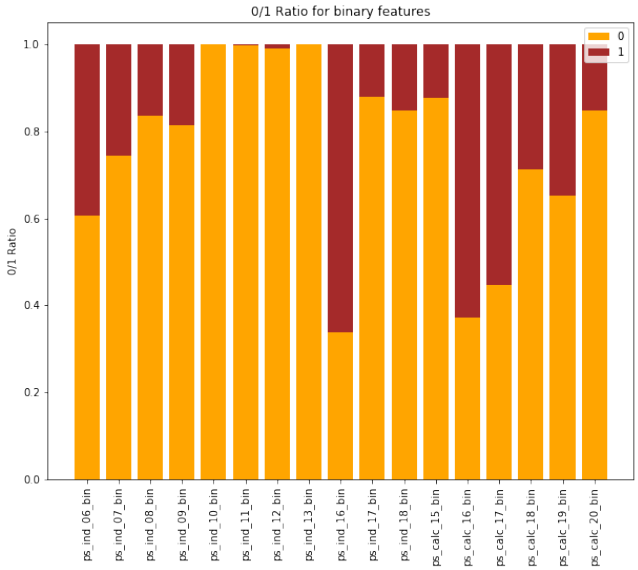

Figure 4.5 0/1 Ratio for binary features

4.2 Processing of Missing Values

Some values in the input datasets were missing (marked with -1). Firstly, I used missingno, a missing data visualization module for Python to find them intuitively. The missing values are presented in Figure 4.6. As we can see, there are two variables that have more than 40% missing values. These two variables are " ps_car_03_cat " and " ps_car_05_ca t ". We will drop them later because they do not provide enough information. After that, I preprocessed the dataset to replace these missing values with the mean of the feature over all examples in the dataset (for continuous features) or with the mode of the feature value over the dataset (for categorical and binary features). This imputation afforded minor improvements to our evaluation metric relative to models trained on data without imputation.

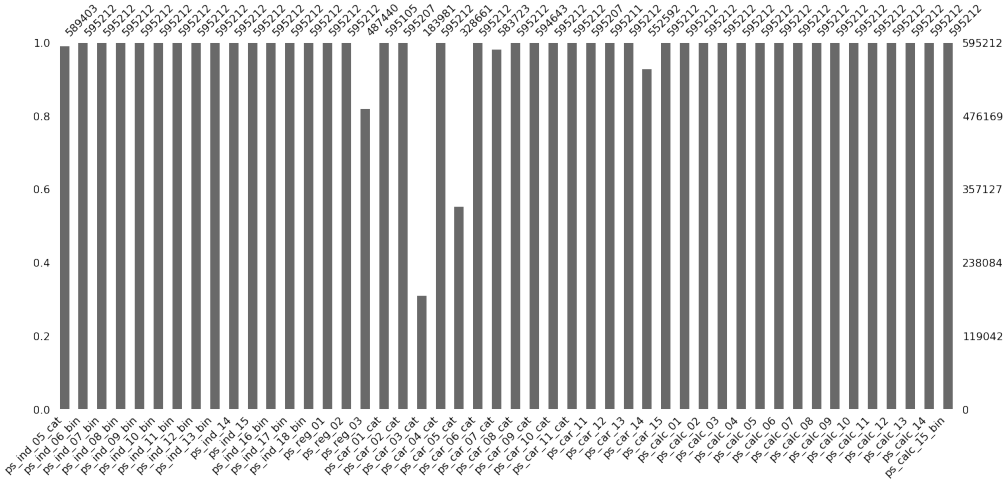

Figure 4.6 Missing value plot

4.3 Dimension reduction

Based on the above exploration, we can reduce the dimension initially by reducing the high missing values columns and irrelevant columns. After transforming each categorical column into multiple binary columns, I attempted to further conduct dimensionality reduction on our dataset by using PCA. However, during training, I decided against doing so after retrieving the variance ratios outputted by PCA. The result showed that 90% of the variance in the training set can be explained by a single principal component. I found this result suspicious, and when I attempted to train models on this

single principal component, they performed measurably worse under our evaluation metric than models trained on the full dataset (results not shown). This suggests the dataset is incompatible with PCA, which may be due to the large number of categorical and binary features it contains. Thus, I'd better go with all 212 columns I got.

4.4 One Hot Encoding

A one-hot encoding consists of representing states using for each a value whose binary representation has only a single digit 1. A one-hot encoding function can be defined as the function that takes a z vector as input and redefines the largest value of z to 1 and all other values of z to 0. For example, in a one-hot encoding, for three possible states, the binary values 001, 010, and 100 can be used. When using machine learning algorithms and deep learning algorithms, it is important to use a one hot encoding in order to reduce the noise while getting the splits for most useful categories.

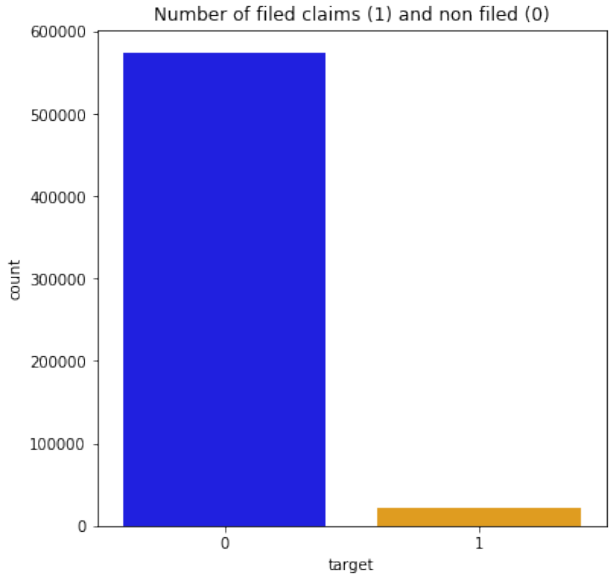

Figure 4.7 Imbalanced dataset

4.5 Resampling Strategies for Imbalanced database

We only have about 3.6% of filed claims for that policy holder (shows in Figure 4.7). This is a highly imbalanced dataset. This is not surprising since people are not willing to change file insurance every year. In general, they do so when they acquire a new vehicle. Since the percentage of file claims is very low, we can understand why the company is having a hard time predicting it.

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) or adding more examples from the minority class (over-sampling). Despite the advantage of balancing classes, these techniques also have their weaknesses. The simplest implementation of over-sampling is to duplicate random records from the minority class, which can cause overfitting. In under-sampling, the simplest technique involves removing

random records from the majority class, which can cause loss of information. After testing, I chose under-sampling strategy for my research.

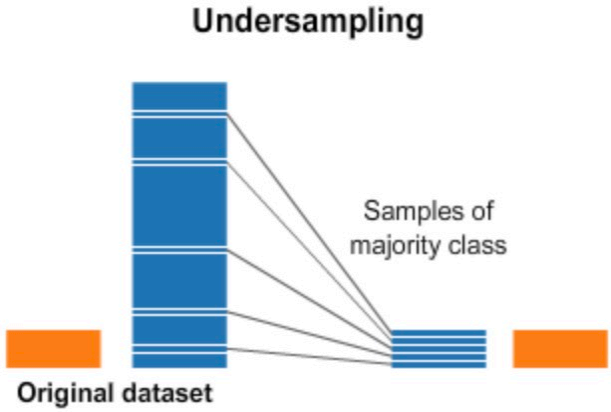

Figure 4.8.1 Under-sampling

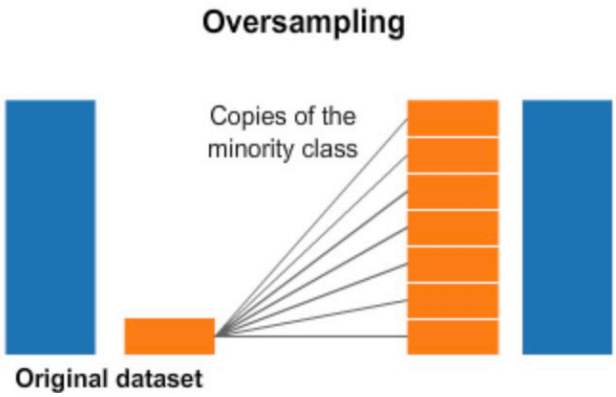

Figure 4.8.2 Over-sampling

5. Code with Documentation

The following links are my Python code in Jupyter Notebook.
Data preprocessing and EDA part:
https://github.com/JiaminWangM/Research-Notebook/blob/master/Portfolio(1)%20Data%20Preprocessing%20%26%20EDA%20Part.ipynb

Machine learning and deep learning modeling part:
https://github.com/JiaminWangM/Research-Notebook/blob/master/Portfolio(2)%20Machine%20Learning%20%26%20Deep%20Learning%20Part.ipynb

6. Results

The prediction accuracy of all the machine learning model is pretty low even after tuning the parameters. But after applying RNN, the accuracy increased greatly. From figure 6.6 we can see that the accuracy is improved step by step and the cost becomes lower and lower. The experimental results show that individually, deep learning model outperformed a simple machine learning model. And data preprocessing also played an important role.

6.1 Exploratory Data Analysis

To gauge the complexity of the dataset and search for any obvious dependencies between feature values and claim-making, I visualized the distributions of each of the variables and the relationship between each individual feature against the target. Although at first these plots might not tell us much, it is important

to have as a reference later on: when an interesting insight related to a feature emerges, it is useful to refer back to the distribution of that feature. These plots and detailed documentation are in the above links.

6.2 Machine Learning and Deep Learning Modeling

The main idea here is to implement some machine learning and deep learning algorithms such as Logistic Regression, Random Forests, Decision Tree, Support Vector Machine and Recurrent Neural Network and compare their performance and prediction results. For machine learning model part, after re-balancing the dataset by using under-sampling strategy, I split the data into training data and testing data and fit different models. Then I printed the confusion matrix and classification report for each model. In order to get a better result, I also used Gridsearch to tune the parameters. For deep learning model part, after comparing the prediction results I set my RNN model with the following elements: the activation function is Softsign, the cost function is reduce-sum, and I chose to optimize my model via AdamOptimizer. The prediction results are as following.

```
print(confusion_matrix(y_test,prediction))
```

```
[[4346 2774]
 [3207 3992]]
```

```
#Create a classification report for the model.
print(classification_report(y_test,prediction))
```

```
             precision    recall  f1-score   support

          0       0.58      0.61      0.59      7120
          1       0.59      0.55      0.57      7199

avg / total       0.58      0.58      0.58     14319
```

Figure 6.1 Support Vector Machine result

```
dt_prediction = dtree.predict(X_test_sc)
```

```
print(confusion_matrix(y_test,dt_prediction))
```

```
[[3843 3338]
 [3377 3761]]
```

```
print(classification_report(y_test,dt_prediction))
```

```
             precision    recall  f1-score   support

          0       0.53      0.54      0.53      7181
          1       0.53      0.53      0.53      7138

avg / total       0.53      0.53      0.53     14319
```

Figure 6.2 Decision Tree result

```
print(confusion_matrix(y_test,rfc_prediction))
```

```
[[4266 2915]
 [3008 4130]]
```

```
print(classification_report(y_test,rfc_prediction))
```

```
             precision    recall  f1-score   support

          0       0.59      0.59      0.59      7181
          1       0.59      0.58      0.58      7138

avg / total       0.59      0.59      0.59     14319
```

Figure 6.3 Random Forests result

```
print(confusion_matrix(y_test,lr_prediction))
```

```
[[4423 2758]
 [3157 3981]]
```

```
print(classification_report(y_test,lr_prediction))
```

```
             precision    recall  f1-score   support

          0       0.58      0.62      0.60      7181
          1       0.59      0.56      0.57      7138

avg / total       0.59      0.59      0.59     14319
```

Figure 6.4 Logistic Regression result

```
('Epoch:', 0, 'Acc =', '0.66050', 'Cost =', '38338.19141', 'Valid_Acc =', '0.52214', 'Valid_C
ost = ', '16358.28418')
('Epoch:', 1, 'Acc =', '0.75063', 'Cost =', '18672.82617', 'Valid_Acc =', '0.52424', 'Valid_C
ost = ', '11651.89844')
('Epoch:', 2, 'Acc =', '0.81878', 'Cost =', '12119.57227', 'Valid_Acc =', '0.51753', 'Valid_C
ost = ', '9715.30078')
('Epoch:', 3, 'Acc =', '0.86188', 'Cost =', '9549.67285', 'Valid_Acc =', '0.52312', 'Valid_Co
st = ', '9480.12305')
('Epoch:', 4, 'Acc =', '0.87932', 'Cost =', '8423.10059', 'Valid_Acc =', '0.52382', 'Valid_Co
st = ', '9859.64258')
()
Optimization Finished!
()
```

Figure 6.5 RNN result

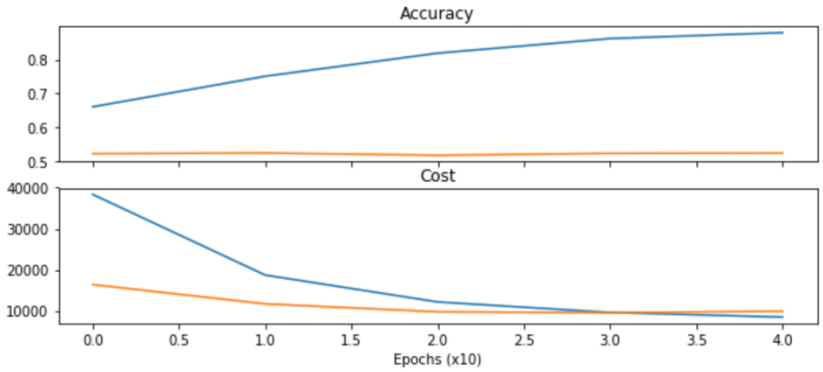

Figure 6.6 RNN result plot

7. Discussion

I have made a deep assessment of the prediction ability and performance of different models. And the prediction results may help Porto Seguro to further tailor their price, and hopefully make auto insurance coverage more accessible to more drivers. After analyzing the prediction result of each model with different parameters, it can be judged that applying many architectures could further enhance accuracy. For the future work, there are a few things I could do to improve these models. First, I could explore more hyper-parameters for the neural net models. Second, I could ensemble the existing models by taking a weighted combination of their predictions.

8. References

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

[2] https://github.com/mehrvch/Porto_Seguro/blob/master/ipynb/1_EDA.ipynb

[3] https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

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

[5]
https://www.kaggle.com/currie32/predicting-fraud-with-tensorflow