**Porto Seguro’s Safe Driver Prediction**

Contents

[**1.** **Introduction** 1](#_Toc43323782)

[**2.** **Dataset** 1](#_Toc43323783)

[**3.** **Data Exploratory Analysis** 1](#_Toc43323784)

[**4.** **Feature Engineering** 5](#_Toc43323785)

[**5.** **Feature Selection/ Feature Extraction** 5](#_Toc43323786)

[**6.** **Feature Scaling** 6](#_Toc43323787)

[**7.** **Methods** 7](#_Toc43323788)

[7.1 **Models Applied** 7](#_Toc43323789)

[**8.** **Experiments and Results** 7](#_Toc43323790)

[**9.** **Future Work** 11](#_Toc43323791)

# **Introduction**

[Porto Seguro](https://www.portoseguro.com.br/), one of Brazil’s largest auto and homeowner insurance companies put this problem statement on Kaggle, with an aim to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. The end goal was to reduce inaccuracies in car insurance company’s claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones.

# **Dataset**

Two datasets are available.

1. Train.csv : It contains the training data, where each row corresponds to a policy holder, and the target columns signifies that a claim was filed.
2. Test.csv : It contains the test data.

|  |  |  |
| --- | --- | --- |
| **Dataset** | **No of records** | **No of features** |
| Train.csv | 595212 | 59 |
| Test.csv | 892816 | 58 |

* 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.
* Values of -1 indicate that the feature was missing from the observation.
* The target column signifies whether a claim was filed for that policy holder.

# **Data Exploratory Analysis**

1. **Check Duplicate Records:**

Duplicate features are the features that have similar values. Duplicate features do not add any value to algorithm training, rather they add overhead and unnecessary delay to the training time. Therefore, it is always recommended to remove the duplicate features from the dataset before training.

This will be done taking the transpose of training data followed by using duplicated() method to identify duplicate rows and finally apply drop\_duplicates() to drop the duplicate features retaining first copy.

The training dataset does not have any duplicates.

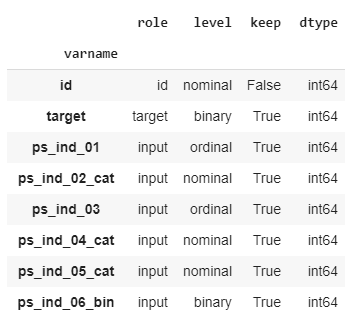
1. **Creation of Metadata:**

To facilitate the data management, meta-information is stored about the variables in a DataFrame. This will be helpful when in selecting specific variables for analysis, visualization, modeling.

Concretely will store:

* **role**: input, ID, target
* **level**: nominal, interval, ordinal, binary
* **keep**: True or False
* **dtype**: int, float, str

Here is a snapshot of a part of the Meta information.



1. **Handling imbalanced data:**

The proportion of records with target=1 is far less than target=0. Almost 96% records have target =0 which means no claims. This imbalance can lead to model inaccuracy.

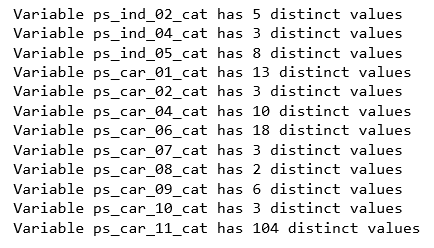
Two possible strategies to deal with this problem are:

* oversampling records with target=1
* undersampling records with target=0

Testing has been done for both ‘undersampling’ and ‘oversampling’. Since the datasets have good number of records, ‘undersampling’ is the preferred approach here.

1. **Data Quality Checks:**
2. **Check Missing values:**

Missing records have been checked separately for training and test set. Interval variables, Ordinal and Binary variables are inspected separately.



* Variables with large % of missing values will be removed.
* Missing values in continuous variables will be replaced by mean.
* Ordinal missing value will be replaced by Mode.
* Other categorical missing values have been left as is.

1. **Checking Cardinality of Categorical variables**

Cardinality refers to the number of different values in a variable. Need to check if there is any high cardinality feature, before creating ‘*dummy*’ variables.

Only one feature **ps\_car\_11\_cat**  has 104 distinct values. ‘*Adding noise and smoothing technique*’ has been applied on this feature.

Rest all categorical variables have less than 20 distinct values.

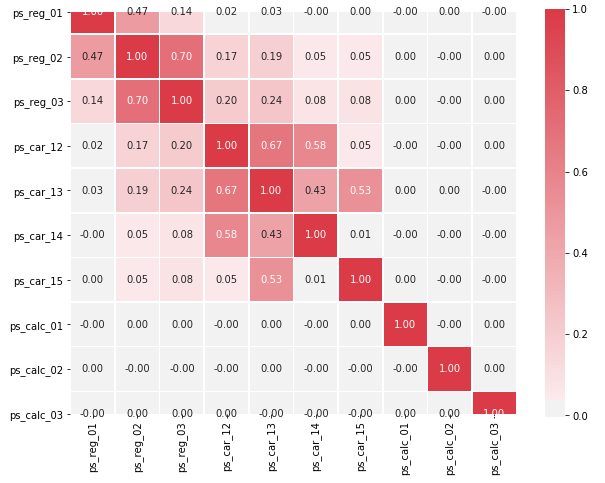
1. **Exploratory data visualization:**

Different features mainly categorical variables and the proportion of customers with target=1 distribution have been studied.

1. **Checking correlation between variables:**

Two or more than two features are correlated if they are close to each other in the linear space. *Heatmap* has been used to visualize*.*

**Interval variables**: Following is the heatmap.



There is a strong correlation between the variables:

* ps\_reg\_02 and ps\_reg\_03 (0.7)
* ps\_car\_12 and ps\_car13 (0.67)
* ps\_car\_12 and ps\_car14 (0.58)
* ps\_car\_13 and ps\_car15 (0.67)

Further, pairplot has been used to visualize the relationship between the highly correlated variables separately. Since there are only 4 pairs, did not drop any from the pairs for now. Advanced ML models should be able to handle.

**Ordinal variables**: No strong correlations have been observed.

# **Feature Engineering**

1. **Create dummy variables:**

The values of the categorical variables do not represent any order or magnitude. For instance, category 2 is not twice the value of category 1. Therefore, dummy variables have been created to deal with that. The first dummy variable has been dropped as this information can be derived from the other dummy variables generated for the categories of the original variable. This step was carried out on combined training and test set.

1. **Create interaction variables:**

Interaction effects indicate that a third variable influences the relationship between an independent and dependent variable. To understand the relationship better and to expand the scope of more hypothesis tests, the interaction variables have been added.

# **Feature Selection/ Feature Extraction**

1. **Checking features with low or zero variance:**

Constant features are the type of features that contain only one value for all the outputs in the dataset. Constant features provide no information that can help in classification of the record at hand. Eventually, it might be better to remove those features.

To do so will use Variance Threshold function. The function requires a value for its threshold parameter. Passing a value of zero for the parameter will filter all the features with zero variance.

There are no zero-variance variables. But if we would remove features with less than 1% variance, then we would remove 28 variables. So, no features have been removed based on variance assuming the classifier will choose the features accordingly.

1. **Selecting features with SelectFromModel:**

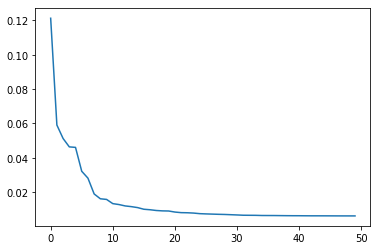
The reference for this has been taken from [GitHub repo of Sebastian Raschka](https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch04/ch04.ipynb). This lets another classifier select the best features and continue with those. It is possible to specify which prefit classifier to use and what the threshold is for the feature importances.

This has not been applied yet. Feature extraction technique PCA has been preferred over this.

1. **Principal Component Analysis (PCA):**

PCA is an unsupervised type of feature extraction, where original variables are combined and reduced to their most important and descriptive components.

Different values of **k (no. of principal components)** have been explored on training dataset. The plot for ‘explained\_variance\_ratio\_’ shows not much variance after ‘k’ exceeds 20.



All the models have been applied after extracting features using PCA setting k=20.

# **Feature Scaling**

Applied standard scaling to the training data as different features have different ranges.

# **Methods**

Will discuss about the methods and models applied on the datasets in this section.

Main target variable to predict is whether the customer will file for insurance claim next year or not.

## **Models Applied**

The datasets are big with around **600k** samples in training set and **900k** in test set without any sampling technique. The number of features were around **164** after applying feature engineering techniques. Since I had to train the models with limited hardware capacity, I managed to explore only few models. The models were trained after extracting features with PCA. Tested on imbalanced datasets and on balanced datasets (both under sampled and oversampled).

1. SVM: A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. Here tested with non-linear kernel.
2. Random Forest: Random forest, like its name implies, consists of many individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction.
3. Gradient Boosting Tree:This techniqueproduces a prediction model in the form of an [ensemble](https://en.wikipedia.org/wiki/Ensemble_learning) of weak prediction models, typically [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning). It builds the model in a stage-wise fashion like other [boosting](https://en.wikipedia.org/wiki/Boosting_(machine_learning)) methods do, and it generalizes them by allowing optimization of an arbitrary [differentiable](https://en.wikipedia.org/wiki/Differentiable_function) [loss function](https://en.wikipedia.org/wiki/Loss_function).
4. XGBoost : XGBoost stands for Extreme Gradient Boosting; it is a specific implementation of the Gradient Boosting method which uses more accurate approximations to find the best tree model.
5. Neural net : A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

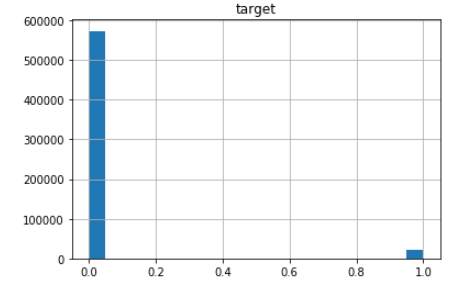
# **Experiments and Results**

The primary challenge here was high number of records and limited hardware capacity to test. Will demonstrate the results for three scenarios – imbalanced dataset, under sampled dataset, oversampled dataset.

After applying PCA and Feature engineering techniques as described in the previous section, trained the models, and tested the results with 5-fold Stratified Cross Validation Technique. Initially tried with 10-fold CV, but for the sake of time switched to 5-fold CV. Multithreading was enabled.

The classification metrics are taken as – Accuracy, Precision, Recall, AUC.

1. **Imbalanced dataset**: The training dataset has high class imbalance. The % of no claims (target = 0) is much higher than the % the claims filed.



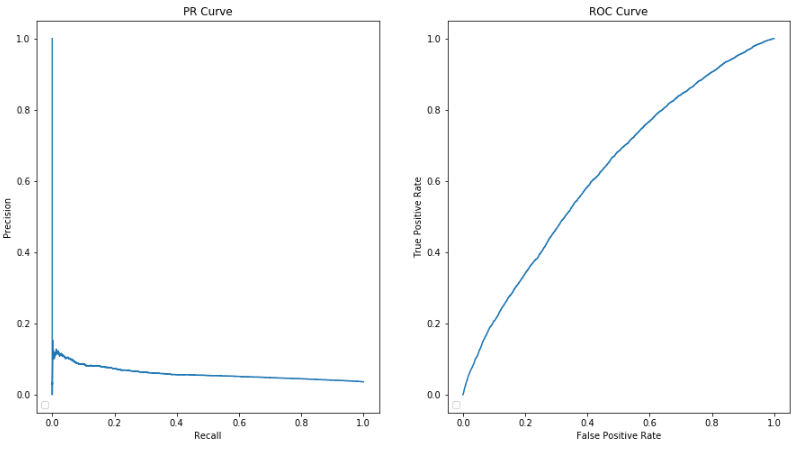
Giving below the results from some cross-validation tests -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| Random Forest | 0.96 | 0.55 | 0.50 | 0.53 |
| Gradient Boosting Tree | 0.96 | 0.58 | 0.50 | 0.51 |
| XGBoost | 0.96 | 0.48 | 0.5 | 0.63 |
| Neural Net | 0.96 | 0.52 | 0.50 | 0.51 |

SVM with non-linear kernel on training data took more than expected time and the results were not good either, so did not include in the table.

It seems, all the models have close ‘Accuracy’ values, but for imbalanced datasets, it cannot be regarded as single metric to assess performance. If we look at all the metrics , specifically AUC – the XGBoost seems to be the apt choice.

Giving below the Precision-Recall curve and ROC Curve for predictions obtained from XGBoost Model .

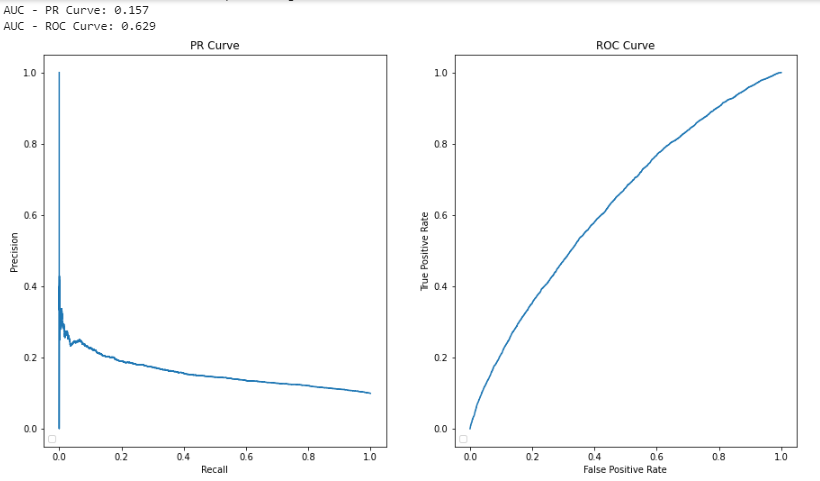


1. **Under-sampled balanced dataset:**

Giving below the results from some cross-validation tests -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| Random Forest | 0.89 | 0.475 | 0.50 | 0.59 |
| XGBoost | 0.90 | 0.45 | 0.69 | 0.76 |
| Neural Net | 0.90 | 0.48 | 0.50 | 0.61 |

Giving below the Precision-Recall curve and ROC Curve for predictions obtained from XGBoost Model and corresponding AUC score.



Tried hyperparameter tuning using GridSearchCV and RandomSearchCV on XGBoost model.

Following combinations seem to have worked better.

*colsample\_bylevel=1,*

*colsample\_bynode=1,*

*colsample\_bytree=1, gamma=0,*

*learning\_rate=0.02, max\_delta\_step=0,*

*max\_depth=3, min\_child\_weight=1,*

*missing=None, n\_estimators=100*

1. **Over-sampled balanced dataset:**

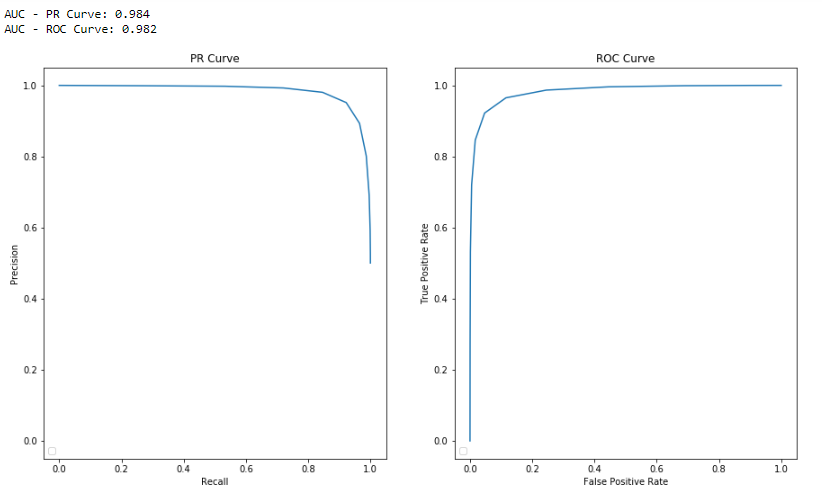
Oversampled datasets have more than 1 Million records (1147036) with equal distribution of target =1 and target =0 which was further splited into training and validation sets via Cross validation tests.

Giving below the results from some cross-validation tests -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| Random Forest | 0.976 | 0.977 | 0.97 | 0.99 |
| XGBoost | 0.89 | 0.90 | 0.89 | 0.94 |

*scipy.stats.mannwhitneyu (xgboost\_cv\_score, rf\_cv\_score) -> statistic=0.0, pvalue=0.006092890177672406*

Giving below the Precision-Recall curve and ROC Curve for predictions obtained from Random Forest Model and corresponding AUC score.



It seems Oversampling technique with Random Forest model performs better, though further testing is needed to assess.

# **Future Work**

1. Run the models excluding ‘calc’ variables, as they look like derived variables.
2. Test with more sampling techniques.
3. Explore more models like Orthogonal RF, CatBoost, Regularized greedy forest along with other Ensemble techniques.
4. Carry out more hypothesis tests.
5. Tuning hyperparameters.